Mathematical Theory and Computational Practice

5th Conference on Computability in Europe, CiE 2009
Heidelberg, Germany, July 19-24, 2009

Abstract Booklet

Klaus Ambos-Spies
Benedikt Löwe
Wolfgang Merkle
(Editors)

University of Heidelberg
Department of Mathematics and Computer Science
Mathematical Logic and Theoretical Computer Science Group
After several years of research activity, the informal cooperation “Computability in Europe” decided to take a more formal status at their meeting in Athens in June 2008: the Association for Computability in Europe was founded to promote the development, particularly in Europe, of computability-related science, ranging over mathematics, computer science, and applications in various natural and engineering sciences such as physics and biology, including the promotion of the study of philosophy and history of computing as it relates to questions of computability. As mentioned, this association builds on the informal network of European scientists working on computability theory that had been supporting the conference series CiE-CS over the years, and now became its new home.

The aims of the conference series remain unchanged: to advance our theoretical understanding of what can and cannot be computed, by any means of computation. Its scientific vision is broad: computations may be performed with discrete or continuous data by all kinds of algorithms, programs, and machines. Computations may be made by experimenting with any sort of physical system obeying the laws of a physical theory such as Newtonian mechanics, quantum theory or relativity. Computations may be very general, depending upon the foundations of set theory; or very specific, using the combinatorics of finite structures. CiE also works on subjects intimately related to computation, especially theories of data and information, and methods for formal reasoning about computations. The sources of new ideas and methods include practical developments in areas such as neural networks, quantum computation, natural computation, molecular computation, computational learning. Applications are everywhere, especially, in algebra, analysis and geometry, or data types and programming. Within CiE there is general recognition of the underlying relevance of computability to physics and a broad range of other sciences, providing as it does a basic analysis of the causal structure of dynamical systems.

This volume is the informal abstract booklet of Mathematical Theory and Computational Practice, the fifth in a series of conferences of CiE, which was held at the Ruprecht-Karls-Universität Heidelberg, Germany, from 19 to 24 July 2009. The first four meetings of CiE were at the University of Amsterdam in 2005, at
the University of Wales Swansea in 2006, at the University of Siena in 2007, and at the University of Athens in 2008.

CiE and its conferences have changed our perceptions of computability and its interface with other areas of knowledge. The large number of mathematicians and computer scientists attending those conferences had their view of computability theory enlarged and transformed: they discovered that its foundations were deeper and more mysterious, its technical development more vigorous, its applications wider and more challenging than they had known. The annual CiE conference has become a major event, and is the largest international meeting focused on computability theoretic issues. Future meetings in Ponta Delgada, Açores (2010, Portugal), Sofia (2011, Bulgaria), and Cambridge (2012, England) are in planning. The series is coordinated by the CiE Conference Series Steering Committee consisting of Arnold Beckmann (Swansea), Paola Bonizzoni (Milano), S. Barry Cooper (Leeds), Benedikt Löwe (Amsterdam, Chair), Elvira Mayordomo (Zaragoza), Dag Normann (Oslo), and Peter van Emde Boas (Amsterdam).

Structure and Programme of the Conference

The conference was based on invited tutorials and lectures, and a set of special sessions on a range of subjects; there were also many contributed papers and informal presentations. The conference proceedings volume has been published in the Springer LNCS series, Volume 5635, and contains 17 of the invited lectures and 34 of the submitted contributed papers. The present volume represents the informal abstract booklet which contains abstracts of the opening lecture, of seven invited plenary and of eleven invited special session talks, another 31 of the contributed talks, as well as 42 abstracts of informal presentations. The fact that a paper is printed in this volume does not constitute publication of the paper or that the programme committee endorsed the publication of the paper. There will be a number of post-proceedings publications, including special issues of *Annals of Pure and Applied Logic*, *Journal of Logic and Computation*, and *Theory of Computing Systems*.

Tutorial Speakers

Pavel Pudlák (Prague), Luca Trevisan (Berkeley).

Invited Speakers

Manindra Agrawal (Kanpur), Jeremy Avigad (Pittsburgh), Mike EDMunds (Cardiff, Opening Lecture), Peter Koepke (Bonn), Phokion Kolaitis (San Jose), Andrea Sorbi (Siena), Rafael D. Sorkin (Syracuse), Vijay V. Vazirani (Atlanta).
Special Sessions

Algorithmic Randomness.
Organizers. Elvira Mayordomo (Zaragoza) and Wolfgang Merkle (Heidelberg).

Computational Model Theory.
Organizers. Julia F. Knight (Notre Dame) and Andrei Morozov (Novosibirsk).
Speakers. Ekaterina B. Fokina, Sergey Goncharov, Russell Miller, Antonio Montalbán.

Computation in Biological Systems — Theory and Practice.
Organizers. Alessandra Carbone (Paris) and Erzsébet Csuhaj-Varjú (Budapest).
Speakers. Ion Petre, Alberto Policriti, Francisco J. Romero-Campero, David R. Westhead.

Optimization and Approximation.
Organizers. Magnús M. Halldórsson (Reykjavik) and Gerhard Reinelt (Heidelberg).

Philosophical and Mathematical Aspects of Hypercomputation.
Organizers. James Ladyman (Bristol) and Philip Welch (Bristol).
Speakers. Tim Button, Samuel Coskey, Mark Hogarth, Oron Shagrir.

Relative Computability.
Organizers. Rod Downey (Wellington) and Alexandra A. Soskova (Sofia)
Speakers. George Barmpalias, Hristo Ganchev, Keng Meng Ng, Richard Shore.

Organisation and Acknowledgements

The conference CiE 2009 was organised by Klaus Ambos-Spies (Heidelberg), Timur Bakibayev (Heidelberg), Arnold Beckmann (Swansea), Laurent Bienvenu (Heidelberg), Barry Cooper (Leeds), Felicitas Hirsch (Heidelberg), Rupert Hödl (Heidelberg), Thorsten Kräling (Heidelberg), Benedikt Löwe (Amsterdam), Gunther Mainhardt (Heidelberg), and Wolfgang Merkle (Heidelberg).

The Programme Committee was chaired by Klaus Ambos-Spies and Wolfgang Merkle:
We are delighted to acknowledge and thank the following for their essential financial support: Deutsche Forschungsgemeinschaft (German Research Foundation), Ruprecht-Karls-Universität Heidelberg (Bioquant and Department of Mathematics and Computer Science), The Elsevier Foundation.

We are proud to offer the programme “Women in Computability” funded by the Elsevier Foundation as part of CiE 2009. The Steering Committee of the conference series CiE-CS is concerned with the representation of female researchers in the field of computability. The series CiE-CS has actively tried to increase female participation at all levels in the past years. Starting in 2008, our efforts are being funded by a grant of the Elsevier Foundation under the title “Increasing representation of female researchers in the computability community”. As part of this programme, we had another workshop, a grant scheme for female researchers, a mentorship programme, and free childcare.

We thank Andrej Voronkov for his EasyChair system which facilitated the work of the Programme Committee and the editors considerably.

Heidelberg and Amsterdam, June 2009

Klaus Ambos-Spies
Benedikt Löwe
Wolfgang Merkle
Table of Contents

Opening Lecture

The Antikythera Mechanism ........................................ 1
  Mike Edmunds

Plenary Talks

The Isomorphism Conjecture ........................................ 2
  Manindra Agrawal

Computability in Ergodic Theory .................................. 3
  Jeremy Avigad

Ordinal Computability ............................................. 4
  Peter Koepke

Foundations and Applications of Schema Mappings .............. 5
  Phokion G. Kolaitis

Computing Using Positive Information ............................ 6
  Andrea Sorbi

Logic : the Quantum :: Geometry : Gravity ....................... 7
  Rafael Sorkin

Combinatorial Algorithms for Convex Programs Capturing Market
Equilibria and Nash Bargaining Solutions ........................ 8
  Vijay V. Vazirani

Special Session Talks

Tracing and Domination in the Turing Degrees ................... 9
  George Barmpalias

IX
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Computably Enumerable Random Reals</td>
<td>11</td>
</tr>
<tr>
<td>Laurent Bienvenu</td>
<td></td>
</tr>
<tr>
<td>Integer Programming and Real-Time Scheduling</td>
<td>12</td>
</tr>
<tr>
<td>Friedrich Eisenbrand</td>
<td></td>
</tr>
<tr>
<td>Computable Numberings of Hyperarithmetical Sets and Complexity of</td>
<td>13</td>
</tr>
<tr>
<td>Countable Models</td>
<td></td>
</tr>
<tr>
<td>Sergey Goncharov</td>
<td></td>
</tr>
<tr>
<td>Limits on Jump Inversion for Strong Reducibilities</td>
<td>15</td>
</tr>
<tr>
<td>Keng Meng Ng</td>
<td></td>
</tr>
<tr>
<td>Computational Strategies for the Heat Shock Response</td>
<td>16</td>
</tr>
<tr>
<td>Ion Petre</td>
<td></td>
</tr>
<tr>
<td>Stochastic Programs and Hybrid Automata for (Biological) Modeling</td>
<td>17</td>
</tr>
<tr>
<td>Alberto Policriti</td>
<td></td>
</tr>
<tr>
<td>On the Relations Between Effective, Hyper-, and Physical Computation</td>
<td>18</td>
</tr>
<tr>
<td>Oron Shagrir</td>
<td></td>
</tr>
<tr>
<td>Domination, Forcing, Array Non recursiveness and Relative Recursive</td>
<td>19</td>
</tr>
<tr>
<td>Enumerability</td>
<td></td>
</tr>
<tr>
<td>Richard A. Shore</td>
<td></td>
</tr>
<tr>
<td>From Optimal Algorithms to Optimal Mechanisms</td>
<td>20</td>
</tr>
<tr>
<td>Marc Uetz</td>
<td></td>
</tr>
<tr>
<td>Bayesian Data Integration for Predicting Gene Function in Malaria</td>
<td>22</td>
</tr>
<tr>
<td>David R. Westhead</td>
<td></td>
</tr>
<tr>
<td>Contributed Talks</td>
<td></td>
</tr>
<tr>
<td>A Note on Universal Numberings</td>
<td>23</td>
</tr>
<tr>
<td>Kuanysh Abeshev and Serikzhan Badaev</td>
<td></td>
</tr>
<tr>
<td>Automatic Classification of Protein Sequences into Families Using</td>
<td>28</td>
</tr>
<tr>
<td>Elongated K-Mean Spectral Clustering</td>
<td></td>
</tr>
<tr>
<td>Ulavappa B. Angadi and M. Venkatesulu</td>
<td></td>
</tr>
<tr>
<td>Temporal Logic Treating Uncertainty and Interacting Agents with</td>
<td>38</td>
</tr>
<tr>
<td>Time Indexes from $\mathbb{Z}$</td>
<td></td>
</tr>
<tr>
<td>Sergey Babenyshev, Vladimir Rybakov, and Renate Schmidt</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Evaluating Algorithms According to their Energy Consumption</td>
<td>48</td>
</tr>
<tr>
<td><em>Hannah Bayer and Markus Nebel</em></td>
<td></td>
</tr>
<tr>
<td>A Characterisation of BD-N</td>
<td>58</td>
</tr>
<tr>
<td><em>Josef Berger</em></td>
<td></td>
</tr>
<tr>
<td>Learning Non-Confluent NLC Graph Grammar Rules</td>
<td>60</td>
</tr>
<tr>
<td><em>Hendrik Blockeel and Robert Brijder</em></td>
<td></td>
</tr>
<tr>
<td>Effective Capacity and Randomness of Closed Sets</td>
<td>70</td>
</tr>
<tr>
<td><em>Paul Brodhead and Douglas Cenzer</em></td>
<td></td>
</tr>
<tr>
<td>Lévy Flights for Ant Colony Optimization in Continuous Domains</td>
<td>79</td>
</tr>
<tr>
<td><em>Roberto Candela, Giulio Cottone, Giuseppe Fileccia Scimemi, and Eleonora Riva Sanseverino</em></td>
<td></td>
</tr>
<tr>
<td>Iterative Rounding for the Closest String Problem</td>
<td>89</td>
</tr>
<tr>
<td><em>Jingchao Chen</em></td>
<td></td>
</tr>
<tr>
<td>Computable Types for Dynamic Systems</td>
<td>99</td>
</tr>
<tr>
<td><em>Pieter Collins</em></td>
<td></td>
</tr>
<tr>
<td>Computable CTL for Discrete-Time and Continuous-Space Dynamic Systems</td>
<td>110</td>
</tr>
<tr>
<td><em>Pieter Collins and Ivan S. Zapreev</em></td>
<td></td>
</tr>
<tr>
<td>Toward a Semantical Model of Organization Growth for Multiagent Systems</td>
<td>120</td>
</tr>
<tr>
<td><em>Graçaliz Pereira Dimuro and Antônio Carlos da Rocha Costa</em></td>
<td></td>
</tr>
<tr>
<td>Taming Implications in Dummett Logic</td>
<td>130</td>
</tr>
<tr>
<td><em>Guido Fiorino</em></td>
<td></td>
</tr>
<tr>
<td>Constraint Message Propagation Dynamics of Nonlinear Boolean Equations</td>
<td>140</td>
</tr>
<tr>
<td><em>Binghui Guo, Wei Wei, and Zhiming Zheng</em></td>
<td></td>
</tr>
<tr>
<td>Function Operators Spanning the Arithmetical and the Polynomial Hierarchy</td>
<td>150</td>
</tr>
<tr>
<td><em>Armin Hemmerling</em></td>
<td></td>
</tr>
<tr>
<td>Unambiguous Functions in Logarithmic Space</td>
<td>162</td>
</tr>
<tr>
<td><em>Grzegorz Herman and Michael Soltys</em></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notions of Reducibility for Mass Problems</td>
<td>176</td>
</tr>
<tr>
<td><em>Kojiro Higuchi and Takayuki Kihara</em></td>
<td></td>
</tr>
<tr>
<td>Computation by Synthetic Cell Signaling and Oscillating Processes</td>
<td>186</td>
</tr>
<tr>
<td>Modelled Using Mass-Action Kinetics</td>
<td></td>
</tr>
<tr>
<td><em>T. Hinze, R. Faßler, G. Escuela, B. Ibrahim, and S. Schuster</em></td>
<td></td>
</tr>
<tr>
<td>Strong Normalization of Program-Indexed Lambda Calculus</td>
<td>196</td>
</tr>
<tr>
<td><em>Norihiro Kamide</em></td>
<td></td>
</tr>
<tr>
<td>A Simple and Practical Valuation Tree Calculus for First-Order Logic</td>
<td>206</td>
</tr>
<tr>
<td><em>Ján Kl'uka and Paul J. Voda</em></td>
<td></td>
</tr>
<tr>
<td>Definability of Closure Operations in the h-Quasiorder of Labeled Forests</td>
<td>216</td>
</tr>
<tr>
<td><em>Oleg V. Kudinov, Victor L. Selivanov, and Anton V. Zhukov</em></td>
<td></td>
</tr>
<tr>
<td>The Relationship Between Measurability and Computability</td>
<td>226</td>
</tr>
<tr>
<td><em>James Lambert</em></td>
<td></td>
</tr>
<tr>
<td>Union Theorems in Type-2 Computation</td>
<td>236</td>
</tr>
<tr>
<td><em>Chung-Chih Li</em></td>
<td></td>
</tr>
<tr>
<td>Two-thirds Bisimulation Indexes</td>
<td>246</td>
</tr>
<tr>
<td><em>Yanfang Ma and Yixiang Chen</em></td>
<td></td>
</tr>
<tr>
<td>An Implementation of Exact Real Number Computation Based on LRT</td>
<td>256</td>
</tr>
<tr>
<td><em>J. Raymundo Marcial-Romero, Claudia Zepeda, J. A. Hernández, Héctor A. Montes-Venegas, and David Linares</em></td>
<td></td>
</tr>
<tr>
<td>On a Hierarchy of 5' → 3' Sensing WK Finite Automata Languages</td>
<td>266</td>
</tr>
<tr>
<td><em>Benedek Nagy</em></td>
<td></td>
</tr>
<tr>
<td>Ackermannian Lower Bounds for Lengths of Bad Sequences of Monomial Ideals over Polynomial Rings in two Variables</td>
<td>276</td>
</tr>
<tr>
<td><em>Florian Pelupessy and Andreas Weiermann</em></td>
<td></td>
</tr>
<tr>
<td>Petri Nets Unfoldings and the Individual/Collective Token Philosophy</td>
<td>279</td>
</tr>
<tr>
<td><em>G. Michele Pinna</em></td>
<td></td>
</tr>
<tr>
<td>Prime Models of Finite Computable Dimension</td>
<td>289</td>
</tr>
<tr>
<td><em>Pavel Semukhin</em></td>
<td></td>
</tr>
<tr>
<td>Generalized Temperley-Lieb Algebras and Quantum Computation</td>
<td>296</td>
</tr>
<tr>
<td><em>Mario Vélez and Juan Ospina</em></td>
<td></td>
</tr>
</tbody>
</table>
Random Diffusion and Congestion on Weighted Complex Networks
Jiang Xin, Shaoting Tang, Lili Ma, Zhanli Zhang, and Zhiming Zheng

Informal Presentations
The Complexity of Computable Conditional Probability
Nathanael L. Ackerman, Cameron E. Freer, and Daniel M. Roy

Computably Categorical Boolean Algebras Enriched by Ideals and Atoms
Pavel Alaev

Comparing Nontriviality for the Exponential Time Classes E and EXP
Timur Bakibayev

Relativised Detour Degrees
Mathias Barra

The Limits of Decidability and Tractability in Growth-Rate Analysis
for Programs
Amir ben-Amram and Lars Kristiansen

Initial Segments of C.E. ibT-Degrees
Philipp Bodewig

Weihrauch Degrees, Omniscience Principles and Weak Computability
Vasko Brattka and Guido Gherardi

Independence of Axioms in Non-Classical Logics and ATP
Karel Chvalovský

The Reverse Mathematics of Artinian Rings
Chris Conidis

Modelling Quantum Information by Chu Spaces
Graziana Conte

Phase Transitions Related to the Pigeonhole Principle
Michiel De Smet and Andreas Weiermann

On Least Enumerations of Partial Structures
Angel V. Ditchev

Combinatorics, Chromosomal Rearrangements and Reconstruction
of Ancestral Genomes
Guénola Drillon
Quantum and Classical Analogue Computation Using Microwaves .... 330
Mark S. Everitt, Martin L. Jones, Vivien M. Kendon, Neil B. Lovett, and Robert C. Wagner

Fixed-Parameter Tractability, Relative Kernelization and the
Effectivization of Structural Connections ............................. 331
Michael Fellows, Juraj Hromkovič, Frances Rosamond, and Monika Steinová

Computing over Non-Linear Structures ................................. 332
Christophe Fouqueré

Hierarchies below the Halting Problem for Additive Machines 333
Christine Gaßner

n-Tardy Sets ............................................................ 334
Peter M. Gerdes

Empty Intervals of Enumeration Degrees and Strong Minimal Covers . 335
Matthew B. Giorgi, Thomas F. Kent, Andrew E. M. Lewis, and Andrea Sorbi

Elaborating Critical Points of Phase Transitions ........................ 336
L. Gordeeva and A. Weiermann

Noncuppable Enumeration Degrees Via Finite Injury .................. 337
Charles M. Harris

On the Form of Witness Terms ......................................... 339
Stefan Hetzl

Parallel Term Rewriting in Neural Networks ............................ 340
Ekaterina Komendantskaya

Definability of Combinatorial Functions ............................... 341
T. Kotek and J. A. Makowsky

The Semantics and Complexity of Successor-Free Nondeterministic
Gödel’s T and PCF ..................................................... 342
Lars Kristiansen and Bedeho Mesghina Wolde Mender

Computably Well-Founded ω-c.e. Linear Extensions .................. 343
Kyung Il Lee
TABLE OF CONTENTS

Random Reference-Switching Functions ................................. 344  
  Chung-Chih Li

Dynamic Algorithms for Reachability Games ........................... 345  
  Jiamou Liu

Some Properties of Traces as a Lattice under the Set Theoretic Inclusion 346  
  Gunther Mainhardt

Sequential Functionals at Type Level 2 ................................. 347  
  Dag Normann

Real Hypercomputation with Infinite Oracle Queries .................... 348  
  Arno Pauly

Index Sets and Scott Sentences ........................................ 349  
  Sara Quinn

A Crypto System Based on Symmetric Key Block Cipher ................ 350  
  N. Radha

Algorithmic Reducibilities and Borel Equivalence Relations .......... 351  
  Jan Reimann

On the Computational Content of Bolzano-Weierstraß .................. 352  
  Pavol Safarik

An Impact of Infinity: Comparison of Two Reverse Mathematics ...... 353  
  Kentaro Sato

A Characterization of the Join-Irreducible Medvedev Degrees ......... 354  
  Paul Shafer

Some Applications of the Jump Inversion Theorem for the Degree Spectra 355  
  Alexandra A. Soskova and Ivan N. Soskov

Embedding Countable Partial Orderings in Degree Structures .......... 356  
  Mariya I. Soskova and Ivan N. Soskov

On Semilattices of Σ-Degrees of Structures ........................... 357  
  Alexey Stukachev

Solving a PSPACE-Complete Problem with DNA Computing .............. 358  
  Thomas Zerjatke
On the Role of Quantum Metalanguage in Brain Quantum Computation 359
P. Zizzi and E. Pessa

Author Index ............................... 361
The first known mechanical calculator was discovered just over a century ago in a shipwreck off the coast of the Mediterranean island of Antikythera. It dates from the 1st century B.C.E., and is now known as the Antikythera Mechanism. This device is an order of magnitude more complicated than any surviving mechanism from the following millennium (it contained more than 30 gear wheels), and there is no surviving precursor. It is clear from its structure and inscriptions that its purpose was astronomical, including eclipse prediction. In this illustrated review, I will outline the new results from our international research team, which has been using the most modern imaging methods to probe the device and its functions. The extraordinary sophistication of the Mechanism’s design has fundamental implications for the development of Greek technology, astronomy and philosophy.
The Isomorphism Conjecture

Manindra Agrawal
Indian Institute of Technology, Kanpur

The Isomorphism Conjecture states that all many-one complete sets for NP under polynomial time reductions are polynomial time isomorphic to each other. The conjecture was made by Berman and Hartmanis in 1977 inspired, in part, by the analogous result for ce-complete sets under computable reductions. The conjecture has had a very interesting history with opinions changing on its correctness over time. In this talk, I will survey the results on the conjecture over the last 32 years.
Computability in Ergodic Theory

Jeremy Avigad

Department of Philosophy and Department of Mathematical Sciences
Carnegie Mellon University

In dynamical systems and ergodic theory, one tries to characterize the behavior of systems that evolve over time. These fields, however, often use methods that are difficult to interpret in computable terms.

For example, let $T$ be a measure-preserving transformation of a space $(X, B, \mu)$, let $f$ be a measurable function from $X$ to the real numbers, and for every $x$ in $X$ and $n$ in $\mathbb{N}$ let $(A_n f)_x = (fx + f(Tx) + \ldots + f(T^{n-1}x))/n$. The pointwise ergodic theorem says that this sequence of averages converges for almost every $x$, and the mean ergodic theorem says that the sequence $(A_n f)$ converges in the $L^2$ norm. But, in general, one cannot compute a rate of convergence from the initial data. Describing joint work with Philipp Gerhardy and Henry Townser, I will explain how proof-theoretic methods provide classically equivalent formulations of the ergodic theorems which are computably valid, and yield additional information.

For a second example, Szemerdi’s theorem is a combinatorial result that asserts that for any $\delta > 0$ and $k$ there is an $n$ large enough such that any subset of $\{1, \ldots, n\}$ of density at least $\delta$ contains an arithmetic progression of length $k$. In 1977, Furstenberg published a remarkable proof of Szemerdi’s theorem that involves a structural analysis of measure-preserving systems that is even harder to interpret in combinatorial terms. I will discuss this structure theorem, and computability-theoretic aspects thereof.
Ordinal Computability

Peter Koepke

Mathematisches Institut, Universität Bonn
Endenicher Allee 60, D 53115 Bonn, Germany
koepke@math.uni-bonn.de

Ordinal computability uses ordinals instead of natural numbers in abstract machines like register or TURING machines. We give an overview of the computational strengths of $\alpha$-$\beta$-machines, where $\alpha$ and $\beta$ bound the time axis and the space axis of some machine model. The spectrum ranges from classical TURING computability to $\infty$-$\infty$-computability which corresponds to GÖDEL’s model of constructible sets. To illustrate some typical techniques we prove a new result on Infinite Time Register Machines (= $\infty$-$\omega$-register machines) which were introduced in [1]: a real number $x \in \omega^2$ is computable by an Infinite Time Register Machine iff it is TURING computable from some finitely iterated hyperjump $0^{(n)}$.

For the full paper see the LNCS proceedings volume.

References


* Keywords: Ordinal machines, Infinite Time Register Machines, Constructible sets.
** The author wants to thank JOEL HAMKINS and PHILIP WELCH for a very inspiring discussion at the EMU 2008 workshop at New York in which the techniques and results bounding the strength of Infinite Time Register Machines were suggested and conjectured.
Foundations and Applications of Schema Mappings

Phokion G. Kolaitis

University of California Santa Cruz and IBM Almaden Research Center

Schema mappings are high-level specifications, typically expressed in some logical formalism, that describe the relationship between two database schemas. Schema mappings constitute the essential building blocks in formalizing the main data inter-operability tasks, including data exchange and data integration. The aim of this talk is to present an overview of recent developments in the study of schema mappings with emphasis on the interaction of this area of research with logic and computation.
Computing Using Positive Information

Andrea Sorbi

University of Siena, 53100 Siena, Italy,
sorbi@unisi.it,
http://www.dsmi.unisi.it/sorbi/

The need of formalizing a satisfactory notion of relative computability of partial functions leads to enumeration reducibility, which can be viewed as computing with nondeterministic Turing machines using positive information. Enumeration reducibility formalizes the idea of a partial function being computable relatively to another partial function, or the idea of a set being effectively enumerable relatively to (any enumeration of) another set.

If viewed as a reducibility on partial functions, enumeration reducibility gives rise to a degree structure, known as the partial degrees (Medvedev, Myhill). Of particular interest are also certain models of relative computability of partial functions (Kleene, Sasso, Davis) that give rise to reducibilities that are stronger than enumeration reducibility. All these reducibilities originate degree structures that embed the Turing degrees as the degrees of total functions. We will survey some of the most interesting properties of these degree structures, which have not been extensively studied.

If viewed as a reducibility on sets, then we get the degree structure known as the enumeration degrees (Friedberg and Rogers). The enumeration degrees are isomorphic to the partial degrees, so they still embed the Turing degrees. Indeed, interest in the enumeration degrees is often motivated by the fact that in many cases the wider context of the enumeration degrees allows one to recast and extend classical results on the Turing degrees with deeper and more insightful proofs. Again, a lively line of research consists in investigating positive reducibilities on sets that are stronger than enumeration reducibility, with emphasis often given to s-reducibility, which appears often in computability theory and applications.

Finally, we consider uniform positive computations, with applications to the Medvedev lattice.
I will rephrase the question, "What is a quantal reality?" as "What is aquan-
tal history?" (the word history having here the same meaning as in the phrase
sum-over-histories). The answer I will propose modifies the rules of logical in-
ference in order to resolve a contradiction between the idea of reality as a single
history and the principle that events of zero measure cannot happen (the Kochen-
Specker paradox being a classic expression of this contradiction). The so-called
measurement problem is then solved if macroscopic events satisfy classical logic,
and this can be decided by a calculation. The resulting conception of reality involves neither multiple worlds nor external observers. It is therefore
suitable for quantum gravity in general and causal sets in particular.
Combinatorial Algorithms for Convex Programs
Capturing Market Equilibria and Nash Bargaining Solutions

Vijay V. Vazirani
Georgia Institute of Technology
Atlanta, Georgia

Over the last 50 years, the primal-dual paradigm has had two highly successful "lives" – in combinatorial optimization and in approximation algorithms. In addition to yielding efficient and practically useful algorithms, it has also provided deep insights into the canonical combinatorial structure underlying the problems solved. Via a most unexpected route, having to do with the solution of some fundamental problems from game theory and mathematical economics, a third life of this paradigm appears to be emerging: combinatorial algorithms for solving certain classes of convex programs. Besides providing an in-depth perspective on this new development, I will introduce the fascinating problems that led to it and point out the challenges that lie ahead.
In recent years, research in algorithmic randomness has enriched classical computability theory with new notions and concepts, which give new insights to the subject. An example is the ‘highness’ notion of almost everywhere domination, which was introduced by Dobrinen and Simpson in [DS04]. Recall that, given functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$, we say that $f$ dominates $g$ if $f(n) \geq g(n)$ for almost all $n \in \mathbb{N}$.

**Definition 1 (Dobrinen and Simpson [DS04]).** A Turing degree $a$ is called almost everywhere (a.e.) dominating, if for almost all $X \in 2^\omega$ and all functions $g \leq_T X$, there is a function $f \leq_T a$ which dominates $g$.

Kurtz showed that $0'$ is a.e. dominating. This notion is very related to the highness property from classical computability theory: recall that a set $A$ is high if $A' \geq_T \emptyset''$. This means that if we can answer $\Sigma^0_1(A)$ questions, then we can answer any $\Sigma^0_1(\emptyset')$ question. In this sense, $A$ is close to the halting problem $\emptyset'$, hence the name ‘high’. Martin showed that $A$ is high iff it can compute a function which dominates all computable functions. Hence, it is easy to see that every a.e. dominating degree is high. Toward a characterization of the a.e. dominating degrees, Dobrinen and Simpson asked if this notion is equivalent to either highness or Turing completeness. It is now known that the class of a.e. dominating degrees lies strictly in between high and complete degrees, even in the local structure of computably enumerable degrees.

There has been an interest in clarifying the connections of this highness property with concepts from classical computability theory. For example, what role it plays in the partial ordering of the Turing degrees and whether it can be expressed purely in degree theoretic terms, without resorting to measure or randomness. In this respect, the following questions were raised.

Recall that a sequence of sets $(T_i)$ is a trace for a function $f$, if $f(n) \in T_n$ for all $n \in \mathbb{N}$. We say that $(T_i)$ has bound $h$, if $|T_n| < h(n)$ for all $n \in \mathbb{N}$. A degree $a$ c.e. traceable, if there is a computable function $h$ such that every function $f \leq_T a$ has a c.e. trace with bound $h$. A set $X$ is c.e. traceable by $Y$ if there is a computable function $h$ such that every function $f \leq_T X$ has a $Y$-c.e. trace with bound $h$. An analogous definition holds for degrees.

– (Simpson, 2006) Is there a minimal a.e. dominating degree?
– (Nies, Problem 8.6.4 in [Nie09]) Is there a c.e. traceable a.e. dominating degree?
Our main result shows that each a.e. dominating degree is array non-computable, which answers these questions in the negative.

**Theorem 1.** If $c$ is c.e. and is c.e. traceable by $a$, then no function that is computable in $c$ dominates every function computable in $a$.

**Corollary 1.** Every a.e. dominating degree is array non-computable. In particular, it is not minimal.

**References**


On Computably Enumerable Random Reals

Laurent Bienvenu

Department of Mathematics and Computer Science
University of Heidelberg, Germany

In this talk, we will discuss a particular class of reals that play an important role in algorithmic randomness: computably enumerable reals. Recall that a real $x$ is computably enumerable (c.e. for short) if the set of rationals $q < x$ is a c.e. subset of $\mathbb{Q}$ (or equivalently if there exists a computable increasing sequence of rationals that converge to $x$).

The first example of a c.e. random real was given by Chaitin [2]. He constructed a real (now referred to as Chaitin’s Omega) that can be seen as the halting probability of a universal (optimal) Turing machine $M$, and argued that $n$ bits of Omega are enough to decide the halting problem for $M$ on inputs of size at most $n$ (which in turns proves that $n$ bits of Omega have Kolmogorov complexity at least $n$).

Solovay observed that Chaitin’s Omega has an interesting property: among all c.e. reals, it is maximally hard to approximate, a property which we now called “Solovay completeness”.

More than 20 years after Chaitin and Solovay’s work, it was proven by Calude, Hertling, Khoussainov, Wang [3], Kučera and Slaman [4] that remarkably the three properties “being random”, “being Solovay complete” and “being the halting probability of an optimal Turing machine” are in fact equivalent for c.e. reals!

In this talk, we will present a unified proof of these results, in a non-technical and rather “visual” way (in particular, no prior knowledge of algorithmic randomness will be needed). If time permits, we will discuss more recent topics, such as lowness for Omega (see Miller [6]), Solovay functions (see Bienvenu and Downey [1]) and the links between c.e. reals and completions of Peano arithmetic (see Levin [5]).

References

Real time scheduling is concerned with tasks which periodically release jobs, each of which has to be finished before a certain deadline. Such scheduling problems play an increasing role, since microprocessors are replacing mechanical devices to control and trigger safety-critical applications, for example in fly-by-wire and drive-by-wire. I will show in this talk how classical problems in real-time scheduling, like response-time-computation are related to classical problems in integer programming, in particular to Diophantine approximation. I close with open problems from the interplay of Real-Time scheduling and the theory of integer programming.
Computable Numberings of Hyperarithmetical Sets and Complexity of Countable Models

Sergey Goncharov

gonchar@math.nsc.ru
Institute of Mathematics of SBRAS
Novosibirsk State University

This talk is connected with the study of computable numberings and some applications to complexity of countable models. The notion of a computable numbering of a family of arithmetical and hyperarithmetical sets was introduced by S. Goncharov and A. Sorbi in the framework of their general approach to the notion of computability for families of constructive objects admitting a formal description in some language with some given Gödel numbering of the formulas. We will discuss some problems relative to complexity of countable models of theories with countable isomorphic types of countable models relative to arithmetical and hyperarithmetical hierarchies and some another questions about complexity on countable models. We will consider different classes of theories with countable sets of countable isomorphic types of countable models.

References


Limits on Jump Inversion for Strong Reducibilities

Keng Meng Ng

Victoria University of Wellington

The study of strong reducibilities has been a major part of computability for a long time. As is well known, tabular reducibilities such as weak truth table (wtt-) and truth table (tt-) reducibilities place restrictions on oracle access. In recent times, truth table reducibility has become a central area of interest as it has been shown to be a natural reducibility to study in algorithmic randomness.

Recent work by Anderson analyzing an old result of Mohrherr renewed interest in the variations of the classical jump inversion theorems, particularly their connection with the strong reducibilities. We prove that the analogs of Shoenfield’s and Sacks’ jump inversion results fail when considering tt- and wtt-reducibilities, and in fact they fail in more or less the strongest way that they can. One of our main results show that for any computable sequence of $\Delta^0_2$ sets $\{V_e\}_{e \in \mathbb{N}}$, there exists a $\Delta^0_2$ set $S \geq_{tt} \emptyset'$ such that for every $e$, $V'_e \not\equiv_{wtt} S$.

We also examine the role of strong tabular reducibilities in the completion of pseudojump operators. We show that there is a pseudojump operator $V$ such that $V^X >_T X$ for every $X$, and there is no c.e. set $A$ such that $V^A \equiv_{wtt} \emptyset'$. 


Computational Strategies for the Heat Shock Response

Ion Petre

ABO Akademi Finnlnd

Elevated temperatures cause proteins in living cells to misfold. They start forming larger and larger aggregates that can eventually lead to the cell’s death. The heat shock response is an evolutionary well conserved cellular response to massive protein misfolding and it is driven by the need to keep the level of misfolded proteins under control. We consider in this talk a recently proposed new molecular model for the heat shock response in eukaryotes, consisting of a temperature-induced activation mechanism, chaperoning of misfolded proteins and self-regulation of the chaperon synthesis. We discuss: (i) how to build a mathematical model for the heat shock response, (ii) how to estimate the numerical values of its parameters, and (iii) how to validate the model. We also address the question of why is this level of complexity needed for implementing what could in principle also be achieved with a far simpler design.
Stochastic Programs and Hybrid Automata for (Biological) Modeling

Alberto Policriti
Universita di Udine, Italia

We present a technique to associate to stochastic programs written in stochastic Concurrent Constraint Programming a semantics in terms of a lattice of hybrid automata. The aim of this construction is to provide a framework to approximate the stochastic behavior by a mixed discrete/continuous dynamics with a variable degree of discreteness.

Work in collaboration with: Luca Bortolussi
On the Relations Between Effective, Hyper-, and Physical Computation

Oron Shagrir

Hebrew University of Jerusalem

My aim is to examine the relations between three notions of computation: effective, hyper and physical computation. The question is which constraints are imposed and which are violated when we switch from one notion to another. I discuss this question in the context of physical machines that compute non-recursive functions by performing super-tasks. There are three types of such supertask machines: accelerating (Copeland), shrinking (Davies), and M-H machines (Pitowsky, Hogarth). It is assumed that these hyper-computers are non-effective since they can accomplish infinitely many steps in finite time. I will argue, however, that this constraint is compatible with effectiveness. These hyper-computers are not effective in that they assume a different notion of determinism. Effectiveness requires that every (but the initial) state is to be determined by the previous configuration of the machine. But in these hyper-computers the end state is not determined this way, but is the limit of the series of configurations that precede it. The constraint of infinitely-many-steps-in-finite-time is introduced when we want to implement these hyper-computers in physical systems; it is thus a constraint on physical computation. When all is in place (and time is allowed), we can resolve Copeland’s paradox.
Domination, Forcing, Array Nonrecursiveness and Relative Recursive Enumerability

Richard A. Shore
Cornell University
Ithaca NY 14853 USA

We present some abstract theorems showing how domination properties equivalent to not being $\text{GL}_2$ or array recursive can be used to construct sets generic for different notions of forcing. These theorems are then applied to give simple proofs of several old results. We also give a direct uniform proof of a recent result of Ambos-Spies, Ding, Wang and Yu [2009] that every degree above any not in $\text{GL}_2$ is recursively enumerable in a 1-generic degree strictly below it. Our major new result is that every array nonrecursive degree is r.e. in some degree strictly below it. Our analysis of array nonrecursiveness and construction of generic sequences below $\text{ANR}$ degrees also reveal a new level of uniformity in these types of results. This is joint work with Mingzhong Cai.
From Optimal Algorithms to Optimal Mechanisms

Marc Uetz

University of Twente, Applied Mathematics, P.O. Box 217, 7500 AE Enschede, The Netherlands, m.uetz@utwente.nl

Abstract. The talk gives an introduction into optimal mechanism design, exemplified on recent results on the design of optimal mechanisms for single machine scheduling problems from [1].

Mechanism design can be compared with the design of algorithms for classical optimization, with the difference that it adds another dimension: Next to constraints such as resource or time constraints, in the design of mechanisms we also address incentive constraints. These are a result of private information of individuals. Starting from the assumption that some of the relevant data is private to selfish agents, they might be tempted to act untruthfully, for example in reporting about this data. Thereby they can manipulate the outcome of the mechanism towards their own benefit. In short, private information creates an incentive for gaming. This way, classical optimization problems get a game theoretic dimension. The solution is to complement the outcome by payments to the agents, and extend the constraint set by conditions that rule out the incentives for manipulation. These constraints are called incentive constraints.

The design of optimal mechanisms is recognized as an intriguing issue in the economic literature since it was first studied by Myerson [4] for the case of single item auctions. There, the goal is to design an auction in which bidders place bids to win a single item, such that the auction maximizes the sellers expected revenue. More generally, we can say that optimal mechanism design is concerned with coordinating the selfish behavior of non-cooperative agents, while minimizing the total expected cost for the mechanism designer.

In this talk, we will review some basic concepts and key results for optimal mechanism design, including the link between mechanism design and graph theory, which has recently been proven to be very fruitful: A graph theoretic interpretation of the incentive constraints, as used by Malakhov and Vohra [3] for example, allows not only new characterizations of classical concepts in mechanism design, such as the famous revenue equivalence theorem [2], but it is also helpful to derive new insights and results, including optimal mechanisms in concrete settings.

To exemplify this, we study the design of optimal mechanisms in a setting where jobs of different durations compete for being processed by a single server that can only handle one job at a time. Jobs have weights expressing their individual cost for waiting. In the classical combinatorial optimization setting,
this problem has a trivial optimal solution that can be found in $O(n \log n)$ time, also known as Smith’s rule [5]. We show how this result can be carried over to a setting where the weights are private to the jobs, and jobs need to be compensated for waiting. The resulting mechanism deploys a modification of Smith’s rule to minimize total expected costs for the mechanism designer. We also discuss the difficulties that arise when not only the weights, but also the processing times are private information. For more details, see [1].

The talk is based on joint work with B. Heydenreich, D. Mishra, R. Müller, and R. Vohra.

References

Bayesian Data Integration for Predicting Gene Function in Malaria

David R. Westhead
University of Leeds

Malaria caused by the parasite Plasmodium falciparum is one of the world's most deadly diseases, and with growing resistance to conventional drugs, new treatments and vaccines are urgently needed. Understanding the function of each P. falciparum gene is likely to help this cause but unfortunately the sequencing of the P. falciparum genome in 2002 suggested that 60% of its genes could not be assigned a function by homology methods. Therefore, new methods of gene function prediction are needed to help annotate many of these uncharacterised P. falciparum genes. To address this, we train a naive Bayes classifier on multiple sources of data and subsequently apply a modified version of the Gene Set Enrichment Analysis Algorithm to predict gene function in P. falciparum. To define gene function, we exploit the hierarchical structure of the Gene Ontology, specifically using the Biological Process category. We demonstrate the value of integrating multiple data sources by achieving accurate predictions on genes that cannot be annotated using simple homology based methods.

In collaboration with: Philip M. R. Tedder, James R. Bradford, Chris J. Needham, Glenn A. McConkey, Andrew J. Bulpitt
A Note on Universal Numberings

Kuanysh Abeshev and Serikzhan Badaev

1 Kazakh National University, 71 Al-Farabi ave., Almaty 050038, Kazakhstan, kuanqk@gmail.com,

2 Kazakh National University, 71 Al-Farabi ave., Almaty 050038, Kazakhstan, Serikzhan.Badaev@kaznu.kz

Abstract. The problems caused by non-monotonic uniform enumerations are discussed. It is shown that the collection of finitely many finite extensions of a set in a given class in the Ershov hierarchy forms a universal subset of that class.

Key words: computable numbering, universal numbering, hierarchy.

1 Introduction.

Roughly speaking, a principal numbering for a class of numberings is a numbering in the class which can simulate any numbering in the class. If we consider the computable numberings of the unary partial computable functions, i.e. the uniformly computable sequences \( \psi_0, \psi_1, \ldots \) of the unary partial computable functions, then the standard Gödel numbering \( \varphi_0, \varphi_1, \ldots \) is a classical example of a principal numbering, since for any such sequence, \( \psi_e = \varphi_{f(e)} \) for some computable function \( f \) and all \( e \in \omega \). Analogously, the standard Gödel numbering \( \{W_e\}_{e \in \omega} \) of the computably enumerable (c.e. for short) sets is another example of a principal numbering for the class of c.e. sets.

We follow the monograph [3] of Yu.L. Ershov in Russian and the survey papers [1], [4] for terminology and notations that are commonly used in the theory of numberings. A mapping \( \alpha : \omega \rightarrow A \) of the set \( \omega \) of natural numbers onto a family \( A \) of c.e. sets is called a computable numbering of \( A \) if the set \( \{(x, n) \mid x \in \alpha(n)\} \) is c.e. And a family \( A \) of subsets of \( \omega \) is called computable if it has a computable numbering. In other words, a computable family \( A \) is a uniformly c.e. class of sets, and every computable numbering \( \alpha \) of \( A \) defines a uniform c.e. sequence \( \alpha(0), \alpha(1), \ldots \) of the members of \( A \). A numbering \( \alpha \) is called reducible to a numbering \( \beta \) (in symbols, \( \alpha \leq \beta \)) if \( \alpha = \beta \circ f \) for some computable function \( f \). Two numberings \( \alpha, \beta \) are called equivalent if they are reducible to each other.

If \( \text{Com}(A) \) stands for all computable numberings of a computable family \( A \) of c.e. sets then a numbering \( \alpha : \omega \rightarrow A \) is called principal if \( \alpha \in \text{Com}(A) \) and \( \beta \leq \alpha \) for each numbering \( \beta \in \text{Com}(A) \). It is easy to see that \( \alpha \) is a computable numbering if and only if \( \alpha \leq W \), i.e. \( \alpha(e) = W_{f(e)} \) for some computable function \( f \) and all \( e \in \omega \). Thus, the numbering \( W \) is a principal numbering of the class.
\( \Sigma^0_n \) of all c.e. sets. For arbitrary numberings \( \alpha \) and \( \beta \) with \( \alpha = \beta \circ f \), we can think of \( \alpha \) as being computable relatively to \( \beta \) (\( f \) allows to simulate \( \alpha \) from \( \beta \)). Therefore, a principal numbering of a computable family \( \mathcal{A} \) of c.e. sets is just one by means of which we can simulate all possible uniform computations of the sets from \( \mathcal{A} \).

For a given computable family \( \mathcal{A} \) of c.e. sets, two main ways of constructing principal numberings are known. The first way is based on the idea of considering uniform computations of all computable numberings, or at least of witnesses from each equivalence class of numberings, lying in \( \text{Com}(\mathcal{A}) \). Essentially, this way is epitomized in Rice’s description of the classes of c.e. sets whose index sets in \( W \) are c.e.

The second way originated from the notion of a standard class, introduced by A. Lachlan in [6]. Generalizations of the notion of standard class by A. I. Mal’tsev [7] and Yu. L. Ershov [3] provided very fruitful tools for constructing principal numberings. We will discuss them later. Now we formulate one of the finest results on principal numberings.

**Theorem 1** (Lachlan, [6]). Every finite family of c.e. sets has a principal numbering.

The aim of our talk is to discuss the main problems concerning principal numberings, when dealing with generalized computations.

## 2 Computable Numberings in Hierarchies.

In [5], S. S. Goncharov and A. Sorbi offered a general approach for studying classes of objects which admit a constructive description in a formal language via a Gödel numbering for formulas of the language. According to their approach, a numbering is computable if there exists a computable function which, for every object and each index of this object in the numbering, produces some Gödel index of its constructive description.

We use below the symbol \( \Sigma^{-1}_n \) to denote the class of level \( n \) of the Ershov hierarchy of sets, whereas as usual \( \Sigma^0_n \) denotes the class of level \( n \) of the arithmetical hierarchy. The notion of a computable numbering for a family \( \mathcal{A} \) of sets in the class \( \Sigma^i_n \), with \( i \in \{-1,0\} \), may be deduced from the Goncharov–Sorbi approach as follows. A numbering \( \alpha \) of a family \( \mathcal{A} \subseteq \Sigma^i_n \) is \( \Sigma^i_n \)-computable if \( \{\langle x, m \rangle : x \in \alpha(m)\} \in \Sigma^i_n \), i.e. the sequence \( \alpha(0), \alpha(1), \ldots \) of the members of \( \mathcal{A} \) is uniformly \( \Sigma^i_n \). We will denote the set of all \( \Sigma^i_n \)-computable numberings of a family \( \mathcal{A} \subseteq \Sigma^i_n \) by \( \text{Com}^i_n(\mathcal{A}) \). Since \( \mathcal{A} \subseteq \Sigma^i_n \) implies \( \mathcal{A} \subseteq \Sigma^i_m \) for all \( m > n \), it follows that we should be careful in defining the notion of principal numbering.

**Definition 1.** Let \( \mathcal{A} \subseteq \Sigma^i_n \) and let \( m \geq n \). A numbering \( \alpha : \omega \rightarrow \mathcal{A} \) is called universal in \( \text{Com}^i_m(\mathcal{A}) \) if \( \alpha \in \text{Com}^i_m(\mathcal{A}) \) and \( \beta \leq \alpha \) for all \( \beta \in \text{Com}^i_m(\mathcal{A}) \).

The precise meaning of the phase “a uniform \( \Sigma^i_n \) sequence \( \alpha(0), \alpha(1), \ldots \) of the members of \( \mathcal{A} \)” can be explained as follows. Let \( A(n, x, t) \) denote a function satisfying the following conditions:
1. range(A) ⊆ {0, 1};
2. A(e, x, 0) = 0, for all e and x.

We can treat this function as uniform procedure for computing the sets α(e).
Given e and x, A(e, x, 0) = 0 means that initially the number x is not enumerated into α(e). The number x stays outside of α(e) until the function λA(e, x, t) changes its value from 0 to 1. When this happens, the number x is enumerated into α(e). Now, x remains in α(e) until λA(e, x, t) changes the value from 1 to 0. In this case, the number x is taken off the set α(e). And again we wait for the value of λA(e, x, t) to change from 0 to 1, to put x into α(e) for the second time, and so on.

It is easy to check that, for A ⊆ \( \Sigma^0_1 \), a numbering α is \( \Sigma^0_1 \)-computable if and only if there exists a computable function A such that, for all e, x, \( \lambda A(e, x, t) \) is a monotonic function, and
\[
x \in \alpha(e) \iff \lim_{t} A(e, x, t) = 1.
\]

If \( A \subseteq \Delta^0_2 \) then a numbering α is \( \Delta^0_2 \)-computable if and only if there exists a computable function A such that, for all e, x,
\[
\lim_{t} A(e, x, t) \text{ exists, and } x \in \alpha(e) \iff \lim_{t} A(e, x, t) = 1.
\]

If \( A \subseteq \Sigma^0_2 \) then a numbering α is \( \Sigma^0_2 \)-computable if and only if there exists a computable function A such that, for all e, x,
\[
x \in \alpha(e) \iff \lim_{t} A(e, x, t) \text{ exists and } \lim_{t} A(e, x, t) = 1.
\]

If \( A \subseteq \Sigma^0_{n+1} \) then a numbering α is \( \Sigma^0_{n+1} \)-computable if and only if there exists a computable function A such that, for all e, x,
\[
|\{t : A(e, x, t + 1) \neq A(e, x, t)\}| \leq n + 1
\]
where the symbol \( |X| \) denotes the cardinality of a given set X.

For a \( \Sigma^0_n \)-computable numbering α, we say that such a computable function A represents a \( \Sigma^0_n \) computation of α(e).

Finally, for families \( A \subseteq \Sigma^0_{n+3} \) we can use criteria of computability similar to the ones given above, but with the relevant function A computable relatively to the appropriate iteration of the jump of the empty set.

Note that the computable function \( A(e, x, t) \) above is monotonic in \( t \) only in the classical case of c.e. sets (i.e. \( A \subseteq \Sigma^0_1 \)). It seems that the non-monotonic behavior of this function is the main reason for Theorem 1 to fail in all non-classical cases. We recall the following known result.

**Theorem 2** (Badaev, Goncharov, Sorbi, [2]). *Let A be any finite family of \( \Sigma^0_{n+2} \) sets. Then A has an universal numbering in Com\(^+\)\(^{n+2}\)(A) if and only if A contains a least set under inclusion.*
3 Universal Numberings of Finite Families in the Ershov Hierarchy.

In this section, we try to find universal numberings for some finite families in the hierarchy of Ershov. We are not sure whether such numberings can be found for all finite families.

Theorem 3. For every $n$, the class $\Sigma_{n+2}^{-1}$ of the Ershov hierarchy has a universal numbering in $\text{Com}_{n+2}^{-1}(\Sigma_{n+2}^{-1})$.

The proof is straightforward since it is easy to construct uniformly all $\Sigma_{n+2}^{-1}$-computable numberings for all $\Sigma_{n+2}^{-1}$-computable families. We will denote this universal numbering by $W^{(-1,n+2)}$.

The next theorem originated as an attempt to adapt the idea of a $wn$-subset, to the class $\Sigma_{n+2}^{-1}$ equipped with the numbering $W^{(-1,n+2)}$. We will use the notion of $wn$-subset in a form which is slightly different from the original one of Ershov, [3].

Definition 2. A family $A \subseteq \Sigma_k^i$ is called a $wn$-subset of $\Sigma_k^{-1}$ if there exist a c.e. set $I$ and a sequence $\{V_e\}_{e \in \omega}$ such that

1. $I$ contains the index set of the family $A$ with respect to the numbering $W^{(-1,k)}$;
2. $V$ is a $\Sigma_k^{-1}$-computable numbering;
3. for every $e \in I$, $V_e \in A$;
4. for every $e \in I$, if $W_e^{(-1,k)} \in A$ then $V_e = W_e^{(-1,k)}$.

Lemma 1. If a family $A \subseteq \Sigma_k^i$ is a $wn$-subset of $\Sigma_k^{-1}$ then $A$ has a universal numbering in $\text{Com}_{k}^{-1}(A)$.

Proof. The proof is a straightforward modification of the original proof of Ershov, [3].

Theorem 4. Let $k > 1$ and $m > 0$ be any numbers. If $F_0, F_1, \ldots, F_m$ is a sequence of finite sets and $B \in \Sigma_k^{-1}$ is a set such that no $F_i$ in the sequence intersects $B$, then the family $A = \{B \cup F_i : i \leq m\}$ is a $wn$-subset of $\Sigma_k^{-1}$.

Proof. We build by a stage construction a computable function $V(e, x, s)$ which will represent a numbering $V$. The set $I$ will be defined at the end of the construction.

Let $B(x, s)$ be a computable function which represents a $\Sigma_k^{-1}$ computation of $B$. We can assume that $B(x, s) = 0$ for all pairs $(x, s)$ with $x \in F_0 \cup F_1 \cup \cdots \cup F_m$. We let $V(e, x, s) = B(x, s)$ for all triples $(e, x, s)$ with $x \notin F_0 \cup F_1 \cup \cdots \cup F_m$. So, we can fix $e$ and describe, uniformly in $e$, how to construct the values needed to define $V(e, x, s)$ for all pairs $(x, s)$ with $x \in F_0 \cup F_1 \cup \cdots \cup F_m$.

We let $P(e, x, s)$ be a computable function which represents the numbering $W^{(-1,k)}$, and denote $F_0 \cup F_1 \cup \cdots \cup F_m$ by $F$. 

26
Construction. For \( s = 0 \) we let \( V(e, x, 0) = 0 \) for all \( x \in F \). For the definition of \( V(e, x, s + 1) \) we distinguish the following cases.

*Case 1*: There exists \( i \leq m \) such that

\[
P(e, x, s) = 1 \iff x \in F_i
\]

for all \( x \in F \).

Then let

\[
V(e, x, s + 1) = \begin{cases} 
1, & \text{if } x \in F_i; \\
0, & \text{if } x \in F \setminus F_i.
\end{cases}
\]

*Case 2*: Otherwise, let \( V(e, x, s + 1) = V(e, x, s) \) for all \( x \in F \).

Now, we define the set \( I \). If the sequence \( F_0, F_1, \ldots, F_m \) contains the empty set then let \( I = \omega \). Otherwise, let

\[
I = \{ e : \exists x \exists s (x \in F \land V(e, x, s) = 1) \}.
\]

It remains only to check that the requirements of definition 2 are satisfied by this sequence \( \{V_e\}_{e \in \omega} \) and the set \( I \).

**Question 1.** Do there exist finite families in the Ershov hierarchy without universal numberings?

**References**

Automatic classification of protein sequences into families using elongated K-mean spectral clustering

Ulavappa. B. Angadi* and M. Venkatesulu
Department of Computer Applications, Kalasalingam University
Krishnankoil, Srivilliputtur (via), Tamil Nadu, India, 626 190.
angadiub@gmail.com
venkatesulu_m2000@yahoo.com

Abstract. An important and difficult problem in bioinformatics is automatic classification of protein sequences into families, only when sequences information is available and some functionally related sequences share low similarity. Two critical issues remain unsolved, the one is to determine number of families and super families in given data set and the other is effective clustering of meager data. Spectral algorithm is based on spectral theory, random walk and normal cut, and uses the leading eigen vectors. Spectral theory allows the classification of proteins into families based on a global treatment. The proposed elongated K-mean spectral clustering technique can automatically determine the number of clusters/families and classify set of proteins into families/super families, which are very close to SCOP families/super families. The results obtained from different sets of data indicate that the proposed approach leads to reasonable good clusters.

Key words: protein clustering, classification

1 Introduction

In bioinformatics, enormous biological data is being accumulated in public database due to genome-sequencing projects all over the globe. Exploiting and compelling need to transform biological data into useful information, and knowledge about biological functions and structures is becoming an important and challenging task to both computer scientists and biologists. Biological sequence analysis and classification are major directions in predicting the function or the structure of sequences.

An important and intricate setback in bioinformatics is automatic classification of protein sequences into families based on only sequence information, and some functionally related sequences share low sequence similarity. In addition such classification gives insight into the features associated with the biological role of the families.

Many methods are currently available for clustering of protein sequences into families and super families. In spite these methods, still two critical issues remain unresolved: (i) to determine number of families and super families in a given data set and (ii) to effective clustering on given meager data (some functionally related sequences having low similarity)

Most of previous approaches assumed that the number of clusters are known and fixed manually. Some methods are crucial to set the threshold to a value that will provide number of clusters. If the threshold value is too conservative then only very close sequences will be assigned to the same cluster. This type of grouping would not be very informative, since it is affirmed being that proteins that are very similar in sequences have the likelihood of evolutionary related; moreover, a conservative threshold is likely to generate many singleton clusters. On other hand, a relaxed threshold would have the opposite effect of including many unrelated proteins into the same cluster.

Traditionally, most of the methods deal with similarity relationships in a pair wise manner. In many cases sequences have diverged to the extent that their common origin is untraceable by a direct sequence comparison, means that proteins with lower sequence identities but whose
functions and structures are very similar for example globins like proteins identities are only 15% [1]. In such case these methods fail to provide correct information about the functionality of the sequence. To overcome this issue, some of the approaches start by building a multiple alignment of sequences. These methods suffer from a number of drawbacks such as sensitive on initialization, computationally expensive and etc.

Elongated K-mean spectral clustering algorithm is based on spectral theory, random walk and normal cut. This approach for clustering of protein sequences is fast and highly accurate. It avoids the problems mentioned above, while spectral theory allows the classification of proteins into families based on a global treatment of all relationships in similarity space simultaneously. Spectral method use the leading eigenvectors of a matrix derived from the similarity information among data. They deal with the study of global properties of a dataset by making only local (pair wise) similarity measurement between data points. The global properties that emerge are best understood in terms of a random walk formulation on the graph. Spectral-based method have shown empirical success and a recent work has highlighted the conditions under which these methods can be expected to perform well in protein sequence classification [2][3]. The Spectral method has gained popularity in a variety of application domains from segmenting images to clustering parallel scientific computation tasks. Spectral clustering has come to the fore in recent years as powerful approaches to a variety of clustering problems. In protein sequence data, spectral clustering assigns a protein to a cluster by taking into account of all the distances between every pair of proteins in the set.

Typically, spectral clustering algorithm [4] involves constructing an affinity matrix from the data, calculating diagonal matrix whose \( D(i, i) \) element is the sum of A’s \( i^{th} \) row, construct the Matrix \( L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \), finding \( x_1, x_2, \ldots, x_k \) k largest eigenvectors and forming the matrix \( X = [x_1, x_2, \ldots, x_k] \), renormalizing each of X’s row to have unit length and finally applying K-mean or any other classification algorithm. In this paper, we refer spectral clustering of protein sequences [2] as spectral clustering and the proposed algorithm automatically determines the number of clusters using elongated mean spectral algorithm [5] referred to as elongated mean spectral clustering.

The proposed algorithm will perform better classification compared to other spectral based techniques [2][3] used to classify protein sequences into families and super families. We use the random subsets of the SCOP database. BLAST[7] E-value is used to prepare similarity matrix and then we apply elongated mean spectral clustering algorithm. We show how these results are superior to the results obtained from spectral algorithm[2] and TribeMCL [3] with E-value as similarity measures between pair of sequences. By running these algorithms on several data sets using E-values by various substitution matrices (BLOSUM45, BLOSUM62, BLOSUM80, PAM30 and PAM 70) and compare performance measures defined in terms of precision and recall, F-measures and time and space complexity.

2 Materials and Methods

2.1 Spectral clustering

In order to apply spectral method to clustering of protein sequences, it is re-formulated as the problem of partitioning a weighted undirected graph into a set of discrete clusters. Each node in the graph corresponds to a protein sequence and the weight on each edge corresponds to similarity between the two protein sequences. The discrete graph partitioning is an NP hard problem. Here, eigenvectors of a graph’s Laplacian are used for a solving a relaxation of an NP-hard problem. Then partitioning the graph is based on K largest eigenvectors.
2.2 Automatic determination of families using elongated mean spectral method

Finding a suitable similarity metric among sequences is one of the main issues in clustering the protein sequences. As used by others, we have also chosen BLAST E-values as similarity measure. We construct an affinity matrix $S(i,j)$ representing similarity measure between sequences $i$ and $j$ using BLAST E-value by running BLAST all-by-all and convert into p-values using sigmoid function. In general, the matrix will not be symmetrical and we obtain a symmetric matrix by assigning to each $S(i,j)$ and $S(j,i)$, the higher value of two values and $S(i,i)=1$. In this, we have done numerous experiments using BLAST global identities and BLAST E-Values with sigmoid convention function. The BLAST E-Values performed better than BLAST identities for protein sequences. But, in case of DNA sequences global identities are better than E-values. We run the algorithm on all three data with various substitution matrices such as BLOSUM45, BLOSUM62, BLOSUM80, PAM30 and PAM 70. In these BLOSUM62 show better results (Figure 1). We find that BLOSUM62 matrix produces reliable results and hence, we employ BLOSUM62 matrix with default parameters (BLOSUM62, 11, 1, 0.267, 0.041, 0.14, 1.9, -30) as substitution matrix in our experiment.

Algorithm-1: Automatic determination of families

Given a dataset consisting of N x N symmetric similarity matrix S.
1. Form N x N an affinity matrix by BLAST P-value
2. Normalise $L = D^{-1/2} S D^{-1/2}$
3. Compute $K$ eigenvectors with largest eigenvalues of $L$ and form matrix $X$ of N x K
4. Initialize $q = 2$ and two centers from rows of matrix $X$ on maximum value in 1$^{st}$ and 2$^{nd}$ columns
5. Select the first $q$ columns from matrix $X$ and assemble them in N x q of Matrix $Y$ and initializing the $q+1$ center in the origin.
6. Perform elongated K-means cluster with $q+1$ centers on $Y$ (Algorithm-2).
7. If the $q+1$ th cluster contain any data points, then there must be at least an extra cluster; set $q=q+1$ and go back to step 5. Otherwise, end algorithm.

Algorithm -2: Elongated K-means algorithm (Given centers)

1. For each $c_i$ compute the distance of all points $x$ from it as follows:
   
   If $c_i^T c_i > e$ i.e if the centre is not very near the origin ($e$ is a parameter to be fixed by the user)
   
   $e$-dist($x, c_i$) = $(x-c_i)^T M (x-c_i)$
   
   where $M = 1/\lambda (I_q - c_i^T c_i/c_i^T c_i) + \lambda c_i^T c_i/c_i^T$

   $\lambda$ is the sharpness parameter that controls the elongation (the smaller, the more elongated the clusters)

   If the centre is very near the origin, $c_i^T c_i \leq e$, the distances are measured using the Euclidean distance.

2. Using this distance measure, assign each point $x$ to the nearest centre. Update the location for each centre by taking the mean of all the data assigned to it.

3. Return to step 1 and repeat until there is no change in the location

We present two principal points as to why elongated mean clustering works better than the spectral k-mean and MCL. i) Mahalanobis distance measures (elongated mean) and ii) Adding a center in origin.
**Mahalanobis distance**

The most important question in clustering algorithm is distance measure. Euclidean distance is the most commonly used distance metric in practice. Euclidean distance favors hyperspherical shaped clusters of equal size. The clusters in a data are not always compact, isolated and hyperspherical. Therefore Euclidean distance has the undesirable effect on splitting elongated and ellipsoid shaped clusters. Intuitive eigenvectors are non-spherical and generally in ellipsoid.

\[
d(x,c) = \sqrt{(x-c)^t \Sigma^{-1} (x-c)}
\]

and with origin \(d(x,0) = \sqrt{x^t \Sigma^{-1} x}\).

Points with the same distance of the origin is \(x^t \Sigma^{-1} x = c^2\), which is the general equation of an ellipsoid centered at the origin.

We can obtain useful insight into the eigen decomposition of the affinity matrix by recalling that finding the eigenvector corresponding to the largest eigenvalue of a K-dimensional symmetric matrix \(A\)

\[
A = \lambda_1 e_1 e_1^t + \lambda_2 e_2 e_2^t + \cdots + \lambda_p e_p e_p^t
\]

A symmetric KxK matrix A is positive definite matrix \(e^t A e > 0\) and eigenvectors are mutually orthogonal and is equivalent to the following optimization problem

\[
\text{Max}(e^t A e)\ \text{subject to}\ e^t e = 1
\]

**Adding a center with zeros**

Eigenvectors are mutually orthogonal \(e^t e = 1\). Let \(L_i\) be eigenvector associated with \(r_{ni}\) rows of the eigenvector matrix are distributed along with vector \(L_i\), \(i = 1..K\) and \(1 < n_i < n\). Any set of mutually orthogonal vectors in the K dimensional clustering space will equally qualify as an admissible basis of clustering eigenvectors. Therefore, rows of the matrix that has as columns the clustering eigenvectors will cluster along K mutually orthogonal axes in K dimensional space.

In general the centers of the observations of two orthogonal vectors will differ from the origin. If a centre is within a cluster, all the points in the cluster will be very near to it, while points in another cluster will be farther from centre of other clusters than from the origin.

**3 Results**

In this experiment we consider the problem of clustering protein sequences according to their evolutionary relatedness and particularly some related proteins have very low sequence similarity such as Globin-like [1].

**3.1 Analysis of E-Values within and between families/superfamilies**

A protein in the same superfamily/family can display varying degree of sequences similarity. Regarding this, we perform an all-against-all BLAST and examine the distribution of BLAST E-values within and between families/superfamilies. We calculate the central domain by minimum E-value all-against-all within the family and find out the closest domain with central domain between families/superfamilies (Figure 1-left). We analyze by visualization of distribution of E-values of all domains in the family versus to all domains of other families (Figure 1-right). By visual itself we can understand how the domains of globins-like super families closest to other domains belongs to a different super family. For this analysis we use the data of 342 domains belonging to 5 super families and 13 families. In this type of data, a threshold does not exist for a perfect clustering. Further, we perform k-mean clustering on E-values to show how difficult is clustering using sequence distance (Fig.3-right-down) and clustering is quantified by F-measures with superfamilies:0.4468 for data set 507, 0.4587 for data set 457 and 0.4587 for data set 342.
3.2 Analysis with various substitution matrices

The usage of appropriate substitution matrices other than BLOSUM62 may give crucial clusters. Hence, we run the elongated mean spectral algorithm with various substitution matrices: BLOSUM45, BLOSUM62, BLOSUM80, PAM30 and PAM70 on all three data sets. F-measures and number of clusters identified by the algorithm are shown figure 2 BLOSUM 45 and BLOSUM 62 are better than BLOSUM 80, PAM30 and PAM70 for the 342 and 507 data sets. BLOSUM 62 and BLOSUM 80 are better than BLOSUM 80, PAM30 and PAM70 for the 457 and 507 data sets. Overall BLOSUM 62 is resulted better in all data sets. Therefore, in our experiments we choose BLOSUM 62 substitution matrix for further experiments to compare elongated spectral algorithm with other spectral techniques. In this analysis, impact of number of sequences in a family can be assessed. The data set 342 and 457 shows better than 507, because half of the families in the data set 507 are singleton families. On this it can be conclude, the proposed algorithm works better on non-singleton families data set.

3.3 Analysis elongated spectral clustering with other spectral technique.

There are a number of published methods available, the recently published spectral clustering[2] and TrimbeMCL[3] shows better results over all other methods and proved that the application of spectral theory in clustering of protein sequences is efficient. Further we prove that our algorithm out performs compared to all the above algorithms.
In this paper, to evaluate the performance of the algorithm, we present the results obtained by our method and compare them with the results obtained from spectral clustering[2] and TrimbMCL[3] and we use the data set of 507 sequences used by Alberto Paccanaro[2], and two more datasets 457 sequences and 342 sequences. These data sets were collected from SCOP and having very low similarity measures between sequences within the families and superfamilies. In 507 sequences dataset, data belongs to 6 super families namely Globin-like(88), EF-hand(83), Cuperdoxins(78), (Trans)glycosidases(83), Thioredoxin-like(81) Membrane all-alpha(94). The data set contained six superfamilies and 51 families.

Our algorithm successfully separates protein sequences into superfamilies and families according to SCOP classification. The SCOP database is manually curated database contains extensive information relating to protein domains and structures, and proteins organized in hierarchical manner at four main levels: class, fold, super family, family. These proteins have been expertly classified into families based on their folding patterns and a variety of other information. Family information given is well understood and accurately presented.

Families

Visualization results are presented in Figure-3 for the 507 sequences dataset from the proposed algorithm, the elongated mean spectral clustering algorithm correctly separated all the members of Globin like superfamily and grouped into 3 families; Globins, Phycocyanin-like, truncated hemoglobin and nerve tissue mini-hemoglobin together. EF-hand proteins alienated correctly, but it grouped the 8 families into 3 groups; Calbindin D9K members clustered correctly with some part of S100 proteins, Osteonectin, Parvalbumin, Calmodulin-like, Eps15 homology domain (EH domain) in one cluster and EF-hand modules in multidomain proteins and penta-Ef hand proteins together in another cluster. In Cuperdoxins superfamily class only one sequence from (Trans) glycosidases is wrongly assigned to Cuperdoxins, regarding grouping into families, it splits 5 families into 2 classes; one is Plastocyanin/azurin-like and Periplasmic domain of cytochrome c oxidase subunit II with only two proteins are wrongly presented between these two classes and another is Multidomain cupredoxins, Nitrosocyanin and Ephrin-b2 ectodomain with hardly only one wrong member. Our algorithm classified the (Trans) glycosidases proteins into 4 classes: i) alpha-Amylases and beta-Amylase with two wrong members, ii) beta-glycanases are grouped correctly except one protein, iii) All Type II chitinase proteins are grouped correctly, and iv) In this, Family I of glycosyl hydrolase, beta-N-acetylhexosaminidase catalytic domain, Beta-D-glucan exohydrolase, 1 4-beta-N-acetylmuraminidase and Bee venom hyaluronidase are merged together. In Thioredoxin-like superfamily, the families Thioltransferase, PDI-like, spliceosomal protein U5-15Kd, Disulfide bond isomerase, Glutathione peroxidase-like, Thioredoxin-like 2Fe-2S ferredoxin and Arsenate reductase ArsC are distributed in two clusters except 4 sequences; Calsequestrin and Disulphide-bond formation facilitator (DSBA) are in third cluster; Glutathione S-transferases, Phosducin and Endoplasmic reticulum protein ERP29 are also distributed in two clusters. In the last superfamily in the data set Membrane all-alpha, the family Photosynthetic reaction centre, L-, M- and H-chains is identified correctly and the remaining almost all proteins are separated correctly and distributed in various clusters. In this data set, there are 19 major families, in other words 19 families have more than 5 domains. Our method clearly identifies 15 families out of 19 families. 

Superfamily

Elongated mean spectral clustering is out performing compared to other two techniques while separating proteins into superfamilies. Globin-like and EF-hand proteins are separated clearly. In Cupredoxins hardly only one protein (1bag:1-347) is wrongly assigned. In (Trans)glycosidases superfamily two proteins (1gyc A:1-130 and 1gyc A:131-300) are wrongly placed from Thioredoxin-like superfamily and one protein (1bag:1-347) to Cupredoxins.
Thioredoxin-like superfamily, 6 proteins (1c3w-A, 1e8s-A, 1e12-A, 1h68-A, leys-L and 1ocr-B:1-90) are wrongly assigned from Membrane all-alpha and a protein 1jak A:151-506 from (Trans)glycosidases, and almost all Membrane all-alpha superfamily proteins are correctly separated and hardly 6 proteins assigned to Thioredoxin-like. In this dataset only 10 protein sequences are misplaced.

Figure-3 shows the results obtained from other two methods[2][3] respectively. The elongated mean spectral clustering method clearly performed well than the other two techniques, spectral[2] and MCL[2]. Spectral clustering [2] algorithm groups almost all the sequences close superfamily. Some sequences are wrongly assigned from Globin-like to EF-hand and (Trans) glycosidases to Thioredoxin-like. In spectral [2], selecting K the number of clusters is based on eigengap using predefined threshold, which is technically not perfect. This is inconsistence and sensitive to initialization. MCL algorithm has created many irrelevant clusters and many sequences are wrongly assigned between superfamilies.

In order to evaluate the performance of our implementation with spectral [2] and MCL[3] algorithms, we use F-measure. According to Yang and Liu[10], this measure was first introduced by C. J van Rijsbergen[11], which combines both precision (p) and recall(r) [9] with equal weights. $F(r,p)=2rp/(r+p)$, where p = a/(a+c) and r = a/(a+b), a is true positive, b is false negative and c is false positive

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Measures</th>
<th>Elongated mean Spectral</th>
<th>Spectral</th>
<th>MCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 507 Set</td>
<td>Precision</td>
<td>0.8222</td>
<td>0.7169</td>
<td>0.7301</td>
</tr>
<tr>
<td>Recall</td>
<td>0.6842</td>
<td>0.6783</td>
<td>0.2746</td>
<td></td>
</tr>
<tr>
<td>F-Measure</td>
<td>0.7469</td>
<td>0.6971</td>
<td>0.3991</td>
<td></td>
</tr>
<tr>
<td>No. of Clusters</td>
<td>8</td>
<td>8</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Data 457 Set</td>
<td>Precision</td>
<td>0.9201</td>
<td>0.8879</td>
<td>0.7898</td>
</tr>
<tr>
<td>Recall</td>
<td>0.6575</td>
<td>0.6235</td>
<td>0.3357</td>
<td></td>
</tr>
<tr>
<td>F-Measure</td>
<td>0.7669</td>
<td>0.7326</td>
<td>0.4711</td>
<td></td>
</tr>
<tr>
<td>No. of Clusters</td>
<td>10</td>
<td>10</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Data 342 Set</td>
<td>Precision</td>
<td>0.9479</td>
<td>0.9332</td>
<td>0.8004</td>
</tr>
<tr>
<td>Recall</td>
<td>0.8174</td>
<td>0.7533</td>
<td>0.4267</td>
<td></td>
</tr>
<tr>
<td>F-Measure</td>
<td>0.8581</td>
<td>0.8337</td>
<td>0.5567</td>
<td></td>
</tr>
<tr>
<td>No. of Clusters</td>
<td>7</td>
<td>7</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Comparison of results for clustering the sequences into superfamily

| Algorithms | Measures | Elongated mean Spectral | Spectral | MCL |
| Data 342 Set | Precision | 0.6764 | 0.5947 | 0.4703 |
| Recall | 0.8140 | 0.6944 | 0.5768 |
| F-Measure | 0.7388 | 0.6407 | 0.5181 |
| No. of Clusters | 28 | 28 | 38 |
| Data 457 Set | Precision | 0.7066 | 0.7211 | 0.5528 |
| Recall | 0.7922 | 0.6568 | 0.6349 |
| F-Measure | 0.7470 | 0.6875 | 0.5910 |
| No. of Clusters | 23 | 23 | 28 |
| Data 342 Set | Precision | 0.7789 | 0.7221 | 0.6471 |
| Recall | 0.8418 | 0.6902 | 0.7165 |
| F-Measure | 0.8091 | 0.7058 | 0.6800 |
| No. of Clusters | 14 | 14 | 25 |

Table 2. Comparison of results for clustering the sequences into family
The algorithm is evaluated by F-measures applying it on all the three datasets. Experiment results of our proposed algorithm and previous work[2][3] are listed in table-1 and table-2 for classifying proteins into superfamily and family respectively. The resulting clusters show 74%, 76% and 85% F-measures for classifying sequences into superfamily on 507dataset, 457dataset and 342dataset respectively and 72%, 72% and 80% F-measures for classifying sequences into family on 507dataset, 457dataset and 342dataset respectively. For calculating F-measures in superfamily classification, we apply fixed number of clusters (K given) in the proposed algorithm. While doing families classification, the proposed technique automatically determine the number of clusters which are very close to number of SCOP families in 457 and 342 datasets.

4 Discussion

We described a fast and efficient method for clustering protein sequences according their evolutionary relatedness.

The key abstractions made by the algorithm are the representation, symmetrification and Laplacian normalization of sequence information in a matrix, and spectral clustering with elongated mean. This has three main looping steps i) adding one more centre with zero values to centre matrix and adding one eigenvector to affinity eigenvector matrix ii) performing elongated mean clustering iii) if any data assigned to new centre then update centres and continues again step -1 otherwise terminates.

The algorithm is sufficiently general and simple and implemented MATLAB on Window-XP operating system, Pentium –IV, 3GHz and 1GB RAM. The algorithm allows the efficient clustering of any arbitrary set of protein sequences given in all pairwise similarities obtained from BLAST. The BLAST program has used for E-value similarity measures, which is downloaded from NCBI and implemented in XP-window operating system. Source codes are freely available on request to the author.

We have extensively validated the performance of the algorithm in terms of speed and accuracy also. We have assessed the performance of the algorithm in terms of the quality of protein family descriptions based on SCOP[1] database. These database contains extensive information relating to protein domains and structures, and proteins organized in hierarchical manner at four main levels: class, fold, super family, family. These proteins have been expertly classified into families based on their folding patterns and a variety of other information. Family information given is well understood and accurately presented.

Earlier we pointed out the following problems i) determining number of families using hard threshold and ii) effective clustering on a given meager data using local method. In global methods, assign a protein to a cluster taking into account the distance between every pair of proteins in the entire set, and for this reason they should be less sensitive to assigning sequences in wrong cluster. Spectral clustering methods are global methods and we have shown that the results obtained with our implementation are much better than the results obtained with other two global methods [2][3]. In spectral clustering, Euclidean distance measure has the undesired effect. We proved that Mahalanobis distance gives better results and very appropriate distance measure for spectral clustering(refer to the materials and methods section). Discovering number of clusters based on a drop in the magnitude of eigenvalues is lack of justification and fails in many cases. In our case, introduced a center with zeros to create new cluster if it exists, based on distance between the centers of clusters and a point belonging to different cluster is farther than from origin(refer to the materials and methods section). In MCL, inflation parameters modifies the random walks, boosting probabilities on strong intra-cluster walks and demoting weak inter-cluster walks to promote the emergence of clusters in the graph. This is only an approximation to the relaxation process.
implied by the data. The spectral clustering technique usually performs better on large size of data. The result obtained by elongated mean spectral for families shown in Table 1 and Table 2, the proposed algorithm performs improved on non-singleton families data set (Fig.2).

We believe that the algorithm represents a significant improvement over currently available clustering techniques.

5 Conclusions

In this paper we have shown that the elongated mean spectral algorithm clearly separates the set of related proteins using only sequence information, which are having very low sequences similarity. The global methods by means of considering entire set of sequences will provide a satisfactory answer than local methods. Experiment results obtained from different set of proteins indicate that the proposed approach leads to reasonable good clusters. We can find clusters that are close to the protein superfamilies and families in SCOP. The quality of the clustering is impressive and validated using SCOP database. In this, we conclude that the Mahalanobis distance measure is relevant to spectral clustering and the technique introducing a centre with zeros to determine number of clusters in a given data set is very understandable and justified.
References


Temporal Logic Treating Uncertainty and Interacting Agents with Time Indexes from $\mathcal{Z}$.

Sergey Babenyshev\textsuperscript{1}, Vladimir Rybakov\textsuperscript{1}, and Renate Schmidt\textsuperscript{2}

\textsuperscript{1} Department of Computing and Mathematics, Manchester Metropolitan University, John Dalton Building, Chester Street, Manchester M1 5GD, U.K.

V.Rybakov@mmu.ac.uk and S.Babenyshev@mmu.ac.uk

\textsuperscript{2} School of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, U.K.

schmidt@cs.man.ac.uk

\begin{abstract}
The paper investigates\textsuperscript{3} a new temporal logic $\mathcal{LTL}^Z_{UA}$, which combines operations of LTL, an operation for logical uncertainty, operations for agents’ knowledge, global and local knowledge, and an operation for knowledge via agents’ interaction. We study the issues of satisfiability and decidability of $\mathcal{LTL}^Z_{UA}$. The main result is an obtained algorithm for recognizing theorems of $\mathcal{LTL}^Z_{UA}$ (which ensures that $\mathcal{LTL}^Z_{UA}$ and the satisfiability problem for $\mathcal{LTL}^Z_{UA}$ are decidable).

\textbf{Key words:} temporal logic, multi-agent systems, uncertainty, interacting agents, decision algorithms, Kripke/Hintikka models
\end{abstract}

1 Introduction

The studies of temporal, modal and multi-agent logics were largely motivated by requirements of the research in the areas of Artificial Intelligence (AI) and Computer Science (CS). The former logics were designed to describe reasoning about knowledge, properties of computational processes and basic logical laws pertaining to the domain areas. To reason about multi-agent systems, several approaches were studied. One approach uses multi-modal logics (cf. Fagin et al. \cite{fagin1977 entailment, fagin1978 complexity}, Halpern and Shore \cite{halpern1990 knowledge}), where modal operations $K_i$ are used to represent cognitive abilities of individual agents. Choosing a logical language as a tool, there is a need to counterbalance expressiveness and simplicity. If a chosen language is too expressive, there is a danger that undecidability can occur (cf. Kacprzak \cite{kacprzak2010 undecidability}, using the reduction of decidability to the domino problem).

As to applications of temporal logics, they were first suggested to specify properties of programs in the late 1970’s (cf. Pnueli \cite{pnueli1977 temporal}). The most used temporal

\textsuperscript{3} This research is supported by Engineering and Physical Sciences Research Council (EPSRC), U.K., grant EP/F014406/1
framework is the linear-time propositional temporal logic LTL, which has been extensively studied from the point of view of various prospects of applications (cf. e.g. Manna and Pnueli [6], Clark E. et al. [7], Barringer, Fisher, Gabbay and Gough [8]). Model checking for LTL eventually formed a strong direction in the logical and computer science research, strongly influenced by Automata Theory (cf. Vardi [9, 10]). Various syntactic approaches have been developed and implemented as automated theorem provers [11–13].

Temporal logics themselves can be considered as a special cases of hybrid logics, e.g. as bimodal logics with some laws imposed on the interaction of modalities to imitate the flow of time. The mathematical theory devoted to the study of axiomatizations of temporal logics and development of their semantics, (based on Kripke/Hintikka-like models and temporal Boolean algebras) formed a highly technical branch in non-classical logics (cf. van Benthem [14, 15], Gabbay and Hodkinson [16], Hodkinson [17]). Axiomatizations of various (uni)-temporal linear logics are summarized in de Jonghe et al. [18].

In present time, combinations of temporal logics with multi-agent logics is an active research area. For example, Meyden and Shilov [19] studied the modal logic of knowledge and linear time, and showed (Theorem 1 [19], stated without proof) that this logic, with operations until and common knowledge, is undecidable (while some its fragments are decidable, it is proven with estimates on complexities for decision procedures). In the book Fagin et al. [1] (Section 4.3, Knowledge and Multi-Agent Systems: Incorporating Time), a combination of LTK with knowledge base operation $K_{KB}$ is suggested. The paper of Penczek and Lomusico [20] considers a computation-tree logic of knowledge (CTLK) aimed for verifying epistemic properties of multi-agent systems. The paper Hoek, Wooldridge [21] considers reductions of model checking for $CKL_n$ to linear temporal logic model checking. One more aspect that attracts a lot of attention is the logical treatment of uncertainty. Various formalizations for uncertainty in the logical framework are considered in recent publications (cf. [22, 23]).

In our paper we introduce and study a new temporal logic $\mathcal{LT}_{ZU}^{\mathbb{Z}_A}$, which combines operations of LTL, an operator $\text{Unc}_n$ for logical uncertainty, operations for agents’ knowledge, global and local knowledge, and the operation for knowledge via agents’ interaction $\text{KnI}$ (this operation may be expressed as a dual counterpart of the common knowledge operation introduced in Fagin et al. [1]). The meaning of the formula $\text{IntK}_\varphi$ may be interpreted as “there is a path consisting of agents’ accessibility relations, which leads from the current state to another where $\varphi$ holds (i.e., known)”. The main motivation for introducing this logic is its high expressive power, which combines the powers of the constituent logics and, therefore, may model all inherent properties. Besides that, it allows to describe the interactions of included logical operations.

The logic is defined as the set of all formulas valid in all Kripke/Hintikka-like models $\mathcal{Z}_C$, which are based on time indexed by the set $\mathbb{Z}$ of all integer numbers. The problems of satisfiability and decidability for $\mathcal{LT}_{ZU}^{\mathbb{Z}_A}$ are of primary interest for us. We construct an algorithm which recognizes theorems of $\mathcal{LT}_{ZU}^{\mathbb{Z}_A}$ (the existences of such algorithm implies that $\mathcal{LT}_{ZU}^{\mathbb{Z}_A}$ and the satisfiability problem
for $\mathcal{LT\mathcal{L}_T^A}$ are decidable). The algorithm works as follows: an arbitrary formula in the language of $\mathcal{LT\mathcal{L}_T^A}$ is, first, transformed into a rule in a special normal reduced form, which, then, is checked for validity on special models (of size efficiently bounded by the size of the rule) w.r.t. valuations of a special kind.

In our best knowledge, there are no yet published papers studying logics combining uncertainty, time operations and agents' knowledge (in particular, the case of agents interacting via common knowledge operation).

Notice the contrast between our results and the results of Meyden and Shilov [19], where modal logic of knowledge and linear time (with operations until and common knowledge) is claimed to be undecidable. Our choice for embedding agents' logic in $\mathcal{LTL}$ locally allowed us to get decidability. More specifically, note that $\mathcal{LT\mathcal{L}_T^A}$ differs from the logic in [19] in that respect, that in [19] agents have individual valuations (observations) and the operations $\sim_j$ ($R_j$ in our case) are defined in [19] by the way of agreement of the agents' observations on the computational runs. In our case, all $R_j$'s are arbitrary equivalence relations on sets $C(i), i \in \mathbb{Z}$, which is the most general possible case.

Our paper contains all definitions, notation and preliminary facts to follow the presentation, and is, in this sense, self-contained.

2 Definitions, Language and Semantics of $\mathcal{LT\mathcal{L}_T^A}$

We start with description of semantic structures motivating introduction of logical language for $\mathcal{LT\mathcal{L}_T^A}$. The basic semantic objects upon which we ground our logic are the following Kripke/Hintikka models. A frame

$$
\mathcal{Z}_C := \langle \bigcup_{i \in \mathbb{Z}} C(i), R, R_1, \ldots R_m, \text{Next, Prev} \rangle
$$

is a tuple, where $\mathbb{Z}$ is the set of all integer numbers, all $C(i)$ are disjoint nonempty sets ($C(i) \cap C(j) = \emptyset$ if $i \neq j$), $R$ is a binary linear relation for time, where

$$
\forall a, b \in \bigcup_{i \in \mathbb{Z}} C(i) (aRb) \iff [a \in C(i) \land b \in C(j) \land i < j] \lor \exists i \in \mathbb{Z} \left[ a, b \in C(i) \right].
$$

Relations $R_j$ represent agents' accessibility relations in time clusters $C(i)$: by definition, any $R_j$ is a reflexive, transitive and symmetric relation, and $\forall a, b \in \bigcup_{i \in \mathbb{Z}} C(i)$, $aR_j b \iff \exists i \in \mathbb{Z} \left[ a, b \in C(i) \right]$. Next is defined by a Next $b \iff [\exists i((a \in C(i)) \land (b \in C(i + 1)))$, and a Prev $b \iff b \text{ Next } a$.

Informal meaning of these frames is as follows. Any frame $\mathcal{Z}_C$ represents possible unbounded (in time) computation on a multi-processor system: $i \in \mathbb{Z}$ (any integer number $i$) simulated time tick $i$, any $C(i)$ consists of processors (computational units) evolved in computation in time $i$. Any $R_j$ is an accessibility relation for the agent $j$ between these computational units. $R$ is the accessibility relation in $\mathcal{Z}_C$ in time from now to future and past (as it is specified above it is a linear relation). Within any $C(i)$ (which means in the same time moment) all computational units ($u \in C(i)$) are mutually accessible by time (but not by agents accessibility relations).
To build our logical language, we start from the language of LTL, and, to handle uncertainty, we add the new (unary) logical operation $\text{Unc}$. Also, together with the standard language of LTL (with operations $\mathbf{N}$ (next) and $\mathbf{U}$ (until)), we will use new binary logical operations $\mathbf{U}_w$ (weak until), $\mathbf{U}_s$ (strong until), $\mathbf{S}$ (since), $\mathbf{S}_w$ (weak since), $\mathbf{S}_s$ (strong since) and $\mathbf{N}^{-1}$ (previous). To describe agents’ knowledge we, as always, will use agents knowledge operations $\mathbf{K}_j$, $1 \leq j \leq m$, and else operations $\mathbf{GK}_L$ and $\mathbf{GK}_G$ for local and global knowledge, and the unary operation $\mathbf{KnI}$ for to be known by agents’ interaction. The formation rules for formulas are as usual, and the intended meanings of the operations are as follow:

- $\text{Unc}\varphi$ means the statement $\varphi$ is uncertain in the current state of the current time cluster;
- $\mathbf{K}_j\varphi$ means the agent $j$ knows $\varphi$ in the current state of a time cluster;
- $\mathbf{GK}_L\varphi$ means that $\varphi$ is local general knowledge in the current state;
- $\mathbf{GK}_G\varphi$ means $\varphi$ is global general knowledge in the current state;
- $\mathbf{KnI}\varphi$: in the current state $\varphi$ may be known by interaction between agents;
- $\mathbf{N}\varphi$ has the meaning $\varphi$ holds in the next time cluster of states (state);
- $\mathbf{N}^{-1}\varphi$ means $\varphi$ holds in the previous time cluster of states (state);
- $\varphi\mathbf{U}\psi$ can be read: $\varphi$ holds until $\psi$ will hold;
- $\varphi\mathbf{S}\psi$ $\varphi$ says that since $\psi$ was true, $\varphi$ holds until now;
- $\varphi\mathbf{U}_w\psi$ has the meaning $\varphi$ weakly holds until $\psi$ will hold;
- $\varphi\mathbf{U}_s\psi$ has the meaning $\varphi$ strongly holds until $\psi$ will hold;
- $\varphi\mathbf{S}_w\psi$ $\varphi$ says that since $\psi$ was true, $\varphi$ weakly holds until now;
- $\varphi\mathbf{S}_s\psi$ $\varphi$ means that since $\psi$ was true, $\varphi$ strongly holds until now.

For any collection of propositional letters $\text{Prop}$ and any frame $\mathcal{Z}_C$, a valuation in $\mathcal{Z}_C$ is a mapping which assigns truth values to elements of $\text{Prop}$ in $\mathcal{Z}_C$. Thus, for any $p \in \text{Prop}$, $V(p) \subseteq \mathcal{Z}_C$. We will call $\langle \mathcal{Z}_C, V \rangle$ a model (a Kripke/Hintikka model). For any such model $\mathcal{M}$, the truth values are extended from propositions of $\text{Prop}$ to arbitrary formulas as follows (for $a \in \mathcal{Z}_C$, we denote $(\mathcal{M}, a) \models_V \varphi$ to say that the formula $\varphi$ is true at $a$ in $\mathcal{M}_C$ w.r.t. $V$). The rules are as follows:

\[
\forall p \in \text{Prop}, \quad (\mathcal{M}, a) \models_V p \iff a \in V(p); \quad (\mathcal{M}, a) \models_V \varphi \land \psi \iff (\mathcal{M}, a) \models_V \varphi \land (\mathcal{M}, a) \models_V \psi; \quad (\mathcal{M}, a) \models_V \neg \varphi \iff \neg[(\mathcal{M}, a) \models_V \varphi].
\]

Further, to compute uncertainty we use:

\[
(\mathcal{M}, a) \models_V \text{Unc}\varphi \iff \exists i[a \in C(i) \land \exists b \in C(i)(\mathcal{M}, b) \models_V \varphi \land \exists c \in C(i)(\mathcal{M}, c) \models_V \neg \varphi].
\]

That is, we say $\varphi$ is uncertain at a state of a time cluster $C(i)$ if there are two states in $C(i)$, i.e. in time $i$, where $\varphi$ is true at one of these states and is false at the another one. This looks as quite plausible way to express uncertainty of $\varphi$ (though, clearly, one of possible ones, it could be many ways to talk about uncertainty). Next, by definition, $(\mathcal{M}, a) \models_V \mathbf{K}_j\varphi \iff \forall b((a, R_j, b) \Rightarrow (\mathcal{M}, b) \models_V \varphi)$. As usual, $\mathbf{K}_j\varphi$ says that $\varphi$ holds in all states available for the agent $j$. Next, $(\mathcal{M}, a) \models_V \mathbf{GK}_L\varphi \iff \forall j \forall b((a, R_j, b) \Rightarrow (\mathcal{M}, b) \models_V \varphi)$. Thus, $\varphi$ is local general knowledge if it holds in all states which are accessible in the current time point.
for every agent. $GK_L$ is more commonly referred to as the $E$-operation, for “everyone knows”:

$$(M, a) \models V GK_G \varphi \iff \forall b((a R b) \Rightarrow (M, b) \models V \varphi).$$

Thus, $\varphi$ is global general knowledge if it holds in all states in all future time clusters.

$$(M, a) \models V KnI \varphi \iff \exists a_{i_1}, a_{i_2}, \ldots, a_{i_k} \in M [aR_{i_1}a_{i_1}R_{i_2}a_{i_2} \ldots R_{i_k}a_{i_k} & (M, a_{i_k}) \models V \varphi].$$

Thus, if $KnI \varphi$ holds, $\varphi$ is known by interaction between the agents, i.e. there is a path of transitions via the agents’ accessibility relations which leads to a state where $\varphi$ holds. Rules for LTL-operations and their relatives are

$$(M, a) \models V N \varphi \iff \forall b((a Next b) \Rightarrow (M, b) \models V \varphi);$$

$$(M, a) \models V N^{-1} \varphi \iff \forall b((a Prev b) \Rightarrow (M, b) \models V \varphi);$$

$$(M, a) \models V \varphi U \psi \iff \exists b((aRb) \& ((M, b) \models V \psi) \& \forall c((aRcRb) \& \neg (bRc) \Rightarrow (M, c) \models V \varphi));$$

$$(M, a) \models V \varphi U_w \psi \iff \exists b((aRb) \& ((M, b) \models V \psi) \& \forall c((aRcRb) \& \neg (bRc) \& c \in C(i)) \Rightarrow \exists d \in C(i)(M, d) \models V \varphi);$$

$$(M, a) \models V \varphi S \psi \iff \exists b((bRb) \& ((M, b) \models V \psi) \& \forall c((bRcRa) \& \neg (cRb) \Rightarrow (M, c) \models V \varphi));$$

$$(M, a) \models V S_w \psi \iff \exists b((bRb) \& ((M, b) \models V \psi) \& \forall c((bRcRa) \& \neg (cRb) \& c \in C(i)) \Rightarrow \exists d \in C(i)(M, d) \models V \varphi);$$

$$(M, a) \models V S_x \psi \iff \exists b((bRb) \& b \in C(i)) \& \forall c \in C(i)((M, c) \models V \psi) \& \forall c((bRcRa) \& \neg (cRb) \Rightarrow (M, c) \models V \varphi)].$$

Given a Kripke structure $M := (Z_C, V)$ and a formula $\varphi$, (i) $\varphi$ is satisfiable in $M$ (notation: $M \models_{Sat} \varphi$) if there is a state $b$ of $M$ ($b \in Z_C$) where $\varphi$ is true: $(M, b) \models V \varphi$. (ii) $\varphi$ is valid in $M$ (denotation: $M \models \varphi$) if, for any $b$ of $M$ ($b \in Z_C$), the formula $\varphi$ is true at $b$ $((M, b) \models V \varphi)$.

For a frame $Z_C$ and a formula $\varphi$, $\varphi$ is satisfiable in $Z_C$ (denotation $Z_C \models_{Sat} \varphi$) if there is a valuation $V$ in the frame $Z_C$ such that $(Z_C, V) \models_{Sat} \varphi$. $\varphi$ is valid in $Z_C$ (notation $Z_C \models \varphi$) if not $(Z_C \models_{Sat} \neg \varphi)$.

**Definition 1.** The logic $LTL^{LTL}_{IA}$ is the set of all formulas which are valid in all frames $Z_C$. 

42
We say that a formula $\varphi$ is *satisfiable* iff there is a valuation $V$ in a Kripke frame $Z_C$ which makes $\varphi$ satisfiable: $Z_C, V \models \text{Sat} \varphi$. Clearly, a formula $\varphi$ is satisfiable iff $\neg \varphi$ is not a theorem of $\mathcal{LTL}_{ZA}^w$: $\neg \varphi \notin \mathcal{LTL}_{ZA}^w$, and vice versa, $\varphi$ is a theorem of $\mathcal{LTL}_{ZA}^w$ ($\varphi \in \mathcal{LTL}_{ZA}^w$) if $\neg \varphi$ is not satisfiable. Using the operation $\mathcal{U}$ we, in the well known way, can define all standard modal and temporal operations, e.g. $\Diamond$ (possible), $\Box$ (necessary), $F$ (holds eventually), $G$ (holds henceforth), etc. Using these derived logical operations we can easily describe general knowledge operations accepted in the current framework:

$G_{K_{\varphi}} \equiv \Box (K\varphi)i \in [1, n] \in LTL_{ZA}^w$. Thus the initially specified language for $LTL_{ZA}^w$ is a bit superfluous and we can omit operations for local and global general knowledge because they are expressible via the others. The logic $LTL_{ZA}^w$ evidently is more expressive compared to standard $LTL$ and multi-agent logics in $S5$-like languages. For instance, the formula $\Box \neg K_1 \neg \varphi$ says that, for any future time cluster and for any state $a$ of this cluster the knowledge $\varphi$ is *discoverable* for agent 1, it has access to a state $b$ where $\varphi$ holds.

The new temporal operations $U_s$ and $U_w$ bring new unique features to the language. For instance the formula $\Box_w \varphi := \neg (\top \Box_s \neg \varphi)$ codes *weak necessity*, it says that in any future time cluster $C(i)$ there is a state where $\varphi$ is true. So, this formula way code the *non-vacuity* for a task $p$ computation: in any tick in future at least one possessor unit works with $p$.

The formula $(\neg \varphi U_w \Box \varphi) \wedge \Diamond \Box \varphi$ codes that, there is a minimal time point $i$ since which $\varphi$ holds in all states of all future time clusters, but before the time point $i$ the formula $\varphi$ is false in a state of any time cluster. Such properties are problematic to be expressed in standard modal or temporal operations. Notice, that the operations $U_s$ and $U_w$ may be encoded using standard operation $U$ and the belief operation (universal modality on whole $C(i)$ for each $i$). But, vice versa, the belief operation locally may be expressed by $U_s$. And the operation belief is too strong: it covers all agents’ knowledge operations and $\text{IntK}$. The belief operation would not really much matter in a real computational environment (such as, for example web sites and agents’ possible internet links), since there is no reason to assume that there is an omniscient supervisor — an agent who can use any web link and can open any web site. Therefore our approach is more general, and we prefer to use temporal operations $U_s$ and $U_w$.

### 3 Key Results, Decidability Algorithm for $LTL_{ZA}^w$

The principal problems we are dealing with are decidability and satisfiability problems for $LTL_{ZA}^w$. The basic technique we use is based on the reduction of formulas in the language of $LTL_{ZA}^w$ to special inference rules and the verification of the validity these rules in frames $Z_C$. This aims to implicitly model non-nested universal modality, which will be a useful instrument and to consider only rules (to which formulas are reduced) with non-nested non-Boolean logical operations (this simplifies the proofs and avoids the necessity to consider nested operations, and hence proofs by induction over formula complexity). This approach combines (i) techniques to handle inference rules from [24] – [25] (where
[25] solves decidability of LTL w.r.t. admissibility and again decidability of LTL itself) and (ii) techniques for a hybrid of LTL and usual knowledge logic with autonomous agents (V. Rybakov, workshop on Hybrid Logics, 2007, Dublin).

A (sequential) (inference) rule is a relation \( r := \varphi_1(x_1, \ldots, x_n) \land \ldots \land \varphi_l(x_1, \ldots, x_n) / \psi(x_1, \ldots, x_n) \), where \( \varphi_1(x_1, \ldots, x_n), \ldots, \varphi_l(x_1, \ldots, x_n) \) and \( \psi(x_1, \ldots, x_n) \) are formulas constructed out of letters \( x_1, \ldots, x_n \). The letters \( x_1, \ldots, x_n \) are the variables of \( r \), we use the notation \( x_i \in \text{Var}(r) \).

**Definition 2.** A rule \( r \) is said to be valid in a Kripke model \( \langle Z_C, V \rangle \) (notation \( Z_C \models_V r \)) if

\[
[\forall a \ ((Z_C, a) \models_V \bigwedge_{1 \leq i \leq l} \varphi_i)] \Rightarrow [\forall a \ ((Z_C, a) \models_V \psi)].
\]

Otherwise we say \( r \) is refuted in \( Z_C \), or refuted in \( Z_C \) by \( V \), and write \( Z_C \not\models_V r \). A rule \( r \) is valid in a frame \( Z_C \) (notation \( Z_C \not\models r \)) if, for any valuation \( V \), \( Z_C \not\models_V r \).

For any formula \( \varphi \) we can convert it into the rule \( x \rightarrow x/\varphi \) and employ a technique of reduced normal forms for inference rules as follows. Evidently,

**Lemma 1.** A formula \( \varphi \) is a theorem of \( \mathcal{LTL}^{\mathcal{LA}} \) iff the rule \( (x \rightarrow x/\varphi) \) is valid in any frame \( Z_C \).

A rule \( r \) is said to be in reduced normal form if \( r = \varepsilon / x_1 \) where

\[
\varepsilon := \bigvee_{1 \leq j \leq l} \bigwedge_{1 \leq i, k \leq n, i \neq k} \left[ x_i^{t(j,i,0)} \land (\text{Unc}_x x_i)^{t(j,i,1)} \land (N x_i)^{t(j,i,2)} \land \right. \\
(N^{-1} x_i)^{t(j,i,3)} \land (x_i U x_k)^{t(j,i,k,0)} \land (x_i U w x_k)^{t(j,i,k,1)} \land (x_i U s x_k)^{t(j,i,k,2)} \land \\
(x_i S x_k)^{t(j,i,k,3)} \land (x_i S w x_k)^{t(j,i,k,4)} \land (x_i S s x_k)^{t(j,i,k,5)} \land \\
\left. \bigwedge_{1 \leq q \leq m} (-K q x_i)^{t(j,i,q,6)} \land Kn I x_i^{t(j,i,4)} \right].
\]

all \( x_s \) are certain letters (variables), \( t(j, i, z) \), \( t(j, i, k, z) \in \{0, 1\} \) and, for any formula \( \alpha \) above, \( \alpha^0 := \alpha \), \( \alpha^1 := -\alpha \).

**Definition 3.** Given a rule \( r_{nf} \) in reduced normal form, \( r_{nf} \) is said to be a normal reduced form for a rule \( r \) iff, for any frame \( Z_C \),

\[
Z_C \not\models r \iff Z_C \not\models r_{nf}.
\]

By following verbatim Lemma 3.1.3 and Theorem 3.1.11 in [26] we obtain

**Theorem 1.** There exists an algorithm running in (single) exponential time, which, for any given rule \( r \), constructs its normal reduced form \( r_{nf} \).
Decidability of $\mathcal{LT}\mathcal{L}_{TA}^{ZU}$ will follow (by Lemma 1) if we find an algorithm recognizing rules in reduced normal form which are valid in all frames $Z_C$. The starting point to handle interactions of agents is

**Lemma 2.** A rule $r_{nf}$ in reduced normal form is refuted in a frame $Z_C$ if and only if $r_{nf}$ can be refuted in a frame with time clusters of size square exponential from $r_{nf}$.

For any frame $Z_C$ and some integer numbers $k_1, m_1, k_2, m_2$, where $m_2 > k_2 > k_1 + 3, k_1 > m_1$ we construct the frame $Z_C(k_1, m_1, k_2, m_2)$ from $Z_C$ as follows.

$$Z_C(k_1, m_1, k_2, m_2) := \langle \bigcup_{m_1 \leq i \leq m_2} C(i), R, R_1, \ldots, R_m, Next \rangle,$$

where $R$ is the accessibility relation from $Z_C$ extended by pairs $(x, y)$, where $x \in C(i), y \in C(j)$ and $i, j \in [k_1, m_1)$, or $i, j \in [k_2, m_2]$. Any relation $R_j$ is simply transferred from $Z_C$, and $Next$ and $Prev$ are taken from $Z_C$ and extended by

$$\forall a \in C(m_2) \forall b \in C(k_2)(a \ Next \ b = true);$$

$$\forall a \in C(m_2) \forall b \in C(k_2)(b \ Prev \ a = true);$$

$$\forall a \in C(m_1) \forall b \in C(k_1)(a \ Prev \ b = true);$$

$$\forall a \in C(m_1) \forall b \in C(k_1)(b \ Next \ a = true).$$

For any valuation $V$ of letters from a formula $\varphi$ in $Z_C(k_1, m_1, k_2, m_2)$ the truth value of $\varphi$ can be defined at elements of $Z_C(k_1, m_1, k_2, m_2)$ by the rules similar to the ones given for the frames $Z_C$ above (just in accordance with the meaning of logical operations). Due to limitations on the length of the paper we omit the detailed description of these rules. Using Lemma 2 as the basis, we can derive:

**Lemma 3.** A rule $r_{nf}$ in reduced normal form is refuted in a frame $Z_C$ iff $r_{nf}$ can be refuted in a frame $Z_C(k_1, m_1, k_2, m_2)$ by a valuation $V$ of special kind, where the size of the frame $Z_C(k_1, m_1, k_2, m_2)$ is triple exponential in $r_{nf}$.

We do not specify in the statement of the lemma above (due to space limitations) the properties required for the valuation $V$, but they are essential since a frame $Z_C(k_1, m_1, k_2, m_2)$ does not belong to the class of generating $\mathcal{LT}\mathcal{L}_{TA}^{ZU}$-frames. From Theorem 1, Lemma 1 and Lemma 3 we derive:

**Theorem 2.** The logic $\mathcal{LT}\mathcal{L}_{TA}^{ZU}$ is decidable. The algorithm for checking a formula to be a theorem of $\mathcal{LT}\mathcal{L}_{TA}^{ZU}$ consists in verification of validity rules in reduced normal form at frames $Z_C(k_1, m_1, k_2, m_2)$ of size triple-exponential in the size of reduced normal forms w.r.t. valuations of special kind.

It is possible also to apply the technique from this paper to weakened versions of the logic $\mathcal{LT}\mathcal{L}_{TA}^{ZU}$, say with omitted strong or weak versions of the operations $U$ or $S$, with omitted $N$ or $N^{-1}$ and to obtain similar results about decidability. Also some restrictions for agents accessibility relations $R_j$ may be considered by the introduction of a hierarchy between $R_j$. 

45
Conclusion, Future Work

The paper develops a technique for proving decidability in $\mathcal{LT}_{\mathcal{IA}}$ and in a number of similar logics. The suggested approach is proven to be flexible enough to work with a variety of logics from AI and CS. There are many prospective avenues of research on logic $\mathcal{LT}_{\mathcal{IA}}$ and its variants. For instance, the next most prospective candidate for the research is a variant of this language with a hierarchy of interacting agents.

Besides that, it is interesting whether it is possible to extend the methods of this paper to handle the case of hybrid non-linear temporal logics (e.g. branching time logics, $S4_T$, $K4_T$) with interacting agents. An open question is the question of axiomatizability. Another interesting problem concerns complexity issues and possible ways of refining the complexity bounds in the algorithm. Decidability questions of admissible inference rules in $\mathcal{LT}_{\mathcal{IA}}$ and in fusions of LTL based at $\mathcal{N}$ (without Since and Previous) and multi-modal logics with interacting agents are not investigated yet. The problem of describing bases for rules admissible in such logics is also open to date.

References

Evaluating Algorithms According to their Energy Consumption

Hannah Bayer and Markus Nebel

University of Kaiserslautern, Department of Computer Sciences, Gottlieb-Daimler-Strasse, 67633 Kaiserslautern, Germany, \{h_bayer,nebel\}@informatik.uni-kl.de

Abstract. This work deals with the analysis of algorithms according to their energy consumption. So far it was a common belief that faster algorithms consume less energy than slower ones. This work presents results indicating that this is not universally valid. For this purpose an energy model shall be introduced which is used to determine the energy consumption of algorithms with regard to the input size. Thereafter the algorithms will be compared to each other regarding both to their run time and energy consumption.

Key words: power consumption, analysis of algorithms, algorithm engineering

1 Motivation

Conventionally algorithm engineering is concerned with the run time and algorithms have therefore been evaluated with respect to their performance. Accordingly the run time of algorithms was the ultimate factor to be analyzed and optimized over the past years. However, over the years processors got faster and consumed more energy and the variety of fields where computers and embedded systems are used grew. So nowadays the power consumption of algorithms is an important factor to be taken into account.

There are many differently motivated reasons for trying to find ways to save energy consumed by computers which shall not be exhausted here. Existing solutions for minimizing energy consumption are multifaceted and span all components and architectural layers. There ACPI is to be mentioned, which allows the operating system to gain direct control over the power consumption as it can power down components after some time of inactivity. Another approach is to change the voltage according to the load of the system like in [1] and thus lower the power consumption. A lower consumption can naturally be realized through optimizations of hardware too but this seldom had been the main goal for the development of new processors. Finally the optimization of software shall be contemplated where previous works have already developed methods to lower the power consumption of algorithms. The main idea there is to optimize the process of compiling programs written in higher languages to assembler or machine code. As an example [2] should be mentioned were the consumption is
lowered by choosing a special alignment of the instructions. Some of those works as [3–11] do not directly derive methods to reduce the power consumption but present techniques and models to calculate the actual power consumed by an algorithm. Additionally there are papers concerning tools ([12] and [13]) simulating the energy consumption.

In the sequel it shall be described how the simulation tool XEEMU ([12]) can be used to compare the power consumption of algorithms and an energy model shall be introduced which allows for the theoretical analysis of power consumption and thus can be used for the quantification of power consumed during the execution of algorithms written in assembler code. Even if it is common belief that faster algorithms need less energy – and knowledge of their run time therefore is sufficient – the simulations and calculations based on the model to be introduced will be used to analyze the expected energy consumption of several algorithms from searching and sorting. As a surprise the results show that there are pairs of algorithms where the faster one consumes more energy.

The rest of this paper is organized as follows. First some informations about the general approach will be given, in section 3 the preconditions for the simulation will be elucidated and the results of the simulations performed will be presented. Section 4 contains the development of the theoretical model whereas section 5 engages in the discussion of the analytic results derived from this model.

2 Preliminaries

The algorithms to be analyzed are taken out of the range of searching and sorting. To refer to a consistent base the assembler code MIX [14] will be used to describe the different algorithms for the theoretical model as well as for the simulation. As MIX has some instructions that do not exist for modern processors it is required that those instructions are ”translated” by two ore more independent instructions on the processors to be simulated and those to be modelled. To retrieve concrete data two processors will be used, the proprietary DSP from Fujitsu and the ARM7TDMI commonly used in high end embedded systems whereas the simulation tool XEEMU can be used to simulate the power consumption of an ARM5-processor. Even if the power consumption of processors for embedded systems is already optimized somewhat comparing to other processors nevertheless they will be interesting to study, for the design of embedded systems did potentially take other possibilities of reduction of power consumption into account and maybe the prospects of those have already been exhausted.

3 Simulation

To gain a first impression of the power consumption of algorithms we will present the results from the simulation of certain searching algorithms originated from simulation with XEEMU ([12]) which were achieved by a student [15]. For every algorithm the average case run time and the average case power consumption stem from several passes of simulations. As the effects of cache misses change
with different cache hierarchies and different sizes of cache the effects do not merely depend on the algorithm itself. Hence, the cache size in XEEMU was chosen such that no cache misses took place after an initial process of once reading the input (the power consumption of this process was subtracted from the total power consumption). For the simulation of the average power consumption two versions were derived; one including and one excluding the power consumption of the cache. The latter will be interesting as the cache size is (as described above) chosen to be large and as a larger cache consumes more power even if no cache misses occur (for example for cache management) the total power consumption is thus fairly dominated by the power consumption of the cache.

In the sequel the average case simulation results for three different types of sequential sorting algorithms will be presented as well as results for "Binary Search", "Uniform Binary Search" and "Fibonacci Search". In Figure 1 it can be seen that for the run time the "Sequential Search" was significantly slower than the other two algorithms and it shows that the power consumption of all three algorithms appear to be the same if the consumption of the cache is taken into account. If this consumption is subtracted it shows that there are differences between all three algorithms and thus shows that even if the Quicker Sequential Search is not significantly faster than the "Quick Sequential Search" it consumes less power. Figure 2 pictures the run time and the power consump-

![Fig. 1. Average Case Sequential Searches (Simulations).](image-url)

tion of Binary Search, Uniform Binary Search and Fibonacci Search. Relative to the simulated run time the Fibonacci Search is clearly the worst of the three algorithms whereas it cannot be stated clearly which one of the other two algorithms is the faster one. Considering the total power consumption including the consumption of the cache Figure 2 shows that the Fibonacci Search is better than the Binary Search which should be chosen over the Uniform Binary Search. If one does not take the consumption of the cache into account the Fibonacci Search seems not to be the best but not the worst either whereas it shows that the Binary Search seems to consume more power than the Uniform Binary Search at most input sizes.

Thus it can be seen that the correlation between run time an power consumption is not necessarily existing. As the simulation of algorithms does take a lot of
time especially for the average case and provides only limited insight into reason for the behaviour observed further examinations were made with the help of a theoretical model that can be directly applied to algorithms written in assembler code which helped to get more insight into the reasons for the behaviour observed.

4 Energy Model and Analysis

We will continue by describing our theoretical model of energy consumption and its derivation. In [4] a energy model based on a risc-architecture is presented. According to this work, the main power consumption can be divided into two parts. One part is based on the instructions and the other one is based on the actual data used. Different instructions are performed by different components of the processor like the ALU or the multiplier or by a combination of components. As those components consume different amounts of power this results in different power consumptions for the variety of instructions. Furthermore, since not every component of the processor is used for every instruction, components can be switched off when they are idle. Switching components off and on consumes energy and is therefore required to be noticed in an energy model. The power consumption in the CPU resulting from the processed data depends on the number of ones in the binary representation in the actual input and on the hamming-distances between two following sets of data the so-called bit-toggling. Another model presented in [3] basically discriminates between two different kinds of power consumption, the consumption of the instruction itself and the consumption of the overhead resulting of the on and off switching of components. Furthermore this model does account for the consumption of pipeline stalls and cache misses.

Our model shall analyze the power consumption as independently as possible from the data processed, therefore all mere data-dependent power consumption shall be left unregarded. Furthermore Pipeline stalls will not be accounted for as it is hard to calculate how often they are going to occur on average for a given algorithm with a specific size of input and as the existing knowledge regarding the power consumption of pipeline stalls is limited. In contrast the
existing informations about power consumption referring to cache misses could be used easily, but average case analysis of cache misses is yet not very common and therefore the existing results concerning their occurrence are few and not very exact. Hence, only the consumption of the instructions and the overhead between two instructions will be taken into account. The conclusions drawn later all are subject to the presumption that pipeline stalls and cache misses do not alter the results if the effects of those would be considered in the model.

Further on it is presumed that the instructions of the processor to be modeled can be ordered into groups where all instruction contained in one group \( G \) do have a very similar power consumption and the overhead between an instruction and any other instructions is nearly the same for all instructions of \( G \).

As the algorithms to be analyzed are written in the assembly language MIX, every instruction of MIX will be filed into one such group. To apply our model to a specific processor \( P \) it is necessary to "map" the instruction set of \( P \) to the MIX instructions, as already mentioned in the preliminaries. Since the goal is an energy model which can be applied to a wide variety of processors the groups for the MIX instructions are defined in a way that not only the instruction set of one processor can be mapped to the MIX-instructions. This results in some groups having the same power consumption for a specific processor and the same overhead for the switching from and to other groups. For an example all MIX instructions that add or substract an immediate from the value in a register like DECA and INCA are grouped. Those instructions all use the ALU and for all of them the same sort of data and similar sources for the data are used, thus they should combined in one group.

We will use the following notation (as denoted in Figure 3): Every group is assigned with a factor standing for the value of the power consumption, for the overhead between two groups the factor will be noted as the names of the factors of the two groups separated by a colon. The value of the factors change according to the processor to be modeled.

Note that with the assignation of factors to the overheads the analysis of the power consumption differs from the classical runtime analysis: two algorithms which use the same set of instructions executed the same number of times could still have a different power consumption if the instructions of those algorithms are not executed in the same order.

Below the application of the energy model to *Sequential Search* will be discussed to exemplify how algorithms were analyzed; further on the consumption derived for the DSP (ARM7TDMI) will be notated as \( E_{DSP} \) (\( E_{ARM} \)). The algorithm in Figure 3 is directly adopted from [14] supplemented with the overhead between two instruction and the group-factor of instructions and overheads.

**Analysis for a Successful Search**

Initially the accurate values for the expected occurrences of the instructions and the overheads are calculated based on input size \( N \). In addition the runtime is evaluated. The following formulas are adopted from [14].

\[ S = 1 \ (S = 1 \text{ if the search was successful, } S = 0 \text{ otherwise}) \]

The number of comparisons \( C \) is based on the assumption that every element
line label | instruction | # executions | group-factor
--- | --- | --- | ---
1 START | LDA K | 1 | $\alpha_1$
2 | | 1 | $\alpha_1 : \alpha_6$
3 | ENT1 1-N | 1 | $\alpha_6$
4 | | 1 | $\alpha_6 : \alpha_8$
5 2H | CMPA KEY+N,1 | C | $\alpha_8$
6 | JE SUCCESS | C | $\alpha_8 : \alpha_{11}$
7 | INC1 1 | C-S | $\alpha_{11}$
8 | | C-S | $\alpha_{11} : \alpha_7$
9 | | C-S | $\alpha_7$
10 | | C-S | $\alpha_7 : \alpha_{11}$
11 | J1NP 2B | C-S | $\alpha_{11}$
12 FAILURE EQU * | 1-S | $\alpha_{11}$

Overhead resulting of Jumps: line 11 to line 5: C-1 times $\alpha_{11} : \alpha_8$

Fig. 3. MIX Code amended for the analysis.

of the searched set of elements is searched with the same probability.

$C = \frac{N+1}{2}$

**Average Power Consumption**

To analyze the power consumption $E$ of an algorithm one does only have to add the quantities of the factors which depend on the size of input. $E = \alpha_1 + \alpha_1 : \alpha_6 + \alpha_6 + \alpha_6 : \alpha_8 + C \cdot (\alpha_8 + \alpha_8 : \alpha_{11} + \alpha_{11}) + (C - S)(\alpha_{11} : \alpha_7 + \alpha_7 + \alpha_7 : \alpha_{11} + \alpha_{11}) + (C - 1)(\alpha_{11} : \alpha_8)$

With the values for the group-factors the following power consumption of the two processors can be proven:

$E_{DSP} = (92, 55 + 109, 75 \cdot N) mA$

$E_{ARM} = (113, 025 + 114, 165 \cdot N) mA$

**Runtime in u (Units of time) [14]:** $(2, 5 \cdot N + 3, 5) u$

Now it is easy to compare algorithms regarding their power consumption. To visualize this the expected run time and the expected power consumption of the algorithms to be compared have been put into different plots (see for example Figure 4), where the x-axis is the input size and the y-axis is the run time or respectively the power consumption. With respect to our model it can be seen easily which algorithm is faster and which algorithm has the lower power consumption.

Another interesting fact to consider is the leakage power which is the power that is always consumed whether the processor is idle or not. The crucial point here is whether the evaluation according to the power consumption has to be altered due to leakage power if an idle processor would be turned off immediately. If there are two algorithms (with the slower one consuming less energy) the first with the run time $t$ and the power consumption $e$ and the second respectively with $t'$ and $e'$ one only needs to solve the equation

$$e + t \cdot l = e' + t' \cdot l$$

for $l$ in order to get the leakage power it would take to delete the observed discrepancy between run time and energy consumption. If $l$ is larger than the
leakage power of the specific processor the discrepancy still holds if leakage power is taken into account.

5 Results Based on the Average Case Analysis

For most problems it holds that the fastest algorithm is the one with the lowest power consumption like for example for quicksort compared with merge sort or heapsort (which we have proven). But there are exceptions that cannot be disregarded. Those exceptions can be divided into different types. First are those algorithms that are faster but consume more energy than other algorithms for certain scopes of input sizes. Secondly those that are faster but consume more energy for all input sizes and last there are those where the statement of the second type holds true even when leakage power is taken into account. Good examples of the first kind are given by the algorithms for sequential search.

Figure 4 illustrates the average case run time and the power consumption of Sequential Search, Quick Sequential Search and Quicker Sequential Search as described in [14]. Whereas one cannot see a significant difference between the run time and the power consumption for great input sizes this is different for small input sizes as shall be seen in the sequel. Table 1 depicts the intersection points of the graphs regarding the size of input for the three types of sequential searches and thus shows the scope of input sizes, where the faster algorithm consumes more energy than the slower one. It is to be mentioned that the order of the graphs is the same for the analysed energy consumption of the ARM7TDMI and the simulated ARM-processor. As shown in Table 1 the intersection points of the graphs of the run time and the power consumption do not result from the same input sizes. For example whereas “Quicker Sequential Search” is to be preferred to “Quick Sequential Search” at almost any input size regarding the run time the same does not hold for the energy consumption. Therefore it would be better to use “Quick Sequential Search” for smaller input values and to switch to “Quicker Sequential Search” for larger input values to save energy. The algorithms “Uniform Binary Search” and “Fibonacci Search” shall exemplify the second kind. In Figure 5 one can see the results of our analysis.

![Fig. 4. Average Case Sequential Searches (model based theoretical analysis).](image-url)
Table 1. Intersection Points.

<table>
<thead>
<tr>
<th>Runtime</th>
<th>$E_{DSP}$</th>
<th>$E_{ARM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential Search &amp; Quick Sequ. S.</td>
<td>9</td>
<td>6.58</td>
</tr>
<tr>
<td>Sequential Search &amp; Quicker Sequ. S.</td>
<td>6.667</td>
<td>7.077</td>
</tr>
<tr>
<td>Quick Sequ. Search &amp; Quicker Sequ. S.</td>
<td>2</td>
<td>7.763</td>
</tr>
</tbody>
</table>

Fig. 5. Average Case Binary Searches (model based theoretical analysis).

(corresponding formulae will not be presented due to lack of space). We observe that with regard to the run time the "Fibonacci Search" should be preferred to the "Uniform Binary Search" this cannot be said regarding the power consumption. Looking at the consumption of the DSP the "Fibonacci Search" is slightly worse but with regard to the ARM7TDMI the discrepancy becomes significant. As the same discrepancy was already observed for the results of the simulation it is clearly worth analysing algorithms with this theoretical model that were not simulated. Furthermore it is to be mentioned that the graphs plotted above are only those for the average case but the basic message is the same for the worst case.

If leakage power is accounted for the result changes for the power consumption of the DSP, the "Fibonacci Search" does no longer consume more energy than the "Uniform binary search". Contrary to that the result does not change for the ARM7TDMI.

Similarly significant is the comparison of "Straight Insertion Sort", "Straight Selection Sort" and "List Insertion Sort". In Figure 6 again the average case is plotted. Similar to the illustration above for the binary searches it can be stated for the three sorting algorithms in question that the fastest algorithms is not the one with the lowest energy consumption. Regarding the run time "List Insertion Sort" is the worst algorithm but regarding the power consumption on the DSP it is the best and "Straight Selection Sort" is the worst regarding the power consumption of both DSP and ARM7TDMI but has definitely not the longest run time. Again the shown plots pictures the average case but in difference to the binary search algorithms the average case is not similar to the worst case, nonetheless for the worst case the order of the algorithms changes as well from
run time to energy consumption.
The results for the quadratic sorting algorithms do not change by taking the
leakage power into account and thus the algorithms are of the third kind men-
tioned above.

6 Resume

Our results from the analysis have proven that, assuming our model to be re-
alistic, the faster algorithm is not necessarily the one with the lower power
consumption. Even if one takes account of leakage power the gained results al-
most always keep the same. Considering using different algorithms to save energy
requires to analyze the relevant algorithms according to the specific processor.
For some problems an appropriate use of the right algorithm could save a great
amount of energy especially if a particular problem is solved very frequently.
As the obtained results are based on a theoretical model and the simulation of
few algorithms on one processor (for which model and simulation provide similar
results for processors from the same processor architecture) the next step will be
to affirm the results on basis of more simulation or to apply the existing energy
model to more processors especially to processors not designed for embedded
systems. Furthermore there are some options to improve the existing model and
data. To examine the behaviour of algorithms to the energy consumption on
specific processors more closely it would be necessary to explore the effect of
cache misses and pipeline stalls. Another interesting option would be to obtain
advanced knowledge about the reasons for the observed effects and thus be able
to develop strategies for designing energy efficient algorithms.

References

1. Hsu, C.H.: Compiler-directed dynamic voltage and frequency scaling for cpu power


A Characterisation of BD-N

Josef Berger
Mathematisches Institut
Universität München
Theresienstraße 39, D-80333 München, Germany
jberger@mathematik.uni-muenchen.de

Abstract. Ishihara’s principle BD-N says that every pseudobounded enumerable set of natural numbers is bounded. We provide an equivalent of this axiom which is based on relations rather than sets.

Keywords: constructive mathematics, boundedness principle

We work within the framework of Bishop’s constructive mathematics as introduced in [3]. Greek letters stand for functions from \( \mathbb{N} \) into \( \mathbb{N} \). The principle BD-N was introduced in [1] as an axiom which is constructively equivalent to the statement: ‘every sequentially continuous map from a separable metric space into a metric space is pointwise continuous.’

BD-N Suppose that \( \varphi \) has the property

\[
\forall \alpha \exists m \forall n (n \geq m \rightarrow \varphi(\alpha n) \leq n).
\]  

(1)

Then

\[
\exists n \forall m (\varphi m \leq n).
\]

This axiom was widely used and discussed in constructive mathematics, e.g. in [2, 4]. Here we give a relation-based equivalent of BD-N.

RBD-N Let \( A(n, m) \) be a decidable formula. Suppose that

\[
\forall n, m (A(n, m) \rightarrow A(n + 1, m))
\]  

(2)

and

\[
\forall \alpha \exists m \forall n (n \geq m \rightarrow A(n, \alpha n))
\]  

(3)

Then

\[
\exists n \forall m A(n, m).
\]

Proposition 1. The axioms BD-N and RBD-N are equivalent.
Proof. Assume RBD-N. Fix a function $\varphi$ and suppose that

$$\forall \alpha \exists m \forall n \geq m (\varphi(\alpha n) \leq n).$$

Define

$$A(n, m) \overset{\text{def}}{=} \varphi m \leq n.$$

Then (2) and (3) are fulfilled and RBD-N implies that $\varphi$ is bounded.

Now assume BD-N. Suppose that the decidable formula $A(n, m)$ fulfills (2) and (3). By applying (3) to constant functions, we obtain

$$\forall m \exists n A(n, m).$$

Define a function $\varphi$ by

$$\varphi m = \min \{n \mid A(n, m)\}.$$

Note that

$$\forall m, n (A(n, m) \leftrightarrow \varphi m \leq n).$$

By (3) and (4), we obtain

$$\forall \alpha \exists m \forall n (n \geq m \rightarrow \varphi(\alpha n) \leq n).$$

Thus BD-N implies that

$$\exists n \forall m (\varphi m \leq n).$$

By (4), this implies

$$\exists n \forall m A(n, m).$$

References

Learning Non-Confluent NLC Graph Grammar Rules

Hendrik Blockeel$^{1,2}$ and Robert Brijder$^1$

$^1$ Leiden Institute of Advanced Computer Science, Universiteit Leiden, The Netherlands, rbrijder@liacs.nl
$^2$ Department of Computer Science, Katholieke Universiteit Leuven, Belgium, hendrik.blockeel@cs.kuleuven.be

Abstract. Grammar inference deals with determining (preferable simple) models/grammars consistent with a set of observations. There is a large body of research on grammar inference within the theory of formal languages. However, there is surprisingly little known on grammar inference for graph grammars. In this paper we take a further step in this direction and work within the framework of node label controlled (NLC) graph grammars. Specifically, we characterize, given a set of disjoint and isomorphic subgraphs of a graph $G$, whether or not there is a NLC graph grammar rule which can generate these subgraphs to obtain $G$. This generalizes previous results by assuming that the set of isomorphic subgraphs is disjoint instead of non-touching. This leads naturally to consider the more involved “non-confluent” graph grammar rules.

1 Introduction

Grammar inference, also called grammar induction, is a general line of research where one is concerned with determining a “simple” grammar that is consistent with a given set of possible and impossible outcomes. Hence, one “goes back” in the derivation: instead of determining the generative power of a grammar, one determines the grammar given the generated output. This topic is well-studied for formal languages, especially with respect to context-free languages, see e.g. [7,5], however, relatively little is known for graph grammars.

The topic of inference of graph grammars is considered in [6] and uses their so-called Subdue scheme developed in [3]. In [2] a rigorous approach of grammar inference within the framework of node label controlled (NLC) graph grammars [4], a natural and well-studied class of graph grammars, is initiated. There it is characterized, given a set $S$ of non-touching isomorphic graphs of a graph $G$, whether or not there is a graph grammar consisting of one rule able to generate the graphs of $S$ to obtain $G$. We continue this research and generalize this result for the case where these graphs are disjoint instead of non-touching. Such a generalization requires one to deal with a number of issues. Most notably, one has to deal with non-confluence issues: the generated graph depends on the order in which touching subgraphs are generated.

Due to space constraints, proofs of the results are omitted, but can be found in an extended version [1] of this paper.
2 Notation and Terminology

We consider (simple) graphs $G = (V, E)$, where $V$ is a finite set of nodes and $E \subseteq \{\{x, y\} \mid x, y \in V, x \neq y\}$ is the set of edges – hence no loops or parallel edges are allowed. We denote $V(G) = V$ and $E(G) = E$. For $S \subseteq V$, the induced subgraph of $G$ is $(S, E')$ where $E' \subseteq E$ and for each $e \in E$ we have $e \in E'$ iff $e \subseteq S$. We consider only induced subgraphs, and therefore we often write just “subgraph” instead of induced subgraph. The neighborhood of $S \subseteq V$ in $G$, denoted by $N_G(S)$, is $\{v \in V \setminus S \mid \{s, v\} \in E \text{ for some } s \in S\}$. If $S = \{x\}$ is a singleton, then we also write $N_G(x) = N_G(S)$. A labelled graph is a triple $G = (V, E, l)$ where $(V, E)$ is a graph and $l : V \rightarrow L$ is a node labelling function, where $L$ is a finite set of labels. As usual, graphs are considered isomorphic if they are identical modulo the identity of the nodes. It is important to realize that for labelled graphs, nodes identified by an isomorphism have identical labels. In graphical depictions of labelled graphs we always represent the nodes by their labels.

Subgraphs $S_1$ and $S_2$ are called disjoint if $V(S_1)$ and $V(S_2)$ are disjoint. They are called touching if they are disjoint and there is an edge $e \in E(G)$ with one vertex in $S_1$ and the other in $S_2$. Moreover, they are called non-touching if they are disjoint and not touching.

Define, as usual, for $W_1, W_2 \subseteq V$, $W_1 \times W_2 = \{(x_1, x_2) \mid x_1 \in W_1, x_2 \in W_2\}$. Define $u((x_1, x_2))$ to be $\{x_1, x_2\}$, and define $\pi_i((x_1, x_2)) = x_i$ for $i \in \{1, 2\}$. Often, for a function $f : X \rightarrow Y$ we write $f(D) = \{f(x) \mid x \in D\}$ for $D \subseteq X$. Also, if $x_1, x_2 \in X$, we write $f(((x_1, x_2)) = (f(x_1), f(x_2))$.

3 NLC Graph Grammars

Typically, a graph grammar transforms a graph $G$ by replacing an (induced) subgraph $H$ by another graph $H'$ where $H'$ is embedded in the remaining part $G \setminus H$ of the original graph in a way prescribed by a so-called graph grammar embedding relation. The node label controlled (NLC) graph grammars are the simplest class of these grammars, where $H$ is a single node. Note that for the grammars the exact identities of the nodes are not important as multiple copies of $H'$ may be inserted. Hence, we consider labelled graphs where the embedding relation is defined w.r.t. node labels instead of nodes. In this section we recall informally the notions and definitions concerning NLC grammars used in this paper, and refer to [4] for a gentle and more detailed introduction to these grammars.

A NLC graph grammar is a system $Q$ consisting of a set of node labels $L$, an embedding relation $E \subseteq L^2$, and a set of productions $P$ where a production is of the form $N \rightarrow S$ where $N \in L$ and $S$ is a (labelled) graph. In this paper we will focus on the case $|P| = 1$. Hence $Q$ can be denoted as a rule $r = N \rightarrow S/E$ (if $L$ is understood from the context of considerations). Given a graph $G$, $r$ can be applied to any node $v$ labelled by $N$. The result of applying $r$ to $v$ in $G$ is that $v$ is removed from $G$ along with the edges adjacent to $v$, and (a copy of) $S$ is
added to $G$, and an edge $e = \{x, y\}$ is added to $G$ iff $x \in V(S)$, $y \in N_G(v)$ and $(l(x), l(y)) \in E$ (recall that $l$ is the labelling function). To avoid confusion with embedding relations, the set of edges of a graph $G$ are written in the remainder as $E(G)$ and not as $E$.

![Graph Diagram](image)

**Fig. 1:** The derivation of a graph $G$ (left-hand side) to $G'$ (right-hand side).

*Example 1.* Let $G$ be the graph on the left-hand side of Figure 1. Consider the grammar rule $r = N \rightarrow S/E'$, where $S$ is the graph

```
  a  a
  |   |
  b —— c
```

and $E' = \{(a, b), (b, a), (c, c), (a, N), (c, N)\}$. (Note that formally we have only defined $S$ up to isomorphism, however as we have seen this is not an objection.) Then Figure 1 depicts one possible derivation from $G$ to a graph $G''$ (on the right-hand side of the figure) for which no rule is applicable anymore. Note that there is one other possible derivation to a “terminal” graph $G'''$ (i.e., a graph without nodes labelled by $N$): to obtain $G'''$ we choose first the right-hand node labelled by $N$ (the one not connected to the node labelled by $b$) in $G$ in the derivation. Note that $G''$ and $G'''$ are different graphs. We assume that the set of labels $L$ is $\{a, b, c, N\}$. This example will be our running example of this paper.

In [2] the inference of NLC grammars with exactly one rule $r = N \rightarrow S/E$ are studied where moreover $S$ does not contain a node labelled by $N$ and $E$ does not contain a tuple containing $N$. This is sufficient for the case where the subgraphs isomorphic to $S$ are non-touching. To consider the case where the subgraphs are disjoint, we allow $E$ to contain tuples containing $N$. However, we do require that $S$ remains without nodes labelled by $N$. Therefore there is no “real” recursion: no nodes labelled by $N$ can be introduced in any derivation.

## 4 Known results: Non-touching graphs

In this section we recall some notions and a result from [2] which we will need in subsequent sections. First we define in this context the notion of compatibility.
Definition 2. Let $G$ be a graph and $S$ be an induced subgraph of $G$. We say that $E \subseteq L \times L$ is compatible for $S$ (in $G$) if there is a graph $F$ such that an application of NLC grammar rule $N \rightarrow S' / E$ to $F$ “creates” $S$ and obtains graph $G$. Note: $S'$ is (isomorphic to) $S$.

Example 3. Reconsider our running example. Hence we again let $G'$ be the graph at the right-hand side of Figure 1. Moreover we let $S_1$ and $S_2$ be the subgraphs of $G'$ of isomorphic to $S$ where $S_1$ is the one connected to the node labelled by $b$ and $S_2$ is the other one. Note that $S_1$ and $S_2$ are disjoint and touching in $G'$. We have that, e.g., $E_1 = \{(b,a),(c,c)\}$, $E_2 = \{(b,a),(c,c),(a,b)\}$ or $E_3 = \{(b,a),(c,c),(c,b),(b,b)\}$ is compatible for $S_2$ in $G'$. The middle graph of the figure is a graph $F$ such that an application of the NLC grammar rule $N \rightarrow S/E$ to $F$ “creates” $S_2$ and obtains graph $G'$.

To characterize the notion of compatibility, the notions of inset and outset for arbitrary $Q \subseteq V^2$ (where $V$ is the set of nodes of $G$) are crucial.

Definition 4. Let $Q \subseteq V^2$. We define the inset of $Q$, denoted by $I_Q$, as the set 
\[ \{(l(x),l(y)) \mid (x,y) \in E(G), (x,y) \in Q\} \]
and outset of $Q$, denoted by $O_Q$, as the set 
\[ \{(l(x),l(y)) \mid (x,y) \not\in E(G), (x,y) \in Q\} \].

Let $S$ be an induced subgraph of $G$. Then the inset (outset, resp.) of $S$, denoted by $I_S$ ($O_S$, resp.), is defined to be the inset (outset, resp.) of $Q = V(S) \times N_G(V(S))$.

The following lemma, given and proven in [2], characterizes compatibility for a single graph $S$ in terms of the inset and outset of $S$: the inset are tuples that should be in $E$, while the outset are tuples that should not be in $E$.

Lemma 5. Let $S$ be an induced subgraph of $G$, and let $E \subseteq L \times L$. Then $E$ is compatible for $S$ iff $I_S \subseteq E \subseteq L^2 \setminus O_S$ (i.e., $E$ separates $I_S$ from $O_S$).

Hence, there is a compatible $E$ for $S$ in $G$ iff $I_S \cap O_S = \emptyset$.

Example 6. Reconsider again our running example. Then $I_{S_2} = \{(b,a),(c,c)\}$ and $O_{S_2} = \{(a,a),(a,c),(c,a),(b,c)\}$ (w.r.t. $G'$). Since $I_{S_2} \cap O_{S_2} = \emptyset$, there is a compatible $E$ for $S_2$ in $G'$. We have that $I_{S_2} \subseteq E \subseteq L^2 \setminus O_{S_2}$ holds for, e.g., $E_1$, $E_2$ and $E_3$ in Example 3.

We consider now sequences of subgraphs to be generated by a single graph rule. Note that these graphs must necessarily be mutually isomorphic.

Definition 7. Let $G$ be a graph and $S_1,S_2,\ldots,S_n$ be induced subgraphs of $G$ isomorphic to $S$. We say that $E \subseteq L \times L$ is compatible for $(S_1,S_2,\ldots,S_n)$ (in $G$) if there are graphs $G_0,\ldots,G_n$ such that $G_n = G$ and for each $i \in \{1,\ldots,n\}$, $G_i$ is obtained from $G_{i-1}$ by applying a NLC grammar rule $N \rightarrow S/E$ that “creates” $S_i$.

Note that, in general, the order of the elements $(S_1,S_2,\ldots,S_n)$ is important. E.g. a given $E$ may be compatible for $(S_1,S_2)$ while it is incompatible for $(S_2,S_1)$ (we will see such an example in the next section).
However, for a set of mutually non-touching and isomorphic subgraphs $S_i$ for $i \in \{1, \ldots, n\}$ of $G$, the order of the elements is not important. Thus, $E \subseteq L \times L$ being compatible for $C = (S_1, S_2, \ldots, S_n)$ implies that $E$ is compatible for any permutation of $C$. In fact we have that, $E \subseteq L \times L$ is compatible for $S_1$, for $S_2$, ..., and for $S_n$ if and only if it is compatible for $C$ (or any permutation of $C$). Therefore, in this case, Lemma 6 is trivially generalized: $E \subseteq L \times L$ is compatible for $(S_1, S_2, \ldots, S_n)$ iff $\cup_i I_{S_i} \subseteq E \subseteq L^2 \setminus (\cup_i O_{S_i})$ (as noted in [2]).

5 Two touching graphs

In this section we consider the case where a single NLC grammar rule $N \rightarrow S/E$ generates disjoint subgraphs which can (possibly) touch each other. Hence, this generalizes Lemma 6 by replacing the non-touching condition into disjointness.

To this aim we allow the non-terminal $N$ to be present in tuples of the embedding relation $E$ of the NLC grammar rule $N \rightarrow S/E$. This introduces the issue of non-confluency: the order in which non-terminals are replaced by subgraphs influences the obtained graph. Example 1 illustrates this as the different graphs $G'$ and $G''$ can both be obtained from the original graph $G$.

As we will see the inset and outset between the nodes of two touching graphs turns out to be crucial.

**Definition 8.** Let $S_1$ and $S_2$ be touching graphs in $G$. For $Q_1 = V(S_2) \times (V(S_1) \cap N_G(S_2))$, we denote $I_{Q_1}$ and $O_{Q_1}$ by $I_{(S_1, S_2)}$ and $O_{(S_1, S_2)}$, respectively. Moreover, for $Q_2 = V(S_2) \times V(S_1)$, we denote $I_{Q_2}$ and $O_{Q_2}$ by $I_{((S_1, S_2))}$ and $O_{((S_1, S_2))}$, respectively.

Notice that these in- and outsets, e.g. $I_{(S_1, S_2)}$, are concerned with the tuples going from $S_2$ to $S_1$. This is because these tuples are important in the second step in the derivation to $G$ which first creates $S_1$ (first step) followed by the creation of $S_2$ (second step).

We now state some basic properties of the insets and outsets of Definition 9. Note first that $I_{(S_1, S_2)} = I_{((S_1, S_2))}$. In fact, it is equal to the inset of

\[(V(S_2) \cap N_G(S_1)) \times (V(S_1) \cap N_G(S_2)).\]

Also note that, for node labels $x$ and $y$, we have $(x, y) \in I_{((S_1, S_2))}$ iff $(y, x) \in I_{((S_2, S_1))}$. This holds similarly for $O_{((S_1, S_2))}$; however, this does not hold in general for $O_{(S_1, S_2)}$. Moreover note that $O_{(S_1, S_2)} \subseteq O_{((S_1, S_2))}$ (as they are the outsets of $Q_1$ and $Q_2$ resp., where $Q_1 \subseteq Q_2$), and

\[O_{((S_1, S_2))} \setminus O_{(S_1, S_2)} = I(V(S_2) \times (V(S_1) \setminus N_G(S_2))).\]

Finally note that $\pi_2(I_{(S_1, S_2)}) = l(V(S_1) \cap N_G(S_2))$. We will use these basic properties in the remainder of this paper.

**Example 9.** In our running example, we have $I_{(S_1, S_2)} = \{(b, a), (c, c)\}$, $O_{(S_1, S_2)} = \{(a, a), (a, c), (b, c), (c, a)\}$, and $O_{((S_1, S_2))} = L^2 \setminus I_{(S_1, S_2)}$ with $L' = \{a, b, c\}$. Moreover, we have $I_{(S_2, S_1)} = \{(a, b), (c, c)\}$, $O_{(S_2, S_1)} = \{(a, c), (b, b), (b, c), (c, b)\}$, and $O_{((S_2, S_1))} = L^2 \setminus I_{(S_2, S_1)}$. 64
We now adapt the definition of inset and outset for a graph $S$, by incorporating the issues related to touching graphs.

**Definition 10.** Let $S_1, \ldots, S_n$ be distinct subgraphs of $G$, and let $Q = (\bigcup_{i \in \{1, \ldots, n\}} V(S_i)) \times \left( N_G(V(S_i)) \setminus \bigcup_{j \in \{1, \ldots, n\}} V(S_j) \right)$. We denote $I_Q$ and $O_Q$ by $I_{[S_1, \ldots, S_n]}$ and $O_{[S_1, \ldots, S_n]}$, respectively.

Note that $I_{[S_1, S_2]} = I_{[S_2, S_1]}$ and if $S_1$ and $S_2$ are non-touching, we have $I_{[S_1, S_2]} = I_{S_1} \cup I_{S_2}$.

**Example 11.** In our running example, we have $I_{[S_1, S_2]} = \{(a, b)\}$, and $O_{[S_1, S_2]} = \{(b, b), (c, b)\}$.

Definitions 9 and 11 are to separate three types of insets and outsets. Roughly speaking, the two types of insets and outsets of Definition 9 deal with the tuples between $S_1$ and $S_2$, while the type of inset and outset of Definition 11 deals with the tuples from $S_1$ to the “outside world” (the nodes in the neighborhood of $S_1$ which do not belong to $S_2$) plus the tuples from $S_2$ to the “outside world” (the nodes in the neighborhood of $S_2$ which do not belong to $S_1$).

We now characterize the embedding relations $E$ such that $E$ is compatible for $(S_1, S_2)$ where $S_1$ and $S_2$ are touching subgraphs of $G$.

**Lemma 12.** Let $S_1$ and $S_2$ be touching subgraphs of $G$. Then $E \subseteq L \times L$ is compatible for $(S_1, S_2)$ iff the following conditions hold:

1. $I_{(S_1, S_2)} \subseteq E$,
2. $\{(x, N) \mid x \in \pi_2(I_{(S_1, S_2)})\} \subseteq E$,
3. If $e \in O((S_1, S_2))$, then either $(\pi_2(e), N) \notin E$ or $e \notin E$ (or both), and
4. $I_{[S_1, S_2]} \subseteq E \subseteq L^2 \setminus (O_{[S_1, S_2]}$).

Moreover, if this is the case, then we have $E \cap O_{(S_1, S_2)} = \emptyset$.

Intuitively, condition (4) of Lemma 13 deals with the edges of $S_1$ and $S_2$ to the “outside world”, while conditions (1) to (3) deal with the edges between $S_1$ and $S_2$. Conditions (1) and (2) state the tuples that must necessarily be in $E$, while condition (3) states requirements on which tuples must not (together) be in $E$.

Since $E \cap O_{(S_1, S_2)} = \emptyset$ by Lemma 13, we may modify conditions (1) and (3) of the previous lemma as follows:

1’. $I_{(S_1, S_2)} \subseteq E \subseteq L^2 \setminus O_{(S_1, S_2)}$,
3’. If $e \in O((S_1, S_2)) \setminus O_{(S_1, S_2)} = l(V(S_2) \times (V(S_1) \setminus N_G(S_2)))$, then either $(\pi_2(e), N) \notin E$ or $e \notin E$ (or both).

However, in this way the condition $E \cap O_{(S_1, S_2)} = \emptyset$ is explicitly assumed and not part of the result as stated in the lemma.

**Remark 13.** Note that by condition (4) of the lemma, we may go even further and instead state “$e \in l(V(S_2) \times (V(S_1) \setminus N_G(S_2))) \setminus O_{[S_1, S_2]}$” in condition (3).  

\[ \square \]
Example 14. We continue our running example. As we have seen, an $E \subseteq L \times L$ compatible for $(S_1, S_2)$ in $G'$ allows, given the graph $G$ on the left-hand side of Figure 1, for the generation of the middle graph (in the figure) and subsequently the generation of $G'$. We will now determine, using Lemma 13 and the modified conditions below the lemma, the constraints on $E$ for it to be compatible for $(S_1, S_2)$.

Recall that $I_{(S_1, S_2)} = \{(b, a), (c, c)\}$, $O_{(S_1, S_2)} = \{(a, a), (a, c), (b, c), (c, a)\}$, $I_{[S_1, S_2]} = \{(a, b)\}$, and $O_{[S_1, S_2]} = \{(b, b), (c, b)\}$. Moreover, $\{(x, N) \mid x \in \pi_2(I_{(S_1, S_2)}) = l(V(S_1) \cap N_G(S_2)) \} = \{(a, N), (c, N)\}$. Hence, by conditions (1'), (2), and (4) of Lemma 13 we have $\{(a, b), (b, a), (c, c), (a, N), (c, N)\} \subseteq E$ and

$$E \cap \{(a, a), (a, c), (b, b), (b, c), (c, a), (c, b)\} = \emptyset.$$ 

Now, $O_{(S_1, S_2)} \setminus O_{([S_1, S_2])} = l(V(S_2) \times (V(S_1) \setminus N_G(S_2))) = \{(a, b, b, b, (c, b)\}$.

Hence by condition (3') either $(b, N) \notin E$ or $(a, b) \notin E$. The latter is a contradiction, hence $(b, N) \notin E$. Consequently,

$$E = \{(a, b), (b, a), (c, c), (a, N), (c, N)\}$$

is compatible for $(S_1, S_2)$, in fact, in this case, it is the unique $E$ such that it is compatible for $(S_1, S_2)$ in $G'$. Note that adding $(b, N)$ to $E$ would indeed make it incompatible – the generated graph would then have edges from the node labelled $b$ in $S_1$ to the two nodes labelled $a$ in $S_2$. Also note that this $E$ is not compatible for $(S_2, S_1)$ in $G'$.

6 Set of touching graphs

Let $S$ be a set of mutually isomorphic and disjoint subgraphs of $G$. In this section we turn to the question of whether or not there is an $E \subseteq L \times L$ and a linear ordering $C = (S_1, S_2, \ldots, S_n)$ of $S$ such that $E$ is a compatible embedding relation for $C$.

The following result is easily obtained from Lemma 5.

Lemma 15. Let $G$ be a graph, $E \subseteq L \times L$, and $C = (S_1, \ldots, S_n)$ be a linear ordering of $S$. Then $E$ is compatible for $C$ iff (1) $I_{[S_1, \ldots, S_n]} \subseteq E \subseteq L^2 \setminus (O_{[S_1, \ldots, S_n]})$ and (2) for each two touching $S_i$ and $S_j$ with $i < j$, we have that the first three conditions of Lemma 13 hold w.r.t. $E$ and $(S_i, S_j)$.

As we have seen in Section 4, the compatibility of $E$ for the case where the elements of $S$ are mutually non-touching is much less involved: it does not depend on the ordering $C$ of $S$ – it only depends on $S$. For touching graphs, the situation is different as the conditions in Lemma 13 are not symmetric: e.g. $I_{(S_i, S_j)}$ and $I_{(S_j, S_i)}$ generally differ. Hence, we must choose a linear ordering in a “compatible” way. First, we focus on the question whether or not there exists an $E$ compatible for a given linear ordering $C$ of $S$.

To this aim consider the following graph that represents, w.r.t. an ordering $C$ of $S$, whether or not subgraphs $S_i$ and $S_j$ in $S$ touch.
Definition 16. Let $G$ be a graph and let $C = (S_1, S_2, \ldots, S_n)$ be an ordering of $S$. The touching graph of $G$ w.r.t. $C$, is the directed graph $(S, \{(S_i, S_j) \mid S_i \text{ and } S_j \text{ touch, and } i < j\})$.

For $e = (S_i, S_j) \in E(D)$ we write $O_e = O_{(S_i, S_j)}$ and $O_{(e)} = O_{((S_i, S_j))}$ (and similarly for the insets $I_e$ and $I_{(e)}$).

We now obtain a main result. It turns out that, using Lemma 13, the existence of an embedding relation $E$ for given ordering $C$ is elegantly characterized.

Theorem 17. Let $G$ be a graph, let $C = (S_1, S_2, \ldots, S_n)$ be an ordering of $S$, and let $D$ be the touching graph of $G$ w.r.t. $C$. There is a compatible $E \subseteq L \times L$ for $C$ iff

\[
(I_{[S_1,\ldots,S_n]} \cup (\bigcup_{e \in E(D)} I_e)) \cap O_{[S_1,\ldots,S_n]} = \emptyset, \quad (1)
\]

\[
\pi_2(\bigcup_{e \in E(D)} I_e) \cap \pi_2(I_{[S_1,\ldots,S_n]} \cap (\bigcup_{e \in E(D)} O_{(e)})) = \emptyset, \quad (2)
\]

\[
(\bigcup_{e \in E(D)} I_e) \cap (\bigcup_{e \in E(D)} O_{(e)}) = \emptyset. \quad (3)
\]

Moreover, if this is the case, then

\[
(I_{[S_1,\ldots,S_n]} \cup (\bigcup_{e \in E(D)} I_e)) \cap (\bigcup_{e \in E(D)} O_{e}) = \emptyset. \quad (4)
\]

Hence, by Theorem 20, for a given set $S$ of disjoint subgraph and an ordering $C$ of $S$, it is computationally efficient to determine whether or not a compatible $E$ exists. Indeed, we ‘only’ need to compute the sets $I_{[S_1,\ldots,S_n]}$, $O_{[S_1,\ldots,S_n]}$, $\bigcup_{e \in E(D)} I_e$, and $\bigcup_{e \in E(D)} O_{(e)}$, where $D$ be the touching graph of $G$ w.r.t. $C$.

7 Determining compatible sequences of subgraphs

In the previous section we described, below Theorem 20, a method to determine a compatible $E \subseteq L \times L$ for given ordering $C$ of a set $S$ of disjoint subgraphs. In this section we discuss a method to determine, given $S$, a ordering $C$ of $S$ for which there is a compatible $E$. A naïve method would simply check the conditions of Theorem 20 for every possible ordering $C$ until one such $C$ satisfies the conditions. However, as we will show, we can do much better.

First we state the following corollary to Theorem 20 – is better suited to computationally determine a suitable $C$ for given $S$.

Corollary 18. Let $G$ be a graph and $C = (S_1, \ldots, S_n)$ be a sequence of induced subgraphs of $G$ and let $D$ be the touching graph of $G$ w.r.t. $C$. There is a compatible $E \subseteq L \times L$ for $C$ iff

\[
I_{[S_1,\ldots,S_n]} \cap O_{[S_1,\ldots,S_n]} = \emptyset, \quad (5)
\]

\[
I_{[S_1,\ldots,S_n]} \cap O_e = \emptyset \text{ and } I_e \cap O_{[S_1,\ldots,S_n]} = \emptyset \text{ for all } e \in E(D), \quad (6)
\]

\[
I_e \cap O_{(f)} = \emptyset \text{ and } \pi_2(I_e) \cap \pi_2(I_{[S_1,\ldots,S_n]} \cap O_{(f)}) = \emptyset \text{ for all } e, f \in E(D). \quad (7)
\]
Note that Equality 5 (in the corollary) is computationally much easier to check compared to Equality 6, while that equality in turn is much easier to check compared to Equality 7. Note that the condition $I_{[s_1,\ldots,s_n]} \cap O_e = \emptyset$ in Equality 6 is taken from Equality 4, and can hence be removed. As we will see below, we do include it to speed up the evaluation of Equality 7.

For touching graphs $S_1$ and $S_2$, we call $e = (S_1, S_2)$ admissible (w.r.t. $S$) if both $I_{[s_1,\ldots,s_n]} \cap O_e = \emptyset$ and $I_e \cap O_{[s_1,\ldots,s_n]} = \emptyset$. Hence Equality 6 (in the previous corollary) thus states that all edges of $D$ are admissible.

**Definition 19.** The admissible touching graph of $G$ w.r.t. $S$ is the directed graph $A = (S, \{e \in S^2 \mid e \text{ admissible}\})$ (we allow anti-parallel edges).

Consequently, given $S$, we have for admissible graph $A = (S, E_A)$ w.r.t. $S$ and touching graph $T_C = (S, E_{TC})$ w.r.t. any ordering $C$ of $S$, $E_{TC} \subseteq E_A$.

Now, the algorithm to determine an ordering $C$ of $S$ such that there is an $E \subseteq L \times L$ where $E$ is compatible for $C$ is as follows.

1: **procedure** FINDORDERING($G, S$)  
2: \textbf{if} Eq. 5 (of Corollary 21) does not hold \textbf{then} \hfill $\triangleright$ Checking Eq. 5  
3: \hspace{1cm} stop and there is no such $C$  
4: \textbf{end if}  
5: \textbf{Construct the admissible touching graph} $A$ of $G$ w.r.t. $S$.  
6: \hspace{1cm} Let $F_A$ be the condition “there are two touching subgraphs such that $A$ does not have an edge between them”.  
7: \hspace{1cm} progress $\leftarrow$ true  
8: \hspace{1cm} \textbf{while} $F_A$ does not hold \textbf{and} progress holds \textbf{do} \hfill $\triangleright$ Optimization  
9: \hspace{2cm} \textbf{if} there are distinct edges $e$ and $f$ such that $e$ has no anti-parallel edge and Eq. 7 does not hold for $e$ and $f$ \textbf{then}  
10: \hspace{3cm} remove $f$ from $A$  
11: \hspace{3cm} else  
12: \hspace{4cm} progress $\leftarrow$ false  
13: \hspace{3cm} \textbf{end if}  
14: \hspace{1cm} \textbf{end while}  
15: \hspace{1cm} \textbf{if} $F_A$ holds \textbf{then} \hfill $\triangleright$ Checking Eq. 6  
16: \hspace{2cm} stop and there is no such $C$  
17: \hspace{1cm} \textbf{end if}  
18: \hspace{1cm} \textbf{if} there is a topological ordering $C$ of $A$ such that Eq. 7 holds \textbf{then}  
19: \hspace{2cm} stop and $C$ is a suitable ordering \hfill $\triangleright$ Checking Eq. 7  
20: \hspace{1cm} \textbf{end if}  
21: \hspace{1cm} stop and there is no such $C$  
22: \textbf{end procedure}

It is clear now that including equality $I_{[s_1,\ldots,s_n]} \cap O_e = \emptyset$ in the definition of admissible is an optimization: the less number of edges in $A$, the less number of possible ordering $C$ for which Equality 7 needs to be checked. Moreover, lines 7
to 14 represent another optimization to reduce the number of edges of $A$ by detecting local incompatibilities with Equality 7.

8 Discussion

In this paper we considered the problem of graph grammar inference for the case where one is given a disjoint set $S$ of isomorphic subgraphs to be generated by a single rule $r = N \rightarrow S/E$, where the embedding relation $E$ is allowed to contain tuples containing $N$. In this way we generalize results in [2]. This result is to be seen as a further step towards a systematic account of NLC graph grammar inference.

Formally, we characterized, given a $S$, the existence of an ordering $C$ of $S$ and a $E \subseteq L \times L$ such that $E$ is compatible for $C$. Moreover, if such a $C$ exists, then it is shown to be a topological ordering of a suitable graph that identifies admissible pairs of touching subgraphs. The efficiency of the resulting algorithm depends significantly on the cardinality of $S$ – for small $S$ the algorithm seems feasible, however this has yet to be verified in practice.

Finding a graph $S$, such that the set $S$ of subgraphs of $G$ isomorphic to $S$ is (1) “compressible”, i.e. there is a compatible embedding relation for suitable ordering of $S$, and (2) optimal (either in cardinality, or in some other measure) remains to be investigated.

Also, it is natural to consider the case where for rule $r = N \rightarrow S/E$, $N$ is allowed to be a label on a nodes of $S$ instead of $N$ contained in (tuples of) $E$. This would have the consequence that an infinite number of graphs can be generated by $r$, and, moreover, multiple copies of $S$ can overlap – loosening the restriction of disjointness considered here.

References

Effective Capacity and Randomness of Closed Sets

Paul Brodhead\textsuperscript{1} and Douglas Cenzer\textsuperscript{2}

\textsuperscript{1} University of Hawaii, Honolulu, HI 96822, brodhead@math.hawaii.edu
\textsuperscript{2} Department of Mathematics, University of Florida,
P.O. Box 118105, Gainesville, Florida 32611
Phone: 352-392-0281 Fax: 352-392-8357
cenzer@math.ufl.edu

Abstract. We investigate the connection between measure and capacity for the space \( C \) of closed subsets of \( 2^\mathbb{N} \). For any computable measure \( \mu^* \), a computable capacity may be defined by letting \( T(Q) \) be the measure of the family of closed sets \( K \) which have nonempty intersection with \( Q \). We prove an effective version of the Choquet’s theorem by showing that every computable capacity may be obtained from a computable measure in this way. We establish conditions that characterize when the capacity of a random closed set equals zero or is \( > 0 \). We construct for certain measures an effectively closed set with positive capacity and with Lebesgue measure zero.

Keywords: Computability, Randomness, \( \Pi^0_1 \) Classes

1 Introduction

The study of algorithmic randomness has been an active area of research in recent years. The basic problem is to quantify the randomness of a single real number. Here we think of a real \( r \in [0, 1] \) as an infinite sequence of 0’s and 1’s, i.e as an element in \( 2^\mathbb{N} \). There are three basic approaches to algorithmic randomness: the measure-theoretic approach of Martin-Löf tests, the incompressibility approach of Kolmogorov complexity, and the betting approach in terms of martingales. All three approaches have been shown to yield the same notion of (algorithmic) randomness. The present paper will mainly use notions from the measure-theoretic approach, incorporating a number of non-trivial results in this area. For background and history of algorithmic randomness we refer to [6, 8].

Prefix-free (Chaitin) complexity for reals is defined as follows. Let \( M \) be a prefix-free function with domain \( \subset \{0, 1\}^* \). For any finite string \( \tau \), let \( K_M(\tau) = \min\{|\sigma| : M(\sigma) = \tau\} \). There is a universal prefix-free function \( U \) such that, for any prefix-free \( M \), there is a constant \( c \) such that for all \( \tau \)

\textsuperscript{*}This research was partially supported by NSF grants DMS 0532644 and 0554841 and 652372. Thanks also to the American Institute of Mathematics for support during the 2006 Effective Randomness Workshop.
We let \( K(\sigma) = K_U(\sigma) \). Then \( x \) is said to be random if there is a constant \( c \) such that \( K(x|n) \geq n - c \) for all \( n \). This means a real \( x \) is random exactly when its initial segments are not compressible.

From the measure-theoretic viewpoint, a real \( x \) is Martin-Löf random if for any effective sequence \( S_1, S_2, \ldots \) of c.e. open sets with \( \mu(S_n) \leq 2^{-n}, x \notin \cap_n S_n \).

In a series of recent papers [1–4], P. Brodhead, S. Dashti and the authors have defined the notion of (algorithmic) randomness for closed sets and continuous functions on \( 2^\mathbb{N} \). Some definitions are needed. For a finite string \( \sigma \in \{0,1\}^n \), let \( |\sigma| = n \). For two strings \( \sigma, \tau \), say that \( \tau \) extends \( \sigma \) if \( |\sigma| \leq |\tau| \) and \( \sigma(i) = \tau(i) \) for \( i < |\sigma| \). For \( x \in 2^\mathbb{N} \), \( \sigma \subseteq x \) means that \( \sigma(i) = x(i) \) for \( i < |\sigma| \). Let \( \sigma^\perp \tau \) denote the concatenation of \( \sigma \) and \( \tau \) and let \( \sigma^\neg i \) denote \( \sigma^\perp(i) \) for \( i = 0,1 \). Let \( x[n] = (x(0),\ldots,x(n-1)) \). Two reals \( x \) and \( y \) may be coded together into \( z = x \oplus y \), where \( z(2n) = x(n) \) and \( z(2n+1) = y(n) \) for all \( n \). For a finite string \( \sigma \), let \( I(\sigma) \) denote \( \{x \in 2^\mathbb{N} : \sigma \subseteq x\} \). We shall call \( I(\sigma) \) the interval determined by \( \sigma \). Each such interval is a clopen set and the clopen sets are just finite unions of intervals. \( B \) denote the Boolean algebra of clopen sets.

Now a nonempty closed set \( P \) may be identified with a tree \( T_P \subseteq \{0,1\}^* \) where \( T_P = \{ \sigma : P \cap I(\sigma) \neq \emptyset \} \). Note that \( T_P \) has no dead ends. That is, if \( \sigma \in T_P \), then either \( \sigma^\neg 0 \in T_P \) or \( \sigma^\neg 1 \in T_P \). For an arbitrary tree \( T \subseteq \{0,1\}^* \), let \( [T] \) denote the set of infinite paths through \( T \). It is well-known that \( P \subseteq 2^\mathbb{N} \) is a closed set if and only if \( P = [T] \) for some tree \( T \). \( P \) is a \( \Pi^0_1 \) class, or an effectively closed set, if \( P = [\emptyset] \) for some computable tree \( T \). \( P \) is a strong \( \Pi^0_2 \) class, or a \( \Pi^0_2 \) closed set, if \( P = [T] \) for some \( \Delta^0_2 \) tree. The complement of a \( \Pi^0_1 \) class is sometimes called a c.e. open set. We remark that if \( P \) is a \( \Pi^0_1 \) class, then \( T_P \) is a \( \Pi^0_1 \) set, but it is not, in general, computable. There is a natural effective enumeration \( P_0, P_1, \ldots \) of the \( \Pi^0_1 \) classes and thus an enumeration of the c.e. open sets. Thus we can say that a sequence \( S_0, S_1, \ldots \) of c.e. open sets is effective if there is a computable function, \( f \), such that \( S_n = 2^\mathbb{N} - P_{f(n)} \) for all \( n \). For a detailed development of \( \Pi^0_1 \) classes, see [5].

It was observed in [2] that there is a natural isomorphism between the space \( C \) of nonempty closed subsets of \( \{0,1\}^\mathbb{N} \) and the space \( \{0,1,2\}^\mathbb{N} \), defined as follows. Given a nonempty closed \( Q \subseteq 2^\mathbb{N} \), let \( T = T_Q \) be the tree without dead ends such that \( Q = [T] \). Let \( \sigma_0, \sigma_1, \ldots \) enumerate the elements of \( T \) in order, first by length and then lexicographically. We then define the code \( x = x_Q = x_T \) by recursion such that for each \( n \), \( x(n) = 2 \) if both \( \sigma_0^\neg 0 \) and \( \sigma_0^\neg 1 \) are in \( T \), \( x(n) = 1 \) if \( \sigma_0^\neg 0 \notin T \) and \( \sigma_0^\neg 1 \in T \), and \( x(n) = 0 \) if \( \sigma_0^\neg 0 \in T \) and \( \sigma_0^\neg 1 \notin T \).

For a finite tree \( T \subseteq \{0,1\}^{\leq n} \), the finite code \( \rho_T \) is similarly defined, ending with \( \rho_T(k) \) where \( \sigma_k \) is the lexicographically last element of \( T \cap \{0,1\}^n \).

We defined in [2] a measure \( \mu^* \) on the space \( C \) of closed subsets of \( 2^\mathbb{N} \) as follows.

\[
\mu^*(\mathcal{X}) = \mu(\{x_Q : Q \in \mathcal{X}\})
\]

for any \( \mathcal{X} \subseteq C \) and \( \mu \) is the standard measure on \( \{0,1,2\}^\mathbb{N} \). Informally this means that given \( \sigma \in T_Q \), there is probability \( \frac{1}{3} \) that both \( \sigma^\neg 0 \in T_Q \) and \( \sigma^\neg 1 \in T_Q \)
and, for $i = 0, 1$, there is probability $\frac{1}{3}$ that only $\sigma \sim i \in T_Q$. In particular, this means that $Q \cap I(\sigma) \neq \emptyset$ implies that for $i = 0, 1$, $Q \cap I(\sigma \sim i) \neq \emptyset$ with probability $\frac{2}{3}$.

Brodhead, Cenzer, and Dashti [2] defined a closed set $Q \subseteq 2^\mathbb{N}$ to be (Martin-Löf) random if $x_Q$ is (Martin-Löf) random. Note that the equal probability of $\frac{1}{3}$ for the three cases of branching allows the application of Schnorr’s theorem that Martin-Löf randomness is equivalent to prefix-free Kolmogorov randomness. Then in [2, 3], the following results are proved. Every random closed set is perfect and contains no computable elements (in fact, it contains no $n$-c.e. elements). Every random closed set has measure 0 and has box dimension $\log_2 \frac{4}{3}$.

**Definition 1.** A capacity on $\mathcal{C}$ is a function $T : \mathcal{C} \rightarrow [0, 1]$ with $T(\emptyset) = 0$ such that

(i) $T$ is monotone increasing, that is,

$$Q_1 \subseteq Q_2 \rightarrow T(Q_1) \leq T(Q_2).$$

(ii) For $n \geq 2$ and any $Q_1, \ldots, Q_n \in \mathcal{C}$

$$T(\bigcap_{i=1}^n Q_i) \leq \sum_{I} \{(\pm 1)^{|I|+1}T(\bigcup_{i \in I} Q_i) : \emptyset \neq I \subseteq \{1, 2, \ldots, n\}\}.$$  

-this is the alternating of infinite order property.

(iii) If $Q = \bigcap_n Q_n$ and $Q_{n+1} \subseteq Q_n$ for all $n$, then $T(Q) = \lim_{n \to \infty} T(Q_n)$.

We will also assume, unless otherwise specified, that the capacity $T(2^\mathbb{N}) = 1$.

The standard (hit-or-miss) topology on $\mathcal{C}$ has a sub-basis of sets of two types, where $Q$ is any closed set.

$$V(Q) = \{K : K \cap Q \neq \emptyset\};$$

$$W(Q) = \{K : K \subseteq Q\}.$$

We will say that a capacity $T$ is computable if it is computable on the family of clopen sets. It follows that the capacity of any $\Pi^0_1$ class is upper semi-computable.

A basis for the hit-or-miss topology may be formed by taking finite intersections of these. But we can define a smaller basis for the $\sigma$-field of Borel sets. For each $n$ and each finite tree $A \subseteq \{0, 1\}^{\leq n}$, let

$$U_A = \{K : (\forall \sigma \in A)(K \cap I(\sigma) \neq \emptyset) \& (\forall \sigma \notin A)(K \cap I(\sigma) = \emptyset)\}.$$  

That is,

$$U_A = \{K : T_K \cap \{0, 1\}^{\leq n} = A\}.$$  

For any finite $n$ and any tree $T \subseteq \{0, 1\}^{\leq n}$, define the clopen set $[T] = \bigcup_{\sigma \in T} I(\sigma)$. Then $K \cap [T] \neq \emptyset$ if and only if there exists some $A \subseteq \{0, 1\}^{\leq n}$ such that $K \in U_A$ and $A \cap T \neq \emptyset$. That is,

$$V([T]) = \bigcup\{U_A : A \cap T \neq \emptyset\}.$$
Similarly, \( K \subseteq [T] \) if and only if there exists some \( A \subseteq \{0, 1\}^n \) such that \( K \in U_A \) and \( A \subseteq T \). That is,

\[
W([T]) = \bigcup\{U_A : A \subseteq T\}.
\]

Note that \( V(Q) \) is also closed in this topology, since if \( Q = \cap_n Q_n \) where \( Q_n \) are clopen, then \( K \notin V(Q) \) if and only if \( K \cap Q_n = \emptyset \) for some \( n \), which is if and only if \( K \in W(2^{\mathbb{N}} \setminus Q_n) \).

Now any closed set \( Q = [T] \) is a decreasing intersection of clopen sets \( Q_n = [T^n] \), where \( T^n = T \cap \{0, 1\}^n \). Then we have \( W(Q) = \cap_n W(Q_n) \) by definition and \( V(Q) = \cap_n V(Q_n) \) by compactness. Thus it suffices to define \( \mu(V(Q)) \) and \( \mu(W(Q)) \) for clopen sets and then take \( \mu(V(Q)) = \lim_n \mu(V(Q_n)) \) and similarly for \( W(Q) \). Furthermore, for clopen \( Q \), \( W(Q) \) is the complement of \( V(2^{\mathbb{N}} \setminus Q) \), so that it suffices to define \( \mu(V(Q)) \) for clopen sets to get a measure on \( C \). It follows from the discussion above that it suffice to define the measure on sets of the form \( U_A \).

As described above and in [3], we may identify a nonempty closed set \( Q \) with an element \( x_Q \in \{0, 1, 2\}^{\mathbb{N}} \). A probability measure on \( \{0, 1, 2\}^{\mathbb{N}} \) may be defined as in [9] from a function \( d : \{0, 1, 2\}^* \to [0, 1] \) such that \( d(\lambda) = 1 \) and, for any \( \sigma \in \{0, 1, 2\}^* \),

\[
d(\sigma) = \sum_{i=0}^{2} d(\sigma^{-i}).
\]

The corresponding measure \( \mu_d \) on \( \{0, 1, 2\}^{\mathbb{N}} \) is then defined by letting \( \mu_d(I(\sigma)) = d(\sigma) \). Since the intervals \( I(\sigma) \) form a basis for the standard product topology on \( \{0, 1, 2\}^{\mathbb{N}} \), this will extend to a measure on all Borel sets. If \( d \) is computable, then \( \mu_d \) is said to be computable. The measure \( \mu_d \) is said to be nonatomic or continuous if \( \mu_d(\{x\}) = 0 \) for all \( x \in \{0, 1, 2\}^{\mathbb{N}} \). We will say that \( \mu_d \) is bounded if there exist bounds \( b, c \in (0, 1) \) such that, for any \( \sigma \in \{0, 1, 2\}^* \) and \( i \in \{0, 1, 2\} \),

\[
b \cdot d(\sigma) < d(\sigma^{-i}) < c \cdot d(\sigma).
\]

It is easy to see that any bounded measure must be continuous. We will say that the measure \( \mu_d \) is regular if there exist constants \( b_0, b_1, b_2 \) with \( b_0 + b_1 + b_2 = 1 \) such that for all \( \sigma \) and for \( i \leq 2 \), \( d(\sigma^{-i}) = b_i d(\sigma) \).

Now let \( \mu_d^* \) be defined by

\[
\mu_d^*(Z) = \mu_d(\{x_Q : Q \in Z\}).
\]

**Proposition 1.** For any \( d \), the measure \( \mu_d^* \) is defined on all Borel sets in the hit-or-miss topology on \( C \). Furthermore, if \( d \) is computable, then \( \mu_d^* \) is computable on the family of clopen sets.

**Proof.** By the discussion above it suffice to show that \( \mu_d^*(U_A) \) is defined for all \( A \subseteq \{0, 1\}^n \). But it is easy to see that

\[
K \in U_A \iff \rho_A \supseteq x_K
\]

so that in fact \( U_A \) is a clopen set in \( C \). The computability now follows from the fact that \( \mu_d^*(U_A) = \mu_d(I(\rho_A)) \). \( \square \)
2 Capacity and Randomness

This section presents the effective version of Choquet’s theorem. We compute the capacity of a random closed set under certain probability measures. We construct a $\Pi^0_1$ class with measure zero but with positive capacity.

Define $T_d(Q) = \mu_d^*(V(Q))$. That is, $T_d(Q)$ is the probability that a randomly chosen closed set meets $Q$. We next give an effective version of the well-known classical result connecting measure and capacity. For details on capacity and random set variables, see [7].

**Theorem 1.** If $\mu_d^*$ is a (computable) probability measure on $\mathcal{C}$, then $T_d$ is a (computable) capacity.

**Proof.** This is easily verified. Certainly $T(\emptyset) = 0$. The alternating property follows by basic probability. For (iii), suppose that $Q = \cap_n Q_n$ is a decreasing intersection. Then by compactness, $Q \cap K \neq \emptyset$ if and only if $Q_n \cap K \neq \emptyset$ for all $n$. Furthermore, $V(Q_{n+1}) \subseteq V(Q_n)$ for all $n$. Thus

$$T(Q) = \mu(V(Q)) = \mu(\cap_n V(Q_n)) = \lim_n \mu(V(Q_n)) = \lim_n T(Q_n).$$

The computability of $T$ is easily verified. That is, for any clopen set $I(\sigma_1) \cup \cdots \cup I(\sigma_k)$ where each $\sigma_j \in \{0,1\}^n$, we compute the probability distribution for all trees of height $n$ and add the probabilities of those trees which contain one of the $\sigma_i$. □

The (noneffective) converse of this result is due to Choquet; see [7] for details. Here is an effective version of Choquet’s theorem.

**Theorem 2.** If $T$ is a computable capacity, then there is a computable measure $\mu_d^*$ on the space of closed sets such that $T = T_d$.

**Proof.** Given the values $T(U)$ for all clopen sets $I(\sigma_1) \cup \cdots \cup I(\sigma_k)$ where each $\sigma_i \in \{0,1\}^n$, there is in fact a unique probability measure $\mu_d$ on these clopen sets such that $T = T_d$ and this can be computed as follows.

Suppose first that $T(I(i)) = a_i$ for $i < 2$ and note that each $a_i \leq 1$ and $a_0 + a_1 \geq 1$ by the alternating property. If $T = T_d$, then we must have $d((0)) + d((2)) = a_0$ and $d((1)) + d((2)) = a_1$ and also $d((0)) + d((1)) + d((2)) = 1$, so that $d((2)) = a_0 + a_1 - 1$, $d((0)) = 1 - a_1$ and $d((1)) = 1 - a_0$. This will imply that $T(\tau) = T_d(\tau)$ when $|\tau| = 1$. Now suppose that we have defined $d(\tau)$ and that $\tau$ is the code for a finite tree with elements $\sigma_0, \ldots, \sigma_n = \sigma$ and thus $d(\tau \cdot \bar{i})$ is giving the probability that $\sigma$ will have one or both immediate successors. We proceed as above. Let $T(I(\sigma \cdot \bar{i})) = a_i : T(I(\sigma))$ for $i < 2$. Then as above $d(\tau \cdot \bar{2}) = d(\tau) \cdot (a_0 + a_1 - 1)$ and $d(\tau \cdot \bar{i}) = d(\tau) \cdot (1 - a_i)$ for each $i$. □

We say that $K \in \mathcal{C}$ is $\mu_d^*$-random if $x_K$ is Martin-Löf random with respect to the measure $\mu_d$. 

74
Theorem 3. Let $d$ be the uniform measure with $b_0 = b_1 = b_2 = \frac{1}{3}$. Then for any $\mu^*_d$-random closed set $R$, $T_d(R) = 0$.

Proof. Fix the measure $d$ on $\{0,1,2\}^\mathbb{N}$ as described above so that $d(\sigma^i) = d(\sigma) \cdot b$ for $i = 0,1$ and $d(\sigma^2) = d(\sigma) \cdot (1 - 2b)$. Let $\mu^* = \mu^*_d$. We will compute the probability, given two closed sets $Q$ and $K$, that $Q \cap K$ is nonempty. Let

$$Q_n = \bigcup\{I(\sigma) : \sigma \in \{0,1\}^n \land Q \cap I(\sigma) \neq \emptyset\}$$

and similarly for $K_n$. Then $Q \cap K \neq \emptyset$ if and only if $Q_n \cap K_n \neq \emptyset$ for all $n$. Let $p_n$ be the probability that $Q_n \cap K_n \neq \emptyset$ for two arbitrary closed sets $K$ and $Q$, relative to our measure $\mu^*$.

It is immediate that $p_1 = \frac{7}{9}$, since $Q_1 \cap K_1 = \emptyset$ only when $Q_1 = I(i)$ and $K_1 = I(1 - i)$. Next we will determine the quadratic function $f$ such that $p_{n+1} = f(p_n)$. There are 9 possible cases for $Q_1$ and $K_1$, which break down into 4 distinct cases.

**Case I:** There are two chances that $Q_1 \cap K_1 = \emptyset$.

**Case II:** There are two chances that $Q_1 = K_1 = I(i)$, so that $Q_{n+1} \cap K_{n+1} \neq \emptyset$ with probability $p_n$.

**Case III:** There are four chances where $Q_1 = 2^\mathbb{N}$ and $K_1 = I(i)$ or vice versa, so that once again $Q_{n+1} \cap K_{n+1} \neq \emptyset$ with probability $p_n$.

**Case IV:** There is one chance that $Q_1 = K_1 = 2^\mathbb{N}$, in which case $Q_{n+1} \cap K_{n+1} \neq \emptyset$ with probability $1 - (1 - p_n)^2 = 2p_n - p_n^2$. This is because $Q_{n+1} \cap K_{n+1} = \emptyset$ if and only if both $Q_{n+1} \cap I(i) \cap K_{n+1} = \emptyset$ for both $i = 0$ and $i = 1$.

Adding these cases together, we see that

$$p_{n+1} = \frac{6}{9} p_n + \frac{1}{9} (2p_n - p_n^2) = \frac{8}{9} p_n - \frac{1}{9} p_n^2.$$ 

It follows that the sequence $\langle p_n \rangle_{n \in \omega}$ is computable and we will see that the limit is zero. Let $f(p) = \frac{8}{9} p - \frac{1}{9} p^2$.

Elementary calculus shows that $f$ has fixed points at $p = -1$ and $p = 0$ and that for $0 < p < 1$, $0 < f(p) < p$. Since $p_0 = \frac{7}{9}$, it follows that the sequence $\langle p_n \rangle_n$ is monotonic decreasing. Thus the limit exists and $\lim_{n \to \infty} p_n = 0$ (since it must be a fixed point of $f$).

Thus the probability that $Q \cap K \neq \emptyset$ equals $\lim_{n \to \infty} p_n = 0$. Next we will obtain a Martin-Löf test to prove our result.

For each $m, n \in \omega$, let

$$B_m = \{(K, Q) : K_m \cap Q_m \neq \emptyset\},$$

so that $\mu^*(B_m) = p_m$ and let

$$A_{m,n} = \{Q : \mu^*(\{K : K_m \cap Q_m \neq \emptyset\}) \geq 2^{-n}\}.$$

Let $C_m$ be the number of trees of height $m$ without dead ends.
Then we have

\[ \mu^*(A_{m,n}) \leq 2^{n+1} \mu^*(B_m), \]

by the following argument. For each \( Q \) in \( A_{m,n} \) there are \( 2^{-n} \) possible choices for \( K_m \) such that \( K_m \cap Q_m \neq \emptyset \) and thus at least \((\frac{1}{2})2^{-n}\mu^*(A_{m,n})C_m^2\) ordered pairs altogether in \( B_m \) (since each pair might be counted twice).

Now define a computable sequence \( (m_n)_{n \in \omega} \), so that \( p_{m_n} < 2^{-n-1} \). Let

\[ S_n = A_{m,n} \]

and let

\[ S'_n = \cup_{r>n} S_r. \]

It follows that

\[ \mu^*(S_n) \leq 2^{n+1} \mu^*(B_{m,n}) < 2^{n+1}2^{-2n-1} = 2^{-n} \]

and therefore

\[ \mu^*(S'_n) \leq \sum_{r>n} 2^{-r} = 2^{-n} \]

Now let \( R \) be a random closed set. The sequence \( (S'_n)_{n \in \omega} \) is a computable sequence of c.e. open sets with measure \( \leq 2^{-n} \), so that there is some \( n \) such that \( R \notin S_n \). Thus for \( r > n \), \( \mu^*(\{K : K_{m_r} \cap R_{m_r} \neq \emptyset\}) < 2^{-r} \) and it follows that

\[ \mu^*(\{K : K \cap R \neq \emptyset\}) = \lim_n \mu^*(\{K : K_{m_n} \cap R_{m_n} \neq \emptyset\}) = 0. \]

Thus \( T_d(R) = 0 \), as desired. \( \Box \)

This result depends on the particular measure. For different regular measures, the capacity of a random closed set can have different values. The following can be proved.

**Theorem 4.** Let \( d \) be the uniform measure with \( b_0 = b_1 = b > 0 \) and \( b_2 = 1 - 2b > 0 \) and let \( \hat{b} = 1 - \frac{\sqrt{2}}{2} \). Then

(a) If \( b \leq \hat{b} \), then for any \( \mu_d^* \)-random closed set \( R \), \( T_d(R) = 0 \).

(a) If \( b > \hat{b} \), then there is a \( \mu_d^* \)-random closed set \( R \) with \( T_d(R) > 0 \).

Thus for certain measures, there exists a random closed set with measure zero but with positive capacity. A random closed set may not be effectively closed. But we can also construct an effectively closed set with measure zero and with positive capacity.

**Theorem 5.** For the regular measure \( \mu_d \) with \( b = b_1 = b_2 \), there is a \( \Pi^0_1 \) class \( Q \) with Lebesgue measure zero and positive capacity \( T_d(Q) \).
Proof. First let us compute the capacity of $X_n = \{ x : x(n) = 0 \}$. For $n = 0$, we have $T_d(Q_0) = 1 - b$. Now the probability $T_d(X_{n+1})$ that an arbitrary closed set $K$ meets $X_{n+1}$ may be calculated in two distinct cases. As in the proof of Theorem 3, let

$$K_n = \bigcup \{ I(\sigma) : \sigma \in \{0, 1\}^n \& K \cap I(\sigma) \neq \emptyset \}$$

Case I If $K_0 = 2^\mathbb{N}$, then $T_d(X_{n+1}) = 1 - (1 - T_d(X_n))^2$.

Case II If $K_0 = I((i))$ for some $i < 2$, then $T_d(X_{n+1}) = T_d(X_n)$.

It follows that

$$T_d(X_{n+1}) = 2b T_d(X_n) + (1 - 2b)(2T_d(X_n) - (T_d(X_n))^2)$$

$$= (2 - 2b)T_d(X_n) - (1 - 2b)(T_d(X_n))^2$$

Now consider the function $f(p) = (2 - 2b)p - (1 - 2b)p^2$, where $0 < b < \frac{1}{2}$. This function has the properties that $f(0) = 0$, $f(1) = 1$ and $f(p) > p$ for $0 < p < 1$. Since $T_d(X_{n+1}) = f(T_d(X_n))$, it follows that $\lim_n T_d(X_n) = 1$ and is the limit of a computable sequence.

For any $\sigma = (n_0, n_1, \ldots, n_k)$, with $n_0 < n_1 < \cdots < n_k$, similarly define $X_\sigma = \{ x : (\forall i < k)x(n_i) = 0 \}$. A similar argument to that above shows that $\lim_n T_d(X_{\sigma} - n)/T_d(X_\sigma) = 1$.

Now consider the decreasing sequence $c_k = \frac{c_{k+1} + 1}{2^{k+1}}$ with limit $\frac{1}{2}$. Choose $n = n_0$ such that $T_d(X_n) \geq \frac{3}{4} = c_0$ and for each $k$, choose $n = n_{k+1}$ such that $T_d(X_{n_0, \ldots, n_k, n}) \geq c_{k+1}$. This can be done since $c_{k+1} < c_k$. Finally, let $Q = \bigcap_k X_{(n_0, \ldots, n_k)}$. Then $T_d(Q) = \lim_k T_d(X_{(n_0, \ldots, n_k)}) \geq \lim_k c_k = \frac{1}{2}$. \qed

3 Conclusions and Future Research

In this paper, we have established a connection between measure and capacity for the space $C$ of closed subsets of $2^\mathbb{N}$. We showed that for a computable measure $\mu^*$, a computable capacity may be defined by letting $T(Q)$ be the measure of the family of closed sets $K$ which have nonempty intersection with $Q$. We have proved an effective version of the Choquet’s theorem by showing that every computable capacity may be obtained from a computable measure in this way.

For the uniform measure $\mu$ under which a node $\sigma$ in $T$ has exactly one immediate extension $\sigma \supset i$ with probability $b$ for $i = 0, 1$ (and hence $\sigma$ has both extensions with probability $1 - 2b$), we have established conditions on $b$ that characterize when the capacity of a random closed set equals zero or is $> 0$.

We have also constructed for each such measure an effectively closed set with positive capacity and with Lebesgue measure zero.

In future work, we plan to extend our results to more general measures where for each string $\sigma \in T_Q$, the probability that $\sigma \supset i \in T_Q$ depends on $\sigma$. For example, such a measure on the space of closed sets may be defined by making the probability that both extensions $\sigma \supset i$ of a node $\sigma \in T$ belong to $T$ equal to $1 - \frac{2}{n^2}$ and the probability that just one extension belongs to $T$ equal to $\frac{1}{n}$, where $n = |\sigma|$.
References


Lévy Flights for Ant Colony Optimization in Continuous Domains

Roberto Candela¹, Giulio Cottone², Giuseppe Fileccia Scimemi² and Eleonora Riva Sanseverino¹

¹ University of Palermo, DIEET, Viale delle Scienze Palermo, Italy
² University of Palermo, DISAG, Viale delle Scienze Palermo, Italy

Abstract. In this paper, the authors propose the use of the Lévy probability distribution as leading mechanism for solutions differentiation in an efficient and bio-inspired optimization algorithm, ant colony optimization in continuous domains, ACOR. In the classical ACOR, new solutions are constructed starting from one solution, selected from an archive, where Gaussian distribution is used for parameter diversification. In the proposed approach, the Lévy probability distributions are properly introduced in the solution construction step, in order to couple the ACOR algorithm with the exploration properties of the Lévy distribution. ...

1 Introduction

Recently, a lot of work has been carried out in bio-inspired computational optimization, especially in continuous domains. Among the methods set up, we can cite Evolutionary computation [1], [2] and nature inspired methods such as Ant Colony Optimization, ACO. The latter was initially developed for combinatorial optimization [3], [4] and has been recently adapted to continuous optimization [5]. Ant Colony Optimization is inspired by the ants foraging behavior and requires that the problem is partitioned into a finite set of components these being intermediate targets before reaching the ultimate goal. The solution is generally the minimum-cost strategy followed by the agent (ant) to reach the target. In Ant Colony Optimization for continuous optimization, ACOR, the partition of the problem into finite set is given by the intrinsic search space decomposition into the different dimensions.

ACOR is a population based algorithm, therefore, for each ant two main steps must be performed: the ant based solution construction and the pheromone update. The first step comprises a number of sub-operations such as: for each search space dimension: a) probabilistic choice of one base solution from an archive of best-so-far solutions; b) perturbation of the relevant parameter following a gaussian probability distribution. The second step simply consists in the archive update. Indeed, the Gaussian distribution used for the perturbation of each parameter is built using the information derived from the entire archive. The closer the solutions are in a given dimension, the smaller the standard deviation. With
respect to this issue, the authors have noticed in some cases a limited ability to perform exploration. This is especially true when multimodal functions must be minimized. Other authors [6], [7] have already observed this behavior with evolutionary programming. In particular, Lee and Yao [6] propose an adaptive LEP, Lévy based Evolutionary Programming algorithm in which the Gaussian mutation is replaced by a Lévy distributed mutation. In this paper the authors propose to modify the ACOR algorithm by replacing the Gaussian mutation with a Lévy distributed mutation, called ACOR\textsubscript{L}. The effects are similar to those attained in [7], although different performances are observed in certain cases. The application section reports tests over a set of test functions taken from [6].

2 The Lévy probability distribution

In the field of global optimization, various phenomena have been already studied in the literature such as thermodynamics and evolution in the eighties. More recently, one of the laws governing enhanced diffusive processes called Lévy flights has been considered for modeling the perturbation mechanism in global optimization. In this paper, the Lévy distribution is considered for generating step size during the ACOR search. In this case, this This distribution has the property of generating points that can be far from the starting ones. This can be better understood considering that Gaussian white noises random variables are symmetric about their mean and do not allow any skewness of the distribution or unilateral random input. The only distribution that allows such a great variability and obeys to a generalized central limit theorem is the so called \(\alpha\)-stable Lévy distribution, introduced by the mathematician Paul Lévy about 1920 [8]. Stable distributions are characterized by heavy-tailed probability density function that causes infinite variance and are defined by four coefficients [9]. A random variable \(X\) is said to have a \(\alpha\)-stable distribution if there are parameters \(0 < \alpha \leq 2\), \(\sigma > 0\), \(1 \leq \beta \leq 1\), \(\mu \in \mathbb{R}\) such that its characteristic function \(\phi_X(\theta)\) has the form:

\[
\phi_X(\theta) = \begin{cases} 
\exp\left\{-\sigma^\alpha|\theta|^\alpha \left(1 - i\beta (\text{sign}(\theta)) \tan \frac{\pi \alpha}{2}\right) + i\mu \theta\right\}, & \text{if } \alpha \neq 1 \\
\exp\left\{-\sigma|\theta| \left(1 + i\beta \frac{2}{\pi} (\text{sign}(\theta)) \ln|\theta|\right) + i\mu \theta\right\}, & \text{if } \alpha = 1 
\end{cases}
\]

The four parameters affect the shape of the distribution in an essential way and it is common to introduce an appropriate notation to take them into account. We denote with the symbol \(X \sim S_\alpha(\sigma, \beta, \mu)\) a stable random variable with assigned parameters characterizing (1). Some properties of the stable distribution, not proved but straightforward from the definition of the characteristic function, will help to better clarify their meaning.

2.1 Addition of constant.

Let \(X \sim S_\alpha(\sigma, \beta, \mu)\) and let \(a\) a real parameter. Then, adding \(a\) to \(X\) gives a random variable \(X + a\) with distribution \(X \sim S_\alpha(\sigma, \beta, \mu + a)\). The parameter \(\mu\) is thus a shift parameter. The parameter \(\mu\) cannot be identified in general as
the mean of the distribution, because for $0 < \alpha < 1$, the mean of the variable $X \sim S_\alpha(\sigma, \beta, \mu)$ diverges. Only in the interval $1 < \alpha \leq 2$, the two concepts actually coincide.

2.2 Multiplication by a constant.

Let be $X \sim S_\alpha(\sigma, \beta, \mu)$ and $a$ real. Then, multiplying $X$ by $a$ gives a random variable $aX$ with distribution $X \sim S_\alpha(|a|\sigma, \text{Sign}(a)\beta, a\mu)$ if $\alpha \neq 1$ and $X \sim S_\alpha(|a|\sigma, \text{Sign}(a)\beta, a\mu - (2/\pi)a(\log|a|)\sigma)\beta)$ if $\alpha = 1$. $\sigma$ is called scale parameter. When $\alpha = 2$, the characteristic function (1) becomes the characteristic function of a random variable normal distributed with mean $\mu$ and variance $2\sigma^2$, indicated as $X \sim N(\mu, \sqrt{2}\sigma)$. In general, the scale parameter does not coincide with the standard deviation, that, for $0 < \alpha < 2$ is infinite.

2.3 Symmetry property.

Conversely to the normal distributions, stable distributions show great flexibility for different settings of parameters $\sigma, \beta, \mu$. The skewness parameter $\beta$ gives the measure of dissymmetry of the probability density function about the shift value $\mu$. Indeed, the random variable $X \sim S_\alpha(\sigma, \beta, \mu)$ is symmetric about $\mu$ if and only if $\beta = 0$. If also $\mu = 0$, the characteristic function is real and consequently the probability density function is symmetric as shown by (1). When $\alpha = 2$ the skewness parameter does not influence the characteristic function, because the term in $\beta$ actually disappears, due to the presence of the tangent function in (1). Conventionally, in this case $\beta$ is assumed to be zero. The normal distribution is indeed defined just by two parameters $X \sim S_2(\sigma, \beta, \mu) = N(\mu, \sqrt{2}\sigma)$. In Figure 1, two trajectories following the normal ($\alpha=2$) and the Lévy distribution ($\alpha=1.6$) are reported. The trajectories are generated by adding a Lévy distributed quantity having zero mean and $\sigma=1$ to $x_1$ and $x_2$.

![Fig. 1. Trajectories of Lévy motion: (a) typical normal path $\alpha = 2$; (b) competition between jumps and small fluctuation at $\alpha = 1.6$](image)

The normal path in panel (a) is sample continuous (similar, but not to be confused with the continuity of a function, and descending from the application
of the Kolmogorov criterion \[10\] while in panel (b) the Lévy path is not sample continuous as long jumps and clustered small fluctuations are present alternate with clustered small fluctuation. This is a consequence of the heavy tails of the Lévy distribution and it is influenced by the stability index: indeed, if \(\alpha\) goes to zero, jumps become bigger and fluctuations vanish; conversely, if \(\alpha=2\) the continuous (no jumps) normal behaviour is attained. Then, loosely speaking, we could say that Lévy paths tend to escape from a bounded region, while normal paths localize.

3 Function optimization using ACOR\(_L\)

As said in the introduction, Ant Colony Optimization was first proposed for combinatorial optimization problems. Since its emergence many attempts have been made to use it for tackling continuous problems. More recently, M. Dorigo and K. Socha \[5\] have proposed the natural extension of the ACO algorithm to continuous domains, ACOR\(_L\). The idea that is central to the way ACOR works is the incremental construction of solutions based on the biased (by pheromone) probabilistic choice of solution components. At each construction step, the ant chooses a Probability Density Function. Details about the ACOR implementation are out of the scope of this paper, for further details please refer to \[5\]. In what follows, the main steps of the ACOR\(_L\) algorithm are briefly outlined.

Create an archive \(T\) of \(k\) solutions, \(T = \{x^1, x^2, \ldots x^k\}\). Where \(x^r = [x^r_1, x^r_2, \ldots x^r_N]\). Order the solutions of the archive \(T\) according to their objective function value. Given a decision variable \(x_i, i=1, N\), an ant constructs a solution by performing \(N\) construction steps. At construction step \(i\), the ant chooses a value for the variable \(x_i\). At this construction step, only the information related to the \(i\)-th dimension is used. Select a base solution \(r\) from the archive \(T\) to be modified according to the following probability:

\[
p_r = \frac{\omega_r}{\sum_{j=1}^k \omega_j}
\]

where

\[
\omega_r = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(r-1)^2}{2q^2k^2}}
\]

which essentially defines the weight \(\omega_r\) to be a value of the Gaussian function with argument \(r\), mean 1 and standard deviation \(qk\), where \(q\) is a parameter of the algorithm. When \(q\) is small, the best-ranked solutions are strongly preferred, and when it is large, the probability becomes less dependent on the rank of the solution.

All the components \(x^r_i\) for \(i=1\) to \(N\) of the chosen \(r\)-th solution in the following steps are perturbed following the Lévy distribution. As already pointed out, the Lévy distribution is characterised by four parameters: the scale parameter, \(\sigma\), the skewness parameter, \(\beta\), the shift parameter \(\mu\) and the \(\alpha\) parameter.

The first is defined as:
\[ \sigma_i^r = \xi \sum_{i=1}^{k} \frac{|x_i^e - x_i^r|}{k-1} \] (4)

where \(\xi\) is a parameter user-defined in the algorithm ranging from 0 and 1. The higher the value of this parameter the slower the convergence speed.

The third parameter, \(\mu\), is the value of the i-th parameter of the base solution itself (\(x_i^e\)). The second parameter is set to 0, namely no dissymmetry of the probability density function about the shift value \(\mu\). The fourth parameter \(\alpha\) is a control parameter set by the user and its value ranges between 0 and 2. So the i-th parameter is newly determined. The same procedure is repeated for all the N parameters. At the end, once the solution is entirely constructed, it is evaluated and if better than any of the solutions in \(T\), it is included into the archive set \(T\). From what was said above, if the used defined parameter \(\alpha\) is set to 2, the \(ACOR_L\) coincides with ACOR. The proposed algorithm, \(ACOR_L\), for function optimization works as follow.

Algorithm 1 ACOR\(_L\) Pseudocode

Random creation of the solutions archive of size \(k\)
Choice of \(\xi, q, \alpha, \mu\)
while not(termination) do
  for z=1 to \(m\) do
    Choice of one solution from the archive using (2)
    for all parameter (Ant construction) do
      Calculate standard deviation \(\sigma_i^r\) using (4)
      Modify the i-th parameter in the following way:
      \(x_i^r = x_i^r + S\alpha(\sigma_i^r, 0, 0)\)
    end for
    Evaluation of the new solution
  end for
  Archive update
end while

4 Experimental results and analysis

We applied ACOR with Gaussian perturbation and \(ACOR_L\) with Lévy perturbation to a set of benchmark optimization problems (\(ACOR_L\) with \(\alpha=2\) is standard ACOR). Table I shows the benchmark functions and the ranges of the variables that have been used in this study. These were considered in an early study [6]. We divided the functions into three classes: functions without local minima, \(f_1\), with many local minima, \(f_5, f_6, f_7\), and with a few local minima, \(f_{11}\). In order to compare the attained results with those attained using classical ACOR, for both algorithms the following parameters values taken from [5] have
been chosen: parameter $\xi=0.85$; parameter $q=0.0001$; archive size: 50; number of ants: 2. Parameter $\alpha$ has been set to different values in different runs, in order to assess its influence on the efficiency of the algorithm. Thus the following values of $\alpha$ have been taken: $2.0, 1.8, 1.6, 1.4, 1.2, 1.0, 0.8$. The termination condition is based on the maximum number of function evaluations, which has been set to 150000 for $f_1, f_5, f_6, f_7$, and 3000 for $f_{11}$. Table 1 reports the functions $f_1, f_5, f_6, f_7$ and $f_{11}$ [6]. Table 2 shows the experimental results of a comparison between the performance of ACOR and ACOR$_L$ with the above parameters and with $\alpha=1.8$. Other values of $\alpha$ have shown a worst behaviour.

### Table 1. Benchmark functions used in this study.

<table>
<thead>
<tr>
<th>Test function</th>
<th>$N_s$</th>
<th>$f_1$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1 = \sum_{i=1}^{N} x_i^2$</td>
<td>30 [-100, 100]$^N$</td>
<td>\multicolumn{5}{c</td>
<td>}{2.36E-144}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_5 = \sum_{i=1}^{N} [x_i^2 - 10\cos(2\pi x_i) + 10]$</td>
<td>30 [-5.12, 5.12]$^N$</td>
<td>\multicolumn{5}{c</td>
<td>}{2.34E-143}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_6 = -20\exp[-0.2\sqrt{\frac{1}{N} \sum_{i=1}^{N} x_i^2}] - \exp[\frac{1}{N} \sum_{i=1}^{N} \cos(2\pi x_i)]$</td>
<td>30 [-32, 32]$^N$</td>
<td>\multicolumn{5}{c</td>
<td>}{1.26E+01}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_7 = \frac{1}{\prod_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos(\frac{x_i}{N}) + 1}$</td>
<td>30 [-600, 600]$^N$</td>
<td>\multicolumn{5}{c</td>
<td>}{1.23}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_{11} = (1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2))(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)$</td>
<td>2 [-2, 2]$^N$</td>
<td>\multicolumn{5}{c</td>
<td>}{1.07E-02}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 2. Experimental results, mean of the best solution over 100 independent runs, from standard ACOR and ACOR$_L$. The number in the parentheses indicate standard deviations. The third column shows the t-test. The asterisk indicates that the difference is not negligible.

<table>
<thead>
<tr>
<th></th>
<th>$f_1$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOR</td>
<td>2.36E-144</td>
<td>4.63E+01</td>
<td>5.63E-01</td>
<td>8.93E-03</td>
<td>4.62</td>
</tr>
<tr>
<td>($\alpha=1.8$)</td>
<td>(2.34E-143)</td>
<td>(1.26E+01)</td>
<td>(1.23)</td>
<td>(1.07E-02)</td>
<td>(1.14E+01)</td>
</tr>
<tr>
<td>ACOR$_L$</td>
<td>2.24E-203</td>
<td>6.17E+01</td>
<td>2.28</td>
<td>2.02E-02</td>
<td>3.27</td>
</tr>
<tr>
<td>($\alpha=2.0$)</td>
<td>(0)</td>
<td>(1.68E+01)</td>
<td>(1.38)</td>
<td>(2.71E-02)</td>
<td>(2.7)</td>
</tr>
<tr>
<td>t-test</td>
<td>1.08</td>
<td>-7.73*</td>
<td>-9.25*</td>
<td>-3.88*</td>
<td>1.15</td>
</tr>
</tbody>
</table>

It is clear from the table above that ACOR$_L$ performed no worse or better in a statistically meaningful sense than ACOR on all benchmark functions. This is clearly shown by the values of the t-test.

In what follows, a study of the behaviour of ACOR$_L$ for the different considered values of $\alpha$ has been carried out on some of the test functions. In the figures each point represents the mean or the standard deviation or the median of the best so far solution at each iteration for a sample of 100 runs. Fig.2 shows the optimization processes, averaged over 100 independent trials, for ACOR$_L$ (ACOR is ACOR$_L$ with $\alpha=2$) with different values of $\alpha$ over function $f_1$. In this
case, ACOR converges faster than ACOR\textsubscript{L} for all values of $\alpha$. The acronym FES stays for function evaluations.

Figure 2 shows an improved behaviour of the algorithm with higher values of $\alpha$, namely as the diversification mechanism approaches the Gaussian mutation ($\alpha=2$).

Lévy distribution for the perturbation allowing long jumps (smaller values of $\alpha$) can be detrimental when the population is close to the global optimum. The offspring generated by long jumps tends to move away from the global optimum.

A different behaviour can be observed for functions showing many local minima, such as $f_5$. The following figures show the behaviour of ACOR\textsubscript{L} over $f_5$.

After a fast descent of the mean, ACOR gets stuck into local minima and does not improve its performance compared to $\alpha=1.6$ and $\alpha=1.8$. These both improve their mean value till the very end of the run, reaching lower values of the
objective function. The behaviour is confirmed looking at the standard deviation. In ACOR, the standard deviation first increases allowing a wide exploration of the search space, then decreases and after about 30000 evaluations it does not change anymore. At the same time, it can be observed that the mean value does not move anymore. For different values of $\alpha$, and in particular for $\alpha=1.8$, the standard deviation decreases gradually, while the search process approaches the minimum. In this case, it is also interesting to observe the median, see figure 4.

![Fig. 4. Optimization process of ACOR$_L$ over test function $f_5$. Median.](image)

The behaviour of the standard deviation observed for $\alpha=2$ shows a limited tendency to diversification of the attainable solutions. Higher diversification can be observed for $\alpha=1.8$ and $\alpha=1.6$ for a longer part of the process, leading to lower values of the mean and of the median at the end of the process. Lower median means that a high diversification has brought a large number of very good results and a limited number of bad results over the set of independent runs. A similar behaviour has been observed for the other functions showing many local minima ($f_6$ and $f_7$). For functions showing few local minima, such as $f_{11}$, the following figures show a still different behaviour. The means and standard deviations this time have a very similar behaviour for all values of $\alpha$. Thus performances can be considered similar.

Finally a comparison of performances has been carried out between the LEP algorithm proposed in [6] and the ACOR$_L$ with $\alpha=1.8$.

It is clear from the table 3 above that ACOR$_L$ performed better than LEP on all benchmark functions except for the function $f_5$ and not significantly for the function $f_{11}$.

5 Conclusion

In this paper, a new perturbation operator, based on Lévy distribution, is proposed for Ant Colony Optimization in continuous domains. The modified algorithm is here called ACOR$_L$. The behaviour of the algorithm in some interesting
Fig. 5. Optimization process of ACOR_L over test function f_{11}. Standard deviation and mean.

<table>
<thead>
<tr>
<th></th>
<th>f_1</th>
<th>f_5</th>
<th>f_6</th>
<th>f_7</th>
<th>f_{11}</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOR_L</td>
<td>2.36E-144</td>
<td>4.63E+01</td>
<td>5.63E-01</td>
<td>8.93E-03</td>
<td>4.62</td>
</tr>
<tr>
<td></td>
<td>(2.34E-143)</td>
<td>(1.26E+01)</td>
<td>(1.23)</td>
<td>(1.07E-02)</td>
<td>(1.14E+01)</td>
</tr>
<tr>
<td>LEP</td>
<td>1.98E-03</td>
<td>3.83E+01</td>
<td>9.75E-01</td>
<td>4.12E-02</td>
<td>3.26</td>
</tr>
<tr>
<td></td>
<td>(3.51E-04)</td>
<td>(8.98)</td>
<td>(2.20)</td>
<td>(4.78E-02)</td>
<td>(3.69E-01)</td>
</tr>
<tr>
<td>t-test</td>
<td>56.41*</td>
<td>-4.01*</td>
<td>1.47</td>
<td>6.44*</td>
<td>-0.84</td>
</tr>
</tbody>
</table>

Table 3. Experimental results, mean averaged over 100 runs, from LEP [6] and ACOR_L. The number in the parentheses indicate standard deviations. The t-test values listed are LEP-ACOR_L. The asterisk indicates that the difference is not negligible.

cases has been observed and studied in the paper. As it can be noted the wider exploration potential of the Lévy distribution allows the algorithm in case of multimodal functions showing many local minima to attain statistically significant better performance than standard ACOR. Further studies will be addressed towards the implementation of an adaptive version of ACOR_L.

References

Iterative Rounding for the Closest String Problem

Jingchao Chen
School of Informatics, Donghua University
2999 North Renmin Road, Songjiang District, Shanghai 201620, P. R. China
chen-jc@dhu.edu.cn

Abstract. The closest string problem is an NP-hard problem, whose task is to find a string that minimizes maximum Hamming distance to a given set of strings. This can be reduced to an integer program (IP). However, to date, there exists no known polynomial-time algorithm for IP. In 2004, Meneses et al. introduced a branch-and-bound (B&B) method for solving the IP problem. Their algorithm is not always efficient and has the exponential time complexity. In the paper, we attempt to solve efficiently the IP problem by a greedy iterative rounding technique. The proposed algorithm is polynomial time and much faster than the existing B&B IP for the CSP. If the number of strings is limited to 3, the algorithm is provably at most 1 away from the optimum. The empirical results show that in many cases we can find an exact solution. Even though we fail to find an exact solution, the solution found is very close to exact solution.

1 Introduction

The task of finding a string that is close to each string in a given set of strings is one of combinatorial optimization problems, which arise in computational molecular biology and coding theory. This problem is called the closest string problem (CSP). We introduce some notations to defining more precisely the CSP. Let \( \Sigma \) stand for a fixed finite alphabet. Its element is called character, and a sequence of characters over it is called string, denoted by \( s \). The length and \( i \)-th character of \( s \) are denoted by \( |s| \) and \( s[i] \), respectively. \( d(s,t) \) is defined as the Hamming distance between two equal-length strings \( s \) and \( t \), i.e. the number of characters where they do not agree. This may be formulated as \( d(s,t) = \sum f(s[i],t[i]) \), where \( f(s[i],t[i]) \) is one if \( s[i] \neq t[i] \), and zero otherwise. Let \( \Sigma^n \) be the set of all strings of length \( n \) over \( \Sigma \). Then, the CSP is defined exactly as follows.

Given a finite set \( S = \{s_1,s_2,\ldots,s_m\} \) of \( m \) strings, each of which is in \( \Sigma^n \), the objective is to find a center string \( t \) of length \( n \) over \( \Sigma \) minimizing the distance \( d \) such that, for every string \( s_i(1 \leq i \leq m) \) in \( S \), \( d(t,s_i) \leq d \).

The CSP has received the attention of many researchers in the recent few years. The literature abounds with the CSP. In theory, Frances and Litman [FL1] have proven that it is NP-hard. However, if the distance \( d \) is fixed, the exact
solution to the problem can be found in polynomial time [GN1,BG1]. For the
general case where $d$ is variable, one is involved in studying approximation algo-
rithms. There have been some approximation algorithms with good theoretical
precision. For example, Gasieniec et al. [GJ1] and Lanctot et al. [LL1] devel-
oped independently a 4/3-approximation algorithm. On the basis of this, Li et
al. [LM1] presented a polynomial-time approximation scheme (PTAS). However,
the PTAS is not practical.

Meneses et al. [ML1] studied many approximation algorithms, and found that
the mentioned-above algorithms are of only theoretical importance, not directly
applicable to bioinformatics practice because of high time complexity. For this
reason, they suggested reducing the CSP to an integer-programming (IP) prob-
lem, and then using branch-and-bound (B&B) algorithm to solve the IP problem.
Unfortunately, integer programs are also NP-hard. So far, no polynomial-time
algorithm for solving integer programs has been found. Furthermore, the B&B
has its own drawbacks. It leads easily to memory explosion due to excessive ac-
cumulation of active nodes. In fact, our empirical results show that the B&B IP
is not efficient. In despite of instances of moderate size, the B&B IP fails to find
an optimal solution sometimes.

We want to find efficiently an exact solution via a technique called iterative
rounding. The reason for using this technique is because we noted that Jain
[JA1], Cherian and Vempala [CV1] used it and succeeded in getting a better
approximation algorithm for the generalized steiner network problem. Although
our problem is different from their problem, both are NP-hard. Therefore, we
believe this technique is applicable to the CSP. The iterative rounding method
used here is a greedy one. It may be outlined as follows. First we formulate the
CSP as an IP, and then use the LP solution to round some of higher valued
variables, finally repeatedly re-solve the LP for the remaining variables until all
variables are set. The method has small memory requirement, and can avoid
memory explosion of the B&B IP. It is a polynomial time algorithm which can
find an exact solution in a very short time for a CSP instance of moderate size
in many cases. The computational experiments reveal that our algorithm is not
only much faster than the existing one, but also has high quality. If the number
of strings is limited to 3, the error of the algorithm is proven to be at most one.

Unlike the existing rounding schemes, our rounding scheme is iterative, not
random, while the existing ones such as the rounding scheme of Lanctot et al.
[LL1] are random. An important contribution of our algorithm is in setting up
a new approach for finding the exact CSP algorithm with the polynomial-time.

2 Iterative rounding for the CSP

The CSP can be reduced to a 0-1 Integer Programming problem as follows.

$$\min_d \sum_{a \in \Xi} x_{a,j} = 1, \quad j = 1, \ldots, n$$

$$n - \sum_{j=1}^n x_{a[i,j],j} \leq d, \quad i = 1, \ldots, m$$
where $x_{a,j} \in \{0,1\}$, $a_{[i,j]} \in \Sigma$, and $d$ is a non-negative integer. Solving this IP problem by applying directly LP (Linear Programming) relaxation and randomized rounding does not work well because randomized rounding procedure leads to large errors, especially when the optimal distance $d$ is small [LM1]. Therefore, we decided to find other rounding techniques. Jain [JA1] used iterative rounding to get a 2-approximation algorithm for the generalized Steiner network problem. Based on our observation, iterative rounding is suited also for the CSP. Hence, we use it to solve the CSP. The following pseudo-code is a CSP algorithm with iterative rounding.

Algorithm A

Formulate the CSP as an IP.

$V_1 \leftarrow \emptyset$, $V_0 \leftarrow \emptyset$

for $i = 1$ to $n$

1. Fix all variables in $V_1$ to 1, and all variables in $V_0$ to 0
2. Solve the LP for the sub-CSP on the unfixed variables
3. Pick a variable $x_{b,m}$ with highest value, i.e., $x_{b,m} = \max\{x_{a,k} | k = 1, \ldots, n, a \in \Sigma \text{ and } x_{a,k} \notin V_1\}$
4. $V_1 \leftarrow V_1 \cup \{x_{b,m}\}$
5. $V_0 \leftarrow V_0 \cup \{x_{a,m} | a \neq b \text{ and } a \in \Sigma\}$

end for

Convert $V_1$ into a solution (a center string $t$) to the CSP as follows.

$t[k] \leftarrow a$ for all $x_{a,k} \in V_1$.

Clearly, Algorithm A is a polynomial-time algorithm. Furthermore, we have

Theorem 1. If the input consists of only two strings, i.e., $S = \{s_1, s_2\}$, then Algorithm A always find an exact solution to the CSP.

Proof. Without loss of generality, we assume

\[
\begin{array}{c}
  s_1 \\
  s_2
\end{array}
\begin{array}{c}
  \underbrace{000 \ldots 000} \\
  \underbrace{111 \ldots 111}
\end{array}
\]

(Notice, in the case when the same positions of two strings $s_1$, $s_2$ have the same characters, the proof is simpler than in the above case.)

It is easy to see that the 1st LP optimal solution to the CSP is

\[
x_{11}^1 + x_{12}^1 + x_{13}^1 + \cdots + x_{1n}^1 = n/2
\]

where $x_{1k}^1 + x_{0k}^1 = 1$ for $k = 1, \ldots, n$.

Without loss of generality, assume $x_{11}^1 = \max\{x_{a,k}^1 | k = 1, \ldots, n, a \in \{0,1\}\}$.

If $n \geq 2$, there exists $0 \geq x_{12}, x_{13}, \ldots, x_{1n} \geq 1$ such that

\[
x_{12} + x_{13} + \cdots + x_{1n} = n/2 - 1
\]

Say, $x_{12} = x_{13} = \cdots = x_{1n} = (n-2)/(2(n-1))$ is just a solution to this equation.

Then, when $n \geq 2$, setting $x_{11}$ to 1, we can get the 2nd LP optimal solution

\[
1 + x_{12}^2 + x_{13}^2 + \cdots + x_{1n}^2 = n/2,
\]

By induction on $n$, we can prove that the $k$-th ($k = 1, \ldots, n$) LP optimal solution satisfies
\[ x^k_{11} + x^k_{12} + x^k_{13} + + x^k_{1n} = n/2, \]

where at least \( k - 1 \) values out of \( x^k_{11}, \ldots, x^k_{1n} \) are integers.

Hence, if \( n \) is even, if and only if there are \( n/2 \) one’s among \( x^n_{11}, \ldots, x^n_{1n} \). This is just an optimal solution to the CSP.

if \( n \) is odd, assume \( x^n_{1n} \) is not an integer. We have \( (n - 1)/2 \) one’s among \( x^n_{11}, \ldots, x^n_{1n} \) if setting \( x^n_{1n} \) to 0, and \( (n + 1)/2 \) one’s otherwise. Both two cases are an optimal solution to the CSP. Therefore, the theorem is proved. \( \square \)

Define the error of an algorithm as the difference between the exact solution (distance) and the solution obtained. We have

**Theorem 2.** If the input consists of only three binary strings, i.e., \( S = \{s_1, s_2, s_3\} \), then the error of Algorithm A is at most one.

**Proof.** In general, any three strings can be simplified into

\[
\begin{array}{cc}
\alpha & 000\ldots000 \\
\beta & 000\ldots000 \\
\gamma & 111\ldots111 \\
\end{array}
\]

Assume that \( \alpha \neq 0, \beta \neq 0, \gamma \neq 0 \), and the closest string \( t \) (optimal solution) is of the following form,

\[
\begin{array}{cc}
\alpha & 00\ldots0011\ldots11 \\
\beta & 00\ldots0011\ldots11 \\
\gamma & 00\ldots0011\ldots11 \\
\end{array}
\]

where \( t^\alpha_0 \) is the number of 0’s in the \( \alpha \) substring of \( t \). Similarly for \( t^\alpha_0, t^\beta_0, t^\beta_1, t^\gamma_0, t^\gamma_1 \).

Assume the distances between \( t \) and the three strings are \( D_1, D_2 \) and \( D_3 \), respectively, we have

\[
\begin{align*}
(d(t, s_1)) = D_1 \\
(d(t, s_2)) = D_2 \\
(d(t, s_3)) = D_3
\end{align*}
\]

(1)

The optimal distance is denote by \( D \). Then \( D = \max(D_1, D_2, D_3) \).

the following proposition is true.

\[
\begin{align*}
t^\alpha_0 = 0 & \quad or \quad t^\beta_0 = 0 & \quad or \quad t^\gamma_0 = 0
\end{align*}
\]

(2)

If it is false, by (1) we have

\[
\begin{align*}
(t^\alpha_1 - 1) + (t^\beta_1 + 1) + (t^\gamma_0 - 1) = D_1 - 1 \\
(t^\alpha_1 - 1) + (t^\beta_0 - 1) + (t^\gamma_1 + 1) = D_2 - 1 \\
(t^\alpha_0 + 1) + (t^\beta_0 - 1) + (t^\gamma_0 - 1) = D_3 - 1
\end{align*}
\]

It follows that \( D = \max(D_1 - 1, D_2 - 1, D_3 - 1) = D - 1 \), which is a contradiction.

By (2), we have that one of the following three propositions is true.

\[
\begin{align*}
(a) & \quad t^\alpha_1 = 0 \quad can \quad constitute \quad a \quad optimal \quad solution, \quad but \quad t^\alpha_1 \neq 0 \quad cannot. \\
(b) & \quad t^\beta_0 = 0 \quad can \quad constitute \quad an \quad optimal \quad solution, \quad but \quad t^\beta_0 \neq 0 \quad cannot. \\
(c) & \quad t^\gamma_0 = 0 \quad can \quad constitute \quad an \quad optimal \quad solution, \quad but \quad t^\gamma_0 \neq 0 \quad cannot.
\end{align*}
\]

Here we consider only the 2nd case to prove the theorem, since other cases is similar. That is, assume

\[
\begin{align*}
\text{for any optimal solution,} & \quad t^\gamma_0 = 0
\end{align*}
\]

(3)

This implies
By (3) and (6), (1) can be rewritten as

\[ t_1^\alpha + (t_1^\beta - 1) + t_0^\gamma = D_1 - 1 \]
\[ t_1^\alpha + (t_0^\beta + 1) + t_1^\gamma = D_2 + 1 \]
\[ t_0^\beta + (t_0^\beta + 1) + t_0^\gamma = D_3 + 1 \]

\( t_0^\beta + 1 \) is also a optimal solution, which is in contradiction with (3).

Without loss of generality, suppose

\[ \alpha \leq \gamma \]

(If \( \alpha > \gamma \), the subsequent proof is similar). This implies

\[ t_1^\alpha \leq t_0^\beta \]

(6)

If it is false, let \( T_1^\alpha = t_1^\alpha - t_0^\beta \) and \( T_0^\alpha = t_0^\beta + t_0^\gamma \), we can rewrite (1) as

\[ T_1^\alpha + t_1^\beta = D_1 - 2t_0^\gamma \]
\[ T_1^\alpha + T_0^\beta + \gamma = D_2 \]
\[ T_0^\alpha + t_0^\beta = D_3 \]

By (4), we have

\[ D = \max(D_2, D_3) = \max(T_1^\alpha + t_0^\beta + \gamma, T_0^\alpha + t_0^\beta) > \gamma \]

However, in fact, by fixing \( t_1^\alpha = t_0^\beta = t_0^\gamma = 0 \), solving (1) yields \( (D_1, D_2, D_3) = (\beta, \gamma, \alpha) \). Then by (4) and (5), \( D \leq \max(\beta, \gamma, \alpha) \leq \gamma \), which is a contradiction.

By (3) and (6), (1) can be rewritten as

\[ \beta + T_0^\gamma = D_1' \]
\[ T_1^\gamma = D_2 \]
\[ \alpha + T_0^\gamma = D_3 \]

(7)

where \( T_0^\gamma = t_0^\gamma - t_1^\alpha, T_1^\gamma = T_1^\gamma + t_1^\alpha \) and \( D_1' = D_1 - 2t_1^\alpha \).

This implies

\[ |D_3 - D_2| \leq 1 \]

(8)

If it is false, by (7), we can obtain a solution with \( t_0^\beta = 1 \), which is in contradiction with (3).

Let \( x_1^0, x_2^0, \ldots, x_n^0 \) be 0-variables of the LP, \( x_1^1, x_2^1, \ldots, x_n^1 \) 1-variables. Define

\[ L_0^\alpha = x_0^0 + x_2^0 + \cdots + x_\alpha^0 \]
\[ L_1^\alpha = x_1^1 + x_2^1 + \cdots + x_\alpha^1 \]
\[ L_0^\beta = x_{\alpha+1}^0 + x_{\alpha+2}^0 + \cdots + x_{\alpha+\beta}^0 \]
\[ L_1^\beta = x_{\alpha+1}^1 + x_{\alpha+2}^1 + \cdots + x_{\alpha+\beta}^1 \]
\[ L_0^\gamma = x_{\alpha+\beta+1}^0 + x_{\alpha+\beta+2}^0 + \cdots + x_n^0 \]
\[ L_1^\gamma = x_{\alpha+\beta+1}^1 + x_{\alpha+\beta+2}^1 + \cdots + x_n^1 \]

Let \( d_1, d_2 \) and \( d_3 \) denote the distances between the three strings and the center string of the LP, respectively. Then,

\[ L_1^\alpha + L_1^\beta + L_0^\gamma = d_1 \]
\[ L_1^\alpha + L_0^\beta + L_1^\gamma = d_2 \]
\[ L_0^\alpha + L_0^\beta + L_0^\gamma = d_3 \]
\[ L_0^\alpha + L_1^\beta = \alpha \quad L_0^\beta + L_1^\beta = \beta \quad L_0^\gamma + L_1^\gamma = \gamma \]

(9)

Let \( d \) denote the optimal distance of the LP. Then \( d = \max(d_1, d_2, d_3) \). The goal of the LP is to find a minimum \( d \) satisfying (9). Next we analyze the error caused by Algorithm A to solve the LP given in (9).
Depending on \((D'_1, D_2, D_3) = (D, D, D - 1)\) or not, we proceed to our proof. First, let us consider
\[
(D'_1, D_2, D_3) \neq (D, D, D - 1)
\] (10)
This implies
\[
(D_2, D_3) = (D, D)
\] (11)
If it is false, by (8), we have
\[
(D_2, D_3) = (D - 1, D) \text{ or } (D_2, D_3) = (D, D - 1)
\]
Then by (4), we have that \(t_0^\beta = 1\) is also an optimal solution, which is in contradiction with (3).
By (11) and (7), it is easy to verify \(D = (\alpha + \gamma)/2\). Then
\[
d \leq D = (\alpha + \gamma)/2
\] (12)
The addition of the 2nd and 3rd equation in (9) yields
\[
\alpha + 2L_0^\beta + \gamma = d_2 + d_3 \leq 2d \leq \alpha + \gamma
\]
It follows that \(L_0^\beta = 0\). Thus
\[
L_1^\beta = \beta
\]
This implies that without rounding error, Algorithm A fixes all the letters in the \(\beta\) substring into 1. It remains to how to compute \(L_0^\alpha, L_1^\alpha, L_0^\gamma\) and \(L_1^\gamma\).

Let \(M_0^\alpha\) be the maximum of \(t_0^\alpha\) such that
\[
\begin{align*}
t_0^\alpha + \beta + t_1^\gamma & \leq D_1 \\
t_1^\alpha + t_1^\gamma & \leq D_2 \\
t_0^\alpha + t_1^\gamma & \leq D_3
\end{align*}
\] (13)
Similarly, \(M_1^\alpha, M_0^\gamma\) and \(M_1^\gamma\) are the maximum of \(t_1^\alpha, t_0^\gamma\) and \(t_1^\gamma\) s.t. (13). Let \(\alpha_0(i)\) be the number of letters in the substring fixed to 0 by the \(i\)-th rounding operation of Algorithm A. Similarly for \(\alpha_1(i), \gamma_0(i)\) and \(\gamma_1(i)\). If for all \(i \leq n = \alpha + \beta + \gamma\), \(\alpha_0(i) \leq M_0^\alpha, \alpha_1(i) \leq M_1^\alpha, \gamma_0(i) \leq M_0^\gamma\) and \(\gamma_1(i) \leq M_1^\gamma\), Algorithm A attains an exact solution. Otherwise, there exists \(k\) such that only one of \(\alpha_0(i), \alpha_1(i), \gamma_0(i)\) and \(\gamma_1(i)\) exceeds its maximum. Without loss of generality, assume \(\alpha_1(k) = M_1^\alpha + 1\) (other cases, proof is similar). By (13), there exist \(N_0^\gamma\) and \(N_1^\gamma\) such that
\[
\begin{align*}
N_0^\gamma + N_1^\gamma & = \gamma \\
\alpha_1(k) + \beta + N_0^\gamma & = C_1 \\
\alpha_1(k) + N_1^\gamma & = C_2 \\
\alpha_0(k) + N_0^\gamma & = C_3 \leq D \\
(C_1, C_2) & = (D - 1, D) \text{ or } (C_1, C_2) = (D + 1, D) \text{ or } (C_1, C_2) = (D - 1, D)
\end{align*}
\] (14)
Below we justify
\[
\text{for all } i > k, \alpha_1(i) = \alpha_1(k)
\] (15)
Assume the solution of the \(i\)-th \((i > k)\) LP is
\[
\begin{align*}
L_1^\alpha + \beta + L_0^\gamma & = d_1 \\
L_1^\alpha + L_1^\gamma & = d_2 \\
L_0^\alpha + L_0^\gamma & = d_3
\end{align*}
\] (16)
Clearly \(L_1^\alpha \geq \alpha_1(k), L_0^\alpha \leq \alpha_0(k)\)
(17)
By (14), we have
\[
d_1 - d_3 = L_0^\alpha - L_0^\gamma + \beta \geq \alpha_1(k) - \alpha_0(k) + \beta = C_1 - C_3 \geq 0
\] (18)
Therefore \(d = \max(d_1, d_2, d_3) = \max(d_1, d_2) = \max(L_0^\alpha + \beta + L_0^\gamma, L_1^\alpha + L_1^\gamma)\). Namely, \(d\) decreases as \(L_1^\alpha\) decreases. Thus, by (17) we have
\[ L_1^\alpha = \alpha_1(k) \] (19)

The claim of (15) is proved. Next we shall show that

for all \( i > k, \alpha_1(k) + \beta + \gamma_0(i) \leq D \) & \( \alpha_1(k) + \gamma_1(i) \leq D \)

\[ d = d_1 = d_2 = (C_1 + C_2)/2 \] (20)

By (16), (19), (15), we have

\[ d_1 + d_2 = 2L_1^\alpha \] (21)

Therefore, by (18), we have

\[ d = \max(d_1, d_2) \geq (d_1 + d_2)/2 = (C_1 + C_2)/2 \] (21)

(14) can be rewritten as

\[ \alpha_1(k) + \beta + (N_0^\gamma - ((C_1 - C_2)/2)) = (C_1 + C_2)/2 \]
\[ \alpha_1(k) + (N_0^\gamma + ((C_1 - C_2)/2)) = (C_1 + C_2)/2 \]
\[ \alpha_0(k) + (N_0^\gamma) = (C_1 + C_2)/2 \] (22)

Clearly, \( L_0^\gamma = (N_0^\gamma - ((C_1 - C_2)/2)) \) is a feasible solution of the \( i \)-th \((i > k)\) LP, but not necessarily optimal. Therefore \( d \leq (C_1 + C_2)/2 \). By the constraint of \( C_1, C_2 \) and \( C_3 \) in (14), it is easy to verify

\[ C_3 - (C_1 - C_2)/2 \leq (C_1 + C_2)/2. \]

Thus, by (21) and (22), the claim of (20) is proved.

Below we shall prove

\[ \exists j > k \text{ s.t. } \alpha_1(k) + \beta + \gamma_0(j) = D + 1 \text{ implies } \forall i > j, \gamma_0(i) = \gamma_0(j) \] (23)

Assume \( j > k, \alpha_1(k) + \beta + \gamma_0(j) = D + 1, i > j \)

Then, the \( L_0^\gamma \) of the \( i \)-th LP satisfies \( L_0^\gamma \geq \gamma_0(j) \) (25)

Then, by (19), (24) we have

\[ L_0^\alpha + \beta + L_0^\gamma \geq \alpha_1(k) + \beta + \gamma_0(j) = D + 1 \]

Thus
\[ d \geq D + 1 \] (26)

On the other hand, by (14), we can prove

\[ \alpha_1(k) + \beta + \gamma_0(j) = D + 1 \]
\[ \alpha_1(k) + \gamma_1(j) \leq D + 1 \]
\[ \alpha_0(k) + \gamma_0(j) \leq D + 1 \]

Therefore \( L_0^\gamma = \gamma_0(j) \) is a feasible solution of the \( i \)-th LP. It means \( d \leq D + 1 \).

Thus, by (26), \( d = D + 1 \). This implies \( L_0^\gamma \leq \gamma_0(j) \). Then by (25), the claim of (23) is proven.

In a way similar to the proof of (23), we can prove

\[ \exists j > k \text{ s.t. } \alpha_1(k) + \gamma_1(j) = D + 1 \text{ implies } \forall i > j, \gamma_1(i) = \gamma_1(j) \] (27)

By (20), (23), (27) and the previous proof, we conclude that in the case \((D'_1, D_2, D_3) \neq (D, D, D-1)\), the error of Algorithm A is at most one. Now we consider the case

\[ (D'_1, D_2, D_3) = (D, D, D-1) \]

The addition of the 1st and 2nd equation in (7) yields

\[ \beta + \gamma = D'_1 + D_2 = 2D \] (28)

The addition of the 1st and 2nd equation in (9) yields

\[ 2L_1^\alpha + \beta + \gamma = d_1 + d_2 \leq 2d \leq 2D \]

Then by (28), \( L_1^\alpha = 0 \). This is equivalent to \( L_0^\alpha = \alpha \). That is, without rounding error, Algorithm A fix all the letters of the \( \alpha \) substring into 0. It remains to how to compute \( L_0^\beta, L_1^\beta, L_0^\gamma \) and \( L_1^\gamma \). By symmetry, we can prove in a way similar to
the previous that Algorithm A computes $L^\beta_0$, $L^\beta_1$, $L^\gamma_0$ and $L^\gamma_1$ within one error of optimal distance.

Based our empirical observation, the error caused by the algorithm was always within one. Hence, for any $m$, the number of the input strings, we have

Conjecture 1. For any input, the error of Algorithm A is at most one.

3 Improving the running time and quality of the solution

To speed up the algorithm, we present Algorithm B, which picks multiple (not single) variables of higher values to round up at a time. That is, in the rounding phase, this algorithm searches always for multiple higher valued variables, and then set them to one’s, and the other variables at the same positions to zero’s. Selection is done by parameter $\Theta$, which is set to 0.9 in our experiment. As long as $x_{a,j} \geq \Theta$, we set the solution of the $j$-th position to $a$.

**Algorithm B**

**Input:** $s_1, s_2, \ldots, s_m$ and a threshold $\Theta \geq 0.9$

**Output:** a center string $t \in \Sigma$ close to every string $s_i$

1. for $1 \leq j \leq n$ do $t[j] \leftarrow \phi \notin \Sigma$.
2. repeat the following process until all $t[j] \neq \phi$.

   2.1 Solve the LP-relaxation

   2.2 Let $x'_{a,j}$ be the value of $x_{a,j}$ for the LP optimal solution.

      if there exists an $x'_{a,j} \geq \Theta$

      then for all $x'_{a,j} \geq \Theta$ and $t[j] = \phi$ do $t[j] \leftarrow a$

      else find $x'_{b,j}$ such that $x'_{b,j} = \max\{x'_{a,j} | a \in \Sigma, t[j] = \phi\}$

      \[ t[k] \leftarrow b \]

To get a higher precision, we improve Algorithm B by Algorithm C. It tries not only the best, but also the second best. If the first solution is not optimal, we select 8 positions to be re-solved the most possibly in the increasing order of variable values. The first position of a solution to be re-solved is one out of the 8 positions. Its value is set to the character corresponded by the second best valued variables. We update the initial setting to find a new solution. Thus, using 8 different settings, we can find 8 different solutions. Finally, we choose the best one out of 9 solutions, including the 1st solution.

**Algorithm C**

1. Let first$[k]$, second$[k]$ store the largest value of $x$’s variables in the $k$-th position, second$[k]$ the character with the second largest value.

2. Invoke Algorithm B with the following modification: the “else” statement of

   **Algorithm B** is revised as

   find $x'_{b,j} = \max\{x'_{a,j} | a \in \Sigma, t[j] = \phi\}$

   \[ t[k] \leftarrow b \quad \text{first}[k] = x'_{b,j} \]

   second$[k] \leftarrow c$ with $x'_{c,j} = \max\{x'_{a,j} | a \in \Sigma, a \neq b\}$
3. if the objective value of $t = \text{that of the LP rounded up}$, return.
else $T \leftarrow t$

4. for $1 \leq i \leq 8$ do

   for $1 \leq j \leq n$ do $t[j] \leftarrow \phi$
   
   $t[k_i] \leftarrow \text{second}[k_i]$, where first$[k_i]$ is $i$-th smallest

   Use Step 2 of Algorithm B to re-solve the CSP

   if the current solution $t$ is better than $T$ then $T \leftarrow t$

5. $t \leftarrow T$

<table>
<thead>
<tr>
<th>Instance</th>
<th>Average distance</th>
<th>Max distance error</th>
<th>Average time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 300</td>
<td>175.00</td>
<td>175.00</td>
<td>175.00</td>
</tr>
<tr>
<td>10 400</td>
<td>231.67</td>
<td>231.67</td>
<td>231.67</td>
</tr>
<tr>
<td>10 500</td>
<td>293.00</td>
<td>293.00</td>
<td>293.00</td>
</tr>
<tr>
<td>10 600</td>
<td>347.00</td>
<td>347.00</td>
<td>347.00</td>
</tr>
<tr>
<td>10 700</td>
<td>409.00</td>
<td>409.00</td>
<td>409.00</td>
</tr>
<tr>
<td>10 800</td>
<td>462.67</td>
<td>462.67</td>
<td>462.67</td>
</tr>
<tr>
<td>15 300</td>
<td>185.33</td>
<td>185.67</td>
<td>185.67</td>
</tr>
<tr>
<td>15 400</td>
<td>246.67</td>
<td>247.33</td>
<td>246.67</td>
</tr>
<tr>
<td>15 500</td>
<td>306.67</td>
<td>307.00</td>
<td>306.67</td>
</tr>
<tr>
<td>15 600</td>
<td>366.67</td>
<td>367.00</td>
<td>366.67</td>
</tr>
<tr>
<td>15 700</td>
<td>428.67</td>
<td>428.67</td>
<td>428.67</td>
</tr>
<tr>
<td>15 800</td>
<td>491.00</td>
<td>491.00</td>
<td>491.00</td>
</tr>
<tr>
<td>20 300</td>
<td>190.67</td>
<td>191.00</td>
<td>191.00</td>
</tr>
<tr>
<td>20 400</td>
<td>252.33</td>
<td>252.67</td>
<td>252.67</td>
</tr>
<tr>
<td>20 500</td>
<td>315.33</td>
<td>315.33</td>
<td>315.33</td>
</tr>
<tr>
<td>20 600</td>
<td>379.67</td>
<td>380.00</td>
<td>380.00</td>
</tr>
<tr>
<td>20 700</td>
<td>443.33</td>
<td>443.33</td>
<td>443.33</td>
</tr>
<tr>
<td>20 800</td>
<td>505.00</td>
<td>505.00</td>
<td>505.00</td>
</tr>
<tr>
<td>25 300</td>
<td>195.00</td>
<td>196.00</td>
<td>196.00</td>
</tr>
<tr>
<td>25 400</td>
<td>259.00</td>
<td>260.00</td>
<td>259.67</td>
</tr>
<tr>
<td>25 500</td>
<td>323.00</td>
<td>323.67</td>
<td>323.67</td>
</tr>
<tr>
<td>25 600</td>
<td>387.67</td>
<td>388.00</td>
<td>387.67</td>
</tr>
<tr>
<td>25 700</td>
<td>451.00</td>
<td>451.33</td>
<td>451.33</td>
</tr>
<tr>
<td>25 800</td>
<td>515.67</td>
<td>516.67</td>
<td>516.67</td>
</tr>
<tr>
<td>30 300</td>
<td>197.33</td>
<td>197.67</td>
<td>197.67</td>
</tr>
<tr>
<td>30 400</td>
<td>263.00</td>
<td>263.67</td>
<td>263.33</td>
</tr>
<tr>
<td>30 500</td>
<td>328.33</td>
<td>329.00</td>
<td>328.67</td>
</tr>
<tr>
<td>30 600</td>
<td>392.67</td>
<td>393.00</td>
<td>393.33</td>
</tr>
<tr>
<td>30 700</td>
<td>459.33</td>
<td>460.00</td>
<td>459.67</td>
</tr>
<tr>
<td>30 800</td>
<td>523.00</td>
<td>523.33</td>
<td>523.67</td>
</tr>
</tbody>
</table>
4 Simulations

On Celeron 2.2GHz CPU, we tested two algorithms: our Algorithm C and the B&B IP by Meneses et al. which is referred to as the best IP for the CSP so far.

We carried out many experiments, including McClure data set [ML1] and random instances over the alphabet with 2 characters, 4 characters and 20 characters. In all experiments, our algorithm’s performance was very good. For the limit of space, we present only the empirical results for random instances over the alphabet with 4 characters. In Table 1, we provided three instances for each entry. Parameters $m$ and $n$ stand for the number of strings and the string size. “distance” and “time” refer to the minimum distance found, and the running time in milliseconds. LP average distance is computed as $(|d_1| + |d_2| + |d_3|)/3$. The reason for taking the ceiling here is because the optimal solution for the CSP is no less than the ceiling of the LP value. In the 6th, 7th column, Max distance error is defined as $\max_{i=1}^{3} \{|d_i - d_i^{LP}|\}$, where $d_i$ is the $i$-th solution, and $d_i^{LP}$ is the $i$-th LP fractional solution. The maximum time allowed for each instance was set to 1000 seconds. As was seen in Table 1, we found always an exact solution except for a few instances. In terms of running time, our improvement was huge. Our algorithm was from 32 up to 912 times faster than the B&B IP. In other experiments, which is not listed here, it was even 1765 times faster. In some cases, its speed was even close to one for computing an LP. Notice, our algorithm invokes generally many LP solvers. Even so, in the worst case, it was only 20 times slower than computing an LP.

References

Computable Types for Dynamic Systems

Pieter Collins

Centrum Wiskunde en Informatica
Postbus 94079
1090 GB Amsterdam
Pieter.Collins@cwi.nl

Abstract. In this paper, we develop a theory of computable types suitable for the study of dynamic systems in discrete and continuous time. The theory uses type-two effectivity as the underlying computational model, but we quickly develop a type system which can be manipulated abstractly, but for which all allowable operations are guaranteed to be computable. We apply the theory to the study of differential inclusions, reachable sets and controllability.

1 Introduction

Dynamic systems which are used to model time-varying processes in almost all fields of science. Such systems are studied by means of computer simulation of a mathematical model, using approximate numerical schemes and inexact floating-point arithmetic. While this is usually sufficient to obtain an understanding of the system evolution, such an approach cannot be used to rigorously analyse the system behaviour. In order to perform a rigorous analysis we can either resort to an algebraic approach, which is usually too inefficient to be of practical use, or try to perform numerical computations with bounds on the errors. This naturally leads to questions concerning the best representation of the data, and what is actually possible to compute.

There are a number of existing approaches to continuous mathematics in which computational issues are considered, from informal treatments in constructive analysis [BB85], through more formal approaches using domain theory [GHK+03] or locale theory [Joh02], to the low-level theory of type-two effectivity [Wei00]. Each of these approaches has its advantages and disadvantages. Constructive analysis is closest to “working mathematics”, and thus is easiest to work with, but the relationship to computation is not explicit. The formal approaches such as domain and locale theory are powerful, but are hard to understand without a background in logic, and are not set up directly as a theory of computation (though the relationship with computation can be formalised). The theory of type-two effectivity provides an explicit connection with digital computation, but has a cumbersome notation which makes it unwieldy to work with for higher-level mathematics.

The aim of this paper is to give an exposition of a theory of computation in analysis, leading to results on computability of the evolution of some important classes of

* This research was supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO) Vidi grant 639.032.408.
dynamic systems. We use type-two effectivity to provide a direct link with real computation, but aim to hide the low-level details in a theory of data types which can then be manipulated in a natural way using constructive mathematics. We then address the concrete problems of computing the finite-time evolution of nondeterministic finite-dimensional systems in both discrete time and continuous time.

The paper is organised as follows. In Section 2 we provide a brief sketch of a topological (non-computable) type theory. In Section 3, we develop a theory of computable types for spaces of points, sets and functions. In Section 4 we apply this type theory to the study of dynamic systems.

2 Type Theory and General Topology

In this section, we review existing results on topological spaces without any explicit computational structure. The main aim is to explore a topological type theory, in which the types are topological spaces.

2.1 Type theory

In a standard type theory, we are most interested in products $X \times Y$ and exponentials $Y^X$. In the category of sets, products to the normal Cartesian product, and exponentials to functions from $X$ to $Y$. The most important operations on these constructions are projection $\pi_i : X_1 \times X_2 \to X_i$, and isomorphism between the type $Y^A \times X$ and the type $(Y^X)^A$ via the bijection $f \mapsto \hat{f}$ defined by $\hat{f}(a)(x) = f(a, x)$. We say a category is Cartesian closed if we can form products and exponentials with the required properties.

2.2 Topological types

In the topological category, we identify the exponential type $Y^X$ with $C(X; Y)$, the space of continuous functions from $X$ to $Y$. It remains to provide a topology on $C(X; Y)$, and it is here that problems arise. For it is well-known that it is not always possible to find a topology on $C(X; Y)$ such that the set of continuous function $A \times X \to Y$ is in bijective correspondence with the set of continuous functions $A \to C(X; Y)$ for all spaces $A$. Such a topology can always be found if the space $X$ is core-compact (see [EH02]), in which case the topology is the well-known Isbell topology.

If $X$ is a Hausdorff space, then core-compactness is equivalent to local compactness. While most spaces which are commonly used as state spaces in dynamical systems theory are locally-compact Hausdorff spaces the space of possible solutions, or trajectory space is a function space, typically $X^N$, $X^Z$ or $C(\mathbb{R}^+; X)$. Such function spaces are decidedly not locally-compact; typically no compact set has an open subset. We therefore need a theory which can handle non-locally-compact spaces.

Fortunately, there are subcategories of the category of topological spaces which are Cartesian closed. Most interesting for us are the quotients of second-countable spaces, sometimes known as the quotients of countably-based (QCB) spaces. It is known that the full category of QCB spaces is Cartesian closed (see [Sch02,ELS04]), so we can
define topological types in this category. Even more promising, these spaces can be given a *computable structure* by means of representations by infinite sequences on some finite alphabet. Further, these spaces include all spaces encountered in the study of finite-dimensional dynamics systems, including Euclidean space, manifolds, continuous functions on these spaces, and hyperspaces of open, closed and compact sets.

### 2.3 Sets and maps in type theory

Starting from (topological) types $X$ and $Y$, we can define products $X \times Y$ and exponentials $Y^X$, the latter corresponding to continuous functions $C(X; Y)$. However, using these operations, we can also define subsets of spaces. The key to this construction is the Sierpinski space $S = \{\top, \bot\}$ with open sets $\{\}, \{\top\}, \{\top, \bot\}$. Here, $\top$ denotes “true”, and $\bot$ denotes “divergent” or “don’t know”, and can be thought of as representing a computation which never halts.

It is easy to see that the set of open subsets of $X$, denoted $O(X)$, is in bijective correspondence with the space of continuous functions $X \rightarrow S$, since a set $U$ is open if, and only if, the function $f_U : X \rightarrow S$ defined by $f_U(x) = \top \iff x \in U$ is continuous. Hence we should identify $O(X)$ and $S^X$ in a topological type theory. The closed subsets $A(X)$ are in bijective correspondence with their complements. In the language of topos theory, we say that $S$ is the *subobject classifier* for the topological category.

Similarly, if $A$ and $C$ are sets, then we can define functions $g_A, h_C : O(X) \rightarrow S$ by $g_A(U) = \top \iff A \cap U \neq \emptyset$ and $h_C(U) = \top \iff C \subseteq U$. We will henceforth write $A \ni U$ for $A \cap U \neq \emptyset$. It is clear that the function $g_A$ satisfies $g_A(U \cup V) = g_A(U) \vee g_A(V)$, and $h_C(U \cap V) = h_C(U) \wedge h_C(V)$. Further, given a “sufficiently reasonable” topology on $O(X)$, the functions $g_A$ and $h_C$ are continuous for $A$ closed and $C$ compact, any such continuous function $g_A$ satisfying the union condition arises in this way from a closed set, and any such continuous function $h_C$ satisfying the intersection condition arises in this way from a compact set. By “sufficiently reasonable”, it suffices that inclusion $x \in U$ is continuous in both $x$ and $U$.

Following [Tay08], we call the space of closed sets with the topology induced by $g_A$ as the space of *overt* sets. We can therefore identify overt sets and compact sets with subtypes of $S(S^X)$.

### 3 Computable Analysis

We now give a computational meaning to our purely topological theory. The theory is based on the theory of type-two effectivity of [Wei00], though the development here is more abstract, similar to that of Schröder [Sch02].

#### 3.1 Machine-computability

We fix a finite alphabet $\Sigma$, and give the space $\Sigma^\omega$ the product topology. In particular, $\Sigma^\omega$ is a second-countable, locally-compact zero-dimensional Hausdorff space. By considering *type-two* Turing machines working on streams of data identified with $\Sigma^\omega$, we
define a set of *machine-computable* partial functions \( \eta : \Sigma^\omega \times \cdots \times \Sigma^\omega \to \Sigma^\omega \). The domain of a machine-computable function is always a \( G_\delta \) set. (Recall that a \( G_\delta \)-set is a countable intersection of open sets.) The set of machine-computable functions is countable, and closed under composition. We also define the uncountable set of *machine-continuous* functions \( \eta : \Sigma^\omega \times \cdots \times \Sigma^\omega \to \Sigma^\omega \) to be the continuous functions with \( G_\delta \)-domain.

In this paper, we do not need to concern ourselves with the exact definition of machine-computable function. However, we will need to use some more abstract results on continuous and computable functions, which can be found in [Wei00, Theorem 2.3.5].

**Theorem 1.**

1. There exists a surjective function \( \delta : \Sigma^\omega \to C(\Sigma^\omega; \Sigma^\omega) \) whose range consist of all machine-continuous partial functions (with \( G_\delta \)-domain) and a machine-computable function \( \varepsilon : \Sigma^\omega \times \Sigma^\omega \to \Sigma^\omega \) such that \( \delta(p)(q) = \varepsilon(p, q) \) for all \( q \).

2. For any computable function \( f : \Sigma^\omega \times \Sigma^\omega \to \Sigma^\omega \), there is a computable function \( s : \Sigma^\omega \to \Sigma^\omega \) such that \( f(p, q) = \varepsilon(s(p), q) \) for all \( p, q \).

The first part means that there is a naming system of machine-continuous functions making function evaluation computable. The second part means that given a computable function of two variables, the operation of binding the first variable is computable.

### 3.2 Computable types

In this section, we show how to define computable structures on mathematical objects, in particular, on topological spaces.

**Definition 1 (Representation).** A representation of a set \( M \) is a partial surjective function \( \delta : \subset \Sigma^\omega \to M \).

Representations \( \delta_1 \) and \( \delta_2 \) of \( M \) are equivalent if there are machine-computable functions \( \eta_1, \eta_2 \) such that \( \delta_1 \circ \eta_1 = \delta_2 \) and \( \delta_2 \circ \eta_2 = \delta_1 \).

A computable type is an equivalence class of pairs \( (M, \delta) \), where \( (M, \delta_1) \) is equivalent to \( (M, \delta_2) \) if \( \delta_1 \) and \( \delta_2 \) are equivalent. We shall usually denote a computable type with underlying set \( M \) by \( \mathcal{M} \).

Note that the domain of a representation is an arbitrary set. Where possible, it is useful to use topologically “nice” sets as domains to help checking whether a sequence is a valid name. The following definition shows how representations induce a computable structure on general sets.

**Definition 2 (Computable function).** Let \( f : M_1 \times \cdots \times M_k \to Y \) be a function, and \( \delta_{M_i} : \subset \Sigma^\omega \to M_i \) and \( \delta_Y : \subset \Sigma^\omega \to Y \) be representations. Then a machine-continuous function \( \eta : \subset (\Sigma^\omega)^k \to \Sigma^\omega \) realises \( f \) if \( f(\delta_{X_1}(p_1), \ldots, \delta_{X_k}(p_k)) = \delta_Y(\eta(p_1, \ldots, p_k)) \) for all \( p_i \in \text{dom}(\delta_i) \). We say \( f \) is computable if it is realised by a machine-computable function.
Recall that a continuous function $\delta : A \rightarrow X$ is a quotient map if $\delta$ is surjective and whenever $\phi : A \rightarrow Y$ is continuous and $\phi(p) = \phi(q)$ whenever $\delta(p) = \delta(q)$, then there exists a map $g : A \rightarrow Y$ such that $\phi = g \circ \delta$. A continuous function $\delta : A \rightarrow X$ is universal, if whenever $\phi : A \rightarrow X$ is continuous, there exists a continuous function $\eta : A \rightarrow A$ such that $\phi = \delta \circ \eta$.

**Definition 3 (Admissible representation).** An representation $\delta$ of a topological space $(X, \tau)$ is admissible if it is a universal quotient map.

We require an admissible representation of a topological space to be a universal so that it captures all the topological information about a space. We require it to be a quotient map so that continuity on $(X, \tau)$ is reflected in $\Sigma^\omega$. The standard representations of a second-countable Kolmogorov ($T_0$) space as defined in [Wei00, Chapter 3] are always admissible in the above sense. The following lemma (see [Sch02]) shows that we can extract the topology of $X$ from an admissible representation $\delta$.

**Lemma 1.** There is at most one topology $\tau$ on $X$ making $\delta : \subset \Sigma^\omega \rightarrow X$ an admissible representation of $(X, \tau)$.

In general there are many non-equivalent representations admissible for a given topological space. Not all spaces have admissible representations with “nice” names; there is no admissible representation of $C(C(\mathbb{R}; \mathbb{R}); \mathbb{R})$ with a $G_\delta$ domain.

Up to equivalence there is only one admissible representation on $\mathbb{R}$ making arithmetic and comparison computable, so we can talk of a canonical type $\mathcal{R}$ of real numbers [Bau00, Theorem 5.5.18]; see also [Bra98]. Most other types used in analysis can be built from $\mathcal{R}$ in a natural way.

**Theorem 2.** Let $\delta_X$, $\delta_Y$ be admissible representations of spaces $X$ and $Y$.

1. If $f : X \rightarrow Y$ be continuous, then there exists a machine-continuous function $\eta$ such that $\text{dom}(\eta) \supset \text{dom}(\delta_X)$ and $\delta_Y \circ \eta = f \circ \delta_X$ on $\text{dom}(\delta)$.

2. Suppose $f : X \rightarrow Y$, and there exists a machine-continuous function $\eta$ with $\text{dom}(\eta) \supset \text{dom}(\delta)$ such that $f \circ \delta_X = \delta_Y \circ \eta$ on $\text{dom}(\delta)$. Then $f$ is continuous.

The first part of the theorem says that any continuous function is realised by a machine-continuous function. The second part that any function realised by a machine-continuous function (in particular, by a machine-computable function) is continuous.

### 3.3 Types of continuous functions

We now give the main results of computability of operations on continuous functions. The next result asserts that function evaluation is computable.

**Theorem 3.** Let $X$ and $Y$ be spaces with admissible representations $\delta_X$ and $\delta_Y$. Then there is a canonical representation $\delta_{[X \rightarrow Y]}$ of $C(X; Y)$ such that evaluation $C(X; Y) \times X \rightarrow Y$ is computable. This representation defines a canonical type $C(X; Y)$.

The next theorem asserts computability of the fundamental operations on types.
Theorem 4. Let $W$, $X$ and $Y$ be spaces with admissible representations $\delta_W$, $\delta_X$ and $\delta_Y$ respectively. Let $\mathcal{W}$, $\mathcal{X}$ and $\mathcal{Y}$ denote the types of $(W, \delta_W)$, $(X, \delta_X)$ and $(Y, \delta_Y)$. Then there is a computable bijection between the types $\mathcal{Y}^{\mathcal{W} \times \mathcal{X}}$ and $\mathcal{Y}^{\mathcal{X} \times \mathcal{W}}$ taking $f$ to $\hat{f}$.

The following theorem shows that if we can effectively evaluate a function $f : X \to Y$, then we can compute a name. We will repeatedly use this result to prove computability of a function type. The proof essentially follows from Theorem 1.

Theorem 5. Suppose $\mathcal{X}$ and $\mathcal{Y}$ are computable types, and $f : X \to Y$. Then if we can effectively evaluate $f(x)$ for all $x \in X$, we can effectively compute $f$ in $C(\mathcal{X}; \mathcal{Y})$.

3.4 Point and set types

We now use the computable function types to derive the set types we need to study systems.

Definition 4 (Types of open, closed and compact sets).

1. We identify the type of open sets, denoted $\mathcal{O}(X)$ with the space of continuous functions $\mathcal{X} \to \mathcal{S}$ by $f_U(x) = T \iff x \in U$.
2. We identify the type of closed sets, denoted $\mathcal{A}(X)$ with the space of continuous functions $\mathcal{X} \to \mathcal{S}$ by $f_A(x) = T \iff x \notin U$.
3. We identify the type of overt sets, denoted $\mathcal{V}(X)$ with the space of continuous functions $f_A : \mathcal{O}(X) \to \mathcal{S}$ satisfying $f_A(U \cup V) = f_A(U) \lor f_A(V)$.
4. We identify the type of compact sets, denoted $\mathcal{K}(X)$ with the space of continuous functions $f_C : \mathcal{O}(X) \to \mathcal{S}$ satisfying $f_C(U \cap V) = f_C(U) \land f_C(V)$.

Concrete representations for these types in the case of Euclidean/metric spaces are given in [BW99,BP03]. Most of the basic set-theoretic operations, including union and intersection, are computable without additional assumptions on the space $X$. However, we will sometimes need to work in Hausdorff spaces in which the apartness relation is computable.

Definition 5. We say $\mathcal{X}$ is a effectively separated if the function $s : \mathcal{X} \times \mathcal{X} \to \mathcal{S}$ defined by $s(x, y) = T \iff x \neq y$ is computable. We say $\mathcal{X}$ is a effectively separable if there is a computable function $r : \mathcal{N} \to \mathcal{X}$ whose range is dense in $X$.

Note that although any singleton set in a $T_1$ space is closed, singleton function is only continuous in a Hausdorff ($T_2$) space.

We have the following computability results on types. Note that having developed the basic theory, and shown that computing a function type is equivalent to being able to evaluate it, the rest of the theory is almost trivial.

Theorem 6. The following operations on sets are computable:

1. Finite intersection $\mathcal{O} \times \mathcal{O} \to \mathcal{O}$ and arbitrary union $\mathcal{O}^\mathcal{N} \to \mathcal{O}$.
2. Finite union $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$ and arbitrary intersection $\mathcal{A}^\mathcal{N} \to \mathcal{A}$.
3. Complement $\mathcal{O} \to \mathcal{A}$ and $\mathcal{A} \to \mathcal{O}$.
4. Countable union $\mathcal{V}^\mathcal{N} \to \mathcal{V}$ and finite union $\mathcal{K} \times \mathcal{K} \to \mathcal{K}$.
5. Finite intersection $\mathcal{V} \times \mathcal{O} \to \mathcal{V}$ and $\mathcal{K} \times \mathcal{A} \to \mathcal{K}$.
6. Singleton $\mathcal{X} \to \mathcal{V}$, $\mathcal{X} \to \mathcal{K}$.
7. Closure $O \to V$ if $X$ is effectively separable, and identity $K \to A$ if $X$ is effectively separated.

8. Preimage $C(X; Y) \times O(Y) \to O(X)$.

9. Image $C(X; Y) \times V(X) \to V(Y)$ and $C(X; Y) \times K(X) \to K(Y)$.

Due to the strong conditions on admissible representations, and application of Theorem 5, the proofs are almost deceptively straightforward.

4 Computation for Dynamic Systems

We now use the computable type theory developed in the previous section to give some results on computable properties of dynamic systems.

4.1 Spaces of multifunctions

When considering solutions of nondeterministic systems, we are often interested in function spaces with set-valued types, or hyperspaces of functions.

The set of solutions of a dynamic system is the space of continuous functions $\xi : T \to X$, where $T$ is the time domain, and $X$ is the state space. For an autonomous system, we require time-invariance, that if $\xi$ is a solution and $s \in T$, then the function defined by $\eta(t) = \xi(t + s)$ is also a solution. We also require the property of state, that if $\xi$ and $\eta$ are solutions with $\xi(s) = \eta(s)$, then there is a solution $\zeta$ with $\zeta(t) = \xi(t)$ for $t \leq s$, and $\zeta(t) = \eta(t)$ for $t \geq s$.

For a deterministic system, there is only one trajectory through a given initial state. The solution space may be represented either as a function $\phi : X \times T \to X$, or as a function $\hat{\phi} : X \to C(T; X)$. By the exponentiation property, these representations are equivalent.

In a nondeterministic system there may be many different trajectories with the same initial state. If the time domain is $\mathbb{R}$, we call the resulting dynamics a multiflow. In this case, there are many different ways of representing the solution space. The simplest way of representing the solution space is as the behaviour of the system, which is simply the set of all solutions, $\Phi \in \mathcal{P}(C(T; X))$. However, we can also represent the solution space as a function $\hat{\Phi} : X \to \mathcal{P}(C(T; X))$ such that $\xi(0) = x$ for all $\xi \in \Phi(x)$. Another useful representation is in terms of the finite reachability operator, $\hat{\Phi} : X \times T \to \mathcal{P}(X)$.

We shall see that while $\Phi$ and $\hat{\Phi}$ are classically equivalent, in a computational setting $\Phi$ may be hard to define and contains less information. Further, $\Phi$ and $\hat{\Phi}$ are classically and computationally equivalent for compactly-generated systems, but not for others.

The following result gives relationships between multiflow representations:

**Lemma 2.** The function $B : K(X) \times O(Y) \to O(C(X; Y))$ defined by $B(K, V) = \{ f \mid f(K) \subset V \}$ is computable, as is the function $B : K(X) \times A(Y) \to A(C(X; Y))$ defined by $B(K, A) = \{ f \mid f(K) \subset A \}$.

We now consider multiflows taking values in the class of overt and compact sets.
Lemma 3.

1. The types $\hat{\Phi} : \mathcal{X} \to \mathcal{K}(\mathcal{T} ; \mathcal{X})$ satisfying $\xi(0) = x$ for all $\xi \in \hat{\Phi}(x)$, and the types $\hat{\Phi} : \mathcal{X} \times \mathcal{T} \to \mathcal{K}(\mathcal{X})$ are equivalent. We can also compute $\hat{\Phi}$ from $\Phi$, and $\Phi$ from $\hat{\Phi}$ if $X$ is compact.
2. If $\mathcal{T} = \mathbb{R}$, the type $\hat{\Phi} : \mathcal{X} \to \mathcal{V}(\mathcal{T} ; \mathcal{X})$ is strictly stronger than both $\Phi \in \mathcal{V}(\mathcal{T} ; \mathcal{X})$ and $\hat{\Phi} : \mathcal{X} \times \mathcal{T} \to \mathcal{V}(\mathcal{X})$. The latter are incomparable.

The proofs of the above results are straightforward.

4.2 Computability theory for multivalued maps

A multivalued map is a function $F : X \to \mathcal{P}(Y)$. For a set $A \subset X$, we define $F(A) = \bigcup \{ F(x) \mid x \in A \}$. For $B \subset Y$, we define $F^{-1}(B) = \{ x \in X \mid F(x) \cap B \neq \emptyset \}$ and $F^{-1}(B) = \{ x \in X \mid F(x) \subset B \}$. Note that $F^{-1}(B) = X \setminus F^{-1}(Y \setminus B)$.

For $F : X \to \mathcal{P}(Y)$ and $G : Y \to \mathcal{P}(Z)$, we define $G \circ F : X \to \mathcal{P}(Z)$ by $G \circ F(x) = G(F(x))$. We say that $F$ is lower-semicontinuous if $F^{-1}(V)$ is open whenever $V$ is open, and upper-semicontinuous if $F^{-1}(B)$ is closed whenever $B$ is closed.

We are interested in the case that $F$ is a continuous function from $X$ to a hyperspace of subsets of $Y$; in particular, for $F : X \to \mathcal{V}(Y)$ and $F : X \to \mathcal{K}(Y)$. In this case, we have the following properties. The proof is in [Col05].

Theorem 7. The following types are computably equivalent:

1. $F : \mathcal{X} \to \mathcal{V}(\mathcal{Y})$, $F^{-1} : \mathcal{O}(\mathcal{Y}) \to \mathcal{O}(\mathcal{X})$ and $F : \mathcal{V}(\mathcal{X}) \to \mathcal{V}(\mathcal{Y})$.
2. $F : \mathcal{X} \to \mathcal{K}(\mathcal{Y})$, $F^{-1} : \mathcal{A}(\mathcal{Y}) \to \mathcal{A}(\mathcal{X})$ and $F : \mathcal{K}(\mathcal{X}) \to \mathcal{K}(\mathcal{Y})$.

We can therefore compute the forward-time evolution of discrete-time multivalued systems. We denote the type of non-negative integers by $\mathcal{N}$.

Corollary 1. The behaviour $\hat{\Phi}$ of a discrete-time system $F$ is such that:

1. If $F : \mathcal{X} \to \mathcal{X}$, then $\hat{\Phi} : \mathcal{X} \to \mathcal{C}(\mathcal{N}, \mathcal{X})$ is computable from $F$.
2. If $F : \mathcal{X} \to \mathcal{V}(\mathcal{X})$, then $\hat{\Phi} : \mathcal{X} \to \mathcal{V}(\mathcal{C}(\mathcal{N}, \mathcal{X}))$ is computable from $F$.
3. If $F : \mathcal{X} \to \mathcal{K}(\mathcal{X})$, then $\hat{\Phi} : \mathcal{X} \to \mathcal{K}(\mathcal{C}(\mathcal{N}, \mathcal{X}))$ is computable from $F$.

4.3 Computability theory for differential systems

In this section we consider the computability of systems defined by differential equations or differential inclusions. For simplicity, we assume that $X$ is a Euclidean space $\mathbb{R}^n$, though these results also extend to differential manifolds and locally-compact Banach spaces. We state the results in this section without proof, as we need to go back to first principles to solve the differential systems. In particular, we need to resort to the classical Arzela-Ascoli theorem to assert the existence of solutions. We denote the type of real numbers by $\mathcal{R}$.

Theorem 8. Let $f : X \to X$ be locally-Lipschitz continuous. Then the solution operator of $\dot{x} = f(x)$ is computable $\mathcal{C}(\mathcal{X} ; \mathcal{X}) \times \mathcal{X} \to \mathcal{C}(\mathcal{R}, \mathcal{X})$. 
The proof is essentially standard [DM70], though is too long to include here. A simple proof can be found in [CG08]. Note that we can weaken the locally-Lipschitz condition to simply requiring uniqueness of solutions [Ruo96].

We now turn to nondeterministic differential systems as defined by differential inclusions \( \dot{x} \in F(x) \). For an introduction to differential inclusions, see [AC84]. Following the well-known Filippov solution concept, we may first need to compute the convex hull of the right-hand side.

**Lemma 4.** Closed convex hull is a computable operator \( V \to V \) and \( K \to K \).

We can now state the main theorems on computability of solutions of differential inclusions. The continuous case was first proved in [PVB96], but easily splits into the lower- and upper-semicontinuous cases. The one-sided Lipschitz condition was developed in [Gab07].

**Theorem 9.**
1. Let \( F \) be one-side locally-Lipschitz closed-convex-valued lower-semicontinuous. Then the solution operator of \( \dot{x} \in F(x) \) is computable \( C(X;\mathcal{V}(\mathcal{X})) \times \mathcal{X} \to \mathcal{V}(C(R,\mathcal{X})) \).
2. Let \( F \) be compact-convex-valued upper-semicontinuous. Then the solution operator of \( \dot{x} \in F(x) \) is computable \( C(X;\mathcal{K}(\mathcal{X})) \times \mathcal{X} \to \mathcal{K}(C(R,\mathcal{X})) \).

### 4.4 Computability theory for infinite-time properties

We now apply the results of Section 4.2 to prove computability of some infinite-time operators in discrete-time dynamical systems. The results can be found in [Col05,Col07]. We will need the following result, which shows that we can separate compact and closed sets.

**Lemma 5.** There is a recursively enumerable set \( D \) of pairs \( (A,B) \in \mathcal{O} \times \mathcal{K} \) such that \( A \subseteq B \) such that for any compact \( K \) and open \( U \), there exist \( (A_i,B_i) \) such that \( K \subseteq A_i \) and \( B_i \subseteq U \).

We define the **reachable set** of a system \( F \) with initial state set \( X_0 \) as

\[
\text{reach}(F,X_0) = \{ x \in X \mid \exists \text{ solution } \xi \text{ and } t \in T \text{ with } \xi(0) \in X_0 \text{ and } \xi(t) = x \}.
\]

**Theorem 10.** The reachable set operator \( \text{reach} \) is computable as a function \( C(X;\mathcal{V}(\mathcal{X})) \times \mathcal{V}(\mathcal{X}) \to \mathcal{V}(\mathcal{X}) \), but not as a function \( C(X;\mathcal{K}(\mathcal{X})) \times \mathcal{K}(\mathcal{X}) \to \mathcal{K}(\mathcal{X}) \).

**Proof.** We can write \( \text{reach}(F,X_0) = \bigcup_{i=0}^{\infty} R_i \), where \( R_0 = X_0 \) and \( R_{i+1} = R_i \cup F(R_i) \), proving computability. To prove uncomputability, consider the system \( f : \mathbb{R} \to \mathbb{R} \) defined by \( f_\epsilon(x) = \epsilon + x + x^2 - x^4 \). Then \( \text{reach}(f_0,\{-1/2\}) \subset [-1,0] \), but \( \text{reach}(f_\epsilon,\{-1/2\}) \not\subset [-1,1/2] \) for any \( \epsilon > 0 \).

We define the **chain-reachable set** of \( F \) as limit of all \( \epsilon \)-orbits, or equivalently as

\[
\text{chain reach}(F,X_0) = \bigcap \{ U \in \mathcal{O}(X) | \text{cl}(U) \text{ is compact, and } X_0 \cup F(\text{cl}(U)) \subset U \}.
\]

**Theorem 11.** If \( \text{chain reach}(F,X_0) \) is bounded, then \( \text{chain reach} : C(X;\mathcal{K}(\mathcal{X})) \times \mathcal{K}(\mathcal{X}) \to \mathcal{K}(\mathcal{X}) \) is computable, and is the optimal \( \mathcal{K}(\mathcal{X}) \)-computable over-approximation to reach.

**Proof.** It is clear that \( \text{chain reach}(F,X_0) = \bigcap \{ A_i | (A_i,B_i) \in D \text{ and } X_0 \cup F(B_i) \subset A_i \} \), proving computability. The proof of optimality involves considering perturbations.
4.5 Computability theory for control systems

A noisy control system with state space $X$, input space $U$ and noise space $V$ is a function $f : X \times U \times V \to X$. We assume that $U$ is an overt space and $V$ a compact space, and define $F_U : X \to X \times U$, $F_U(x) = \{(x, u) \mid u \in U\}$, and $F_V : X \times U \to X$ by $F_V(x, u) = \{f(x, u, v) \mid v \in V\}$. The controllable set $\text{ctrl}(f, T, S)$ of $f$ with target set $T$ and safe set $S$ is determined recursively by $T_0 = T \cap S$ and $T_{i+1} = T_i \cup \{x \in X \mid \exists u \in U, \forall v \in V, f(x, u, v) \in T_i\} \cap S$. The following result was proved in [Col08]:

**Theorem 12.** The controllable set operator $\text{ctrl} : \mathcal{C}(\mathcal{X}, \mathcal{U}, \mathcal{V}; \mathcal{X}) \times \mathcal{O}(\mathcal{X}) \times \mathcal{O}(\mathcal{X}) \to \mathcal{O}(\mathcal{X})$ is computable.

**Proof.** The multivalued functions $F_U : X \to \mathcal{V}(X \times U)$ and $F_V : (X \times U) \to \mathcal{K}(X)$ can be computed from $f$, $U$ and $V$. Write $T_{i+1} = T_i \cup (F_U^{-1}(F_V^{-1}(T_i)) \cap S)$ and $C = \bigcup_{i=0}^{\infty} T_i$.

5 Conclusions

In this paper, we have developed a computable type theory sufficient for allowing the analysis of dynamic systems. Further extensions involve to infinite-dimensional systems and stochastic systems, which require additional work to prove computability of the evolution. Additional work is also required to understand the function space topologies involved, especially the topology on $O(C(X; Y))$.

Acknowledgement The author would like to thank Martín Escardó for useful advice and suggestions regarding function space topologies, and the anonymous referees.

References


Computable CTL for Discrete-Time and Continuous-Space Dynamic Systems *

Pieter Collins and Ivan S. Zapreev

Centrum Wiskunde & Informatica,
Science Park 123, 1098 XG Amsterdam
Pieter.Collins@cwi.nl, I.Zapreev@cwi.nl

Abstract. Dynamic systems are widely applied for modelling and analysis in physiology, biology, chemistry and engineering. The high-profile and safety-critical nature of such applications has resulted in a large amount of work on formal methods for dynamic systems: mathematical logics, computational methods, formal verification, and etc. In our work, we focus on the verification approach called model checking, and its computability aspects. In this approach, a desired system property, specified using some logical formalism, is verified against the dynamic-system model via an exhaustive state-space exploration. This process typically involves computation of reachable and/or chain-reachable sets that in certain cases can not be obtained due to the continuity of state-space domain. Therefore, in this paper, we use topological approach along with the computability results of Type Two theory of Effectivity in order to construct a computable CTL semantics for discrete-time and continuous-space dynamic systems.

Keywords: Computability, Model Checking, CTL, Dynamic Systems

1 Introduction

A dynamic system describes the system-state evolution over time. Sometimes the system has inputs and/or outputs and then it is called a control system. The inputs may represent either uncontrollable disturbances or control signals, and the outputs may represent quantities that either need to be measured or controlled. Dynamic systems are often used in modelling and analysis of processes in physiology, biology, chemistry and engineering. The commonly known application fields are air traffic control, automated manufacturing, chemical process control, and etc. Verification of performance requirements for dynamic systems is essential for designing systems, such as power plants and robots, and analysing natural phenomena, such as chemical reactions and biological processes.

Given a formal model $M$ of a system design, along with a specification formula $\phi$ that represents a desired system property, formal verification answers the question: “Does the model $M$ satisfy the property $\phi$?”. This question is typically put as a formula $M \models \phi$, that uses a satisfiability relation $\models$.

* This research was supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO) Vidi grant 639.032.408.
The system model $\mathcal{M}$, representing the dynamic system, can be described using various formalisms, such as time-automaton, hybrid automaton, differential equation, differential inclusion and others. The system property $\phi$ is typically described using a logical formalism, some sort of temporal or modal logic, such as $LTL$ [13], $CTL$ [4], $CTL^*$ [9], propositional modal/temporal $\mu$-calculus [10]. There are two main methods for the verification of modal or temporal logic properties, namely: model-checking algorithms, and deductive proof systems. The latter ones are Hilbert-style (axiomatic), tableaux, or Gentzen-style proof systems, whereas the former ones are algorithms for exhaustive state-space exploration. The focus of this paper will be on model checking.

As assumed by its name, except for the system property, model-checking required a formal model of the system. In case of dynamic systems, the model includes an evolution function, that defines how the system progresses over time, the domain on which the system operates (the state-space), and boundary conditions that provide the system’s initial states. Typical system properties, that need to be verified, are reachability – “Does the system reach the certain set of goal states?” – and repeated reachability – “Does the system return to the set of goal states infinitely often?” The model-checking algorithms, for verifying such properties, require computation of system’s reachable states (images or pre-images of sets of states under the system’s evolution function), as well as computation of union, intersection of sets, and testing them for inclusions.

Unfortunately, the state-spaces of dynamic systems are typically not only infinite, but also continuous (e.g. $\mathbb{R}^n$) or hybrid (e.g. a product of $\mathbb{R}^n$ and $\mathbb{Z}$). At the same time, for model-checking algorithms to be sound, they have to be in some sense computable, i.e. effectively implementable on digital computers. These requirements, respectively, rise two problems. The first is that the ordinary computability and complexity theory is not powerful enough to express the computability of real-valued functions and therefore sets of any continuous or hybrid domain. The second is that digital computers do not allow for exact computations on continuous domains (e.g. a subset of $\mathbb{R}$). This is why there is a need for approximate algorithms that can guarantee arbitrarily high levels of accuracy. A natural solutions to both of these problems are offered by Type-2 Theory of Effectivity (TTE) [15], which defines computability based on Turing machines with finite and infinite input/output sequences. TTE has been already applied to analysis of computability of reachable sets of control systems in [6].

The aim of this paper is to provide a “computable” logic for model-checking of discrete-time and continuous-space dynamic systems (DTCSDDs). This should be seen as a first step towards a computable logic for general hybrid systems. Our results are based on TTE and the way we approach the problem is divided into the following steps. First, we overview several modal logics for hybrid systems and choose $CTL$ as the logic to work with. Second, since TTE hardly relies on topologies, we analyse the original $CTL$ semantics and alter it to become computable (topological) for the case of DTCSDDs models.

The rest of the paper is organised as follows. The preliminary material is provided in Section 2. Since TTE hardly relies on topological spaces, we begin
our discussion by recalling some of the important aspects of topological theory. Then, since DTCSDS models are typically expressed by multivalued-maps, we discuss continuity aspects of the latter ones and provide some of their properties. Further, we talk about TTE and computability of various sets/functions. After that, we conclude the section by an overview of various available logics for hybrid systems. In Section 3 we construct a computable semantics of CTL on DTCSDS models. This is done by considering the standard CTL semantics and then altering it in such a way that its model-checking becomes computable. The latter sketches the approximate model-checking algorithms of CTL on DTCSDSs that can be implemented with the use of digital computers. Section 4 concludes.

2 Preliminaries

2.1 Topological Spaces

Let us recall several important notions and results from the topological theory. A topological space is a pair $T = (X, \tau)$ where $X$ is an arbitrary set and $\tau \subseteq 2^X$ is such that: $\emptyset, X \in \tau$, $\forall U_1, U_2 \in \tau \Rightarrow U_1 \cap U_2 \in \tau$, and $\forall U \subseteq \tau \Rightarrow \bigcup_{U \in \tau} U \in \tau$. For a topological space $T$, elements of $\tau$ are called open and their complements in $X$ are called closed. Let $x \in X$ and $B \subseteq X$ then $B$ is a neighbourhood of point $x$ if there exists an open set $U \in \tau$ such that $x \in U \subseteq B$. Let $B \subseteq X$ and $U \subseteq \tau$ then $U$ is an open cover of $B$ if $B \subseteq \bigcup_{U \in \tau} U$. Let $S \subseteq X$, then the set $\text{Int} (S) = \bigcup \{U | U \subseteq S \land U \in \tau\}$ is called the interior of $S$ and $\text{Cl} (S) = \cap \{A | S \subseteq A \land A \text{ is closed}\}$ is called the closure of $S$. A set $C \subseteq X$ is compact iff every open cover of $C$ has a finite sub cover. A subset of $X$ is pre-compact iff its closure is compact. For a topological space we have $\mathcal{O} –$ a set of open, $\mathcal{A} –$ a set of closed, and $\mathcal{K} –$ a set of compact sets.

Let $T = (X, \tau)$ be a topological space. Then $\beta \subseteq \tau$ is a base of the topology $\tau$ if every element of $\tau$ can be represented as a union of elements from $\beta$. A topological space is called second countable if its topology has a countable base. A Hausdorff space ($T_2$ space) is a topological space such that $\forall x, y \in X$ where $x \neq y$ there exist $U_x, U_y \in \tau$ such that $x \in U_x$, $y \in U_y$ and $U_x \cap U_y = \emptyset$. A Hausdorff space is called locally compact if every $x \in X$ has a compact neighbourhood.

2.2 DTCSDSs and Multivalued Maps

In our work we consider discrete-time continuous-space dynamic systems, which means that the state space of the system is continuous and the time domain is discrete (the system state changes at discrete time points). In system theory, dynamic systems are given by functions $f : X \times U \to X$, where $X$ is the state space, and $U$ can either represent control or system noise. These functions are typically converted into multivalued maps $F : X \Rightarrow X$ such that $F(x) = f(x, U)$.

A multivalued map $F : X \Rightarrow Y$, also known as multivalued function or multifunction, is a total relation on $X \times Y$. If we define $F (S) = \{F(x) | x \in S\}$ for $S \subseteq X$ then $F$ can be seen as a function $F : X \to 2^Y$. This last definition is more convenient when we want to talk about function composition. For example, for two
multivalued maps $F : X \to Y$, $G : Y \to Z$ and their composition $G \circ F$ we have $G \circ F : X \to Z$ and thus for any $x \in X$ we can simply write $G \circ F(x) = G(F(x))$.

A weak preimage of $F$ on $B \subseteq Y$ is $F^{-1}(B) = \{ x \in X : F(x) \cap B \neq \emptyset \}$ and a strong preimage is $F^\equiv(B) = \{ x \in X : F(x) \subseteq B \}$. The notion of continuity for multivalued maps is an extension of continuity for the regular functions. Let us only note that, for a continuous multivalued map $F$ and an open set $U \subseteq Y$ the preimages $F^{-1}(U)$ and $F^\equiv(U)$ are open sets. The space of continuous multivalued maps $F : X \to 2^Y$ will be denoted as $C(X,Y)$.

### 2.3 Type-2 Theory of Effectivity (TTE)

TTE [15], as well are regular computability theory, is based on Turing machines. The difference is that TTE (type-2) machines allow for infinite computations. In particular they can accept infinite inputs and produce infinite outputs. The computability is first defined on type-2 machines and then is extended to arbitrary functions, sets and their elements by means of notations and representations.

Let $M$ be a type-2 machine with a fixed finite alphabet $\Sigma$, $k \geq 0$ input tapes, one output tape and $Y_i \in \{ \Sigma^*, \Sigma^\omega \}$ where $i \in \{0, \ldots, k \}$. Then, a (partial) string function $f_M : Y_1 \times \ldots \times Y_k \to Y_0$ is computable iff it is realised by a type-2 machine $M$. The latter means that for $y_i \in Y_i$ we have $f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^*$ iff $M$ halts on input $(y_1, \ldots, y_k)$ with $y_0$ on the output tape and $f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^\omega$ iff $M$ computes forever on input $(y_1, \ldots, y_k)$ and writes $y_0$ to the output.

The computability on $\Sigma^*$ and $\Sigma^\omega$ is generalised by means of notations and representations. A notation of set $X$ is a partial surjective function $\nu : \Sigma^* \to X$ and a representation is a partial surjective function $\delta : \Sigma^\omega \to X$. These functions encode elements of the domain $X$ into strings and sequences.

A computable Hausdorff space is a tuple $(X, \tau, \beta, \nu)$ such that $(X, \tau)$ is a second-countable locally-compact Hausdorff $(T_2)$ space; $\beta$ is a countable base of $\tau$ consisting of pre-compact open sets; $\nu$ is a notation of $\beta$; we take effectivity properties in [3] (Lemma 2.3) as axioms; and assume that $C_1 : \beta \to K$ is computable. Let us have a computable Hausdorff space and the Sierpinski space $S$.

Then, the elements of $\mathcal{O}$, $\mathcal{A}$, and $\mathcal{K}$ have so called canonical and equivalent to them (admissible) representations that are needed for the computability results below. Also, let us assume that $F^{-1}(U)$ and $F^\equiv(U)$ are computable for $U \in \tau$ and $F \in C(X,X)$. If all of the above conditions are met then the following operations are computable (continuous): countable union as $\mathcal{O} \times \mathcal{O} \to \mathcal{O}$, complement as $\mathcal{O} \to \mathcal{A}$, subset operation as $\mathcal{K} \times \mathcal{O} \to \mathcal{S}$, the $F^{-1}(\cdot)$ and $F^\equiv(\cdot)$ as $\mathcal{O} \to \mathcal{O}$. The following operations are known to be uncomputable: closure as $\mathcal{O} \to \mathcal{A}$, interior as $\mathcal{A} \to \mathcal{O}$.

### 2.4 Modal Logics

The most commonly used branching and linear temporal logics are $CTL$ [4] and $LTL$ [13]. These logics are incomparable, see [5], and can be united to form a more powerful logic known as $CTL^*$. All of these logics are typically interpreted in terms of Kripke structures. According to [8], temporal extensions of these
Logics are often used to represent properties of hybrid systems. For $CTL$, this includes Timed $CTL$ (TCTL), Integrator $CTL$ (ICTL) and Timed $\mu$-calculus ($TC\mu$). An extension of $LTL$ is, e.g., Metric Temporal Logic ($MTL$).

Topological aspects play an essential role in model checking of hybrid systems. Therefore, having a topological interpretation of logical formulae is very important. Logics that provide topological semantics are called topological logics and deal with topological spaces rather than with Kripke structures. One of the topological logics is a branching-time logic called Dynamical Topological Logic ($DTL$) [11]. A distinguishing feature of this logic is that it contains one topological modality, namely the interior operator. Another topological logic is discussed in [1]. Since the authors claim that modal logics are unable to deal with continuous or mixed (hybrid) dynamical systems, they provide a topological interpretation of a propositional logic with universal and existential quantifiers. Here, atomic propositions are open sets of states and each formulae must result in an open set. This, unlike in [11], requires an altered interpretation of negations, i.e. they result in the interior for the set complement.

Similar to [1], we will require formulae to result in open sets on states. Thus, the $DTL$ semantics is not particularly suitable for us. There are two reasons for that (we assume $f$-continuous and $A$-open). First, the henceforth operator does not result in an open set, because the countable intersection of open sets is not necessarily open. The authors remedy this situation by using Alexandrov spaces, but this limitation rules out all cases of particular interest. Second, the negation operator does not necessarily result in an open set, since a complement of an open set is closed (clopen sets are exceptions). Also the interior operator is not of a major concern for us, since we restrict our interest to atomic propositions given by open sets. For the rest, $DTL$ is a subset of $CTL$. $LTL$ ($CTL^*$) requires evaluation of a formula on each system path and thus assumes reasoning about sets of paths. Since paths are infinite, this requires us to use product topology and to reason about open sets of paths, rather than states. This complicates matters a little, and therefore for this paper we decide to choose $CTL$ that is easier to provide computable semantics for (avoiding reasoning about paths).

### 3 Computable $CTL$ model checking

As it was mentioned in Section 2.4, $CTL$ is the logic we choose to provide a computable semantics for. Our choice is motivated by a relative simplicity of its original semantics (there are no nested path formulae), and a reasonably reach variety of operators ($CTL$ and $LTL$ have a non-trivial common fragment [12]). Below, we consider $CTL$ formulae and argue about which of them are computable and which are not. We also state some desired properties of the set of initial states. Along the way, we construct the computable semantics of $CTL$ that implies corresponding model-checking algorithms.

In Section 2.2 we introduced discrete-time continuous-state control systems. Before we proceed, let us extend the definition of a DTCSDS given before.
Definition 1. A discrete-time continuous-space control systems (DTCSDSs) is a tuple $M = (T, F, L)$ where: $T = (X, \tau, \beta, \nu)$ is a computable Hausdorff space; $F \in C(X, X)$ is a multivalued map which defines the system’s evolution; and $L : X \rightarrow 2^{AP}$ is a labelling function where AP is a finite subset of $\tau$. For any $x \in X$ we have $L(x) = \{ U \in AP | x \in U \}$.

Consider a DTCSDS model $M = (T, F, L)$ with a set of initial states $I \subseteq X$. To derive a computable CTL semantics we should define a computable meaning of $M, I \models \Phi$ (when obvious, we will omit $M$), i.e. the fact that the model $M$ satisfies $\Phi \in CTL$ for all states in $I$. From now on, we assume that $F$ is such that taking $F^{-1}(U)$ and $F^= (U)$ for $U \in \tau$ are computable operations.

Below, in Section 3.1, we first discuss computability of CTL formulae in the original semantics and devise a computable one. Further, Section 3.2 discusses how the modified semantics influences CTL model checking.

3.1 Computable semantics for CTL

CTL is typically interpreted over Kripke structures. A Kripke structure $M$ is a tuple $(S, I, R, L)$ where $S$ is a countable set of states; $I \subseteq S$ is a set of initial states; $R \subseteq S \times S$ is a transition relation such that $\forall s \in S \exists s' \in S : (s, s') \in R$; $AP$ is a finite set of atomic propositions; and $L : S \rightarrow 2^{AP}$ is an labelling (interpretation) function on $S$. A path in a Kripke structure $M$ is an infinite sequence of states $s_0s_1s_2 \ldots$ such that $\forall i \geq 0 : (s_i, s_{i+1}) \in R$. A set of paths starting in state $s$ is denotes as $Paths(s)$. For the same state $s$ there is an infinite computation tree, obtained by unfolding the Kripke structure, with the root $s$, such that $(s', s'')$ is an arc in the tree if $(s', s'') \in R$.

$CTL$ has a syntax that is divided in to state formulae: $\Phi ::= p \mid \neg \Phi \mid \Phi \land \Phi \mid \forall \Phi \mid \exists \Phi$, and path formulae: $\phi ::= X \Phi \mid \Phi U \Phi \mid \Phi R \Phi$. The state formulae have the following semantics: $s \models p$ iff $p \in L(s)$; $s \models \neg \Phi$ iff $(s \not\models \Phi)$; $s \models \Phi \land \Psi$ iff $(s \models \Phi) \land (s \models \Psi)$; $s \models \exists \phi$ iff $\exists \sigma \in Paths(s) : \sigma \models \phi$; $s \models \forall \phi$ iff $\forall \sigma \in Paths(s) : \sigma \models \phi$. The semantics of path formulae is as follows: $\sigma \models X \Phi$ iff $\sigma[1] \models \Phi$; $\sigma \models \Phi U \Psi$ iff $\exists j \geq 0 : (\sigma[j] \models \Psi \land \forall 0 \leq i < j : \sigma[i] \not\models \Phi)$; $\sigma \models \Psi R \Phi$ iff $(\forall i \geq 0 : \sigma[i] \models \Phi) \lor (\exists j \geq 0 : (\sigma[j] \models \Psi \land \forall 0 \leq i < j : \sigma[i] \not\models \Phi))$. In $CTL$, path formulae can only be used as sub formulae of state formulae. For $\Phi \in CTL$ let $Sat(\Phi)$ be a set of states satisfying $\Phi$ on some model $M$. If we can compute $Sat(\Phi)$, then for a given set of initial states $I$, verification of $M, I \models \Phi$ requires verifying $I \subseteq Sat(\Phi)$. Thus, we will often identify a $\Phi$ with the set $Sat(\Phi)$.

Now, let us consider verifying validity of an atomic proposition $p \in AP$. By definition $I \models p \iff I \subseteq Sat(p)$. Since $Sat(p)$ is open, from Section 2.3 we know that, we can only verify $I \subseteq Sat(p)$ in a computable way, if $I$ is a compact. Thus, to make things uniform, we must assume that in model-checking we can only consider sets $I$ that are compact. Then, we also need to require that any $\Phi$ results in an open set of states, since otherwise $I \models \Phi$ is not computable.

In the following, we will see that then we can alter CTL semantics in such a way that any state formula results in an open set of states. This will be done implicitly by induction, assuming that any (state) sub formula results in an open sets of states. Also, we will not discuss the case of $\Phi \land \Phi$ since it is trivial.
Semantics: $I \models \neg \Phi$. By definition $Sat(\neg \Phi) = X \setminus Sat(\Phi)$ which is a closed set and thus $I \subseteq Sat(\neg \Phi)$ is uncomputable. Therefore, we follow the topological semantics given in [1] and define $Sat(\neg \Phi) = \text{Int}(X \setminus Sat(\Phi))$. The latter provides us with the following computable semantics:

$$I \subseteq \text{Int}(X \setminus \Phi) \Rightarrow I \models \neg \Phi.$$ 

Here $X \setminus \text{Sat}(\Phi)$ is closed and computable but $\text{Int}(X \setminus \text{Sat}(\Phi))$ is uncomputable. A possible assumption that the representation of $\text{Sat}(\Phi)$ is such that we can compute $\text{Int}(X \setminus \text{Sat}(\Phi))$, is not very realistic for an arbitrary $\Phi$. Fortunately, it is known that (see e.g. [14]), any CTL formula can be transformed into an equivalent CTL formula in the negation normal form (NNF). In NNF, negation only prefix atomic propositions and thus one can make sure that the representations of those are good enough to make their negations computable.

Semantics: $I \models \forall (X \Phi)$. For an open set $U_\Phi = \text{Sat}(\Phi)$, $F^{\equiv}(U_\Phi)$ is the set of states from which we always go into $U_\Phi$ states in one step. Clearly, $F^{\equiv}(U_\Phi)$ is open and computable and thus $I \subseteq F^{\equiv}(U)$ is also computable. Therefore, we define the computable semantics as:

$$I \models \forall (X \cup U) \leftrightarrow I \subseteq F^{\equiv}(U).$$

Semantics: $I \models \exists (X \Phi)$. Similar to the previous case, since $F^{-1}(U_\Phi)$ is open and computable, we define the computable semantics as:

$$I \models \exists (X U_\Phi) \leftrightarrow I \subseteq F^{-1}(U_\Phi).$$

We use weak preimage to account for the existential quantifier.

Semantics: $I \models \forall (\Phi \cup \Psi)$. As before, we assume that $\Phi$ and $\Psi$ corresponds to open sets $U_\Phi$ and $U_\Psi$. According to the original semantics of CTL:

$$I \models \forall (U_\Phi \cup U_\Psi) \leftrightarrow \forall x \in I : \forall \sigma \in \text{Paths}(x) : \exists j \geq 0 :$$

$$(\sigma [j]) \in U_\Phi \land \forall 0 \leq i < j : \sigma [i] \in U_\Psi$$

In other words, initial set of states $I$ satisfies the formula if, for every state $x \in I$, every path starting in $x$ consists of a finite prefix of states from $U_\Phi$ that is then followed by at least one state from $U_\Psi$. Therefore, the set $S_0 = U_\Psi$ gives states in which the formula $\forall \Phi \cup \Psi$ is satisfied right away. States $S_1 = F^{\equiv}(U_\Psi) \cap U_\Phi$ satisfy the formula, since any path $\sigma \in \text{Paths}(S_1)$ is such that $\sigma [0] \in U_\Phi$ and $\sigma [0] \in U_\Psi$ (because $\sigma [1] \in F(\sigma [0]) \subseteq U_\Phi$). By induction, for any $n \geq 1$ we have that $S_n = F^{\equiv}\left(\bigcup_{i=0}^{n-1} S_i\right) \cap U_\Phi$ is an open set that consists of states satisfying the formula. Clearly, the union $\bigcup_{n=0}^{\infty} S_n$ gives us all system states where the formula is satisfied and thus we can define: $I \models \forall (U_\Phi \cup U_\Psi) \Leftrightarrow I \subseteq \bigcup_{n=0}^{\infty} S_n$.

Notice that, this semantics is computable since every $S_i$ is open and computable and a countable union of $S_i$ is also open and computable.

Semantics: $I \models \exists (\Phi \cup \Psi)$. Since $S_0 = U_\Psi$ and $S_n = F^{-1}\left(\bigcup_{i=0}^{n-1} S_i\right) \cap U_\Phi$ are open and computable, we define the computable semantics as:

$$I \models \exists (U_\Phi \cup U_\Psi) \Leftrightarrow I \subseteq \bigcup_{n=0}^{\infty} S_n'. $$
Semantics: $I \models \forall (\Phi \land \Psi)$. Let us consider $\Phi \land \Psi$ (henceforth operator) — a common abbreviation of $(\text{false } \land \Psi)$. First, we provide a computable semantics for this formula and then discuss a more complex case of the release operator. Intuitively, the $\text{CTL}$ semantics for this formula can be expressed as: $I \models \forall (\Phi \land \Psi) \iff I \subseteq \bigcap_{n=0}^{\infty} S_n$ with $S_0 = U$ and for all $n \geq 1 : S_n = F^= (S_{n-1})$. It is easy to see that $\bigcap_{n=0}^{\infty} S_n$ is a set of states from which every path goes only through states in $U$. Unfortunately, even though every $S_n$ is open, countable intersection of open sets is not necessarily open (see Section 2.1) and thus the semantics above is not computable. Notice that, a set equivalent to $\bigcap_{n=0}^{\infty} S_n$ is given by: 

$$P := \bigcup \{ S \subseteq X \mid \exists S \subseteq U \land S \subseteq F^= (S) \}.$$ 

This set is neither open nor computable. Let us define $P' := \bigcup \{ O \subseteq \tau \mid O \subseteq U \land O \subseteq F^= (O) \}$. Clearly, $P' \subseteq P$ and the set $P'$ is “almost computable”. The latter is because $P'$ is an union of open sets but since $U$ and $F^= (O)$ are open the checks $O \subseteq U$ and $O \subseteq F^= (O)$ are not computable. Also, getting all open subsets from $\tau$ is not a computable operation. To remedy these problems we define the computable semantics as:

$$I \subseteq \bigcup \{ B_r \in \tau \mid \forall (\Phi \land \Psi) \} \Rightarrow I \models \forall (\Phi \land \Psi)$$

Here, each $B_r \in \tau$ is a finite union of base elements (e.g. open rational boxes in $X$) and from the definition of the computable Hausdorff space $\text{Cl} (B_r)$ is computable and compact. Since the union of sets on the left-hand side of the implication is a subset of $P' \subseteq P = \bigcap_{n=0}^{\infty} S_n$, we have that if the formula satisfied in the computable semantics, then it is satisfied in the original semantics.

Semantics: $I \models \exists (\Phi \land \Psi)$. Similar to the previous case, we define

$$I \subseteq \bigcup \{ B_r \in \tau \mid \exists (\Phi \land \Psi) \} \Rightarrow I \models \exists (\Phi \land \Psi)$$

Semantics: $I \models \forall (\Psi \land \Phi)$. According to the original semantics of $\text{CTL}$:

$$I \models \forall (\Psi \land \Phi) \Rightarrow \forall x \in I : \forall \sigma \in \text{Paths} (x) : (\forall i : \sigma[i] \in U_\Phi) \land (\exists j : (\sigma[j] \in U_\Phi \land \forall i \leq j : \sigma[i] \in U_\Phi))$$

Assuming an abbreviation for $GU_\Phi$, we get:

$$I \models \forall (\Psi \land \Phi) \Rightarrow \forall x \in I : \forall \sigma \in \text{Paths} (x) : (\sigma \models GU_\Phi) \land (\sigma \models U_\Phi \land (U_\Phi \land U_\Phi)) \land.$$  

Clearly, a path $\sigma$ satisfies $(\Psi \land \Phi)$ iff every state of the path belongs to $U_\Phi$ or there is a finite prefix of states from $U_\Phi$ where the last state of the prefix is also in $U_\Phi \land U_\Phi$. We already know how to treat universal until and henceforth operators, here we should simply combine the approaches:

$$I \subseteq \bigcup \{ B_r \in \tau \mid \forall (\Phi \land \Psi) \} \Rightarrow I \models \forall (\Psi \land \Phi)$$

Note that, since we require $\text{Cl} (B_r) \subseteq U_\Phi$, the second condition on the set $B_r$ serves two purposes: $(i)$ to provide fixed point characterisations for the paths satisfying $GU_\Phi$; $(ii)$ to allow paths with prefixes in $U_\Phi$ and the last states of these prefixes being in $U_\Phi \land U_\Phi$. The latter results in paths satisfying $(U_\Phi \land (U_\Phi \land U_\Phi))$. Since $B_r \cup (U_\Phi \land U_\Phi)$ is an open set, the resulting semantics is computable.
Semantics: $I \models \exists (\Psi \mathcal{R} \Phi)$. Similar to the previous case, we define:

$$I \subseteq \bigcup \{B_r \in \tau | \text{Cl}(B_r) \subseteq U_{\Phi} \land \text{Cl}(B_r) \subseteq U_{\Psi} \lor \text{Cl}(B_r) \subseteq F^{-1}(B_r \cup (U_{\Psi} \cap U_{\Phi}))\}$$

$$\Rightarrow I \models \exists (U_{\Psi} \mathcal{R} U_{\Phi})$$

### 3.2 Changes in the semantics

A consequence of the (necessary) choice of semantics of the negation and release (henceforth) operators is that there can not be a proof by contradiction (the Law of Excluded Middle does not hold). In other words, if $\Phi$ contains at least one of the above mentioned operator, the fact that $M, I \models \Phi$ does not hold does not imply that $M, I \models \neg \Phi$ holds. Moreover, such a $\Phi$ can be true (on $M, I$) but not computably verifiable.

There is nothing we can do about the Law of Excluded Middle for the henceforth and release operators. Moreover, in [7] it was shown that the provided semantics for former one is optimal. Also notice that, on the logical level $\neg \neg \Phi$ and $\Phi$ are equivalent whereas in the given topological semantics we generally have $I \models \neg \neg \Phi \nleftrightarrow I \models \Phi$. This means that converting $\Phi$ into NNF, prior to model checking, can reduce the effects of the computable (topological) semantics.

### 4 Concluding remarks

In this work we focused on model checking of discrete-time continuous-space control systems and in particular on computability aspects thereof. Due to continuity of the state space, regular computability and complexity theory is not applicable and therefore a more powerful approach is required. Our choice is the Type Two Effectivity theory (TTE). One of our main goals was to provide a logic that would allow to express system properties that are effectively verifiable (computable) in the sense of TTE. After considering several popular logics for hybrid systems, including topological logics, we decided to provide a computable semantics for the branching time logic called $CTL$. We have analysed this logic with respect to its computability aspects. It turned out that logical operators such as negation and release until require a significant change in their interpretation, to make sets of states satisfying formulae using them computable.

As it was discussed in Section 3.1 the computable model checking of $CTL$ on DTCSDSs assumes a compact set of initial states $I$, a $CTL$ state formula $\Phi$, and a system model $M = (T, F, L)$, where $T = (X, \tau, \beta, \nu)$ is a computable Hausdorff space and a map $F \in C(X, X)$ for which $F^{-1}(U)$ and $F^= (U)$ are computable for $U \in \tau$. If $\Phi$ contains negations, then it must be converted into NNF and the open sets, corresponding to atomic propositions that are then prefixed with negations, must have representations that allow for computing interiors of their complements. In computable semantics, if $\Phi$ contains negation or release (henceforth) operators and $I, M \models \Phi$ does not hold then it does not imply that $I, M \models \neg \Phi$ holds. If the formula holds in the computable semantics, then it also holds in the original one.
Currently, we work on extending the approach presented in this paper towards computable model checking of LTL and CTL*. Our long-standing goal is to provide a computable logic for the most general class of hybrid systems. Another plan is to implement the computable model checking algorithms in a framework for reachability analysis of hybrid systems called Ariadne [2].

References

Toward a Semantical Model of Organization Growth for Multiagent Systems

Graçaliz Pereira Dimuro and Antônio Carlos da Rocha Costa
Centro Politécnico, Universidade Católica de Pelotas
Rua Felix da Cunha 412, 96010-000 Pelotas, Brazil, e-mail: {liz,rocha}@ucpel.tche.br

Abstract. This paper extends the Population-Organization model of multiagent systems with a formal notion of organization growth. Organization growth is modeled in a domain-theoretic way: the set of states of organization structures is construed as a lattice, and processes of organization growth are modeled as directed sets in such domain. Simple organization growth processes are illustrated.

Keywords: Population-Organization model, social exchanges, dynamics of organization structures, organization growth

1 Introduction

The PopOrg (Population-Organization) model is a semantical model introduced in [1] aiming to account for the structural dynamics of the organization of multiagent systems (MAS) [2]. The model was extended in different ways in [3–5], to tackle different aspects of such structural dynamics.

The organization structure of the PopOrg model is composed by two levels, namely the micro and the macro-level organization structures (respectively, $\omega$- and $\Omega$-organization structures, for short). In the $\omega$-organization structure, the formal concepts of micro-roles and micro-links are defined, the latter taking into account the organizational exchanges occurring between micro-roles.

The formal notions of social system and organizational links between social systems are to be defined in the $\Omega$-organization structure, the latter accounting for the exchanges between such systems. Properly implemented PopOrg models guarantee that micro-roles and micro-links, and group behaviors and group exchanges at the micro-level organization structure, as well as social systems and their interaction, are properly implemented by agent behaviors and agent exchanges at the population level [5].

This paper aims to lay down the basis for a semantical model of organization development in multiagent systems (MAS) by presenting here an initial, incomplete model, namely, a model of $\omega$-organization growth. To capture the semantical notion of $\omega$-organization growth, a domain-theoretic structure [6, 7] is put in relation to the organization structure of the PopOrg model. Such domain-theoretic structure (a lattice of states of $\omega$-organization structures ordered by a certain organization embedding relation), is able to model the structural effects of the monotonic steps that characterize processes of $\omega$-organization growth.
The paper is organized as follows. In Sect. 2, we briefly review the PopOrg model, introducing the embedding relations and showing related elements that constrain and guide the growth processes of organization structure. The domain-theoretic model of ω-organization growth itself is introduced in Sect. 3, building on an example of process of organization growth. Section 4 is the Conclusion.

2 The PopOrg Model

The Population-Organization model of MAS (PopOrg, for short) [3–5] with a ωΩ-organization structure is defined as tuple \( POPORG = (POP, ORG_ωΩ, IMP_ωΩ) \), where \( POP \) is the population structure, \( ORG_ωΩ = (ORG_ω, ORG_Ω) \) is the two-level organization structure, with a micro-level \( ORG_ω \) and a macro-level \( ORG_Ω \), and \( IMP_ωΩ = (IMP_ω, IMP_Ω) \) is the two-level implementation relation, with an implementation sub-relation for each respective organization level.

The micro-organization structure \( ORG_ω \) is where the concepts of roles and interactions between roles are specified. Also, it is where the concepts of social groups and interactions between social groups are defined. On the other hand, the macro-organization structure \( ORG_Ω \) is where the concept of social systems is defined. In this paper, we focus only in \( ORG_ω \) growth processes.

Let \( \text{Act} = \text{Act}_Ag ∪ \text{Act}_Ro \) be the non-empty universe of actions that may be performed by agents or roles. The way the role actions are composed by agent actions is specified by a role action structuring function \( Ras : \text{Act}_Ro \rightarrow \wp(\text{Act}_Ag) \), with \( Ras(ρ) = A \) meaning that the role action \( ρ \) is structured as the set of agent actions \( A \). The function \( Ras \) can be extended to a function on sets of role actions or on pairs of sets of role actions, both also denoted by \( Ras \).

We assume that an equivalence relation \( \equiv_{\text{Act}} \subseteq \text{Act} × \text{Act} \) is given, where \( a \equiv_{\text{Act}} a' \) means that the action \( a \) is (in some sense) equivalent to the action \( a' \). This equivalence relation is extended to sets of actions and denoted by \( \equiv_{\wp(\text{Act})} \subseteq \wp(\text{Act}) × \wp(\text{Act}) \).\(^1\) Also, we say that a subset of actions \( A \subseteq \text{Act} \) is a compatible subset of actions if the actions in \( A \) can be performed simultaneously.

Let \( T = t_0, t_1, \ldots \) be the a discrete sequence of time instants. A behavior of an agent or a role, is modeled as function \( bh : T \rightarrow \wp(\text{Act}) \) that specifies, for each time \( t \in T \), a compatible subset of actions that may be performed by the agent or the role, respectively (that is, a behavior determines a sequence of compatible sets of actions available for the agent, or role, to perform).

Denote by \( \text{Bh} = [T \rightarrow \wp(\text{Act})] \) the set of all behaviors, which is partitioned into \( \text{Bh} = \text{Bh}_Ag ∪ \text{Bh}_Ro \). An equivalence relation \( \equiv_{\text{Bh}} \subseteq \text{Bh} × \text{Bh} \) is defined by

\[
\text{bh} \equiv_{\text{Bh}} \text{bh}' \iff \forall t \in T : bh(t) \equiv_{\text{Act}} bh'(t).
\]

(1)

The embedding relations between behaviors \( \subseteq_{\text{Bh}} \subseteq \text{Bh} × \text{Bh} \) and subsets of behaviors \( \subseteq_{\wp(\text{Bh})} \subseteq \wp(\text{Bh}) × \wp(\text{Bh}) \) are defined by\(^2\)

\[
b \subseteq_{\text{Bh}} b' \iff \forall t \in T \forall \alpha \in b(t) \exists \alpha' \in b'(t) : \alpha \equiv_{\text{Act}} \alpha'
\]

(2)

\[
B \subseteq_{\wp(\text{Bh})} B' \iff \forall b \in B \exists b' \in B' : b \subseteq \text{Bh} b'.
\]

(3)

\(^1\) Whenever it is clear from the context, the index of a relation may be omitted.

\(^2\) The strict embedding relation \( \sqsubseteq \) is given by \( x \sqsubseteq y \iff x \subseteq y \land x \neq y \).
An exchange process between two agents or two roles is modeled as a function $e : T \to \varphi(\text{Act}) \times \varphi(\text{Act})$ that specifies, for each time $t \in T$, a pair $(A_1, A_2)$ of compatible subsets of actions such that $A_1 \cup A_2$ is also compatible, determining a sequence of exchanges available for any pair of agents or pair of roles to perform, by executing together or interleaving appropriately their corresponding actions. The set of exchange processes is denoted by $\text{Ep} = [T \to \varphi(\text{Act}) \times \varphi(\text{Act})]$, with $\text{Ep} = \text{Ep}_{\text{Ag}} \cup \text{Ep}_{\text{Ro}}$. The equivalence relation $\equiv_{\text{Ep}} \subseteq \text{Ep} \times \text{Ep}$ is defined analogously to Eq. 1, and the embedding relation between exchange processes $\sqsubseteq_{\text{Ep}} \subseteq \text{Ep} \times \text{Ep}$ is given by: $e \sqsubseteq_{\text{Ep}} e' \iff \forall t \in T \ \exists \ (\alpha_1, \alpha_2) \in e(t) \ \exists (\alpha'_1, \alpha'_2) \in e'(t) : (\alpha_1 \equiv_{\text{Act}} \alpha'_1) \wedge (\alpha_2 \equiv_{\text{Act}} \alpha'_2)$. The embedding relation between subsets of exchange processes $\sqsubseteq_{\varphi(\text{Ep})} \subseteq \varphi(\text{Ep}) \times \varphi(\text{Ep})$ is defined analogously to Eq. (3).

2.1 The Population Structure

Let $\text{Ag} \neq \emptyset$ be a universe of agents. The population of a MAS consists of the subset of agents that inhabit it:

**Definition 1.** A population structure is a time-indexed set $\text{POP}^t$ of states of population structure $\text{POP}^t=\{\text{AG}^t,\text{ACT}^t,\text{BH}^t,\text{EP}_{\text{Ag}}^t,\text{Ec}^t\}$, where for $t \in T$:
- $\text{AG}^t \in \varphi(\text{Ag})$ is the system’s population at time $t$;
- $\text{ACT}^t \in \varphi(\text{Act}_{\text{Ag}})$ is the subset of possible agent actions at time $t$;
- $\text{BH}^t \in \varphi(\text{Bh}_{\text{Ag}})$, with $\text{BH}^t \subseteq [T \to \varphi(\text{ACT}^t)]$, is the subset of possible agent behaviors at time $t$;
- $\text{EP}_{\text{Ag}}^t \in \varphi(\text{Ep}_{\text{Ag}})$, with $\text{EP}_{\text{Ag}}^t \subseteq [T \to \varphi(\text{ACT}^t) \times \varphi(\text{ACT}^t)]$, is the subset of possible exchange processes between agents at time $t$;
- $\text{Ec}^t : \text{AG}^t \to \varphi(\text{Ec}^t)$ is the exchange capability function of agents at time $t$, such that for each agent $a \in \text{AG}^t$, the set of all behaviors that $a$ is able to perform at time $t$ is $\text{Ec}^t(a)$;
- $\text{Ec}^t : \text{AG}^t \times \text{AG}^t \to \varphi(\text{EP}_{\text{Ag}}^t)$ is the exchange capability function at time $t$, such that for each pair of agents $a_1, a_2 \in \text{AG}^t$, the set of all exchange processes that $a_1$ and $a_2$ can perform between them at time $t$ is $\text{Ec}^t(a_1, a_2)$;
- $\forall a_1, a_2 \in \text{AG}^t \ \forall e \in \text{Ec}^t(a_1, a_2) \ \forall t' \in T$:
  
  $\text{Prj}_1(e(t')) \subseteq \bigcup \{b(t') | b \in \text{Ec}^t(a_1)\} \land \text{Prj}_2(e(t')) \subseteq \bigcup \{b(t') | b \in \text{Ec}^t(a_2)\}$, (4)

where $\text{Prj}_1, \text{Prj}_2$ are projection functions, so that the agents’ exchange capabilities are constrained by their respective behavioral capabilities.

**Example 1.** In Fig. 1 we show a sample state of the population structure $\text{POP}^t$, where the population is $\text{AG}^t=\{a, a', a''\}$, the set of compatible agent actions is $\text{ACT}_{\text{Ag}}^t=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$, the set of agent behaviors is $\text{BH}_{\text{Ag}}^t=\{b_{h1}, b_{h2}, b'h', bh''\}$ and the set of exchange processes between agents is $\text{EP}_{\text{Ag}}^t=\{e_{aa''}, e_{a'a''}\}$. The behavioral and exchange capability functions are shown in the figure, with:

- $b_{h1}=\{0 \mapsto \{\alpha_1, \alpha_4\}, 1 \mapsto \emptyset, 2 \mapsto \{\alpha_1, \alpha_4\}\}$, $b_{h2}=\{0 \mapsto \emptyset, 1 \mapsto \{\alpha_1, \alpha_2, \alpha_4\}, 2 \mapsto \{\alpha_3\}\}$,
- $b'h'=\{0 \mapsto \{\alpha_1, \alpha_4\}, 1 \mapsto \{\alpha_3, \alpha_4, \alpha_5\}, 2 \mapsto \{\alpha_1, \alpha_2, \alpha_3, \alpha_5\}\}$,
- $bh''=\{0 \mapsto \{\alpha_2, \alpha_3\}, 1 \mapsto \{\alpha_2, \alpha_3\}, 2 \mapsto \{\alpha_4, \alpha_5\}\}$
- $e_{aa''}=\{0 \mapsto \{\{\alpha_1, \alpha_4\}, \{a_2\}\}, 1 \mapsto \{\{\alpha_1, \alpha_4\}, \{\alpha_3\}\}, 2 \mapsto \{\{\alpha_1, \alpha_4\}, \{\alpha_4\}\}\}$,
- $e_{a'a''}=\{0 \mapsto \{\{\alpha_1\}, \{\alpha_2, \alpha_3\}\}, 1 \mapsto \{\{\alpha_3\}, \{\alpha_2, \alpha_3\}\}, 2 \mapsto \{\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}\}$.
for all roles \( r \in \emptyset \)

Micro-level organization structure

\[
\text{LI}_\omega \subseteq \text{RO}_\omega \times \text{RO}_\omega,
\]

where

\[
r \subseteq \text{RO}_\omega \text{ } r' \Leftrightarrow \forall b \in r \ \exists b' \in r' : b \subseteq_{\text{BH}} b'.
\]

A micro-link \( l \) between a pair of micro-roles \( r_1, r_2 \in \text{RO}_\omega \) is defined as a tuple \( l = (r_1, r_2, e) \), specifying an exchange process \( e \in \text{EP}_{\text{RO}} \) that the linked micro-roles \( r_1 \) and \( r_2 \) may have to perform. The set of all micro-links is denoted by \( \text{LI}_\omega \). The embedding relation between micro-links \( \subseteq_{\text{LI}_\omega} \subseteq \text{LI}_\omega \times \text{LI}_\omega \) is defined for \( l = (r_1, r_2, e) \), \( l' = (r'_1, r'_2, e') \), by: \( l \subseteq_{\text{LI}_\omega} l' \Leftrightarrow r_1 \subseteq_{\text{RO}_\omega} r'_1 \land r_2 \subseteq_{\text{RO}_\omega} r'_2 \land e \subseteq_{\text{EP}} e' \).

**Definition 2.** A micro-level organization structure is a time-indexed set \( \text{ORG}_\omega \) of states of micro-level organization structures \( \text{ORG}_\omega^t = (\text{RO}_\omega^t, \text{LI}_\omega^t, \text{LC}_\omega^t) \), where for all \( t \in T \):

- \( \text{RO}_\omega^t \subseteq \varphi(\text{RO}_\omega) \) is the set of micro-roles in the organization at time \( t \);
- \( \text{LI}_\omega^t \subseteq \varphi(\text{LI}_\omega) \) is the set of micro-links existing in the organization at time \( t \);
- \( \text{LC}_\omega^t : \text{RO}_\omega^t \times \text{RO}_\omega^t \rightarrow \varphi(\text{LI}_\omega) \) is the micro-link capability function at \( t \), giving the set of micro-links that may be established between pairs of micro-roles;
- \( \forall t = (r_1, r_2, e) \in \text{LI}_\omega^t : l \in \text{LC}_\omega^t(r_1, r_2) \), i.e., each possible micro-link has to be in the micro-link capability of the two micro-roles that it links at time \( t \);
- \( \forall r_1, r_2 \in \text{RO}_\omega^t \ \forall e \in \text{EP}_{\text{RO}} \ \forall t \in T : (r_1, r_2, e) \in \text{LI}_\omega^t \Rightarrow \forall t' \in T : \text{Prj}_1(e(t')) \subseteq \bigcup \{ b(t') \mid b \in r_1 \} \land \text{Prj}_2(e(t')) \subseteq \bigcup \{ b(t') \mid b \in r_2 \},
\]

where \( \text{Prj}_1, \text{Prj}_2 \) are projection functions, so that the micro-roles’ exchange capabilities are constrained by their respective behavior capabilities.

For each \( t \in T \), \( \text{ORG}_\omega^t \) fixes the sets of possible micro-roles \( \text{RO}_\omega^t \), micro-links \( \text{LI}_\omega^t \), micro-link capability function \( \text{LC}_\omega^t \) and the set \( \text{EP}_{\text{RO}_\omega}^t = \{ e \mid l = (r_1, r_2, e) \in \text{LI}_\omega^t \} \) of all exchange processes between micro-roles in existence at time \( t \), and then \( \text{LI}_\omega^t \subseteq \text{RO}_\omega^t \times \text{RO}_\omega^t \times \text{EP}_{\text{RO}_\omega}^t \). Let \( \text{LC}_\omega \) be the set of all micro-link capability functions.

\[
lc \subseteq_{\text{LC}_\omega} lc' \Leftrightarrow \forall (r_1, r_2, l) \in \text{gr}(lc) \ \exists (r'_1, r'_2, l') \in \text{gr}(lc') : r_1 \subseteq_{\text{RO}_\omega} r'_1 \land r_2 \subseteq_{\text{RO}_\omega} r'_2 \land l \subseteq_{\text{LI}_\omega} l'
\]

**Fig. 1.** A PopOrg with \( \omega \)-organization structure (link implementations not shown)

### 2.2 The Micro-level Organization Structure

A micro-role \( r \) is as a non-empty set of role behaviors in \( \text{BH}_{\text{RO}} \). Denote by \( \text{RO}_\omega \) the set of all micro-roles. The embedding relation between micro-roles is defined as \( \subseteq_{\text{RO}_\omega} \subseteq \text{RO}_\omega \times \text{RO}_\omega \), where

\[
r \subseteq_{\text{RO}_\omega} r' \Leftrightarrow \forall b \in r \ \exists b' \in r' : b \subseteq_{\text{BH}} b'.
\]

Behavioral Capability Function:

\[
\text{BC}_r = \{ (a \rightarrow \{ b_{11}, b_{22} \}, a' \rightarrow \{ b_{12}, b_{22} \}, a'' \rightarrow \{ b_{12}, b_{22} \})
\]

Exchange Capability Function:

\[
\text{EC}_r = \{ (a, a') \rightarrow \{ e_{a,a'}, e_{a',a'} \}, (a, a'') \rightarrow \{ e_{a,a''}, e_{a'',a''} \}\}
\]

Micro-link capability function:

\[
\text{LC}_r = \{ (r_1, r_2) \rightarrow \{ l_{12}, (r_4, r_5) \rightarrow l_{13}, (r_4, r_5) \rightarrow l_{14}, (r_4, r_5) \rightarrow l_{15} \}
\]

The Micro-level Organization Structure

The set of all micro-roles. The micro-role as \( \text{LI}_\omega \) is defined as a tuple \( l \in \text{LI}_\omega \). The embedding relation between micro-roles \( \subseteq_{\text{LI}_\omega} \subseteq \text{LI}_\omega \times \text{LI}_\omega \) is defined, for \( l = (r_1, r_2, e) \), \( l' = (r'_1, r'_2, e') \), by: \( l \subseteq_{\text{LI}_\omega} l' \Leftrightarrow r_1 \subseteq_{\text{RO}_\omega} r'_1 \land r_2 \subseteq_{\text{RO}_\omega} r'_2 \land e \subseteq_{\text{EP}} e' \).

\[
\text{LI}_\omega^t \subseteq \text{RO}_\omega^t \times \text{RO}_\omega^t \times \text{EP}_{\text{RO}_\omega}^t.
\]

\[
\text{LC}_\omega \subseteq \text{LI}_\omega \times \text{RO}_\omega \times \text{RO}_\omega \times \text{EP}_{\text{RO}_\omega}.
\]
where $gr(f)$ denotes the graph of the function $f$.

The embedding relations of sets of micro-roles, micro-links and link capability functions, $\subseteq_{\mathcal{R}}(\mathcal{R}_\omega)$, $\subseteq_{\mathcal{P}}(\mathcal{L}_\omega)$ and $\subseteq_{\mathcal{P}}(\mathcal{L}_\omega)$, are defined analogously to Eq. 3.

Denote by $\text{Org}_\omega$ the set of all micro-level organization structures and let $\text{sts}[\text{Org}_\omega]$ be the set of all states of micro-level organization structures. The embedding relation between states of micro-level organization structures is the relation $\subseteq_{\text{sts}[\text{Org}_\omega]} \subseteq \text{sts}[\text{Org}_\omega] \times \text{sts}[\text{Org}_\omega]$, such that $\forall t, t' \in T$: $(\text{ORG}_1)_\omega \subseteq_{\text{sts}[\text{Org}_\omega]} (\text{ORG}_2)_\omega \iff (\text{RO}_1)_\omega \subseteq_{\text{Org}_\omega} (\text{RO}_2)_\omega \land (\text{L}_1)_\omega \subseteq_{\text{Org}_\omega} (\text{L}_2)_\omega \land (\text{L}_1)_\omega \subseteq_{\text{Org}_\omega} (\text{L}_2)_\omega$.

**Theorem 1.** (Micro-link Preservation Property) If, for some $t, t' \in T$, there exist micro-roles $r_1, r_3 \in \text{RO}_t$ and $r_2, r_4 \in \text{RO}_{t'}$ such that $r_1 \subseteq r_2$ and $r_3 \subseteq r_4$, then whenever there exists a micro-link $l_{13} = (r_1, r_3, e_{13}) \in \text{Lc}(r_1, r_3)$ then there exists a micro-link $l_{24} = (r_2, r_4, e_{24}) \in \text{Lc}(r_2, r_4)$ such that $e_{24} \equiv e_{13}$.

**Proof.** Consider the micro-link $l_{13} = (r_1, r_3, e_{13}) \in \text{Lc}(r_1, r_3)$ and that $r_1 \subseteq r_2$ and $r_3 \subseteq r_4$, for $r_2, r_4 \in \text{RO}_{t'}$. Then, by Eq. (5), for all $b_1 \in r_1$ there is $b_2 \in r_2$ such that $b_1 \subseteq b_2$ and for all $b_3 \in r_3$ there is $b_4 \in r_4$ such that $b_3 \subseteq b_4$. Thus, by Eq. (2), for all $\tau \in T$, and for all $\alpha_1 \in b_1(\tau)$, there is $\alpha_2 \in b_2(\tau)$ such that $\alpha_1 \equiv \alpha_2$. Analogously, for all $\tau \in T$ and for all $\alpha_3 \in b_3(\tau)$ there is $\alpha_4 \in b_4(\tau)$ such that $\alpha_3 \equiv \alpha_4$. Thus, for every $\tau \in T$ and every $(\alpha_1, \alpha_3) \in e_{13}(\tau)$ there is a pair of actions $(\alpha_2, \alpha_4)$ such that $(\alpha_2, \alpha_4) \equiv (\alpha_1, \alpha_3)$. That thus defines an exchange process $e_{24}$, given, for all $\tau \in T$, by $e_{24}(\tau) = \{(\alpha_2, \alpha_4) \mid \exists (\alpha_1, \alpha_3) \in e_{13}(\tau) : (\alpha_2, \alpha_4) \equiv (\alpha_1, \alpha_3)\}$. Clearly, it holds that $e_{24} \in \text{Ec}(r_2, r_4(t'))$ and $e_{13} \equiv e_{24}$.

**Corollary 1.** (Micro-link Growing Property) If, for some consecutive time instants $t_i, t_{i+1} \in T$, there exist micro-roles $r_1, r_3 \in \text{RO}_{t_i}$ and $r_2, r_4 \in \text{RO}_{t_{i+1}}$ such that $r_1 \subseteq r_2$ and $r_3 \subseteq r_4$, then whenever there exists a micro-link $l_{13} = (r_1, r_3, e_{13}) \in \text{Lc}(r_1, r_3)$ then there exists a micro-link $l_{24} = (r_2, r_4, e_{24}) \in \text{Lc}(r_2, r_4)$ such that $e_{13} \subseteq e_{24}$.

**Example 2.** Figure 1 shows a sample state of the micro-level organization structure $\text{ORG}_\omega$, where the set of micro-roles is $\text{RO}_\omega = \{r_1, r_2, r_3, r_4\}$, the set of micro-links is $\text{L}_\omega = \{l_{12}, l_{43}, l_{13}, l_{42}\}$ and the micro-link capability function is shown in the figure, with the micro-roles and micro-links given by:

$r_1 = \{b_1, b'_1, b''_1\}$, where $b_1 = \{0 \rightarrow \{\alpha_1\}, 1 \rightarrow \{\alpha_2, \alpha_3\}, 2 \rightarrow \emptyset\}$, $b'_1 = \{0 \rightarrow \emptyset\}$, $b''_1 = \{0 \rightarrow \{\alpha_4\}, 1 \rightarrow \{\alpha_1\}, 2 \rightarrow \emptyset\}$;

$r_2 = \{b_2\}$, where $b_2 = \{0 \rightarrow \{\alpha_2\}, 1 \rightarrow \{\alpha_3\}, 2 \rightarrow \{\alpha_4, \alpha_5\}\}$;

$r_3 = \{b_3\}$, where $b_3 = \{0 \rightarrow \{\alpha_3\}, 1 \rightarrow \{\alpha_2\}, 2 \rightarrow \{\alpha_4, \alpha_5\}\}$;

$r_4 = \{b_4, b'_4\}$, where $b_4 = \{0 \rightarrow \{\alpha_1, \alpha_2\}, 1 \rightarrow \{\alpha_3, \alpha_4\}, 2 \rightarrow \{\alpha_5\}\}$, $b'_4 = \{0 \rightarrow \emptyset, 1 \rightarrow \emptyset, 2 \rightarrow \{\alpha_1, \alpha_2, \alpha_3\}\}$;

$l_{11} = (r_1, r_2, e_{12})$, where $e_{12} = \emptyset \leftrightarrow \{\{\alpha_1, \alpha_4\}, \{\alpha_2\}\}$, $1 \leftrightarrow \{\{\alpha_1, \alpha_3, \alpha_4\}, \{\alpha_3\}\}, 2 \rightarrow \emptyset$;

$l_{43} = (r_4, r_3, e_{43})$, where $e_{43} = \emptyset \leftrightarrow \{\{\alpha_1\}, \{\alpha_3\}\}, 1 \leftrightarrow \{\{\alpha_3\}, \{\alpha_4\}\}, 2 \rightarrow \emptyset$;

$l_{13} = (r_1, r_3, e_{13})$, where $e_{13} = \emptyset \leftrightarrow \{\{\alpha_1\}, \{\alpha_3\}\}, 1 \leftrightarrow \{\{\alpha_3\}, \{\alpha_4\}\}, 2 \rightarrow \emptyset$;

$l_{42} = (r_4, r_2, e_{42})$, where $e_{42} = \emptyset \leftrightarrow \{\{\alpha_1\}, \{\alpha_2\}\}, 1 \leftrightarrow \emptyset, 2 \rightarrow \{\alpha_1, \alpha_2, \alpha_4\}\}$. 

124
2.3 The Implementation Relations

An $\omega\Omega$-implementation relation of the $\omega\Omega$-organization structure $\text{ORG}_{\omega\Omega} = (\text{ORG}_\omega, \text{ORG}_\Omega)$ on the population structure $\text{POP}$ is a pair $\text{IMP}_{\omega\Omega} = (\text{IMP}_\omega, \text{IMP}_\Omega)$, where $\text{IMP}_\omega$ and $\text{IMP}_\Omega$ are the micro-level and macro-level implementation relations, respectively, defining how the population structure $\text{POP}$ implements the micro-level organization structure $\text{ORG}_\omega$, and how $\text{ORG}_\omega$ implements the macro-level organization structure $\text{ORG}_\Omega$. In this paper, only the micro-level implementation relation is discussed.

**Definition 3.** A $\omega$-level implementation relation for $\text{ORG}_\omega = (\text{RO}_\omega, \text{LI}_\omega, \text{LC}_\omega)$ on $\text{POP} = (\text{AG}, \text{ACT}, \text{BH}, \text{EP}_\text{Ag}, \text{BC}, \text{EC})$, according to a role action structuring function $\text{Ras} : \text{Act}_\text{Ro} \rightarrow \varnothing(\text{Act}_\text{Ag})$, is a time-indexed set $\text{IMP}_\text{Ras}^\omega$ of micro-level implementation relation states, with $(\text{IMP}_\text{Ras}^\omega)^t \subseteq (\text{RO}_\omega^t \times \text{AG}^t) \cup (\text{LI}_\omega^t \times \text{EP}_\text{Ag}^t)$, where at each time $t \in T$:

- $\text{RO}_\omega^t \times \text{AG}^t$ is the set of all possible micro-role implementations, that is, the set of all possible ways of assigning micro-roles to agents at time $t$, so that if $(r, a) \in (\text{IMP}_\text{Ras}^\omega)^t$, then the micro-role $r$ is implemented by agent $a$ at time $t$ (possibly in a shared, non-exclusive way);
- $\text{LI}_\omega^t \times \text{EP}_\text{Ag}^t$ is the set of all possible micro-link implementations, that is, the set of possible ways of implementing micro-links, so that if $(l, e) \in (\text{IMP}_\text{Ras}^\omega)^t$, then the micro-link $l$ is said to be implemented (in a possibly shared, non-exclusive way) by the agent exchange process $e$ at time $t$.

**Definition 4.** A micro-role $r \in \text{RO}_\omega^t$ in $\text{ORG}_\omega^t$ is said to be properly implemented by $(\text{IMP}_\text{Ras}^\omega)^t$ on $\text{POP}^t$ if and only if there is a subset $A \subseteq \text{Ag}^t$ of agents such that the following conditions hold:

- $\forall a \in A : (r, a) \in (\text{IMP}_\text{Ras}^\omega)^t$, that is, every agent in $A$ participates in the implementation of the micro-role $r$ at time $t$;
- $\forall t' \in T : \bigcup\{\text{Ras}(b_r(t')) | b_r \in r\} \subseteq \bigcup\{b_a(t') | b_a \in \text{BC}^t(a), a \in A\}$, that is, the set of behaviors required by the micro-role $r$ can be performed by the agents in $A$ (in a possibly non-exclusive, shared way), under the role action structuring function $\text{Ras}$.

**Definition 5.** A micro-link $l = (r_1, r_2, e) \in \text{LI}_\omega^t$ in $\text{ORG}_\omega^t$ is said to be properly implemented by $(\text{IMP}_\text{Ras}^\omega)^t$ on $\text{POP}^t$ if and only if there is a subset $E \subseteq \bigcup\{\text{BC}^t(a_1, a_2) | (r_1, a_1), (r_2, a_2) \in \text{IMP}_\omega\}$ of the set of exchange processes that belong to the exchange capability of the agents that implement $r_1$ and $r_2$ at time $t \in T$, such that the following conditions hold:

- $\forall e' \in E : (l, e') \in (\text{IMP}_\text{Ras}^\omega)^t$, that is, each exchange process in $E$ participates in the implementation of the micro-link $l$ at time $t \in T$;
- $\forall t' \in T : \bigcup\text{Ras}(e(t')) \subseteq \bigcup\{e'(t') | e' \in E\}$, that is, every projection $\text{Pr}j_i (i=1,2)$ of the exchange process required by $l$ can be realized by the set of corresponding projections of the exchange processes of $E$ (in a possibly non-exclusive, shared way).³

³ We use the following coordinate-wise notation: $\bigcup\{(X_1, \ldots, X_n)|\text{expr}_0\} \equiv_{\text{def}} (\bigcup\{X_1|\text{expr}_1\}, \ldots, \bigcup\{X_n|\text{expr}_n\})$, with $\text{expr}_0 \Leftrightarrow \text{expr}_1 \land \ldots \land \text{expr}_n$. 

125
Definition 6. \( \text{IMP}_\omega^{\text{Ras}} \) is a proper \( \omega \)-level implementation relation of \( \text{ORG}_\omega \) on \( \text{POP} \) if and only if, at each time \( t \), each micro-role \( r \in \text{RO}_t^\omega \) in \( \text{ORG}_t^\omega \) and micro-link \( l \in \text{LI}_t^\omega \) in \( \text{ORG}_t^\omega \) is properly implemented by \( \text{IMP}_\omega^{\text{Ras}} \) on \( \text{POP}_t^\omega \).

Example 3. Consider Ex. 1 and 2, and Fig. 1. The implementation relation state \( \text{IMP}_\omega^{\text{Ras}} t = \{(r_1, a), (r_1, a'), (r_2, a''), (r_3, a'''), (r_4, a''') \} \cup \{(l_1, e_\text{a'''a'''}) \} \) is a proper implementation relation state of \( \text{ORG}_t^\omega \) on \( \text{POP}_t^\omega \), when \( \text{Ras} \) is a unitary injection, i.e., \( \text{Ras}(\alpha) = \{ \alpha \} \).

3 The Domain of States of \( \omega \)-Organization Structures

Let \( \text{sts}[\text{Org}_\omega] \) be set of all states of \( \omega \)-level organization structures. Then the structure \( \text{sts}[\text{Org}_\omega] \subseteq \text{sts}[\text{Org}_\omega] \) is a partial order, since the embedding relation \( \subseteq \text{sts}[\text{Org}_\omega] \) is reflexive, transitive and antisymmetric.

The operations of substitution of roles in links and in capability functions are defined, respectively, by \( (r_1, r_2, e)[(r_3, r_4)/(r_1, r_2)] = (r_3, r_4, e) \) and \( (r_1, r_2) \mapsto L) [(r_3, r_4, L')/(r_1, r_2, L)] = (r_3, r_4) \mapsto L' \), where \( L' = \{(r_3, r_4)/(r_1, r_2) \mid t \in L \} \).

Definition 7. The union of two states of micro-level organization structures \( \text{ORG}_1^\omega = ((\text{RO}_1^\omega), (\text{LI}_1^\omega), (\text{LC}_1^\omega)) \) and \( \text{ORG}_2^\omega = ((\text{RO}_2^\omega), (\text{LI}_2^\omega), (\text{LC}_2^\omega)) \) is the state of the micro-level organization structure \( ((\text{ORG}_1)_{\omega} \sqcup (\text{ORG}_2)_{\omega}) = (((\text{RO}_1)_{\omega} \sqcup (\text{RO}_2)_{\omega})^i, ((\text{LI}_1)_{\omega} \sqcup (\text{LI}_2)_{\omega})^i, ((\text{LC}_1)_{\omega} \sqcup (\text{LC}_2)_{\omega})^i), \) where:

\[
\begin{align*}
&\begin{cases}
(\text{LC}_1)^i(r, r') & \text{if } r, r' \in (\text{RO}_1)^i \land r, r' \notin (\text{RO}_2)^i \\
(\text{LC}_2)^i(r, r') & \text{if } r, r' \in (\text{RO}_2)^i \land r, r' \notin (\text{RO}_1)^i \\
\emptyset & \text{if } [r \in ((\text{RO}_1)^i - (\text{RO}_2)^i) \land r' \in ((\text{RO}_2)^i - (\text{RO}_1)^i) \lor r' \in ((\text{RO}_1)^i - (\text{RO}_2)^i) \lor r \in ((\text{RO}_2)^i - (\text{RO}_1)^i)] \land r \in r_1 \land r \subseteq r_1 \land r_2 \subseteq r_2 \land r_3 \subseteq r_3 \land r_4 \subseteq r_4.
\end{cases}
\end{align*}
\]

where, for \( i = 1, 2 \); \((\text{LC}_1)^i = \{(r_1, r_2) \mapsto L) [(r_3, r_4, (\text{LI}_1)_{\omega} \sqcup (\text{LI}_2)_{\omega})^i)/(r_1, r_2, L)]
\]

\((r_1, r_2) \mapsto L) \in (\text{LC}_1)^i \land r_3, r_4 \in ((\text{RO}_1)_{\omega} \sqcup (\text{RO}_2)_{\omega})^i \land r_1 \subseteq r_3 \land r_2 \subseteq r_4 \).

Definition 8. The intersection of two states of micro-level organization structures \( \text{ORG}_1^\omega = ((\text{RO}_1^\omega), (\text{LI}_1^\omega), (\text{LC}_1^\omega)) \) and \( \text{ORG}_2^\omega = ((\text{RO}_2^\omega), (\text{LI}_2^\omega), (\text{LC}_2^\omega)) \) is the state of \( \omega \)-level organization structure \( ((\text{ORG}_1)_{\omega} \cap (\text{ORG}_2)_{\omega})^i = (((\text{RO}_1)_{\omega} \sqcap (\text{RO}_2)_{\omega})^i, ((\text{LI}_1)_{\omega} \sqcap (\text{LI}_2)_{\omega})^i, ((\text{LC}_1)_{\omega} \sqcap (\text{LC}_2)_{\omega})^i), \) where:
\begin{align*}
((RO_1)_\omega \cap (RO_2)_\omega)^t &= \{ r \in (RO_1)_\omega \cup (RO_2)_\omega \mid -\exists r' \in (RO_1)_\omega \cup (RO_2)_\omega : r' \sqsubseteq r \} ; \\
((LI_1)_\omega \cap (LI_2)_\omega)^t &= \{ l \in (LI_1)_\omega \cup (LI_2)_\omega \mid -\exists l' \in (LI_1)_\omega \cup (LI_2)_\omega : l' \sqsubseteq l \},
\end{align*}
where, for \( i = 1, 2 \):
\begin{align*}
(LI_1)_\omega &= \{(r_1, r_2, e)[(r_3, r_4)/(r_1, r_2)] \mid (r_1, r_2, e) \in (LI_1)_\omega \land r_3, r_4 \in ((RO_1)_\omega \cap (RO_2)_\omega)^t \land r_3 \sqsubseteq r_1 \land r_4 \sqsubseteq r_2 \} ; \\
(LC_1)_\omega \cap (LC_2)_\omega)^t &= \{(RO_1)_\omega \cap (RO_2)_\omega)^t \times ((RO_1)_\omega \cap (RO_2)_\omega)^t \rightarrow ((LI_1)_\omega \cap (LI_2)_\omega)^t ,
\end{align*}
such that \( ((LC_1)_\omega \cap (LC_2)_\omega)^t (r, r') = \begin{cases} 
\langle LC_1 \rangle^t((r, r')) & \text{if } r, r' \in (RO_1)_\omega \land r, r' \not\in (RO_2)_\omega \\
\langle LC_2 \rangle^t((r, r')) & \text{if } r, r' \in (RO_2)_\omega \land r, r' \not\in (RO_1)_\omega \\
\{ l \in \langle LC_1 \rangle^t((r, r')) \cup \langle LC_2 \rangle^t((r, r')) \mid -\exists l' \in \langle LC_1 \rangle^t((r, r')) \cup \langle LC_2 \rangle^t((r, r')) : l' \sqsubseteq l \} & \text{if } r, r' \in (RO_1)^t \land r, r' \not\in (RO_2)^t \\
\emptyset & \text{if } [r \in ((RO_1)^t - (RO_2)^t) \land r' \in ((RO_2)^t - (RO_1)^t) \lor r \in ((RO_2)^t - (RO_1)^t) \land r' \in ((RO_1)^t - (RO_2)^t)] 
\end{cases} \}
\end{align*}

where, for \( i = 1, 2 \):
\begin{align*}
\langle LC_1 \rangle^t((r, r')) &= \{(r_1, r_2) \mapsto L)[(r_3, r_4, (\langle LI_1 \rangle \cup (LI_2) \omega)^t)/(r_1, r_2, L)] \mid (r_1, r_2) \mapsto L \in \langle LC_1 \rangle^t \land r_3, r_4 \in ((RO_1) \omega \cap (RO_2) \omega)^t \land r_3 \sqsubseteq r_1 \land r_4 \sqsubseteq r_2 \}.
\end{align*}

Definitions 7 and 8 can be extended to define the union and the intersection of a subset of states of \( \omega \)-organization structures. The union and intersection of states of micro-level organization structures are closed in \( \text{sts}[\text{Org}_\omega] \). Then:

**Lemma 1.** The lub of any subset of states \( O^t_\omega = \{(\text{ORG}_1)^t_\omega, (\text{ORG}_2)^t_\omega, \ldots \} \subseteq \text{sts}[\text{Org}_\omega] \) is the state of micro-level organization structure given by the union of \( O^t_\omega \), \( \bigcup_i (\text{ORG}_i)_\omega^t = \bigcup_i (\text{RO}_i)_\omega^t \cup \bigcup_i (\text{LI}_i)_\omega^t \cup \bigcup_i (\text{LC}_i)_\omega^t \).

**Proof.** Analogously to Def. 7, \( \bigcup_i (\text{RO}_i)_\omega^t = \{ r \in \bigcup_i (\text{RO}_i)_\omega^t \mid -\exists r' \in \bigcup_i (\text{RO}_i)_\omega^t : r \sqsubseteq r' \} \) is the union of the sets of roles of each \( (\text{ORG}_i)_\omega^t \subseteq \bigcup_i (\text{RO}_i)_\omega^t \) at time \( t \). Suppose that there exists \( (\text{ORG}_k)_\omega^t = ((\text{RO}_k)_\omega^t, (\text{LI}_k)_\omega^t, (\text{LC}_k)_\omega^t) \subseteq O^t_\omega \), such that \( \bigcup_i O^t_i \subseteq \bigcup_i (\text{ORG}_i)_\omega^t \). Then, for all \( t \in T \), we have that \( \bigcup_i (\text{RO}_i)_\omega^t \subseteq \bigcup_i (\text{RO}_i)_\omega^t \). This means that for all \( r \in \bigcup_i (\text{RO}_i)_\omega^t \), there exists \( r' \in \bigcup_i (\text{RO}_i)_\omega^t \) such that \( r \sqsubseteq r' \). However, the possibility of an \( r' \not\in \bigcup_i (\text{RO}_i)_\omega^t \) such that \( r \sqsubseteq r' \) is a contradiction, since, by hypothesis, there is no such \( r' \). Then, one has that \( \bigcup_i (\text{RO}_i)_\omega^t = \bigcup_i (\text{RO}_i)_\omega^t \). By similar arguments one has that \( \bigcup_i (\text{LI}_i)_\omega^t = \bigcup_i (\text{LI}_i)_\omega^t \) and \( \bigcup_i (\text{LC}_i)_\omega^t = \bigcup_i (\text{LC}_i)_\omega^t \). Thus, it holds that \( \bigcup_i O_i^t = \bigcup_i (\text{RO}_i)_\omega^t \) and \( \bigcup_i O_i^t \) is the lub of \( O_i^t \).

**Lemma 2.** The glb of any subset of states \( O^t_\omega = \{(\text{ORG}_1)^t_\omega, (\text{ORG}_2)^t_\omega, \ldots \} \subseteq \text{sts}[\text{Org}_\omega] \) is the state of micro-level organization structure given by the intersection of \( O^t_\omega \), \( \bigcap_i (\text{ORG}_i)_\omega^t = \bigcap_i (\text{RO}_i)_\omega^t \cap \bigcap_i (\text{LI}_i)_\omega^t \cap \bigcap_i (\text{LC}_i)_\omega^t \).

**Proof.** It is analogous to the proof of Lemma 1.

**Theorem 2.** The structure \( \mathcal{D}_{\text{sts}[\text{Org}_\omega]} = (\text{sts}[\text{Org}_\omega], \subseteq \text{sts}[\text{Org}_\omega]) \) is a complete lattice, called the domain of the micro-level organization structural states, whose top is \( T_\omega = \bigcup \text{sts}[\text{Org}_\omega] \) and whose bottom is \( \bot_\omega = (\text{RO}_\omega, \text{LI}_\omega, \text{LC}_\omega) \), where \( \text{RO}_\omega = \text{LI}_\omega = \text{LC}_\omega = \emptyset \).

127
Proof. From Lemmas 1 and 2, it follows that $\mathcal{D}_{\text{sts}[\text{ORG}_\omega]}$ is a complete lattice, whose least and greatest elements are $\bot_\omega$ and $\bigcup\text{sts}[\text{ORG}_\omega]$, respectively.

**Definition 9.** A process of organization growth is a function $\text{og} : T \to \wp(\mathcal{D}_{\text{sts}[\text{ORG}_\omega]})$, such that (i) $\bigcup \text{og}[T], \bigcap \text{og}[T] \in \text{og}[T]$, and (ii) for all $t, t'$ with $t \leq t'$ it holds that $\text{og}(t) \subseteq \wp(\text{sts}[\text{ORG}_\omega]) \text{og}(t')$ (i.e., $\text{og}[T]$ is totally ordered).

The set $\bigcup \text{og}[T]$ is the set of stages of the organization growth, where $\bigcup \text{og}[T]$ and $\bigcap \text{og}[T]$ are the initial and terminal stages, respectively. If, for all $t \in T$, $\text{og}(t)$ is a unitary set, then the process of organization growth is said to be sequential; otherwise, it is said to be parallel. The process is said to be proper if each of its stages is a properly implemented state of organization structure.

**Example 4. Sample Processes of Organizational Growth.** We refer to the micro-level organization state of Ex. 2. Suppose that there is a growing sequence of micro-roles starting with $r'_1$, so that $r'_1 \subseteq r''_1 \subseteq r_1$ with $r'_1 = \{b'_1\}$, $r''_1 = \{b_1, b'_1\}$, $r_1 = \{b_1, b'_1, b''_1\}$. Also, suppose that there is growing sequence of micro-links starting with $l'_1 \subseteq l''_1 \subseteq l_1$ so that $l''_1 = (r'_1, r_2, \epsilon''_1), l''_1 = (r''_1, r_2, \epsilon''_1), l''_1 = (r''_1, r_2, \epsilon''_1)$, such that the exchange processes involved in each stage of the micro-link growing sequence are compatible with the roles existing at each stage of the micro-role growing sequence.

The growing sequence of the micro-level organization structures $(\text{ORG}_1)^{l_0}_{1} \subseteq (\text{ORG}_1)^{l_1}_{1} \subseteq (\text{ORG}_1)^{l_2}_{1} \subseteq (\text{ORG}_1)^{l_3}_{1}$, with: $(\text{ORG}_1)^{l_0}_{1} = \bot_\omega, (\text{ORG}_1)^{l_1}_{1} = \{(r'_1, r_2), \{l''_1\}, \{(r'_1, r_2) \to \{l''_1\}\}, (\text{ORG}_1)^{l_2}_{1} = \{(r'_1, r_2), \{l''_1\}, \{(r'_1, r_2) \to \{l''_1\}\} \} = \{(r'_1, r_2), \{l''_1\}, \{(r'_1, r_2) \to \{l''_1\}\} \}$, represents the sequential growth of $(\text{ORG}_1)_0$ in $\mathcal{D}_{\text{sts}[\text{ORG}_\omega]}$.

Also consider the growth of $(\text{ORG}_2)_0$, in the same domain, given by the sequence $(\text{ORG}_2)^{l_0}_{1} \subseteq (\text{ORG}_2)^{l_1}_{1} \subseteq (\text{ORG}_2)^{l_2}_{1} \subseteq (\text{ORG}_2)^{l_3}_{1}$, where: $(\text{ORG}_2)^{l_0}_{1} = \bot_\omega, (\text{ORG}_2)^{l_1}_{1} = \{(r'_4, r_3), \{l''_3\}, \{(r'_4, r_3) \to \{l''_3\}\} \} = \{(r'_4, r_3), \{l''_3\}, \{(r'_4, r_3) \to \{l''_3\}\} \}$, $(\text{ORG}_2)^{l_2}_{1} = \{(r'_4, r_3), \{l''_3\}, \{(r'_4, r_3) \to \{l''_3\}\} \} = \{(r'_4, r_3), \{l''_3\}, \{(r'_4, r_3) \to \{l''_3\}\} \}$, and growing sequences of links, exchange processes and link capabilities analogous to those of $(\text{ORG}_1)_0$.

Finally, consider the sequential growth process $(\text{ORG}_3)^{l_0}_{1} \subseteq (\text{ORG}_3)^{l_1}_{1} \subseteq (\text{ORG}_3)^{l_2}_{1} \subseteq (\text{ORG}_3)^{l_3}_{1}$, where: $(\text{ORG}_3)^{l_0}_{1} = \bot_\omega, (\text{ORG}_3)^{l_1}_{1} = \{(r'_4, r_2), \{l''_3\}, \{(r'_4, r_2) \to \{l''_3\}\} \} = \{(r'_4, r_2), \{l''_3\}, \{(r'_4, r_2) \to \{l''_3\}\} \}$, $(\text{ORG}_3)^{l_2}_{1} = \{(r'_4, r_2), \{l''_3\}, \{(r'_4, r_2) \to \{l''_3\}\} \} = \{(r'_4, r_2), \{l''_3\}, \{(r'_4, r_2) \to \{l''_3\}\} \}$, $(\text{ORG}_3)^{l_3}_{1} = \{(r'_4, r_3), \{l''_3\}, \{(r'_4, r_3) \to \{l''_3\}\} \} = \{(r'_4, r_3), \{l''_3\}, \{(r'_4, r_3) \to \{l''_3\}\} \}$.

At time $t_3$, the three growing sequences may merge to construct the state $(\text{ORG})^{l_3}_{1} = (\text{ORG}_1)^{l_3}_{1} \cup (\text{ORG}_2)^{l_3}_{1} \cup (\text{ORG}_3)^{l_3}_{1}$, illustrating the parallel process that constructs the structural state of $(\text{ORG})_\omega$ shown in Ex. 2.

### 4 Conclusion

Modeling organizational development of multiagent systems is a particular important issue when dealing with open multiagent systems, where agent can freely enter or leave the system.

Of course, the notion of development is naturally a contrived notion, requiring the careful examination of the various conceptual ingredients that constitute
This paper aimed to support the introduction of the notion of organization development in the realm of MAS by formalizing one of its conceptual ingredients, namely, the notion of organization growth.

Organization growth is an elementary form of development where only monotonic increase is allowed, meaning that the process is conservative: that is, at all steps in a growth process only the aggregation of new structural or operational elements occurs, no losses or changes in the nature of the extant elements being allowed. Growth processes of that kind can be neatly modeled in domain-theoretic terms, as previously shown in general terms in [8].

Organization development processes, involving non-monotonic steps that may encompass re-organization procedures, require a more general treatment than plain growth processes, and should be the subject of further study.

The result of the paper is an important step in that direction, as it shows how the domain-theoretic structure underlying such monotonic steps looks like, in the case of MAS organizations.

Acknowledgements. Work partially supported by CNPq (Proc. 307185/07-9, 473201/07-0, 304580/07-4).

References
Taming Implications in Dummett Logic

Guido Fiorino

Dipartimentod iM etodi Quantitativi perl eS cienze Economiche ed Aziendali,
Universit`ad iM ilano-Bicocca, Piazza dell’Ateneo Nuovo, 1, 20126 Milano, Italy.
guido.fiorino@unimib.it

Abstract. This paper discusses a new strategy to decide Dummett logic. The strategy relies on a tableau calculus whose distinguishing features are the rules for implicative formulas. The strategy has been implemented and the experimental results are reported.

Key words: Dummet Logic, Tableau Calculi, Automated Theorem Proving

1 Introduction

The aim of this paper is to provide some ideas to reduce the search space of proofs in Dummett logic. Our results apply when implicative formulas have to be handled. The results are provided in the framework of tableau proof systems and they are explained on the basis of the Kripke semantics for Dummett logic.

The history of this logic starts with Gödel, who studied the family of logics semantically characterizable by a sequence of \( n \)-valued \( (n > 2) \) matrices ([8]). In paper [4] Dummett studied the logic semantically characterized by an infinite valued matrix which is included in the family of logics studied by Gödel and proved that such a logic is axiomatizable by adding to any Hilbert system for propositional intuitionistic logic the axiom scheme \((p \rightarrow q) \lor (q \rightarrow p)\).

Dummett logic has been extensively studied also in recent years for its relationships with computer science ([2]) and fuzzy logics ([9]). To perform automated deduction both tableau and sequent calculi have been proposed. Paper [1] provides tableau calculi having the distinguishing feature that a multiple premise rule for implicative formulas signed with \( F \) is provided. We recall that the sign \( F \) comes from Smullyan ([14, 7]) and labels those formulas that in a sequent calculus occur in the right-hand side of \( \Rightarrow \) (as it is explained in Section 2, the sign \( F \) has a meaning also in terms of Kripke semantics). A tableau calculus derived from those of [1] is provided in paper [5]. Its main feature is that the depth of every deduction is linearly bounded in the length of the formula to be proved.

The approach of [1] characterizing Dummett logic by means the multiple premise rule has been criticized because, from the worst case analysis perspective, there are simple examples of sets of formulas giving rise to a factorial number of branches in the number of formulas in the set. Paper [3] shows how to get rid of the multiple premise rule. New rules are provided whose correctness is strictly related to the semantics of Dummett Logic. These ideas have
been further developed in [10, 11], and in paper [12] a graph-theoretic decision procedure is described and implemented. The approach introduced in [3] has also disadvantages with respect to the multiple premise rule proposed in [1] and these disadvantages have been considered in [6], where also a new version of the multiple premise rule is proposed. This version from a practical point of view can reduce the branching when compared with the original one. Paper [6] also provides an implementation that outperforms the one of [12], thus proving that the approach based on the multiple premise rule of [1] deserves attention also from the practical point of view. As a matter of fact, on the one hand the rules of [3] give rise to two branches at most, on the other hand there are cases of formulas that multiple premise calculi decide with a number of steps lower than the calculi based on [3].

The calculi quoted above have the same kind of rule to treat formulas of the kind \( T((A \rightarrow B) \rightarrow C) \), that is formulas that in a sequent calculus would appear in the left-hand side of \( \Rightarrow \):

\[
\frac{S, T((A \rightarrow B) \rightarrow C)}{S, F(A \rightarrow B), T(B \rightarrow C) | S, TC} \quad \frac{\Gamma, B \rightarrow C \Rightarrow \Delta, A \rightarrow B \quad \Gamma, C \Rightarrow \Delta}{\Gamma, (A \rightarrow B) \rightarrow C \Rightarrow \Delta}
\]

Whatever system is used, it is not considered that in the (sub)deduction starting from \( S, F(A \rightarrow B), T(B \rightarrow C) \), respectively starting from the premise \( \Gamma, B \rightarrow C \Rightarrow \Delta, A \rightarrow B \), if \( F(A \rightarrow B) \) occurs in the set, respectively \( A \rightarrow B \) occurs in the right-hand side of \( \Rightarrow \), then the completeness is preserved also if no rule is applied to \( T(B \rightarrow C) \), respectively to \( B \rightarrow C \) in the left-hand side of \( \Rightarrow \). If \( B \) is an implicative formula this strategy avoids to introduce new branches. An analogous remark applies to the case of a set containing \( F B \) and \( T(B \rightarrow C) \), respectively to a sequent of the kind \( \Gamma, B \rightarrow C \Rightarrow B, \Delta \). In this paper these remarks are developed and a tableau calculus is provided. A complete strategy is presented and the experimental results of the prolog implementation are compared with the prolog implementation of [6].

2 Basic Definitions

We consider the propositional language based on a denumerable set of propositional variables \( \mathcal{PV} \) and the logical connectives \( \neg, \land, \lor, \rightarrow \). In the following, formulas (respectively set of formulas and propositional variables) are denoted by letters \( A, B, C \ldots \) (respectively \( S, T, U \ldots \) and \( p, q, r \ldots \)) possibly with subscripts or superscripts.

From the introduction we recall that Dummett Logic (Dum) can be axiomatized by adding to any axiom system for propositional intuitionistic logic the axiom scheme \((p \rightarrow q) \lor (q \rightarrow p)\) and a well-known semantical characterization of Dum is by linearly ordered Kripke models. In the paper model means a linearly ordered Kripke model, namely a structure \( K = \langle P, \leq, \rho, \models \rangle \), where \( \langle P, \leq, \rho \rangle \) is a linearly ordered set with \( \rho \) minimum with respect to \( \leq \) and \( \models \) is the forcing relation, a binary relation on \( P \times \mathcal{PV} \) such that if \( \alpha \models p \) and \( \alpha \leq \beta \), then \( \beta \models p \).
Hereafter we denote the members of $P$, also called worlds or states, by means of lowercase letters of the Greek alphabet.

The forcing relation is extended in a standard way to arbitrary formulas of the language as follows:

1. $\alpha \models A \land B$ iff $\alpha \models A$ and $\alpha \models B$;
2. $\alpha \models A \lor B$ iff $\alpha \models A$ or $\alpha \models B$;
3. $\alpha \models A \rightarrow B$ iff, for every $\beta \in P$ such that $\alpha \leq \beta$, $\beta \models A$ implies $\beta \models B$;
4. $\alpha \models \neg A$ iff for every $\beta \in P$ such that $\alpha \leq \beta$, $\beta \models A$ does not hold.

We write $\alpha \not\models A$ when $\alpha \models A$ does not hold. It is easy to prove that for every formula $A$ the persistence property holds: If $\alpha \models A$ and $\alpha \leq \beta$, then $\beta \models A$. An element $\beta \in P$ is immediate successor of $\alpha \in P$ iff $\alpha \leq \gamma \leq \beta$ holds, then $\alpha = \gamma$ or $\beta = \gamma$ holds. A formula $A$ is valid in a model $\underline{K} = \langle P, \leq, \rho, \models \rangle$ iff $\rho \models A$.

It is well-known that $\textbf{Dum}$ coincides with the set of formulas valid in all models.

The rules of our calculus $\mathbb{D}$ for $\textbf{Dum}$ are in Figures 1 and 2. $\mathbb{D}$ works on signed formulas, that is well-formed formulas prefixed with one of the signs $\{T, F, P, T, T\}$, and on sets of signed formulas (hereafter we omit the word “signed” in front of “formula” in all the contexts where no confusion arises). Before to give the intuition behind the rules of the calculus the meaning of the signs is provided by the relation realizability ($\triangleright$) defined as follows: Let $\underline{K} = \langle P, \leq, \rho, \models \rangle$ be a model, let $\alpha \in P$, let $H$ be a signed formula and let $S$ be a set of signed formulas. We say that $\alpha$ realizes $H$ (respectively $\alpha$ realizes $S$ and $\underline{K}$ realizes $S$), and we write $\alpha \triangleright H$ (respectively $\alpha \triangleright S$ and $\underline{K} \triangleright S$), if the following conditions hold:

1. $\alpha \triangleright TA$ iff $\alpha \models A$;
2. $\alpha \triangleright TA$ iff $\alpha \triangleright TA$ and if $A \equiv (B \rightarrow C)$, then $\alpha \not\models B$;
3. $\alpha \triangleright TA$ iff $\alpha \triangleright TA$ and if $A \equiv B \rightarrow C$, then there exists $\beta \in P$ such that $\alpha < \beta$ and $\beta \not\models B$;
4. $\alpha \triangleright FA$ iff $\alpha \not\models A$;
5. $\alpha \triangleright Fca$ iff $\alpha \models \neg A$;
6. $\alpha \triangleright Tca$ iff $\alpha \models \neg A$;
7. $\alpha \triangleright S$ iff $\alpha$ realizes every formula in $S$;

By the semantical meaning of the signs it follows that $Fca$ and $Tca$ are synonyms respectively of $T\neg$ and $T\neg\neg$, thus $Fca$ and $Tca$-rules are the rules to treat respectively negated and double negated forced formulas. The $Fca$ and $Tca$-rules are designed taking into account that $\textbf{Dum}$ is characterized by linearly ordered Kripke models. The signs $T$ and $T$ are a specialization of the sign $T$ for implicative formulas. The sign $T$ in front of $B \rightarrow C$ conveys the information that at the present state of knowledge the formulas $B \rightarrow C$ and $B$ are respectively forced and not forced. The sign $T$ in front of $B \rightarrow C$ conveys the information that at the present state of knowledge $B \rightarrow C$ is forced and the information that there exists a future state of knowledge where $B$ is not forced. This information is available in the conclusion $S, F(A \rightarrow B), T(B \rightarrow C)$ of the tableau rule $S, T((A \rightarrow B) \rightarrow C) \vdash S, F(A \rightarrow B), T(B \rightarrow C) \mid S, T C$ of the calculi $[1, 5, 6]$, but it
is not exploited. Analogously for the sequent calculi of [3, 12], where the same information is available in the premise $\Gamma, B \rightarrow C \Rightarrow \Delta, A \rightarrow B$ of the rule
\[
\frac{\Gamma, B \rightarrow C \Rightarrow \Delta, A \rightarrow B}{\Gamma, (A \rightarrow B) \rightarrow C \Rightarrow \Delta}
\]
Hereafter we consider tableau calculi.

The same remarks apply to sequent calculi. None of the above quoted calculi has rules taking into account that if $A \rightarrow B$ is not forced, then $B$ is not forced and thus $B \rightarrow C$ is (locally) forced. The formula $B \rightarrow C$ needs to be treated only when disappear the information about the non-forcing of $B$. Thus, roughly speaking, the main idea of the calculus we are presenting can be summarized as follows: a formula of the kind $T(A \rightarrow B)$ does not need to be treated if there is the information that $A$ is not forced. In this case, if $A$ is of the kind $C \rightarrow D$ a branch is avoided. From a syntactical point of view, the sign $\overline{T}$ in front of $A \rightarrow B$ means that in the set at hand $A$ occurs as a consequent of an $\overline{F} \rightarrow$-formula. The sign $\overline{T}$ in front of $A \rightarrow B$ means that in the set at hand the formula $F(A \rightarrow B)$ occurs. The rules are designed to guarantee that the presence of a formula $\overline{T}(B \rightarrow C)$ in a set $S$ implies that $S$ also contains the formula $\overline{F}(A \rightarrow B)$. Note that $\overline{T} \rightarrow \overline{F}$ is the only rule of $\overline{D}$ to introduce $\overline{T}$-formulas. The presence of the formula $\overline{T}(A \rightarrow B)$ in a set $S$ which is a conclusion of the rule $\overline{F} \rightarrow$, implies that also $\overline{F}B$ is in the set. The rules of the calculus behaves on $\overline{T}(A \rightarrow B)$ as they were two premise rules on the formulas $\overline{F}A, \overline{T}(A \rightarrow B)$. The calculus has not rules to treat the $\overline{T}$-formulas. These formulas are treated by the rule $\overline{F} \rightarrow$ and they can be left unchanged or turned into $\overline{T}$-formulas. It is remarked that $\overline{T}$ and $\overline{T}$ are in front of implicative formulas only.

From the meaning of the signs we get the conditions that make a set of formulas inconsistent. A set $S$ is inconsistent if one of the following conditions holds:

\[
\begin{align*}
-\{TA, FA\} \subseteq S; & \quad -\{\overline{TA}, FA\} \subseteq S; & \quad -\{TA, FcA\} \subseteq S; \\
-\{TA, FcA\} \subseteq S; & \quad -\{\overline{TA}, FA\} \subseteq S; & \quad -\{TA, FcA\} \subseteq S; \\
-\{TA, T(A \rightarrow B)\} \subseteq S; & \quad -\{FcA, Tc\alpha\} \subseteq S.
\end{align*}
\]

It is easy to prove the following

**Proposition 1.** If a set of formulas $S$ is inconsistent, then for every Kripke model $\overline{K} = (\overline{P}, \leq, \varphi, \models)$ and for every $\alpha \in P$, $\alpha \nmid S$.

A proof table (or proof tree) for $S$ is a tree, rooted in $S$ and obtained by the subsequent instantiation of the rules of the calculus. The premise of the rules are instantiated in a duplication-free style: in the application of the rules we always consider that the formulas in evidence in the premise are not in $S$. We say that a rule $R$ applies to a set $U$ when it is possible to instantiate the premise of $R$ with the set $U$ and we say that a rule $R$ applies to a formula $H \in U$ (respectively the set $\{H_1, \ldots, H_n\} \subseteq U$) to mean that it is possible to instantiate the premise of $R$ taking $S$ as $U \setminus \{H\}$ (respectively $U \setminus \{H_1, \ldots, H_n\}$).

A closed proof table is a proof table whose leaves are all inconsistent sets. A closed proof table is a proof of the calculus and a formula $A$ is provable iff there exists a closed proof table for $\{FA\}$. 

133
Fig. 1. The invertible rules of \( \mathcal{D} \).

\[
\begin{array}{c}
S, T(A \land B) \quad \frac{S, \top (A \land B)}{T} \\
S, TA, TB \quad \frac{S, F(A \land B)}{F} \\
S, T(A \lor B) \quad \frac{S, FA|S, FB}{S, F(A \lor B)}
\end{array}
\]

\[
\begin{array}{c}
S, T_{cl}(A \land B) \quad \frac{SF(A \land B)}{S,F_{cl}(A \land B)} \\
S, T_{cl}(A \lor B) \quad \frac{SF(A \lor B)}{S,F_{cl}(A \lor B)}
\end{array}
\]

\[
\begin{array}{c}
S, F(A \rightarrow B) \quad \frac{S, F_{e}(A \rightarrow B)}{S, T_{cl}(A \rightarrow B)} \\
S, T((A \land B) \rightarrow C) \quad \frac{S, T_{cl}A|S, TB}{S, T_{cl}(A \rightarrow C), T(B \rightarrow C)}
\end{array}
\]

\[
\begin{array}{c}
S, T_{cl}A \quad \frac{S,T((A \land B) \rightarrow C)}{T} \\
S, T(A \rightarrow (B \rightarrow C))|S, T((B \rightarrow (A \rightarrow C)) \\
S, T(-A \rightarrow B) \quad \frac{S, T((A \lor B) \rightarrow C)}{S, T((A \rightarrow B) \rightarrow C)}
\end{array}
\]

\[
\begin{array}{c}
S, T(A \rightarrow C) \quad \frac{S,T(A \rightarrow B), T(B \rightarrow C)}{T} \\
S, T((A \rightarrow B) \rightarrow C) \quad \frac{S, T(A \rightarrow B))|S, TA|T(A \rightarrow C), T(B \rightarrow C)}{T}
\end{array}
\]

Fig. 2. The non-invertible rules of \( \mathcal{D} \).

\[
\begin{array}{c}
S, T_{cl}A \quad \frac{S,T_{cl}A}{T_{cl-Atom}} \\
S_{c} = \{ TA|TA \in S \} \cup \{ F_{cl}A|F_{cl}A \in S \} \cup \{ TA|T_{cl}A \in S \} \cup \\
\{ TA|T_{cl}A \in S \} \cup \{ TA|T_{cl}A \in S \};
\end{array}
\]

\[
\begin{array}{c}
S_{e} = \{ TA|TA \in S \} \cup \{ F_{cl}A|F_{cl}A \in S \} \cup \{ T_{cl}A|T_{cl}A \in S \} \cup \\
\{ TA|T_{cl}A \in S \} \cup \{ TA|T_{cl}A \in S \} \cup C = B_{i}; \cup \\
\{ T_{cl}A|T_{cl}A \in S \} \cup \{ T_{cl}A|T_{cl}A \in S \} \cup C \neq B_{i}; \cup \\
\{ TA|T_{cl}A \in S \} \cup \{ TA|T_{cl}A \in S \} \cup C \neq B_{i}; \cup \\
\{ TA|T_{cl}A \in S \} \cup \{ TA|T_{cl}A \in S \} \cup C \neq B_{i}; \cup \\
\{ TA|T_{cl}A \in S \} \cup \{ TA|T_{cl}A \in S \} \cup C \neq B_{i}; \cup \\
\{ TA|T_{cl}A \in S \} \cup \{ TA|T_{cl}A \in S \} \cup C \neq B_{i}; \cup
\end{array}
\]

The set \( U = \{ T(B \land C), T(A \land C), F(A \lor B) \} \) put in evidence that both the rule \( \top \land \) (taking \( S = \{ T(A \land C), F(A \lor B) \} \) or \( S = \{ T(B \land C), F(A \lor B) \} \) and the rule \( F \lor \) apply to \( U \). This gives rise to three choices to go on with the proof. After the choice is made, if there is no way to prove the conclusion of the
application of the rule, then the question is if another choice had given a proof. If the rule is invertible, then there is no need to backtrack on another rule: A rule is invertible iff if there exists a proof for the premise, then there exists a proof for the conclusion. The notion of invertible rule is also definable via semantics: a rule is invertible iff if one of the sets in the conclusion is realizable by a model $K$, then the premise is realizable by $K$. It is well-known that the invertibility of the rules of the calculus is a desirable property, since it implies that every choice is deterministic. The calculus $\mathbb{D}$ has two non-invertible rules, namely $F \to$ and $T_{cl}$-Atom. In Section 4 we present a complete strategy such that every choice is deterministic. The strategy relies on respecting a particular sequence in the application of the rules: $T_{cl}$-Atom is applied if no other rule is applicable and $F \to$ is applied if no other rule but $T_{cl}$-Atom is applicable.

3 Correctness

The following proposition states that the rules in Tables 1 and 2 preserve the realizability. This is the main step towards to prove the correctness of $\mathbb{D}$.

**Lemma 1.** For every rule of $\mathbb{D}$, if a world $\alpha$ of a model $K = \langle P, \leq, \rho, \models \rangle$ realizes the premise, then there exists a world of a possibly different model realizing at least one of the conclusions.

**Proof.** The proof proceeds by taking into account every rule of $\mathbb{D}$. Here the proof of the correctness of rule $T \rightarrow \rightarrow$ is provided. Rule $T \rightarrow \rightarrow$: by hypothesis $\alpha \models S, T((A \to B) \to C)$. By definition of forcing of implication we have two cases: (i) $\alpha \models C$, thus $\alpha \models S, TC$; (ii) $\alpha \not\models (A \to B)$. This implies that $\alpha \models B \to C$ and there exists a world $\beta \in P$ such that $\alpha \leq \beta, \beta \models A$ and $\beta \not\models B$. If $\alpha < \beta$, then immediately we get $\alpha \models F(A \to B), T(B \to C)$. If $\alpha = \beta$ let us consider the model $K' = \langle P \cup \{\alpha'\}, \leq', \rho, \not\models' \rangle$ defined as follows:

$$\leq' = \leq \cup \{(\gamma, \alpha') | \gamma \in P \text{ and } \gamma \leq \alpha \} \cup \{(\alpha', \gamma) | \gamma \in P \text{ and } \alpha < \gamma \};$$

$$\not\models' = \not\models \cup \{(\alpha', p) | (\alpha, p) \in \not\models \}.$$

The model $K'$ is obtained from $K$ by adding a new world $\alpha'$ as immediate successor of $\alpha$ and defining the forcing in $\alpha'$ as the forcing in $\alpha$. By structural induction it is easy to prove that in $K'$ the worlds $\alpha$ and $\alpha'$ force the same formulas. Moreover $\alpha \not\models' A$ holds iff $\alpha \not\models A$ holds. Thus the world $\alpha$ of $K'$ realizes the premise of the rule $T \rightarrow \rightarrow$. Finally, since $\alpha' \not\models' B$ holds, we get that $\alpha \models T(B \to C)$ holds. \hfill \Box

**Theorem 1.** If there exists a proof table for $A$, then $A$ is valid in Dummett Logic with $\mathbb{D}$.

4 Complete Strategy to Decide Dummett Logic with $\mathbb{D}$

In the following we sketch the recursive procedure $DUM(S)$ using $\mathbb{D}$ to decide $S$: Given a set $S$ of signed formulas, $DUM(S)$ returns either a closed proof table
for \( S \) or NULL (if there exists a model realizing \( S \)). To describe DUM we use the following definitions and notations. We call \( \alpha \)-rules (respectively \( \beta \)-rules) the rules of Figure 1 with one conclusion (respectively with two conclusions). The \( \alpha \)-formulas (respectively \( \beta \)-formulas) are the kind of the non-atomic signed formulas in evidence in the premise of the \( \alpha \)-rules (respectively \( \beta \)-rules). Let \( S \) be a set of formulas, let \( H \in S \) be an \( \alpha \) or \( \beta \)-formula. With \( \text{Rule}(H) \) we denote the rule corresponding to \( H \) in Figure 1. Let \( S_1 \) or \( S_1 \cup S_2 \) be the nodes of the proof tree obtained by applying to \( S \) the rule \( \text{Rule}(H) \). If \( Tab_1 \) and \( Tab_2 \) are closed proof tables for \( S_1 \) and \( S_2 \) respectively, then \( \frac{\overline{S}}{Tab_1 | \overline{Tab_2} \text{Rule}(H)} \) denotes the closed proof table for \( S \) defined in the obvious way. Moreover, \( \mathcal{R}_i(H) \) 

\( (i = 1, 2) \) denotes the set containing the formulas of \( S_i \) which replaces \( H \). For instance:

\[
\begin{align*}
\mathcal{R}_1(T(A \land B)) &= \{ TA, TB \}, \\
\mathcal{R}_1(T(A \lor B)) &= \{ TA \}, \\
\mathcal{R}_2(T(A \lor B)) &= \{ TB \}.
\end{align*}
\]

In the case of \( \mathcal{F} \rightarrow \) the notation above generalizes. Let \( S_{\mathcal{F} \leftarrow} \) be the set of all the \( \mathcal{F} \rightarrow \) formulas of \( S \). Let \( S_1 \cup \ldots \cup S_n \) the nodes of the proof tree obtained by applying to \( S \) the rule \( \mathcal{F} \rightarrow \). If \( Tab_1 \ldots, Tab_n \) are closed proof tables respectively for \( S_1, \ldots, S_n \), then \( \overline{S^{Tab_1 | \ldots | Tab_n}} \) is the closed proof table for \( S \). \( \mathcal{R}_i(S_{\mathcal{F} \leftarrow}) \) denotes the set of formulas that replace the set \( S_{\mathcal{F} \leftarrow} \) in the \( i \)-th conclusion of \( \mathcal{F} \rightarrow \). For example, given \( S_{\mathcal{F} \leftarrow} = \{ \mathcal{F}(A_1 \rightarrow B_1), \mathcal{F}(A_2 \rightarrow B_2), \mathcal{F}(A_3 \rightarrow B_3) \} \),

\[
\mathcal{R}_2(S_{\mathcal{F} \leftarrow}) = \{ \mathcal{F}(A_1 \rightarrow B_1), TA_2, FB_2, FA_3 \rightarrow B_3 \}.
\]

**FUNCTION DUM (S)**

1. If \( S \) is an inconsistent set, then DUM returns the proof \( S \);
2. If an \( \alpha \)-rule applies to \( S \), then let \( H \) be a \( \alpha \)-formula of \( S \). If \( \text{DUM}((S \setminus \{ H \}) \cup \mathcal{R}_1(H)) \) returns a proof \( \pi \), then DUM returns the proof \( \frac{\overline{S}}{\pi \text{Rule}(H)} \), otherwise DUM returns NULL;
3. If a \( \beta \)-rule applies to \( S \), then let \( H \) be a \( \beta \)-formula of \( S \). Let \( \pi_1 = \text{DUM}((S \setminus \{ H \}) \cup \mathcal{R}_1(H)) \) and \( \pi_2 = \text{DUM}((S \setminus \{ H \}) \cup \mathcal{R}_2(H)) \). If \( \pi_1 \) or \( \pi_2 \) is NULL, then DUM returns NULL, otherwise DUM returns \( \frac{\overline{S}}{\pi_1 | \pi_2 \text{Rule}(H)} \);
4. If the rule \( \mathcal{F} \rightarrow \) applies to \( S \), then let \( S_{\mathcal{F} \leftarrow} = \{ \mathcal{F}(A \rightarrow B) \in S \} \) and let \( n \) be the number of formulas in \( S_{\mathcal{F} \leftarrow} \). If there exists \( i \in \{ 1, \ldots, n \} \), such that \( \pi_i = \text{DUM}((S \setminus S_{\mathcal{F} \leftarrow}) \cup \mathcal{R}_i(S_{\mathcal{F} \leftarrow})) \) is NULL, then DUM returns NULL. Otherwise \( \pi_1, \ldots, \pi_n \) are proofs and DUM returns \( \frac{\overline{S}}{\pi_1 | \ldots | \pi_n F \rightarrow} \);
5. If the rule \( T_{\text{et-Atom}} \) applies to \( S \), then let \( H \) be a \( T_{\text{et-Atom}} \) formula of \( S \). If \( \text{DUM}((S \setminus \{ H \}) \cup \mathcal{R}_1(H)) \) returns a proof \( \pi \), then DUM returns the proof \( \frac{\overline{S}}{\pi T_{\text{et-Atom}}} \), otherwise DUM returns NULL;
6. If none of the previous points apply, then DUM returns NULL.

**END FUNCTION DUM.**

It is useful to remark the following facts: (i) when Step 4 is performed, \( S \) contains atomic formulas, implicational formulas signed with \( \mathcal{F} \) or \( \mathcal{T} \) and implicational formulas of the kind \( S(p \rightarrow B) \), with \( S \in \{ \mathcal{F}, \mathcal{T} \} \). Note that if \( S(p \rightarrow B) \in S \), then \( \mathcal{T}p \notin S \) holds. As a matter of fact, if \( \{ \mathcal{T}(p \rightarrow B), \mathcal{T}p \} \subseteq S \), then \( S \) is incon-
sistent and this case is handled in Step 1; if \{T(p \rightarrow B), Tp\} \subseteq S, then Step 2 is applicable; (ii) when Step 5 is applied the formulas of the kind \(S(p \rightarrow A)\) with \(S \in \{T, \overline{T}\}\) are the only kind of non-atomic formulas in \(S\) and \(Tp \not\in S\); (iii) when Step 6 is applied there is no formula signed with \(T_\text{cl}\) and formulas of the kind \(S(p \rightarrow B)\), with \(S \in \{T, \overline{T}\}\), are the only non-atomic formulas in \(S\) and \(Tp \not\in S\).

The termination of \textsc{Function Dum} is based on the fact that the rules of \(\mathbb{D}\) replace the formulas in evidence in the premise with \textit{simpler formulas}, where \textit{simpler} is based on a measure complexity function. In order to get the completeness of \textsc{Function Dum}, in the following it is proved that given a set of formulas \(S\), if the call \(\textsc{Dum}(S)\) fails to return a proof for \(S\), then from the non-closed tableau there is enough information to build a model \(K = \langle P, \leq, \rho, \models \rangle\) such that \(\rho \triangleright S\).

**Lemma 2 (Completeness).** Let \(S\) be a set of formulas and suppose that \(\text{Dum}(S)\) returns the \textsc{Null} value. Then there exists a Kripke model \(K = \langle P, \leq, \rho, \models \rangle\) such that \(\rho \triangleright S\).

**Proof.** By induction on the number of nested recursive calls.

\textit{Basis:} There are no recursive calls. Then Step 6 has been performed and this implies that \(S\) is not inconsistent (otherwise Step 1 would have been performed) and \(S\) only contains atomic formulas signed with \(T, \overline{T}\) and \(F, \overline{F}\), formulas of the kind \(S(p \rightarrow A)\) with \(S \in \{T, \overline{T}\}\), and \(Tp \not\in S\). Let \(K = \langle P, \leq, \rho, \models \rangle\), where \(P = \{\rho\}\), \(\rho \leq \rho\) and \(\rho \models p\) iff \(Tp \in S\). \(K\) is a model. By considering every possible kind of formula in \(S\), it is easy to prove that \(\rho\) realizes \(S\).

\textit{Step:} Let us assume by induction hypothesis that the proposition holds for all sets \(S^{'}, \text{such that} \text{Dum}(S')\text{ requires less than} n\text{ recursive calls. The proposition is proved to hold for a set} S\text{ requiring} n\text{ recursive calls. All the possible cases where the procedure returns the \textsc{Null} value have to be inspected. Here we provide the case related to the \textsc{Null} instruction performed at Step 4}. Since the \textsc{Null} instruction in Step 4 has been performed, at least a \(\pi_i\) is \textsc{Null}. By induction hypothesis there is a model \(K' = \langle P, \leq', \rho', \models' \rangle\) realizing \((S \setminus S_{F^\infty})_c \cup R_i(S_{F^\infty})\). We define a model \(K = \langle P \cup \{\rho\}, \leq, \rho, \models \rangle\) as follows:

\[
P \cap \{\rho\} = \emptyset; \leq \leq' \cup \{(\rho, \alpha)|\alpha \in P\}; \models \models' \cup \{(\rho, p)|Tp \in S\}.
\]

Since \(\langle P, \leq', \rho'\rangle\) is a linear order, then, by construction, \(\langle P, \leq, \rho\rangle\) is a linear order (note that \(\rho'\) is the only immediate successor of \(\rho\)). The forcing relation is preserved since the formulas of the kind \(Tp \in S\) are in \((S \setminus S_{F^\infty})_c\) and by hypothesis the minimum \(\rho'\) of \(K'\) realizes \((S \setminus S_{F^\infty})_c\). Since the world \(\rho'\) of \(K'\) realizes \(R_i(S_{F^\infty})\) it follows that the world \(\rho\) of \(K\) realizes \(S_{F^\infty}\). To prove that \(\rho\) realizes \(S\) the main task is to prove that \(\widehat{T}\) and \(\overline{T}\)-formulas are realized. If a formula of the kind \(\widehat{T}(B_i \rightarrow C)\) \(\in S\), then \(\widehat{T}(B_i \rightarrow C) \in (S \setminus S_{F^\infty})_c\). By induction hypothesis \(\rho' \triangleright \widehat{T}(B_i \rightarrow C)\), thus \(\rho' \models A \rightarrow B_i\) and \(\rho' \not\models B_i\), and this implies \(\rho \triangleright \widehat{T}(B_i \rightarrow C)\). If a formula of the kind \(\widehat{T}(B_j \rightarrow C)\) \(\in S\), with \(i \neq j\), then \(\widehat{T}(B_j \rightarrow C) \in (S \setminus S_{F^\infty})_c\). By induction hypothesis \(\rho' \triangleright \widehat{T}(B_j \rightarrow C)\), the idea is to prove that \(\rho\) realizes \(\widehat{T}(B_j \rightarrow C)\) in \(S\).
thus $\rho' \models B_j \rightarrow C$ and $\rho' \not\models B_j$ hold. By the semantical meaning of $\hat{T}$ it follows $\rho \models \hat{T}(B_j \rightarrow C)$. If $\hat{T}(A \rightarrow B) \in S$, then $\hat{T}(A \rightarrow B) \in (S \setminus S_{\mathcal{F}_-})_c$ with $A$ atomic and $\mathcal{T}A \notin S$. By construction of $K$, $\rho \not\models A$. Since by induction hypothesis $\rho' \models A \rightarrow B$ we have $\rho \models A \rightarrow B$ and by the meaning of $\mathcal{T}$ we conclude $\rho \models \hat{T}(A \rightarrow B)$ holds. 

\begin{theorem}{Completeness} \label{completeness}\end{theorem}
If $A$ is valid in every model, then $\text{Dum} \left( \{ FA \} \right)$ returns a proof.

\section{The Implementation and the Performances}

The ideas presented in this paper have been integrated in the implementation for EPDL of [6], the result is a new prototype prover for Dummett logic called Dummett Logic Solver for Implications (DLSI). Cause lack of space, in previous sections the focus has been given to the main idea. There are simple improvements that can be applied to the presentation. Note that in the leftmost conclusion of the rule $\mathcal{T} \rightarrow \rightarrow$ the subformula $B$ occurs twice. This is a source of inefficiency since there can be deduction of exponential depth. Using the well-known indexing technique consisting in replacing a formula with new propositional variable (adopted also in [5, 6, 12]) the result is a calculus whose deductions have depth linearly bounded in the size of the formula to be proved. In [6] a sequence of optimizations is described. Among them a new version of the multiple premise rule of [1] is provided. To simplify the presentation the multiple premise rule of [1] is adopted. DLSI and EPDL differ for the logical rules, neither new optimizations nor code optimization has been performed. To compare EPDL and DLSI some experiments have been carried out. The formulas of ILTP library of paper [13] have been considered. The goal of the calculus is to treat efficiently the case of $B$ non-atomic when formulas of the kind $\mathcal{T}((A \rightarrow B) \rightarrow C)$ occur in the proofs. This never happen with the formulas of ILTP library. Thus the substitution consisting in replacing every propositional variable $p_i$ with $q_i \rightarrow (r_i \rightarrow t_i)$ has been applied to every formula of the ILTP library. Experiments have been performed on the formulas resulting by applying this substitution. Figure 3 shows those family formulas on which the performances of EPDL and DLSI differ. The results show that DLSI outperforms EPDL. Moreover on every family, the timings of DLSI increase of a lower factor than EPDL. On the missing family formulas the timings of EPDL and DLSI are comparable. Finally, Figure 4 gives an account of the comparison between EPDL and DLSI on 10000 randomly generated formulas. It is reported the number of formulas solved respectively within 1, 10, 100, 600 and requiring more than 600 seconds and in parenthesis the seconds required to decide all the formulas (i.e. EPDL solves 9823 formulas within 1 second and the time to decide these 9823 formulas is altogether 204 seconds). Experiments show that the ideas on which the calculus presented in this paper relies improve the known proofs strategies.

\footnote{1 Downloadable from http://www.dimequant.unimib.it/~guidofiorino/dlsi.jsp}

\footnote{2 Timings in seconds, experiments performed on Intel(R) Xeon(TM) 3.00GHz}
<table>
<thead>
<tr>
<th>ILTP Formula</th>
<th>EPDL</th>
<th>DLSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYJ201.1</td>
<td>50.67</td>
<td>0.10</td>
</tr>
<tr>
<td>SYJ201.2</td>
<td>51852</td>
<td>0.05</td>
</tr>
<tr>
<td>SYJ203.7</td>
<td>26.15</td>
<td>0.17</td>
</tr>
<tr>
<td>SYJ203.8</td>
<td>69.97</td>
<td>0.20</td>
</tr>
<tr>
<td>SYJ205.1</td>
<td>123</td>
<td>0.01</td>
</tr>
<tr>
<td>SYJ205.2</td>
<td>13689</td>
<td>17.30</td>
</tr>
<tr>
<td>SYJ207.1</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>SYJ207.2</td>
<td>10724</td>
<td>1.18</td>
</tr>
<tr>
<td>SYJ209.7</td>
<td>18.05</td>
<td>0.18</td>
</tr>
<tr>
<td>SYJ209.8</td>
<td>47.85</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ILTP Formula</th>
<th>EPDL</th>
<th>DLSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYJ210.5</td>
<td>1627</td>
<td>0.03</td>
</tr>
<tr>
<td>SYJ210.6</td>
<td>119803</td>
<td>0.05</td>
</tr>
<tr>
<td>SYJ211.1</td>
<td>1.67</td>
<td>0.02</td>
</tr>
<tr>
<td>SYJ211.2</td>
<td>523</td>
<td>0.03</td>
</tr>
</tbody>
</table>

**Fig. 3.** Timings on ILTP formulas modified with substitution $X \rightarrow (Y \rightarrow Z)$.

<table>
<thead>
<tr>
<th>Prover</th>
<th>0-1secs.</th>
<th>1-10secs.</th>
<th>10-100secs.</th>
<th>100-600secs.</th>
<th>&gt;600</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPDL</td>
<td>9823 (204s.)</td>
<td>134 (491s.)</td>
<td>35 (1123s.)</td>
<td>5 (1424s.)</td>
<td>3 (11007s.)</td>
</tr>
<tr>
<td>DLSI</td>
<td>9843 (216s.)</td>
<td>116 (387s.)</td>
<td>35 (943s.)</td>
<td>4 (1004s.)</td>
<td>2 (6561s.)</td>
</tr>
</tbody>
</table>

**Fig. 4.** Timings on randomly generated formulas.

**References**

Constraint Message Propagation Dynamics of Nonlinear Boolean Equations

Binghui Guo\(^1\), Wei Wei\(^2\) and Zhiming Zheng\(^1\)

\(^1\) LMIB and Department of Mathematics, Beihang University, 100191, Beijing, China
\(^2\) LMIB and School of Mathematical Sciences, Peking University, 100871, Beijing, China

Abstract. We focus on the message passing procedures of a family of nonlinear Boolean equations called MAS-nonlinear in this paper. By mapping MAS-nonlinear to factor graph as a graphical representation, two types of message passing mechanisms, warning propagation and survey propagation are investigated to reveal the dynamical behaviors of the constraint message propagation process. Both of these message propagation procedures are proposed in form of iterative equations and preform bifurcation-like phenomena. Our experimental results exhibit that the constraint message propagation dynamics is evolved into disordered states after some threshold. These are well matched to the phase transition phenomena of generating elements that is supposed to be a more rigorous way to explore the complexity of solution space.

Key words: Message Passing, Boolean Equation, Phase Transition

1 Introduction

Constraint satisfaction problems play an important role across a broad spectrum of complex systems in computer science, statistical physics and information theory [1]. In particular, constraint satisfaction problems are focused on in a wide range of areas in theoretical computer science, including computational coding theory [2], computational complexity [3], artificial intelligence [4] and automated electronic design [1]. Many constraint satisfaction problems are well known as NP-complete problems [5] for the high computational complexity, which cannot be determined whether they are satisfiable or unsatisfiable in a polynomial time in the worst case by any known algorithm. One of the best-known NP-complete problems is the random k-SAT problem, the message passing algorithms and phase transition of which are observed by Mézard et al. [6].

As a novel NP-complete problem, the massive algebraic system (MAS) has been proposed as an interesting random constraint satisfaction problem formulated in terms of both linear and nonlinear Boolean equations in our previous

\* This work is supported by the National Key Basic Research Project of China Grant No.2005CB321902 and the Innovation Foundation of BUAA for PhD Graduates.
work [7]. We indicated that the solution space of MAS undergoes one step replica symmetry breaking and calculated the upper bounds and lower bounds by the moments method and the dynamical analysis of leaf-removing algorithm. Since this problem consists with two different subproblems explicitly, we may consider the subproblems independently as the first step to explore properties of solution space. Although the intersection of the solutions of these two subproblems is difficult to calculate, it is still an important issue to detect the solution-space properties of each subproblem individually to get more information. The first subproblem can be viewed as the random 3-XORSAT problem which has been proved to undergo a phase transition phenomenon in [8]. The solution-space structure of this problem undergoes only replica symmetry and one step replica symmetry breaking [9] in its satisfiable phase. The second subproblem takes the form of the nonlinear Boolean equations, in which the solution space is considered to have more complicated geometric characteristics. The information of detailed properties of MAS-nonlinear may provide a basic solution-space landscape of the original joint problem. In our previous works, MAS-nonlinear was studied from the viewpoint of both statistics and algebra. We suggested that as a key factor the generating elements of solution space go through linear, polynomial and exponential phases [7] while equations density increases, which correspond with the structural complexity.

In this paper, we focus on the message passing procedures of MAS-nonlinear. Message passing algorithms have exhibited important applicabilities in many constraint satisfaction problems. For example, a message passing algorithm survey propagation appears to remain effective at solving very large scale instances of random satisfiability problems even with clause densities very close to the SAT-UNSAT threshold [10]. Here we analyze two types of message passing procedures in a family of nonlinear equations represented by factor graph as graphical model. Firstly, based on the analysis of passing mechanism of different types of the warning message we construct the iterative equations of warning propagation and run the warning propagation algorithm to instances with a great number of variables. Secondly, we analyze the propagation mechanism of probability survey message which is a generalization of warning message in this model and construct the iterative equations for different cases. The results show that both warning propagation and survey propagation perform bifurcation-like phenomena as equation density increases, which match the phase transition phenomena of generating elements that is supposed to be a more rigorous way to explore the statistical properties of solution space. This may explain why the generating elements of solution space are evolved into different phases in different periods.

2 Graphical Model of MAS-nonlinear

Since all NP-Complete problems can be transformed to quadratic polynomial systems by polynomial reductions [11], it is interesting to study the statistical properties of a random system that consists of the simplest quadratic equations. As a formalism of this type of random systems, MAS-nonlinear provides a way
to detect how the computational complexity of searching algorithms increases abruptly when constrains density undergoes some threshold. MAS-nonlinear is defined as follows: Consider a set of $N$ variables $x_1, \ldots, x_N$ taking values in $\mathbb{Z}_2$, $M$ quadratic equations on polynomial ring $\mathbb{Z}_2[x_1, \ldots, x_N]$ taking the form of

$$x_i + x_j \cdot x_k = 0,$$

where the variables of equations are chosen randomly and independently from all the $N$ variables, we define this model MAS-nonlinear and write the ratio of number of equations to number of variables $\alpha = M/N$ which is named equation density.

To express this problem on a graph, a bipartite factor graph with $N$ variable nodes and $M$ functional nodes is discussed in this section as the graphical model formulation of MAS-nonlinear. As a very useful representation in various contexts such as statistical inference and error correcting codes, the representation called factor graph is used. We construct the graph as follows: Each equation is represented by a functional node, connected to the various variables which appear in this equation. It means that for each variable we construct it as a variable node and for every equation we construct it as a functional node. Then we connect variable node $i$ to functional node $a$ if and only if variable $x_i$ appears in equation $f_a$. Also we need to keep track of whether the variables appear in linear part or nonlinear part in the equations. We do this by using two types of edges. Label $J^i_a$ denotes the edge from equation node $a$ to variable node $i$. If variable $x_i$ appears as linear-part variable in equation $f_a$, then $J^i_a = 1$ and the corresponding edge is represented as a real line on the factor graph, otherwise we set $J^i_a = -1$ and the corresponding edge is represented as a dashed line on the factor graph. Let $V(i)$ denote the functional nodes attached to variable node $i$. $V_-$ and $V_+$ denote the functional nodes where $x_i$ appears as linear-part variable and nonlinear-part variable respectively. Similarly we denote $V(a) = V_-(a) \cup V_+(a)$ as the variable nodes that appear in equation $f_a$.

3 Warning Propagation of MAS-nonlinear

Some message passing algorithms which turn out to be remarkably powerful in solving constraint satisfaction problems. The general strategy involves two main steps: The first step is organizing the constraint messages and variables in a factor graph, which has been completed in the previous section. The second step is analyzing the probabilistic mechanism how constraint messages exchange on the graph which will be discussed in this section.

Intuitively speaking, message passing algorithms show passing and updating procedure of messages along edges in the graphical model of complex constraints system (factor graph), i.e. all the information collected from the neighbors of variable $x_i$ is transformed to messages broadcasting to its neighbors again. This process is iterated until the passing procedure converges to a steady state or an equilibrium situation. The basic case of message passing is the warning propagation [10]. Along all the edges of the graph messages pass and update based on
the information from the neighbor environment. In the graphical model of MAS-
nonlinear, firstly, we introduce one type of message called warning $u_{a\rightarrow i}$, which
is sent from a functional node $a$ to a variable node neighbor $i$. It means that
equation $f_a$ involves variable $x_i$ and informs variable $x_i$ a message about which
assignment of variable $x_i$ is preferred to. There are three cases of this warning
message: $u_{a\rightarrow i} = -1(+1)$, which means that the variable $x_i$ should take the
value 0(1) to satisfy equation $f_a$ according to the messages that equation $f_a$ has
received from other variable nodes connected to $a$; $u_{a\rightarrow i} = 0$, which means that
the variable $x_i$ is free according to the messages that equation $f_a$ has received
from other variable nodes connected to $a$.

Secondly, we also introduce another type of message $u_{j\rightarrow a}$, which is sent
from a variable node $j$ to a functional node neighbor $a$ similarly. It represents
the constraint information that $j$ collects from all the neighbors except $a$, i.e.
the cavity field at this variable node in the viewpoint of statistical physics. Since
this type of message denotes the sum of warnings that a variable node receives
from all the neighbors except $a$, we write $u_{j\rightarrow a}$ in a uniform representation

$$u_{j\rightarrow a} = \Theta(\sum_{b\in V(j)/a} u_{b\rightarrow j}),$$

where $\Theta(\sum_{b\in V(j)/a} u_{b\rightarrow j}) = -1(+1)$, if $\sum_{b\in V(j)/a} u_{b\rightarrow j} < (>0)$, which means
that the variable node $j$ has received more negative(positive) warning messages
from other functional node connected to $j$; $\Theta(\sum_{b\in V(j)/a} u_{b\rightarrow j}) = 0$, if
$\sum_{b\in V(j)/a} u_{b\rightarrow j} = 0$, which means that the positive and negative warning messages
from other functional node connected to $j$ are equal. Then we observe
the iterative equations of warning propagation procedure. Considering an equation
$f_a$ involves three variables $x_i, x_j, x_k$ as an example, the warning passing
mechanism is illustrated in figure 1.

![Fig. 1. The left side is the case that $x_i$ appears as linear part of equation $f_a$. The right side is the case that $x_i$ appears as nonlinear part of equation $f_a.$](image)

When the variable $x_i$ is in linear part of $f_a$ (figure 1A), the condition of
warning message value can be analyzed explicitly as follows: $u_{a\rightarrow i} = -1$, if and
only if at least one of $u_{j\rightarrow a}$ and $u_{k\rightarrow a}$ is assigned to $-1$; $u_{a\rightarrow i} = +1$, if and
only if both \( u_{j\rightarrow a} \) and \( u_{k\rightarrow a} \) are assigned to +1; \( u_{a\rightarrow i} = 0 \), otherwise. One can transform it to a formalism equation

\[
u_{a\rightarrow i} = \delta(u_{j\rightarrow a} - 1)\delta(u_{k\rightarrow a} - 1) - [\delta(u_{j\rightarrow a} + 1) + (1 - \delta(u_{j\rightarrow a} + 1))\delta(u_{k\rightarrow a} + 1)],
\]

where \( \delta(x) = 0 \), if \( x \neq 0 \); \( \delta(x) = 1 \), if \( x = 0 \). After a little algebra, we get

\[
u_{a\rightarrow i} = 2^{-1}(u_{j\rightarrow a}^2 + u_{k\rightarrow a}^2 - u_{j\rightarrow a} - u_{k\rightarrow a} - u_{j\rightarrow a}u_{k\rightarrow a} - u_{j\rightarrow a}^2u_{k\rightarrow a}^2).
\] (3)

When the variable \( x_i \) is in nonlinear part of \( f_a \) and variable \( x_j \) is in the linear part of \( f_a \) (figure 1B), we get

\[
u_{a\rightarrow i} = 2^{-1}(u_{j\rightarrow a}^2 + u_{j\rightarrow a} + u_{j\rightarrow a}u_{k\rightarrow a} - u_{j\rightarrow a}^2u_{k\rightarrow a}^2).
\] (4)

Combining equations (2), (3) and (4), for an arbitrary but fixed edge we construct the iterative equation of the warning propagation procedure on factor graph of MAS-nonlinear. Based on the updating rule discussed above, we use this rule sequentially and get the following warning propagation algorithm:

**Warning Propagation Algorithm**

**INPUT:**
1. The factor graph of an instance of MAS-nonlinear;
2. A maximal number of sweeps \( I_{max} \) and initial values of the warning messages

**OUTPUT:**
CASE 1: UN-CONVERGED
if warning propagation algorithm has not converged after \( I_{max} \) sweeps.

CASE 2: WARNING SET \( \{\hat{u}_{a\rightarrow i}\} \)
if warning propagation algorithm has converged.

**step 1.** At the initial sweep \( t=1 \): randomly initialize the warnings \( u_{a\rightarrow i}(t = 1) \) on every edge \( a \rightarrow i \) of the factor graph with initial values.

**step 2.** Randomly sweep the set of edges and generate the values \( u_{a\rightarrow i}(t) \), using iterative equations (2), (3) and (4). Update sequentially the warnings on all the edges of the factor graph. Let \( t=t+1 \).

**step 3.** If \( t > I_{max} \), go to **step 4**; if \( t \leq I_{max} \), go to **step 5**.

**step 4.** Return UN-CONVERGED.

**step 5.** If \( u_{a\rightarrow i}(t - 1) = u_{a\rightarrow i}(t - 2) \), go to **step 6**; else return to **step 2**.

**step 6.** Return WARNING SET \( \{\hat{u}_{a\rightarrow i} = u_{a\rightarrow i}(t - 1)\} \)

This algorithm is run to solve \( 10^6 \) randomly generated instances with \( 10^4 \) variables for different equation density \( \alpha \) as and initial values. The experimental results exhibit that for different initial values the finally converged warning propagation messages perform a bifurcation-like process as the equation density increases which is shown in figure 2. As \( \alpha > 0.16 \), the constraint warning messages passing through the factor graph is involved into a disordered state when there are cycles on the graph [12]. It also has been proved that \( 1/6 \) is the threshold that giant connected components appear. It provides an explanation why the computational complexity increases as the equation density increase.
Fig. 2. The distributions of the fraction of the converged warning message taking 0 and +1 with different initial values.

4 Survey Propagation of MAS-nonlinear

The warning propagation procedure has given us a landscape of the dynamics how constraint messages pass through the factor graph of MAS-nonlinear in different equation densities. It is clear that warning propagation algorithm has a good performance in the easy-satisfiable phase because we had a way to restrict the uniform measure to the configurations in one given cluster globally. But there is no way to achieve this type of measure globally in the replica symmetry breaking phase case which occurs as the equation density is larger than some threshold. So one has to introduce generalized types of messages from the cavity method in order to handle such a situation. Similarly to the analysis of survey propagation of random SAT problem [10], we construct the survey propagation procedure on the factor graph of MAS-nonlinear.

As a generalization of warning propagation mechanism, there are two main differences between survey propagation and warning propagation: Firstly, the messages do not just take three states, 0, +1 or -1, but are probability distributions. Secondly, the survey propagation considers a wider range of information collected by variable from the surrounding environment. We first introduce another type of message called survey $\eta_{a \rightarrow i} \in [0, 1]$, which is sent from a functional node $a$ to a variable node neighbor $i$. It is the probability that the variable node $i$ receives a constraint message from functional node $a$. There are three types of survey message: $\eta_{a \rightarrow i}^+ \in [0, 1]$ is the probability that equation $f_a$ restricts variable $x_i$ to +1; $\eta_{a \rightarrow i}^- \in [0, 1]$ is the probability that equation $f_a$ restricts variable $x_i$ to 0; $\eta_{a \rightarrow i}^0 \in [0, 1]$ is the probability that equation $f_a$ can not restrict variable $x_i$ to any fixed value. Intuitively speaking, the message passing through one edge consists of all the three types of messages $\eta_{a \rightarrow i}^+, \eta_{a \rightarrow i}^-, \eta_{a \rightarrow i}^0$, which survey and mix the information from the environment.

Meanwhile, we introduce three types of performing messages $\Pi_{j \rightarrow a}^+, \Pi_{j \rightarrow a}^-$, $\Pi_{j \rightarrow a}^0$ that the variable node $j$ is constrained to value +1, constrained to value 0, constrained to joker state which means that $x_j$ is a free variable. In other words, $\Pi_{j \rightarrow a}^+$ is the intensity that variable node $j$ contributes to functional node $a$ as +1; $\Pi_{j \rightarrow a}^0$ is the intensity that variable node $j$ contributes to functional
node \(a\) as any value; \(\Pi_{j \rightarrow a}^{-}\) is the intensity that variable node \(j\) contributes to functional node \(a\) as 0. Then we get the equations of these types of messages:

\[
\Pi_{j \rightarrow a}^{0} = \prod_{b \in V_{a}(j)} \eta_{b \rightarrow a}^{0},
\]

\[
\Pi_{j \rightarrow a}^{+} = \prod_{b \in V_{a}(j)} (1 - \eta_{b \rightarrow a}^{-}) - \prod_{b \in V_{a}(j)} \eta_{b \rightarrow a}^{0},
\]

\[
\Pi_{j \rightarrow a}^{-} = \prod_{b \in V_{a}(j)} (1 - \eta_{b \rightarrow a}^{+}) - \prod_{b \in V_{a}(j)} \eta_{b \rightarrow a}^{0}.
\]

---

**Fig. 3.** The left side is the case that \(x_{i}\) appears as linear part of equation \(f_{a}\). The right side is the case that \(x_{i}\) appears as nonlinear part of equation \(f_{a}\).

Next we construct the iterative equations of survey propagation procedure. We note that the survey message is correlated with that the variable appears as linear or nonlinear part. Considering an equation \(f_{a}\) involves three variables \(x_{i}, x_{j}, x_{k}\) as an example, the survey message passing mechanism is illustrated in figure 3. When the variable \(x_{i}\) is in linear part of \(f_{a}\) (figure 3A), the condition of warning message value can be analyzed explicitly as follows: The survey message from equation \(f_{a}\) to variable \(x_{i}\) is +1, if and only if the other two variables \(x_{k}, x_{j}\) both contribute to \(f_{a}\) as +1s; the survey message from equation from equation \(f_{a}\) to variable \(x_{i}\) is 0, if and only if at least one; there is a joker survey message from equation \(f_{a}\) to \(x_{i}\) in the rest case. Then we transform it to formalism equations

\[
\eta_{a \rightarrow i}^{+} = \frac{\Pi_{j \rightarrow a}^{+}}{\Pi_{j \rightarrow a}^{+} + \Pi_{j \rightarrow a}^{-} + \Pi_{j \rightarrow a}^{0}} \cdot \frac{\Pi_{k \rightarrow a}^{+}}{\Pi_{k \rightarrow a}^{+} + \Pi_{k \rightarrow a}^{-} + \Pi_{k \rightarrow a}^{0}},
\]

\[
\eta_{a \rightarrow i}^{-} = (1 - \frac{\Pi_{j \rightarrow a}^{+}}{\Pi_{j \rightarrow a}^{+} + \Pi_{j \rightarrow a}^{-} + \Pi_{j \rightarrow a}^{0}}) \cdot (1 - \frac{\Pi_{k \rightarrow a}^{+}}{\Pi_{k \rightarrow a}^{+} + \Pi_{k \rightarrow a}^{-} + \Pi_{k \rightarrow a}^{0}})
\]

\[
- \frac{\Pi_{j \rightarrow a}^{0}}{\Pi_{j \rightarrow a}^{+} + \Pi_{j \rightarrow a}^{-} + \Pi_{j \rightarrow a}^{0}} \cdot \frac{\Pi_{k \rightarrow a}^{0}}{\Pi_{k \rightarrow a}^{+} + \Pi_{k \rightarrow a}^{-} + \Pi_{k \rightarrow a}^{0}},
\]

\[
\eta_{a \rightarrow i}^{0} = 1 - \eta_{a \rightarrow i}^{+} - \eta_{a \rightarrow i}^{-}.
\]
When the variable $x_i$ is in nonlinear part of $f_a$ and variable $x_j$ is in the linear part of $f_a$ (figure 3B), the condition of warning message value can be analyzed explicitly as follows: The survey message from $f_a$ to variable $x_i$ is +1, if and only if $x_j$ has a performing message to $f_a$ as 1 and $x_k$ should not have any performing message to $f_a$ as 0 (else contradiction will occur); the survey message from $f_a$ to variable $x_i$ is 0, if and only if $x_j$ has a performing message to $f_a$ as 0 and $x_k$ has a performing message to $f_a$ as +1. the rest case, except the case that $x_j$ has a performing message to $f_a$ as +1 and $x_k$ has a performing message to $f_a$ as 0 (contradicted case), there is a joker survey message from $f_a$ to $x_i$.

By introducing $\{M_{a-i}^0, M_{a-i}^+, M_{a-i}^\ast\}$ as the normalization to ensure that $\eta_{a-i}^0, \eta_{a-i}^+, \eta_{a-i}^\ast$ are the corresponding survey messages probabilities ($\eta_{a-i}^0 + \eta_{a-i}^+ + \eta_{a-i}^\ast = 1$) as

$$
M_{a-i}^+ = \frac{\Pi_j^{+} \cdot \Pi_k^{+} + \Pi_j^{-} + \Pi_j^{0}}{\Pi_j^{+} \cdot \Pi_k^{+} + \Pi_j^{-} + \Pi_j^{0}},
$$

$$
M_{a-i}^- = \frac{\Pi_j^{-} \cdot \Pi_k^{-} + \Pi_j^{+} + \Pi_j^{0}}{\Pi_j^{+} \cdot \Pi_k^{-} + \Pi_j^{-} + \Pi_j^{0}},
$$

$$
M_{a-i}^0 = \frac{1}{\Pi_j^{+} \cdot \Pi_k^{+} + \Pi_j^{-} + \Pi_j^{0}} \cdot (\Pi_j^{0} + \Pi_j^{-} + \Pi_j^{0}).
$$

Let $M_{a-i} = M_{a-i}^+ M_{a-i}^- + M_{a-i}^0$, we get

$$
\eta_{a-i}^+ = \frac{M_{a-i}^+}{M}, \quad \eta_{a-i}^- = \frac{M_{a-i}^-}{M}, \quad \eta_{a-i}^0 = \frac{M_{a-i}^0}{M}. \quad (11)
$$

By the iterative equations (5–11), all the surveys $\eta_{a-i}^0, \eta_{a-i}^+, \eta_{a-i}^\ast$ can evolve on the factor graph of MAS-nonlinear. If this procedure has reached convergence states, one can use the fixed points of these surveys $\eta_{a-i}^0, \eta_{a-i}^+, \eta_{a-i}^\ast$ in a decimation procedure in order to find a solution. The survey propagation mechanism indeed relies on the clustering of the solution space, which is the emergence of pure phase (a cluster of solutions). Considering one pure phase $\beta$ and an edge $(a, i)$ of the factor graph, if in every solution of cluster $\beta$, the variable nodes $j, k \in V(a)/i$ do not satisfy functional node $a$, then a warning $u_{a-i}^\beta = 1$ is passed along the edge from $a$ to $i$ in survey propagation. The survey propagation message along this edge is the probability distribution of these warnings of different clusters when one picks up a cluster $\beta$ randomly: $\eta_{a-i} = \sum_{\beta} u_{a-i}^{\beta}$. Indeed, the survey message is the average messages of different pure phases, and when the whole solution space is only in one pure phase, the survey message is just the warning message. So basically the survey message in a randomly chosen cluster possesses the form of probability distribution.

This survey propagation algorithm is run in $10^6$ randomly generated instances with $10^3$ variables for equation density $\alpha = 0.1, 0.2, \ldots, 1$ by population dynamics. The distribution performs a bifurcation-like process as the equation density increases. Here we just illustrate $\eta_{a-i}^0$ as an example in figure 4. Compare with the analytic results of generating elements of the solution space of
MAS-nonlinear [12], we suggest there are several thresholds which the distribution of survey propagation messages undergoes. The phase transition phenomena may probably be a refined picture of the solution set of nonlinear Boolean equations and could be exploited in the design of efficient algorithms to solve some types of complex systems. Let 1 − t be the fraction of cavity biases that are trivial, we can obtain nontrivial solutions of t when $\alpha > 0.25$ from the equation $t = 1 - \exp(-3\alpha + 3\alpha(1 - t)(1 - \frac{1}{3}t))$, which means the ergodicity of the finite flipping among the solution breaks. Since the theoretical phase transition point from replica symmetry to one-step replica symmetry is strictly at a critical value $\alpha_d = 0.25$, the distribution of the converged set of the survey propagation algorithms is single-point distribution (trivial) when $\alpha_d \leq 0.25$ and two-points distribution when $\alpha_d < 0.25$. Indeed, in the survey propagation of the cavity bias this could be an evidence that the whole solution space breaks into dynamically isolated regions. It is considered that another threshold $\alpha \approx 0.33$ which indicates the phase transition of the instability of one-step replica symmetry breaking. The results that the distribution of the converged set of the survey propagation algorithm is disordered, which is matched to the conjecture of the analysis of complexity of MAS-nonlinear well [12].

![Fig. 4. The spectral distribution of the converge set $\{\hat{\eta}_{\alpha \rightarrow i}^0\}$ of the survey propagation algorithm when the equation density $\alpha = 0.1, \ldots, 1$.](image)

5 Conclusion

Two types of message passing procedures, warning propagation and survey propagation on factor graph of MAS-nonlinear, are studied in this paper. The iterative equations of the propagation mechanism of warning messages from variable nodes to functional nodes and messages from functional nodes to variable nodes is proposed to manifest the basic mode of constraint message propagation. The dynamics of warning propagation performs a bifurcation-like process and is evolved into disordered states after the threshold corresponding to the
emergency point of giant connected components. Furthermore, the survey propagation on the factor graph is studied as a generalization of warning propagation. The survey message is considered as a survey of the elementary messages in various pure phases from the viewpoint of statistical physics, which is characterized by a probability distribution. It is observed experimentally that survey propagation also performs bifurcation-like phenomena as equation density increases, which matches the phase transition phenomena of generating elements that is supposed to be a more rigorous way to explore the statistical properties and geometrical organization of solution space. The results is an explanation of why the generating elements of solution space are evolved into different phases, when the equation density undergoes different thresholds.

In the future work, it is interesting to develop a mathematical analysis of the structure of the disordered states that warning and survey messages converge to. We conjecture that the disordered converged set is possibly chaotic for the dynamics of message propagation. In addition, it is expected to introduce new methods from symbolic computation and dynamical system to detect the relation between the constraint message propagation and generating elements of solution space and analyze the complexity of the solution space of random nonlinear Boolean systems more explicitly.

References

Function Operators Spanning
the Arithmetical and the Polynomial Hierarchy
– Extended Abstract –

Armin Hemmerling

Ernst-Moritz-Arndt–Universität Greifswald, Institut für Mathematik und Informatik
D–17487 Greifswald, Germany

Abstract. A modified version of the classical \( \mu \)-operator as well as the
first value operator and the operator of inverting unary functions, applied
in combination with the composition of functions and starting from the
primitive recursive functions, generate all arithmetically representable
functions. Moreover, the nesting levels of these operators are closely re-
lated to the stratification of the arithmetical hierarchy. Related results
are shown for some further operators known from computability and
complexity theory.
The close relationships between nesting levels of operators and the strat-
ification of the hierarchy also hold for suitable restrictions of the oper-
ators with respect to the polynomial hierarchy if one now starts with
the polynomial-time computable functions. For example, it follows that
questions around P vs. NP and NP vs. coNP can equivalently be ex-
pressed by closure properties of function classes under these operators.

Key words: Arithmetical hierarchy, polynomial hierarchy, P vs. NP,
NP vs. coNP, minimalization, first value operator, inversion of functions

This extended abstract presents the main results of [7]. All details of proofs
as well as some more motivations and accompanying results can be found there.

In the following two sections, we deal with computability-theoretic variants
of operators and their relationships to the arithmetical hierarchy. This gives the
background and some motivation of the complexity-theoretic setting and results
concerning polytime computability which are the main goal of our investigations
and will be presented in sections 3-5.

1 Nesting levels of function operators

By Kleene’s Normal Form Theorem, every computable partial number-theoretic
function \( f : \mathbb{N}^m \rightarrow \mathbb{N} \) can be represented in the form \( f = h \circ \mu(g) \) with suitable
primitive recursive functions \( h \) and \( g \). On the other hand, the set \( \text{FCom} \) of all
computable (partial, number-theoretic) functions is closed under the \( \mu \)-operator
of minimalization. \( \mu \) assigns to any \((m + 1)\)-ary function \( g \) an \( m \)-ary function
\( f = \mu(g) \), where

\[
f(x) \simeq \begin{cases} x \quad \text{if } g(y_x, x) = 0, \text{ and } g(y, x) \downarrow > 0 \text{ for all } y < y_x, \\
\uparrow \quad \text{if there is no } y_x \text{ of the above kind.}
\end{cases}
\]
What happens if the operator of brutal (or lazy) minimalization, $\mu$, is considered instead of $\mu$? Now let the $m$-ary function $f = \mu(g)$ be simply defined by

$$f(x) \simeq \min\{y : g(y, x) = 0\}, \text{ where } \min\emptyset \simeq \uparrow.$$

If $g$ is a total function, then obviously $\mu(g) = \mu(g)$ For $g(0, x) \uparrow$, it follows that $\mu(g)(x) \uparrow$, whereas $\mu(g)(x)$ might still be defined however.

We shall see that $\mu$ is closely related to the first value operator $\phi$. This leads from a function $g$ of an arity $m + 1$ to the $m$-ary function $f = \phi(g)$ defined by

$$f(x) \simeq \begin{cases} g(y_x, x) & \text{if } y_x = \min\{y \in \mathbb{N} : g(y, x) \downarrow\} \text{ and this set is nonempty,} \\ \uparrow & \text{if } g(y, x) \uparrow \text{ for all } y \in \mathbb{N}. \end{cases}$$

A more general version of $\phi$ was used in [3] in order to characterize the classes of the Ershov hierarchy, cf. [4, 5]. In [6], we introduced the denotation first value operator and used it in order to establish hierarchies of function classes. For total functions $g$, it always holds $\phi(g)(x) = g(0, x)$. So the operator $\phi$ becomes only interesting if it is applied to properly partial functions.

The operator of inversion of unary functions was known from the early days of computability theory, cf. [10]. Nowadays it has got considerable importance within structural complexity and cryptology, cf. [12]. Here we consider the operator $\vartheta$ which to any function $g : \mathbb{N} \rightarrow \mathbb{N}$ assigns a unary function $f = \vartheta(g)$, the reverse (also readable as regular in reverse) of $g$, defined by

$$f(y) \simeq \min\{x : g(x) = y\}, \text{ where } \min\emptyset \simeq \uparrow.$$

Let $FPrim$ denote the set of all primitive recursive functions, and $FPaPrim$ be the set of all partial primitive recursive functions. The latter ones are the restrictions of primitive recursive (total) functions to primitive recursive domains. By $\text{Clos}_{\{\omega_1, \omega_2, \ldots\}}(\text{FC})$ we denote the closure of a function class FC under the operators $\omega_1, \omega_2, \ldots$. This is the smallest class which includes FC and is closed under all $\omega_i$. Thus,

$$\text{Clos}_{\{\omega, \mu\}}(\text{FPrim}) = \text{Clos}_{\{\omega, \mu\}}(\text{FPaPrim}) = \text{FCom} = \text{Clos}_{\{\omega, \mu\}}(\text{FCom}).$$

In order to estimate the power of $\omega$ for $\omega \in \{\mu, \phi, \vartheta\}$, one could ask for a characterization of the sets $\text{Clos}_{\{\omega, \omega\}}(\text{FC})$ with $\text{FC} \in \{\text{FPrim}, \text{FPaPrim}, \text{FCom}\}$. To get more detailed results, we here consider the nesting degrees of functions with respect to the operators $\omega$. This technique goes back to the early sixties of the past century when degrees of (primitive recursive) functions with respect to the operator of primitive recursion were studied. Related results and references can be found in [13, 1]. In order to avoid confusions with the Turing or other degrees, throughout this paper we shall denote the nesting degrees (with respect to operators $\omega$) as $(\omega^\dagger)$-levels.

For a class $\text{FC}$ of number-theoretic functions and an arbitrary operator $\omega$, let $\omega(\text{FC})$ denote the set of all functions obtained by applying the operator $\omega$ exactly once to arguments from FC. In particular, if $\omega$ is unary, as usual in the
cases we shall deal with, $\omega(\text{FC})$ is just the image set of FC under $\omega$. The $\omega$-levels are the following function classes $\text{FLev}_\omega(k)$:

$$\begin{align*}
\text{FLev}_\omega(0) &= \text{FPaPrim}, \\
\text{FLev}_\omega(k+1) &= \text{Clos}_{\{\circ, \omega\}}( \text{FLev}_\omega(k) \cup \omega(\text{FLev}_\omega(k)) ) \quad \text{for all } k \in \mathbb{N}.
\end{align*}$$

This means that $\text{FLev}_\omega(k)$ contains just the functions obtained from $\text{FLev}_\omega(0) = \text{FPaPrim}$ by applying the composition and the operator $\omega$, where applications of $\omega$ are nested at most $k$ times. It holds $\text{Clos}_{\{\circ, \omega\}}( \text{FPaPrim} ) = \bigcup_{k=1}^{\infty} \text{FLev}_\omega(k)$ and $\text{FLev}_\omega(k) \subseteq \text{FLev}_\omega(k+1)$.

By Kleene’s Normal Form Theorem, $\text{FLev}_\mu(k) = \text{FCom}$ for all $k \geq 1$. In the next section we shall see that $\bar{\mu}, \phi$ and $\sigma_{\text{IDom}}$ yield more interesting levels. Now we first show that these three operators lead to the same hierarchy of levels.

**Proposition 1.** If $\omega_1, \omega_2 \in \{ \bar{\mu}, \phi, \sigma_{\text{IDom}} \}$, then $\text{FLev}_{\omega_1}(k) = \text{FLev}_{\omega_2}(k)$ for all $k \in \mathbb{N}$.

This is shown by induction on $k$. The initial step for $k = 0$ is trivial. To prove that $\text{FLev}_{\omega_1}(k+1) \subseteq \text{FLev}_{\omega_2}(k+1)$ follows from $\text{FLev}_{\omega_1}(k) = \text{FLev}_{\omega_2}(k)$, it is enough to show that $\omega_1(\text{FLev}_{\omega_1}(k)) \subseteq \text{Clos}_{\{\circ, \omega_2\}}( \text{FLev}_{\omega_1}(k) \cup \omega_2(\text{FLev}_{\omega_1}(k)) )$ if the second class is equal to $\text{Clos}_{\{\circ\}}( \text{FLev}_{\omega_1}(k) \cup \omega_2(\text{FLev}_{\omega_1}(k)) )$. This inclusion follows since each operator $\omega_1$ can be expressed by $\circ$ and $\omega_2$, where no nesting of $\omega_2$ is needed. Only some partial primitive recursive functions have to be employed in addition. More precisely, for $\omega_1, \omega_2 \in \{ \bar{\mu}, \phi, \sigma_{\text{IDom}} \}$ and every function class FC which is closed under composition and includes the partial primitive recursive functions, we have

$$\omega_1(\text{FC}) \subseteq \text{Clos}_{\{\circ\}}( \text{FC} \cup \omega_2(\text{FC}) ).$$

For details of the proof, the reader is referred to [7].

Now we shall simply write $\text{FLev}(k)$ instead of $\text{FLev}_\omega(k)$ with $\omega \in \{ \bar{\mu}, \phi, \sigma_{\text{IDom}} \}$.

## 2 How nesting levels span the arithmetical hierarchy

The *arithmetical hierarchy* is usually considered both as the hierarchy of the classes $\Sigma_k$, $\Pi_k$ and $\Delta_k$, for $k \in \mathbb{N}$, but also as the union over all these classes:

$$\text{AH} = \bigcup_{k=0}^{\infty} \Sigma_k.$$  

Here $\Sigma_k$ contains just those relations $A \subseteq \mathbb{N}^m$, $m \geq 1$, which are representable in the form

$$A = \{ x \in \mathbb{N}^m : \exists y_1 \forall y_2 \ldots Q y_k(y_1, \ldots , y_k, x) \in B \}$$

with $B \in \text{Com}$, which is the class of all *computable relations*, and $Q \in \{ \exists, \forall \}$ such that the prefix of quantifiers in the above representation becomes alternating. To prove the following results, we also have to use the known representation of the classes of the arithmetical hierarchy by means of relative computability (via *oracle Turing machines* - OTMs) and the jump operator.
A function $f : \mathbb{N}^n \rightarrow \mathbb{N}$ is called \textit{arithmetically representable} iff its graph,

$$\text{graph}(f) = \{(x, y) : f(x) = y\},$$

belongs to \text{AH}. Let \text{FAH} denote the set of all arithmetically representable functions. The stratification of the arithmetical hierarchy can be transferred to \text{FAH} by considering the classes

$$\text{FS}_{\Sigma_k} = \{f : \text{graph}(f) \in \Sigma_k\}, \text{ for } k \in \mathbb{N}.\]$$

In particular, $\text{FS}_{\Sigma_1} = \text{FCom}$. The special class $\text{FS}_{\Sigma_0}$ is less interesting in the present context.

Since arithmetical representability of functions is hereditary under $\circ$ and $\omega \in \{\mu, \phi, g\}$, we have $\text{Clos}_{\{\circ, \omega\}}(\text{FPaPrim}) \subseteq \text{FAH}$. The converse inclusion holds true, too, and the nonzero levels of functions are closely related to the stratification of the arithmetical hierarchy.

**Proposition 2.** For all $k \in \mathbb{N}$, $\text{FLev}(1 + 2k) = \text{F}_{\Sigma_{1+k}}$.

The proof is by induction on $k$, by means of standard techniques of computability theory. Thus, one step in the arithmetical hierarchy corresponds to two steps in the hierarchy of nesting levels of the operators $\mu$, $\phi$ and $g$.

Besides the relatives of the $\mu$-operator considered so far, five further operators known from computability and complexity theory will be dealt with in the sequel. They all assign to arbitrary $(m+1)$-ary functions $g$ certain $m$-ary functions which are defined as follows:

$$\begin{align*}
\text{lim}(g)(x) &\simeq \left\{ \begin{array}{ll}
z & \text{if there is a } y_0 \in \mathbb{N} \text{ such that } g(y, x) = z \text{ for all } y \geq y_0, \\
\uparrow & \text{if there are no such } z \text{ and } y_0; \\
\end{array} \right. \\
\text{min}(g)(x) &\simeq \min \{g(y, x) : g(y, x) \downarrow\}, \text{ where min}(\emptyset) \simeq \uparrow; \\
\text{max}(g)(x) &\simeq \left\{ \begin{array}{ll}
\max\{g(y, x) : g(y, x) \downarrow\} & \text{if this set is nonempty and finite,} \\
\uparrow & \text{if the above set is empty or infinite;} \\
\end{array} \right. \\
\mathcal{P}(g)(x) &\simeq \left\{ \begin{array}{ll}
\text{card}\{g(y, x) : g(y, x) \downarrow\} & \text{if this set is finite,} \\
\uparrow & \text{if the above set is infinite;} \\
\end{array} \right. \\
\text{sum}(g)(x) &\simeq \left\{ \begin{array}{ll}
\sum_{y \in \mathbb{N}, g(y, x) \downarrow} g(y, x) & \text{if there are only finitely many } y \\
\uparrow & \text{with } g(y, x) \downarrow > 0, \\
\end{array} \right. \\
\end{align*}$$

As usual, let the empty sum take the value 0, i.e., $\text{sum}(g)(x) = 0$ if all $g(y, x)$ are undefined. The limit operator is usually restricted to total functions $g$. Our results remain valid, however, if we allow the limit of partial functions, too.

It turns out that the nesting levels of the operators introduced so far span the arithmetical hierarchy in several typical ways. In particular, $\text{lim}$, $\text{max}$, $\mathcal{P}$ and $\text{sum}$ are more powerful than $\mu$, $\phi$ and $g$.

**Proposition 3.** For all operators $\omega \in \{\text{lim, max, } \mathcal{P}, \text{sum}\}$ and $k \geq 1$, we have $\text{FLev}_\omega(k) = \text{FS}_{\Sigma_{k+1}}$. 

153
For the limit operator this can be shown by induction, using standard techniques of computability. More precisely, in generalization of Shoenfield’s Limit Lemma it holds

$$F_{\Sigma_{k+1}} = \lim_{k \text{ times}} \left( \lim(F\text{Prim}) \ldots \right).$$

Having shown the assertion for the limit operator, one can prove that $\max, \sharp$ and $\text{sum}$ have of the same power in the following sense:

If $\omega_1, \omega_2 \in \{ \max, \sharp, \text{sum}, \text{lim} \}$ and $FC \in \{ \text{FPaPrim} \} \cup \{ F_{\Sigma_k} : k \geq 1 \}$, then

$$\omega_1(FC) \subseteq \text{Clos}_{\{ \circ \}} (FC \cup \omega_2(FC)).$$

It should be noticed that we here use an essentially stronger supposition and other proof techniques than for Proposition 1.

In [7] it is also shown that the operator $\min$ is closely related to $\mu, \phi$ and $\varrho$:

**Proposition 4.** For all $k \geq 1$, $F\text{Lev}_\min(k) = F\text{Lev}(k + 1)$.

From the viewpoint of computability theory one might ask why we did not start the nesting levels of operators $\omega$ with $F\text{Lev}_\omega(1) = F\text{Com}$ and proceed then, as we did, with $F\text{Lev}_\omega(k + 1) = \text{Clos}_{\{ \circ \}} (F\text{Lev}_\omega(k + 1) \cup \omega(F\text{Lev}_\omega(k + 1)))$. This would not change the levels $F\text{Lev}(k)$ of $\mu, \phi$ and $\varrho$, for $k \geq 1$. The levels $F\text{Lev}_\min(k)$ would become equal to $F\text{Lev}(k)$, and for $\omega \in \{ \text{lim, max, } \sharp, \text{sum} \}$ we would have $F\text{Lev}_\omega(k) = F_{\Sigma_k}$, however.

The main reason to start the levels with $\text{FPaPrim}$ instead of $F\text{Com}$ is that we are highly interested in studying the analogous classification with respect to polynomial-time computability. In our opinion, however, there are more similarities between the classes $\text{FP}$ and $\text{FPaPrim}$ than between $\text{FP}$ and $F\text{Com}$. So we want to know what happens if we start the stratification of levels at $\text{FPaPrim}$.

## 3 Operators on polytime functions

The theory of computational complexity, in particular that of polynomial-time computability, is mainly devoted to the machine-oriented point of view, where computing devices operate on words over finite alphabets. So the usual complexity classes consist of languages, and complexity classes of functions usually consist of word functions. In the present paper, we continue to deal preferably with sets of numbers and number-theoretic functions, respectively. This corresponds to the point of view taken in bounded arithmetic, cf. [8]. Of course, the following results and techniques can directly be transferred to word functions and languages if, between words, one uses the order by length and lexicographic comparing based on an ordering of the underlying alphabet.

All complexity-theoretic notions applied to sets of numbers or to number-theoretic functions are meant with respect to the (modified) binary expansion of numbers, where a binary word of length $l = |w|, w = b_l b_{l-1} \ldots b_1 \in \{ 1, 2 \}^*$, represents the number $\gamma(b_l b_{l-1} \ldots b_1) = \sum_{i=1}^l b_i \cdot 2^{i-1}$. The empty word $\Lambda$ represents the number 0, $\gamma(\Lambda) = 0$. The mapping $\gamma : \{ 1, 2 \}^* \rightarrow \mathbb{N}$ is a bijection.
between the words from \( \{1, 2\}^* \) and the natural numbers with respect to which the ordering of words by length and lexicographic comparing corresponds to the natural ordering of numbers. For a number \( x \in \mathbb{N} \), \( \|x\| \) denotes the length of its binary expansion, \( \|x\| = |\gamma^{-1}(x)| \). For an \( m \)-tuple \( \mathbf{x} = (x_1, \ldots, x_m) \), let \( \|\mathbf{x}\| = \|\langle x_1, \ldots, x_m\rangle\| \), this is the length of (the binary expansion of) the Cantor number \( \langle x_1, \ldots, x_m\rangle \).

A function \( f : \mathbb{N}^m \rightarrow \mathbb{N} \) is called polynomial-time computable (briefly: polytime) iff the related word function \( f_{\mathbf{w}} : (\{1, 2\}^*)^m \rightarrow \{1, 2\}^* \), which is defined by \( f_{\mathbf{w}}(w_1, \ldots, w_m) \simeq \gamma^{-1} \circ f(\gamma(w_1), \ldots, \gamma(w_m)) \), is computable in polynomial time by a (deterministic) Turing machine (TM) in the usual sense. Let FP denote the class of all polytime functions. FP is closed under composition, and all polytime functions are polynomially length-bounded, i.e., for any \( f \in \text{FP} \) there is a polynomial \( p \) such that \( f(\mathbf{x}) = y \) implies \( \|y\| \leq p(\|\mathbf{x}\|) \).

In the sequel, we shall use the same denotations for some classes of (number-theoretic) functions as they occur in literature for related classes of word functions. Also for classes of sets of number tuples, we shall use the same denotations as they are usually applied to the related complexity classes of languages. The denotations of the latter ones will here get the index "w" indicating that they concern (sets of) words. So let the class \( \text{P} \) consist of all \( m \)-sets of number tuples, \( A \subseteq \mathbb{N}^m \) that are decidable in polynomial time with respect to the binary expansions of the Cantor numbers:

\[
\text{P} = \{ A \subseteq \mathbb{N}^m : m \geq 1 \text{ and } \gamma^{-1}(x_1, \ldots, x_m) : (x_1, \ldots, x_m) \in A \} \in \text{P}_\mathbf{w} ,
\]

where \( \text{P}_\mathbf{w} \) denotes the usual class of languages decidable by polynomially time-bounded (deterministic) TMs. The classes \( \text{NP} \) and \( \text{NP}_\mathbf{w} \) are analogously related to each other. Of course, \( \text{P} = \text{NP} \) iff \( \text{P}_\mathbf{w} = \text{NP}_\mathbf{w} \), and this is the classical \( \text{P} \) vs. \( \text{NP} \) problem.

Polytime functions are partial in general, but, analogously to partial primitive recursive functions, they can be regarded as restrictions of total polytime functions to polynomially decidable domains: \( f \in \text{FP} \) iff there are a total function \( \tilde{f} \in \text{FP} \) and a set \( A \in \text{P} \), \( A \subseteq \mathbb{N}^m \), such that \( f(\mathbf{x}) \simeq \begin{cases} \tilde{f}(\mathbf{x}) & \text{if } \mathbf{x} \in A, \\ \uparrow & \text{otherwise.} \end{cases} \)

Hence \( \phi(g) \in \text{FCom} \) if \( g \in \text{FP} \). On the other hand, the unrestricted application of the first value operator to polytime functions leads already to all computable functions, as one easily shows:

**Lemma 1.** To any \( f \in \text{FCom} \) there is a \( g \in \text{FP} \) such that \( f = \phi(g) \). Thus, \( \phi(\text{FP}) = \text{FCom and } \text{Clos}_{\circ,\phi} (\text{FP}) = \text{FAH} \).

In the context of polynomial-time computability, we want to exclude such trivial constructions leading to functions \( \omega(g) \notin \text{FP} \) for certain \( g \in \text{FP} \). This is done by restricting the search domain for \( y \) in a polynomially length-bounded way depending on the remaining argument \( \mathbf{x} \), in building \( \omega(g)(\mathbf{x}) \) from the set of all values \( g(y, \mathbf{x}) \). A simple but sufficiently general way consists in restricting the operator \( \omega \) to functions which are undefined for arguments \((y, \mathbf{x})\) if \( y \) lies
outside a polynomial length-bound depending on \( x \). So, an \((m+1)\)-ary function \( g \) is said to be 1-polynomial iff there is a polynomial \( p \) such that

\[
g(y, x) \uparrow \text{ for all } y \in \mathbb{N} \text{ and } x \in \mathbb{N}^m \text{ satisfying } \|y\| > p(\|x\|).
\]

Now, for all numbers \( y \) with \( g(y, x) \downarrow \) we have \( 0 \leq y < 2^p(\|x\|) - 1 \).

Let \( \omega' \) denote the restriction of an operator \( \omega \in \{\mu, \overline{\mu}, \phi, \max, \sum, \min\} \) to 1-polynomial functions of arities \( \geq 2 \). Since the analogous restriction \( \lim' \) yields only the nowhere defined function, it is not of interest.

For an adequate restriction of the operator \( \varrho \), we employ a notion well known in complexity theory. A function \( g : \mathbb{N} \rightarrow \mathbb{N} \) is said to be honest iff there is a polynomial \( p \) such that

\[
\varrho(g)(y) = x \text{ implies } \|x\| \leq p(\|y\|).
\]

Let \( \varrho' \) denote the restriction of the operator \( \varrho \) to honest (unary) functions.

It is easily seen that all functions from \( \text{Clos} \{\varrho, \omega'\} (\text{FP}) \) are polynomially length-bounded, for all \( \omega' \in \{\mu', \overline{\mu'}, \phi', \varrho', \max', \sum', \min'\} \).

The polytime \( \omega' \)-levels \( \text{FLev}'_{\omega'}(k) \) are defined similarly to the \( \omega \)-levels. In contrast to them, however, we now start with the basic class FP: Let

\[
\text{FLev}'_{\omega'}(0) = \text{FP},
\]

\[
\text{FLev}'_{\omega'}(k+1) = \text{Clos} \{\varrho\} (\text{FLev}'_{\omega'}(k) \cup \omega'(\text{FLev}'_{\omega'}(k))) \text{ for } k \in \mathbb{N}.
\]

It turns out that the operators \( \overline{\mu}', \phi', \varrho', \min' \) and \( \max' \) yield the same polytime levels:

**Proposition 5.** For all \( \omega'_1, \omega'_2 \in \{\overline{\mu}', \phi', \varrho', \min', \max'\} \) and all \( k \in \mathbb{N} \), we have \( \text{FLev}'_{\omega'_1}(k) = \text{FLev}'_{\omega'_2}(k) \).

The proof is quite analogous to that of Proposition 1, namely based on the mutual expressibilities of the operators under consideration. More precisely, for any \( \omega'_1, \omega'_2 \in \{\overline{\mu}', \phi', \varrho' \min', \max'\} \) and every function class FC which is closed under composition and includes FP, we have \( \omega'_1(\text{FC}) \subseteq \text{Clos} \{\varrho\} (\text{FP} \cup \omega'_2(\text{FC})) \).

In the sequel, we shall write \( \text{FLev}'(k) \) instead of \( \text{FLev}'_{\omega'}(k) \) with an arbitrary \( \omega' \in \{\overline{\mu}', \phi', \varrho', \min', \max'\} \).

## 4 Relationships to the polynomial hierarchy

Analogously to the arithmetical hierarchy, the (number-theoretic version of the) **polynomial hierarchy** is considered both as the hierarchy of classes \( \Sigma^p_k, \Pi^p_k \) and \( \Delta^p_k \), for \( k \in \mathbb{N} \), but also as the union over them: \( \text{PH} = \bigcup_{k=0}^{\infty} \Sigma^p_k \). To characterize the classes \( \Sigma^p_k \), one can use bounded quantifications, where

\[
\forall^m z \ldots \text{ stands for } \forall z (\|z\| \leq m \Rightarrow \ldots) \text{ and } \\
\exists^m z \ldots \text{ stands for } \exists z (\|z\| \leq m \land \ldots).
\]
\( \Sigma^p_k \) consists of all relations \( A \subseteq \mathbb{N}^m, m \geq 1 \), which are representable in the form

\[
A = \{ x \in \mathbb{N}^m : \exists p_1(\|x\|) y_1 \forall p_2(\|x\|) y_2 \ldots Q^{p_k(\|x\|)} y_k (y_1, \ldots, y_k, x) \in B \}
\]

with \( B \in \mathbb{P} \), polynomials \( p_1, \ldots, p_k \), and \( Q \in \{\exists, \forall\} \) such that the prefix of quantifiers becomes alternating.

An equivalent characterization can be given by means of deterministic and nondeterministic polynomial-time oracle Turing machines (POTM and NPOTM, respectively). For a class \( C \) of sets, let \( P^C \) and \( NP^C \) denote the class of all sets \( A \subseteq \mathbb{N}^m, m \geq 1 \), which can be accepted by POTMs and NPOTMs, respectively, using oracle sets belonging to \( C \). Then

\[
\Sigma^p_0 = \Pi^p_0 = \Delta^p_0 = \mathbb{P} ; \quad \Sigma^p_{k+1} = \text{NP}^{\Sigma^p_k} , \quad \text{and} \quad \Pi^p_{k} = \text{co} \Sigma^p_{k} \quad \text{for all} \quad k \in \mathbb{N}.
\]

The classes \( \Delta^p_{k+1} \), however, are usually defined in the following special way:

\[
\Delta^p_{k+1} = \text{P}^{\Sigma^p_k}.
\]

Let the function classes \( \text{FP}^C \) and \( \text{FNP}^C \) consist of all functions computable by POTMs and NPOTMs, respectively, which use oracle sets from \( C \). The meaning should be clear for the deterministic case. Computation of a function by a nondeterministic machine means that, for any input, all terminating computations yield the same result, namely the function value at this input, and there is at least one such terminating computation iff the function is defined at this input. Hence \( f \in \text{FNP}^C \) iff there is a 1-polynomial function \( g \in \text{FP}^C \) such that

i) \( g(y_1, x) = g(y_2, x) \) whenever \( g(y_1, x) \downarrow \) and \( g(y_2, x) \downarrow \), and

ii) \( f = \phi'(g) \).

So we have \( \text{FP}^C \subseteq \text{FNP}^C \), \( A \in \text{P}^C \) iff \( \chi_A \in \text{FP}^C \), and \( A \in \text{NP}^C \) iff \( \chi_A^0 \in \text{FNP}^C \), where \( \chi_A \) denotes the (total) characteristic function of a set \( A \), in contrast to its (partial) semicharacteristic function \( \chi_A^0 \).

Analogously to the definition of the classes \( \text{FSigma}_k \) in the arithmetical hierarchy, any class \( C \) consisting of sets determines a class of (polynomially length-bounded) functions by

\[
\text{F}_p[C] = \{ f \in \text{FAll} : \text{graph}(f) \in C \text{ and } f \text{ is polynomially length-bounded} \}.
\]

One easily shows that \( \text{F}_p[\text{NP}^C] = \text{FNP}^C \), whereas the analogous graph theorem for the deterministic case does not hold, probably. For example, we have \( \text{F}_p[\text{P}] = \text{FP} \text{ iff } \text{P} = \text{UP} \). This is a result by J. Grollmann and A. Selman, cf. Satz 9.7 in [18]. Of course, \( \text{FP}^\text{P} = \text{FP} \). The class \( \text{FNP}^\text{P} \) contains exactly those functions which are computable by NPOTMs with the empty oracle set or, equivalently, without using the oracle. This class will also be denoted by \( \text{FNP} \), and its elements are called nondeterministically computable in polynomial time. \( \text{FNP} \) is just the number-theoretic analogue if Selman’s class \( \text{NPSV} \), see [14, 15].

It is natural to define the following function classes of the polynomial hierarchy:

\[
\text{F}_{\Sigma^p_k} = \text{F}_p[\Sigma^p_k] \quad \text{for } k \in \mathbb{N}.
\]
Thus, $\text{FP} \subseteq \text{F}_p[P] = \text{F}_{\Sigma_0^p}$, $\text{F}_{\Sigma_1^p} = \text{F}_p[\text{NP}] = \text{FNP}$, and

$$\text{F}_{\Sigma_k^{p}} = \text{F}_p[\Sigma_{k+1}^{p}] = \text{F}_p[\text{NP}^{\Sigma_k}] = \text{FNP}^{\Sigma_k}$$

for all $k \in \mathbb{N}$.

Let $\text{FPH}$ denote the class of all polynomially length-bounded functions with graphs from $\text{PH}$:

$$\text{FPH} = \text{F}_{p}[\text{PH}] = \bigcup_{k \in \mathbb{N}} \text{F}_{\Sigma_k^p}.$$

Obviously, $\bigcup_{k \in \mathbb{N}} \text{FLev'}(k) = \text{Clos}_{\{\omega', \phi', \mu', \sum', \min', \max'\}}(\text{FP})$. It is not hard to show that this class coincides with the polynomial hierarchy. The analogue even holds for the operator $\mu'$:

**Proposition 6.** For every operator $\omega' \in \{\mu', \mu', \phi', \phi', \min', \max'\}$, we have $\text{Clos}_{\{\omega', \phi', \mu', \sum', \min', \max'\}}(\text{FP}) = \text{FPH}$.

This yields the following characterization of the P vs. NP problem in terms of a closure property of FP with respect to our function operators.

**Theorem 1.** For every $\omega' \in \{\mu', \mu', \phi', \phi', \min', \max'\}$, as well as for any other operator $\omega'$ which satisfies $\text{FNP} \subseteq \text{Clos}_{\{\omega', \phi', \mu', \sum', \min', \max'\}}(\text{FP}) \subseteq \text{FPH}$, we have

$$\text{P} = \text{NP} \iff \text{FP} \text{ is closed under } \omega'.$$

In order to localize the polytime levels within the polynomial hierarchy, we have to consider the function classes

$$\text{F}_{\Delta_k^{p}} = \text{F}_{p}^{\Sigma_k}$$

for all $k \in \mathbb{N}$.

In particular, $\text{F}_{\Delta_1^{p}} = \text{F}_{p}^{\Sigma_0^p} = \text{FP}$, and $\text{F}_{\Delta_2^{p}} = \text{F}_{p}^{\Sigma_1^p} = \text{F}_{p}^{\text{NP}}$. From $\text{F}_{\Sigma_1^p} = \text{FNP}$ upwards, we have the usual inclusions:

**Lemma 2.** For all $k \geq 1$, $\Sigma_k^p \subseteq \Delta_{k+1}^{p} \subseteq \Sigma_{k+1}^p$.

The polytime levels, even for the operator $\mu'$, coincide with our deterministic function classes:

**Theorem 2.** For all $k \in \mathbb{N}$, we have $\text{FLev'}(k) = \text{F}_{\Delta_{k+1}^{p}}$. Moreover, it holds $\text{FLev'}(k) = \text{FLev'}(k)$.

The proof (of the first part) is by induction on $k$. It is mainly based on the following result which can be shown by adapting a technique established by Krentel [9] in order to prove that $\text{MAXP} \subseteq \Delta_2^p$, see also [18].

**Lemma 3.** Every total function $f \in \Delta_{k+2}^p$ belongs to $\phi'(\Delta_{k+1}^p)$.

It can be shown that the operators $\phi'$ and $\sum'$ define the same polytime levels.

**Proposition 7.** $\text{FLev'}_{\phi'}(k) = \text{FLev'}_{\sum'}(k)$ for all $k \in \mathbb{N}$.
Unfortunately, we did not succeed in localizing the levels of \#' and sum' within or compared to the polynomial hierarchy. It is even open whether they are contained in FPH. This corresponds to the open questions concerning Valiant’s counting class \#P well-known from complexity theory, cf. Chapter 9 in [2]. One easily sees that \#P = \#(FP) ⊆ FLev\#(1). Thus, our unsolved problems concern already the first nonzero level of \# or sum', and it could even be that FLev\#(1) is not contained in FPH.

5 Equivalences to a collapse of the polynomial hierarchy

By Theorem 2 and Lemma 2, the polytime levels FLev'(k), k ∈ N, span the polynomial hierarchy of functions, which also characterizes the usual polynomial hierarchy of (classes of) sets. Thus, the polynomial hierarchy collapses iff the sequence of operator levels collapses to some level FLev'(k). The latter holds iff FLev'(k) = FLev'(k + 1), and this is the case iff FLev'(k) is closed under some (equivalently: under every) operator \( \omega \in \{ \mu, \mu', \phi', \phi, \min, \max \} \). In this section, we establish several operator-related equivalent formulations of the question whether \( \Sigma^p_{k+1} \) is closed under complement. This problem is equivalent to \( \Sigma^p_{k+1} = \Pi^p_{k+1} \) and to \( \Sigma^p_{k+1} = \text{FPH} \). For \( k = 0 \), it is just the NP vs. coNP problem.

We start with a basic chain of inclusions:

**Proposition 8.** For all \( k \in \mathbb{N} \),

\[
\begin{align*}
\text{FP}^{\Sigma^p_k} & \subseteq \text{FP}^{\Sigma^p_{k+1} \cap \Pi^p_{k+1}} \subseteq \text{FNP}^{\Sigma^p_k} \subseteq \phi'(\text{FP}^{\Sigma^p_k}) \subseteq \text{FP}^{\Sigma^p_{k+1}} \subseteq \text{FPH}.
\end{align*}
\]

The second inclusion follows by means of the proof technique of Lemma 3. The other ones or the equalities are easily shown or hold by definition.

Let the operator \( \tau \) of totalization assign to a function \( f \) the function \( \tau(f) = \bar{f} \) of the same arity, which is defined by \( \bar{f}(x) = \begin{cases} f(x) + 1 & \text{if } f(x) \downarrow, \\ 0 & \text{if } f(x) \uparrow. \end{cases} \)

As a counterpart of totalization, the operator \( \pi \) of partialization assigns to a function \( f \) the function \( \pi(f) = \bar{f} \) defined by \( \bar{f}(x) = \begin{cases} f(x) - 1 & \text{if } f(x) \downarrow > 0, \\ \uparrow & \text{otherwise}. \end{cases} \)

Thus, \( \pi(f) = h_0 \circ f \), where \( h_0(x) \simeq \begin{cases} x - 1 & \text{if } x > 0, \\ \uparrow & \text{if } x = 0. \end{cases} \) For example, all function classes, which contain FP and are closed under composition, are closed under partialization, too. This holds for all classes occurring in Proposition 8, possibly except \( \phi'(\text{FP}^{\Sigma^p_k}) \). On the other hand, all the classes occurring in Proposition 8, possibly except FNP\#, are closed under totalization.

The variety of statements known as equivalent to \( \Sigma^p_{k+1} = \Pi^p_{k+1} \) can now be enriched by some ones concerning equations between and closure properties of the above function classes.
Theorem 3. The following statements are equivalent:

(a) $\Sigma_{k+1}^p = \Pi_{k+1}^p$;
(b) $\text{FP} \Sigma_{k+1}^p \cap \Pi_{k+1}^p = FNP \Sigma_k^p$;
(c) $\text{FNP} \Sigma_k^p = \phi'(\text{FP} \Sigma_k^p)$;
(d) $\phi'(\text{FP} \Sigma_k^p) = \text{FP} \Sigma_{k+1}^p$;
(e) $\phi'(\text{FP} \Sigma_k^p)$ is closed under composition;
(f) $\phi'(\text{FP} \Sigma_k^p)$ is closed under partialization;
(g) $\text{FNP} \Sigma_k^p$ is closed under totalization.

Also, $\text{FP} \Sigma_k^p = \text{FP} \Sigma_{k+1}^p \cap \Pi_{k+1}^p$ iff $\Delta_{k+1}^p = \Sigma_{k+1}^p \cap \Pi_{k+1}^p$, and from $\Sigma_{k+1}^p = \Pi_{k+1}^p$ it follows that $\text{FP} \Sigma_{k+1}^p \cap \Pi_{k+1}^p = \text{FPH}$ and conversely.

The structure of our proof of the equivalence of the statements (a)–(g) can be sketched by the following scheme:

(b) $\longleftrightarrow$ (a) $\longleftrightarrow$ (d)

(g) $\longleftrightarrow$ (c) $\longleftrightarrow$ (e) $\longleftrightarrow$ (f)

The remaining assertions follow by standard arguments. For details, the reader is again referred to [7].

The most interesting special case of Proposition 8 and Theorem 3 is obtained for $k = 0$ and, accordingly, $\Sigma_0^p = \text{NP}$, $\Pi_0^p = \text{coNP}$, $\text{FNP} \Delta_1^p = \text{FP} \Delta_1^p = \text{FP}$, $\text{FNP} \Sigma_0^p = \text{FNP}$, and $\Delta_2^p = \text{P}$. We re-formulate these results as a corollary. It should be mentioned that the implication $\text{NP} = \text{coNP} \Rightarrow \text{FNP} = \text{F} \Delta_2^p$ was essentially known from [15], Theorem 4.

Corollary 1. We have $\text{FP} \subseteq \text{FP} \Sigma^p \cap \text{coNP} \subseteq \text{FNP} \subseteq \phi'(\text{FP}) \subseteq \text{F} \Delta_2^p \subseteq \text{FPH}$, and the following statements are equivalent:

(a) $\text{NP} = \text{coNP}$;
(b) $\text{FP} \Sigma^p \cap \text{coNP} = \text{FNP}$;
(c) $\text{FNP} = \phi'(\text{FP})$;
(d) $\phi'(\text{FP}) = \text{F} \Delta_2^p$;
(e) $\phi'(\text{FP})$ is closed under composition;
(f) $\phi'(\text{FP})$ is closed under partialization;
(g) $\text{FNP}$ is closed under totalization.

Moreover, $\text{FP} = \text{FP} \Sigma^p \cap \text{coNP}$ iff $\text{P} = \text{NP} \cap \text{coNP}$, and from $\text{NP} = \text{coNP}$ it follows that $\text{FP} \Sigma^p \cap \text{coNP} = \text{FPH}$ and conversely.
Besides the results presented so far, hierarchies of function classes between \( \text{FNP}^{\Sigma_k^p} \) and \( \phi'(\text{FP}^{\Sigma_k^p}) \), for \( k \in \mathbb{N} \), are established in [7]. They correspond to the generalized Boolean hierarchies over \( \Sigma_{k+1}^p = \text{NP}^{\Sigma_k^p} \) which lead from \( \Sigma_{k+1}^p \) to \( \Delta_{k+2}^p = \text{P}^{\Sigma_{k+1}^p} \) and were explored in detail for \( k = 0 \).

References

6. Hemmerling, A.: Hierarchies of function classes defined by the first-value operator. RAIRO - Theoretical Informatics and Applications 42 (2008), 253-270
17. Soare, R.I.: Computability theory and applications. Springer-Verlag, Berlin et al., forthcoming
Abstract. We investigate different variants of unambiguity in the context of computing multi-valued functions. We propose a modification to the standard computation models of Turing Machines and configuration graphs, which allows for unambiguity-preserving composition. We define a notion of reductions (based on function composition), which allows nondeterminism but controls its level of ambiguity. In light of this framework we establish reductions between different variants of path counting problems. We obtain improvements of results related to inductive counting.

Keywords: logarithmic space, reduction, unambiguity

1 Introduction

The notion of nondeterminism is a fundamental notion in theoretical computer science. Numerous restrictions of nondeterminism have been introduced and studied. A well known restriction is that of unambiguity, in which the machine is not “aware” of the existence of an accepting path (and makes nondeterministic choices), but the path itself is required to be unique.

The unambiguous version of logarithmic space, called UL, has been explicitly considered for the first time in [1] and [2]. In the latter paper, UL variants allowing polynomially many accepting computation paths, as well as variants that consider not only accepting, but all reachable or all paths, have been proposed. Some inclusions between these classes were presented, and the classes ReachUL and StrongUL have been shown to be closed under complementation.

The Immerman-Szelepcsényi technique of inductive counting has been extended in [3], allowing the removal of ambiguity at the cost of a relatively small increase in required computation space. StrongUL has been shown by Allender and Lange to be contained in deterministic space \(O\left(\frac{\log(n)^2}{\log\log(n)}\right)\) (see [4]). In [5], Lange has exposed a problem complete for ReachUL.

Finally, inductive counting has been used again by Reinhardt and Allender in [6] to show that UL and NL coincide in the nonuniform setting (i.e., NL/poly=UL/poly), and thus also in the uniform setting under some hardness assumptions (see [7] for details).

The contributions of the present work are threefold: We provide a modification of the standard model of oracle Turing Machines, which allows for nondeterministic computation of deterministically valued (i.e., well defined) functions, as
opposed to functions based on properties of computation trees of nondetermi-
nistic machines (as, for example, the function classes $\#L$, GapL, etc.; see [1]).
This type of computation has been previously defined ([8]), but it was based on
deterministic machines having access to an oracle for a language from a possibly
nondeterministic class. Our model agrees with this approach when full power
of nondeterminism is allowed, but has the advantage of being easily adaptable to
classes of limited ambiguity.

We introduce a notion of nondeterministic, unambiguous reductions. In our
model, oracles are used only as a tool of function composition, and not as sources
of additional computational power. Therefore our reductions work in a similar
way to many-one reductions, and are well fit for the purpose of comparing the
ambiguity-complexity of functions. Within this framework, we analyze variants
of the path-counting problem—in particular, by Propositions 5.1 and 5.2 we
obtain the equivalence of counting up to any constant number of (arbitrary or
simple) paths.

Finally, we take a closer look on the inductive counting technique of [9, 10],
which allows us to combine the results of [3] and [6] into Algorithm 6.7: an
unambiguous algorithm for reachability on graphs with restricted ambiguity of
shortest paths.

The full version of this work, containing more formal details and proofs of
propositions, has been submitted by the first author as a Ph.D. thesis to the
School of Graduate Studies of McMaster University.

2 Quering Turing Machines

The usual model of Turing Machines poses a technical problem when composing
computations with sub-linear space bounds: the input and output tapes, not
subject to the space bounds, become an internal tape of the composed machine,
which should obey the space restrictions. To deal with that issue we employ
its well-known modification: instead of producing a (possibly long) output in
its entirety, the machine computes just a single character at a requested index
(if the index happens to point beyond the output, the answer will be a blank
symbol). Moreover, we use the same approach to access the machine’s input—it
writes the index of the input character it is interested in on one of its tapes, and
queries an oracle (by entering a special state). Therefore such a machine can be
seen as one rewriting requests about its output to (sequences of) queries
about its input, and generating an answer (by entering a special answer state)
based on the results of these queries. We call such a (nondeterministic) machine
a Quering Turing Machine (QTM).

We will use the usual notion of configuration for Turing Machines: a con-
figuration is a tuple consisting of the current state, together with the contents
of all tapes (note that this does include the request tape, but not the input, as
the latter is only available via an oracle) and the positions of the heads. Any

---

1 We will use the words: input, output, request, query, and answer, to mean precisely
the roles described here.
configuration with an answer state is called an answer configuration. As with every oracle machine, there are two kinds of state changes (two "yield" relations): those intrinsic to the machine itself, and those “performed” by the oracle (called extrinsic), which is not part of the machine.

Using the power of nondeterministic guesses (and, later on, talking about unambiguity properties) requires the ability to terminate branches on which the computation “went wrong” (i.e., invalid guesses have been made). The usual model incorporates such situations into the “reject” answer from the machine, we make such failures explicitly distinct from any possible answer the machine might give. We will represent the failure by any configuration that does not yield a new one. Furthermore, we do not require the answer states to be final—terminating or continuing the computation is seen as yet another nondeterministic choice (this will significantly simplify the formal definition of composition for computation graphs). Moreover, as the input oracle will often be substituted with another QTM, we allow oracles in principle to give inconsistent (“nondeterministic”) answers as well as to fail, and make all failures unrecoverable, i.e., propagating from any component to the whole computation.

With the above in place, the only externally visible difference between a Quering Turing Machine and an oracle is that the former needs to be provided with an input (oracle) before one can talk about its answers. Therefore we will freely use the term “oracle” to refer also to closed Quering Turing Machines—those with a specific input “plugged in.”

Given these, we can formally define the outcome of a computation as follows:

**Definition 2.1.** *A QTM can answer* \( a \) *on request* \( i \) *given input oracle* \( O \) *iff any configuration with “*\( a \)" answer state is reachable from the initial configuration (on* \( i \) *)via the reflexive transitive closure of the union of intrinsic and extrinsic yield relations.***

**Definition 2.2.** *An oracle consistently computes (returns) a string* \( X \in \Sigma^* \) *iff, for any request* \( i \), *it can only answer* \( X[i] \) *(or fail, which is always allowed).***

**Definition 2.3.** *A QTM* \( M \) *is sound for the function* \( \phi : \alpha \rightarrow \beta \) *iff, when supplied with an input oracle consistently returning* \( X \in \alpha \), *it consistently computes* \( \phi(X) \).

Note that according to the above definition, a single machine might be sound for many functions—in particular, a machine that always fails is sound for every possible function on \( \Sigma^* \)! Therefore we define:

**Definition 2.4.** *For any* \( a \in \Sigma \), *we say that a QTM* \( M \) *is* \( a \)-**total** *for a function* \( \phi : \alpha \rightarrow \beta \) *iff, whenever supplied an infallible input oracle consistently returning* \( X \in \alpha \) *and given the request* \( i \) *such that the* \( i \)-**th character of* \( \phi(X) \) *is* \( a \), *it can answer* \( a \). *M is** total *for* \( \phi \) *iff it is* \( a \)-**total** *for* \( \phi \) *for every* \( a \in \Sigma \).

By identifying failures with either “accept” or “reject” answers of traditional nondeterministic deciders, existential and universal acceptance can be easily seen to correspond exactly to sound QTM’s being 1-total and 0-total, respectively.
As both input and output are implicit in a QTM, we need special arrangements to give a meaningful definition of the space consumed by it. Remembering that whenever an oracle is queried about an index beyond its output (and only then), it answers with a blank symbol, we can define our space bounds:

**Definition 2.5.** The size (length) of an oracle is the smallest value of a query (i.e., the smallest character index) to which it might respond with a blank.

**Definition 2.6.** A QTM \( M \) operates in space \( f(n) \) iff, whenever supplied with an input oracle of size \( n \), it reads or writes no more than \( f(n) \) cells on all its tapes, including the oracle tape.

Building on the above, we can naturally define some complexity classes:

**Definition 2.7.** For a space bound \( f(n) \geq \log(n) \), the class \( \text{QFunc}(f(n)) \) consists of all functions that have sound and total QTMs operating in space \( O(f(n)) \).

**Definition 2.8.** For a space bound \( f(n) \geq \log(n) \), the class \( \text{QSpace}(f(n)) \) (and co-\( \text{QSpace}(f(n)) \)) consists of those languages, whose characteristic functions have sound, 1-total (respectively, 0-total) QTMs operating in space \( O(f(n)) \).

The classes defined this way correspond naturally to the classical ones: \( \text{QSpace}(f(n)) = \text{NSpace}(f(n)) \), \( \text{co-QSpace}(f(n)) = \text{co-NSpace}(f(n)) \) and \( \text{QFunc}(f(n)) = \text{FNSpace}(f(n)) \).

Composing QTM computations will be done in the most natural way: using separate tapes for the two machines, and invoking the program of the inner one whenever the outer one wants to make an input query. The following can be easily seen:

**Proposition 2.9.** If \( M \) and \( N \) are QTMs sound for \( \phi \) and \( \psi \), respectively, then their composition is sound for \( \phi \circ \psi \). If they are total, so is the composition. Finally, if they run in space \( f(n) \) and \( g(n) \), respectively, then their composition operates within space \( O(f(2^{O(g(n))})) \).

### 3 Quering Computation Graphs and Ambiguity

We extend the concept of configuration graphs to allow processing oracles queries.

**Definition 3.1.** A Quering Computation Graph (QCG) \( (V, E, S, c) \) is a directed graph with a distinguished subset \( S \subseteq V \), together with a coloring function \( c : V \cup E \to \Sigma \cup \{\bot\} \) (\( \bot \) denoting “no color”), such that there is at most one vertex of every color (i.e., \( c(u) = c(v) \neq \bot \Rightarrow u = v \)), but there may be many edges of a single color.

Intuitively, we take the usual configuration graph representation of nondeterministic computation (i.e., edges following the intrinsic yield relation) and add colored vertices and edges to represent answer configurations and transitions dependent on oracle queries (i.e., extrinsic yield), respectively. The requirement of
there being at most one answer configuration for any specific answer can be easily fulfilled by making the machine erase the contents of all tapes before entering an answer state.

Oracles (and, equivalently, closed QTM) do not issue any input queries. Thus their operation can be modeled with the following restriction of QCGs:

**Definition 3.2.** A Closed Computation Graph (CCG) is a Quering Computation Graph in which all edges are uncolored.

To compose computations (i.e., making a machine use the answers of another one as its input) we need a corresponding operation on QCGs. Let us define it as follows:

**Definition 3.3.** Given QCGs \(G = \langle V_G, E_G, S_G, c_G \rangle\) and \(H = \langle V_H, E_H, S_H, c_H \rangle\), and a function \(f : V_G \to S_H\), the \(f\)-composition of \(G\) and \(H\) (denoted \(G \circ_f H\)) is a QCG \(\langle V, E, S, c \rangle\) with:

\[
V = V_G \times V_H, \\
S = \{\langle u, f(u) \rangle : u \in S_G \}, \\
E = E_1 \cup E_2 \cup E_3, \text{ where} \\
E_1 = \{\langle \langle u, f(u) \rangle, \langle v, f(v) \rangle \rangle : c_G(\langle u, v \rangle) = \bot \}, \\
E_2 = \{\langle \langle u, x \rangle, \langle u, y \rangle \rangle : \langle x, y \rangle \in E_H \}, \\
E_3 = \{\langle \langle u, x \rangle, \langle v, f(v) \rangle \rangle : c_G(\langle u, v \rangle) = c_H(x) \neq \bot \}, \\
c(\langle u, x \rangle) = \begin{cases} 
c_G(u) & \text{if } x = f(u), \\
\bot & \text{otherwise}, 
\end{cases} \\
c(\langle \langle u, x \rangle, \langle v, y \rangle \rangle) = \begin{cases} 
c_H(\langle x, y \rangle) & \text{if } u = v, \\
\bot & \text{otherwise}. 
\end{cases}
\]

We say that an edge in the \(f\)-composition is of type 1, 2 or 3, depending on which of the sets \(E_1\), \(E_2\) and \(E_3\) it belongs to.

The correspondence between \(f\)-composition and “plugging in” one QTM as an input oracle of another is as follows. The function \(f\) represents the way of extracting the oracle query (and so the initial configuration of the inner machine) from the configuration of the outer machine (usually \(f\) is as simple as taking a segment of the configuration corresponding to the contents of the oracle tape, in which case we will omit it entirely and simply write \(G \circ H\)). The transitions in the composed computation can be divided into three groups (edges of type 1, 2, and 3, respectively): the uncolored (i.e., not depending on the oracle answers) transitions of the outer machine, the inner computation, and transferring the answer of the inner to the outer machine (in which case the color of the answer has to match that of the “conditional” edge).

The following are expected consequences of our definitions:

**Observation 3.4.** The composition of two QCGs is a QCG. The composition of a QCG and a CCG is a CCG. Furthermore, this composition is associative.
To capture the degree of ambiguity of a computation, we look at the shape of its CCGs:

**Definition 3.5.** For a family $\mathcal{C}$ of CCGs, we say that a QTM $M$ is a $\mathcal{C}$-machine iff, when supplied with any consistent input, its CCG belongs to $\mathcal{C}$.

**Definition 3.6.** The class $\mathcal{C}$-QFunc$(f(n))$ consists of all functions that have sound, total $\mathcal{C}$-machines operating in space $O(f(n))$. The classes $\mathcal{C}$-QSpace$(f(n))$ and $\text{co-}\mathcal{C}$-QSpace$(f(n))$ can be defined analogously.

To be able to talk about classical deterministic and non-deterministic algorithms, we introduce two classes of CCGs: $D$—those of out-degree 1, and $N$—the class of all CCGs. Computational (un)ambiguity is enforced by limiting the number of distinct ways to reach (from a source vertex) a node in the CCG. The variant of this restriction will be denoted by specifying the following (orthogonal) aspects:

1. The number of paths allowed (as a function of the size of the graph, with $k$ and $p$ standing for arbitrary constants and polynomials, respectively),
2. The types of paths that are counted:
   - $A =$ all paths,
   - $S =$ simple paths (i.e., without loops),
   - $M =$ minimal-length paths.
3. The types of target nodes of the paths of interest:
   - $A =$ all nodes,
   - $F =$ colored (“final”) nodes only.

For example, $pAF$-graphs are those Closed Computation Graphs with at most $p(n)$ paths between a source and any final vertex, and $1MA$-graphs—those with a unique minimal-length path to any (reachable) vertex.

In the above notation, a number of classical complexity classes can be captured in a unified manner. In particular

$$L = D\text{-QSpace}(\log(n)), FL = D\text{-QFunc}(\log(n)),$$

$$UL = 1AF\text{-QSpace}(\log(n)), RUL = 1AA\text{-QSpace}(\log(n)),$$ and

$$\text{FewL} = \bigcup_{p(n)\in n^{O(1)}} pAF\text{-QSpace}(\log(n)).$$

We denote ($\text{REM}$ contains graph classes closed under edge removal):

$$\text{ALL} := \{D, N\} \cup \{pAF, pAA, pSF, pSA, pMF, pMA|p \in n^{O(1)}\},$$

$$\text{UNI} := \{D, 1AF, 1AA, 1SF, 1SA, 1MF, 1MA\},$$

$$\text{REM} := \{D, N\} \cup \{pAF, pAA, pSF, pSA|p \in n^{O(1)}\}.$$
4 Reductions

To talk about the relative complexity of different problems (functions), we would like to introduce a notion analogous to many-one log-space reductions, a notion that would allow nondeterminism but at the same time limit its level of ambiguity. The very nature of Quering Turing Machines (being queried multiple times about different characters of their output) suggests employing some variant of Turing reductions, but these have been shown (see [11–14]) to be very sensitive to the exact definition.

We have decided to take a path similar to that of [14]: we allow the original input to be transformed in a parametrized way, requiring that the transformation can be performed in one of the classes \(C\text{-QFunc}(\log(n))\), and the “parameters” fit within the desired space bound. The resulting model ends up being close to many-one reducibility (as it is based on function composition), but with each reduction consisting of two parts—the family of input transformations, and the actual algorithm, allowed to query the oracle on any member of this family:

**Definition 4.1.** A function \(\phi : \alpha \to \beta\) is \(C/D\)-reducible to a function \(\psi : \gamma \to \delta\) (written \(\phi \preceq_{C/D} \psi\)) iff there exist a family of functions \(\theta_i : \alpha \to \gamma\), and a function \(\xi : \delta^* \to \beta\) such that:

- taking \(\theta(X) := (\theta_i(X))_i\) we have \(\theta \in C\text{-QFunc}(\log(n))\) (i.e., the functions \(\theta_i\) can be “uniformly” computed in \(C\text{-QFunc}(\log(n))\)),
- \(\xi \in D\text{-QFunc}(\log(n))\),
- for every \(X \in \alpha\), \(\xi((\psi \circ \theta_i(X))_i) = \phi(X)\).

If the complexity class of \(\theta_i\) or \(\xi\) is not known, we will use the function itself as the subscript/superscript of \(\preceq\). Moreover, we will omit the subscript/superscript entirely if the corresponding function is the identity.

The following technical result, extending Proposition 2.9, is the key to making use of unambiguous, nondeterministic reductions:

**Lemma 4.2.** For any transformations \(\psi : \alpha \to \beta\), \(\phi : \beta \to \delta\), and Closed Computation Graph classes \(C \in \text{UNI}\) and \(D \in \text{ALL}\), \(C \subseteq D\), if there exist:

- a \(D\)-machine \(M\) sound (and total) for \(\phi\), working in space \(f(n)\), and
- a \(C\)-machine \(N\) sound (and total) for \(\psi\), working in space \(g(n)\),

then we can build a \(D\)-machine sound (and total) for \(\phi \circ \psi\), requiring space \(O(f(2^{O(g(n))}))\).

**Proof.** Using the natural composition of \(M\) and \(N\) meets the soundness, totality, and space requirements according to Proposition 2.9. It remains to show how to obtain the desired (un)ambiguity properties. First, let us make the following simple observation about the composition:

**Observation 4.3.** Every path in the \(f\)-composition of Quering Computation Graphs \(G\) and \(H\) has the following structure:
– (optionally) a path in one of the copies of $H$ (edges of type 2), followed by one edge of type 3,
– a (possibly empty) path in $G$, with uncolored edges followed directly (as type 1), and colored edges represented by paths in copies of $H$ (each ending at $H$'s colored vertex, with a type 3 edge following it),
– (optionally) a path in one of the copies of $H$.

The Closed Computation Graph corresponding to the new machine on any input $X$ is of course the composition of the Quering Computation Graph of $M$ and the Closed Computation Graph of $N$ on $X$. Therefore its paths follow our observation. Requiring $C$ to be a subset of $D$ makes its ambiguity constraints apply to at least the types of paths we are concerned with. Making it one of the $\text{UNI}$ classes prevents $N$ from increasing the number of paths of interest in the overall computation within a single query processing. As we are about to show, with some precautions we can avoid any other paths of interest from appearing and thus complete the proof.

The cases in which we consider all paths (to either all reachable or all final vertices) are immediate consequences of Observation 4.3. If we count simple paths only, it is enough to notice that a cycle in the composition graph must mirror one in either of the components. The matters get slightly more complicated with minimum-length paths, as we must make some guarantees regardless of the time needed to process any $N$ queries. To achieve that, we introduce an additional counter tape, and we make every step of $M$ take an amount of time larger than all possible $N$ queries combined (in the query graph it might be seen as making type 1 and type 3 edges “longer”—i.e., replacing them with sequences of edges).

As $M$ uses space $f(2^{O(g(n))})$, it cannot take more than $2^{O(f(2^{O(g(n))}))}$ steps. If each of them was an oracle query, they would add up to at most $2^{O(f(2^{O(g(n))})+g(n))}$ steps. Therefore a counter of length $O(f(2^{O(g(n))})+g(n)) = O(f(2^{O(g(n))})$ is enough for the purpose. Now a minimum-length path in the new machine must be a minimum length path of $M$ augmented with some queries. Moreover, each of them has to be minimum-length within the query, or otherwise a shorter overall path would exist to the same configuration.

Using the above lemma we can justify the definition of our notion of reduction, showing that the right properties of computation graphs are maintained after the reduction is applied:

**Proposition 4.4.** For Closed Computation Graph classes $C \in \text{UNI}$ and $D \in \text{ALL}$, $C \subseteq D$, and transformations $\phi : \alpha \rightarrow \beta$ and $\psi : \gamma \rightarrow \delta$, if $\phi \in C\text{-QFunc}(f(n))$ and $\psi \in \mathcal{C}\text{-QFunc}(f(n))$, then $\phi \in D\text{-QFunc}(f(n))$.

5 Counting up to a constant number

We are now going to consider the (functional) problem of **path counting**. $\text{Count}$ will denote counting all paths of type $\mathcal{X}$ (following the notation for
ambiguity classes, e.g., SF denoting simple paths from start to colored vertices), taking the maximum over all start-end pairs.

In this work we are going to focus on bounded version(s) of counting—the problem \textbf{Count}_k\mathcal{X} will be the one of counting up to \(k\) paths (i.e., the set of answers being \(\{0, 1, \ldots, k - 1, k^+\}\), with \(k^+\) denoting “\(k\) or more”) of type \(\mathcal{X}\).\footnote{In this notation, a result of Allender, Reinhardt and Zhou (Theorem 5.1 in [7]) implies that for a polynomial \(p\), \textbf{Count}_{p\text{SF}} \in \text{QFunc}(\log(n))\) (i.e., limited counting can be solved nondeterministically in logarithmic space, but with no bounds on ambiguity).}

The canonical problem of reachability (denoted Reach) is of course equivalent to counting “up to one” path.

Most of the problems discussed might vary in difficulty when given different “promises” about the input graph. Therefore we employ the following consistent notation: \(\mathcal{C}-\alpha\) denotes the problem \(\alpha\) on graphs in class \(\mathcal{C}\).

How do counting problems for different values of \(k\) relate to each other? Obviously, decreasing the counter limit can only make the problem easier, as we can simply glue together the previously distinct answers. In the other direction, the following can be shown (recall that \(\text{REM}\) is the family of CCG classes closed under edge removal):

**Proposition 5.1.** For any class \(\mathcal{C} \in \text{REM}\) and constant \(k \geq 1\), \(\mathcal{C}\text{-Count}(k + 1)\text{SF} \preceq^\text{D} \mathcal{C}\text{-Count}^1\text{SF} \mathcal{C}\text{-Count}^k\text{SF}\).

In words, we show that given a graph \(G\) from a class \(\mathcal{C} \in \text{REM}\), and an algorithm for \(\mathcal{C}\text{-Count}^1\text{SF}\), we can create a sequence of graphs \((G_i)_i\) such that the answer to \(\mathcal{C}\text{-Count}(k + 1)\text{SF}(G)\) can be obtained deterministically from the answers \((\mathcal{C}\text{-Count}^k\text{SF}(G_i))_i\).

**Proof.** Let us first look at the case of \(k = 1\). Our algorithm works as follows:

on graph \(G\):
1. if \(\mathcal{C}\text{-Count}^1\text{SF}(G) = 0\), answer 0
2. for every edge \(e\) in \(G\), let \(G_e\) be the same as \(G\) but with \(e\) removed
3. for every edge \(e\) in \(G\), let \(c_e := \mathcal{C}\text{-Count}^1\text{SF}(G_e)\)
4. remove edges from \(G\), leaving only those for which \(c_e = 0\); call the result \(G'\)
5. answer \(2 - \mathcal{C}\text{-Count}^1\text{SF}(G')\) (note that 2 really means \(2^+\))

First, let us discuss the graph modification. The steps 2 to 4 are just a conceptual convenience—the graphs \(G_e\) and \(G'\) are never produced explicitly. Instead, whenever asked whether an edge \(e = (u, v)\) is in \(G'\), we answer “yes” if both \(e \in G\), and \(\mathcal{C}\text{-Count}^1\text{SF}(G - e) = 0\).

Now, if there is no path between the source \(s\) and the target \(t\) in \(G\), we will discover it in step 1. If there is exactly one such simple path, removing any of its edges would disconnect \(s\) from \(t\). Thus the same path is going to be present in \(G'\) and the algorithm will return 1. If there are at least two simple paths, consider the vertex \(x\) at which they diverge for the first time. Removing any single outgoing edge of \(x\) will not disconnect \(s\) and \(t\), and thus \(x\) will become
a sink in \( G' \). But as any path from \( s \) to \( t \) has to go through \( x \), there will be none, and our algorithm will correctly return 2. The procedure is thus sound and total. Moreover, as the only modification of the graph is removing edges and we have chosen \( \mathcal{C} \) to be one of the classes closed under this operation, all calls to \( \mathcal{C}\text{-Count}1\mathsf{SF} \) will have their promise fulfilled.

We can now proceed to higher values of \( k \). It is clear that we only need to distinguish the cases of “exactly \( k \)” and “\( k+1 \) or more” paths (the other answers can be copied exactly from \( \mathcal{C}\text{-Count}k\mathsf{SF} \)). Having at least 2 paths guarantees the existence of the first point of divergence, as discussed above. Moreover, the same way of deleting edges makes the vertices on the “common prefix” of the paths have out-degree 1 in \( G' \), which allows us to deterministically find the split-point \( x \). Now, \( x \) has at least two “meaningful” successors (on paths to the target)—thus if there are exactly \( k \) paths of interest, at most \( k-1 \) of them can pass through any of the successors. Therefore, if we modify the graph to leave exactly one of \( x \)’s outgoing edges (repeatedly for each of them), we can use \( \mathcal{C}\text{-Count}k\mathsf{SF} \) to determine the exact count of the paths of interest.

It is also possible to extend the above result to count all, instead of only simple, paths:

**Proposition 5.2.** For any class \( \mathcal{C} \in \mathbb{REM} \) and constant \( k \geq 1 \),

\[
\mathcal{C}\text{-Count}(k + 1)\mathsf{AF} \preceq \mathcal{D}\text{Count}1\mathsf{AF} \mathcal{C}\text{-Count}k\mathsf{AF}.
\]

**Proof.** First, let us note that if there is any non-simple path from the source to the target, we can obtain infinitely many paths by choosing the number of times we traverse its cycle. Therefore, knowing how to count simple paths, the problem of counting all paths becomes a matter of cycle detection. Let us recall the proof of Proposition 5.1 and look at the (only) path \( \pi \) leaving \( s \) in \( G' \).

If \( G \) contains a non-simple path from \( s \) to \( t \), the first vertex that is visited twice on that path must lie either on \( \pi \) or “after” (and thus be reachable from) the divergence point \( x \). In the latter case, the number of paths from one of the successors of \( x \) to \( t \) will be infinite, in which case the call to \( \mathcal{C}\text{-Count}k\mathsf{AF} \) will return “\( k^+ \)” and the whole procedure will correctly answer “(\( k+1 \)+)”. Thus we only need to detect a situation in which some vertex \( y \in \pi \) lies on a cycle, or equivalently, \( y \) is reachable from some successor \( z \) of \( y \). As we can deterministically enumerate over all vertices on \( \pi \) and all successors of each of them, it remains to show how we can answer the question of \( y \) being reachable from \( z \).

Let us then introduce an additional modification of our input graph, namely the change of source and target vertices. It is obvious that it can be done deterministically in \( \mathsf{QFunc}(n) \). Moreover, as we are guaranteed that the new source \( z \) is reachable from the old source \( s \), and likewise, the old target \( t \) is reachable from the new target \( y \), we can see that the “interesting” paths in the new graph form a subset of those in the old one. From this it follows that the new graph belongs to \( \mathcal{C} \), and thus we can simply use \( \mathcal{C}\text{-Count}1\mathsf{AF} \) to check whether \( y \) is reachable from \( z \).
Corollary 5.3. For any classes $C \in \text{REM}$, $D \in \text{UNI}$, and constant $k \geq 1$, $C$-Reach $\in D$-$\text{QFunc}(\log(n)) \iff C$-Count$kAF \in D$-$\text{QFunc}(\log(n))$, $C$-Reach $\in D$-$\text{QFunc}(\log(n)) \iff C$-Count$kSF \in D$-$\text{QFunc}(\log(n))$.

6 Inductive Counting

In this section we revisit the algorithms based on the technique called “inductive counting.” They all look at the vertices of the input graph reachable from the source $s$ in concentric “layers,” with layer $k$ (denoted by $L_k$) consisting of those whose distance (the length of the shortest path) from $s$ is at most $k$. Counting the vertices in these layers allows us to solve Reach—it is enough to compare the counts of $L_n$ for the graph with the target vertex present and removed.

Let us start with the breakthrough due to Immerman and Szelepcsényi (see [9, 10]). Denoting the number of vertices in $L_k$ by $C_k$, we can calculate $C_{k+1}$ from $C_k$ within $\text{QFunc}(\log(n))$ (guess denotes making a nondeterministic choice).

Algorithm 6.1 (Inductive Counting).

1. set $C_{k+1} := 1$
2. for every $v \in V - \{s\}$:
3. set $C_k' := 0$, $F := \text{false}$
4. for every $u \in V$:
5. guess whether $u \in L_k$, if not—move to the next $u$
6. guess a path from $s$ to $u$ of length $\leq k$ (or fail)
7. set $C_k' := C_k' + 1$
8. if $(u, v) \in E$, set $F := \text{true}$
9. if $C_k' < C_k$, fail
10. if $F = \text{true}$, set $C_{k+1} := C_{k+1} + 1$

It is not difficult to see that on all of the nondeterministic branches that have not failed, the value of $C_{k+1}$ has been computed correctly (again, see [9, 10] for details). Analyzing more carefully the nondeterministic branches on which the above algorithm may succeed, we can show more:

Proposition 6.2. $1\text{AA-Reach} \in 1\text{AF-QFunc}(\log(n))$, $1\text{SA-Reach} \in 1\text{SF-QFunc}(\log(n))$ and $1\text{MA-Reach} \in 1\text{MF-QFunc}(\log(n))$.

Proof. The guesses made in step 5 are of no consequence here, as there is only one way of guessing that will not lead to a failure later on. The only ambiguity is therefore introduced in step 6. But the guesses made there correspond to the paths in the input graph, and therefore any uniqueness promises about them yield analogous unambiguity properties of the accepting paths.

Note that the result of [5] is actually stronger, showing that $1\text{AA-Reach} \in 1\text{AA-QFunc}(\log(n))$. However, it is based on a different algorithm. Here we are trying to show that a careful analysis of the same algorithm in light of our framework might provide stronger results “for free”.

\[172\]
The Immerman and Szelepcsényi algorithm guesses paths between the same pairs of vertices over and over again, and thus the result does not immediately extend to higher path counts. However, we can modify the algorithm following Buntrock, Hemachandra and Siefkes (see [3]) to guess and verify all the paths to every reachable vertex. First, let us note that a graph can be easily (deterministically in log-space) transformed to a one which is equivalent w.r.t. the number of paths between vertices, and has out-degree 2. Having done this, and assuming that we have guessed the number \( p \) of distinct paths (of length at most \( l \)) between vertices \( u \) and \( v \), we can use the following procedure to verify their existence:

**Algorithm 6.3.**

\[ \text{guesspaths}(G, u, v, p, l): \]
1. if \( p = 1 \), guess a path from \( u \) to \( v \) of length \( \leq l \) and return
2. guess \( w \), the first divergence point of paths from \( u \) to \( v \)
3. guess a path from \( u \) to \( w \) (or fail), let \( l' < l \) be its length
4. let \( w' \) and \( w'' \) be the two successors of \( w \)
5. guess the number \( p' \) (\( 0 < p' < p \)) of distinct paths from \( w' \) to \( v \)
6. let \( p'' := p - p' \), \( l'' := l - l' - 1 \)
7. \[ \text{guesspaths}(G, w', v, p', l'') \]
8. \[ \text{guesspaths}(G, w'', v, p'', l'') \]

By simple inductive reasoning it can be seen that if \( p \) has been guessed correctly (too high), the algorithm will succeed on exactly one (none, respectively) computation branch. The situation of the \( p \) provided being too low is a bit less fortunate, as then the procedure might succeed on multiple computation branches (effectively guessing any \( p \) distinct paths from \( u \) to \( v \)). However, we can keep track of the collective number of paths \( T_k \) to all vertices in \( L_k \), and use it to cut off these “unfortunate” branches:

**Algorithm 6.4.**

1. set \( T_{k+1} := 1 \)
2. for every \( v \in V \):
3.   set \( T' := 0 \), \( r := 0 \)
4.   for every \( u \in V \):
5.     guess the number \( p \geq 0 \) of distinct paths from \( s \) to \( u \) of length \( \leq k \)
6.     \[ \text{guesspaths}(G, s, u, p, k) \]
7.     set \( T'_k := T'_k + p \)
8.     if \( \langle u, v \rangle \in E \), set \( r := r + p \)
9.     if \( T'_k < T_k \), fail
10. set \( T_{k+1} := T_{k+1} + r \)

It is not difficult to show that the value of \( T_{k+1} \) will be correctly computed on exactly one nondeterministic branch, and that all other branches will fail. Moreover, a smart use of tail recursion makes it possible to carry \text{guesspaths} (and thus the whole Algorithm 6.4) within space \( O(\log(np) \log(p)) \). Thus Buntrock et al. obtain the following:
Algorithm 6.1 considers all paths to every vertex. It is however possible to adapt it to consider only the minimal length paths (following Allender and Reinhardt, see [6]), by counting not only the number of vertices in \( L_k (C_k) \), but also the sum of lengths of the shortest paths to those vertices (denoted \( \Sigma_k \)). We refer the reader to [6] for details on this algorithm.

Knowing \( \Sigma_k \) in addition to \( C_k \) does not seem to have any interesting consequences. However, if there is a unique shortest path from \( s \) to every reachable vertex, the algorithm will compute the correct value on exactly one computation branch (as all the nondeterministic choices will have exactly one “valid” value). From here, Allender and Reinhardt conclude what in our terms can be expressed as the following:

**Corollary 6.6.** \( 1 \text{MA-Reach} \in 1 \text{AF-QFunc}(\log(n)) \).

As it turns out, the same “double counting” technique can be applied to Algorithm 6.4. First, we can make sure that \texttt{guesspaths()} considers only paths of length exactly \( l \) (instead of up to \( l \)). Then, making \( T_k (\Sigma_k) \) denote the collective number (sum of lengths, respectively) of all shortest paths to all vertices in \( L_k \), we can compute \( T_{k+1} \) and \( \Sigma_{k+1} \) from \( T_k \) and \( \Sigma_k \) as follows:

**Algorithm 6.7.**

1. set \( T_{k+1} := T_k, \Sigma_{k+1} := \Sigma_k \)
2. for every \( v \in V - \{s\} \):
3. \hspace{1em} set \( T_k' := 0, \Sigma_k' := 0, r := 0 \)
4. \hspace{1em} guess \( d \leq k + 2 \) (minimal \( d \) for which \( v \in L_d \), with \( k + 2 \) denoting “more than \( k + 1 \)”)
5. \hspace{1em} for every \( u \in V \):
6. \hspace{2em} guess whether \( u \in L_k \), if not—move to the next \( u \)
7. \hspace{2em} guess \( l \leq k \) (minimal for which \( u \in L_l \))
8. \hspace{2em} guess \( p \geq 1 \) (the number of distinct paths of length \( l \) from \( s \) to \( u \))
9. \hspace{2em} guesspaths\((G, s, u, p, l)\)
10. \hspace{2em} set \( T_k' := T_k' + p, \Sigma_k' := \Sigma_k' + lp \)
11. \hspace{2em} if \( \langle u, v \rangle \in E \), then
12. \hspace{3em} if \( l + 1 < d \), fail
13. \hspace{3em} if \( l + 1 = d \), set \( r := r + p \)
14. \hspace{2em} if \( T_k' < T_k \) or \( \Sigma_k' > \Sigma_k \), fail
15. \hspace{2em} if \( r = 0 \) and \( d \leq k + 1 \), fail
16. \hspace{2em} if \( d = k + 1 \), set \( T_{k+1} := T_{k+1} + r \) and \( \Sigma_{k+1} := \Sigma_{k+1} + dr \)

**Corollary 6.8.** \( 1 \text{MA-Reach} \in 1 \text{AF-QFunc}(\log(np) \log(p)) \).

**Corollary 6.9.** For a constant \( k \), \( k \text{MA-Reach} \in 1 \text{AF-QFunc}(\log(n)) \).
References

Notions of Reducibility for Mass Problems

Kojiro Higuchi¹ and Takayuki Kihara²

¹ Mathematical institute, Tohoku University, 6-3, Aramaki Aoba, Aoba-ku, Sendai, Miyagi, Japan
sa7m24@math.tohoku.ac.jp

² Mathematical institute, Tohoku University, 6-3, Aramaki Aoba, Aoba-ku, Sendai, Miyagi, Japan
sa7m10@math.tohoku.ac.jp

Abstract. Medvedev and Muchnik introduced reducibility notions for mass problems. We introduce new reductions which are weaker than Medvedev reducibility and stronger than Muchnik reducibility, and compare these degree structures restricted to Π⁰₁ classes in the Cantor space.

Key words: Classical computability and degree structures, Reducibilities and relative computation, Computational learning, Effectively closed sets

1 Reducibility Notions

1.1 Introduction and Preliminaries

The central interest in computability theory is the degrees of unsolvability of mathematical problems. In the process of this research, computability theorists have come up with many reducibility notions for decision problems; many-one reducibility, bounded truth-table reducibility, truth-table reducibility, weak truth-table reducibility, Turing reducibility and so on. Most of reducibility notions stronger than Turing degree focus attention, not just whether one is computable related to another or not, but what about the access mechanism of the oracle. For the research of reducibility notions for recursively enumerable sets, we refer the reader to Section X.7 and 8 in the book of Odifreddi [10].

The similar question also arises in the context of the research of the degrees of unsolvability of mass problems. A decision problem is a subset of ω, and a mass problem is a subset of ωω. We view a mass problem P ⊆ ωω as representing a set of solutions to some problem, where we do not require an existence of a solution and a uniqueness of a solution. Indeed, in most cases, we consider a mass problem with uncountably many solutions. To give the degrees of unsolvability among mass problems, Medvedev [8] introduced a new reducibility notion in 1955, calling it the degrees of difficulty. We say that a mass problem P is Medvedev reducible to another mass problem Q (written P ≤ₘ Q) if there
exists a computable functional $\Phi$ such that $\Phi(g)$ is a solution of $P$ for any solution $g \in Q$. Whereas, Muchnik [9] defined another reducibility notion for mass problems in 1963. We say that a mass problem $P$ is Muchnik reducible to another mass problem $Q$ (written $P \leq_{\text{w}} Q$) if for any solution $g \in Q$ there exists a computable functional $\Phi$ such that $\Phi(g) \in P$.

Most of research of degree structures of mass problems focus on a $\Pi^0_1$ definable one. $\Pi^0_1$ mass problems ($\Pi^0_1$ classes) naturally arise in many branches of mathematics and meta-mathematics. For example, for a consistent axiomatizable theory $T$, all theorems of $T$ become a computably enumerable decision problem, and all consistent completion of $T$ become a $\Pi^0_1$ mass problem. Actually, $\Pi^0_1$ classes have become a fundamental notion in computability theory. It is noteworthy that $\Pi^0_1$ classes arise naturally not only in meta-mathematics but also in many areas of mathematics just as recursively enumerable sets, for example, the problem of finding an prime ideal of a recursively enumerable ring, that of finding an ordering of a formally real field, and so on. $\Pi^0_1$ classes also have been founded in combinatorics, game theory, algorithmic information theory, non-monotonic logic, Logic Programming and so on. See also “$\Pi^0_1$ Classes in Mathematics” written by Cenzer and Remmel [4].

Cenzer and Hinman [3] showed that the $\Pi^0_1$ Medvedev degrees is dense. So Simpson [11] asked “Is the $\Pi^0_1$ Muchnik degrees also dense?” However, this question still remains as an open problem. The reason for this might be that the fact we cannot fix the computable functional in Muchnik reduction creates the potential for trouble to analyze the Muchnik degree structure. Namely, even if we assure that $A$ is Muchnik reducible to $B$, this fact does not give a knowledge about the algorithm solving $A$ by using the information of the solution of $B$ as the oracle. So we should consider, not just whether one is solvable related to another or not, but how to find algorithm to solve it. Now, we pay attention to the way to find an algorithm which computes some solution of $P$ via an oracle $g$, where $g \in Q$ is a given solution. Kihara [7] introduced new reducibility notions for mass problems in accordance with this point of view. At this point, by using tools of the Gold-style computational learning theory (inductive inference or learning in limit), we give the characterization of Muchnik reduction (lemma 4), and we show that we can obtain natural intermediate reductions between Medvedev reduction and Muchnik reduction. Furthermore, we note that this reductions obviously induce degree structures, and thus many problem on these degree structures arise. See question 1 and 2.

1.2 Notation and Conventions

Throughout this paper, a lower-case Roman character denotes some natural number, a capital-case Roman character denotes some set of natural numbers, and a capital-case Greek character denotes some computable functional whenever there is no mention about it. We first introduce some notations. Let $\{\Phi_e\}_{e \in \omega}$ be an effective enumeration of all partial computable functionals. For $\sigma \in 2^\omega$, we let $|\sigma|$ denote the length of $\sigma$, and let $\sigma^-$ denote an immediate predecessor node of $\sigma$, i.e. $\sigma^- = \sigma \upharpoonright (|\sigma| - 1)$. We also define $[\sigma] = \{f \in 2^\omega : f \supset \sigma\}$ and
\( (\sigma) = \{ \tau \in 2^{<\omega} : \tau \supseteq \sigma \} \). Let \( T(\sigma) \) denote \( T \cap (\sigma) \). For any tree \( T \subseteq 2^{<\omega} \), we also let \( [T] \) be the set of all infinite paths of \( T \). For any tree \( T \) and any \( n \), \( T^n \) denotes the set of members of \( T \) of length \( n \), and \( (\sigma)^n = \{ \tau \in 2^n : \tau \supseteq \sigma \} \). For any \( k, \sigma \supseteq k \) and \( k \supseteq \sigma \) are abbreviation for \( \sigma \supseteq \langle k \rangle \) and \( \langle k \rangle \supseteq \sigma \) respectively. We set \( \sigma \supseteq T = \{ \sigma \supseteq \tau : \tau \in T \} \). We say that \( \sigma \in T \) is a \textit{leaf} if there is no \( \tau \in T \) with \( \tau \supseteq \sigma \). Let \( L(T) \) denote all leaves of a tree \( T \). We say that a mass problem \( P \subseteq 2^{<\omega} \) is \( \Pi^0_1 \) class if there is a computable relation \( R \) such that \( P = \{ X : (\forall n) R(n, X) \} \) holds. In particular, a \( \Pi^0_1 \) class \( P \) is \textit{special} if \( P \) is nonempty and \( P \) has no computable member. The following lemma is useful.

**Lemma 1** (see Alfeld [2]). For a mass problem \( P \subseteq 2^{<\omega} \), the following are equivalent.

1. \( P \) is a \( \Pi^0_1 \) class.
2. \( P = [T_P] \) for some computable binary tree \( T_P \).
3. There is a computable \( \subseteq \)-decreasing sequence of computable binary trees \( \{T_i\}_{i \in \omega} \) without dead ends such that \( P = \bigcap_{i \in \omega} [T_i] \).
4. There is a computable \( \subseteq \)-increasing sequence of finite binary trees \( \{T_i\}_{i \in \omega} \) such that \( P = \bigcup_{i \in \omega} T_i \), where there is a computable function \( l \) such that \( L(T_i) \subseteq 2^{\leq l(i)} \) and \( L(T_i^{<\omega}) = L(T_{i+1})^{<\omega} \) hold for any \( i \), and \( L^{<\omega} \) denotes a set of all nodes of \( L \) with height \( <k \).

### 1.3 New Reducibility Notions

Now, we introduce new reductions which are weaker than Medvedev reducibility and stronger than Muchnik reducibility.

**Definition 1** (Kihara [7]). Let \( P \) and \( Q \) be subsets of \( \omega^{<\omega} \).

1. A reduction \( P \leq_b Q \) denotes that \( P \leq_w Q \) and we know such algorithm is in bounded range, i.e.

\[
P \leq_b Q \iff (\exists b)(\forall f \in Q)(\exists e < b) \Phi_e(f) \in P.
\]

2. A reduction \( P \leq_l Q \) denotes that \( P \leq_w Q \) and such algorithm is learnable in the limit, i.e.

\[
P \leq_l Q \iff (\exists \Psi)(\forall f \in Q) \Phi_{\lim_n \Psi(f \upharpoonright n)}(f) \in P.
\]

3. A reduction \( P \leq_{bl} Q \) denotes that \( P \leq_w Q \) and such algorithm is learnable in the limit, where the learner changes his mind only finitely many times:

\[
P \leq_{bl} Q \iff (\exists \Psi)(\exists b)(\forall f \in Q) \left( \Phi_{\lim_n \Psi(f \upharpoonright n)}(f) \in P \right.
\]

\[
\left. \land |\{ n \in \omega : \Psi(f \upharpoonright n) \neq \Psi(f \upharpoonright n + 1) \}| < b \right).
\]

4. A reduction \( P \leq_{bel} Q \) denotes that \( P \leq_w Q \) and such algorithm is learnable in the limit with bounded errors:

\[
P \leq_{bel} Q \iff (\exists \Psi)(\exists b)(\forall f \in Q)
\]

\[
( \Phi_{\lim_n \Psi(f \upharpoonright n)}(f) \in P \land |\{ \Psi(f \upharpoonright n) : n \in \omega \}| < b )
\].
5. A reduction \( P \leq_{tl} Q \) denotes that \( P \leq_{w} Q \) and such algorithm is learnable by a team of learners:

\[
P \leq_{tl} Q \iff (\exists \Psi_0, \ldots, \Psi_b)(\forall f \in Q)(\exists j \leq b) \Phi_{\lim_n, \psi_j(f \upharpoonright n)}(f) \in P.
\]

6. A reduction \( P \leq_{li} Q \) denotes that \( P \leq_{w} Q \) and such algorithm is infimum limit of some algorithm. i.e.

\[
P \leq_{li} Q \iff (\exists \Psi)(\forall f \in Q) \Phi_{\lim_{inf_n, \psi(f \upharpoonright n)}}(f) \in P.
\]

In the above definition, subscripts \( b, l, bl, tl \) and \( li \) mean bounded, learning, learning with bounded mind-change, learning with bounded error, learning by a team and infimum limit, respectively. We call such a computable function \( \Psi : 2^{<\omega} \rightarrow \omega \) a learner. We use \( \Psi \) and \( \Psi^* \) as a symbol denoting some learner. Note that, not confusing the meanings of \( \Psi(\sigma) \) and \( \Phi(\sigma) \) for a learner \( \Psi \) and a computable functional \( \Phi \). In the former case, \( \Psi(\sigma) \) simply denotes the output (the inference) of the learner \( \Psi \) based on the current input \( \sigma \). In the latter case, however, we use \( \sigma \) as an initial segment of some oracle information, and so really \( \Phi(\sigma) \) means a string \( \langle \Phi(\sigma;0), \Phi(\sigma;1), \Phi(\sigma;2), \ldots \rangle \). Now, we study properties of these reductions.

**Lemma 3.** All reducibilities defined above are preorders. \( \square \)

**Lemma 4.** For any \( P, Q \subseteq \omega^\omega \), the following conditions hold:

1. \( P \leq_s Q \rightarrow P \leq_b Q \rightarrow P \leq_{tl} Q \rightarrow P \leq_{w} Q \),
2. \( P \leq_s Q \rightarrow P \leq_{bl} Q \rightarrow P \leq_{bel} Q \rightarrow P \leq_{l} Q \rightarrow P \leq_{tl} Q \rightarrow P \leq_{li} Q \rightarrow P \leq_{w} Q \).

**Proof.** The “only if” part is obvious. For the “if” part, we will inductively define \( \Psi(\sigma) \) and \( l(\sigma, e) \) for each \( \sigma \in 2^{<\omega} \) and \( e \in \omega \). First, let \( \Psi(\emptyset) = 0 \) and we define \( l(\emptyset, e) = 0 \) for each \( e \). We assume that, for any \( \tau \in 2^{<\omega} \) with \( |\tau| < |\sigma| \), we have already defined \( \Psi(\tau) \), and also defined \( l(\tau, e) \) for each \( e \in \omega \). Then, we define \( \Psi(\sigma) \) and \( l(\sigma, e) \) for each \( e \) as follows:

\[
\Psi(\sigma) = \begin{cases} 
\mu e < |\sigma| \quad & \text{if such } e \text{ exists,} \\
|\sigma| & \text{o.w.}
\end{cases}
\]

\[
l(\sigma, e) = \begin{cases} 
\mu e < |\sigma| \quad & \text{if } e = \Psi(\sigma), \\
l(\sigma^-, e) & \text{o.w.}
\end{cases}
\]

By our assumption \( Q \leq_{w} P \), \( \lim_{inf_n} \Psi(f \upharpoonright n) \) exists for all \( f \in P \). Thus \( \Phi_{\lim_{inf_n}, \psi(f \upharpoonright n)}(f) \in Q \) holds. So \( Q \leq_{li} P \) since \( \Psi \) is computable. \( \square \)

**Theorem 1.** For any mass problems \( P \) and \( Q \), if \( P \leq_{bel} Q \) holds then \( P \leq_{b} Q \) holds.
Proof. We assume $P \leq_{bel} Q$ via $b$ and $\Psi$. Let $A$ be a set $\{ (k, l) : l < k < b \}$ and notice that this set is finite. Now, we define an algorithm $\Phi_{k,l}$ for any $(k, l) \in A$, and we ensure the following property:

$$(\forall g \in Q)(\exists (k, l) \in A) \Phi_{k,l}(g) \in P.$$

The algorithm $\Phi_{k,l}$ considers that

- the number of algorithms occurring in hypothesis of the learner $\Psi$ is just $k$,
- and the limit of hypothesis formed by the learner $\Psi$ is the $l$-th smallest one among such $k$ algorithms.

Formally, the algorithm $\Phi_{k,l}$ proceeds as follows for $g$. Let $B_n^g$ be a set $\{ e : (\exists m \leq n) e = \Psi(g \upharpoonright m) \}$, $p_k^g = \min \{ |B_n^g| : k \}$, and $e_{k,l}^g$ be the $l$-th smallest value of $B_{p_k^g}^g$. $\Phi_{k,l}(g)$ computes $\Phi_{e_{k,l}^g}(g)$. Obviously, $\Phi_{k,l}$ is a computable functional. Furthermore, for any $g \in Q$, the mind of $\Phi(g \upharpoonright *)$ changes just $k$ times for some $k$ and $\lim_n \Phi(g \upharpoonright n)$ converges to the right index $e$ of an algorithm which solves the mass problem $P$. We notice that such $e$ is equivalent to $e_{k,l}^g$ for some $l < k$. Thus, for any $g \in Q$, there are $k$ and $l$ such that $\Phi_{e_{k,l}^g}(g)$ computes a solution of $P$. So, if $i_{k,l}$ is an index of $\Phi_{k,l}$ then we obtain $P \leq_b Q$ via $b = \max \{ |i_{k,l} : (k, l) \in A| \}$. \hfill \Box

Now, it is natural that we take a keen interest in the degree structures induced by reductions we defined. For these structures, some properties are easily seen. Indeed, we can obviously generalize the proof of density of $s$-degrees of Cenzer-Hinnman [3], and show densities of $b$-degrees and $bl$-degrees. Namely, the following statement holds.

**Proposition 2** Suppose $r$ is bl or b. For any special $\Pi^0_1$ classes $P$ and $Q$ with $P <_r Q$, there exists a $\Pi^0_1$ class $R$ such that $P <_r R <_r Q$.

**Question 1.**

1. Is the reducibility relation $\leq_l$ dense with respect to non-empty $\Pi^0_1$ classes?

2. Is the first-order theory (or some fragment of that) of the $\Pi^0_1$ degree structure induced by any of reductions decidable?

## 2 Separating Results

### 2.1 $\leq_{bl}$ is Stronger than $\leq_{bel}$

**Theorem 3.** There exist $\Pi^0_1$ classes $P$ and $Q$ such that $P \equiv_{bel} Q$ holds but $P \not\leq_{bl} Q$ holds.

**Proof (Sketch).** For $P \subseteq 2^\omega$, we construct $Q \supseteq P$, and $Q$ is a subset of $6^\omega$. First, we make special $\Pi^0_1$ classes $P_0, P_1 \subseteq 2^\omega$ so that every $f \in P_0$ and $g \in P_1$ are Turing incomparable, and let $P$ be $0^\omega \cup 1^\omega$. We know that any $\Pi^0_1$ class $P$ has a computable binary tree $T_P$ with $P = [T_P]$ by lemma 1. For a tree $T$, let $T^{(i+1)} \subseteq \{ i, i + 1 \}^{<\omega}$ denote a tree satisfying $\sigma \in T \iff \sigma^{(i+1)} \in T^{(i+1)}$ for any $\sigma$, where $\sigma(n) + 1 = \sigma(n^{(i+1)})$ for each $n < |\sigma| = |\sigma^{(i+1)}|$. And then, we use trees $T^{(i+2)}_{P_0} \subseteq \{ 2, 3 \}^{<\omega}$ and $T^{(i+4)}_{P_1} \subseteq \{ 4, 5 \}^{<\omega}$.
Requirements  We need to construct a $\Pi^0_1$ class $Q$, meeting the following for all $k \in \omega$ and all computable functions $\Psi$:

$$R : (\exists \Psi^*)(\forall f \in Q) \left( (\Phi_{\Psi^*}(f \upharpoonright n))(f) \in P \land |\{e : (\exists n) e = \Psi^*(f \upharpoonright n)\}| < 3\right).$$

$$P^b_\Psi : (\forall f \in Q) \left( (\Phi_{\lim_n \Psi(f \upharpoonright n)}(f) \in P) \implies (\exists f \in Q) \left(|\{n : \Psi(f \upharpoonright n + 1) \neq \Psi(f \upharpoonright n)\}| > b\right).$$

Our construction proceeds in stages. A stage $s$ consists of substage $t \leq s$. At substage $t$, a strategy to satisfy the $t$-th requirement works. A strategy is said to be a S-strategy if it works for the requirement $S$. Our construction builds a uniformly computable binary tree sequence $P = Q[0] \subseteq Q[1] \subseteq Q[2] \subseteq \ldots$, and we define $Q = \bigcup_s Q[s]$, where, for any node of $Q[s]$, its length is less than or equal to $s$, and if $i \in \sigma$ does not belong to $Q[s]$ for any $i < 6$ for $\sigma \in Q<s$, then no nodes extending from $\sigma$ belong to $Q[s+1]$. This construction makes $Q$ be a $\Pi^0_1$ class.

First, we enumerate all $P$-requirements. At stage $s$, we will construct a tree $Q[s]$ whose all nodes are less than or equal to the length $s$. At substage $t$ of stage $s$, many strategies for the $t$-th $P$-requirement act. Indeed, for each $\beta \in T^s_P$, each node $\alpha \in T^s_Q(\beta)[s]$ acts to satisfy the $t$-th $P$-requirement. Each strategy $\alpha$ has six parameters, $\text{Start}(\alpha)$, $i^\alpha$, $n^\alpha$, $\rho^\alpha_0$, $\rho^\alpha_1$, and $\sigma^\alpha$. Then, for any parameter $p$ of $\alpha$, $p^\alpha[s]$ denotes the value of $p^\alpha$ at stage $s$, where $\text{Start}(\alpha)[s]$ specify a starting location for the action of the strategy $\alpha$ at stage $s$, and we ensure $\rho^\alpha_0[s] \subseteq \{2,3\}^{<\omega}$ and $\rho^\alpha_1[s] \subseteq \{4,5\}^{<\omega}$. Intuitively, the parameter $\rho^\alpha_0[s]$ means a node of a tree $T^\alpha_P$ which is already coded into $\sigma^\alpha[s]$ at stage $s$.

We assume that the $t$-th $P$-requirement is $P^b_\Psi$. Now, each $P^b_\Psi$-strategy $\alpha$ proceeds as follows, where $p^\alpha$ is abbreviation for $p^\alpha[s]$ for each parameter $p$.

0. In the case $\text{Start}(\alpha)[s] = 0$, define $i^\alpha[s+1] = 0$ and $\rho^\alpha_0[s+1] = \rho^\alpha_1[s+1] = \emptyset$. Furthermore set $\sigma^\alpha[s+1] = \beta^{-2}$, where this action signals us to start connecting the tree $T^\alpha_P$ to the node $\beta^{-2}$, and then we ensure $\beta^{-2} T^\alpha_P \subseteq T^\alpha_Q$ if $\beta$ is an extendible node of $T^\alpha_P$. We set $\text{Start}(\alpha)[s+1] = 1$.

1. In the case $\text{Start}(\alpha)[s] = 1$, we see whether or not

$$\Phi_{\Psi(\alpha)}(\alpha)[s] \upharpoonright n^\alpha + 1 \in T^\alpha_P(<i^\alpha)).$$

(1)

If the above formula holds, then we define $\sigma^\alpha[s+1] = \alpha$, where this actions signal us to start connecting the tree $T^\alpha_P(\rho^\alpha_1^{-i^\alpha})$ to the node $\sigma^\alpha[s+1]$. We can ensure $\sigma^\alpha[s+1] T^\alpha_P(\rho^\alpha_1^{-i^\alpha}) \subseteq T^\alpha_Q$ when $\alpha$ extends from some extendible node of $T^\alpha_P$, and both $\rho^\alpha_0$ and $\rho^\alpha_1$ are extendible nodes of $T^\alpha_P$ and $T^\alpha_Q$, respectively. And then, set $\rho^\alpha_n[s+1] = \rho^\alpha_n^{-i^\alpha}(\sigma^\alpha[s+1])$, where $(\sigma, \tau)(m) = \tau(m + |\sigma|)$ for each $m$. Furthermore, we switch from $i^\alpha$ to $1 - i^\alpha$ for a parameter $i^\alpha[s+1]$. Moreover, we define a mind-change $m^\alpha[s+1]$ as follows:

$$m^\alpha[s+1] = |\{n < |\sigma^\alpha| : \Psi(\sigma^\alpha[s+1] \upharpoonright n + 1) \neq \Psi(\sigma^\alpha \upharpoonright n + 1)\}|.$$
2. In the case $\text{Start}(\alpha)[s] = 2$, the strategy $\alpha$ does not act.

At the end of stage $s > 0$, for each parameter $p$ of $\alpha$, if $p^\alpha[s + 1]$ is undefined then we define $p^\alpha[s + 1]$ to be $p^\alpha[s]$. Now, we define $T_Q[s + 1]$ as follows, where $p^\alpha$ is abbreviation for $p^\alpha[s + 1]$ for each parameter $p$ of $\alpha$.

$$T_Q[s + 1] = T_Q[s] \cup T_P^{s+1} \cup \{\sigma^\alpha \rightarrow \tau \in (\alpha)^{s+1} : \alpha \in T_Q^s[s] \land \tau \in T_P^s(\rho_{\pi,\alpha})\}.$$

Finally, we define $p^\alpha[s + 1] = p^\alpha^\rightarrow[s + 1]$ for each node $\alpha \in T_Q^{s+1}[s + 1]$ and for any parameter $p$.

We define a learner $\Psi^*$ as follows:

- If the last bit of $\tau$ is 0 or 1, then $\Psi^*(\tau)$ outputs an algorithm $\Gamma$ with $\Gamma(\text{g}; n) = g(n)$ for each $g$ and $n$.
- If the last bit of $\tau$ is 2 or 3, then $\Psi^*(\tau)$ outputs an algorithm $\Gamma_0$, where $
\Gamma_0(\text{g}; 0) = 0$ and $\Gamma_0(\text{g}; n) = g \uparrow \{2, 3\}^{\omega}$ for each $g$ and $n > 0$.
- If the last bit of $\tau$ is 4 or 5, then $\Psi^*(\tau)$ outputs an algorithm $\Gamma_1$, where $\Gamma_1(\text{g}; 0) = 1$ and $\Gamma_1(\text{g}; n) = g \uparrow \{4, 5\}^{\omega}$ for each $g$ and $n > 0$.

We complete the construction. $\square$

2.2 $\leq_t$ is Not Weaker than or Equal to $\leq_b$

**Theorem 4.** There exist $\Pi_1^0$ classes $P$ and $Q$ such that $P \leq_b Q$ holds but $P \not\leq_t Q$ holds.

**Proof (Sketch).** This proof is similar to the proof of theorem 3.

**Requirements** We need to construct a $\Pi_1^0$ class $Q$, meeting the following for all computable functions $\Psi$:

$$R : (\exists e_0, e_1, e_2)(\forall f \in Q)(\exists i < 3) \Phi_{e_i}(f) \in P.$$

$$P_{\Psi} : (\exists f \in Q) \Phi_{\lim_n \Psi(f|n)}(f) \notin P.$$

Our construction is similar to the construction of theorem 3, except that we do not need to care mind-changes. We let $\Phi_{e_i}(f)$ be an algorithm extracting all bits of $\{2i, 2i + 1\}$ from $f$. Namely, $\Phi_{e_i}(f) = f \uparrow \{2i, 2i + 1\}^{\omega}$, where $(f \uparrow \{2i, 2i + 1\}^{\omega})(n) + 2i$ is the value of the $n$-th least digit among all digits in $f$ whose value belongs to $\{2i, 2i + 1\}$. $\square$

**Corollary 5** 1. There exist $\Pi_1^0$ classes $P$ and $Q$ such that $P \leq_{tt} Q$ holds but $P \not\leq_t Q$ holds.

2. There exist $\Pi_1^0$ classes $P$ and $Q$ such that $P \leq_b Q$ holds but $P \not\leq_{bel} Q$ holds.
\section*{2.3 \( \leq_b \) is Not Weaker than or Equal to \( \leq_t \)}

**Theorem 6.** For any special \( \Pi_1^0 \) class \( P \), there is a \( \Pi_1^0 \) class \( Q \) such that \( P \equiv_t Q \) holds but \( P \not\leq_b Q \) holds.

**Proof (Sketch).** Let \( P \) be an special \( \Pi_1^0 \) class. Now, we construct \( \Pi_1^0 \) class \( Q \supset P \) satisfying following requirements.

**Requirements** We need to construct a \( \Pi_1^0 \) class \( Q \), meeting the following for all \( b \in \omega \):

\[
R : (\exists f)(\forall \sigma \in Q) \Phi_{\lim_n \Psi(f \mid n)}(f) \in P.
\]

\[
P_b : (\exists f \in Q)(\forall j < b) \Phi_j(f) \notin P.
\]

First, we enumerate all \( P \)-requirements. Let \( L \) be an infinite recursive set of all leaves of \( T_P \). We pick \( \sigma_b \) as the \( b \)-th node of \( L \) for each \( b \)-th requirement \( P_b \). At stage \( s \), we will constructs a tree \( Q[s] \) whose all nodes are less than or equal to the length \( s \). At substage \( t \) of stage \( s \), only one strategy \( \sigma_t \) for the \( t \)-th \( P \)-requirement acts. Each strategy \( \alpha \) has three parameters, \( \text{Start}(\alpha), n^\alpha \), and \( \sigma^\alpha \). Then, for any parameter \( p \) of \( \alpha \), \( p^\alpha[s] \) denotes the value of \( p^\alpha \) at stage \( s \), where \( \text{Start}(\alpha)[s] \) specify a starting location for the action of the strategy \( \alpha \) at stage \( s \).

We assume that the \( t \)-th \( P \)-requirement is \( P_t \). Now, each \( P_t \)-strategy \( \alpha = \sigma_t \) proceeds as follows, where \( p^\alpha \) is abbreviation for \( p^\alpha[s] \) for each parameter \( p \).

**0.** In the case \( \text{Start}(\alpha)[s] = 0 \), define \( n^\alpha[s+1] = 0 \) and \( \sigma^\alpha[s+1] = \sigma_t \), where this action signals us to start connecting the tree \( T_P \) to the node \( \sigma_t \). However, we remark that we does not ensure \( \sigma_t \cap T_P \subseteq T_Q \). We set \( \text{Start}(\alpha)[s+1] = 1 \).

**1.** In the case \( \text{Start}(\alpha)[s] = 1 \), we see whether or not

\[
(\forall \tau \in T_Q^{s \leq s}(\sigma^\alpha))(\exists j < b) \Phi_j(\tau)[s] \upharpoonright n^\alpha + 1 \in T_P.
\]

If the above formula holds, then, we pick the minimal \( j \) among such \( j \)'s for all \( \tau \in T_Q^{s \leq s}(\sigma^\alpha) \). Furthermore, we also pick \( \tau \in T_Q^{s \leq s}(\sigma^\alpha) \) with \( \Phi_j(\tau)[s] \upharpoonright n^\alpha + 1 \in T_P \). And then, we define \( \sigma[s+1] = \tau \). Intuitively, this actions signal us to start connecting the tree \( T_P \) to the node \( \sigma^\alpha[s+1] \) and trimming all nodes which extend from \( \sigma^\alpha \) and are nestled along \( \sigma^\alpha[s+1] \). Moreover, we set \( n^\alpha[s+1] = n^\alpha + 1 \) and \( \text{Start}(\alpha)[s+1] = 1 \) again.

At the end of stage \( s > 0 \), for each parameter \( p \) of \( \alpha \), if \( p^\alpha[s+1] \) is undefined then we define \( p^\alpha[s+1] \) to be \( p^\alpha[s] \). Now, we define \( T_Q[s+1] \) as follows, where \( p^\alpha \) is abbreviation for \( p^\alpha[s+1] \) for each parameter \( p \) of \( \alpha \).

\[
T_Q[s+1] = T_Q[s] \cup T_P^{s+1} \cup \{ \tau \in 2^{s+1} : (\exists \alpha \in L)(\tau \subseteq \sigma^\alpha \vee \tau \in T_P(\sigma^\alpha)) \}.
\]

We define a learner \( \Psi \) as follows:

- If no \( \alpha \subseteq \tau \) with \( \alpha \in L \) exists, then \( \Psi(\tau) \) outputs an algorithm \( \Gamma \) with \( \Gamma(g; n) = g(n) \) for each \( g \) and \( n \).
Corollary 7  1. There exist $\Pi_1^0$ classes $P$ and $Q$ such that $P \leq_{ul} Q$ holds but $P \not\leq_b Q$ holds.
2. There exist $\Pi_1^0$ classes $P$ and $Q$ such that $P \leq_{bl} Q$ holds but $P \not\leq_1 Q$ holds.

2.4 $\leq_{ul}$ is Stronger than $\leq_w$

Theorem 8. There exist $\Pi_1^0$ classes $P$ and $Q$ such that $P \leq_w Q$ holds but $P \not\leq_{ul} Q$ holds.

Proof (Sketch). First, we arrange infinite many $\Pi_1^0$ classes $\{P_i\}_{i \in \omega}$ so that every $f \in P_i$ and $g \in P_j$ are Turing incomparable for each $i \neq j$ (such $\Pi_1^0$ classes exist by Jockusch-Soare [6]'s theorem 4.1), and let $P$ be a $\Pi_0^0$ classes such that $P = \{S \cup \bigcup_{i}(\beta_i \setminus P_i) \cup \bigcup_{i}(\delta_i \setminus P_i)\}$, where $\beta_i \subseteq \beta_{i+1}$ and $\beta_i \subseteq \delta_{i+1}$ for each $i$ and $S = \bigcup_{i} \beta_i \equiv_T \emptyset$ (such $P$ exists by the proof of Alfeld [1]'s theorem 6.3).

We note that any path $f$ of $P_i$ does not compute $S$ for any $i$ since otherwise $f$ computes some path in $P_j$ for any $j$ whose degree is low $\leq_T \emptyset'$, and it contradicts our definition of $P_i$ and $P_j$.

Our construction is similar to the construction of theorem 3, except that we do not need to care mind-changes, $Q$ is a subset of $\omega^\omega$ with $Q = [T_Q]$ for some recursively bounded computable tree $T_Q$, and the strategy $\alpha$ which works for a $P^t_\Psi$-requirement $[(\exists f \in Q)(\forall j < t)\Phi_{\lim_n, \psi_{j(f(n)}}(f) \not\in P]$ acts with $2^t$ trees $\{T_{P_i}\}_{i < 2^t}$ and $2^t + 2t + 2$ parameters $\{\text{Start}(\alpha)\}, \{i_k^\alpha\}_{k < t}, \{n_k^\alpha\}_{k < t}, \{\mu_k^\alpha\}_{k < 2^t}$ and $\sigma^\alpha$. Let $i^\alpha = \sum_{k=0}^{t-1} 2^k \cdot i_k^\alpha$, replace $(1)$ by $\Phi_{\Phi_{\pi}(\alpha)}(\alpha)[s] \upharpoonright n_k^\alpha + 1 \in T_P(\beta_i^\alpha) \cup T_P(\delta_i^\alpha)[s]$, and pick $k$ having the least $n_k^\alpha$ in $k$'s satisfying above formula. Let $n_k^\alpha[+1] = n_k^\alpha + 1$ and $i_k^\alpha[s+1] = 1 - i_k^\alpha$, and our construction proceeds as lemma 3.

2.5 $\leq_s$ is Stronger than $\leq_{bt}$

Theorem 9. There exists $\Pi_1^0$ classes $P$ and $Q$ such that $P \leq_{bt} Q$ holds but $P \not\leq_s Q$ holds.

This theorem follows from the following fact which can be proved by using finite injury priority argument: There exist $\Pi_1^0$ classes $P_0$ and $P_1$ such that $P_0 \cup P_1 <_s 0^\bot P_0 \cup 1^\bot P_1$ hold.

2.6 $\leq_b$ is Much Stronger than $\leq_w$

Theorem 10. For any special $\Pi_1^0$ class $Q$, there exists infinitely many $\Pi_1^0$ classes $\{P_i\}_{i \in \omega}$ such that $Q \equiv_w P_i$ but $Q \not\equiv_b P_i \not\equiv_b P_j$ for each $i \neq j$.

We can prove this theorem by using finite injury priority argument. This theorem asserts that, for nonempty $\Pi_1^0$ classes, any nonzero $w$-degree has infinitely many $b$-degrees.
3 Conclusion and Open Questions

We finished comparing reducibility notions defined by us (see Fig.1). Of course, many other questions naturally comes up. We list such questions for our reducibility notions (see question 2).

![Diagram]

**Fig. 1.** Comparison of reducibility notions for $\Pi^0_1$ mass problems.

**Question 2.**
1. Does there exist a non-zero degree w.r.t. one reduction including only one degree w.r.t. another reduction?
2. Which $\Pi^0_1$ degree structure has the greatest element above all positive measure $\Pi^0_1$ classes? (This does not hold in the $s$-degree, however this holds in the $w$-degrees, and actually also does in the $l$-degrees.)
3. Which reduction can reduce $\text{DNR}_2$ to $\text{DNR}_k$ for each $k \in \omega$? ($s$-reduction cannot do it, but $w$-reduction and actually $tl$-reduction also can do it.)

**References**

Computation by Synthetic Cell Signaling and Oscillating Processes Modelled using Mass-Action Kinetics

T. Hinze¹, R. Faßler¹, G. Escuela², B. Ibrahim³, and S. Schuster¹

¹Friedrich Schiller University Jena, Department Bioinformatics, D-07743 Jena
²Friedrich Schiller University Jena, Bio Systems Analysis Group, D-07743 Jena
³German Cancer Research Center, Molecular Biology of Centrosomes and Cilia, D-69120 Heidelberg

{hinze,raf,schuster}@minet.uni-jena.de, gabi.escuela@uni-jena.de, b.ibrahim@dkfz-heidelberg.de

Abstract. Biocomputing can cope with the challenge of efficiently programming minimal autonomous systems lacking central control. In this context, the chemical programming metaphor incorporates encoding techniques into molecular or spatial structures concerning artificial reaction networks. Inspired by the idea of constructing chemical computers based on minimal chemistry requirements, we introduce a design principle for synthetic chemical register machines operating on binary numbers. The architecture of such principle is composed of cell signaling network motifs which act as toggle switches flanked by oscillators. Alongside dynamical simulations, we present a case study – addition of binary numbers – which illustrates the practicability of our design principle.

Keywords: chemical computing, cell signaling network, mass-action kinetics, Belousov-Zhabotinsky reaction, Brusselator, register machine

1 Introduction

For more than 20 years, decentralized data processing systems in which molecules form the storage medium and (bio)chemical reactions modify these molecules in terms of executing a computation have been studied [9, 15]. Theoretical investigations as well as experimental implementations in vitro, in vivo, and in silico established separate fields of research according to supplementary assumptions in chemistry: While DNA- and RNA-based computing approaches utilize the convertibility of polymeric structures [14], membrane- or cell-based computers are focused on the presence of interconnected compartments able to separate molecules and reactions as part of the computational process [13]. Additionally, genetic circuits employ inhibition and activation of gene expressions [4, 8, 10].

All these fields of research have in common the usage of specific assumptions beyond pure chemistries in order to achieve Turing-complete models for computation. Although those assumptions are essential for successful wetlab implementations, the question about minimal requirements in chemistry arises
for construction of universal chemical computers from a theoretical perspective. Following an obvious intuition, the number of molecular copies within a species can represent any natural number and therefore encode unbounded memory contents. From an engineering point of view, small differences between huge discrete amounts of molecules are hard and error-prone to detect. An idea to overcome this insufficiency by adopting engineering principles of electronic circuits for construction of chemical computational units was introduced in [12] restricted to single logic gates. Bridging the gap between single logic gates and register machines can be seen as a challenging task in terms of chemical computing. For instance, weakening of chemical signals by decreasing molecular concentrations over time in conjunction with lack of synchronization can affect the composition of logic gates in complex circuits. We introduce a hierarchical design principle for synthetic chemical register machines based on recurrent motifs found in cell signaling and metabolic networks. Embedded oscillating reactions provide clock signals for synchronization. Furthermore, a binary encoding of natural numbers representing register contents is used. The number of registers as well as their length can be arbitrarily chosen. Instructions are represented by specific molecular species whose interactions emulate the register machine program control synchronized by two offset clocks. The dynamical behavior of the register machine is modeled using mass-action kinetics assuming molecular concentrations instead of discrete particle numbers. Further related work includes chemical abstract machines as concurrent systems without kinetics [3], and chemical description of neural Turing machines [11].

The paper is structured as follows: Section 2 defines a denotation of register machines operating on binary numbers. As further prerequisites, cell signaling networks as reaction systems and mass-action kinetics for description of their dynamical behavior are briefly introduced. Subsequently, a representation of boolean values as molecular species concentrations is given. Section 3 adopts this encoding scheme together with an extension of the Belousov-Zhabotinsky reaction [2,16] in order to obtain clock signals. Furthermore, a chemical implementation of clock-controlled master-slave flip-flops and their extension to binary registers is discussed in Section 4, while Section 5 explains the program control of the chemical register machine. Finally, integer addition is presented as an example for simulation results of the dynamical register machine behavior.

2 Prerequisites

Register Machines on Binary Numbers

A register machine on binary numbers is a tuple $M = (R, L, P, #_0)$ consisting of the finite set of registers $R = \{R_1, \ldots, R_m\}$ each with binary representation of a natural number $R_h \in \{0, 1\}^*$, a finite set of jump labels (addresses) $L = \{#_0, \ldots, #_n\}$, a finite set $P$ of instructions, and the jump label of the initial instruction $#_0 \in L$. Available instructions are: $\#_i$: INC $R_h$ and jump to $#_j$, $\#_i$: DEC $R_h$ # $j$ (nonnegatively decrement register $R_h$ and jump to # $j$), $\#_i$: IFZ $R_h$ # $j$ # $p$ (if $R_h = 0$ then jump to # $j$ else jump
to \( \#_p \), and \#i HALT (terminate program and output register contents). We assume a pre-initialization of all registers at start with zero and unique labeling: \( \forall p, q \in P . (p = \#_i : v) \land (q = \#_j : w) \land ((i \neq j) \lor (v = w)) \).

**Chemistry of Cell Signaling**

Cell signaling manages information processing within living organisms supplied by metabolism. The chemical principle of operation is mainly based on controlled protein-protein interactions forming signal transduction pathways [6]. Currently, more than 6,000 signal inducers or transmitters have been listed in the public Transpath database. Within a pathway, receptor or messenger proteins become successively activated by changing their configuration over time. Here, proteins alternating act as catalysts and reactants. Typical reaction networks are composed of recurrent structural motifs sketched in Fig. 1.

![Fig. 1. Typical structural motifs in cell signaling (A) and metabolic (B) networks](image)

**Modeling Chemical Reactions using Mass-Action Kinetics**

The dynamical behavior of considered chemical reaction networks describes the species concentrations over time. According to biologically predefined motifs, a variety of models exists to formulate the reaction kinetics. Since most of them imply specific assumptions, we restrict ourselves to general mass-action kinetics [5]. Here, a continuous approach to express the dynamical behavior considers production and consumption rates \( v_p \) and \( v_c \) of each species \( S \) in order to change its concentration by \( \frac{d[S]}{dt} = v_p - v_c \). Within mass-action kinetics, these rates result from the reactant concentrations, their stoichiometric factors \( a_{i,h} \in \mathbb{N} \) (reactants), \( b_{i,h} \in \mathbb{N} \) (products) and kinetic constants \( k_h \in \mathbb{R}_+ \) assigned to each reaction quantifying its velocity. For a reaction system with a total number of \( n \) species and \( r \) reactions indexed by \( h \in \{1, \ldots, r\} \)

\[
a_{1,h}S_1 + a_{2,h}S_2 + \ldots + a_{n,h}S_n \xrightarrow{k_h} b_{1,h}S_1 + b_{2,h}S_2 + \ldots + b_{n,h}S_n
\]
the corresponding ordinary differential equations read [7]:

$$\frac{d[S_i]}{dt} = \sum_{h=1}^{r} \left( k_h \cdot (b_{i,h} - a_{i,h}) \cdot \prod_{l=1}^{n} [S_l]^{a_{l,h}} \right) \quad \text{with} \quad i = 1, \ldots, n$$

In order to obtain a concrete trajectory, all initial concentrations \([S_i](0) \in \mathbb{R}_+\), \(i = 1, \ldots, n\) are allowed to be set according to the needs of the reaction system.

**Chemical Encoding of Binary Values**

Each boolean variable \(x \in \{0, 1\}\) is represented by two correlated species \(X^T\) and \(X^F\) with complement concentrations \([X^T] \in \mathbb{R}_+\) and \([X^F] \in \mathbb{R}_+\) such that \([X^T] + [X^F] = c\) holds where \(c\) is a constant. The boolean value of the variable \(x\) is determined whenever one of the following conditions is fulfilled: The inequality \([X^T] \ll [X^F]\) indicates “false” \((x = 0)\) and \([X^F] \ll [X^T]\) indicates “true” \((x = 1)\). In case of none of these strong inequalities holds, the system would consider the variable \(x\) to be in a mixture of both states.

**3 A Chemical Clock by Extending an Oscillating Reaction Network**

A chemical counterpart of a clock is necessary in order to sequentialize the register machine instruction processing. Positive edges of clock signals can trigger micro-operations like register increment or jump to the next machine instruction. In our chemical machine model, an extended oscillating reaction network provides all clock signals. As preferred network template for permanent oscillation, we adopt the well-studied Belousov-Zhabotinsky reaction [2, 16] formalized as Brusselator. It is depicted in the upper-left part of Fig. 2 whose dynamical behavior results in periodic peak-shaped signals. By using a cascade of downstream switching and maintaining reactions, we extend the primary oscillator. In this way, a normalization with respect to signal shape and concentration course can be reached. Our idea employs both converse output signals \(O^T_i\) and \(O^F_i\) of the previous cascade stage as triggers for a subsequent chemical toggle switch. Thus, high and low concentration levels are more and more precisely separated over time, and the switching delay in between becomes shortened, see lower-left parts of Fig. 2. After three cascade stages, the quality of the chemical clock signal turns out to be suitable for our purposes.

For technical reasons (two-phase register machine instruction processing, see Section 5), two offset clocks with designated output species \(C_1\) and \(C_2\) are employed. Owning the same network structure, they only differ in the time point when coming into operation caused by individual initializations (species producing clock signals \(C_1\): \([O^F_{0,C_1}](0) = 2, [O^T_{0,C_1}](0) = 1\); corresponding species for clock signals \(C_2\): \([O^F_{0,C_2}](0) = 0, [O^T_{0,C_2}](0) = 0\); species with identical initial concentrations: \([P_i,C_3](0) = 3, [P_2,C_3](0) = 1, [W_C](0) = 0, [O^F_{i,C_3}](0) = 1, [O^T_{i,C_3}](0) = 0, i \in \{1, 2, 3\}\), \(C \in \{C_1, C_2\}\) and kinetic constants \(k_s = k_{mo} = 3\). \(C_1\) and \(C_2\)
provide nonoverlapping clock signals whose offset constitutes approximately a half clock cycle, see Fig. 2 right.

4 Constructing Master-Slave Flip-Flops and Binary Registers

Based on chemical logic gates together with clock signals, chemical registers for binary numbers can be constructed. For this purpose, we first introduce a reaction network that mimics a master-slave flip-flop (MSFF). Afterwards, a chain of MSFFs forms a register $R_h$ ($h \in \{1, \ldots, |R|\}$) with arbitrary but finite length of $l_h$ bits. It processes binary numbers $b_{h_0} b_{h_{-1}} \ldots b_2 b_1$ with $b_\alpha \in \{0, 1\}$.

Furthermore, each register is equipped with predefined triggers in order to carry out micro-operations “increment”, “nonnegative decrement”, and “comparison to zero”, each processed within one clock cycle. Each network motif representing a MSFF is assumed to act as a bit storage unit with self-reproducible components in order to obtain Turing completeness. If new storing capacity within a register is needed, a corresponding subnetwork $b_{h_{+1}}$ is added.

Within a MSFF, bit setting is coupled to specific edges of the clock signal in order to prevent premature switches. In our MSFF implementation, bit setting is organized in two phases (master and slave part). Within the master part, a bit can be preset using specific master species $M^T$ and $M^F$ co-triggered by positive edges of the clock signal $C_1$, while the subsequent slave part finalizes the setting by forwarding the preset bit from the master species to the correlated slave species $S^T$ and $S^F$ triggered by positive edges of the offset clock signal $C_2$.
Fig. 3. Chemical reaction network of a register capable of processing a binary number \(b_{l_h} b_{l_{h-1}} \ldots b_2 b_1\) with \(b_{\alpha} \in \{0, 1\}\) including interfaces for micro-operations increment, nonnegative decrement, and comparison to zero.

A subnetwork consisting of eight switching reactions (see graymost highlighted boxes within each MSFF in Fig. 3) covers this task.

With regard to the functionality of a register machine, a sequence of interconnected MSFFs represents a register. The interconnections between neighboured MSFFs reflects the capability of incrementing and decrementing register contents. In case of incrementation, designated trigger molecules \(INC^I_h\) effect a successive bit flipping: Starting from the least significant bit \(b_1\), “1” is consecutively converted into “0” until “0” appears first time which is finally converted into “1”. Intermediate carry species \(E^F_{\alpha}\) act as forwarding triggers between consecutive bits, see Fig. 3. Decrementation is organized in a similar way using initial triggers \(DEC^I_h\) and intermediate molecules of carry species \(F^D_{\beta}\). In order to achieve nonnegative processing, a species \(E^F_{h}\) indicating equality to zero, set by a satellite network, prevents decrementation of binary strings 0\ldots0. Fig. 3 shows the reaction network structure of a register whose species \(F^I_{\alpha}, F^D_{\beta}, M^F_{\gamma}, M^T_{\kappa}, S^F_{\gamma},\) and \(S^T_{\gamma}\) are specific with respect to both register identifier \(h\) and bit position \(l_h\) within the register. Any comparison to zero is done by a satellite network which uses presence of any species \(M^T_{\kappa}\) with \(\kappa = 1, \ldots, l_h\) as triggers
in order to flip an equality indicator bit $e$ (species $E^F_h$ and $E^P_h$) onto “0”, while all species $M^F_\kappa$ with $\kappa = 1, \ldots, h$ are needed for flipping onto “1”, respectively. The indicator $e$ can be used for program control, see next section. As a further byproduct of each micro-operation on a register, molecules of the form $\#_j \in L$ encoding the jump label of the subsequent machine instruction are released.

## 5 Implementing a Chemical Program Control

A sequence of reactions directly derived from the given program $P$ of the underlying register machine $M = (R, L, P, \#_0)$ carries out the program control as follows: For each jump label $\#_j \in L$ we introduce a dedicated label species $\#_j$ with initial concentrations $[\#_0](0) = 1$ and $[\#_\kappa](0) = 0$ for $\kappa \in \{1, \ldots, |L| - 1\}$. Accordingly, a set of instruction species $I_{\nu} \in \{INC^j_h, DEC^j_h \mid \forall h \in \{1, \ldots, |R|\} \land \forall j \in \{0, \ldots, |L| - 1\}\} \cup \{IFZ^{j,q}_h \mid \forall h \in \{1, \ldots, |R|\} \land \forall j, q \in \{0, \ldots, |L| - 1\}\} \cup \{HALT\}$ is created with initial concentration $[I_{\nu}](0) = 0$. Furthermore, for each instruction in $P$ a network motif consisting of a program-control reaction with kinetic constant $k_p < k_s$ and a consecutive bypass reaction with $k_b \leq k_s$ is defined. Following the two-phase structure of a register machine instruction, these reactions first consume its incipient label species, then produce the corresponding instruction species as an intermediate product and finally convert it into the label species of the subsequent instruction if available. In order to strictly sequentialize the execution of instructions according to the program $P$, clock species $C_1$ and $C_2$ with offset concentration course provided by both oscillators trigger program-control and bypass reactions alternating as catalysts. The set of reactions for each type of register machine instructions is defined in Table 1.

Instruction species of the form $INC^j_h$ act as triggers for incrementing the contents of register $R_h$ done within its reaction network part, see Fig. 3. Here, $INC^j_h$ is converted into the byproduct $\#_j$ that provides the label species of the subsequent instruction. Accordingly, species $DEC^j_h$ initiate a concerted set of reactions decrementing register $R_h$ nonnegatively. Instruction species of the form $IFZ^{j,q}_h$ utilize a reaction network module attached to register $R_h$ that releases

**Table 1.** Encoding scheme of register machine’s instruction set into reactions

<table>
<thead>
<tr>
<th>instruction</th>
<th>reactions</th>
</tr>
</thead>
</table>
| $\#_i : INC R_h \ #_j$ | $\#_i + C_2 \xrightarrow{k_p} INC^j_h + C_2$  
$INC^j_h + C_1 \xrightarrow{k_s} \#_j + C_1$ |
| $\#_i : DEC R_h \ #_j$ | $\#_i + C_2 \xrightarrow{k_p} DEC^j_h + C_2$  
$DEC^j_h + C_1 \xrightarrow{k_s} \#_j + C_1$ |
| $\#_i : IFZ R_h \ #_j \ #_q$ | $\#_i + C_2 \xrightarrow{k_p} IFZ^{j,q}_h + C_2$  
$IFZ^{j,q}_h + E^F_h + C_1 \xrightarrow{k_s} \#_j + E^P_h + C_1$  
$IFZ^{j,q}_h + E^F_h + C_1 \xrightarrow{k_s} \#_q + E^P_h + C_1$ |
| $\#_i : HALT$ | $\#_i + C_2 \xrightarrow{k_p} HALT + C_2$ |
two species $E^T_h$ and $E^F_h$ whose concentrations indicate whether or not $R_h = 0$. Instruction species of the form $INC^j_h$, $DEC^j_h$, and $IFZ^j_h$ react into the corresponding label species $#^j_1$ and $#^j_3$. Since there is no reaction with instruction species $HALT$ as reactant, the program stops in this case. Fig. 4 illustrates an example of a chemical program control which also gives an overview about the interplay of all predefined modules.

6 Case Study: Integer Addition

The reaction network processing $R_2 := R_2 + R_1; R_1 := 0$ including previous register initialization $(R_1, R_2) := (2, 1)$ on two bit registers ($l_1 = 2$, $l_2 = 2$) emulates a case study of the integer addition “2 + 1” whose dynamical behavior using $k_s = 3$, $k_m = 1$, $k_{mo} = 3$, $k_b = 0.5$, $k_p = 1$ is shown in Fig. 5.

Fig. 5. Dynamical behavior of a chemical register machine acting as an adder

All simulations of the dynamical register machine behavior were carried out using CellDesigner version 3.5.2, an open source software for free academic use (www.celldesigner.org). The register machine implementation in SBML (Systems Biology Markup Language) is available from the authors upon request.
7 Discussion

Molecular computing approaches mostly benefit from a massive data parallel principle of operation. One might argue that nondeterministic models for computation like Chomsky grammars could be the best choice to formalize the computational capacity of molecular interactions. Indeed, those descriptions are known. Since contextual information is required in order to generate recursive enumerable languages, molecular substructures or spatial data, both preferably represented as strings, provide this information in these cases. Here, we restrict ourselves to symbol objects encoding molecular concentrations without any substructural or spatial specification. Although the emulation of a deterministic machine is quite slow in comparison to idealized nondeterministic term rewriting, technical details of performing a sequence of computational steps can be studied precisely according to mass-action kinetics. So, concentration gradients reach maxima or minima over the time course asymptotically. For that reason, there are correlations between kinetic parameters of the whole reaction network in order to guarantee the finalization of a micro-operation within a clock cycle.

The execution time of chemical register machine programs could be reduced by parallelization using a shared memory model whose principle of operation follows the CREW strategy (concurrent read exclusive write). To this end, the sequential program $P$ is decomposed into a number of nonoverlapping threads $T_i \subseteq P$ with $\bigcup T_i = P$ whereas each thread subsumes independent instructions allowed to run concurrently within one clock cycle. On the one hand, serially direct consecutive writing instructions (INC, DEC) on pairwise different registers might form a common thread for writing. On the other hand, any set of serially direct consecutive reading instructions (IFZ) represents a thread candidate for reading. Eventually, each halting instruction (HALT) becomes a separate thread. The chemical program control then acts thread-based: While reactions performing all instructions within one thread run simultaneously, a series of threads is controlled by handing over dedicated return label species which initiate the subsequent thread. This way, no further explicit synchronization mechanisms are required.

8 Conclusions

This paper presents a universal design principle for pure chemical register machines operating on binary numbers up to an arbitrary length. In contrast to discrete term rewriting techniques to evolve chemical reactions, we consistently employ continuous molecular concentrations due to mass-action reaction kinetics. No further assumptions like inhibiting reactions, spatial or substructural molecular data are needed. The proposed reaction scheme representing a register machine can run automatically without external intervention. For construction of the reaction scheme, we adopt ideas from circuit engineering, particularly definition of a set of recurring modules interconnected via communication interfaces. Starting from basic motifs found in cell signaling for storing and flipping boolean values together with a clock derived from an oscillating network.
based on the Belousov-Zhabotinsky reaction, a hierarchical composition of register machine elements is employed: modules of elementary motifs are arranged to form master-slave flip-flops by using clock signals. A cascade of those flip-flops acts as a register for a binary number. Boolean circuits perform incrementation and decrementation of a register as well as its comparison to zero. So, instructions can be triggered via designated sequences of reactions corresponding to the register machine program. Although the sequentialization of data processing causes a slow-down of computations in comparison to nondeterministic models, the effort for synchronization within the reaction network can be kept low and micro-operations within computational steps can be studied in detail more easily. Further work will deal with parallelization of instructions independently operating on distinct registers.

Acknowledgments

We gratefully acknowledge funding from the German Federal Ministry of Education and Research (project 0315260A) within the Research Initiative in Systems Biology.

References

Strong Normalization of Program-Indexed Lambda Calculus

Norihiro Kamide
Waseda Institute for Advanced Study,
1-6-1 Nishi Waseda, Shinjuku-ku, Tokyo 169-8050, JAPAN.
logician-kamide@aoni.waseda.jp

Abstract. Dynamic logics (DLs) are known as useful modal logics for program analysis, and typed λ-calculi for DLs have been required to obtain good computational interpretations. In this paper, program-indexed typed λ-calculus $\lambda_{DL}$, which has a Curry-Howard correspondence with an intuitionistic variant of DLs, is introduced, and the strong normalization theorem for $\lambda_{DL}$ is proved.

Keywords: Dynamic logic, natural deduction, strong normalization

1 Introduction

Dynamic logics (DLs) are known as useful logics for program analysis, and have widely been studied by many researchers [5]. Natural deduction (ND) systems for DLs have been required to obtain good computational interpretations and practical applications. Some sound and complete ND-systems for DLs over the first-order language of Peano arithmetic were introduced by Honsell and Miculan [6]. As mentioned in [6], this approach was inspired by an unpublished paper by Stirling, where a ND-system for deterministic DL was sketched. The approach by Honsell and Miculan was motivated to give adequate encodings of DL and Hoare logic in an interactive proof development environment, Coq. In [6], some completeness and encoding results were obtained for two ND-systems $S_{ND}(DL)$ and $S^{a}_{ND}(DL)$. However, (strong) normalization and Curry-Howard correspondence for such a system have not yet been studied. Indeed, obtaining a Curry-Howard correspondence with DLs was remained as an open question [6]. The present paper provides a partial solution to this open question, investigating a modified fragment of $S_{ND}(DL)$ and $S^{a}_{ND}(DL)$.

The aim of this paper is then to obtain a strongly normalizable typed $\lambda$-calculus (and ND-system) for an intuitionistic DL with a Curry-Howard correspondence. To obtain such a framework, the logical and program inference rules in $S_{ND}(DL)$ and $S^{a}_{ND}(DL)$ are generalized and restricted. Firstly, the logical and program inference rules are indexed by a (possibly empty) sequence $[d_0][d_1] \cdots [d_n]$ of program modalities. This indexing gives us a natural generalization of the standard simply typed $\lambda$-calculus, and allows us to obtain a
simple strong normalization proof. Secondly, the infinitary inference rule for the program iteration operator $\ast$:

$$\frac{\{ b; b; \cdots; b \} \alpha \mid j \in \omega}{[b^j]\alpha}$$

is restricted by replacing the set $\omega$ of natural numbers by an initial segment $\omega_l$ of a fixed positive integer $l$. The idea of restricting $\omega$ is also inspired from the work by Honsell and Miculan, where such a restriction was motivated for investigating the expressive power of a finitely proof system. Finally, two new constructors $\iota$ and $\iota^{-1}$ concerning the program composition operator $;$ are developed in order to obtain the corresponding typed $\lambda$-calculus.

The contents of this paper are then summarized as follows. In Section 2, the ND-system $N_{DL}$, which is called a program-indexed ND-system, is introduced, and some provable formulas in $N_{DL}$ are addressed. In Section 3, the typed $\lambda$-calculus $\lambda_{DL}$, which is called a program-indexed typed $\lambda$-calculus, is introduced. This calculus just corresponds to $N_{DL}$ via the Curry-Howard correspondence. In Section 4, the strong normalization theorem for $\lambda_{DL}$ (also for $N_{DL}$) is proved using the standard method presented in [4].

Finally in this section, we discuss other related approaches to ND-systems for DLs and their neighbors. As pointed out in [6], there are few approaches to ND-systems for DLs, despite this direction is important for obtaining useful applications and intuitive computational interpretations. A sound and complete Rasiowa-Sikorski-style proof system, which provides a ND-style reasoning method, was introduced by Demri and Orłowska [3] for an extension of propositional DL with demonic non-deterministic program operators. A dynamic $\lambda$-calculus was introduced by Kohlhase and Kuschet [7] in order to obtain a logical foundation of discourse theories. This calculus is, however, different from the standard approach to DLs in the sense that it does not use the program operators. Although as mentioned above, there are few results for ND-systems for DLs, there are some results for ND-systems for linear-time temporal logics (LTLs). For example, a judgment ND-system for a fragment of LTLs was introduced by Davies and Pfenning [2] in order to analyze staged computation in programming applications. A strongly normalizable ND-system for the until-free fragment of LTL was studied by Baratella and Masini [1] from the point of view of pure mathematical logic. Roughly speaking, the $\{ X \text{ (next-time)}, \rightarrow \}$-fragments of both Davies-Pfenning’s and Baratella-Masini’s systems are essentially equivalent, and the fragment may be equivalent to the $\{ X, \rightarrow \}$-fragment of Prior’s next-time temporal logic [8]. If the system $N_{DL}$ proposed in this paper is modified by deleting the inference rules for the program operators and replacing the program modalities by the nested next-time modalities, then such a resulting system is nearly equivalent to a ND-system for the $\{ X, \rightarrow \}$-fragment of Prior’s logic.
2 Program-indexed natural deduction system

Formulas are constructed from (countable) propositional variables, \( \rightarrow \) (implication), \( \land \) (conjunction), and \([b]\) (program modal operator) where \( b \) is a program. Programs are constructed from (countable) atomic programs, \( \lor \) (non-deterministic choice), \( ; \) (composition) and \( * \) (finite iteration). Lower-case letters \( b, c, ... \) are used for programs and Greek lower-case letters \( \alpha, \beta, ... \) are used for formulas. The symbol \( \omega \) is used to represent the set of natural numbers. The symbol \( \omega_l \) where \( l \) is a fixed positive integer is used to represent the set \( \{i \in \omega \mid i \leq l\} \). Lower-case letters \( i, j, ... \) are sometimes used for any natural numbers. For a program \( b \), an expression \( b^i \) with \( i \in \omega \) is defined inductively by \( b^0 \equiv \emptyset \) and \( b^{i+1} \equiv b^i ; b \). An expression \( \emptyset \alpha \) means \( \alpha \), and expressions \( \emptyset ; b \alpha \) and \( b ; \emptyset \alpha \) mean \( [b]\alpha \).

Definition 1 Formulas and programs are defined by the following grammar, assuming \( p \) and \( e \) represent propositional variables and atomic programs, respectively:

\[
\begin{align*}
\alpha &::= p \mid \alpha \rightarrow \alpha \mid \alpha \land \alpha \mid [b]\alpha \\
\beta &::= e \mid b \cup b \mid b ; b \mid b^*
\end{align*}
\]

The symbol PR is used to represent the set of all programs including the empty program \( \emptyset \). An expression \([d]\alpha\) is used to represent \([d_0];[d_1] ; \cdots ; [d_i]\) with \( i \in \omega \) and \( b_0 \equiv \emptyset \), i.e., \([d]\) can be the empty sequence. Also, an expression \( \hat{d} \) is used to represent \( d_0 ; d_1 ; \cdots ; d_i \) with \( i \in \omega \) and \( b_0 \equiv \emptyset \).

Definition 2 (\( N_{DL} \)) Let \( l \) be a fixed positive integer.

The inference rules of \( N_{DL} \) are of the form: for any \( k \in \omega_l \),

\[
\begin{align*}
&\frac{}{[d]\beta} \quad (-I) \\
&\frac{[\hat{d}]\alpha \rightarrow \beta}{[d](\alpha \rightarrow \beta)} \quad (\rightarrow I) \\
&\frac{[\hat{d}]\alpha}{[d](\alpha \rightarrow \beta)} \quad (\rightarrow E) \\
&\frac{[\hat{d}]\alpha_1 \land [\hat{d}]\alpha_2}{[d](\alpha_1 \land \alpha_2)} \quad (\land I) \\
&\frac{[\hat{d}]\alpha_1 \land [\hat{d}]\alpha_2}{[d]\alpha_1 \land [d]\alpha_2} \quad (\land E_1) \\
&\frac{[\hat{d}]\alpha_1 \land [\hat{d}]\alpha_2}{[d]\alpha_2} \quad (\land E_2) \\
&\frac{[\hat{d}][b]\alpha \land [d][c]\alpha}{[d][b \cup c]\alpha} \quad (\cup I) \\
&\frac{[\hat{d}][b \cup c]\alpha}{[d][b]\alpha \land [d][c]\alpha} \quad (\cup E_1) \\
&\frac{[\hat{d}][b \cup c]\alpha}{[d]\alpha} \quad (\cup E_2) \\
&\frac{[\hat{d}][b ; c]\alpha}{[d][b ; c]\alpha} \quad (; I) \\
&\frac{[\hat{d}][b ; c]\alpha}{[d][b ; c]\alpha} \quad ;) \\
&\frac{[\hat{d}][b ; c]\alpha}{[d][b ; c]\alpha} \quad (E) \\
&\frac{[\hat{d}][b ; c]\alpha}{[d][b ; c]\alpha} \quad (\forall I) \\
&\frac{[\hat{d}][b ; c]\alpha}{[d][b ; c]\alpha} \quad (\forall E) \\
&\frac{[\hat{d}][b ; c]\alpha}{[d][b ; c]\alpha} \quad (\exists I) \\
&\frac{[\hat{d}][b ; c]\alpha}{[d][b ; c]\alpha} \quad (\exists E).
\end{align*}
\]

Remark that \((\exists I)\) has \((l + 1)\)-premises, and the \( k \) appeared in \((\exists E)\) is in \( \omega_l \). This means that the formula of the form \([b^k]\alpha\) is interpreted as \( \alpha \land [b]\alpha \land [b]^2\alpha \land \cdots \land [b^l]\alpha \). Thus, a formula \( p \land [b]\alpha \land [b]^2\alpha \land \cdots \land [b]^{l+1}\alpha \rightarrow [b^k]\alpha \) may not be provable in \( N_{DL} \). If the unbounded versions of \((\exists I)\) and \((\forall E)\) by replacing \( \omega_l \) by
\( \omega \) are assumed, then the underlying system \( N_{DL}^{\omega} \) is more natural than \( N_{DL} \). But, the strong normalization result for \( N_{DL}^{\omega} \) has not yet been obtained, and hence our result for \( N_{DL} \) is regarded as a partial result.

Strictly speaking, \( N_{DL} \) is just the logic parameterized by a fixed concrete positive integer \( l \), and hence, such an \( l \)-parameterized logic should precisely be denoted as e.g., \( N_{DL[l]} \). But, since we don’t need to specify such an integer \( l \) in the following discussion, we will use the abstract name “\( N_{DL} \)” instead of the concrete “\( N_{DL[l]} \)”. We also use the same positive integer \( l \) through the following discussion.

The terminologies of the standard natural deduction system are used. The notions of proof \( \) (in \( N_{DL} \), discussion.  be defined and the strong normalization theorem for \( N_{DL} \) is not an axioms scheme in the standard dynamic logic. This axioms scheme is appeared in Prior’s next-time temporal logic \[8\] where \( \textit{next-time operator} \ X \).

Terms \( 3 \) Program-indexed typed \( \lambda \)-calculus

Terms are constructed from variables, a \( \lambda \)-abstraction \( \lambda \), an application operator \( \cdot \) (it is always omitted), a pairing function \( \langle, \rangle \), an \( (l + 1) \)-ary pairing function \( \langle, ..., \rangle \), projection functions \( \pi_1, \pi_2, ..., \pi_{l+1} \) and two new constructors \( \iota \) and \( \iota^{-1} \). The intended meaning of \( \iota \) and \( \iota^{-1} \) is presented as the equations:

\[
\iota^{-1}(\iota M[b][c][\alpha]) = M[b][c][\alpha] \quad \text{and} \quad (\iota(\iota^{-1} M[b][c][\alpha])) = M[b][c][\alpha].
\]
(Untyped) terms are defined as usual, and types are defined as the same way as in Definition 1, replacing “formulas” by “types”. Variables are denoted as \( x, x_\alpha, y, \ldots \), untyped terms are denoted as \( M, M_n, N, \ldots \), types are denoted as \( \alpha, \beta, \gamma, \ldots \), and typed terms are denoted as \( M^\alpha, N^\beta, L^\gamma, \ldots \). Typed terms are sometimes denoted as \( M, N, L, \ldots \) by omitting the types. It is assumed that in a \( \lambda \)-term, the same variables do not occur simultaneously as both free and bound variables. It is also assumed that in a \( \lambda \)-term, there are no iterated occurrences of the same bound variable \( x \), such as \( \cdots \lambda x^\alpha. (\cdots \lambda x^\alpha. (\cdots) \cdots \cdots \cdots \) An expression \( [N^\alpha/x^\alpha]M^\beta \) means, in a usual sense, the substitution of \( N^\alpha \) to a free variable \( x^\alpha \) in \( M^\beta \). For the new constructor \( \iota' \in \{1, \iota^{-1}\} \), we also assume the condition \( [N^\alpha/x^\alpha](\iota'M^\beta)\gamma = (\iota'[N^\alpha/x^\alpha]M^\beta)\gamma \). To avoid the clash of bound variables by substitutions, \( \alpha \)-conversions are occasionally assumed.

**Definition 4** The notion of degree \( d(b) \) of a program \( b \) is defined by:

1. \( d(e) = 1 \) for any atomic program \( e \),
2. \( d(b \cup c) = \max(d(b), d(c)) + 1 \),
3. \( d(b^*) = (l \times d(b)) + l \).

The notion degree \( d(\alpha) \) of a type \( \alpha \) is defined by:

1. \( d(p) = 1 \) for any atomic type \( p \),
2. \( d(\alpha \circ \beta) = d(\alpha) + d(\beta) + 1 \) where \( \circ \in \{\to, \land\} \),
3. \( d([b]\alpha) = d(b) + d(\alpha) \).

For example, \( d([c]p) = 4 \), \( d([e]c)p = 3 \), \( d(e^*) = 2 \times l \) and \( d(e^l) = (2 \times l) - 1 \) where \( e \) and \( p \) are an atomic program and an atomic type, respectively. Remark that \( d([b]; c]\alpha) > d([b][c]\alpha) \) and \( d([b^*]\alpha) > d([b^l]\alpha) \) for any programs \( b, c \) and any type \( \alpha \).

**Definition 5** Typed \( \lambda \)-terms for \( \lambda_{DL} \) are defined as follows.

1. if \( x[^\alpha]\alpha \) is a typed variable, then it is a typed \( \lambda \)-term.
2. if \( x[^\alpha]\alpha \) and \( M[^\beta]\beta \) are typed \( \lambda \)-terms, then \( (\lambda x[^\alpha]\alpha.M[^\beta]\beta)^[^\alpha]\alpha(x\to\beta) \) is a typed \( \lambda \)-term.
3. if \( M[^\alpha]\alpha(x\to\beta) \) and \( N[^\alpha]\alpha \) are typed \( \lambda \)-terms, then \( (M[^\alpha]\alpha(x\to\beta)\to)^[^\alpha]\alpha N[^\alpha]\alpha \) is a typed \( \lambda \)-term.
4. if \( M[^\alpha]\alpha \) and \( N[^\alpha]\beta \) are typed \( \lambda \)-terms, then \( (M[^\alpha]\alpha\land N[^\alpha]\beta)^[^\alpha]\alpha \) is a typed \( \lambda \)-term.
5. if \( M[^\alpha]\alpha(x\land\beta) \) is a typed \( \lambda \)-term, then \( (\pi_1 M[^\alpha]\alpha(x\land\beta))^[^\alpha]\alpha \) and \( (\pi_2 M[^\alpha]\alpha(x\land\beta))^[^\alpha]\beta \) are typed \( \lambda \)-terms.
6. if \( M[^\alpha]\alpha(x\land\beta) \) and \( N[^\alpha][c]\alpha \) are typed \( \lambda \)-terms, then \( (M[^\alpha]\alpha(x\land\beta)\land N[^\alpha][c]\alpha)^[^\alpha]\alpha \) is a typed \( \lambda \)-term.
7. if \( M[^\alpha]\alpha(x\land\beta) \) is a typed \( \lambda \)-term, then \( (\pi_1 M[^\alpha]\alpha(x\land\beta))^[^\alpha]\alpha \) and \( (\pi_2 M[^\alpha]\alpha(x\land\beta))^[^\alpha]\beta \) are typed \( \lambda \)-terms.
8. if \( M[^\alpha][c]\alpha \) is a typed \( \lambda \)-term, then \( (\iota M[^\alpha][c]\alpha)^[^\alpha]\alpha \) is a typed \( \lambda \)-term.
9. if \( M[^\alpha][c]\alpha \) is a typed \( \lambda \)-term, then \( (\iota^{-1} M[^\alpha][c]\alpha)^[^\alpha]\alpha \) is a typed \( \lambda \)-term.
10. if $M_1^{[\tilde{d}]\alpha}, M_1^{[\tilde{d}]b\alpha}, \ldots, M_i^{[\tilde{d}]b\alpha}$ are typed $\lambda$-terms, then $(M_0^{[\tilde{d}]\alpha}, M_1^{[\tilde{d}]b\alpha}, \ldots, M_i^{[\tilde{d}]b\alpha})$ is a typed $\lambda$-term.

11. if $M^{[\tilde{d}]b\alpha}$ is a typed $\lambda$-term, then $(\pi_1 M^{[\tilde{d}]b\alpha})^{[\tilde{d}]\alpha}, (\pi_2 M^{[\tilde{d}]b\alpha})^{[\tilde{d}]b\alpha}, \ldots, (\pi_{i+1} M^{[\tilde{d}]b\alpha})^{[\tilde{d}]b\alpha}$ are typed $\lambda$-terms.

**Definition 6 ($\lambda_{DL}$)** The typed $\lambda$-calculus $\lambda_{DL}$ is defined by reductions for the typed $\lambda$-terms defined in Definition 5. In the following, the transformation process from the left hand side of $\succ$ to the right hand side of $\succ$ is called a reduction, and the term of the left hand side of $\succ$ is called a redex.

1. $(\lambda x^{[\tilde{d}]\alpha} M^{[\tilde{d}]\beta})^{[\tilde{d}]\alpha / x / \beta} \succ [N^{[\tilde{d}]\alpha / x / \beta}] M^{[\tilde{d}]\beta}$.
2. $(\pi_1 M^{[\tilde{d}]\alpha}, N^{[\tilde{d}]\beta})^{[\tilde{d}]\alpha \land \beta} \succ M^{[\tilde{d}]\alpha}$.
3. $(\pi_2 M^{[\tilde{d}]\alpha}, N^{[\tilde{d}]\beta})^{[\tilde{d}]\alpha \land \beta} \succ N^{[\tilde{d}]\beta}$.
4. $(\pi_1 M^{[\tilde{d}]b\alpha}, N^{[\tilde{d}]c\alpha})^{[\tilde{d}]b\alpha \land \omega c\alpha} \succ M^{[\tilde{d}]b\alpha}$.
5. $(\pi_2 M^{[\tilde{d}]b\alpha}, N^{[\tilde{d}]c\alpha})^{[\tilde{d}]b\alpha \land \omega c\alpha} \succ N^{[\tilde{d}]c\alpha}$.
6. $(\iota^{-1} M^{[\tilde{d}]b\alpha})^{[\tilde{d}]b\alpha} \succ M^{[\tilde{d}]b\alpha}$.
7. $(\pi_{k+1} M^{[\tilde{d}]\alpha}, M_1^{[\tilde{d}]\beta\alpha}, \ldots, M_i^{[\tilde{d}]b\alpha})^{[\tilde{d}]b\alpha / \iota} \succ M^{[\tilde{d}]b\alpha / \iota}$ with $1 \leq k \leq \omega_{l+1}.
8. (Compatible closure): if $M \succ N$, then $\lambda x.M \succ \lambda x.N, ML \succ NL, LM \succ LN, \langle M, L \rangle \succ \langle N, L \rangle, \langle L, M \rangle \succ \langle L, N \rangle, \langle \ldots, M, \ldots \rangle \succ \langle \ldots, N, \ldots \rangle, \pi_1 M \succ \pi_1 N, \pi_2 M \succ \pi_2 N, \pi_k M \succ \pi_k N$ with $2 \leq k \leq \omega_{l+1}, \iota M \succ \iota N$ and $\iota^{-1} M \succ \iota^{-1} N$.

**4 Strong normalization**

**Definition 7** A typed $\lambda$-term is said to be normal if it contains no redex. A sequence $M_0^{[\alpha]}, M_1^{[\alpha]}, \ldots$ of typed $\lambda$-terms is called a reduction sequence if it satisfies the following conditions (1) $M_i^{[\alpha]} \succ M_{i+1}^{[\alpha]}$ for all $0 \leq i$ and (2) the last typed $\lambda$-term in the sequence is normal if the sequence is finite. A typed $\lambda$-term $M^{[\alpha]}$ is called strongly normalizable if each reduction sequence starting from $M^{[\alpha]}$ is terminated.

We now start to prove the strong normalization theorem for $\lambda_{DL}$, using the method presented in [4]. In the following, SN means the set of all strongly normalizable typed $\lambda$-terms for $\lambda_{DL}$, and TERM means the set of all typed $\lambda$-terms for $\lambda_{DL}$. In order to show $\text{TERM} \subseteq \text{SN}$ (i.e., the strong normalization theorem), we will define the set RED of reducible terms, and will show $\text{TERM} \subseteq \text{RED} \subseteq \text{SN}$. First, we will show $\text{RED} \subseteq \text{SN}$ by induction on the degree of a type, and second, will show $\text{TERM} \subseteq \text{RED}$ by induction on the construction of a term.

**Definition 8** The set RED of reducible terms of type $\gamma$ (for $\lambda_{DL}$) is defined by induction on the (degree of) type $\gamma$ as follows.

1. $M^{[\tilde{d}]b} \in \text{RED}_{[\tilde{d}]b}$ if $M^{[\tilde{d}]b} \in \text{SN}$, for any atomic type $p$. 

201
2. $M[(\alpha \rightarrow \beta)] \in \text{RED}_{[d](\alpha \rightarrow \beta)}$ iff $\forall N[\tilde{a}] \in \text{RED}_{[d]\alpha}$ $[(M[\tilde{a}] (\alpha \rightarrow \beta))_N[\tilde{a}]\alpha \tilde{a}] \beta \in \text{RED}_{[d]\beta}$.

3. $M[\tilde{a}] (\alpha \wedge \beta) \in \text{RED}_{[d](\alpha \wedge \beta)}$ iff $(\pi_1 M[\tilde{a}] (\alpha \wedge \beta)) [\tilde{a}] \alpha \in \text{RED}_{[d]\alpha}$ and $(\pi_2 M[\tilde{a}] (\alpha \wedge \beta)) [\tilde{a}] \beta \in \text{RED}_{[d]\beta}$.

4. $M[\tilde{a}] [b \cup c] \alpha \in \text{RED}_{[d][b \cup c] (\alpha \wedge \beta)}$ iff $(\pi_1 M[\tilde{a}] [b \cup c] (\alpha \wedge \beta)) [\tilde{a}] \alpha \in \text{RED}_{[d][b]\alpha}$ and $(\pi_2 M[\tilde{a}] [b \cup c] (\alpha \wedge \beta)) [\tilde{a}] \beta \in \text{RED}_{[d][c]\alpha}$.

5. $M[\tilde{a}] [b] ; c \alpha \in \text{RED}_{[d][b] ; c \alpha}$ iff $(\pi_1 M[\tilde{a}] [b] ; c \alpha) [\tilde{a}] [b] \beta \in \text{RED}_{[d][b] \alpha}$.

6. $M[\tilde{a}] [b^*] \alpha \in \text{RED}_{[d][b^*] (\alpha \wedge \beta)}$ iff $(\pi k M[\tilde{a}] [b^*] (\alpha \wedge \beta)) [\tilde{a}] [b^{k-1}] \alpha \in \text{RED}_{[d][b^{k-1}]\alpha}$ for all $k$ with $1 \leq k \in \omega_{l+1}$.

**Definition 9** A typed $\lambda$-term $M^\alpha$ for $\lambda_{DL}$ is said to be neutral if $M$ is one of the forms $x$, $NP$, $\pi_1 N$, $\pi_2 N$, $\pi_k N$ with $2 < k < \omega_{l+1}$, and $\iota^{-1} N$.

If $M^\alpha \in SN$, then an expression $v(M^\alpha)$ means the least number which bounds the length of every reduction sequence beginning with $M^\alpha$.

**Lemma 10** For all typed $\lambda$-term $M^\alpha$ for $\lambda_{DL}$, $M^\alpha$ satisfies the following four conditions.

- **CR1.** if $M^\alpha \in \text{RED}_\alpha$, then $M^\alpha \in \text{SN}$.
- **CR2.** if $M^\alpha \in \text{RED}_\alpha$ and $M^\alpha \succ N^\alpha$, then $N^\alpha \in \text{RED}_\alpha$.
- **CR3.** if $M^\alpha$ is neutral, then $\forall N^\alpha$ [if $M^\alpha \succ N^\alpha$ and $N^\alpha \in \text{RED}_\alpha$, then $M^\alpha \in \text{RED}_\alpha$].
- **CR4.** if $M^\alpha$ is neutral and normal, then $M^\alpha \in \text{RED}_\alpha$. Remark that (CR4) is a special case of (CR3).

**Proof.** By induction on the degree $d(\alpha)$ of the type $\alpha$. The cases $\alpha \equiv [d] p$ (atomic), $\alpha \equiv [\tilde{a}] (\beta \rightarrow \gamma)$ and $\alpha \equiv [\tilde{a}] (\beta \wedge \gamma)$ can be proved in a similar way as in [4]. We show only the cases $\alpha \equiv [d][b] ; c\beta$ and $\alpha \equiv [d][b^*] \beta$.

- **Case (\alpha \equiv [d][b] ; c\beta).**

  **(CR1):** Suppose $M[\tilde{a}] [b] ; c\beta \in \text{RED}_{[d][b] ; c\beta}$. Then, $(\iota^{-1} M[\tilde{a}] [b] ; c\beta) [\tilde{a}] [b] \alpha \beta \in \text{RED}_{[d][b] ; c\beta}$ by the definition of RED. By the induction hypothesis of (CR1) with $d([\tilde{a}] [b] ; c\beta) > d([\tilde{a}] [b] \alpha \beta)$, we obtain $(\iota^{-1} M[\tilde{a}] [b] ; c\beta) [\tilde{a}] [b] \alpha \beta \in \text{SN}$. Moreover, we have $v((\iota^{-1} M[\tilde{a}] [b] ; c\beta) [\tilde{a}] [b] \alpha \beta) \geq v(M[\tilde{a}] [b] ; c\beta)$, because from any reduction sequence $M[\tilde{a}] [b] ; c\beta \succ M_1[\tilde{a}] [b] ; c\beta \succ M_2[\tilde{a}] [b] ; c\beta \succ \cdots$, one can construct a reduction sequence $(\iota^{-1} M[\tilde{a}] [b] ; c\beta) [\tilde{a}] [b] \alpha \beta \succ (\iota^{-1} M_1[\tilde{a}] [b] ; c\beta) [\tilde{a}] [b] \alpha \beta \succ \cdots$. So $v(M[\tilde{a}] [b] ; c\beta)$ is finite, and hence $M[\tilde{a}] [b] ; c\beta \in \text{SN}$.

  **(CR2):** Suppose $M[\tilde{a}] [b] ; c\beta \succ N[\tilde{a}] [b] ; c\beta$. Then, $(\iota^{-1} M[\tilde{a}] [b] ; c\beta) [\tilde{a}] [b] \alpha \beta \succ (\iota^{-1} N[\tilde{a}] [b] ; c\beta) [\tilde{a}] [b] \alpha \beta$. By the hypothesis, we have $M[\tilde{a}] [b] ; c\beta \in \text{RED}_{[d][b] ; c\beta}$, and hence $(\iota^{-1} M[\tilde{a}] [b] ; c\beta) [\tilde{a}] [b] \alpha \beta \in \text{RED}_{[d][b] ; c\beta}$ by the definition of RED. By
the induction hypothesis of (CR2) with \( d(\hat{d}[b ; c]_\beta) > d([\hat{d}][b][c]_\beta) \), we obtain 
\((\nu^{-1} N[\hat{d}][b ; c]_\beta) \hat{d}[b][c]_\beta \in \text{RED}_{\hat{d}[b][c]_\beta} \), and hence \( N[\hat{d}][b ; c]_\beta \in \text{RED}_{\hat{d}[b][c]_\beta} \).

(CR3): Let \( M[\hat{d}][b ; c]_\beta \) be neutral and suppose all the \( N[\hat{d}][b ; c]_\beta \) such that \( M[\hat{d}][b ; c]_\beta \succ N[\hat{d}][b ; c]_\beta \in \text{RED}_{\hat{d}[b][c]_\beta} \). Since \( M[\hat{d}][b ; c]_\beta \) is neutral, 
\((\nu^{-1} M[\hat{d}][b ; c]_\beta) \hat{d}[b][c]_\beta \succ (\nu^{-1} N[\hat{d}][b ; c]_\beta) \hat{d}[b][c]_\beta \) and \((\nu^{-1} N[\hat{d}][b ; c]_\beta) \hat{d}[b][c]_\beta \in \text{RED}_{\hat{d}[b][c]_\beta} \) because of the hypothesis \( N[\hat{d}][b ; c]_\beta \in \text{RED}_{\hat{d}[b][c]_\beta} \) and the definition of \( \text{RED} \). Since \((\nu^{-1} M[\hat{d}][b ; c]_\beta) \hat{d}[b][c]_\beta \) is neutral and all the typed \( \lambda \)-terms one step from \((\nu^{-1} M[\hat{d}][b ; c]_\beta) \hat{d}[b][c]_\beta \) are in \( \text{RED}_{\hat{d}[b][c]_\beta} \), we can apply the induction hypothesis of (CR3) with \( d(\hat{d}[b ; c]_\beta) < d([\hat{d}][b][c]_\beta) \), and obtain 
\((\nu^{-1} M[\hat{d}][b ; c]_\beta) \hat{d}[b][c]_\beta \in \text{RED}_{\hat{d}[b][c]_\beta} \). Therefore we obtain \( M[\hat{d}][b ; c]_\beta \in \text{RED}_{\hat{d}[b][c]_\beta} \) by the definition of \( \text{RED} \).

- Case \((\alpha \equiv [\hat{d}][b^*]_\beta) \).

(CR1): Suppose \( M[\hat{d}][b^*]_\beta \in \text{RED}_{\hat{d}[b^*]_\beta} \). Then, by the definition of \( \text{RED} \), 
\((\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \in \text{RED}_{\hat{d}[b^*]_\beta} \) for all \( k \leq 1 \leq k \in \omega_{l+1} \). We have \( k - 1 \leq l \) and \( d([\hat{d}][b^*]_\beta) < d([\hat{d}][b^*]_\beta) \). Hence we can apply the induction hypothesis of (CR1), and obtain \((\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \in \text{SN} \). Moreover, we have \( v((\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta) = v(M[\hat{d}][b^*]_\beta) \), because from any reduction sequence \( M[\hat{d}][b^*]_\beta \succ M_1[\hat{d}][b^*]_\beta \succ M_2[\hat{d}][b^*]_\beta \succ \cdots \), one can construct a reduction sequence \( (\pi_k M_1[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \succ (\pi_k M_2[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \succ \cdots \). So \( v(M[\hat{d}][b^*]_\beta) \) is finite, and hence \( M[\hat{d}][b^*]_\beta \in \text{SN} \).

(CR2): Suppose \( M[\hat{d}][b^*]_\beta \succ N[\hat{d}][b^*]_\beta \). Then, 
\((\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \succ (\pi_k N[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \) for all \( k \leq 1 \leq k \in \omega_{l+1} \). By the hypothesis, we have \( M[\hat{d}][b^*]_\beta \in \text{RED}_{\hat{d}[b^*]_\beta} \), and hence \( (\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \in \text{RED}_{\hat{d}[b^*]_\beta} \) by the definition of \( \text{RED} \). We have \( k - 1 \leq l \) and \( d([\hat{d}][b^*]_\beta) < d([\hat{d}][b^*]_\beta) \). Hence we can apply the induction hypothesis of (CR2), and then obtain \((\pi_k N[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \in \text{RED}_{\hat{d}[b^*]_\beta} \). Thus, \( N[\hat{d}][b^*]_\beta \in \text{RED}_{\hat{d}[b^*]_\beta} \) by the definition of \( \text{RED} \).

(CR3): Let \( M[\hat{d}][b^*]_\beta \) is neutral and suppose all the \( N[\hat{d}][b^*]_\beta \) such that 
\( M[\hat{d}][b^*]_\beta \succ N[\hat{d}][b^*]_\beta \) \( \in \text{RED}_{\hat{d}[b^*]_\beta} \). Since \( M[\hat{d}][b^*]_\beta \) is neutral, 
\((\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \in \text{RED}_{\hat{d}[b^*]_\beta} \) for all \( k \leq k \in \omega_{l+1} \) cannot itself be a reduct. Thus, we obtain 
\((\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \succ (\pi_k N[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \) \( \in \text{RED}_{\hat{d}[b^*]_\beta} \), because of the hypothesis \( N[\hat{d}][b^*]_\beta \in \text{RED}_{\hat{d}[b^*]_\beta} \) and the definition of \( \text{RED} \). We have that \( (\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \) is neutral and all the typed \( \lambda \)-terms one step from \((\pi_k M[\hat{d}][b^*]_\beta) \hat{d}[b^*]_\beta \) are in \( \text{RED}_{\hat{d}[b^*]_\beta} \), and that \( k - 1 \leq l \) and \( d([\hat{d}][b^*]_\beta) < d([\hat{d}][b^*]_\beta) \). Thus, we can
apply the induction hypothesis of (CR3), and obtain \( \pi_k M^{[\tilde{d}][b^*][\beta]} \in \text{RED}_{\tilde{d}}\). Therefore, we obtain \( M^{[\tilde{d}][b^*][\beta]} \in \text{RED}_{\tilde{d}} \) by the definition of RED.

By (CR1) of Lemma 10, we have RED \( \subseteq \text{SN} \). Using (CR1) – (CR4) in Lemma 10, we can prove the following lemma.

**Lemma 11** The following conditions hold for \( \lambda_{DL} \).

1. If \( x^{[\tilde{d}]\alpha} \) is a typed variable, then \( x^{[\tilde{d}]\alpha} \in \text{RED}_{\tilde{d}}[\tilde{d}]\alpha \).
2. For any \( M^{[\tilde{d}]\beta} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \) and any \( N^{[\tilde{d}]\alpha} \in \text{RED}_{\tilde{d}}[\tilde{d}]\alpha \), if \( [N^{[\tilde{d}]\alpha}/x^{[\tilde{d}]\alpha}]M^{[\tilde{d}]\beta} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \), then \( (\lambda x^{[\tilde{d}]\alpha} M^{[\tilde{d}]\beta})^{[\alpha\rightarrow\beta]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \).
3. If \( M^{[\tilde{d}]\alpha} \in \text{RED}_{\tilde{d}}[\tilde{d}]\alpha \) and \( N^{[\tilde{d}]\beta} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \), then \( (\lambda M^{[\tilde{d}]\alpha}, N^{[\tilde{d}]\beta})^{[\alpha\wedge\beta]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \).
4. If \( M^{[\tilde{d}]\beta} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \) and \( N^{[\tilde{d}]\alpha} \in \text{RED}_{\tilde{d}}[\tilde{d}]\alpha \), then \( \langle M^{[\tilde{d}]\beta}, N^{[\tilde{d}]\alpha} \rangle^{[\beta\wedge\alpha]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \).
5. If \( M^{[\tilde{d}]\alpha} \in \text{RED}_{\tilde{d}}[\tilde{d}]\alpha \) and \( N^{[\tilde{d}]\beta} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \), then \( \langle \lambda M^{[\tilde{d}]\beta} \rangle^{[\alpha\rightarrow\beta]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \).

6. If \( M^{[\tilde{d}]\alpha} \in \text{RED}_{\tilde{d}}[\tilde{d}]\alpha \) and \( N^{[\tilde{d}]\beta} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \), then \( \langle \lambda M^{[\tilde{d}]\beta} \rangle^{[\alpha\rightarrow\beta]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta \).

**Proof.** (1) is obvious by (CR4). (3)–(6) are similar. We show only (5).

- (5). Suppose \( M^{[\tilde{d}]\beta}[b][c]\alpha \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta[\tilde{d}]\beta \). We will show \( \langle \lambda M^{[\tilde{d}]\beta}[b][c]\alpha \rangle^{[\beta\\rightarrow\alpha]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta[\tilde{d}]\beta \), i.e., it is enough to show \( \langle \lambda M^{[\tilde{d}]\beta}[b][c]\alpha \rangle^{[\beta\\rightarrow\alpha]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta[\tilde{d}]\beta \). Because of (CR1) and the hypothesis, we have \( M^{[\tilde{d}]\beta}[b][c]\alpha \in \text{SN} \). Thus, we can consider \( v(M^{[\tilde{d}]\beta}[b][c]\alpha) \).

In the following, we show \( \langle \lambda M^{[\tilde{d}]\beta}[b][c]\alpha \rangle^{[\alpha\\rightarrow\beta]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta[\tilde{d}]\beta \) by induction on \( v(M^{[\tilde{d}]\beta}[b][c]\alpha) \). This typed \( \lambda \)-term converts (1) \( M^{[\tilde{d}]\beta}[b][c]\alpha \) or (2) \( \langle \lambda M^{[\tilde{d}]\beta}[b][c]\alpha \rangle^{[\alpha\\rightarrow\beta]} \) where \( M^{[\tilde{d}]\beta}[b][c]\alpha > N^{[\tilde{d}]\beta}[b][c]\alpha \). For the case (2), we obtain \( N^{[\tilde{d}]\beta}[b][c]\alpha \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta[\tilde{d}]\beta \) by (CR2), and we have \( v(M^{[\tilde{d}]\beta}[b][c]\alpha) > v(N^{[\tilde{d}]\beta}[b][c]\alpha) \). So we obtain \( \langle \lambda M^{[\tilde{d}]\beta}[b][c]\alpha \rangle^{[\alpha\\rightarrow\beta]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta[\tilde{d}]\beta \) by the induction hypothesis. In both cases, the neutral term \( \langle \lambda M^{[\tilde{d}]\beta}[b][c]\alpha \rangle^{[\alpha\\rightarrow\beta]} \) converts to reducible terms only, and by (CR3), it is reducible. Therefore \( \langle \lambda M^{[\tilde{d}]\beta}[b][c]\alpha \rangle^{[\alpha\\rightarrow\beta]} \in \text{RED}_{\tilde{d}}[\tilde{d}]\beta[\tilde{d}]\beta \).

An expression \( [N_1^{[\beta_1]}/x_1^{[\beta_1]}, ..., N_n^{[\beta_n]}/x_n^{[\beta_n]}] \) denotes the simultaneous substitution. Using Lemma 11, we can prove the following lemma.

**Lemma 12** Let \( M^\alpha \) be a typed \( \lambda \)-term for \( \lambda_{DL} \). If \( N_1^{[\beta_1]} \in \text{RED}_{\beta_1}, ..., N_n^{[\beta_n]} \in \text{RED}_{\beta_n} \), then \( [N_1^{[\beta_1]}/x_1^{[\beta_1]}, ..., N_n^{[\beta_n]}/x_n^{[\beta_n]}] M^\alpha \in \text{RED}_{\alpha} \).
Proof. By induction on the construction of $M$. Let $\sigma = [N_1^{\beta_1}/x_1^{\beta_1}, \ldots, N_n^{\beta_n}/x_n^{\beta_n}]$.

Case ($M^\alpha \equiv x_i^{\beta_i}$ ($1 \leq i \leq n$)): Obvious, i.e., $\sigma x_i^{\beta_i} \equiv N_i^{\beta_i} \in \text{RED}_\beta$.

Case ($M^\alpha \equiv x^\alpha$ and $x^\alpha \neq x_1^{\beta_1}, \ldots, x_n^{\beta_n}$): By Lemma 11 (1).

Case ($M^\alpha \equiv (\lambda x^\beta \cdot N^{\beta})(\beta \rightarrow \gamma)$): By using Lemma 11 (2).

Case ($M^\alpha \equiv (N^{\beta}, L^{\gamma})^\alpha$ where $(\cdot)$ is a pairing $(\cdot, \cdot)$ or an application): By the hypothesis of induction, we have $\sigma N^{\beta} \in \text{RED}_\beta$ and $\sigma L^{\gamma} \in \text{RED}_\gamma$. We thus obtain $\sigma M^\alpha \equiv (\sigma N^{\beta}, \sigma L^{\gamma})^\alpha \in \text{RED}_\alpha$ by Lemma 11 (3–4) or by the definition.

Case ($M^\alpha \equiv (\pi M^{\beta})^\alpha$ where $\pi$ is $\pi_1, \pi_2, \pi_k$ with $2 < k \in \omega_{l+1}$ or $\iota^{-1}$): By the hypothesis of induction, we have $\sigma M^{\beta} \in \text{RED}_\beta$. This fact derives $\sigma M^{\beta} \in \text{RED}_\alpha$ by the definition. Therefore we obtain $\sigma (\pi M^{\beta})^\alpha \in \text{RED}_\alpha$.

Theorem 13 (Strong normalization) All typed $\lambda$-terms for $\lambda_{DL}$ are strongly normalizable.

Proof. In Lemma 12, taking $N_1 \equiv x_1, \ldots, N_n \equiv x_n$, we have $M^\alpha \in \text{RED}_\alpha$ for any typed $\lambda$-term $M^\alpha$ for $\lambda_{DL}$, i.e., $\text{TERM} \subseteq \text{RED}$. Since we already have $\text{RED} \subseteq \text{SN}$, we obtain $\text{TERM} \subseteq \text{SN}$.

Acknowledgments. We would like to thank the referees for their valuable comments. This research was supported by the Alexander von Humboldt Foundation and the Japanese Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B) 20700015, 2009.

References

Abstract. We present a proof calculus for first-order logic with definitional extensions which is simple – it has only one rule. It is also practical, because it can be used in an intelligent proof assistant for verification of computer programs.

Keywords: formal systems, proof theory.

1 Introduction

The idea that boolean valuation trees can be used as formal calculus for first-order logic was certainly implicit in the work of Beth which lead to his development of semantic tableaux. In this paper we present a calculus for first-order logic with finite valuation trees being the proofs. The calculus has a flavor of Hilbert systems (one rule and many axioms), yet it exhibits, as it should, a close connection to semantic tableaux and Gentzen style sequent calculi. To the best knowledge of the authors, this idea is not directly present in any extant proof calculus, except perhaps in a rather rudimentary form, in the decision procedures based on the calculus of Davis and Putnam.

Our calculus seems to be of interest on its own in pure logic, especially in classroom situations and in self-contained expositions of (classical) first-order logic. This is because the role of syntax in our exposition is absolutely minimal – just the finite valuation trees – and even they are, per definitionem, of semantic character.

Yet, our main motivation for the development of the calculus is entirely pragmatical. It is to be a formal basis for a new version of our intelligent proof assistant (IPA) for a programming language CL (Clausal Language) [CL97]. Programs in CL are just certain implications in extensions by definitions of Peano Arithmetic.

IPAs should offer the natural mathematical style of proofs because their purpose is to assist humans in development and verification of correct programs. Verification at least doubles the amount of mental effort of programming, which is already high. Note that cut elimination is not a central issue in IPAs because natural mathematical proofs are impossible without cuts in the form of lemmas.

A typical work in an IPA is shown in Fig. 1. A theory is extended by a definition of the symbol $f$ (a program) and a lemma $(n)$ about it is proved. The
Fig. 1. A schema of development of proofs by a proof assistant in extensions of theories.

lemma is then used in a proof of the theorem \((m)\), which is itself done in the style of extensions. The theory is locally extended with the symbol \(g\) and a local lemma \((i)\) about it is proved. Both lemmas are used to finish the proof of \((m)\).

This paper is developed as a purely logical exposition of proofs in extensions of first-order theories starting from propositional logic (Sect. 2). We then treat quantification (Sect. 3), equality (Sect. 4), and extensions by definitions (Sect. 5). The paper has been shortened for the CiE abstract booklet. Refer to [KV09a] for a full version with detailed proofs and an example. We have substantially reworked and simplified the calculus in the meantime. This development is presented in [KV09b].

2 Propositional Logic

2.1 Syntactic preliminaries. We use the standard notions of language (denoted by \(L\)) for first-order logic with predicate symbols \((P, R, \ldots)\), function symbols \((f, g, \ldots)\), and terms \((t, s, \ldots)\). We will be concerned with countable languages only. Formulas \((A, B, \ldots)\) are built up from atomic formulas by the propositional connectives \(\neg, \land, \lor, \rightarrow\), and the quantifiers \(\forall, \exists\) binding variables \((x, y, z, \ldots)\). We use \(A \leftrightarrow B\) as an abbreviation for \((A \rightarrow B) \land (B \rightarrow A)\).

A sentence is a formula with no free variables. A set of sentences is a theory if codes of all its sentences form a recursive set. Theories are denoted by the letters \(T\) and \(S\). We write \(T\) and \(A, T\) as abbreviations of \(T \cup S\) and \(\{A\} \cup T\) respectively. Finite theories are denoted by the capital greek letters \(\Gamma\) and \(\Delta\).

Note that within this section, we treat quantified sentences (i.e., \(\forall x\ A[x]\) or \(\exists x\ A[x]\)) as atomic. We have defined sentences with quantifiers from start in order to avoid having separate definitions of propositional and quantified sentences.
2.2 Propositional semantics. A truth assignment is a set \( \mathcal{I} \) of atomic sentences. A truth assignment \( \mathcal{I} \) propositionally satisfies an atomic or quantified sentence \( A \), in writing \( \mathcal{I} \models_p A \), if \( A \in \mathcal{I} \). We inductively extend the relation \( \models_p \) to all sentences using the classical interpretation of propositional connectives. We write \( \mathcal{I} \models_p T \) if \( \mathcal{I} \models_p A \) for all \( A \in T \). We define \( \mathcal{I} \not\models_p T \) to hold if \( \mathcal{I} \models_p A \) for some \( A \in T \).

We say that \( T \) propositionally implies \( S \), in writing \( T \models_p S \), if for all \( \mathcal{I} \) s.t. \( \mathcal{I} \models_p T \) we have \( \mathcal{I} \models_p S \).

We say that \( T \) propositionally implies one of \( S \), in writing \( T \models_p \Rightarrow_p S \), if for all \( \mathcal{I} \) s.t. \( \mathcal{I} \models_p T \) we have \( \mathcal{I} \not\models_p S \).

We call \( T \) the antecedent of the assertion \( T \models_p \Rightarrow_p S \), and \( S \) its succedent. Any assertion \( T, T' \models_p \Rightarrow_p S, S' \) is called a weakening of \( T \models_p S \).

2.3 Initial propositional properties. The assertions listed in Fig. 2 are called the initial propositional properties. The properties are given parenthesized names. The names relate to their use in Lemma 2.11 for the treatment of eponymous rules of the propositional sequent calculus \( G3cp \) [TS00, page 77]. The rules of \( G3cp \) are listed in Fig. 3. Note that we use Dragalin’s variant with \( \neg \) instead of \( \bot \) [Dra79].

![Initial propositional properties](image)

It is decidable whether \( \Gamma \models_p \Delta \) is a weakening of an initial property.

The following lemma asserts obvious properties of the relation \( \models_p \). They form the basis of our proof calculus.

2.4 Lemma.

(a) The initial propositional properties are true.
(b) (Weakening) If \( T \models_p S \), then \( T, T' \models_p S, S' \).
(c) (Cut) \( T \models_p S \) iff \( A, T \models_p S \) and \( T \models_p A, S \).

2.5 Valuation trees. A valuation tree (denoted by \( \mathcal{D}, \mathcal{E} \)) is a tree with each node either a leaf or an internal node with two ordered predecessors. A leaf is designated by the symbol \( \circ \). Internal nodes are labeled with sentences, and written down as

\[
\begin{array}{c}
\mathcal{D} \\
A
\end{array}
\]

where \( \mathcal{D} \) and \( \mathcal{E} \) are the respective subtrees.

In Par. 2.6 we will fix the properties of valuation trees in a more general way, but the following informal interpretation should give the reader an intuition about them.
A valuation tree assigns the sentence $A$ in an internal node the truth value “true” in the left subtree $D$ and “false” in the right subtree $E$. We will have $D$ proves $A$ iff for every path $p$ through $D$ we have:

\[ \{ B \mid B \in p \text{ is true} \}, \{ \neg B \mid B \in p \text{ is false} \} \vdash_p A. \]

Moreover, for each $p$, the truth of the satisfaction relation will be decided only by the form of the sentences involved without involving semantics.

2.6 Proofs in propositional logic. We will now define a five-place relation $D$ $p$-witnesses (propositionally proves) $T, \Gamma \models_p S, \Delta$, in writing $D \vdash_p T; \Gamma \models_p S; \Delta$, as the least relation satisfying:

- $\vdash_p T; \Gamma \models_p S; \Delta$ if the assertion $(T \land \Delta)$, $\Gamma \models_p (S \land \Gamma)$, $\Delta$ is a weakening of an initial propositional property,
- $\frac{D}{E} \vdash_p T; \Gamma \models_p S; \Delta$ if $D \vdash_p T; A, \Gamma \models_p S; \Delta$ and $E \vdash_p T; \Gamma \models_p S; A, \Delta$.

We abbreviate $D \vdash_p \emptyset; \Gamma \models_p \emptyset; \Delta$ and $D \vdash_p T; \emptyset \models_p S; \emptyset$ to $D \vdash_p \emptyset \models_p S, \emptyset$ respectively.

2.7 Lemma (Soundness). If $D \vdash_p T; \Gamma \models_p S, \Delta$, then $T, \Gamma \models_p S, \Delta$.

Proof. By induction on $D$ using the preceding lemmas.

2.8 Remark. Note that our proofs, i.e., the valuation trees $D$, are not, as it is usual, related to syntactic objects (i.e., to sequents), but by Lemma 2.7 they rather witness the truth of a semantic property. In order to avoid ambiguity, the informal phrase “$D$ $p$-witnesses $T, \Gamma \models_p S, \Delta$” should be understood as the assertion “$D \vdash_p T; \Gamma \models_p S, \Delta$”.

Also note that the finite sets $\Gamma$ and $\Delta$ in the $p$-witnessing relation are necessary in order to have the relation decidable. Informally, $T$ is the set of all assumptions, while $\Gamma$ contains a “working subset” (and likewise for $S$ and $\Delta$).

We could have introduced a three-place relation $D \vdash_p T \models_p S$, but then, even though $T$ and $S$ are recursive, the relation would not be decidable as expected, but only semi-decidable (r.e.). This is because $\vdash_p T \models_p S$ would then mean the r.e. relation “$T \models_p S$ is a weakening of an initial property”.

2.9 Lemma (Proof weakening). If $D \vdash_p T; \Gamma \models_p S; \Delta$, then also $D \vdash_p (T, T'); (\Gamma, \Gamma') \models_p (S, S'); (\Delta, \Delta')$.

Proof. By induction on $D$.

2.10 Free cut free valuation trees. A valuation tree $E$ $p$-witnessing $T, \Gamma \models_p S, \Delta$ contains a free cut if it contains a non-leaf subtree such that

\[ \frac{D_1}{A} \frac{D_2}{T'} \vdash_p T; \Gamma' \models_p S; \Delta' \]

and the cut formula $A$ is neither in $T, S$ nor it is an immediate subformula of a formula from $\Gamma', \Delta'$. If $E$ contains no free cuts, it is free cut free.
2.11 Lemma (Reduction of G3cp proofs to \(\vdash_p\) trees). If there is a derivation \(D\) of a closed sequent \(\Gamma \Rightarrow \Delta\) in G3cp, then there is a valuation tree \(D^*\) p-witnessing \(\Gamma \Rightarrow_p \Delta\). \(D^*\) is free cut free if \(D\) is cut free.

Proof. By induction on the derivation \(D\) of a sequent \(\Gamma \Rightarrow \Delta\) in the calculus G3cp. The construction of the valuation tree for selected propositional rules is depicted in Fig. 3. \(\square\)

\[
\begin{array}{c|c|c}
D & D^* \\
\hline
Ax & A, \Gamma \Rightarrow A, \Delta & \circ \vdash (Ax) \\
L\neg & \frac{\frac{D_1}{\Gamma \Rightarrow A, \Delta}}{\neg A, \Gamma \Rightarrow \Delta} & \circ \vdash (L\neg) & D_1^* \\
R\neg & \frac{\frac{D_1}{A, \Gamma \Rightarrow \Delta}}{\Gamma \Rightarrow \neg A, \Delta} & D_1^* & \circ \vdash (R\neg) \\
L\rightarrow & \frac{\frac{D_1}{\Gamma \Rightarrow A, \Delta}}{A \rightarrow B, \Gamma \Rightarrow \Delta} & \frac{\frac{D_2}{B, \Gamma \Rightarrow \Delta}}{A} & \circ \vdash (L\rightarrow) & D_2^* \\
R\rightarrow & \frac{\frac{D_1}{A, \Gamma \Rightarrow B, \Delta}}{\Gamma \Rightarrow A \rightarrow B, \Delta} & \frac{\frac{D_2}{B}}{A} & \circ \vdash (R\rightarrow) & D_2^* \\
\text{Cut} & \frac{\frac{D_1}{A, \Gamma \Rightarrow \Delta}}{A \Rightarrow A, \Delta} & \frac{\frac{D_2}{\Gamma \Rightarrow \Delta}}{A} & D_1^* & D_2^* \\
\end{array}
\]

Fig. 3. Translation of selected G3cp proofs to valuation trees.

2.12 Theorem (Soundness and completeness of \(\vdash_p\)). We have \(T \Rightarrow_p S\) iff \(D \vdash_p T \Rightarrow_p S\) for a free cut free \(D\).

Proof. Completeness follows from 2.11 and completeness of G3cp. \(\square\)

3 Quantification Logic

In this section, we extend our calculus to the first-order quantification logic without equality.

3.1 Semantics of first-order quantification logic. We assume the standard notion of a structure \(M\) for a first-order language \(\mathcal{L}\) not containing the equality symbol \(=\). A structure has a non-empty domain \(M\), assigns meaning to function
and predicate symbols of $\mathcal{L}$, and extends the meaning in the usual way to terms and formulas. We write $\mathcal{M} \models A$ when $\mathcal{M}$ satisfies the sentence $A$.

We define $\mathcal{M} \models T$ ($\mathcal{M}$ is a model of $T$), $T \models S$, $\mathcal{M} \models T$, and $T \models S$ analogously to their propositional counterparts with the additional condition that if $T \models S$ or $T \models S$, then $T$ and $S$ are theories in the same language.

### 3.2 Logical consequence in expansions of structures.

We denote by $\mathcal{L}[\overline{F}]$ a language which is just like $\mathcal{L}$, but contains new predicate or function symbols $\overline{F} = F_1, \ldots, F_n$. We write $A[\overline{F}]$ to express that the formula $A$ may contain symbols $\overline{F}$. Let $\mathcal{M}$ be a structure for a language $\mathcal{L}$. As usual, a structure $\mathcal{M}'$ is an expansion of $\mathcal{M}$ to $\mathcal{L}[\overline{F}]$ if $\mathcal{M}'$ is a structure for $\mathcal{L}[\overline{F}]$ with the same domain as $\mathcal{M}$, and coincides with $\mathcal{M}$ on all symbols of $\mathcal{L}$.

We write $T \models_{\mathcal{L},\overline{F}} S$ if all of the following conditions are satisfied: (a) $\mathcal{L}$ is a language, and $\overline{F}$ are new symbols, (b) $T$ is a theory in $\mathcal{L}$, (c) $S$ is a theory in $\mathcal{L}[\overline{F}]$, and (d) for each structure $\mathcal{M}$ for $\mathcal{L}$ such that $\mathcal{M} \models T$, there is an expansion $\mathcal{M}'$ of $\mathcal{M}$ to $\mathcal{L}[\overline{F}]$ such that $\mathcal{M}' \models S$.

### 3.3 Lemma.

(a) If $T \models_{\emptyset} S$, and $T$, $S$ are in the same language, then $T \models S$.
(b) $T \models S$ with $T$ and $S$ in $\mathcal{L}$ iff $T \models_{\mathcal{L},\emptyset} S$.
(c) (Weakening) If $T \models_{\mathcal{L},\overline{F}} S$, then $T, T' \models_{\mathcal{L},\overline{F},\emptyset} S, S'$ for any new $\overline{G}$, $T'$ in $\mathcal{L}$, and $S'$ in $\mathcal{L}[\overline{F}, \overline{G}]$.

### 3.4 Initial quantification properties.

The initial quantification properties are (i) the initial properties from Fig. 2 with $\models_{\emptyset}$ replaced by $\models_{\mathcal{L},\emptyset}$ provided $A$ and $B$ are from $\mathcal{L}$; (ii) the following initial properties of quantifiers:

\[
\begin{align*}
(\exists) & \quad A[x/t] \models_{\mathcal{L},\emptyset} \exists x A[x], \\
(L\exists) & \quad \exists x A[x] \models_{\mathcal{L},\emptyset} A[x/t], \\
(R\exists) & \quad A[x/c] \models_{\mathcal{L},\emptyset} \forall x A[x], \\
(R\forall) & \quad \emptyset \models_{\mathcal{L},\emptyset} \forall x A[x], \neg A[x/c]
\end{align*}
\]

provided $x$ is the only free variable in $A[x]$ from $\mathcal{L}$, $t$ is a closed term from $\mathcal{L}$, and $c \notin \mathcal{L}$.

Note that by using the notion of logical consequence in expansions of structures we never have to deal with formulas containing free variables.

### 3.5 Lemma. Initial quantification properties are true.

### 3.6 Theorem (Expansion cuts).

$T \models_{\mathcal{L},\emptyset} S$ iff (i) $A[\overline{F}], T \models_{\mathcal{L}[\overline{F}],\emptyset} S$ and (ii) $T \models_{\mathcal{L},\overline{F}} A[\overline{F}], S$.

### 3.7 Proofs in quantification logic.

Thm. 3.6 can be strengthened by replacing in it everywhere $\emptyset$ with $\overline{G}$. As can be seen from the form of the initial properties in Par. 3.4 and from the following definition of proofs in $\vdash_{q\exists}$ we will use the notion of expansion consequence in an even weaker form with $\overline{F}$ containing at most one symbol (including none).

A seven-place relation $\mathcal{D}$ q-witnesses $T, \Gamma \models_{\mathcal{L},\overline{F}} S, \Delta$, in writing $\mathcal{D} \vdash_{q\exists} T; \Gamma \models_{\mathcal{L},\overline{F}} S; \Delta$, is defined as the least relation satisfying:
• $\vdash_q T; \Gamma \models_{\mathcal{L}, \mathcal{F}} S; \Delta$ if the assertion $(T \cap \Delta), \Gamma \models_{\mathcal{L}, \mathcal{F}} (S \cap \Gamma), \Delta$ is a weakening of an initial quantification property,

• $\frac{\mathcal{D}}{A[\bar{F}]} \vdash_q T; \Gamma \models_{\mathcal{L}, \emptyset} S; \Delta$ if $\mathcal{D} \vdash_q T; A[\bar{F}], \Gamma \models_{\mathcal{L}[\bar{F}], \emptyset} S; \Delta$ and $\mathcal{E} \vdash_q T; \Gamma \models_{\mathcal{L}, \emptyset} S; A[\bar{F}], \Delta$.

3.8 Lemma (Reduction of G3c proofs to $\vdash_q$ trees). If there is a derivation $\mathcal{E}$ in G3c of a closed sequent $\Gamma \models \Delta$ in language $\mathcal{L}$, then there is a valuation tree $\mathcal{E}^*$ q-witnessing $\Gamma \models_{\mathcal{L}, \emptyset} \Delta$. $\mathcal{E}^*$ is free cut free if $\mathcal{E}$ is cut free.

Proof. The proof goes along the lines of the proof of Lemma 2.11 with the difference that we need to take into account the extensions of $\mathcal{L}$ with Henkin constants in the translated valuation tree $\mathcal{E}^*$. The translation of rules of G3c is the same as in Lemma 2.11 for the propositional rules (see Fig. 3). The translation of quantification rules is in Fig. 4. □

\begin{center}
\begin{tabular}{c c c}
\hline
$\mathcal{D}$ & $\mathcal{D}^*$ \\
\hline
$\text{D}_1$ & $\text{D}_1$ \\
$\text{R}\exists$ & $\vdash_q (\text{R}\exists)$ \\
$\Gamma \models A[x/t], \exists x A[x], \Delta$ & $\frac{A[x/t]}{\text{D}_1^*}$ \\
$\Gamma \models \exists x A[x], \Delta$ & \\
$\text{L}\forall$ & $\vdash_q (\text{L}\forall)$ \\
$A[x/t], \forall x A[x], \Gamma \models \Delta$ & $\frac{A[x/t]}{\text{D}_1^*}$ \\
$\forall x A[x], \Gamma \models \Delta$ & \\
$\text{L}\exists$ & $\vdash_q (\text{L}\exists)$ \\
$A[y/c], \exists x A[x], \Gamma \models \Delta$ & $\frac{A[x/c]}{\text{D}_1^*}$ \\
$\exists x A[x], \Gamma \models \Delta$ & \\
$\text{R}\forall$ & $\vdash_q (\text{R}\forall)$ \\
$\Gamma \models A[y/c], \Delta$ & $\frac{A[x/c]}{\text{D}_1^*}$ \\
$\Gamma \models \forall x A[x], \Delta$ & $\frac{\neg A[x/c]}{\neg A[x/c]}$ \\
\hline
\end{tabular}
\end{center}

Fig. 4. Translation of G3c proofs to valuation trees.

3.9 Theorem (Soundness and completeness of $\vdash_q$). We have in quantification logic $T \models_{\mathcal{L}, \emptyset} S$ iff $\mathcal{D} \vdash_q T \models_{\mathcal{L}, \emptyset} S$ for a free cut free $\mathcal{D}$.

Proof. The soundness direction $(\leftarrow)$ is proved similarly to Lemma 2.7, and the completeness direction $(\rightarrow)$ follows from Lemma 3.8. □

4 First-order Logic with Equality

Had a language $\mathcal{L}$ of the quantification calculus from the previous section contained the binary equality predicate symbol $=$, it would have to be treated as
a non-logical one, meaning that the usual properties of \( = \) would have to be supplied by axioms in \( T \).

For the rest of this paper, we treat the equality symbol as a logical one, and we will accordingly modify our proof calculus by strengthening the use of the initial properties without changing the definition of expansion cuts.

4.1 Semantics of equality. For the rest of this paper, we assign in structures the usual interpretation of the equality symbol assumed to be in \( L \). This means that, although in the definition of \( \models \) we add the clauses \( M \models s = t \), no other semantic definition needs to be explicitly changed. Thus, in particular, Lemma 3.3, Thm. 3.6, and Lemma 3.5 remain to hold.

4.2 Equality in sequent calculi. We have reduced the completeness proofs for the calculus \( \vdash_p \) and \( \vdash_q \) to the completeness of sequent calculi \( G3cp \) and \( G3c \) respectively. In this section we will reduce the equation calculus \( G3c = [NvP98] [TS00, page 134] \) to the calculus \( EC⇔⊇≡D \) defined in Par. 4.4. The former calculus contains two rules dealing with equality:

\[
\begin{align*}
\text{Ref} & \quad \frac{t = t, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta} \\
\text{Rep} & \quad \frac{P[x/s], P[x/t], t = s, \Gamma \Rightarrow \Delta}{P[x/t], t = s, \Gamma \Rightarrow \Delta} \quad \text{for atomic } P.
\end{align*}
\]

4.3 Equality closure. For a theory \( T \) and a set of closed terms \( D \) we define the equality closure of \( T \) over \( D \) as the smallest set \( Eq^*_D(T) \) satisfying:

- \( T \cup \{(t = t) \mid t \in D\} \subseteq Eq^*_D(T) \),
- \( P[x/s] \in Eq^*_D(T) \) provided \( P[x/t], (t = s) \in Eq^*_D(T) \) and \( P[x] \) is an atomic formula with at most \( x \) free.

Let Terms\((T)\) denote the set of all subterms occurring in the atomic sentences of \( T \). We set \( Eq^*_S,T,F^*(T) := Eq^*_D(T) \) where

\[
D = \text{Terms}(T) \cup (\text{Terms}(S) \setminus \{s \mid \text{term } s \text{ contains any of } \bar{F}\}).
\]

4.4 Proofs in first-order logic with equality. We now extend the relation of \( q \)-witnessing (see Par. 3.7) to first-order logic with equality without adding any new initial properties or changing the extension cuts. The difference is that instead of letting \( \circ \) witness a set \( \Gamma \) in antecedents of \( \models \), we saturate \( \Gamma \) with equalities by forming its equality closure.

The seven place relation \( D e\text{-witnesses } T, \Gamma \models_{\mathcal{L},\bar{F}} S, \Delta, \) in writing \( D \vdash_e T; \Gamma \models_{\mathcal{L},\bar{F}} S; \Delta, \) is the least relation satisfying:

- \( \circ \vdash_e T; \Gamma \models_{\mathcal{L},\bar{F}} S; \Delta \) if \( (T \cap \Delta), Eq^*_{\Delta,\bar{F}}(\Gamma) \models_{\mathcal{L},\bar{F}} (S \cap \Gamma), \Delta \) is a weakening of an initial quantification property,
- \( \frac{D \models_{\bar{F}} E}{A[\bar{F}]} \vdash_e T; \Gamma \models_{\mathcal{L},\emptyset} S; \Delta \) if \( D \vdash_e T; A[\bar{F}], \Gamma \models_{\mathcal{L}[\bar{F}],\emptyset} S; \Delta, \) and \( E \vdash_e T; \Gamma \models_{\mathcal{L},\bar{F}} S; A[\bar{F}], \\Delta. \)

Note that the relation is decidable because the equality closure for finite sets is finite.
4.5 Lemma (Reduction of $G_{3c}^{-}$ proofs to $\vdash_e$ trees). If there is a derivation $E$ in $G_{3c}^{-}$ of a closed sequent $\Gamma \Rightarrow \Delta$ in a language $\mathcal{L}$, then there is a valuation tree $E^*$ $e$-witnessing $\Gamma \Rightarrow_{L,\emptyset} \Delta$. $E^*$ is free cut free if $E$ is cut free.

Proof. Equational rules can be permuted above all other rules in a derivation. We may assume that $E$ is in this form. We show that if a sequent $\Gamma' \Rightarrow \Delta'$ in $\mathcal{L}'$ is derived by equational rules only, then we have $\circ \vdash_e \Gamma' \Rightarrow_{L',\emptyset} \Delta'$. We can thus eliminate all equational rules from the top of the derivation $E$, and translate the remaining rules as in Lemma 3.8. $\Box$

4.6 Theorem (Soundness and completeness of $\vdash_e$). We have $T \Rightarrow_{L,\emptyset} S$ iff $D \vdash_e T \Rightarrow_{L,\emptyset} S$ for a free cut free $D$.

Proof. Soundess is proved by induction, completeness follows from Lemma 4.5. $\Box$

4.7 Discussion. Our treatment of full first-order logic uses equality automatically. This is not only possible, but also feasible, by the existence of an efficient congruence closure algorithm of [DST80].

We have on purpose proved the completeness of our calculi by reduction of corresponding complete sequent calculi. The reason was that we wished to exhibit similarities between sequent proofs and valuation trees. In a self-contained exposition we would employ the well-known direct method used with sequent calculi. The method is even more natural with valuation trees. We must permit infinite valuation trees and assure that we systematically assign truth values to all sentences in $T, S$ as well as to all immediate subformulas of sentences appearing on branches closer to the root of the constructed valuation tree. This means in particular, that we must use all terms of $\mathcal{L}$ for the instantiation of quantifiers. If such a systematic procedure fails to stop with a finite valuation tree, and thus fails to $e$-witness the property $T \Rightarrow S$, then the tree must contain an infinite consistent path. We then perform the well-known construction of a structure giving all sentences on the path their assigned values. The structure will thus satisfy $T$ and falsify all of $S$.

5 Extensions of Theories by Definitions

In this section we treat extension of theories by definitions of predicate and function symbols. For that it suffices to extend the base case of the proof predicate $\vdash_e$ with new initial properties justifying the extensions.

5.1 Initial extension properties. Initial extension properties are the initial quantification properties (see Par. 3.4) plus the following ones:

(PDef) $\emptyset \Rightarrow_{L,R} \forall \bar{x} (R(\bar{x}) \leftrightarrow A(\bar{x}))$

(FDef) $\forall \bar{x} \exists y B(\bar{x}, y) \Rightarrow_{L,R} \forall \bar{x} \forall y (f(\bar{x}) = y \leftrightarrow B(\bar{x}, y))$.

for any formulas $A[x_1, \ldots, x_n]$, $B[x_1, \ldots, x_n, y]$ in $\mathcal{L}$ with only the indicated variables free and any symbols $R, f$ not in $\mathcal{L}$.
5.2 Lemma. Initial extension properties are true. □

5.3 Proofs in extension logic. We define a new relation \( D \) \( d \)-witnesses \( T, \Gamma \vdash_{\mathcal{L}, \vec{F}} S, \Delta \), in writing \( D \vdash_{d} T; \Gamma \vdash_{\mathcal{L}, \vec{F}} S; \Delta \). The relation is just like the \( \vdash_{e} \) relation (see Par. 4.4) but we permit initial extension properties instead of quantificational ones.

Thm. 5.4 asserts the well-known conservativity of extensions by definitions where any use of new symbols in a \( \vdash_{d} \) proof of a property not containing new symbols can be eliminated so the property has an \( \vdash_{e} \) proof (see, e.g., [Sho67] or [TS00, page 124]). Thm. 5.5 asserts that the \( \vdash_{d} \) calculus is sound. We cannot have completeness unless we are willing to examine all possible definitional extensions. For that we would have to restrict the definition of \( \vdash_{\mathcal{L}, \vec{F}} \) in Par. 3.2 to Henkin and definitional extensions. To examine all extensions is, however, not the intent of introducing new symbols. They are introduced in order to manage the complexity of proofs by abbreviating long formulas.

5.4 Theorem. If \( D \vdash_{d} T \vdash_{\mathcal{L}, \emptyset} S \), then \( D^{*} \vdash_{e} T \vdash_{\mathcal{L}, \emptyset} S \) for some \( D^{*} \). □

5.5 Theorem. If \( D \vdash_{d} T \vdash_{\mathcal{L}, \vec{F}} S \), then \( T \vdash_{\mathcal{L}, \vec{F}} S \).

Proof. Similarly to the proof of soundness in Thm. 4.6 except that we use Lemma 5.2 in the base case instead of Lemma 3.5. □

References


Definability of Closure Operations in the $h$-Quasiorder of Labeled Forests

Oleg V. Kudinov$^1$ *, Victor L. Selivanov$^2$ **, and Anton V. Zhukov$^3$ ***

$^1$ S.L. Sobolev Institute of Mathematics
Siberian Division Russian Academy of Sciences
kud@math.nsc.ru
$^2$ A.P. Ershov Institute of Informatics Systems
Siberian Division Russian Academy of Sciences
vseliv@nspu.ru
$^3$ Novosibirsk State Pedagogical University
zhukan@ngs.ru

Abstract. We prove that for any $k \geq 3$ natural closure operations on the quotient structure of the $h$-quasiorder of finite (and also of countable) $k$-labeled forests are first-order definable provided that the minimal non-smallest elements are allowed as parameters. This strengthens our previous result that for any $k \geq 3$ each element of the $h$-quasiorder of finite $k$-labeled forests is definable in the ordinary first order language and, respectively, each element of the $h$-quasiorder of (at most) countable $k$-labeled forests is definable in the language $L_{\omega_1\omega}$, in both cases provided that the minimal non-smallest elements are allowed as parameters. Similar results hold true for two other relevant structures: the $h$-quasiorder of finite (resp. countable) $k$-labeled trees and of finite (resp. countable) $k$-labeled trees with a fixed label of the root element.

Keywords. Labeled tree, labeled forest, closure operation, $h$-quasiorder, definability.

1 Introduction

In [He93,Se04,KS07,KS07a] the structure $(\mathcal{F}_k; \preceq)$, $k \geq 2$, of finite $k$-labeled forests with the $h$-quasiorder $\preceq$ was studied. In [Se07] the structure $(\tilde{\mathcal{F}}_k; \preceq)$ of countable $k$-forests was introduced and studied.

These two main structures of our work are interesting in their own right since the $h$-quasiorder is one in a series of relations on words, trees and forests relevant to computer science [Ku06]. The interest to these structures was initially motivated by their close relationship to the Wadge reducibility and Boolean hierarchy of $k$-partitions [He93,Se04,Se07,Se07a].

* Supported by DFG-RFBR Grant 06-01-04002 and by RFBR grants 08-01-00336 and 07-01-00543a.
** Supported by DFG-RFBR Grant 06-01-04002 and by RFBR grant 07-01-00543a.
*** Supported by RFBR Grant 07-01-00543a.

216
Recall that the \( h \)-quasiorder \( \leq \) of forests labeled by natural numbers from the set \( k = \{0, 1, \ldots, k-1\}, k \geq 2 \), is defined as follows: \( F \leq G \), if there is a monotone mapping from \( F \) to \( G \) that preserves the labelings. The \( h \)-equivalence is the equivalence relation induced by the \( h \)-quasiorder. By \( G\mathcal{F}_k \), (resp. \( G\mathcal{F}_k \)) we denote the set of equivalence classes of finite (resp. countable) \( k \)-labeled forests (formal definitions are given in the next section). For any \( i < k \), let \( p_i \) be the unary operation on \( G\mathcal{F}_k \), (resp. on \( G\mathcal{F}_k \)) induced by the operation that adjoins a new root element to a given finite (resp. countable) \( k \)-labeled forest and labels the root by \( i \). The main result of this paper is now formulated as follows:

**Theorem 1.** For any \( k \geq 3 \) and \( i < k \), the operation \( p_i \) is first-order definable in the structures \((G\mathcal{F}_k; \leq, [0], \ldots, [k-1])\) and \((G\mathcal{F}_k; \leq, [0], \ldots, [k-1])\) where \([0], \ldots, [k-1]\) are the minimal non-smallest elements of the corresponding posets.

Due to the mentioned close relationship of the \( h \)-quasiorder to Wadge reducibility (in particular, the structure of Wadge degrees of \( k \)-partitions of the Baire space with \( \Delta^0_k \)-components is isomorphic to \((G\mathcal{F}_k \setminus \{[0]\}; \leq) \) [Se07a]) such definability results lead to a definability theory for some initial segments of Wadge degrees which is parallel to the extensive (and more complicated) definability theory for the degree structures in computability theory.

Section 2 recalls basic definitions and known facts we will rely upon. In Section 3 we prove our main results, including Theorem 1.

## 2 Basic Definitions and Known Facts

In an attempt to make the paper more self-contained, we cite here some notions and facts frequently used in the subsequent proofs.

We use some standard notation and terminology on posets, in particular \( x < y \) denotes \( x \leq y \land y \not\leq x \), and \( x | y \) denotes \( x \not\leq y \land y \not\leq x \). By a finite forest we mean a finite poset in which every lower cone is a chain. By a countable forest we mean an (at most) countable poset in which every lower cone is a chain and which does not contain infinite chains. A finite (resp. countable) tree is a finite (resp. countable) forest having the least element (called the root of the tree).

Let \( \mathcal{F}_k, \overline{\mathcal{F}}_k, \mathcal{T}_k \) and \( \overline{\mathcal{T}}_k \) be the classes of all finite \( k \)-forests, countable \( k \)-forests, finite \( k \)-trees and countable \( k \)-trees, respectively. For each \( i < k \), let \( \mathcal{T}_k^i \) (resp. \( \overline{\mathcal{T}}_k^i \)) be the set of finite (resp. countable) \( k \)-trees which carry the label \( i \) on their roots. We use \( \leq \) also to denote the partial ordering (of the \( h \)-equivalence classes) induced by the \( h \)-quasiorder \( \leq \). For technical reasons we consider also the empty \( k \)-forest \( \emptyset \in \mathcal{F}_k \) (which is not a tree) assuming that \( \emptyset \leq P \) for each \( P \in \mathcal{F}_k \). A \( k \)-tree \((T; \leq, c) \in G\overline{\mathcal{T}}_k \) is repetition free if \( c(x) \neq c(y) \) whenever \( y \) is an immediate successor of \( x \) in \((T; \leq)\).

Let \( G\overline{\mathcal{F}}_k, G\overline{\mathcal{T}}_k, G\overline{\mathcal{T}}_k^i, G\mathcal{F}_k, G\mathcal{T}_k, G\mathcal{T}_k^i \) be the quotient sets of respectively \( \overline{\mathcal{F}}_k, \overline{\mathcal{T}}_k, \mathcal{T}_k, \mathcal{T}_k^i \) modulo the \( h \)-equivalence. We call \( h \)-equivalence classes of \( k \)-forests and \( k \)-trees generalized \( k \)-forests and generalized \( k \)-trees, respectively.
We call a generalized $k$-forest (resp. a generalized $k$-tree) finite if it contains a finite $k$-forest (resp. a finite $k$-tree).

For an arbitrary $F \in \mathcal{F}_k$, $[F]$ denotes the $h$-equivalence class of $F$ in any of the structures under consideration. For example, $[0]$ can denote element of $G\mathcal{F}_k$, $G\mathcal{T}_k^0$, $G\mathcal{F}_k^0$, $G\mathcal{T}_k$ or $G\mathcal{T}_k^0$. Which variant is meant, depends on the context. In fact, one can identify $h$-equivalence classes from different quotient structures that have nonempty intersections (it follows from the observation that any two $h$-equivalent $k$-trees must have the same label on their roots). So we can think that $G\mathcal{T}_k^i \subseteq G\mathcal{T}_k \subseteq G\mathcal{F}_k$ and so on. Throughout this paper, we use capital letters to denote the “ordinary” $k$-forests and trees, and lower-case letters to denote the corresponding $h$-equivalence classes.

We define the join $\bigcup S$ of a countable set $S$ of $k$-forests as the disjoint union of forests from the set $S$. We call the join of a finite set $S$ finite and of a countable set $S$ countable. For arbitrary $k$-trees $T_0, T_1, \ldots$, their join $F = T_0 \sqcup T_1 \sqcup \cdots$ is a $k$-forest whose trees are exactly $T_0, T_1, \ldots$. The $h$-equivalence is a congruence relation that respects the join, so we apply the term “join” and the notation $\bigcup$ also to the operation on the $h$-equivalence classes of $k$-forests, induced by the “ordinary” join.

For every countable $k$-forest $F$ and every $i < k$, let $p_i(F)$ be the countable $k$-tree obtained from $F$ by joining a new smallest element and assigning the label $i$ to this element. In particular, $p_i(\emptyset)$ will be the singleton tree carrying the label $i$. The operation $p_i$ preserves the $h$-equivalence, so we use the same notation $p_i$ to denote also the operation induced by $p_i$ on the generalized $k$-forests.

Any finite $k$-forest is equivalent to a term of signature $\{\sqcup, p_0, \ldots, p_{k-1}, \emptyset\}$ without variables. E.g., the words 1210 and 2010 (i.e., $k$-labeled chains) are equivalent to $p_1(p_2(p_1(p_0(\emptyset))))$ and $p_2(p_0(p_1(p_0(\emptyset))))$, respectively. Below we omit parenthesis whenever they are clear from the context, e.g. we could write the last term as $p_2p_0p_1p_0(\emptyset)$. Every countable $k$-forest can be represented by a term as well, but this time countable joins are allowed in terms.

Recall that a relation in a structure is first-order definable (abbreviation: definable) if there is a first-order formula of signature of the structure true exactly on the tuples that satisfy the relation. More generally, a relation in a structure is definable in a language $L$ (abbreviation: $L$-definable) if there is a formula in the language $L$ of signature of the structure true exactly on the tuples that satisfy the relation. In the countable case, we need the language $L_{\omega_1\omega}$, whose only difference from the language of first-order logic is the possibility to use countable conjunctions and disjunctions. An element is definable (in a language $L$) if the corresponding singleton set is definable (in the language $L$). A function is definable if its graph is definable.

**Proposition 1** ([Se07]). (i) Every $T \in \mathcal{T}_k$ (resp. $T \in \mathcal{F}_k$) is $h$-equivalent to some repetition free $S \in \mathcal{T}_k$ (resp. $S \in \mathcal{F}_k$).

(ii) Every $t \in G\mathcal{T}_k$ (resp. $t \in G\mathcal{F}_k$) can be represented in the form $t = p_i(\bigcup U)$, where $U$ is a finite subset of $G\mathcal{T}_k \setminus G\mathcal{T}_k^i$ (resp. a countable subset of $G\mathcal{T}_k \setminus G\mathcal{T}_k^i$).

218
(iii) The sequence \( (G\bar{T}_k^0, \ldots, G\bar{T}_k^{k-1}) \) (resp. \( (G\bar{T}_k^0, \ldots, G\bar{T}_k^{k-1}) \)) is a partition of \( G\bar{T}_k \) (resp. of \( G\bar{T}_k \)).

Next we formulate some properties of the sets \( G\bar{F}_k \) and \( G\tilde{F}_k \) enriched by the closure operations \( p_i \). For this we recall the following notion introduced in [Se82]. By a semilattice with discrete closures (a dc-semilattice for short) we mean a structure \((S; \cup, p_0, \ldots, p_{k-1})\) satisfying the following axioms:

1) \((S; \cup)\) is an upper semilattice; as usual, by \( \leq \) we denote the induced partial order on \( S \) defined by \( x \leq y \iff x \cup y = y \).

2) For any \( i < k \), \( p_i \) is a closure operation on \((S; \leq)\), i.e., it has the properties \( x \leq p_i(x) \), \( x \leq y \rightarrow p_i(x) \leq p_i(y) \) and \( p_i(p_i(x)) \leq p_i(x) \).

3) For all distinct \( i, j < k \), \( p_i(x) \leq p_j(y) \rightarrow p_i(x) \leq y \).

4) Any \( p_i(x) \) is join-irreducible, i.e., \( p_i(x) \leq y \cup z \rightarrow (p_i(x) \leq y \vee p_i(x) \leq z) \).

By a \( \sigma \)-semilattice we mean an upper semilattice in which every countable set of elements has a supremum. By a \( dce \sigma \)-semilattice we mean a \( dc \sigma \)-semilattice \((S; \cup, p_0, \ldots, p_{k-1})\) such that \((S; \cup)\) is a \( \sigma \)-semilattice and the axiom 4) of \( dc \sigma \)-semilattices holds for suprema of countable subsets of \( S \), i.e. \( p_i(x) \leq \bigcup_{j<\omega} y_j \) implies that \( p_i(x) \leq y_j \) for some \( j < \omega \); we express this by saying that \( p_i(x) \) is \( \sigma \)-join-irreducible.

**Proposition 2 ([Se04,Se07]).** (i) For any \( k \geq 2 \), \((G\bar{F}_k; \cup, p_0, \ldots, p_{k-1})\) is a \( dc \sigma \)-semilattice and \((G\bar{F}_k; \cup, p_0, \ldots, p_{k-1})\) is a \( dce \sigma \)-semilattice. In fact, both structures \((G\bar{F}_k; \leq)\) and \((G\bar{F}_k; \leq)\) are distributive lattices.

(ii) For any \( k \geq 2 \), \((G\bar{F}_k; \leq)\) and \((G\bar{F}_k; \leq)\) are well founded and have no infinite antichains. Furthermore, the rank of \((G\bar{F}_k; \leq)\) (resp. of \((G\bar{F}_k; \leq)\)) is \( \omega \) (resp. \( \omega_1 \)).

For a \( dce \sigma \)-semilattice \( S \), let \( \sigma ji(S) \) (resp. \( ji(S) \)) denote the set of non-zero \( \sigma \)-join-irreducible (resp. of non-zero join-irreducible) elements of \( S \), then of course \( \sigma ji(S) \subseteq ji(S) \). By canonical representation of \( x \in G\bar{F}_k \) we mean a representation \( x = \bigcup \mathcal{Y} \) for some finite antichain \( \mathcal{Y} \subseteq ji(G\bar{F}_k) \).

**Proposition 3 ([Se04,Se07,Se07a]).** (i) \( ji(G\bar{F}_k) = G\bar{T}_k \).

(ii) \( \sigma ji(G\tilde{F}_k) = G\tilde{T}_k \).

(iii) \( ji(G\tilde{F}_k) \) coincides with the set of elements of the form \( t_0 \cup t_1 \cup \cdots \), for some ascending chain \( t_0 \leq t_1 \leq \cdots \) in \( G\tilde{T}_k \).

(iv) Any element of \( G\bar{F}_k \) has a unique canonical representation.

Define the function \( t \) from \( G\tilde{T}_k \) to \( G\tilde{F}_k \) by \( t = \bigcup \{ s \in G\tilde{T}_k : s < t \} \). By Proposition 2(ii), this definition is correct because the set \( \{ s \in G\tilde{T}_k : s < t \} \) is countable. Moreover, by the same proposition, if \( t \in G\tilde{T}_k \) then \( t' \in G\bar{F}_k \) because in this case \( \{ s \in G\tilde{T}_k : s < t \} \) is finite. Obviously, \( t' \) is the biggest element \( f \) of \( G\tilde{F}_k \) such that \( f < t \), and the function \( x \mapsto x' \) is definable in \((G\bar{F}_k; \leq)\) and \((G\tilde{F}_k; \leq)\).

We call a set of generalized \( k \)-trees primitive if it has a greatest element, otherwise a family is nonprimitive. Note that join of a primitive family is equal
to its greatest element. Obviously a set is primitive iff its join is a generalized k-tree. For any countable \( U \subseteq \mathcal{G}T_k \) and \( a \in U \), denote \( U_a = \bigcup \{\hat{x} \mid x \in U\} \setminus \hat{a} \) where \( \hat{x} = \{y \in \mathcal{G}T_k \mid y \leq x\} \) is the lower cone defined by \( x \) and \( \hat{a} \) is the upper cone defined by \( a \).

For a countable set \( U \subseteq \mathcal{G}T_k \), let \( \text{max}(U) \) denote the set of maximal elements of \( U \) (under \( \leq \)) and let \( \text{nm}(U) \) be the set of elements of \( U \) which have no maximal elements above them. Let also \( j\text{max}(U) = \bigcup \text{max}(U) \) and \( j\text{nm}(U) = \bigcup \text{nm}(U) \).

Let us recall a nontrivial explicit description of \( t' \), \( t = p_i(\bigsqcup U) \), by induction on the rank of \( t \) in \( (\mathcal{G}T_k; \leq) \) in terms of \( \text{max}(U) \) and \( \text{nm}(U) \) using the repetition-free representation from Proposition 1. Note that items (i)–(iv) of Proposition 4 below exclude each other, items (i)–(iii) deal with the case \( \text{nm}(U) = \emptyset \), and item (iv) — with the case \( \text{nm}(U) \neq \emptyset \). Therefore (i)–(iii) apply to both finite and countably infinite generalized trees while (iv) applies only to the infinite generalized \( k \)-trees. Observe that (iii) is exactly the case of primitive \( U \) and (i)–(iii) describe the canonical representation of \( t' \).

Proposition 4 is illustrated by Figures 1 and 2.

Fig. 1: Items (i)–(iii) of Proposition 4 (\( \text{nm}(U) = \emptyset \)).
Proposition 4 ([KS07,KSZ08]). Let $t = p_i(\bigcup \mathcal{U})$ for $i < k$ and a countable \( \mathcal{U} \subseteq G\overline{T}_k \setminus G\overline{T}^i_k \).

(i) If \( \mathcal{U} = \emptyset \), i.e. \( t = p_i(\emptyset) \), then \( t' = \emptyset \).

(ii) If \( \mathcal{U} = \{x_0, \ldots, x_n\} \), where \( n > 0 \) and \( x_0, \ldots, x_n \) are pairwise incomparable elements in \( G\overline{T}_k \setminus G\overline{T}^i_k \), then \( t' = (p_i(x_0 \cup \cdots \cup x_n))' = p_i(y_0) \cup \cdots \cup p_i(y_n) \), where \( y_j = x_j' \cup (\bigsqcup_{i \neq j} x_i) \), and \( p_i(y_0), \ldots, p_i(y_n) \) are pairwise incomparable.

(iii) If \( \mathcal{U} = \{p_j(x)\} \), where \( j \neq i \) and \( x \in G\overline{\mathcal{F}}_k \), then \( t' = (p_ip_j(x))' = p_j(x) \cup p_i((p_j(x))') \) and \( p_j(x), p_i((p_j(x))') \) are incomparable.

(iv) If \( \text{nm}(\mathcal{U}) \neq \emptyset \), and \( \max(\mathcal{U}) = \{x_1, \ldots, x_n\} \), where \( x_1, \ldots, x_n \) are pairwise incomparable generalized \( k \)-trees (the case \( n = 0 \) corresponds to \( \max(\mathcal{U}) = \emptyset \)), then \( t' = \bigcup \tilde{\mathcal{U}} \), where

\[
\tilde{\mathcal{U}} = \{p_i(y_1), \ldots, p_i(y_n)\} \cup \{p_i(\bigcup \text{nm}(\mathcal{U})_a \cup j\max(\mathcal{U})) \mid a \in \text{nm}(\mathcal{U})\}
\]

with \( y_j = x_j' \cup (\bigsqcup_{i \neq j} x_i) \cup j\text{nm}(\mathcal{U}) \) for \( j \in \{1, \ldots, n\} \); moreover, the elements \( p_i(y_1), \ldots, p_i(y_n) \) are pairwise incomparable, \( \max(\tilde{\mathcal{U}}) = \{p_i(y_1), \ldots, p_i(y_n)\} \) and \( \text{nm}(\tilde{\mathcal{U}}) = \{p_i(\bigcup \text{nm}(\mathcal{U})_a \cup j\max(\mathcal{U})) \mid a \in \text{nm}(\mathcal{U})\} \).

3 Main Results

In this section we prove the main results of this paper but first let us establish some lemmas. The following fact was observed in [KS07,KSZ08].

Lemma 1. There exists a first-order formula \( \text{tr}(x) \) of signature \( \{\leq\} \) which defines \( G\mathcal{T}_k \) in \( (G\mathcal{F}_k; \leq) \) and \( G\overline{T}_k \) in \( (G\overline{\mathcal{F}}_k; \leq) \) for all \( k \geq 2 \).
Proof. Let \( \text{ir}(x) \) be a first-order formula of signature \( \{\leq\} \) that defines in every semilattice exactly the non-zero join-irreducible elements (see [KS07] for details). We can take the following formula as \( \text{tr}(x) \):

\[
\text{ir}(x) \land \bigcup \{ y : \text{ir}(y) \land y < x \} < x.
\]

Indeed, \( \text{ir}(x) \) defines \( G^T_k \) in \( GF_k \) [Se04], and \( \bigcup \{ y : \text{ir}(y) \land y < x \} = x' < x \) for each \( x \in G^T_k \). In the case \( x \in GF_k \), \( \text{ir}(x) \) is true iff \( x \in ji(GF_k) \). By Proposition 3, \( ji(GF_k) \) coincides with the set of elements of the form \( t_0 \sqcup t_1 \sqcup \cdots \) for some ascending chain \( t_0 \leq t_1 \leq \cdots \) in \( G^T_k \). Hence for any \( x \in ji(GF_k) \) we have: \( x \in G^T_k \) iff \( \bigcup \{ y : \text{ir}(y) \land y < x \} < x \). \( \Box \)

Define the binary relation \( < \) on \( G^T_k \) as follows: \( s < t \) iff \( t = p_i(s) \) and \( s \not\in G^T_k \) for some \( i < k \). As usual, the restriction of \( < \) to \( G^T_k \) is denoted again by \( < \).

Lemma 2. There exists a first-order formula \( \phi(s, t) \) that defines \( < \) in \( (G^T_k ; \leq) \) and \( (G^T_k ; \preceq) \) for all \( k \geq 2 \).

Proof. We can take the following formula as \( \phi(s, t) \):

\[
\exists x(s \land y(t < s \iff (y \leq s \land y \leq x)) \land y < s(y < x)).
\]

Observe that, by Propositions 2, 3 and the definition of \( ' \), the second conjunct is true in \( (G^T_k ; \leq) \) iff \( t' = s \sqcup x \) is true in \( (GF_k ; \leq) \), and \( \forall y < s(y < x) \) is true in \( (GF_k ; \leq) \).

Our argument works uniformly for both structures. Let \( s < t \) be true in a structure, so \( t = p_i(s) \) and \( s \not\in G^T_k \) for some \( i < k \). By the remark above and Proposition 4, the element \( x = p_i(s') \) witnesses the truth of \( \phi(s, t) \) in the structure.

In the opposite direction, let \( \phi(s, t) \) be true. By Proposition 4, either \( t = p_i(t_1) \) for some \( t_1 \in G^T_k \setminus G^T_k \) or \( t = p_i(t_1 \sqcup t_2) \) for some incomparable \( t_1, t_2 \in G^T_k \setminus G^T_k \). Towards a contradiction, suppose that the second alternative holds. By Proposition 4, \( t' = p_i(t'_1 \sqcup t_2) \sqcup p_i(t_1 \sqcup t'_2) \), so, by symmetry, w.l.o.g. \( s = p_i(t'_1 \sqcup t_2) \) and \( x = p_i(t_1 \sqcup t'_2) \). Since \( t_2 < p_i(t_2) \leq s \), we have \( t_2 \leq s' \leq x' < x \), and hence, by Proposition 2(ii), \( t_2 \leq t_1 \) (a contradiction with \( t_1 \sqcup t_2 \)) or \( t_2 \leq t'_2 \) (a contradiction with a property of the function \( ' \)). Thus, in fact the first alternative holds. But then \( t_1 = s \) and hence \( s < t \). \( \Box \)

Let \( S \) (resp. \( \tilde{S} \)) denote the set of all generalized \( k \)-trees of the form \( p_ip_j(f) \), for some \( i \neq j \) and \( f \in GF_k \) (resp. \( f \in GF_k \)).

Lemma 3. There exists a first-order formula \( \psi(t) \) that defines \( S \) in \( (GF_k ; \leq) \) and \( \tilde{S} \) in \( (GF_k ; \leq) \) for all \( k \geq 2 \).

Proof. By the Lemmas 1 and 2, the formula \( \text{tr}(t) \land \exists s \phi(s, t) \) makes the job. \( \Box \)

Define the binary relation \( <_1 \) on \( G^T_k \) as follows: \( s <_1 t \) iff \( s \) is a maximal element in some (or, equivalently, any) countable set \( U \subseteq G^T_k \setminus G^T_k \) such that \( t = p_i(\sqcup U) \). Obviously, \( s < t \) implies \( s <_1 t \).
**Lemma 4.** There is a first-order formula \( \phi_1(s, t) \) that defines \( \preceq_1 \) in \((G\mathcal{T}_k; \leq)\) and \((G\overline{T}_k; \leq)\) for all \( k \geq 2 \).

**Proof.** We can take the following formula as \( \phi_1(s, t) \):

\[
\exists t_0 \leq t(\phi(s, t_0) \land \neg \exists x(t_0 < x \land \psi(x))).
\]

Indeed, let \( i \) satisfy \( t \in G\overline{T}_k^i \). Note that \( s \preceq_1 t \) implies \( s \notin G\overline{T}_k^i \) and \( t = p_i(s \sqcup f) \) for some \( f \in G\overline{F}_k \), \( s \notin f \). Then element \( t_0 = p_i(s) \) witnesses the truth of \( \phi_1(s, t) \).

Indeed, \( \phi(s, t_0) \) is true by Lemma 2. Suppose, towards a contradiction, that the second conjunct in parenthesis is false, i.e. \( t_0 < x \leq t \) for some \( x \) of the form \( p_jp_i(y), j \neq i \). In both possible cases \( i = j, i \neq j \) we get a contradiction using Proposition 2(i).

In the opposite direction, let \( t \in G\overline{T}_k^i, s \in G\overline{T}_k^j \), and let \( \phi_1(s, t) \) be true with a satisfying value \( t_0 \in G\overline{T}_k \). Then \( \phi(s, t_0) \) and \( \neg \exists x(t_0 < x \land \psi(x)) \) are true.

By Lemma 2, \( t_0 = p_i(s) \) for some \( l < k, j \neq l \). In fact we have \( i = l \) (otherwise, \( t_0 < x \leq t \) and \( \psi(x) \) are true for \( x = p_l(t_0) \), a contradiction). Let now \( t = p_i(\bigsqcup \mathcal{U}) \) for some countable \( \mathcal{U} \subseteq G\overline{T}_k \setminus G\overline{T}_k^i \). Since \( s \leq t_0 \leq t \) and \( j \neq i, s \leq u \) for some \( u \in \mathcal{U} \) by Proposition 2(i). Note that \( s \notin u \) for all \( u \in \mathcal{U} \) (otherwise, \( t_0 < x \leq t \) and \( \psi(x) \) are true for \( x = p_i(u) \), a contradiction). Therefore, \( s \in \max(\mathcal{U}) \) and \( s \preceq_1 t \). \( \square \)

Define the ternary relation \( P \) on \( G\overline{T}_k \) as follows: \( P(s_1, s_2, t) \) iff \( s_1|s_2 \) and \( t = p_i(s_1 \sqcup s_2) \), for some \( i < k \) with \( s_1, s_2 \notin G\overline{T}_k^i \).

**Lemma 5.** There exists a first-order formula \( \phi_2(s_1, s_2, t) \) that defines \( P \) in \((G\mathcal{T}_k; \leq)\) and \((G\overline{T}_k; \leq)\) for all \( k \geq 2 \).

**Proof.** We can take the following formula as \( \phi_2(s_1, s_2, t) \):

\[
s_1|s_2 \land \forall s(\phi_1(s, t) \leftrightarrow (s = s_1 \lor s = s_2)) \land \forall t_1 \leq \ell\forall s(\phi(s, t_1) \rightarrow (s \leq s_1 \lor s \leq s_2)).
\]

Indeed, let \( t = p_i(\bigsqcup \mathcal{U}) \) for a countable \( \mathcal{U} \subseteq G\overline{T}_k \setminus G\overline{T}_k^i \). From Lemmas 4, 3 and Proposition 2 it follows that \( \phi_2(s_1, s_2, t) \) is true iff \( s_1|s_2 \) and \( \max(\mathcal{U}) = \{s_1, s_2\} \) iff \( P(s_1, s_2, t) \) is true. \( \square \)

Define the binary relation \( Q \) on \( G\overline{T}_k \) as follows: \( Q(s_1, s_2) \) iff \( s_1|s_2 \) and \( s_1, s_2 \in G\overline{T}_k^i \) for some \( i < k \).

**Lemma 6.** There is a first-order formula \( \theta(s_1, s_2) \) that defines \( Q \) in \((G\mathcal{T}_k; \leq)\) and \((G\overline{T}_k; \leq)\) for all \( k \geq 2 \).

**Proof.** We can take the formula \( \exists^{k-1}t \phi_2(s_1, s_2, t) \) (where \( \exists^{k-1}t \ldots \) means “there are exactly \( k-1 \) values of \( t \) such that . . .”) as \( \theta(s_1, s_2) \). Indeed, if \( s_1 = p_i(f_1) \) and \( s_2 = p_i(f_2) \) are incomparable, then there are exactly \( k-1 \) generalized \( k \)-trees which can be taken as \( t \), namely \( t = p_j(t_1 \sqcup t_2) \) for any \( j \in k \setminus \{i\} \). But for \( s_1, s_2 \) with different root labels, there are only \( k-2 \) such variants of \( t \). \( \square \)

The next fact was established in [KS07,KSZ08].

223
Lemma 7. For all \( k \geq 3 \) and \( x \in \{[ij] : i, j < k, i \neq j\} \) there is a first-order formula that defines \( x \) in \((G\mathcal{T}_k; \leq, [0], \ldots, [k - 1])\) and \((G\mathcal{F}_k; \leq, [0], \ldots, [k - 1])\).

We are ready to prove the main result of this section:

Theorem 2. For all \( k \geq 3 \) and \( i < k \), the set \( G\mathcal{T}_k \) (resp. \( G\mathcal{F}_k \)) is definable in the structure \((G\mathcal{T}_k; \leq, [0], \ldots, [k - 1])\) (resp. \((G\mathcal{T}_k; \leq, [0], \ldots, [k - 1])\)).

Proof. Again, the following formula \( \eta(t) \) works for both cases:

\[
t = [i] \lor \bigvee_{j \in k \setminus \{i\}} [\theta(t, [ij]) \lor \exists s(\theta(s, t) \land \theta(s, [ij]))].
\]

If \( \eta(t) \) is true (in some of the structures) then \( i \) is the root label of \( t \) by Lemma 6. Note that, by Lemma 7, \( \eta(t) \) is equivalent to a formula of signature \( \{\leq, [0], \ldots, [k - 1]\}. \) Assume now that \( t \) has the root label \( i \). For any \( x \in G\mathcal{F}_k \), let \( l(x) = \{i < k : [i] \leq x\} \), i.e., \( l(x) \) is the set of labels assigned to nodes of some (or, equivalently, of any) forest representing \( x \). If \( l(t) = \{i\} \) then \( t = [i] \) and \( \eta(t) \) is true. If \( \{i\} \subset l(t) \subset k \), there exists \( j \in k \setminus l(t) \). Then \( \theta(t, [ij]) \) is true, hence \( \eta(t) \) is true.

Finally, consider the case \( l(t) = k \). Let \( j, a \) be any distinct numbers from \( k \setminus \{i\} \) (here we use the requirement \( k \geq 3 \)). Let \( s \) be a generalized \( k \)-tree with the root label \( i \) which has only the labels \( i \) and \( a \), and has rank greater than the rank of \( t \) (\( s \) exists by Proposition 2; in case of the structure \((G\mathcal{T}_k; \leq, [0], \ldots, [k - 1])\), \( s \) will be a sufficiently long chain \([iaia\ldots])\). It is easy to check that \( s|t \) and \( s|[ij] \). Since \( s, t, [ij] \) have the same root label, \( \theta(s, t) \) and \( \theta(s, [ij]) \) are true. Therefore \( \eta(t) \) is also true. \( \Box \)

The last theorem immediately implies the following assertion:

Corollary 1. (i) For all \( k \geq 3 \) and \( i < k \), the set \( G\mathcal{F}_k \) (resp. \( G\mathcal{T}_k \)) is definable in the structure \((G\mathcal{F}_k; \leq, [0], \ldots, [k - 1])\) (resp \((G\mathcal{T}_k; \leq, [0], \ldots, [k - 1])\)).

(ii) Theorem 1 holds.

Proof. (i) Follows from Theorem 2 and Lemma 1.

(ii) Let \((S; \cup, p_0, \ldots, p_{k-1})\) be a dc-semilattice. It is easy to check that \( y = p_i(x) \) iff \( y \) is the smallest element of \( p_i(S) \) with \( x \leq y \). Thus, \( p_i \) is definable in (an enrichment of) \((S; \leq)\) iff \( p_i(S) \) is definable in (the enrichment of) \((S; \leq)\). Since \( G\mathcal{T}_k = p_i(G\mathcal{F}_k) \), the assertion for \( G\mathcal{F}_k \) follows from Theorem 2 and Proposition 2. The same argument applies to the tilde-case. \( \Box \)

Finally, we show that results of this paper imply the results of [KSZ08].

Corollary 2. (i) For any \( k \geq 3 \), each element of \((G\mathcal{F}_k; \leq, [0], \ldots, [k - 1])\) is first-order definable and each element of \((G\mathcal{T}_k; \leq, [0], \ldots, [k - 1])\) is \( L_{\omega_1\omega} \)-definable.

(ii) For any \( k \geq 3 \), each element of \((G\mathcal{T}_k; \leq, [0], \ldots, [k - 1])\) is first-order definable and each element of \((G\mathcal{F}_k; \leq, [0], \ldots, [k - 1])\) is \( L_{\omega_1\omega} \)-definable.
Proof. (i) Follows from Corollary 1 and the fact that each element of $G\mathcal{F}_k$ (resp. $G\tilde{\mathcal{F}}_k$) is the value of some finite (resp. countable) term of signature $\{p_0, \ldots, p_{k-1}, \sqcup, [\emptyset]\}$ without free variables.

(ii) Is checked by induction on the ranks of elements, using the term representation above. □

Remarks. 1. The last proof is shorter than the corresponding proof in [KSZ08] but the proof in [KSZ08] contains some essential additional information on the function $\emptyset$.

2. Mutatis mutandis, the results of this paper hold also for the structure $(G\mathcal{T}^0_k; \leq, [01], \ldots, [0(k-1)])$ and its tilde-version considered in [KSZ08].

References


The Relationship Between Measurability and Computability

James Lambert

The University of Bristol james.lambert@bristol.ac.uk

Abstract. Let a real number $w$ be measurable if and only if we can provide an experimenter with an allowed accuracy $\epsilon$ and a finite set of instructions and which when followed will yield an experimentally measured rational number within $\epsilon$ of $w$. I discuss the merits of Geroch and Hartle’s definition of measurability and their desideratum for future physical theories, namely that their measurable numbers be computable in the Turing sense. The thesis of this paper is that computability is not a purely mathematical property and that the computational power available to us is determined by the nature of the universe we find ourselves in. Computability should be thought of in practical terms; I argue computers are a special class experimental apparatus in that both their input and output are symbolic. Under this approach, whether a number is producible through measurement or computation becomes an epistemic distinction; computation is the use of equipment known to be reliable, measurement the observation of systems with unknown behaviour.

1 Introduction

Suppose some well verified physical theory $T$ predicts that the value of an observable quantity, say the kinetic energy of a particle, at a particular time will be a real number $w$ Joules. So far so good. Now suppose $w$ is in fact non-Turing computable, all we have is an equation describing $w$ but we do not know its decimal expansion. How might we set about testing $T$? While we may have measuring devices capable of producing numerical output pertaining to the observable in question, we have no way of telling how close the results they give are to the predicted outcome. In their 1986 paper [1] Geroch and Hartle introduce the notion of a measurable number and a formal desideratum for any future physical theory to avoid problems like this

\[
\text{... that all the measurable numbers specified by the theory be computable numbers.}
\]


With a measurable number defined as follows:

Regard a number $w$ as measurable if there exists a finite set of instructions for performing an experiment such that a technician, given an abundance of unprepared raw materials and an allowed error $\epsilon$, is able by following those instructions to perform the experiment, yielding ultimately a rational number within $\epsilon$ of $w$.

Geroch and Hartle 1986 [1] pp.542
As the authors point out this definition serves as a close analogy to their definition of computable, which is equivalent to Turing’s:

A real number \( w \) is said to be computable if there exists a (Fortran) program with a single input instruction such that, when any positive rational number \( \epsilon \) is read in, the program prints some rational number \( r \) with \( |w - r| < \epsilon \) and then halts.

Geroch and Hartle 1986 [1] pp.539 emphasis theirs

The requirement that the experimenter begin with unprepared raw materials is claimed to resemble the standard requirement that for a number to be Turing computable there must be a Turing machine that prints it beginning with a blank tape. In their own words

Measurable numbers are to be those that flow from the operation of physical laws, not from mere initial conditions. The analogous requirement, in the computable case is that all memory locations initially be blank. Otherwise, any number would be computable.


So what numbers are measurable? \( \pi \) is claimed to be measurable because the theory of continuous elastic solids combined with Euclidean geometry provides us with a clear algorithm an experimenter can follow to obtain \( \pi \) to any desired accuracy namely:

1: Build a lathe.
2: Machine a disk to appropriate precision.
3: Measure the disk’s diameter, call it \( a \).
4: Measure the disk’s circumference, call it \( b \).
5: Compute the value of \( b \) divided by \( a \).

Some observations. Firstly we can only expect this algorithm to work with arbitrary accuracy if solids really are continuous, under another physical theory such as Quantum Mechanics this algorithm would break down for sufficiently small \( \epsilon \). Measurability as currently defined is dependent both on the way the universe actually behaves, and upon our current physical theory. The claim that a number \( w \) is measurable must be prefixed with the words “According to theory \( T \) . . .". The reason for this dependence is twofold. It is a physical theory which predicts that as \( \epsilon \to 0 \) the value of \( \frac{b}{a} \) converges, this is an empirical claim underwritten by the fact that when we measure the disk’s dimensions we are examining a stable feature of reality, a fact that may or may not obtain. Also before we can even formulate an algorithm like the one above we need to presuppose something about how the world behaves, otherwise we have no reason to believe instructions such as Build a lathe can actually be carried out. For a more realistic example suppose our theory were Quantum Mechanics, the command

Build a device which given any quantum state \( |\phi\rangle|0\rangle \) reliably evolves to the state \( |\phi\rangle|\phi\rangle \).
If Quantum Mechanics is an accurate description of reality this instruction is impossible to follow; such a device is forbidden by the no-cloning theorem. (see Nielsen and Chuang 2000 [2] pp.532)

A number’s measurability is theory dependent for two reasons: First it is a theory that locates stable observable features of the world and second a theory is needed to determine what instructions an experimenter will actually be able to follow.

The 5th step of the algorithm above involves a calculation. Calculation has both physical and mathematical significance; it is a physical process carried out by man or machine that is held by the experimenter to yield a mathematical entity, a number, via that number’s numeral. In the above example the experimenter may be instructed to write the decimal expansions for the disk’s diameter and circumference on paper and perform a calculation according to the rules of long division. Alternatively he may construct a computer and use this to perform the calculation. Both methods assume certain physical processes are possible, the (rather meagre) requirement that numerals can be written on paper in the case of the former, and the considerably less trivial requirement that computing devices are possible to construct in the case of the latter. In fact an amusing paradox arises if the computer is to be built from semiconductor transistors since Quantum Mechanics provides our understanding of these materials yet contradicts Continuum Mechanics’ picture of matter as continuous. The point being that where calculation is needed it must be performed by a device/person which does not depend on the theory T being true. It is a little unclear how much of a problem this presents. On the one hand there are numerous examples of computing devices whose operation is explicable in entirely classical (non-quantum, non-relativistic) terms; early electromagnetic relay machines built by Konrad Zuse’s or those built by the Bell Telephone Company[3] in the 1930s perhaps. On the other hand, in the absence of a better understanding of the human brain it is difficult to assert outright that an experimenter can perform calculations at all under a given physical theory; some authors such as Roger Penrose[4] have suggested the brain’s operation may only be finally understood in terms of hitherto undiscovered laws of quantum gravity for instance. The upshot being that requiring the existence of an experimenter, let alone one who can compute, may itself be at odds with our chosen theory. Perhaps the only way out of this is to insist we are only interested in physical theories which support a reasonable level of computation.

So far it is clear that a number \( w \) is measurable only with respect to a certain theory \( T \) because it is \( T \) which tells us what properties of the physical world can be measured and also the set of instructions our experimenter can follow. Our choice of units is important, too; the intuitive concept of a measurable number is one that “flows” from the operation of physical laws”  [1] as a result we cannot simply assume that our experimenter begins with measuring devices since these would count as prepared materials. Allowing prepared devices would mean any number \( m \) could be measurable simply by starting with two rods, one which is \( m \) times the length of the other and using the shorter rod as a measuring rod. This would be unacceptable for Geroch and Hartle since they propose the measurable numbers of a theory ought to be computable. Allowing prepared devices means we could start our experimenter off with rods the ratio of whose lengths is any
non-computable number. Thus any theory of continuous matter compatible with the existence of measuring rods would have non-computable, measurable numbers.

In light of their position on ‘prepared materials’ Geroch and Hartle reject certain somewhat surprising numbers as non-measurable. The Earth-Moon mass ratio, for instance, is deemed non-measurable because the Earth and Moon are considered prepared materials. The proton-electron mass ratio on the other hand is regarded as measurable because protons and electrons are considered ‘raw’ materials. It is unclear how Geroch and Hartle decide which numbers (in this case ratios of masses) have “flow[ed] from the operation of physical laws” and which haven’t. Why is the Earth-Moon mass ratio any less a result of the laws of physics than the proton-electron mass ratio? If it did not arise through the action of physical law then how do they suppose it did arise? It is claimed that beginning with unprepared raw materials is analogous to a Turing machine beginning with a blank tape but this comparison needs to be more thorough. A number is computable if it is possible to program a Turing machine, starting with a blank tape to print out that number. This does not mean that if we have a Turing machine for calculating a computable number - say the number of primes between 10 and 20 - and we happen to write ‘4’ on its tape before running it, that the number 4 suddenly becomes non-computable because the Turing machine does not need the input. The same holds true in the measurability situation, if there is a finite set of instructions that tell the experimenter how to assemble the Earth and Moon from scratch then the ratio of their masses is indeed measurable because the experimenter does not need the Earth and Moon as ‘input’. The existence of such instructions requires demonstration, but we should not foreclose judging all macroscopic mass ratios as non-measurable on the grounds that they are ‘prepared’.

The measurable numbers are the numbers a physical theory tells us should be the numerical result of a certain experimental procedure specified with a finite number of instructions. Geroch and Hartle’s point is the normative claim that the measurable numbers of a physical theory should be computable in the Turing sense. This is to avoid awkward situations like the one described in the first paragraph where we attempt to verify that the result of an experiment is indeed converging to a non-computable number as \(\epsilon \to 0\). It is not clear that Geroch and Hartle’s definitions are appropriate for this.

2 Measurability And The Church-Turing Thesis

Before we proceed I would like to draw the reader’s attention to the main positions commonly referred to as The Church-Turing Thesis.

Church’s Thesis The functions that would naturally be regarded as effectively calculable are coextensive with the class of general recursive functions or those definable in \(\lambda\)-calculus.

Turing’s Thesis The functions that would naturally be regarded as effectively calculable are coextensive with the class of functions computable by a Turing machine.

Physical Turing Thesis Any physical computing device may be simulated by a universal Turing machine.
The first position is the position argued for by Alonzo Church in 1936 [5], similarly the second position is a conventional reading of Turing’s paper[6] of the same year. The final position is one advanced by neither Church nor Turing but is prominent in the literature. (see Copeland 2000 [7] for a discussion)

As I have already mentioned measurability and computability have somewhat similar definitions, this similarity becomes more striking if we adopt Turing’s original approach to the computable numbers:

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions

\textit{Alan Turing 1936 [6] pp.1}

Other authors, notably Jack Copeland[8] have emphasised that Turing’s definition of computability sought to capture the intuitive notion of what a person with pen and paper, working in a purely mechanical manner can calculate. Speaking more broadly Turing’s thesis then is about what a person working within some limitations can accomplish, from this perspective measurability and computability are very similar indeed, in fact their differences turn out to be rather small:

1. Experimenters work with many forms of raw material, Turing’s clerk works only with pen and paper.

Their similarities on the other hand:

1. Both concern what a human working in a methodical/mechanical manner can achieve.
2. Both have important finitary constraints:
   (a) finite program/instruction size
3. And idealisations:
   (a) unbounded memory/raw materials
   (b) unbounded running time in the case of Turing machines\footnote{Geroch and Hartle do not state that the experimenter has as much time as he needs, neither do they provide him with a time limit. It is hard to see how they could in any but an arbitrary way. Henceforth I will assume the experimenter, like a Turing machine, may take as much time as he needs.}
4. Both have the goal of producing numerical output.

are substantial.

In light of their similarities in physical terms it is interesting that Geroch and Hartle insist computability is a purely mathematical issue, perhaps closer to Church’s Thesis. This insistence is unusual since their demand a theory’s measurable numbers be computable is intended to ensure an experimenter can calculate the value of some predicted outcome and compare his measured value against it, this sort of calculation would suggest a partially physical reading of computability. Carrying these calculations out may well involve building a computing device of some kind, the possibility of which is contingent on sort of universe we are in. But the sort of universe we are in is exactly what
is at issue, the assumption that Turing machines (or equivalent) can be built must be proved compatible with our current physical theory in advance otherwise we may be assuming the existence of something physically impossible. And should all forms of Turing machines prove to be at odds with our current theory our experimenter (if he himself can exist) cannot compute Turing computable numbers at all.

So far the calculations we have considered are those the experimenter performs on raw data in the attempt to derive the number in question, in the above example the raw data are the diameter and circumference of the machined disk, the number in question $\pi$. But there is another purpose of calculation, namely, to confirm that the result of our measurement lies within $\epsilon$ of the expected value. Now, Geroch and Hartle are unclear on this point, their definition of measurable given above only requires that the measured value be within $\epsilon$ of the expected number $K$, not that the experimenter also know that it is.

So does the experimenter need to know that the measured value is within $\epsilon$ of the predicted value? If so the experimenter needs to be able to calculate the expansion of the predicted value and compare it with his measured value, so the measurable numbers need to be computable. If not then how is the experimenter expected to test the theory at all? If the number in question is non-computable then the best the experimenter can do is attempt to take increasingly precise measurements, if the theory is accurate and is pointing the experimenter to a stable feature of reality he can continue this process indefinitely always adding new decimal places and never changing previously obtained ones, nevertheless he will not know if they are converging, let alone to the predicted value. If on the other hand the theory is a bad one not pointing to any stable feature of reality the experimenter will, sooner or later, find that his increasingly accurate measurements yield inconsistent (divergent) results, with $\epsilon$ set to 5 significant figures he gets 5.8477 yet 6 significant figures gives 5.84763. This situation is an almost perfect parallel for explaining the non-computability of certain functions for instance consider the problem of determining whether a given Turing machine program $n$ eventually halts. To do this we simply run $n$, if it does in fact halt we will eventually find out, but if it runs forever we will never reach a stage at which we determine this. Likewise for our experimenter if our theory is bad he will eventually notice, but if it is accurate he will proceed indefinitely making increasingly precise measurements. If we do not require our experimenter to know his measurements are within $\epsilon$ of the predicted values he is in a weaker position to falsify the theory, he can only tell if his measurements fail to converge, that is the theory fails to pick out a stable feature of reality. He cannot detect the more subtle situation where the theory picks out a stable feature of reality (his sequence converges) but gets its value wrong since he doesn’t know the predicted value. We ought to insist our experimenter know the decimal (or otherwise) expansion of the theory’s predicted outcomes otherwise he is in no position to thoroughly test it.

Returning to the Church-Turing Thesis, it is clear that Geroch and Hartle want a theory’s measurable numbers to be computable so that an experimenter can confirm the validity of a theory to within a given accuracy. I do not think that Geroch and Hartle can maintain their purely mathematical interpretation of Turing’s thesis and at the same time
have computation play the important verificationary rôle in their theory because as I have claimed above what counts as computable for the experimenter is a matter contingent on the nature of the universe he is working in. In some scenario he may have access to Turing equivalent computability, but in others he may have access to a narrower class (primitive recursive for instance) or even a wider class of hypercomputational resources. Indeed Geroch and Hartle seemingly contradict their position on The Church-Turing Thesis with the following claim:

Every computable number is measurable. This is easy to see: Let the instructions direct that the raw materials be assembled into a computer, and that a certain Fortran program – one specified in the instructions – be run on that computer.

\textit{Geroch and Hartle 1986 [1] pp.544}

Which seems to take a very practical approach to computability. In this situation what would prevent us from providing the experimenter with instructions to build a hypercomputer should our theory deems them physically possible? Why should Turing computability be considered a default or ‘gold standard’?

\section{The Epistemology Of Experimental Equipment}

To review, I have argued the computational resources available to experimenters are determined by the laws of nature in their universe, these computational resources are needed to extract predictions from physical theories so they may be tested experimentally. Experimental results are produced by an experimental process described by an algorithm, this algorithm invariably involves the construction of experimental apparatus, that these devices can be built is also a fact determined by the laws of nature.

Why should we be interested in the process of setting up experiments and building devices at all? Other authors\cite{9} have shown that the mere existence of hypercomputers, at least in Newtonian, continuous matter theories, is possible in a reasonably trivial way. A memory device with an infinite number of positions, each smaller than the last so the device as a whole fits into a finite volume, may store every bit of some non-computable number. The existence of such device does not contradict the theory but how they might be constructed by humans is problematic since we would need to compute the number in question. With this in mind we ought to consider the construction of computers and measuring devices, not their mere possibility.

Thinking of the experimental process, especially the construction of experimental equipment and computers, as an algorithmic procedure is a potentially illuminating approach. Measurement devices (rulers, clocks, weight scales etc.) and computers share an important common property, namely, they are all physical objects some aspect of whose behaviour is highly reliable. For instance a ruler must be sufficiently rigid, straight etc. This reliability almost always results from the fact that we designed and built them with these features in mind.\footnote{I leave open the possibility for naturally occurring computers whose reliability has been determined by a process of natural selection, but will not discuss them further here.}
In the case of computing devices, viewing their construction as an algorithmic procedure promises to resolve the long standing debate over the conditions under which a physical system implements a computation. Typically the debate has focused on the apparently arbitrary association of logical or symbolic states to physical machine states, notably Searle[10] and Putnam[11] have argued the association is so loose that practically any physical system can be said to implement any computation rendering implementation a triviality. The design, construction and programming of a computer involves the defining of an interpretation function $F : S \rightarrow P$ from symbolic types to physical states, without this function there is little hope of an experimenter being able to use the device at all. Defining $F$ means writing down its graph which is clearly a computation by any reasonable definition of the term.

In the case of physical experiments we need to place as much trust in our experimental equipment as we do in our computers. our confidence that our devices will deliver informative results is underwritten by our confidence in the physical theories which informed their design. Again, just like computers. The position I am advocating is that computers are special kinds of experimental equipment, special because both their input and output are symbolic. In contrast a measuring device like a set of scales produces symbolic output in response to the input of a mass, a signal generator produces an audio output (for instance) from the symbolic input represented by its dials. Computers are unique in that they are devices whose input and output is symbolic.

A computing device is then any physical system which we can be used to solve equations, compute functions and so on and so forth with a high reliability. For instance with basic electronic components, a voltmeter and an ammeter and unwavering confidence in Ohm’s law ($V = IR$) I can calculate multiplication tables. Suppose $V = 10$ and I have a variable resistor. I can calculate $10 \times n$ by setting the resistor’s value to $\frac{1}{n}$ Ohms and measuring the current passing through it. But this is now beginning to look more like measurement than computation; the situation is turned on its head if instead of testing a given theory we know it to be accurate. In fact if we are convinced that our physical theory in question is an accurate description of the physical system under observation we might view this outcome as a proof of the equation’s solution. In other words we will view the physical system as a computer solving an equation.

Where does all this leave us with respect to measurability and computability? Both are means for producing numerical output, but there is an important epistemic distinction between the two. Computation only involves the use of experimental apparatus, systems whose behaviour we can anticipate. Measurement involves apparatus and a physical system whose behaviour is in some relevant sense unknown to us. The distinction between these scenarios then is the presence of some uncertainty regarding how the set up will behave. As the electronics example above demonstrated, if a system’s behaviour is well known we can use it as a piece of apparatus, a computer for instance.

The point I wish to make is the following: the difference between measurement and computation is epistemic. Computation involves a system whose physical behaviour we know about, an input we prepare and an output we are interested in. Measurement on the other hand involves us knowing a lot about the system’s initial and final states -
through the use of experimental apparatus such as measuring devices - and using these to make inferences regarding its intermediate behaviour.

How does this bear on Geroch and Hartle’s work? I would urge that the notion of computability be generalised to include any computation by any device whether Turing equivalent or otherwise. It may turn out we live in a universe where computation beyond the capabilities of Turing machines is possible, various authors have discussed the possibility for hypercomputation in specially curved spacetimes[13][14][15]. If hypercomputation were possible it would seem reasonable to include it in our class of effectively calculable functions rather than stick with the Turing computable ones. The project then has at its centre the question: If we have computers of such-and-such a power at our disposal, what limitations does that put on the physical theories we can test? The desideratum that all future physical theories satisfy a purely mathematical property, namely their measurable numbers being within the class of Turing computable numbers is the wrong way to cash out the reasonable normative claim that physical theories should be testable. Theory verification is the creation of knowledge, if we are going to discuss this process in terms of measurement and computation we need to make clear how these terms relate epistemically. The reliability of experimental apparatus provides us with two vital pieces of knowledge. Computers allow us to know exactly what the theory predicts. Measurement devices allow us to know exactly how a system behaves. Comparing these results allows us to critically test our theory.

As Geroch and Hartle’s theory stands it has a number of deficiencies, it unnecessarily anchors the verifiability of physical theories to the notion of Turing computability and, in failing to fully appreciate the extent to which our knowledge of the world informs us of what can in fact be computed, neglects the possibility of other potentially more powerful models of computation. Such models if realisable would afford us greater power in testing candidate physical theories, for instance Geroch and Hartle discuss the possibility that a more complete theory of quantum gravity may have non-Turing computable measurable numbers. If this proves to be the case the ability to build a hypercomputer would allow us to verify quantum gravity’s prediction which needless to say would be desirable. Of course were such a theory of quantum gravity to be consistently well verified there would be nothing to stop us using it as a means for designing hypercomputers of a new sort since, as I proposed above, the difference between measurement and computation lies in our confidence of the underlying theory.

References

Union Theorems in Type-2 Computation

Chung-Chih Li
School of Information Technology
Illinois State University
Normal, IL 61790, USA

Abstract. The union theorem [12] indicates that, informally, almost all natural complexity classes at type-1 such as PTIME, PSAPCE, EXP-TIME, EXPSPACE, and so on, fit the precise definition of complexity classes given by Hartmanis and Stearns in [3]. In other words, according to the theorem, the rigorous definition of complexity classes in terms of computable resource bounds is indeed broad enough to encompass most natural complexity classes. However, when we lift the computation to type-2 using oracle Turing machines, the union theorem doesn’t hold without further strengthening some necessary conditions. In [8] we prove a non-union theorem under a less considered cost model known as unit-cost model. In this paper, we examine a more popular cost model known as answer-length-cost model and give a full treatment of this powerful theorem at type-2. We prove and disprove several nontrivial variations of the union theorem based on our framework.

1 Introduction

Let \( \mathbb{N} \) be the set of natural numbers. By computable we mean Turing machine computable. A function is said to be recursive if it is total and computable. Let \( \mathcal{R} \) denote the set of recursive functions. We use \( \varphi_e \) to denote the function computed by the \( e^{th} \) Turing machine. Thus, when we say the computation of \( \varphi_e \), we refer it to the computation of the \( e^{th} \) Turing machine. Let \( \Phi_e \) denote Blum’s complexity measure [1] associated with the computation of \( \varphi_e \). Clearly, there are infinitely many different Turing machines that compute the same function, \( \varphi_e \), with different complexity. In their seminal paper [3], Hartmanis and Stearns give a precise definition of complexity classes as follows. For each \( t \in \mathcal{R} \), the complexity class \( \mathcal{C}(t) \) is defined by:

\[
\mathcal{C}(t) = \left\{ f \in \mathcal{R} \mid \exists e \left[ \varphi_e = f \text{ and } \forall x \left( \Phi_e(x) \leq t(|x|) \right) \right] \right\},
\]

where \( \forall x \) is understood as “for all but finitely many” and \( |x| \) is the length of the bit representation of \( x \in \mathbb{N} \). Despite the fact that the definition in (1) has provided a solid foundation for the study of complexity theory, we prefer to characterize computational complexity classes according to the properties of the resources bounds, not just to name a class by a single function \( t \) as shown in (1). For example, PTIME is a complexity class in which every problem can
be solved by some Turing machine within a number of steps bounded by some polynomial. In other words, "being polynomial" is the property required for the time-bounds for all problems in PTIME. Therefore, we define,

\[ \text{PTIME} = \{ f \mid f \in \text{DTIME}(p) \text{ and } p \text{ is a polynomial} \} , \]

where DTIME(p) follows the definition in (1). Thus, we could better understand PTIME as follows: PTIME = \( \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k) \). Clearly, this union gives us a more intuitive idea about what PTIME is. However, it is not at all obvious that PTIME is indeed a complexity class under the formal definition in (1). Is there a recursive function that determines exactly the same complexity class, PTIME? The same question can be asked elsewhere, e.g., the big-O notation in algorithm analysis, which can be understood as \( O(f) = \bigcup_{k \in \mathbb{N}} \text{DTIME}(k \cdot f) \). Is \( O(f) \) a rigorously defined complexity class? The powerful union theorem provides a positive answer to this kind of questions we just asked. The theorem is proven by McCreight and Meyer [12], which is the first theorem in complexity theory proven by using a priority argument with finite injuries.

**Theorem 1 (The Union Theorem [12]).** For any sequence of recursive functions \( f_0, f_1, f_2, \ldots \) such that, \( \lambda i. x. f_i(x) \) is recursive and, for all \( i, x \in \mathbb{N} \), \( f_i(x) \leq f_{i+1}(x) \), there is a recursive function \( g \) such that \( C(g) = \bigcup_{i \in \mathbb{N}} C(f_i) \).

According to the union theorem, there is \( g \in \mathcal{R} \) such that DTIME\((g) = \text{PTIME} \). Likewise, we can apply the theorem to \( O(f) \), PSPACE, EXPTIME, etc. and claim that they are indeed complexity classes. Clearly, not any arbitrary collection of resource bounds satisfied the two conditions (i) and (ii) in the union theorem. For example, there is no such uniformly effective enumeration that can cover all computable bounds. Thus, we cannot use the union theorem to argue that the class of recursive functions is a complexity class. In fact, Blum [1] proves that given any \( t \in \mathcal{R} \), there always exists a recursive function \( g \) such that, \( g \notin \text{DTIME}(t) \). The simplicity of the two premises required in the union theorem above allows us to apply the theorem to most natural complexity classes. However, we shall argue that we cannot expect the same simplicity at type-2.

The most widely studied type-2 “complexity class” is BFF\(_2\) (Basic Feasible Functional at type-2). With Cook and Kapron’s second-order polynomials [2, 4, 5], BFF\(_2\) seems to be a natural type-2 analog of PTIME. Is BFF\(_2\) a type-2 complexity class under some notion similar to (1)? Unfortunately, since there is no generally accepted machine model for type-2 complexity theory, we are not able to answer this question without a reasonable and workable framework to begin with. The framework must include the selection of computing formalism (i.e., an abstract machine such as the oracle Turing machine), the cost model for such machines, type-2 asymptotical notations, type-2 complexity measures, time bounds and a clocking scheme, and a precise definition of type-2 complexity classes. In the following section we shall give necessary terminology and notation in order to describe our union theorems at type-2. Details about our framework and concerns are discussed in [7–11].
2 Necessary Background for Type-2 Complexity Classes

We will try to keep our notation minimal due to the space constraints. Check [6] for a complete definitions of our notations and proofs. Let $\mathcal{F}$ and $\mathcal{T}$ denote the set of finite functions and total functions, respectively, over $\mathbb{N}$. With a fixed coding method for $\mathcal{F}$, we can assume that $\mathcal{F} \subseteq \mathbb{N}$ and treat any finite function as a natural number. Let $\sigma \in \mathcal{F}$. We use $\sigma \subset f$ to denote that $f$ is an extension of $\sigma$. In [8] we define $\mathbf{T}_2\mathbf{TB}$ (Type-2 Time Bounds) as a class of functions to be used as time bounds for clocking OTM (Oracle Turing Machines). OTM is considered as our formal computing device for type-2 computation. For convenience, we repeat the definition of $\mathbf{T}_2\mathbf{TB}$ in the following.

**Definition 1 (Type-2 Time Bounds)** Let $\beta : \mathcal{F} \times \mathbb{N} \to \mathbb{N}$. We say that:

1. $\beta$ is nontrivial, if for every $(\sigma, a) \in \mathcal{F} \times \mathbb{N}$, $\beta(\sigma, a) \geq |a| + 1$;
2. $\beta$ is bounded, if for every $(f, x) \in \mathcal{T} \times \mathbb{N}$, $\sigma \in \mathcal{F}$, and $\sigma \subset f$, we have $\beta(\sigma, x) \leq \lim_{\tau \to f} \beta(\tau, x)$;
3. $\beta$ is convergent, if for every $(f, a) \in \mathcal{T} \times \mathbb{N}$, there exists $\sigma \in \mathcal{F}$ with $\sigma \subset f$ such that, for all $\tau$ with $\sigma \subseteq \tau$, we have $\beta(\sigma, a) = \beta(\tau, a)$; We use $\beta(\sigma, a) \downarrow$ to denote that $\beta$ converges at $(\sigma, a)$.
4. $\beta$ is $\mathcal{F}$-monotone, if for every $a \in \mathbb{N}$ and $\sigma, \tau \in \mathcal{F}$ with $\sigma \subseteq \tau$, we have $\beta(\sigma, a) \leq \beta(\tau, a)$.

If $\beta$ is computable, nontrivial, bounded, and convergent, we say that $\beta$ is a type-2 time bound. Moreover, if $\beta$ is $\mathcal{F}$-monotone, we say that $\beta$ is strong.

With an appropriate clocking scheme, a precise notion of type-2 complexity classes can be given. Recall from the classical complexity theory, the constructibility property imposed on resource bounds guarantees a basic hierarchy among classes (see [13], pages 68, 82-85). Intuitively, a time constructible function is an efficiently computable function that is large enough to be used as a time bound for some Turing machines to operate. The classical definition of constructibility is rather intuitive and straightforward. This, however, is not the case at type-2. Much of the difficulty is caused by the cost of making oracle queries and reading the answers returned from the oracle. In other words, making queries and taking answers may use up the resource granted by the resource bound. Note that, at type-1, the union theorem has no concern about constructibility. But at type-2, without a reasonable notion of constructibility, we can use a trivial counterexample to disprove the union theorem. Moreover, under the unit-cost OTM model, a rather strong non-union theorem can be proven (Theorem 5 in [8]) where the OTM is not required to read every bit of the oracle answers. In this paper, we will emphasize on the answer-length-cost model, which is a cost model that requires the OTM to read every bit of the answer returned from the OTM. However, we have to distinguish the two models in some definitions and theorems. If it is necessary, we use OTM$^a$ ($M_e^a$) and OTM$^u$ ($M_e^u$) to denote the OTM (with index $e$) under answer-length-cost model and unit-cost model, respectively. Similarly, for a result obtained based on a certain cost model, we use an “$a$” or “$u$” in superscription to indicate the concerned model.
In the following discussion, we will give some notions that are similar to the classical notion of time constructibility. However, we hesitate to consider any of these notions a type-2 analog of time-constructibility because at the present moment it is not clear how do these notions affect the time-hierarchy at type-2; they just serve the purpose of obtaining a reasonable union theorem at type-2.

We first rule out those type-2 time bounds that are too small for any OTM to make queries. To successfully query \( f(q) \), an OTM needs at least \( |q| + 1 \) steps to place \( q \) onto the query tape, whereas an OTM needs another \( |f(q)| + 1 \) steps to read the answer, \( f(q) \). Let \( \|\text{dom}(\sigma)\| = \sum_{i \in \text{dom}(\sigma)}(|i| + 1) \). Therefore, \( \|\text{dom}(\sigma)\| \) is the minimum number of steps an OTM needs to query the entire domain of \( \sigma \). We abuse the notation by \( \|\sigma\| = \sum_{i \in \text{dom}(\sigma)}(|i| + |\sigma(i)| + 2) \). Thus, \( \|\sigma\| \) is the minimum number of steps for an OTM to query the entire domain of \( \sigma \) and read their answers. Let \( M_{e,\beta}^u \) denote the machine obtained from clocking \( M_e^u \) with \( \beta \in T_2TB \), and let \( \varphi_{e,\beta}^u \) be the functional computed by \( M_{e,\beta}^u \) (same as \( M_{e,\beta}^u \) and \( \varphi_{e,\beta}^u \)). Moreover, let \( \varphi_{e,\beta}^u(f, x) \downarrow \) denote that the computation of \( M_{e,\beta}^u(f, x) \) terminates and the value is the same as \( \varphi_{e,\beta}^u(f, x) \). In other words, \( \varphi_{e,\beta}^u(f, x) \downarrow \) means \( M_{e}^u(f, x) \) can finish its computation under \( \beta \).

**Definition 2** Let \( \beta \in T_2TB \) and \( (\sigma, a) \in \mathcal{F} \times \mathbb{N} \).

1. We say that \( (\sigma, a) \) is **\( \beta \)-queriable**, if there is \( M_{e,\beta}^u \), such that on every \( (f, a) \in T \times \mathbb{N} \) with \( \sigma \subset f \), \( M_{e,\beta}^u \) can successfully query \( \text{dom}(\sigma) \) in some order. We say that \( (\sigma, a) \) is **\( \beta \)-queriable witnessed by OTM** \( M_{e,\beta}^u \).

2. We say that \( (\sigma, a) \) is **\( \beta \)-checkable**, if there is \( M_{e,\beta}^u \), such that on every \( (f, x) \in T \times \mathbb{N} \), \( \varphi_{e,\beta}^u(f, x) \downarrow \), and

\[
\varphi_{e,\beta}^u(f, x) = \begin{cases} 1 \text{ if } \sigma \subset f \text{ and } x = a; \\ 0 \text{ otherwise.} \end{cases}
\]

We say that \( (\sigma, a) \) is **\( \beta \)-checkable witnessed by OTM** \( M_{e,\beta}^u \). \( \blacksquare \)

Since every \( \beta \in T_2TB \) must be convergent, it is clear that that not every \( (\sigma, x) \in \mathcal{F} \times \mathbb{N} \) is \( \beta \)-checkable or \( \beta \)-queriable. Suppose that \( (\sigma, a) \) is \( \beta \)-queriable witnessed by \( M_{e,\beta}^u \). Although \( M_{e,\beta}^u \) can gain budget by simply querying \( \text{dom}(\sigma) \), the budget however is based on information of \( \sigma \). Thus, not for every \( \tau \in \mathcal{F} \) with \( \text{dom}(\sigma) = \text{dom}(\tau) \), \( (\tau, a) \) is also \( \beta \)-queriable witnessed by some OTM \( M_{e,\beta}^u \). For a \( \beta \)-queriable \( (\sigma, a) \), \( \beta \) will provide enough budget for an OTM to print out \( \text{dom}(\sigma) \) in some order, but may not be enough for any OTM \( \sigma \) to do the same. If \( (\sigma, a) \) is \( \beta \)-checkable, then \( \sigma \) can be printed in some order by a \( \beta \)-clocked OTM on every \( (f, a) \) with \( \sigma \subset f \). We further define two properties in the following with which the time bounds are more useful for our purposes.

**Definition 3** Let \( \beta \in T_2TB \).

1. We say that \( \beta \) is **accessible** if and only if there is an OTM \( M_{e,\beta}^u \) such that, all minimal locking fragments of \( \beta \) are \( \beta \)-queriable witnessed by \( M_{e,\beta}^u \).

2. We say that \( \beta \) is **useful** if and only if there is an OTM \( M_{e,\beta}^u \) such that, all minimal locking fragments of \( \beta \) are \( \beta \)-checkable witnessed by \( M_{e,\beta}^u \). \( \blacksquare \)
For example, \( \beta(\sigma, a) = |a| + |\sigma(a)| + 1 \) is both accessible and useful, since it allows some OTM on input \((f, a)\) to check value of \(f(0)\), the value. Clearly, if \( \beta \) is useful, then it is also accessible. The reason we want \( \beta \) to be useful is as follows. Suppose OTM \( M_e \) can be computed under \( \beta \). If \( \beta \) is useful, then we can patch \( e \) on some finitely many \((\tau, a)\) under the same budget provided by \( \beta \) as long as \((\tau, a)\) is not a locking fragment of \( \beta \). We will see why we need this later. We say that \( \beta \) is locking detectable if there is a computable function to decide whether \( \beta \) will converge on \((\sigma, x)\). A locking detectable \( \beta \) is not necessarily useful or accessible. Also, if \( \beta \in T \subseteq TB \) is accessible, then \( \beta \) is locking detectable. One can easily verify the following two properties: (1) For every \( a \in \mathbb{N} \), \((\emptyset, a)\) is \( \beta \)-checkable. (2) For every \((\sigma, a) \in \mathcal{F} \times \mathbb{N} \), if \((\sigma, a)\) is \( \beta \)-checkable, then \( \beta(\sigma, a) \geq ||\sigma|| + |a| + 1 \).

Let \( \beta_1 \leq \beta_2 \) denote that, for every \((\sigma, x) \in \mathcal{F} \times \mathbb{N} \), \( \beta_1(\sigma, x) \leq \beta_2(\sigma, x) \). We now define the properties of a sequence of type-2 time bounds required in the union theorems.

**Definition 4** Let \( \langle \beta_i \rangle \) denote a sequence of type-2 time bounds \( \beta_0, \beta_1, \beta_2, \ldots \).

1. We say that \( \langle \beta_i \rangle \) is **uniform** if and only if \( \lambda i, \sigma, x. \beta_i(\sigma, x) \) is recursive.
2. We say that \( \langle \beta_i \rangle \) is **ascending** if and only if, for all \( i \in \mathbb{N} \), \( \beta_i \leq \beta_{i+1} \).
3. We say that \( \langle \beta_i \rangle \) is **useful** if and only if, for all \( i \in \mathbb{N} \), \( \beta_i \) is useful.
4. We say that \( \langle \beta_i \rangle \) is **convergent** if and only if, for every \((f, x) \in T \times \mathbb{N} \), there is a \( \sigma \subset f \) such that, \( \beta_i(\sigma, x) \downarrow \) for every \( i \in \mathbb{N} \).
5. We say that \( \langle \beta_i \rangle \) is **uniformly convergent** if and only if, for every \( n \in \mathbb{N} \) and \((\sigma, x) \in \mathcal{F} \times \mathbb{N} \), if \( \beta_n(\sigma, x) \downarrow \), then for all \( i \in \mathbb{N} \), \( \beta_i(\sigma, x) \downarrow \).
6. We say that \( \langle \beta_i \rangle \) is **strongly convergent** if and only if \( \langle \beta_i \rangle \) is uniformly convergent and there is a recursive locking detector for \( \beta_0 \).

Let \( \langle \beta_i \rangle \) be strongly convergent and let \( \ell \) be a locking detector for \( \beta_0 \). By definition, \( \langle \beta_i \rangle \) is uniformly convergent. Thus, we can use \( \ell \) to detect the convergence of the entire sequence. That is,

\[
[\ell(\sigma, x) = 1] \implies \forall i \in \mathbb{N} [\beta_i(\sigma, x) \downarrow],
\]

and, for all \((f, x) \in T \times \mathbb{N} \), \( \lim_{\sigma \rightarrow f} \ell(\sigma, x) = 1 \).

**Examples:** For every \( i \in \mathbb{N} \) and \((\sigma, x) \in \mathcal{F} \times \mathbb{N} \), define

\[
\alpha_i(\sigma, x) = \begin{cases} 
\sigma(x) + |x|^{i+1} + 1 & \text{if } x \in \text{dom}(\sigma); \\
|x|^{i+1} + 1 & \text{otherwise}.
\end{cases}
\]

\[
\beta_i(\sigma, x) = \begin{cases} 
\sigma(x + i) + |x|^{i+1} + 1 & \text{if } (x + i) \in \text{dom}(\sigma); \\
|x|^{i+1} + 1 & \text{otherwise}.
\end{cases}
\]

One can see that \( \langle \alpha_i \rangle \) and \( \langle \beta_i \rangle \) above are two sequences of type-2 time bounds. Clearly, \( \langle \alpha_i \rangle \) is uniform, ascending, and strongly convergent, while \( \langle \beta_i \rangle \) is uniform,
but neither ascending nor convergent. Moreover, all type-2 time bounds in \( \langle \alpha_i \rangle \) and \( \langle \beta_i \rangle \) are useful.

Let \( F_\beta \) denote the limit functional determined by \( \beta \in T_2 \text{TB} \). That is, for every \( (f, x) \in T \times N \), \( F_\beta(f, x) = \lim_{\sigma \to f} \beta(\sigma, x) \).

**Lemma 1.** Given uniform and ascending \( \langle \beta_i \rangle \), if there exists a total continuous functional \( H : T \times N \to N \) such that, for every \( i \in N \), \( F_{\beta_i} \leq H \), then \( \langle \beta_i \rangle \) is convergent.

Given any \( \beta \in T_2 \text{TB} \), define \( \langle \beta_i \rangle \) as, for each \( i \in N \), let \( \beta_i = i \beta \). It is clear that such \( \langle \beta_i \rangle \) is a counterexample of the inverse of Lemma 1. Referring to the discussing in [7], for any two continuous \( F, G : T \times N \to N \), if the set \( \{(f, x) \mid F(f, x) > G(f, x)\} \) is compact in \( T(F, G) \), we say that \( F \) is almost everywhere less than \( G \), denoted as \( F \leq^*_2 G \). If we relax \( F_{\beta_i} \leq H \) in Lemma 1 to \( F_{\beta_i} \leq^*_2 H \), then we have the following lemma which is stronger than the inverse of Lemma 1 in the sense that we do not require \( \langle \beta_i \rangle \) to be convergent.

**Lemma 2.** For any uniform and ascending \( \langle \beta_i \rangle \), there is a total continuous functional \( H : T \times N \to N \) such that, for every \( i \in N \), \( F_{\beta_i} \leq^*_2 H \).

Note that the functional \( H \) in Lemma 2 is not necessarily computable unless we can effectively determine when does each \( \beta_i \) converge. If we can, then \( H \) is computable since \( \langle \beta_i \rangle \) is uniform and, for every \( (f, x) \in T \times N \), the minimal locking fragment \( (\tau, x) \) and \( F_{\beta_i(\tau + i \beta)}(f, x) \) can be effectively obtained.

### 3 Non-union Theorems

Let \( C(\beta) \) denote the type-2 complexity class determined by \( \beta \in T_2 \text{TB} \) [8]. Similarly, let \( C(\langle \beta_i \rangle) \) denote the union class \( \bigcup_{i \in N} C(\beta_i) \). According to Theorem 2 in [8], if \( \langle \beta_i \rangle \) is ascending, then, for every \( i \in N \), \( C(\beta_i) \subseteq C(\beta_{i+1}) \). Clearly, if \( \langle \beta_i \rangle \) is strongly convergent with a locking detector \( \ell \), then each \( \beta_i \) is a strong type-2 time bound because each \( \beta_i \) can share the same locking detector \( \ell \). The strong convergence of \( \langle \beta_i \rangle \) is strong property that turns out to be one of the necessary hypotheses in our type-2 analog of the union theorem. The following theorem indicated that \( BFF_2 \) can be described by some \( \langle \beta_i \rangle \). The proof uses some results in Cook and Kapron’s [2, 4, 5].

**Theorem 1** There is a uniform and ascending \( \langle \beta_i \rangle \) such that, \( C(\langle \beta_i \rangle) = BFF_2 \).
Just as with the type-1 theory, in general, the union of two arbitrary complexity classes is not always a complexity class. We will see in the next section that some conditions are needed in order to obtain a type-2 union theorem.

**Theorem 2 (Weak Type-2 Non-union Theorem)** There exist $\beta_1 \in T_2TB$ and $\beta_2 \in T_2TB$ such that, $\forall \alpha \in T_2TB, C(\alpha) \neq C(\beta_1) \cup C(\beta_2)$.

Let $C^a(\beta)$ denotes the complexity class determined by $\beta$ under the answer-length-cost model, and $C^u(\beta)$ the complexity class under the unit-cost model. In contexts where the difference between the two models is of no importance, we then simply use $C(\beta)$.

**Theorem 3 (Type-2 Non-Union Theorem)** There is a uniform, ascending, useful, and convergent $\langle \beta_i \rangle$, such that $C^a(\langle \beta_i \rangle)$ is not a type-2 complexity class.

These negative results (non-union theorems) help us to find and justify our rather strong hypotheses for obtaining a type-2 union theorem. For example, convergence is a rather strong hypothesis, but the theorem above shows that it is not sufficient to have a union theorem. Thus, we have to further strengthen the hypothesis by including uniform convergence. Similarly, if we drop the usefulness in the hypotheses, then we can modify the proof of Theorem 3 and have the following negative result.

**Corollary 1** There is a uniform, ascending, and uniformly convergent $\langle \beta_i \rangle$, such that $C^u(\langle \beta_i \rangle)$ is not a type-2 complexity class.

Thus, the usefulness of $\langle \beta_i \rangle$ should be added as a necessary condition in our union theorem. However, it is unclear that usefulness together with uniform convergence are sufficient to obtain a type-2 union theorem.

**Conjecture 1** There is a uniform, ascending, useful, and uniformly convergent $\langle \beta_i \rangle$, such that $C^a(\langle \beta_i \rangle)$ is not a type-2 complexity class.

The following two lemmas are straightforward. We omit the proof.

**Lemma 3.** Let $\langle \beta_i \rangle$ be useful. If there is an $\alpha \in T_2TB$ such that $C^a(\langle \beta_i \rangle) = C^a(\alpha)$, then $\langle \beta_i \rangle$ is convergent.

**Lemma 4.** Let $\langle \beta_i \rangle$ be useful. If there is an $\alpha \in T_2TB$ such that $C^a(\langle \beta_i \rangle) \subseteq C^a(\alpha)$, then $\langle \beta_i \rangle$ is convergent.

If we allow $\langle \beta_i \rangle$ to be not useful, then Lemma 3, can be disproved by constructing a trivial $\langle \beta_i \rangle$. For example, let $C^a(\beta_0) = C^a(\beta_1) = \cdots$ where each $\beta_i$ delays its convergence until an inaccessible point is reached. Thus, no OTM clocked by any $\beta_i$ can query the inaccessible point. In such a way, each $\beta_i$ in the sequence determines the same complexity class and hence $C^a(\beta_0) = C^a(\langle \beta_i \rangle)$ but the convergence of $\langle \beta_i \rangle$ breaks if we choose a different inaccessible point for each $\beta_i$ to converge. Based on the discussion in this section, we have the following theorem as our conclusion.
Theorem 4 There is no \( \beta \in T_2TB \) such that, \( C(\beta) = BFF_2 \).

Using Lemma 4 we can further prove that, there is no \( \beta \in T_2TB \) such that, \( BFF_2 \subseteq C^a(\beta) \). These negative non-union results imply that a straightforward type-2 analog of the Union Theorem does not exist. In the next section we show how to strengthen the hypotheses in order to have a type-2 Union Theorem under answer-length-cost model.

4 Union Theorems

According to Lemma 3, the convergence of \( \langle \beta_i \rangle \) is a necessary condition for \( C(\langle \beta_i \rangle) \) to be a complexity class. However, Theorem 3 states that convergence together with uniformity, ascendancy, and usefulness are not sufficient to obtain a union theorem. Strong convergence turns out to be one of the necessary conditions as indicated in the following theorem. We use a priority argument with finite injuries to the theorem.

Theorem 5 (Type-2 Union Theorem) Suppose that \( \langle \beta_i \rangle \) is (i) uniform, (ii) ascending, (iii) useful, and (iv) strongly convergent. Then, there is an \( \alpha \in T_2TB \) such that, \( C^a(\alpha) = C^a(\langle \beta_i \rangle) \).

Both uniform and strong convergence are very strong conditions in the sense that, for every \( (f, x) \in T \times N \), every \( \beta_i \) has to refer to the same fragment of \( f \). At the moment, we do not see any reasonable way to get rid of this requirement of convergence. Here we discuss an unsuccessful try. We observe that the sample \( \langle \beta_i \rangle \) constructed in the proof of the Type-2 Non-Union Theorem (Theorem 3) is not bounded, i.e., \( \lim_{i \to \infty} F_{\beta_i}(f, x) = \infty \). We may ask, if \( \langle \beta_i \rangle \) is bounded by some continuous functional, can we have a union theorem without requiring \( \langle \beta_i \rangle \) to be uniformly convergent? The next corollary gives a negative result.

Corollary 2 There exist a continuous functional \( F : T \times N \to N \) and a uniform, ascending, and useful \( \langle \beta_i \rangle \) such that, for every \( i \in N \), \( F_{\beta_i} \leq F \), and \( C^a(\langle \beta_i \rangle) \) is not a type-2 complexity class.

Note that if \( \langle \beta_i \rangle \) is bounded by a total continuous functional, then, by Lemma 1, \( \langle \beta_i \rangle \) is convergent but not necessarily uniformly convergent.

Recall that a strong type-2 time bound is an \( \mathcal{F} \)-monotone one, i.e., for every \( \sigma, \tau \in \mathcal{F} \) and \( a \in N \), \( \sigma \subseteq \tau \Rightarrow \beta(\sigma, a) \leq \beta(\tau, a) \). We say that \( \langle \beta_i \rangle \) is strong if and only if every \( \beta_i \) in \( \langle \beta_i \rangle \) is \( \mathcal{F} \)-monotone. Computations clocked with such kind of time bounds have an intuitive advantage that the budget provided by the clock will never shrink during the courses of the computations. Thus, we may want the type-2 time bound \( \alpha \) constructed in the proof of the type-2 Union Theorem to be strong. However, we are strongly skeptical about this. We have the following conjecture.

Conjecture 2 There is a uniform, ascending, and strong \( \langle \beta_i \rangle \) such that, if there is \( \alpha \in T_2TB \) such that \( C^a(\alpha) = C^a(\langle \beta_i \rangle) \), then \( \alpha \) is not strong.
The Type-2 big-O Notation: The big-O notation is a key tool in algorithm analysis. A natural type-2 analog of the big-O notation can be defined as follows.

Definition 5 (Type-2 big-O Notation) Given $\beta \in T_2 T B$, define
\[
O(\beta) = \{ \varphi_e \mid \varphi_e \in C^a(c\beta + d) \text{ for some } c, d \in N \}.
\]

In fact, one of our primary motivations to have a type-2 union theorem is to examine whether $O(\beta)$ is well-defined type-2 complexity class. In our opinion, if the conditions in the our union theorem do not rule out $O(\beta)$ to be a type-2 complexity class, we should consider the conditions reasonable, no matter how strong they are. Clearly, if the $\beta$ is locking detectable, the the sequence $\beta_i$ defined in $O(\beta)$ is strongly convergent. Thus, by Theorem 5, we can prove the following corollary:

Corollary 3 Let $\beta \in T_2 T B$. If $\beta$ is locking detectable and useful, then there is an $\alpha \in T_2 T B$ such that $C^a(\alpha) = O(\beta)$.

Note that, although we have Theorem 9 in [8] asserting that there is an effective operator $\Theta_L$ such that, $\Theta_L(\beta)$ is locking detectable and equivalent to $\beta$, but
\[
[C^a(\beta) = C^a(\Theta_L(\beta))] \neq [C^a(i\beta + i) = C^a(i\Theta_L(\beta) + i)].
\]

On the other hand, if we define $\beta_i = \Theta_L(i\beta + i)$, the strong convergence of $\langle \beta_i \rangle$ may not hold. This is because, if $i \neq j$, the inaccessible points of $\beta_i$ and $\beta_j$ are different. Thus, locking detectability of $\beta$ is required in Corollary 3. We can easily prove the following two addition corollaries.

Corollary 4 Let $\alpha, \beta \in T_2 T B$. If $\alpha$ and $\beta$ are locking detectable and useful, then $O(\alpha + \beta)$ is a type-2 complexity class.

The following corollary states that we can drop the less significant term in the big-O notation. We omit the proof since it is straightforward.

Corollary 5 Let $\alpha, \beta \in T_2 T B$. Suppose that both $\alpha$ and $\beta$ are locking detectable and useful. If $\alpha \leq^* \beta$, then $O(\alpha + \beta) = O(\beta)$.

5 Conclusion

For decades type-2 complexity theory using a machine model remains an untouched territory. This paper is added to a series of our previous ones devoted to building up this theory from scratch. As the framework becomes clearer due to our specific clocking scheme for OTM and the precise definition of type-2 complexity classes, we decided to push the theory further by proving a union theorem. Based on the theorem, as its type-1 counterpart, we can characterize some intuitive complexity classes in a precise way. Unfortunately, the most familiar BFF$_2$ fails to pass the test, i.e., it is not a type-2 complexity class under
our definition. This result on the one hand indicates that our framework may not be broad enough to encompass this intuitive type-2 complexity class. On the other hand, it may provide another legitimate reason to argue that $\text{BFF}_2$ is not precise enough for further investigation on a theoretical base. The hindsight of our investigation in this paper may be that, we give a type-2 analog of the big-O notation and, according to the union theorem we proved, we can argue that it is a well-defined type-2 complexity class under our framework.

References

Two-thirds Bisimulation Indexes*

Yanfang Ma** and Yixiang Chen

Shanghai Key Laboratory of Trustworthy Computing
Software Engineering Institute, East China Normal University
Zhongshan North Road 3663, Shanghai, 200062, P.R.China
{yfma,yxchen}@sei.ecu.edu.cn

Abstract. Two-thirds bisimulation provides a kind of abstract description of an implementation (i.e. a concrete process description) with respect to a specification (i.e. an abstract process description). In order to characterize the approximate two-thirds bisimulation, we present the definition of two-thirds bisimulation index which expresses the degree to which a binary relation between processes is two-thirds bisimulation. λ−two-thirds bisimulation and its substitutivity laws are given in this paper.

Keywords: two-thirds bisimulation; two-thirds bismulation index; process calculus

1 Introduction

As one of the most important and mathematically developed models of communication and concurrency, CCS (Communication and Concurrency System) introduced by R. Milner [1–5] proposes various behavior equivalences, such as strong (weak) bisimulation equivalence, observation equivalence and so on. These equivalences are useful for relating process description at different levels of abstract.

In [6], K. G. Larsen presents two-thirds bisimulation based on probabilistic transition systems in order to characterize that two processes are undistinguished when they have the same sets of observations for all tests. In [7], Jifeng He and Tony Hoare take into account two-thirds simulation in CCS model in order to show the concepts of simulation and mutual refinement coincide with testing equivalence. In He and Hoare’s work, two-thirds simulation is a binary relation between processes that satisfies if a process \( P \) can perform an observable action, then another process \( Q \) can also perform identical action. This condition says that \( Q \) simulates \( P \). At the same time, if \( P \) can refuse a set of actions \( X \), then \( Q \)

* The work is supported by Research and Development Project of High- Technology (Grant No:2007AA01Z189), Natural Science Foundation of China (No. 90718013), Shanghai Leading Academic Discipline Project (B412), STCSM(06JC14022), PhD Program Scholarship Fund of ECNU 2008.
** The response author.
can also refuse this set. If we consider an environment as a set of some actions, then two-thirds simulation describes the communication between processes and environments. Since two-thirds simulation does not present \( P \) simulates \( Q \), so it is a relation which is strictly stronger than that of simulation (being 'half' of bisimulation) and weaker than bisimulation. In this paper, we call two-thirds simulation in CCS model two-thirds bisimulation based on Larsen’s view.

On the other hand, in the real applications of process calculus, specification and implementation are treated as two processes. If there exists a kind of behavior equivalence between them, then the program is treated as correct. So, it is the key to establish a certain behavior equivalence between specification and implementation in order to show the correctness of programs. However, as we know, computers, being physical devices, can not be assumed to behave reliably, so the implementations of a program at best approximates its specification.

In order to describes the mechanism that implementation approximates its specification, Mingsheng Ying [8–10] proposes strong (weak) bisimulation index and proves some algebraical properties. The strong (weak) bisimulation index characterizes the degree to which a binary relation between processes is strong (weak) bisimulation. However, in real application, simulation between processes appears to be more useful than bisimulation, since the latter is often too restrictive. In order to account for the relation between processes and environments, we may choose two-thirds bisimulation to verify the correctness of a program. As stated above, it is useful to describe the approximate mechanism of two-thirds bisimulation. In this paper, we propose the definition of two-thirds bisimulation indexes which characterizes the degree to which a relation is a two-thirds bisimulation.

The paper are organized as follows: in section 2, we recall some definitions and results of CCS and two-thirds bisimulation. Two-thirds bisimulation indexes and \( \lambda \)-two-thirds bisimulation are defined in section 3. Some algebraical properties are proved. In section 4, the substitutivity laws of \( \lambda \)-two-thirds bisimulation is stated. Section 5 shows our future work.

2 Preliminaries

In this section, some fundamental concepts and results of CCS and two-thirds bisimulation are showed.

First, we will give an introduction of CCS. The following contents mainly come from [1–4].

We introduce the names \( A \), the co-names \( \bar{A} \) and labels \( \Gamma = A \cup \bar{A} \). Define that \( a, b, \ldots \) range over \( A \) and \( \bar{a}, \bar{b}, \ldots \) range over \( \bar{A} \); also that \( l, l', \ldots \) range over \( \Gamma \). We also introduce the silent or perfect action \( \tau \) and define \( \text{Act} = \Gamma \cup \{\tau\} \) to the set of actions; \( \alpha, \beta \) range over \( \text{Act} \). Further, we introduce a set \( \mathcal{X} \) of processes variables and a set \( \mathcal{K} \) of processes constants. A mapping \( f : \Gamma \rightarrow \Gamma \) is called a relabeling function if \( f(\bar{l}) = \bar{f(l)} \) for every \( l \in \Gamma \). We may extend a relabeling function \( f \) to be a mapping from \( \text{Act} \) to itself by decreeing that \( f(\tau) = \bar{\tau} = \tau \).
For any subset $X$ of $\Gamma$, define $X^\perp = \{ i : l \in X \}$ and $X^\tau \defeq X \cup \{ \tau \}$. The syntax of CCS is presented in the following definition.

**Definition 1 (Process expression).** The class $\varepsilon$ of process expressions is the smallest of symbol strings satisfying the following conditions:

1. $\emptyset, K \in \varepsilon$;
2. if $\alpha \in \text{Act}$ and $E \in \varepsilon$, then $\alpha.E \in \varepsilon$;
3. if $I$ is an indexing set and $E_i \in \varepsilon (i \in I)$, then $\sum_{i \in I} E_i \in \varepsilon$;
4. if $E_1, E_2 \in \varepsilon$, then $E_1 \parallel E_2 \in \varepsilon$;
5. if $E \in \varepsilon$ and $L \subseteq \Gamma$, then $E \backslash L \in \varepsilon$;
6. if $E \in \varepsilon$ and $f$ is a relabeling function, then $E[f] \in \varepsilon$.

The process expressions without process variables are called processes and the class of processes is denoted by $\mathcal{P}$.

For any $A \in \mathcal{K}$, there is a defining equation $A \defeq P_A$ of $A$, where $P_A \in \mathcal{P}$. Constants provide us a mechanism of recursion in the process calculus.

The following is the transitional semantics of CCS which is presented in the style of G.D. Plotkin’s structural operational semantics [11].

**Definition 2 (Labeled transition system).** Let $(\varepsilon, \text{Act}, \{ \frac{\alpha}{\alpha} : \alpha \in \text{Act} \})$ be a labeled transition system, where the transition relations $\xrightarrow{\alpha}$ (\alpha \in \text{Act}) are given by the following rules:

\[
\begin{array}{c|c|c}
\text{Act} & \frac{\alpha.E \xrightarrow{\alpha} E}{\text{Sum}_j} & E_j \xrightarrow{\alpha} E_j' \\
\text{Com}_{1} & \frac{E \xrightarrow{\alpha} E'}{E \parallel F \xrightarrow{\alpha} E' \parallel F} & \frac{\sum_{i \in I} E_i \xrightarrow{\alpha} E'_j \quad (j \in I)}{\text{Com}_{2} \quad F \xrightarrow{\alpha} F'} \\
\text{Com}_{3} & \frac{E \parallel F \xrightarrow{\alpha} E' \parallel F'} {E \parallel F \xrightarrow{\alpha} E'} & \frac{\text{Res} \quad E \xrightarrow{\alpha} E' \quad E \backslash L \xrightarrow{\alpha} E \backslash L} {(\alpha, \bar{\alpha} \notin L)} \\
\text{Rel} & \frac{E \xrightarrow{\alpha} E'} {E[f] \xrightarrow{f(\alpha)} E'[f]} & \frac{\text{Con} \quad P \xrightarrow{\alpha} P'} {A \xrightarrow{\alpha} P'} \quad (A \defeq P) \\
\end{array}
\]

Let $l \in \Gamma$, we write $E \xrightarrow{l} E'$ if $E \xrightarrow{\tau^*} l \xrightarrow{\tau^*} E'$, where $\tau^*$ is the reflexive and transitive closure of $\tau$.

In next sections, we mainly consider the restriction $(\mathcal{P}, \text{Act}, \{ \frac{\alpha}{\alpha} | P : \alpha \in \text{Act} \})$ of $(\varepsilon, \text{Act}, \{ \frac{\alpha}{\alpha} : \alpha \in \text{Act} \})$ on $\mathcal{P}$, where for each $\alpha \in \text{Act}$, $\frac{\alpha}{\alpha} | P = \frac{\alpha}{\alpha} \cap (\mathcal{P} \times \mathcal{P})$ is the restriction of $\frac{\alpha}{\alpha}$ on $\mathcal{P}$. For simplicity, we always write $\xrightarrow{\alpha}$ for $\frac{\alpha}{\alpha} | P$.

Next, we will review the definition of two-thirds bisimulation and some algebraical properties. In the paper [7], $\text{init}(P)$ is considered as the set of actions in which $P$ can engage at the very beginning,

$\text{init}(P) \defeq \{ \alpha : \text{there exists } P' \text{ such that } P \xrightarrow{\alpha} P' \}$. 

248
Lemma 1. [7]

1. \(\text{init}(\alpha.P) = \{\alpha\}\);
2. \(\text{init}(\sum_{i \in I} P_i) = \bigcup_{i \in I} \text{init}(P_i)\);
3. \(\text{init}(P | Q) = \text{init}(P) \cup \text{init}(Q) \cup \{\tau : \text{there exists } l \in \Gamma \text{ such that } l \in \text{init}(P) \text{ and } \bar{l} \in \text{init}(Q)\}\);
4. \(\text{init}(P \setminus L) = \text{init}(P) \setminus (L \cup \bar{L})\);
5. \(\text{init}(P[f]) = \{f(\alpha) : \alpha \in \text{init}(P)\} = f(\text{init}(P))\);
6. \(\text{init}(A) = \text{init}(P_A) \text{ if } A \overset{\text{def}}{=} P_A\).

Definition 3 (Stability). [7] A process \(P\) is stable if \(\tau \notin \text{init}(P)\).

Definition 4 (Refusal). [7] Let \(X\) be a subset of \(\Gamma\). \(\tau^*\) is the reflexive and transitive closure of \(\tau\). Then \(P\) can refuse \(X\) if there exists \(P'\) such that

\[P \xrightarrow{\tau^*} P' \land \text{init}(P') \cap \bar{X}^\tau = \emptyset.\]

Theorem 1. [7]

1. \(\sum_{i \in I} l_i P_i\) can refuse \(X\) if and only if \(\bar{l}_i \notin X\) for all \(i \in I\);
2. \(\sum_{i \in I} l_i P_i + \sum_{j \in J} \tau Q_j\) can refuse \(X\) if and only if \(Q_j\) can refuse \(X\) for some \(j \in J\);
3. \(P \setminus L\) can refuse \(X\) if and only if \(P\) can refuse \(X \setminus (L \cup \bar{L})\);
4. \(P[f]\) can refuse \(X\) if and only if \(P\) can refuse \(f^{-1}(X)\);
5. Let \(A \overset{\text{def}}{=} P_A\). \(A\) can refuse \(X\) if and only if \(P_A\) does so.

Definition 5 (Two-thirds bisimulation). [7] Let \((P, \text{Act}, \{\alpha : \alpha \in \text{Act}\})\) be a labeled transition system. \(S \subseteq P \times P\) is a binary relation over \(P\). Then \(S\) is called a two-thirds bisimulation if, whenever \(PSQ, l \in \Gamma\),

1. if \(P \xrightarrow{l} P'\), then there exists \(Q \in P\) such that \(Q \xrightarrow{l} Q'\) and \(P'SQ'\);
2. if \(P\) can refuse \(X\), so can \(Q\).

For example, there are two vending machine for selling Coca-Cola, they can be described in the CCS model as follows:

- \(C = \text{1d.CocaCola.collect}\)
- \(C' = \text{1d.CocaCola.collect} + \tau.\text{1d.CocaCola.collect}\)

The machine \(C\) means to buy a cup of Coca-Cola from machine \(C\), you must put in $1, press the button marked ”Coca-Cola” on machine \(C\), and collect your cup of Coca-Cola from the tray. The machine \(C'\) means that the machine either
works normally or makes internal back-check and afterward work normally. We notice that \( C' \) simulates \( C \). At the same time, if a person who can be considered as an external environment provides the set of actions is \( X \) and \( C \) can refuse \( X \), i.e., \( X^* \cap \text{init}(C) = \emptyset \), then according to the definition of refusal, we can get that \( C' \) can also refuse \( X \). So, \( C' \) two-thirds bisimulates \( C \).

We define

\[
\leq_{\frac{2}{3}} = \bigcup \{ S : S \text{ is two-thirds bisimulation} \}.
\]

Then \( \leq_{\frac{2}{3}} \) is the greatest two-thirds bisimulation. If there exists a two-thirds bisimulation \( S \) such that \( PSQ \), then we say \( Q \) two-thirds bisimulates \( P \), denoted by \( P \leq_{\frac{2}{3}} Q \).

3 Two-thirds Bisimulation Indexes

In this section, in order to discuss the approximate version of two-thirds bisimulation, we present the definition of two-thirds bisimulation index which indicates the degree to which a binary relation between processes is two-thirds bisimulation. We also generalize some algebraical properties of two-thirds bisimulation based on two-thirds bisimulation index.

3.1 Two-thirds Bisimulation Index

It is well known that the conditions of matching the identical observable actions and the same refusal set of actions are very rigorous for two-thirds bisimulation. In this subsection, we will develop the approximate version of two-thirds bisimulation which is useful to analyze and understand two-thirds bisimulation.

We first presume there is a metric on actions which describes a certain distance between actions. From the presumed metric on actions, two-thirds bisimulation index of processes is induced. Based on the concept of two-thirds bisimulation index, the notion of \( \lambda \)--two-thirds bisimulation is introduced.

**Definition 6 (Metric space).** [12] Let \( M \) be a nonempty set. \( \rho \) is a mapping from \( M \times M \) into \([0, \infty]\). Then the pair \( (M, \rho) \) is called a metric space if the following conditions are satisfied:

1. \( \rho(x, y) = 0 \) if and only if \( x = y \);
2. \( \rho(x, y) = \rho(y, x) \); and
3. \( \rho(x, z) \leq \rho(x, y) + \rho(y, z) \) for any \( x, y, z \in M \).

If 1. is weaken by 1': \( \rho(x, x) = 0 \) for each \( x \in M \), then \( \rho \) is called a pseudometric, and if 3. is strengthened by 3': \( \rho(x, z) \leq \max\{\rho(x, y), \rho(y, z)\} \) for any \( x, y, z \in M \), then \( \rho \) is called an ultrametric. \( \square \)

Let \( \rho \) be a metric on \( \mathcal{A} \). Naturally, we can extend \( \rho \) to be a mapping from \( \text{Act} \times \text{Act} \) to \([0, \infty]\), denoted by \( \rho_{\text{act}} \) in the following way: \( \forall a, b \in \mathcal{A}, \)

1. \( \rho_{\text{act}}(\tau, \tau) = 0, \)
2. \( \rho_{\text{act}}(a, b) = \rho_{\text{act}}(\bar{a}, \bar{b}) = \rho(a, b), \)
3. \( \rho_{\text{act}}(\bar{a}, \bar{b}) = \rho_{\text{act}}(a, \tau) = \rho_{\text{act}}(\bar{a}, \tau) = \rho_{\text{act}}(a, \bar{b}) = \rho_{\text{act}}(\bar{a}, b) = \infty. \)

It is easy to see that \( \rho_{\text{act}} \) is a metric on \( \text{Act} \). And \( \rho_{\text{act}} \) is also an ultrametric provided \( \rho \) is an ultrametric. For simplicity, we always write \( \rho \) for \( \rho_{\text{act}} \). The following is a numerical generalization of Definition 5.

**Definition 7 (Two-thirds bisimulation index).** Let \( (\mathcal{P}, \text{Act}, \{ \alpha : \alpha \in \text{Act} \}) \) be a labeled transition system. \( \rho \) is a metric on \( \text{Act} \), and \( S \subseteq \mathcal{P} \times \mathcal{P} \). We write

\[
\text{STTb}_S(Q, P'; l) = \inf\{ \rho(l, k) : k \in \Gamma \text{ and there is } Q' \in \mathcal{P} \text{ such that } Q \xrightarrow{k} Q' \text{ and } P'SQ' \},
\]

and

\[
\text{STTb}_S(P, Q) = \sup\{ \text{STTb}_S(Q, P'; l) : P' \in \mathcal{P} \text{ and } l \in \Gamma \text{ with } P \xrightarrow{l} P' \},
\]

\[
\text{RTTb}_S(Q; l) = \inf\{ \rho(l, k) : \text{there is } k \in \Gamma \text{ such that } Q \text{ can refuse } k \},
\]

\[
\text{RTTb}_S(Q; X) = \sup\{ \text{RTTb}_S(Q; l) : l \in X \},
\]

\[
\text{RTTb}_S(P, Q) = \sup\{ \text{RTTb}_S(Q; X) : P \text{ can refuse } X, X \subseteq \Gamma \},
\]

and

\[
\text{TTb}_S = \sup\{ \max\{ \text{STTb}_S(P, Q), \text{RTTb}_S(P, Q) \} : PSQ \},
\]

then \( \text{TTb}_S \) is called two-thirds bisimulation index of \( S \).

Intuitively, if \( P, P', Q \in \mathcal{P} \) and \( l \in \Gamma \) are given, \( PSQ \) and \( P \xrightarrow{l} P' \), then \( \text{STTb}_S(Q, P'; l) \) is the infinitum of between transitions \( l \) and \( k \), where \( k \) are transitions that can be used to complete the diagram of \( Q \xrightarrow{k} Q' \) and \( P'SQ' \). \( \text{STTb}_S(P, Q) \) expresses the degree to which \( Q \) simulates \( P \). If \( P \) can refuse \( X \), for any \( l \in X \), then \( \text{RTTb}_S(Q; l) \) is the infinitum of between actions \( l \) and \( k \), where \( k \) are the actions that can be refused by \( Q \). So \( \text{RTTb}_S(P, Q) \) expresses the degree to which the set of actions which \( P \) can refuse can be refused by \( Q \). So, \( \text{TTb}_S \) expresses the degree to which \( S \) is a two-thirds bisimulation. From the definition, we can point out that the smaller the value of \( \text{TTb}_S \), the higher the degree to which \( S \) is two-thirds bisimulation.

Some basic properties of two-thirds bisimulation index are presented in the following.

**Proposition 1.** 1. If \( S \) is two-thirds bisimulation, then \( \text{TTb}_S = 0 \). Especially, \( \text{TTb}_{1d_S} = 0 \)
2. \( \text{TTb}_{S_1 \circ S_2} \leq \text{TTb}_{S_1} + \text{TTb}_{S_2} \). In particular, if \( \rho \) is ultrametric, then \( \text{TTb}_{S_1 \circ S_2} \leq \max(\text{TTb}_{S_1}, \text{TTb}_{S_2}) \).
3. \( \text{TTb}_{\bigcup_{i \in I} S_i} \leq \sup_{i \in I} \text{TTb}_{S_i} \)

\( \square \)
The first proposition in Proposition 1 tells us that the bisimulation index of a two-thirds bisimulation is 0. The second property indicates that the two-thirds bisimulation index of the composition of two relations does not exceed the sum of the bisimulation indexes of the relation and if the presumed metric on actions is an ultrametric, then it is not greater than the greatest of the two-thirds bisimulation indexes of the factor relations. The final proposition says that if the two-thirds bisimulation indexes of $S_i$ is not less than some value for all $i \in I$, then the two-thirds bisimulation index of $\bigcup_{i \in I} S_i$ is also not less than this value.

A two-thirds bisimulation has an index 0, but it is not true that each relation whose two-thirds bisimulation index is 0 is a two-thirds bisimulation. The following example shows this point.

**Example 1.** Let $T$ be real line with usual (Euclidean) metric. $\rho(x, y) = |x - y|$ for $x, y \in T$. For every $x \in T$, $\bar{x} = x$. And let $\mathcal{P} = \{x, y\}$, $x \xrightarrow{t} x$ for each rational number $t$, $y \xrightarrow{t'} y$ for irrational number $t'$ and $S = \{(x, y)\}$. $X = \{t : t$ is irrational number$\}$ is the greatest set which $x$ can refuse, and $Y = \{t : t$ is rational number$\}$ is the greatest set which $y$ can refuse. Then $STb_S(x, y) = 0$, $RTTb_S(x, y) = 0$. So $TTb_S = 0$. But, $S$ is not two-thirds bisimulation according to the definition of two-thirds bisimulation.

From the definition of two-thirds bisimulation index, we notice that for any relation $S$ over $\mathcal{P}$, we can get a two-thirds bisimulation index. So, we can stratify all relations between processes $\mathcal{P}$ according to their bisimulation indexes.

**Definition 8 (λ—two-thirds bisimulation).** Let $(\mathcal{P}, \text{Act}, \{\xrightarrow{\alpha} : \alpha \in \text{Act}\})$ be a labeled transition systems, $S \subseteq \mathcal{P} \times \mathcal{P}, \lambda \in [0, \infty)$. If $TTb_S \leq \lambda$, then $S$ is called a λ—two-thirds bisimulation.

It is easy to see that any two-thirds bisimulation $S$ is $\infty—two-thirds$ bisimulation. If $\lambda_1 \leq \lambda_2$, $S$ is $\lambda_1—two-thirds$ bisimulation, then $S$ is also $\lambda_2—two-thirds$ bisimulation. If $S$ is a $\lambda_i—two-thirds$ bisimulation, then $S$ is a $\inf_{i \in I} \lambda_i—two-thirds$ bisimulation.

**Corollary 1.**
1. If $S$ is two-thirds bisimulation, then $S$ is 0—two-thirds bisimulation.
2. If $S_i$ is a $\lambda_i—two-thirds$ bisimulation, then $S_1 \circ S_2$ is a $\lambda_1 + \lambda_2—two-thirds$ bisimulation. In particular, if $\rho$ is an ultrametric and $S_1, S_2$ are both $\lambda—two-thirds$ bisimulation, so is $S_1 \circ S_2$.
3. If $S_i$ is $\lambda—two-thirds$ bisimulation, so is $\bigcup_{i \in I} S_i$.

The following definition is useful for us to verify a relation whether it is a $\lambda—two-thirds$ bisimulation.

**Definition 9 ($\leq_{\frac{2}{\lambda}}$).** We define

$$\leq_{\frac{2}{\lambda}} = \bigcup\{S : S \text{ is } \lambda—two-thirds \text{ bisimulation}\}.$$
If there exists a binary relation \( S \subseteq P \times P \) is \( \lambda \)-two-thirds bisimulation such that \( PSQ \), then we write \( P \leq _\lambda ^{\frac{2}{3}} Q \).

Example 2. Let \( S = \{ s_1, s_2, s_3, s_4 \} \), \( T \) be the real line with the Euclidean metric, and \( R = \{ (s_1, s_3), (s_2, s_4) \} \). We define the transitions: \( \xrightarrow{t} = \{ (s_1, s_2), (s_4, s_3) \} \) if \( t \) is a rational number and \( \xrightarrow{t} = \{ (s_2, s_1), (s_3, s_4) \} \) if \( t \) is an irrational number.

Let \( X = \{ t : t \) is irrational number\( \} \) and \( Y = \{ t : t \) is rational number\( \} \).

Then \( s_1, s_4 \) can both refuse \( X \), and \( s_2, s_3 \) can both refuse the set \( Y \). By a routine calculation, we can get \( STtb_R(s_1, s_3) = 0 \), \( RTTb_R(s_1, s_3) = 0 \). Similarly, \( STtb_R(s_2, s_4) = 0 \), \( RTTb_R(s_2, s_4) = 0 \). Furthermore, \( TTb_R = 0 \). Thus \( R \) is a 0-two-thirds bisimulation and \( s_1 \leq _0 ^{\frac{2}{3}} s_3 \) and \( s_2 \leq _0 ^{\frac{2}{3}} s_4 \).

Proposition 2. 1. \( \leq ^{\frac{2}{3}} \subseteq \leq _0 ^{\frac{2}{3}} \), and if \( \rho \) is discrete, i.e., for any \( \alpha, \beta \in Act \) with \( \alpha \neq \beta \), there exists \( \lambda > \rho(\alpha, \beta) \geq \lambda \), then \( \leq _\lambda ^{\frac{2}{3}} \leq _\lambda ^{\frac{2}{3}} \leq _\lambda ^{\frac{2}{3}} P \times P \). If \( \lambda_1 \leq \lambda_2 \), then \( \leq _{\lambda_1} ^{\frac{2}{3}} \subseteq \leq _{\lambda_2} ^{\frac{2}{3}} \).

2. For any \( \lambda \in [0, \infty] \), \( \leq _{\lambda} ^{\frac{2}{3}} \) is \( \lambda \)-two-thirds bisimulation, it is reflexive and \( \leq _{\lambda_1} ^{\frac{2}{3}} \circ \leq _{\lambda_2} ^{\frac{2}{3}} \subseteq \leq _{\lambda_1 + \lambda_2} ^{\frac{2}{3}} \).

The first proposition of Proposition 2 indicates that if a binary relation is two-thirds bisimulation, then the relation must be 0-two-thirds bisimulation. If \( \rho \) is discrete, then two-thirds bisimulation and 0-two-thirds bisimulation are identical; any binary relation between processes and \( \infty \)-two-thirds bisimulation are same. At the same time, if we consider \( \leq _{\lambda} ^{\frac{2}{3}} \) as a function about the parameter \( \lambda \), then \( \leq _{\lambda} ^{\frac{2}{3}} \) is monotonic. The second property says that \( \leq _{\lambda} ^{\frac{2}{3}} \) is \( \lambda \)-two-thirds bisimulation relation between processes.

Proposition 3. Let \( P, Q \in P, \lambda \in [0, \infty] \). Then \( P \leq _{\lambda} ^{\frac{2}{3}} Q \) if and only if \( STtb _{\leq _{\lambda} ^{\frac{2}{3}}} (P, Q) \leq \lambda \) and \( RTTb _{\leq _{\lambda} ^{\frac{2}{3}}} (P, Q) \leq \lambda \).

This proposition provides a recursive characterization of \( \leq _{\lambda} ^{\frac{2}{3}} \). It is more useful for us to prove whether two processes belong to some \( \lambda \)-two-thirds bisimulation.

4 The Substitutivity Laws of \( \lambda \)-two-thirds Bisimulation

When equivalences are discussed, much care is taken to ensure that the equivalences are congruent with respect to the processes combinators in order to support hierarchic development and modular decomposition. It means that if two processes are equivalent, then the new processes which are obtained by combining the given processes are also equivalent. In this section, we will mainly discuss this substitutivity laws of \( \lambda \)-two-thirds bisimulation under various combinators. At first, we need to introduce some definitions.
Definition 10 (\(\lambda\)-round ). [10] Let \((M, \rho)\) be a metric space, \(Y \subseteq M\), \(\lambda \geq 0\). If for any \(x, y \in M\), \(x \in Y\), and \(\rho(x, y) \leq \lambda\) implies \(y \in Y\), then \(Y\) is said to be \(\lambda\)-round. If for some \(\mu > \lambda\), \(Y\) is \(\mu\)-round, then \(Y\) is said to be strongly \(\lambda\)-round.

Definition 11 (Nonexpansive). [10] Let \((M, \rho)\) be a metric space. If for any \(x, y \in M\), \(\rho(f(x), f(y)) \leq \rho(x, y)\), then \(f\) is said to be nonexpansive.

From the definition above, we can see that \(\lambda\)-round is a quite rigorous condition. In paper [10], the author has told us there are only two \(\lambda\)-round sets in the real line when \(\lambda > 0\). One is the empty set and the other is the real line itself. At the same time, he also tells us it is not same case as in the real line in general. For example, if \(M = \bigcup_{i \in I} M_i\) and \(\{M_i\}_{i \in I}\) is pairwise disjoint, and \(\rho(x, y) > \mu\) for any \(x, y \in M\) with \(x \in M_i\) and \(y \in M_j\) \((i \neq j)\), then each \(M_i\) is \(\lambda\)-round for every \(\lambda \leq \mu\).

Next, we consider the substitutivity laws of \(\lambda\)-two-thirds bisimulation under various combinators.

Proposition 4. Let \(P_1 \leq_{\frac{2}{3}} \lambda P_2\), \(\alpha \in \text{Act}, L \subseteq \Gamma\). Then

1. \(\alpha.P_1 \leq_{\frac{2}{3}} \lambda \alpha.P_1\).
2. \(P_1 \setminus L \leq_{\frac{2}{3}} \lambda P_2 \setminus L\) if \(L\) is \(\lambda\)-round.
3. \(P_1[f] \leq_{\frac{2}{3}} \lambda P_2[f]\) if \(f\) is nonexpansive.
4. Let \(P_{1i} \leq_{\frac{2}{3}} \lambda \lambda P_{2i}\). Then \(\sum_{i \in I} P_{1i} \leq_{\frac{2}{3}} \max_{i \in I} \lambda \sum_{i \in I} P_{2i}\).

This proposition states that \(\lambda\)-two-thirds bisimulation is preserved by all combinators except Composition. For the case of Restriction, \(L\) is assumed to be \(\lambda\)-round; for the Relabeling, the relabeling function \(f\) is required to be nonexpansive. Now we have no idea about the Composition, it will be considered in our future work.

5 Future Prospects

In this paper, we mainly discuss the approximate version of two-thirds bisimulation. It describes the approximate correctness of processes which is related by two-thirds bisimulation. We present the definition of \(\lambda\)-two-thirds bisimulation which generalizes the two-thirds bisimulation. The substitutivity laws of \(\lambda\)-two-thirds bisimulation are discussed. We will try to find the substitutivity property of Composition in the future. At the same time, we notice that the refusal set can be considered as an environment, but this environment is static. Next, we will focus on describing the approximate correctness with a dynamic environment.
References

An Implementation of Exact Real Number Computation Based on LRT

J. Raymundo Marcial-Romero\textsuperscript{1}, Claudia Zepeda\textsuperscript{2}, J. A. Hernández\textsuperscript{1}, Héctor A. Montes-Venegas\textsuperscript{1}, and David Linares\textsuperscript{1}

\textsuperscript{1} Universidad Autónoma del Estado de México (UAEM)
Facultad de Ingeniería
rmarcial,xosehernandez,h.a.montes,davidlicar@fi.uaemex.mx
\textsuperscript{2} Benemérita Universidad Autónoma de Puebla (BUAP)
Facultad de Computación czepedac@gmail.com

Abstract. As one of the first theoretical programming languages for exact real number computation, Real PCF showed to be impractical due to the parallel construct needed for even basic operations. In this sense, Marcial et al. [9,8] introduced a variant of Real PCF avoiding the parallelism by introducing a non-deterministic constructor into the language. This language is called LRT. Marcial et al. [11, 12] showed that all computable first order functions (in Brattka sense [3]) are defined in the language. In this paper we present an implementation of LRT and a calculator for first order functions based on Plune's [15] algorithms.

Key words: Exact real-number computation; Sequential Computation; PCF

1 Introduction

In the last two decades, several researches have presented different frameworks for a programming language for exact real number computation [16, 4, 2, 17]. Particularly, Escardó [4] proposed a theoretical programming language for exact real number computation with an abstract data type (representation independent) but a parallel constructor, which is needed even for basic operation like addition. A further research project was to develop a theoretical programming language avoiding parallel constructors since their computational cost (time and storage) is high. Marcial et al. [8, 9] presented a sequential non-deterministic programming language for exact real number computations called LRT. LRT can be seen as Real PCF without the parallel constructor and a non-deterministic constructor added. The non-determinism allowed to avoid the parallelism; a further explanation can be consulted in [10]. Additionally, the non-determinism allows to define relations in LRT, therefore Marcial et al. [11, 12] established computational adequacy between LRT and Brattka relational setting [3]. The relational setting of LRT, implicitly defines computable first order functions, thus they can be implemented in the language.
In his paper Bauer [1] refers to the fact that a direction in constructive mathematics is "get closer to the practice" without disconnecting the theory and the practice. Even more, he stated:

"move practice closer to theory by making sure that practical implementations follow formal specifications that are computed directly from theoretical models".

In this paper we present an implementation of the LRT operational semantics. The base language used to implement LRT was Haskell, which include characteristics to easily implement languages like LRT. Among the main characteristics of Haskell are the lazy evaluation and the natural use of infinite lists. The algorithms implemented are based on Plume’s thesis [15] which converge faster than the correct algorithms proposed in [8].

We are aware that other even faster implementations for exact real-number computation have been presented [6, 1, 5, 7, 13], however our motivation comes from Bauer [1] suggestions.

The paper is organized as follows: in section 2 the language LRT is described. In section 3, the main implementation details are presented. In section 4 a graphical user interface for a calculator is shown. Finally the conclusions are established.

2 The LRT Language

We introduce the LRT language, which amounts to the language considered by Escardó [4] with the parallel conditional removed and a constant rtest_{l,r} added. This is a call-by-name language. Because real-number computations are infinite, and there are no canonical forms for partial real-number computations, it is not clear what a call-by-value operational semantics ought to be.

2.1 Syntax

The language LRT is an extension of PCF with a ground type for real numbers and suitable primitive functions for real-number computation. Its raw syntax is given by

\[
x \in \text{Variable},
\]
\[
t ::= \text{nat} \mid \text{bool} \mid I \mid t \to t,
\]
\[
P ::= x \mid n \mid \text{true} \mid \text{false} \mid (+1)(P) \mid (-1)(P) \mid (=0)(P) \mid \text{if } P \text{ then } P \text{ else } P \mid \text{cons}_{[a,b]}(P) \mid \text{tail}_{[a,b]}(P) \mid \text{rtest}_{l,r}(P) \mid \lambda x : t.P \mid PP \mid YP,
\]

where the subscripts of the constructs cons, tail are rational intervals (sometimes written as a or [a,b]) and those of rtest are rational numbers. Terms of ground type I are intended to compute real numbers in the unit interval.
Because the intention of this paper is not to present the denotational semantics of the language which is based on powerdomains [10], we just present the mathematical objects which describe the \texttt{cons}, \texttt{tail} and \texttt{rtest} constructors. The others are the well known PCF constructors.

The function \texttt{cons}_a : [-1, 1] → [-1, 1] is the unique increasing affine map with image the interval \( a, i.e., \)

\[
\texttt{cons}_{[\underline{a}, \overline{a}]}(x) = \frac{\overline{a} - a}{2} x + \frac{\overline{a} + a}{2}
\]

That is, rescale and translate the interval \([-1, 1]\) so that it becomes \([a, \overline{a}]\), and define \( \texttt{cons}_{[\underline{a}, \overline{a}]}(x) \) to be the interval which results from applying the same rescaling and translation to \( x \).

The function \texttt{tail}_{[\underline{a}, \overline{a}]}(x) : [-1, 1] → [-1, 1] is a left inverse, i.e.

\[
\texttt{tail}_{[\underline{a}, \overline{a}]}(\texttt{cons}_a(x)) = x.
\]

More precisely, the following left inverse is taken, where \( \kappa_a = \overline{a} - a \) and \( \tau_a = \overline{a} + a \):

\[
\texttt{tail}_{[\underline{a}, \overline{a}]}(x) = \max(-1, \min((2x - \tau_a)/\kappa_a, 1)).
\]

The definition of the function \texttt{rtest}_{l,r} : [-1, 1] → \{true, false\}, where \( l < r \) are rational numbers, can be formulated as

\[
\texttt{rtest}_{l,r}(x) = \begin{cases} 
\text{true}, & \text{if } x \in (-\infty, l], \\
\text{true} \cup \text{false}, & \text{if } x \in (l, r), \\
\text{false}, & \text{if } x \in [r, \infty). 
\end{cases}
\]

The function \texttt{rtest}_{l,r} is operationally computable because, for any argument \( x \) given intensionally as a shrinking sequence of intervals, the computational rules systematically establish one of the semidecidable conditions \( a < x \) and \( x < b \) where \( a, b \) are rational numbers.

### 2.2 Operational Semantics

We consider a small-step style operational semantics for our language. We define the one-step reduction relation \( \rightarrow \) to be the least relation containing the one-step reduction rules for evaluation of PCF [14] together with those given below.

We first need some preliminaries. For intervals \( a \) and \( b \) in \( I \), we define

\[
ab = \texttt{cons}_a(b),
\]

where \texttt{cons} is the extension to the interval domain of the function defined previously. This operation is associative, and has the bottom element of \( I \) as its neutral element [4]:

\[
(ab)c = a(bc), \quad a \bot = \bot a = a.
\]
In the interval domain, \( a \subseteq b \) iff \( b \subseteq a \). Moreover,
\[
a \subseteq b \iff \exists c \in \mathcal{I}. \ ac = b,
\]
and this \( c \) is unique if \( a \) has non-zero length, i.e. it is not maximal, and in this case we denote \( c \) by
\[
b \setminus a.
\]
For intervals \( a \) and \( b \), we define
\[
a \leq b \iff \underbar{a} \leq \underbar{b}
\]
and
\[
a \uparrow b \iff \exists c. \ a \subseteq c \text{ and } b \subseteq c.
\]
With this notation, the rules for Real PCF as defined in [4] are:

1. \( \text{cons}_a(\text{cons}_b M) \rightarrow \text{cons}_{ab} M \)
2. \( \text{cons}_a M \rightarrow \text{cons}_a M' \)
3. \( \text{tail}_a(\text{cons}_b M) \rightarrow \text{Y cons}_{[-1,0]} \) if \( b \leq a \)
4. \( \text{tail}_a(\text{cons}_b M) \rightarrow \text{Y cons}_{[0,1]} \) if \( b \geq a \)
5. \( \text{tail}_a(\text{cons}_b M) \rightarrow \text{cons}_{\emptyset \setminus a} M \) if \( a \subseteq b \) and \( a \neq b \)
6. \( \text{tail}_a(M) \rightarrow \text{tail}_a(M') \)
7. if true \( M \ N \rightarrow M \)
8. if false \( M \ N \rightarrow N \)
9. if \( M \ N_1 \ N_2 \rightarrow M' \ N_1 \ N_2 \)

For our language \( LRT \), we add:

11. \( \text{rtest}_{l,r}(\text{cons}_a M) \rightarrow \text{true} \) if \( \overline{a} < r \)
12. \( \text{rtest}_{l,r}(\text{cons}_a M) \rightarrow \text{false} \) if \( l < \underline{a} \)
13. \( \text{rtest}_{l,r} M \rightarrow \text{rtest}_{l,r} M' \) if \( M \rightarrow M' \).

Remark

1. Rule (1) plays a crucial role and amounts to the associativity law. The idea is that both \( a \) and \( b \) give partial information about a real number, and \( ab \) is the result of gluing the partial information together in an incremental way. See [4] for a further discussion including a geometrical interpretation.
2. Notice that if the interval \( a \) is contained in the interval \([l, r]\), rules 11 and 12 can be applied.
3. Rules 11-13 cannot be made deterministic given the particular computational adequacy formulation which is proved in [10].
4. In practice, one would like to avoid divergent computations by considering a strategy for application of the rules. In [10] total correctness of basic algorithms and in [12] total correctness of first order functions are shown, hence any implementation of any strategy will be correct.

For a deeper discussion of the relation between the operational and denotational semantics of \( LRT \), the reader is referred to [10, 12].
3 The Implementation

In this section, we present the Haskell implementation of the reals and the operational semantics described in the previous section. Additionally, we present an algorithm to compute the average function and discuss why the rate of convergence of this algorithm is slower than another based on the three digits representation of the real implemented by Plume.

We represent in Haskell the real numbers by the datatype CREAL which consist of a pair of the form \((\text{mantissa}, \text{exponent})\) where the \text{mantissa} is an infinite list of rational interval in \([-1,1]\) and the \text{exponent} is an integer. The datatype is defined in Haskell in the following way:

\[
\text{data CoTa } = \text{Cons} \left(\frac{a_2-a_1}{2}, \frac{a_2+a_1}{2}\right)
\]

\[
\text{CREAL } = ([\text{CoTa}], \text{Integer})
\]

Notice that we have not restricted the rational intervals to be in the interval \([-1,1]\), however their used in the implementation does. The \text{cons}, \text{tail} and \text{rtest} operations are implemented as follows:

\[
\text{cons::(Rational,Rational)} \rightarrow (\text{Rational}, \text{Rational}) \rightarrow (\text{Rational}, \text{Rational})
\]

\[
\text{cons (a1,a2) (x1,x2)= ((y1*x1)+y2, (y1*x2)+y2)}
\]

\[
\text{where } \{
\begin{align*}
y_1 &= \frac{(a_2-a_1)}{2}; \\
y_2 &= \frac{(a_2+a_1)}{2};
\end{align*}
\}
\]

\[
\text{tail::(Rational,Rational)} \rightarrow (\text{Rational}, \text{Rational}) \rightarrow (\text{Rational}, \text{Rational})
\]

\[
\text{tail (a1,a2) (x1,x2) = max (min ((2*x1+d)/c), (1)) (-1),}
\]

\[
\text{min (max ((2*x2+d)/c) (-1)), (1))}
\]

\[
\text{where } \{
\begin{align*}
c &= a_2-a_1; \\
d &= (-a_2-a_1);
\end{align*}
\}
\]

The \text{if} operator is the already predefined operator in Haskell.

Notice that the non-deterministic \text{rtest} operator can be implemented in two ways as pointed out in the previous section:

\[
\text{rtest :: (Rational, Rational)} \rightarrow \text{[CoTa]} \rightarrow \text{Bool}
\]

\[
\text{rtest 1 r (Cons(x1,x2):xs)}
\]

\[
| x2 <= r = True \\
| x1 >= l = False
\]

or

\[
\text{rtest' :: (Rational, Rational)} \rightarrow \text{[CoTa]} \rightarrow \text{Bool}
\]

\[
\text{rtest' 1 r (Cons(x1,x2):xs)}
\]

\[
| x1 >= l = False \\
| x2 <= r = True
\]
However adequacy of the language as presented in [10] ensures that any of them is a correct and totally convergent program.

To approximate a real number, the first rule of the operational semantics is applied to the elements on the mantissa as many times as precision is required. This is achieved by the first rule of the operational semantics implemented together with the other operational rules as follows:

\[
\text{evaluacion}:: [\text{CoTa}] \rightarrow [\text{CoTa}]
\]

\[
\text{evaluacion} \ (\text{Cons} \ (a,b):[])=\text{Cons} \ (a,b)
\]

\[
\text{evaluacion} \ (\text{Cons} \ (a,b):\text{Cons} \ (c,d):xs)=\text{Cons} \ ((\text{evaluacion} \ (\text{cons} \ (a,b) \ (c,d))):xs)
\]

\[
\text{evaluacion} \ (\text{Cons} \ (a,b):xs)=\text{evaluacion} \ (\text{Cons} \ (a,b):\text{evaluacion} \ xs)
\]

\[
\text{evaluacion} \ (\text{Tail} \ (a,b):\text{Cons} \ (c,d):xs)
\]

\[
\text{if} \ (b\leq c)
\]

\[
\text{then} \ [\text{Cons}(-1,0),\text{Cons}(-1,0),\text{Cons}(-1,0),...]
\]

\[
\text{if} \ (a> d)
\]

\[
\text{then} \ [\text{Cons}(0,1),\text{Cons}(0,1),\text{Cons}(0,1),...]
\]

\[
\text{if} \ ((a \leq c) && (d < b)) || ((a \leq c) && (d \leq b))
\]

\[
\text{then} \ \text{Cons} \ (\text{tail} \ (a,b) \ (c,d)):xs
\]

\[
\text{if} \ ((a < c) && (b < d))
\]

\[
\text{then} \ \text{Cons} \ (\text{tail} \ (a,b) \ (c,b)): (\text{Tail} \ (\text{tail} \ (c,d) \ (c,b))):xs
\]

\[
\text{if} \ ((c < a) && (d < b))
\]

\[
\text{then} \ \text{Cons} \ (\text{tail} \ (a,b) \ (a,d)): (\text{Tail} \ (\text{tail} \ (c,d) \ (a,d))):xs
\]

\[
\text{otherwise} = xs
\]

\[
\text{evaluacion} \ (\text{Tail} \ (a,b):xs)=\text{evaluacion} \ (\text{Tail} \ (a,b):\text{evaluacion} \ (xs))
\]

\[
\text{evaluacion} \ []=\text{error} \ "empty \ list"\]

It is worth to note that this implementation of the operational semantics only works with real numbers in the interval $[-1,1]$. The final result to the desired precision is calculated multiplying both rational numbers at the head of the mantissa by 2 to the power of the exponent.

A real valued function $f: (CREAL)^n \rightarrow (CREAL)^m$ takes as input $n$ pairs of type $CREAL$ and returns $m$ pairs of type $CREAL$.

For example a definition of the average function defined in the language LRT is:

\[
\text{Average}(x,y):I \rightarrow I
\]

\[
\text{Average}(x,y)= \begin{cases} 
\text{if} \ \text{rtest}_{l,r}(x) \\
\text{then if} \ \text{rtest}_{l,r}(y) \\
\text{then} \ \text{cons}_L(\text{Average}(\text{tail}_L(x),\text{tail}_L(y))) \\
\text{else} \ \text{cons}_C(\text{Average}(\text{tail}_L(x),\text{tail}_R(y))) \\
\text{else} \ \text{if} \ \text{rtest}_{l,r}(y) \\
\text{then} \ \text{cons}_C(\text{Average}(\text{tail}_R(x),\text{tail}_L(y))) \\
\text{else} \ \text{cons}_R(\text{Average}(\text{tail}_R(x),\text{tail}_R(y))),
\end{cases}
\]

where $L = [-1,r], C = [l,r]$ and $[l,1]$. This definition can be implemented in Haskell using LRT as:
Average\((x, y) : \text{CREAL} \rightarrow \text{CREAL}\)

\[
\begin{align*}
\text{Average} (x, y) &= \begin{cases} 
\text{if } rtest - \frac{3}{4} \frac{3}{4} (x) \\
\text{then if } rtest - \frac{3}{4} \frac{3}{4} (y) \\
\text{then cons} (-1, \frac{3}{4}) (\text{Average}(\text{tail} (-1, \frac{3}{4})(x), \text{tail}(-1, \frac{3}{4})(y))) \\
\text{else cons}(\frac{3}{4}, \frac{3}{4}) (\text{Average}(\text{tail}(-1, \frac{3}{4})(x), \text{tail}(-\frac{3}{4}, 1)(y))) \\
\text{else if } rtest - \frac{3}{4} \frac{3}{4} (y) \\
\text{then cons}(-\frac{3}{4}, \frac{3}{4}) (\text{Average}(\text{tail}(-\frac{3}{4}, 1)(x), \text{tail}(-1, \frac{3}{4})(y))) \\
\text{else cons}(-\frac{3}{4}, 1) (\text{Average}(\text{tail}(-\frac{3}{4}, 1)(x), \text{tail}(-\frac{3}{4}, 1)(y))),
\end{cases}
\end{align*}
\]

It can be noticed the rational numbers \(l\) and \(r\) where substituted by \(-\frac{3}{4}\) and \(\frac{3}{4}\) respectively. These numbers can be arbitrarily fixed if the conditions \(-1 < l < r < 1\) are considered, see [9] for a discussion. However at the implementation level not always the shortest algorithm guarantees a fast convergence to the desired precision. In this case because the rate of convergence of this program is \((3/4)^n\), this program converges slower than a program whose rate of convergence is \(\frac{1}{2^n}\). Considering the conditions stated previously, it can be easily shown that for this particular program, there are no values of \(l\) and \(r\) which improve or reach the rate of convergence \(\frac{1}{2^n}\) in all possible executions of the program.

**Lemma 1.** There are not values of \(l\) and \(r\) in \([-1, 1]\) such that \(-1 < l < r < 1\) and every execution path of the average program converges at range of convergence of \(1/2^n\) or faster.

**Proof.** To guarantee convergence at range of \(1/2^n\) or faster, the rescaling factor of the \(\text{cons}_{(x, y)}\) equation should satisfy:

\[
((x - x)/2) \leq (1/2)
\]

by the rescaling factor of \(\text{cons}_L\)

\[
((r + 1)/2) \leq (1/2) \Rightarrow r \leq 0
\]

and the rescaling factor of \(\text{cons}_R\)

\[
((1 - l)/2) \leq (1/2) \Rightarrow l \geq 0
\]

contradicting the assumption.

The above lemma does not imply that there is not a program which converges faster, in fact the above program was presented in [8] only as evidence that basic operations like additions can be implemented in LRT.

If we consider the three digit algortihm [15] for the average function which guarantees a rate of convergence of \(\frac{1}{2^n}\) and translate it to LRT, we have the following program:

\[
\begin{align*}
\text{faverage :: (CREAL ,CREAL) } &\rightarrow \text{CREAL} \\
\text{faverage}(x, y) &= \text{ }
\end{align*}
\]
if rtest \( -\frac{1}{2} \) 0 (x)
then
  if rtest \( -\frac{1}{2} \) 0 (y)
    then Cons (-1, 0) (faverage(Tail (-1, 0) x, Tail (-1, 0) y))
    else
      if rtest 0 \( \frac{1}{2} \) (y)
        then Cons (-\frac{3}{4}, \frac{1}{4}) (faverage(Tail (-1, 0) x, Tail (-\frac{1}{2}, \frac{1}{2}) y))
        else Cons (-\frac{1}{2}, \frac{1}{2}) (faverage(Tail (-1, 0) x, Tail (0, 1) y))
      else
        if rtest 0 \( \frac{1}{2} \) (x)
          then
            if rtest \( -\frac{1}{2} \) 0 (y)
              then Cons (-\frac{3}{4}, \frac{3}{4}) (faverage(Tail (-\frac{1}{2}, \frac{1}{2}) x, Tail (-\frac{1}{2}, \frac{1}{2}) y))
              else
                if rtest 0 \( \frac{1}{2} \) (y)
                  then Cons (-\frac{1}{2}, \frac{1}{2}) (faverage(Tail (-\frac{1}{2}, \frac{1}{2}) x, Tail (0, 1) y))
                  else Cons (-\frac{1}{4}, \frac{3}{4}) (faverage(Tail (-\frac{1}{2}, \frac{1}{2}) x, Tail (0, 1) y))
            else
              if rtest \( -\frac{1}{2} \) 0 (y)
                then Cons (-\frac{1}{4}, \frac{3}{4}) (faverage(Tail (0, 1) x, Tail (-\frac{1}{2}, \frac{1}{2}) y))
                else Cons (0, 1) (faverage(Tail (0, 1) x, Tail (0, 1) y))

Program Average divides the interval \([-1, 1]\) in two overlapping intervals \((-1, 3/4] \text{ and } [-3/4, 1]\) resulting in four cases in the program. Program faverage divides the interval \([-1, 1]\) it three overlapping intervals \([-1, 0], [-1/2, 1/2] \text{ and } [0, 1]\) resulting in nine cases in the program. Table 1 presents the time reported by the Glasgow compiler doing \(n\) different average operations with both programs. Although program Average has less code lines than program faverage, the rate of convergence in program faverage is better.

<table>
<thead>
<tr>
<th>Number of operations</th>
<th>Time reported</th>
<th>Program Average</th>
<th>Program faverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>47.50 sec</td>
<td>0.032 sec</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>73.49 sec</td>
<td>0.112 sec</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>99.56 sec</td>
<td>0.136 sec</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>129.51 sec</td>
<td>0.144 sec</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>287.49 sec</td>
<td>0.344 sec</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Time reported by the Haskell Glasgow compiler at doing \(n\) average operations in both programs with precision \(1E-11\).
Basic operations like addition, subtraction, multiplication, division and trigonometric operations like $\sin, \cos, \tan$ among others were programmed in LRT using Plume’s algorithms. The reader can download either the text modules or the graphical interface (described in the next section) from http://fi.uaemex.mx/rmarcial/LRT.

4 An end-user interface for a basic calculator

In order to present a user-friendly interface we implemented a graphical version of the calculator. This implementation consisted on the following modules:

- The calculator interface where the user writes (selects) both the operation to be calculated and the precision with the result has to be presented.
- A lexical analyzer to transform the given input in series of tokens to be classified.
- A syntactical analyzer to parser the tokens.
- A `stringtoreal` function, which transform the operands to its infinite lists representations (we remind that Haskell has a lazy evaluation hence the elements of the list are calculated as they are needed). Also this module contains a function which represents a real number as a pair \((\text{mantissa}, \text{exponent})\) where the \text{mantissa} is in the interval \([-1,1]\) and the \text{exponent} is an integer.
- An evaluator which does the calculations to the required precision, using the LRT operation.

5 Conclusion

We have described an implementation of LRT in the Glasgow Haskell compiler and a basic calculator using Plume’s algorithms. Although these algorithms have the same range of convergence like Plume’s algorithms, our implementation is slower as Table 2 shows. We consider that the growth of the rational intervals during a calculation decreases the efficiency of our implementation due to the number of operations required. We believe that the use of dyadic rational (as used by other implementation \([6,1,5,7,13]\)) and a faster library to compute operation with the rational numbers will improve the efficiency of our implementation. Being the language representation independent, other algorithms proposed for first order computable functions can be programed in LRT. A further work is to translate the best of those algorithms to LRT. In this paper we do not present any efficiency results of the implementation which similar to others, is still the main gap between what is needed and what has been achieved.

References

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{\sin(1/3 + \cos(7/9))}{\cos(1/3 + \sin(7/11))}$</td>
<td>0.716 sec</td>
</tr>
<tr>
<td>$e^{(\tan(3/11) - \tan(2/13))}$</td>
<td>1.048 sec</td>
</tr>
<tr>
<td>$\pi + \arctan(1/3) + (\cos(2/3) * \tan(13/15))$</td>
<td>0.556 sec</td>
</tr>
<tr>
<td>$\sin(3/11 + e^{(1/3)}) + \sin(4/13 - e^{(2/3)})$</td>
<td>0.564 sec</td>
</tr>
</tbody>
</table>

Table 2. Time reported by the Haskell Glasgow compiler. All operations were calculated at precision $1E - 11$


15. Dave Plume. A calculator for exact real number computation. 4th Year Project Report, Department of Computer Science and Artificial Intelligence, University of Edinburgh, 1998.


On a Hierarchy of 5' → 3' Sensing WK Finite Automata Languages* **

Benedek NAGY

Department of Computer Science, Faculty of Informatics
University of Debrecen, 4010 Debrecen, PO Box. 12, Hungary
nbenedek@inf.unideb.hu

Abstract. In this paper we investigate several variations of Watson-Crick automata in which both heads read the doubled DNA strand form 5' to 3' direction. Some versions of these automata recognize exactly the linear context-free languages. The deterministic versions are not so powerful. The language class of 2-deterministic linear languages can be defined by them. A hierarchy of the accepted languages is presented using the all-final, simple, 1-limited and no-state versions and their combinations of these machines. By full reading of both strands the machine accepts non context-free languages, as well.

1 Introduction

DNA computers provide new paradigms of computing [6]. They appeared in the end of last century. In other side, the theory of finite automata is well developed and intensively used both in theory and practice. The Watson-Crick automata (WK automata, introduced in [2]) relate to both fields, they are important in the field of (DNA) computing and also, have important relation to formal language and automata theory. WK-automata are automata reading doubled DNA strings (instead of a tape) with two heads. In most cases these automata augmented with squeezing mechanisms such as weak codings and deterministic sequential transducers characterize the recursively enumerable languages. Watson-Crick automata have also been studied for their modeling capability of certain processes in cells. A DNA strand (it can be represented by a sequence of bases) has two ends, namely 5' and 3' ends. Two strands can build a double strand if they are Watson-Crick complement of each other, i.e., reading the first strand from 5' to 3' and the second strand from 3' to 5' direction the bases are complement of each other. In the nature the direction 5’ to 3’ is preferred, both DNA and RNA polymerase use this direction. It is a natural idea to consider WK automata in which both heads are moving in the same direction concerning the structure of the DNA, i.e., in direction from the end 5' to 3’. A reverse version (having head

* **Keywords**: Watson-Crick automata, DNA computer, formal languages, linear languages, 2-head automata, hierarchy.

**The work is supported by the National Office for Research and Technology (NKTH), Hungary and by a bilateral Japanese-Hungarian project of the TéT Alapítvány.
movements in this opposite physical direction) and sensing versions (the heads sense if they are at the same position) are also mentioned in [6]. In [5] such a version was investigated which finishes the process when the heads meet. In this paper we will consider these sensing $5' \rightarrow 3'$ Watson-Crick automata in which the machine senses when the heads are close to meet. Several variations of these automata will be analysed. Full reading sensing version is also analysed, in which both heads read the full strand in accepting computations. We note here that a kind of extensions of full reading variants are investigated in [4], where the decision on acceptance of a word is done after $m$ runs of correlated full reading.

2 Preliminaries

In this section we recall some well-known concepts of DNA computing and formal language theory. Readers who are not familiar in these topics should read [6, 3].

Let $V$ be an alphabet and $\rho \subseteq V \times V$ be its complementary relation. For instance $V = \{A, C, G, T\}$ is usually used in DNA computing with the Watson-Crick complementary relation $\{(T, A), (A, T), (C, G), (G, C)\}$. The strings built up by complemernter pairs of letters are double strands (of DNA). The sets of these strings are the languages. In this paper $\lambda$ refers for the empty word.

Now the finite state machines are recalled. A 5-tuple $A = (V, Q, q_0, F, \delta)$ is a finite automaton, with the (input) alphabet $V$; $Q$ is the finite (non-empty) set of states; $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final (or accepting) states. The mapping $\delta$ is the transition function. If $\delta : Q \times (V \cup \lambda) \rightarrow 2^Q$, then the device is the non-deterministic; if $\delta : Q \times V \rightarrow Q$ then the machine is deterministic finite automaton. A word $w$ is accepted by a finite automaton if there is a run starting with $q_0$, ending in a state in $F$ and the symbols of the transitions of the path yield $w$.

A Watson-Crick finite automaton (or shortly, a WK automaton) is a finite automaton working on a Watson-Crick tape, that is a double stranded sequence (or molecule) in which the lengths of the strands are equal and the elements of the strands are pairwise complements of each other: 
\[
\begin{bmatrix}
a_1 & a_2 & \cdots & a_n \\
b_1 & b_2 & \cdots & b_n
\end{bmatrix}
\]
with $a_i, b_i \in V$ and $(a_i, b_i) \in \rho$ ($i = 1, \ldots, n$). The notation $\begin{bmatrix}w_1 \\ w_2\end{bmatrix}$ is used only for strings $w_1, w_2$ with equal length and satisfying the complement relation $\rho$. The set of all double stranded strings with this property is denoted by $WK_\rho(V)$. For double stranded strings for which these conditions not necessarily satisfied the notation $\begin{bmatrix}(w_1) \\ (w_2)\end{bmatrix}$ is used throughout the paper. Formally a WK automaton is $M = (V, \rho, Q, q_0, F, \delta)$, where $\rho \subseteq V \times V$ is a symmetric relation, $V, Q, q_0$ and $F$ are the same as at finite automata, and the transition mapping $\delta : Q \times (V^*) \rightarrow 2^Q$ in such a way that $\delta \left(q, \begin{bmatrix}(w_1) \\ (w_2)\end{bmatrix}\right)$ ($w_1, w_2 \in V^*$) is non empty only for finitely many values of $\left(q, \begin{bmatrix}(w_1) \\ (w_2)\end{bmatrix}\right)$.

The elementary difference between finite automata and WK automata besides the doubled tape is the number of heads. The WK automata scan separately
each of the two strands, in a correlated manner. WK automata can read finite strings of the strands at a transition, while finite automata can read only letters, by the usual definition. There are some restricted variations of the WK automata which are widely used in the literature:

- N: stateless, i.e., with only one state: if \( Q = F = \{ q_0 \} \);
- F: all-final, i.e., with only final states: if \( Q = F \);
- S: simple, (at most one head moves in a step) \( \delta : (Q \times (\{ V_{\lambda}^* \} \cup \{ V_a \})) \rightarrow 2Q \);
- 1: 1-limited, (exactly 1 letter read in a step) \( \delta : (Q \times (\{ V_{\lambda}^* \} \cup \{ V_a \})) \rightarrow 2Q \).

Further variations, such as NS, FS, N1 and F1 WK-automata can be identified in a straightforward way by using multiple constraints.

Now we recall some language families related to the Chomsky hierarchy (for full definitions we refer to [3, 7]). A grammar is a construct \( G = (N, V, S, H) \), where \( N, V \) are the non-terminal and terminal alphabets. \( S \in N \) is the initial letter. \( H \) is a finite set of derivation rules. Let \( v, w \in (N \cup V)^* \). Then \( v \Rightarrow w \) is a direct derivation if there exist \( v_1, v_2, v', w' \in (N \cup V)^* \) such that \( v = v_1 v' v_2 \), \( w = v_1 w' v_2 \) and \( v' \rightarrow w' \in H \). The transitive and reflexive closure of \( \Rightarrow \) is the relation derivation \( \Rightarrow^* \). The language generated by a grammar \( G \) is \( L(G) = \{ w | S \Rightarrow^* w \wedge w \in V^* \} \). Two grammars are equivalent if they generate the same language modulo \( \lambda \). From now on we do not care whether \( \lambda \in L \) or not.

Usually in the field of DNA computing the empty string/empty sequence do not belong to any languages, since it is not a molecule. There are various classes of grammars and languages [1, 3]. In linear (Lin) context-free grammars: each rule is one of the next forms: \( A \rightarrow v, A \rightarrow vBw \); where \( A, B \in N \) and \( v, w \in V^* \). The fix-rated linear grammars are linear grammars with the following property: there is a rational number \( k \) such that for each rule of the form: \( A \rightarrow vBw \), \( k = \frac{|v|}{|v|} \) (where \(|v| \) denotes the length of \( v \)). For these grammars the notation \( k\text{-Lin} \) is used. Specially with \( k = 1 \) the even-linear (1-Lin) grammars; while with \( k = 0 \) the regular (Reg) grammars are obtained. The language family regular/linear etc. contains all languages that can be generated regular/linear etc. grammars. In [1] the definition of even-linear languages was extended to any fix-rated linear languages. The classes fix-rated and \( k \)-rated linear languages are strictly between the linear and regular ones for any rational value of \( k \). It can easily be proven that every \( k \)-rated linear grammar has an equivalent one in which for every rule of the form \( A \rightarrow vBw \) where \(|v| = n \) and \(|w| = m \) are relative primes and for all rules of the form \( A \rightarrow w \): \(|u| < n + m \) holds. The finite automata, both deterministic and non-deterministic versions, recognize exactly the regular languages.

3 The sensing 5′ → 3′ WK automata

At the definition of WK automata we left open the interpretation of \( \delta \) and the condition of acceptance of a word. Now we specialize them to get sensing 5′ → 3′ WK automata.

In a 5′ → 3′ WK automaton both heads start from the 5′ end of the appropriate strand. Physically/mathematically and in computing point of view
they read the double stranded sequence in opposite direction, while in biologically/chemically they are going to the same direction. A $5' \rightarrow 3'$ WK automaton is sensing, if the heads sense that they are meeting (i.e., they are close enough to meet in the next step or there is a possibility to read strings at overlapping positions). In the basic version of sensing $5' \rightarrow 3'$ WK automata the process of the input sequence ends if for all pairs of the sequence one of the letters is read. Due to the complementary relation the sequence is fully processed, the automaton makes a decision on the acceptance. In full reading version both of the heads read the whole strand from the end $5'$ to the end $3'$.

In the usual WK automata the state transition is a mapping of the form $(Q \times (V^*_V)) \rightarrow 2^Q$. In a transition $q' \in \delta(q, (u_1,w_2))$ we call $r_l = |w_1|$ and $r_r = |w_2|$ the left and right radius of the transition (they are lengths of the strings that the head from left to right and from right to left read in this step, respectively). The value $r = r_l + r_r$ is the radius of the transition. Since $\delta(q, (u_1,w_2))$ is non-empty only for finitely many triplets of $(q,w_1,w_2)$, there is a transition (maybe more) with maximal radius for a given automaton. Let $\delta$ is extended by the sensing condition in the following way: Let $r$ be the maximum of the values $r_l + r_r$ for the values given in the transition function of the original WK automaton. Then let $\delta' : (Q \times (V^*_V) \times D) \rightarrow 2^Q$ where $D$ is the sensing distance set $\{-\infty, 0, 1, ..., r, +\infty\}$. This set gives the distance of the two heads between 0 and $r$, and gives $+\infty$ when the heads are further than $r$. (In full reading version it is $-\infty$ when the heads are after their meeting point.) Trivially this automaton is finite, and $D$ can be used only to control the sensing, i.e., the appropriate meeting of the heads. To describe the work of the automata we use the concept of configuration. A configuration $(u_1,w_2)(q,s)(u_1',w_2') \in WK^\rho(V)$ consists of the state $q$, the actual sensing distance $s$, and the input $[w_1w_1',w_2w_2'] \in WK^\rho(V)$ in such a way that the first head (upper strand) has already processed the part $w_1$, while the second head (lower strand) has already processed $w_2'$. A step of the automaton, according to the state transition function can be the following two types:

Normal steps: $(u_1,w_2,y) (q,+\infty)(x,w_1,w_2') \Rightarrow (w_1x,w_1,w_2,x,y) \in V^*$ with $|w_2y| - |w_1| > r$, $q.q' \in Q$ if and only if $[w_1xw_1',w_2yw_2'] \in WK^\rho(V)$ and $q' \in \delta(q,y,+,+\infty)$, and $s = \begin{cases} |w_2| - |w_1x|, & \text{if } |w_2| - |w_1x| \leq r; \\ +\infty, & \text{in other cases.} \end{cases}$

In full reading version the steps after the meeting of the heads can analogously be described with $-\infty$ instead of $+\infty$. Moreover in these steps the value of the sensing distance remains $-\infty$.

Now we are describing the sensing steps: $(u_1,w_2,y) (q,s)(xw_1') \Rightarrow (w_1x,w_1,w_2',x,s') \in WK^\rho(V)$, with the value $s' = \begin{cases} s - |x| - |y|, & \text{if } s - |x| - |y| \geq 0; \\ -\infty, & \text{in other cases.} \end{cases}$

This step can be done if and only if $[w_1xw_1',w_2yw_2'] \in WK^\rho(V)$ and $q' \in \delta(q, y, s')$. 

269
The reflexive and transitive closure of the relation $\Rightarrow$ will be denoted by $\Rightarrow^*$. The accepted language can be defined by the final accepting configurations that can be reached from the initial one:

A doubled strand $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ is accepted by a basic variation sensing $5' \rightarrow 3'$ WK automaton $M$ if and only if $(\lambda)_{w_2}(q_0, s_0)(w_1) \Rightarrow^* \begin{bmatrix} w'_1 \\ w'_2 \\ w''_1 \\ w''_2 \end{bmatrix} (q_f, 0)$ for $q_f \in F$ where $\begin{bmatrix} w'_1 \\ w'_2 \\ w''_1 \\ w''_2 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ with the proper value of $s_0$ (it is $+\infty$ if $|w_1| > r$, elsewhere it is $|w_1|$) since the full input is processed by the time the heads meet.

A doubled strand $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ is accepted by a full-reading sensing $5' \rightarrow 3'$ WK automaton $M$ if and only if $(\lambda)_{w_2}(q_0, s_0)(w_1) \Rightarrow^* (w_1)(q_f, -\infty)(\lambda)_{w_2}$ for $q_f \in F$ with the proper value of $s_0$.

The restricted versions $\textbf{F}, \textbf{N}, \textbf{S}, 1, \textbf{FS}, \textbf{F1}, \textbf{NS}$ and $\textbf{N1}$ can be defined analogously by the appropriate restrictions. In type $1$, i.e., in $1$-limited version $D = \{0, 1, +\infty\}$ due to the restricted radius of the allowed transitions. We can also define deterministic sensing $5' \rightarrow 3'$ WK automata. An automaton (of any of the previously described one) is deterministic if there is at most one possible step to continue its run in any configuration that can occur in any procedure.

It is obvious that a double stranded sequence has no start and end, it is symmetric for this purpose. It means that all words that are accepted by a WK automaton has a complement-symmetric pair which is also in the language. This fact does not cause any problem in connection to formal language theory. For instance, double strands having only $A$ and $C$ in a strand (and thus having $T$ and $G$ in the other) can represent languages over a binary alphabet: considering the pair $\begin{bmatrix} A \\ T \end{bmatrix}$ as letter $a$ and $\begin{bmatrix} C \\ G \end{bmatrix}$ as letter $b$ in the new alphabet $V'$.

In the next sections properties, mostly the hierarchy of the accepted language classes of several variations will be discussed.

4 Hierarchy by sensing $5' \rightarrow 3'$ WK automata

In this section we consider the basic variation and its restricted forms. These automata finish the input word, when the heads are meeting.

We start by an example: Let $L$ be the complemener of the so-called marked copy language, i.e., its double stranded version. The double stranded marked copy is defined as $\left\{ \begin{bmatrix} w_1 \\ w_2 \\ c \\ c' \end{bmatrix} \mid w_1 \in \{a, b\}^*, \right\}$, where $a, b, c$ and their complements $a', b', c'$ are in $V$. It can be accepted by a non-deterministic sensing $5' \rightarrow 3'$ WK automaton, as it follows.

First case: The number or the place of $\begin{bmatrix} c \\ c' \end{bmatrix}$ does not match. The two heads stepping simultaneously, one by one till they meet. If any of them found a $c/c'$ before the meeting, then the word is accepted independently of the remaining
parts. If there is no c and c' at all, then the word is also accepted. By this option only words in shape \( \left[ \begin{array}{c} w_1 \\ w_2 \end{array} \right] \left[ \begin{array}{c} c \\ c' \end{array} \right] \left[ \begin{array}{c} w'_1 \\ w'_2 \end{array} \right] \) with \( |w_1| = |w'_1| \) are not accepted. Therefore in the remaining case there must be a letter of \( w_1 \) which is not repeated correctly in the appropriate position of \( w'_1 \). Let the machine guess its position: the first head goes to this position (before the first occurrence of a c). The machine stores the letter found at that place in the state. Then both of the heads step one by one simultaneously till the first head finds the first c. Then the second head checks the letter under it (this position in \( w'_1 \), i.e., in \( w'_2 \) corresponds to the one in \( w_1 \) that is stored in the state if the word has the assumed form). If this letter is not the complement of the stored one, then the machine accepts the word independently of the rest part, else rejects it by this run.

Now we are showing that some of the restrictions do not restrict the accepting power of these machines:

**Theorem 1.** The accepted language classes of the F, S, 1, FS and F1 sensing 5' \( \rightarrow \) 3' WK finite automata are equal to the class of languages that can be accepted by sensing 5' \( \rightarrow \) 3' WK finite automata (without restrictions).

Ideas of proofs: By introducing new intermediate states every step of the arbitrary automata can be divided to two steps in which only one of the heads can move, respectively (S). Introducing new intermediate states these steps can be simulated by a sequence of transitions in which only a head steps by a letter (1). (\( \lambda \)-transitions can be eliminated in the usual way.) All states can be final, since the finishing transitions are sensing transitions, and they can be allowed only in accepting runs (F). By combining the previous arguments and constructions equivalent FS and F1 versions can also be constructed to any original sensing 5' \( \rightarrow \) 3' WK finite automata.

The next theorem is based on one of the main results of [5].

**Theorem 2.** The languages that can be accepted by sensing 5' \( \rightarrow \) 3' WK finite automata and the class of linear context-free languages are the same.

As a corollary of the previous theorem we just proved, that the complement of the marked copy language is linear context-free. Now we present a real hierarchy.

**Proposition 1.** The language class accepted by \( N_1 \) sensing 5' \( \rightarrow \) 3' WK automata are strictly included in the class of languages that can be accepted by \( N \) sensing 5' \( \rightarrow \) 3' WK automata. The language class accepted by \( NS \) sensing 5' \( \rightarrow \) 3' WK automata are strictly included in the class of languages that can be accepted by \( N \) sensing 5' \( \rightarrow \) 3' WK automata. The language class accepted by \( N \) sensing 5' \( \rightarrow \) 3' WK automata are strictly included in the class of languages that can be accepted by arbitrary sensing 5' \( \rightarrow \) 3' WK automata. The language class accepted by \( NS \) sensing 5' \( \rightarrow \) 3' WK automata are strictly included in the class of regular languages. The class of regular language are strictly included in the class of languages that can be accepted by arbitrary sensing 5' \( \rightarrow \) 3' WK automata.
The first three inclusions are trivial by the defined restrictions. In these cases we only present languages that proves the strict inclusions.

The language $\left[ \frac{aa}{bb} \right]^n$ can be accepted by NS machine, but not with N1 machine.

There is an N sensing $5' \rightarrow 3'$ WK automaton that accepts the language of palindromes, but this language cannot be accepted by NS machine. It is obvious that the language of the complement of the marked copy cannot be accepted by any N machine.

For the proof of the place of the regular languages one can observe that NS automata can accept only languages of the form $(x_1 + x_2 + \ldots + x_i)^*(z_1 + \ldots + z_k)(y_1 + y_2 + \ldots + y_j)^*$, where the strings $x_i$ are the strings that the first head can read in a normal (non-sensing) transition, while $y_i$ corresponds to the complement of the strings that the second head can read in a normal step, finally the strings $z_i$ correspond to finishing steps using sensing steps. In this way, the language $a^*b^*c^*$ cannot be accepted by these machines. In other side it is well-known that regular languages are strictly included in the set of fix-rated linear languages (moreover all linear languages are $k$-rated linear for any $k \in \mathbb{Q}$).

Figure 1 summarizes these results by a Hasse diagram. Directed paths represent strict inclusions, while the language classes not having directed path in any direction between them are incomparable. In the figure deterministic versions are also included, they will be detailed in the next section.

5 The deterministic sensing $5' \rightarrow 3'$ WK automata

In this section some new variants of the automata will also be investigated, and a hierarchy result will be given. A WK automaton is deterministic if at each possible configuration at most one step is possible. We will abbreviate the class accepted by these automata by 2-detLin showing that it is deterministic with the 2-head machine.

Now we define another type of restricted automata, namely automata with fixed radius. A sensing $5' - 3'$ WK automaton is said to have fixed radius (we use the term fx WK, for shortly) if all of its transitions in the form $q' \in \delta \left(q, (y, x), +\infty \right)$ has fixed left and right radius, i.e., $|x| = n, |y| = m$ with fixed $n, m \in \mathbb{N}$. This means that the automaton reads equal length strings by the first head in each step, but the sensing case. And the same holds for the second head.

We note here, that by traditional WK automata moving both heads in the same physical direction, the automata with fix radius can be used only when both heads read exactly the same length strings in every step, and so the work of the machine is equivalent to a regular finite automata accepting exactly the regular languages. Contrary to this fact it is more general for $5' \rightarrow 3'$ WK automata.

**Proposition 2.** For any sensing $5' - 3'$ WK automaton with fixed radius there is an equivalent one which is deterministic.

The proof is analogous with the set construction used to prove the equivalence between the non-deterministic and deterministic finite automata.
Theorem 3. The sensing $5' - 3'$ WK automata with fixed radius recognize exactly the fixed linear languages.

The proof is constructive. One can construct a grammar from the automata and vice-versa ($N = Q, S = q_0$, and there is a strong relation between $\delta$ and $H$).

**Corollary 1.** All $k$-linear languages for any value of $k$ are 2-detLin languages.

Since the language of palindromes is even-linear ($k = 1$), it is in 2-detLin.

**Proposition 3.** There are 2-detLin languages that are not fixed linear.

One can construct a deterministic machine that has transitions with various radii and can accept a non fixed linear language. Such an example follows, by the linear grammar ($\{S\}, \{a, b\}, S, \{S \rightarrow aSa, S \rightarrow bSa, S \rightarrow aaaSb, S \rightarrow babaSaba, S \rightarrow \lambda\}$). This language can be accepted by a deterministic N machine.

Further in this section we present some hierarchy results among the concerned variations of WK automata. First observe that in deterministic NS and so in N1 automata only 1 of the heads can move in normal steps, the other only can wait for sensing. In this way only more restricted special regular languages (i.e., in the form $(x_1 + x_2 + \ldots + x_i)^*(z_1 + \ldots + z_k)$) can be accepted than by the non-deterministic version.

To show the strict inclusion between deterministic NS and deterministic N1 and between deterministic N and deterministic NS automata languages the same examples work as we used in the previous section. Between deterministic arbitrary and deterministic N automata languages let us consider the following example. The language generated by the grammar ($\{S, A\}, \{a, b\}, S, \{S \rightarrow aSa, S \rightarrow bAb, A \rightarrow aAb, A \rightarrow bAa, A \rightarrow \lambda\}$) cannot be accepted by N machine, but can be accepted by deterministic WK automata since it is even-linear.

Now we show a language that can be accepted by N, but cannot by deterministic N version: ($\{S\}, \{a, b\}, S, \{S \rightarrow aSbb, aSbb, S \rightarrow \lambda\}$).

Finally we have some equivalently powerful machines.

**Theorem 4.** The accepted language classes of the deterministic F, S, 1, FS and F1 sensing $5' - 3'$ WK finite automata are equal to the class of languages that can be accepted by deterministic sensing $5' - 3'$ WK finite automata (without further restrictions).

The idea of the proof is similar as at Theorem 1 using the fact that in deterministic version there is at most one applicable transition in any time.

### 5.1 Properties of the language class 2-detLin

In this section we give some properties of the language class that can be accepted by deterministic sensing $5' - 3'$ WK automata.

This set is incomparable with the deterministic linear context-free languages defined by the help of deterministic one-turn pushdown automata. (The mentioned even-linear language, the language of palindromes is not deterministic
in traditional sense.) The language family $2$-detLin is not closed under union, concatenation and Kleene-closure. The language classes of $k$-rated linear languages are closed under union. Opposite to this fact, the operation union can destroy the determinism in $2$-detLin (let the upper-strand language is $\{a^n cb^n|n \in \mathbb{N}\} \cup \{a^n db^{2n}|n \in \mathbb{N}\}$, which is union of two fix-rated linear languages with not the same value of $k$, this language, however, is deterministic linear in traditional way, but not in $2$-detLin). The operations concatenation and Kleene-closure can produce non-linear context-free languages (e.g., $\{a^n b^n a^m b^m|n, m \in \mathbb{N}\}$ in one of the strands). By a deterministic automata it is obvious that these languages are parsable in deterministic linear time. This property is one of the most important nice properties of the regular languages that also holds for these type of extensions. This family nicely fits to the extended Chomsky hierarchy (as one can observe in Fig. 1, $\text{Reg} \subseteq k$-Lin $\subseteq 2$det-Lin $\subseteq \text{Lin}$) and has some nice properties, therefore we recommend it to use in applications as well.

6 The full reading sensing $5' \rightarrow 3'$ WK automata

Finally we analyse the full reading variation of the automata. In this variation each of the heads reads the whole word, but in different directions.

It is clear, and it is in [5], that every linear context-free language is accepted by a full reading sensing $5' \rightarrow 3'$ WK automaton. Moreover they can accept non-context-free languages, such as $\left\{ \left[\begin{array}{c} a^n b^m c^n \\ d^m b^m c^m \end{array}\right]|n \in \mathbb{N}\right\}$ where $V = \{a, b, c, a', b', c'\}$ with the complementary relation including the pairs of letters and their primed versions, respectively. Another important language of the mildly context-sensitive language classes, the $\left\{ \left[\begin{array}{c} a^n b^m c^n d^m \\ a^m b^m c^m d^m \end{array}\right]|n, m \in \mathbb{N}\right\}$ can be accepted by these machines.

**Theorem 5.** The language classes accepted by $S$ full reading sensing $5' \rightarrow 3'$ WK automata and by $1$ full reading sensing $5' \rightarrow 3'$ WK automata are the same as the class accepted by arbitrary full reading sensing $5' \rightarrow 3'$ WK automata.

Without reaching the completeness we provide some other hierarchy results.
Proposition 4. The language class accepted by $N_1$ full reading sensing $5' \rightarrow 3'$ WK automata is strictly included in the language class that is accepted by $NS$ full reading sensing $5' \rightarrow 3'$ WK automata, and that is strictly included in the class accepted by $N$ full reading sensing $5' \rightarrow 3'$ WK automata, that is strictly included in the class accepted by $F$ full reading sensing $5' \rightarrow 3'$ WK automata.

We finish the paper by answering some decidability questions that are more detailed in [4] where extended versions of full-reading automata were investigated. The fact that these automata accept non-context-free languages suggests that some decidability problems related to them may not be solvable. First we take a look at the non-emptiness problem, i.e., the question whether the language accepted by a given automaton is empty or not. A closely related problem is the finiteness problem, the question whether the language accepted by a given automaton is finite or not. They are decidable for context-free languages, but in contrast we have the following

Theorem 6. For full-reading sensing $5' \rightarrow 3'$ automata the non-emptiness problem and the finiteness problem are undecidable.

The proof goes by checking Turing-Machine computations written as words.

Corollary 2. Every recursively enumerable language is a homomorphich image of a language accepted by a full reading sensing $5' \rightarrow 3'$ WK automaton.

7 Conclusions

In this paper several results were shown about the hierarchy of language classes accepted by various versions of the sensing $5' \rightarrow 3'$ WK automata (Fig. 1). The deterministic version accept a language class with nice properties: it contains all fixed linear languages and it is parseable in deterministic linear time. Some questions about the hierarchy of full reading versions left open.

References

4. Leupold, P. and Nagy, B.: $m$-run $5' \rightarrow 3'$ Watson-Crick automata, submitted
Acknowledgments for Lengths of Bad Sequences of Monomial Ideals over Polynomial Rings in two Variables

Florian Pelupessy and Andreas Weiermann

Vakgroep Zuivere Wiskunde en Computeralgebra
Krijgslaan 281 Gebouw S22
9000 Ghent
Belgium
pelupessy@cage.ugent.be
weierman@cage.ugent.be

Abstract. A famous result of Hilbert states that in a polynomial ring in finitely many variables over a field every ascending chain of ideals eventually becomes stationary. This theorem is nowadays very familiar as Hilbert’s basis theorem and it is of fundamental importance in algebraic geometry (for example in the theory of Gröbner bases). Moreno Socías [4] has shown that the lengths of such ascending chains of ideals can be bounded primitive recursively (from above and below) when the degrees of the generators of the ideals are bounded according to a linear growth rate condition.

Quite recently MacLagan [3] has shown that there are no infinite antichains in the set of monomial ideals in a polynomial ring in finitely many variables over a field. Aschenbrenner and Pong [1] investigated her result extensively from the viewpoint of the theory of well partial orders and they computed several related (and very interesting) ordinal invariants. In this note we complement their results, in particular Proposition 3.25 in that paper, and show an Ackermannian lower bound on bad (with respect to reverse inclusion) sequences (which satisfy a linear growth rate condition) of monomial ideals in a polynomial ring in just two variables over a field.

This result is somewhat surprising since it shows a fundamental difference between MacLagan’s theorem and Hilbert’s basis theorem. For the general case of a fixed number of variables we expect multiple recursive upper and lower (by the methods of this paper) bounds on the lengths of bad sequences of monomial ideals which satisfy a linear growth rate condition.

We assume basic familiarity with ordinals, their cantor normal forms and primitive recursive functions (the smallest class of functions \( N^d \to N \) which contains the constant functions, projections and successor function and is closed under composition and primitive recursion.) We call a function Ackermannian if it
eventually dominates every primitive recursive function. If \( A \) is defined by:

\[
A_0(i) := i + 1 \\
A_{n+1}(i) := A_n(i) \\
A(i) := A_i(i),
\]

then \( A \) is Ackermannian and any primitive recursive function is eventually dominated by one of the \( A_n \). \( K \) is a field, ideals are from the polynomial ring \( K[X, Y] \) and monomial ideals are ideals generated by monomials. If we have a set of monomials \( G \) we denote the ideal generated by \( G \) with \( \langle G \rangle \). For ideal \( I \) and monomial \( p \) we use \( \langle I, p \rangle \) to denote the ideal generated by the generators of \( I \) and \( p \). For a set of generators \( G \) take for \( |G| := \max\{n + m : aX^nY^m \in G\} \). For an ideal \( I \) we take as its complexity \( |I| := \min\{|G| : \langle G \rangle = I\} \).

**Theorem 1.** Let \( M(l) \) be the smallest \( M \) such that for every sequence \( I_0, \ldots, I_M \) of monomial ideals in \( K(X, Y) \) for which for all \( i \leq M \) with \( |I_i| \leq l + i \) there exist \( i < j < M \) with \( I_i \supseteq I_j \). Then \( M(l) \) is Ackermannian.

As a preliminary examine the ordinals \( \alpha < \omega^{k+1} \). These can be written in cantor normal form:

\[
\alpha = \omega^k \cdot m_k + \cdots + \omega^0 \cdot m_0.
\]

Define:

\[
\Phi_k(\omega^k \cdot m_k + \cdots + \omega^0 \cdot m_0) := \langle X^kY^{m_k+1}, \ldots, X^0Y^{m_k+k+1} \rangle,
\]

where the generators have the form: \( X^iY^{m_k+1+i} \).

Furthermore take \( |\omega^k \cdot m_k + \cdots + \omega^0 \cdot m_0| := m_k + \cdots + m_0 \).

Then \( \alpha < \beta < \omega^{k+1} \) implies \( \Phi_k(\alpha) \not\subseteq \Phi_k(\beta) \) and \( |\Phi_k(\alpha)| = |\alpha| + k + 1 \).

Alternatively one can use \( \alpha \in \mathbb{N}^{k+1} \) ordered lexicographically. Then \( \Phi_k \) and \( |\alpha| \)

are defined as above for \( \alpha = (m_k, \ldots, m_0) \). \( \alpha < \omega^{k+1} \) still implies \( \Phi_k(\alpha) \not\subseteq \Phi_k(\beta) \)

and we still have \( |\Phi_k(\alpha)| = |\alpha| + k + 1 \).

Let \( f_k(l) \) be the maximum length of a descending sequence \( \alpha_0, \ldots, \alpha_R \) of ordinals less than \( \omega^{k+1} \) such that \( |\alpha_i| \leq l + i \). Then straightforward checking gives us \( M(l + k + 1) \geq f_k(l) \). Hence it is sufficient to examine \( f_k(l) \).

Alternatively one can define \( f_k(l) \) in terms of appropriate \( <_{\text{lex}} \)-descending sequences in \( \mathbb{N}^{k+1} \). The proofs that follow work just as well with these sequences.

**Lemma 1.** \( f_1(l) \geq A_0(l) \)

**Proof.** Note that the sequence \( \omega \cdot l, \omega \cdot (l - 1), \ldots, 0 \) already has length \( l + 1 \).

**Lemma 2.** \( f_k+1(l) > f_k(l) \) for \( l, k > 0 \).

**Proof.** We prove this by constructing an appropriate sequence. Take for each \( j > 0 \) a sequence \( \alpha_0^j, \ldots, \alpha_{l-1}^j \) such that \( \alpha_i^j < \omega^{k+1} \) and \( |\alpha_i^j| \leq f_{k-1}(l) + i \) for all \( i \). We now construct our new sequence from these in the following manner:

\[
\omega^{k+1} \cdot l, \omega^{k+1} \cdot (l - 1) + \alpha_0^1, \ldots, \omega^{k+1} \cdot (l - 1) + \alpha_{f_k(l)-1}^1, \ldots, \alpha_0^0, \ldots \alpha_{f_k(2)-1}^0.
\]
The length of this descending sequence is strictly greater than $f_k^{(l)}(2)$. The $i$th element $\beta_i$ ($i > 0$) of this sequence is:

$$\beta_i = \omega^{k+1} \cdot (l-j) + \alpha_i^j,$$

where

$$i = 1 + f_k(2) + \cdots + f_k^{(j-1)}(2) + i'.$$

So we have:

$$|\beta_i| = (l-j) + |\alpha_i^j| \leq (l-j) + f_k^{(j-1)}(2) + i' \leq l + i.$$

**Lemma 3.** For each primitive recursive function $h$ there exists a $k$ such that $h$ is eventually dominated by $f_k$.

**Proof.** Note that each primitive recursive $h$ is dominated by an $A_k$, so it is sufficient to show that for each $k$ there is $k'$ such that $A_k$ is eventually dominated by $f_{k'}$. We prove this with induction on $k$:

- $k=0$: See the first lemma.
- $k+1$: Take $k'$ for $k$. Then $f_{k'+2}(l) > f_{k'+1}(l) > f_{k}(l)$, where

$$A_k^{(l)}(2) \geq A_{k+1}(l).$$

We have used $f_{k'+1}(2) \geq 2l$ for sufficiently large $l$, to prove this we show $f_{k'+1}(n) \geq 2n$ with the sequence:

$$\omega \cdot n, \omega \cdot (n-1) + 2, \omega \cdot (n-1) + 1, \omega \cdot (n-2) + 1, \omega \cdot (n-2) + 0, \ldots, 1, 0$$

where on the dots we have elements of the form $\omega \cdot (n-m) + 1$ or $\omega \cdot (n-m) + 0$.

**Proof (Theorem).** By the previous lemma $n \mapsto f_n(n)$ is Ackermannian. Hence $n \mapsto M(2n+1)$ is also Ackermannian. So $M$ is Ackermannian.

**Theorem 2.** Let $\tilde{M}(l)$ be the smallest $M$ such that for every sequence $I_0, \ldots, I_M$ of monomial ideals in $K(X,Y)$ for which for all $i \leq M$ with $|I_i| \leq l + i$ there exist $i \neq j < M$ with $I_i \supseteq I_j$.

Then $\tilde{M}(l)$ is Ackermannian.

**Proof.** We show $\tilde{M}(l+k+2) \geq f_k(l)$. Given sequence $\alpha_0 > \cdots > \alpha_{f_k(l)-1}$ of ordinals less than $\omega^{k+1}$ such that $|\alpha_i| \leq l + i$ we take:

$$I_i = \langle \Phi_k(\alpha_i), X^{k+1+i} \rangle$$

**References**

Petri Nets Unfoldings and the Individual/Collective Token Philosophy

G. Michele Pinna
Dipartimento di Matematica e Informatica
Università degli Studi di Cagliari, Cagliari, Italy
gmpinna@unica.it

Abstract. The notion of unfolding plays a major role in the so called non sequential semantics of Petri nets. In literature various approaches to this notion have been proposed, taking into account the so called individual token philosophy (the whole history is relevant) and the so called collective token philosophy (part of the history can be forgotten). In this paper we compare and relate two notions of unfolding (one for the individual and the other for the collective token philosophy) and we investigate on the relations between these two approaches and the non sequential behaviour of Petri Nets.

1 Introduction

Petri nets are since almost 40 years one among the most widely used models for concurrent computations. In order to describe computations in this setting two main approaches can be considered: one asserts that each happening of a transition depends on how tokens are consumed and produced by other transitions, and a second declares that the happening of a transitions depends only on the fulfillment of certain conditions, without any concern on how certain conditions were produced. In other words, one approach considers the history relevant whereas the other considers the history irrelevant. To make more precise this concept, consider the following two nets:

In the net $N_1$ the transition $a$ depends on the happening of $b$ or $c$, and if the history would be considered relevant, the information on which of $b$ or $c$ is actually...
executed must be kept, whereas if the history here does not play a major role then the information on which of the two transitions was executed is negligible. The history respecting approach is mainly represented by a notion of unfolding (which will be called $\mathcal{I}$-unfolding) as introduced by Winskel ([1])\(^1\). In this case the transition $a$ depending from $b$ is different from the transition $a$ depending from $c$. Concerning the other possible approach, van Glabbeek and Plotkin in [3] introduced the notion of 1-unfolding to model the irrelevance of the history and contextually introduced a notion of event structures, called configuration structures where this principle is reflected. Several authors have addressed this issue: Gunawardena in [4] introduced a notion, called Muller unfolding (for the net $N_1$ this notion is based on the intuition that the $i-th$ happening of $a$ depends on the happening of $i-1$ occurrences of $b$ or $c$)\(^2\), and introduced a suitable event based semantics, expressing a principle of history irrelevance. In fact already in [5] and [6] Gunawardena advocates the principle of irrelevance of history by introducing causal automata, where the so called or causality can be easily modeled. In [7] a notion of event structure is introduced, where the dependencies among events are modeled in a local fashion, implementing a limited principle of history irrelevance, whereas in [8] a notion of scenario is introduced to adjust some weaknesses of classical partial order semantics for Petri nets, where again the history is not posed as a central notion. In [9] van Glabbeek analyzes the computational interpretations of various firing rules for Petri Nets, distinguishing between the individual token approach (the one where history is relevant) and the collective token one (where history is not relevant), and the author of this paper in [10] relates the 1-unfolding introduced in [3] to an event based model called event automata ([11]).

Clearly both approaches have ground. Let us assume that each transition can fire only once. Consider the following interpretation of the transition in the net $N_1$: $a$ represent the act of printing and $b$ and $c$ the act of producing something to be printed. In the collective token approach the focus is put on the act of printing, whereas in the individual token approach on what it is actually printed, which subsumes the act of printing, thus clearly the individual token approach give rise to a finer grained view with respect the collective token approach. On the other hand, consider now the following interpretation (due to Antoni Mazurkiewicz and reported in [9]): $a$ represents the act of buying a present 1£ worth to a teacher by two pupils contributing with 1£ each. The individual token philosophy would stress which of the 2 £ is used, contrasting the more natural interpretation. In this case the history would produce a kind of distorted interpretation of the modeled phenomenon. The problem of the relevance of the history is evident also in net $N_2$. Here the transition $b$ can consume the token present in the initial marking or consume the the token produced by $a$, and the same for $a$ (consume the token present in the initial marking or consume the the token produced by

---

\(^1\) Also other authors have investigated on this notion, e.g. Engelfriet [2]

\(^2\) Though Gunawardena uses the word unfolding differently from what is done in this paper: here the unfolding of a net is a net respecting some properties.
b). Again the individual token would distinguish between these two occurrences of b, whereas the collective token would not.

In this paper we want to compare two notion of unfoldings introduced in literature, similarly to what van Glabbeek have done in [9] with respect to firing sequences: one is (almost) the classical defined by Winskel and the other is the notion of unfolding introduced by van Glabbeek and Plotkin. Not surprisingly, and in accordance with van Glabbeek findings, we will discover that what we will call the individual unfolding, i.e. the unfolding for the individual token philosophy can be folded onto the collective unfolding (i.e. the unfolding for the collective token philosophy). We show also that the collective unfolding can be unfolded onto the individual one. This account to say that the history can be always recovered. We then consider an event based model associated to these notions of unfolding. With the individual and collective unfoldings we are considering the extrema of a spectrum where dependencies among happening of transitions are considered. On one side there is the individual unfolding with each happening carrying its past, and on the other there is the collective unfolding where all the possible pasts of the happening of a transition are equated. Clearly there are other possibility in between, e.g. the Muller unfolding we have mentioned before, which can be captured by ruling out autoconcurrency, i.e. the possibility that two happenings of the same transition are concurrent (auto concurrency can be ruled out either on the net level, which implies a structural limitation, or on the unfolding level, which implies a semantical limitation). These other notion will be the subject of further investigations.

The paper is organized as follows: in the next section we recall the notions we will use in the rest of the paper. In section 3 we introduce the two notions of unfolding which we will compare in section 4 and relate to event automata in section 5.

2 Preliminaries

To give the formal definitions we need some notation for multisets. With \( \mathbb{N} \) we denote the set of natural numbers. Let \( A \) be a set, a multiset of \( A \) is a function \( M : A \rightarrow \mathbb{N} \). The set of multisets of \( A \) is denoted by \( \mu A \). The usual operations and relations on multisets, like multiset union \( + \) or multiset difference \( - \), are used. We write \( M \leq M' \) if \( M(a) \leq M'(a) \) for all \( a \in A \). If \( M \in \mu A \), we denote by \([M]\) the multiset defined as \([M](a) = 1 \) if \( M(a) > 0 \) and \([M](a) = 0 \) otherwise; sometimes \([M]\) will be confused with the corresponding subset \( \{ a \in A \mid [M](a) = 1 \} \) of \( A \). A multirelation \( f \) from \( A \) to \( B \) (often indicated as \( f : A \rightarrow B \)) is a multiset of \( A \times B \). We will limit our attention to finitary multirelations, namely multirelations \( f \) such that the set \( \{ b \in B \mid f(a,b) > 0 \} \) is finite. Multirelation \( f \) induces in an obvious way a function \( \mu f : \mu A \rightarrow \mu B \), defined as \( \mu f(\sum_{a \in A} n_a \cdot a) = \sum_{b \in B} \sum_{a \in A} (n_a \cdot f(a,b)) \cdot b \) (possibly partial, since infinite coefficients are disallowed). If \( f \) satisfies \( f(a,b) \leq 1 \) for all \( a \in A \) and \( b \in B \), i.e. \( f = [f] \), then we sometimes confuse it with the corresponding set-relation and write \( f(a,b) \) for \( f(a,b) = 1 \).
Nets and morphisms We first review the notions of Petri net and of the token game. Then we recall the notion of morphism. Nets and morphism are a category, but in this paper the focus is not on the categorical approach to nets, but rather on the various relations among the notions of unfolding that are definable on nets. Nevertheless morphisms are quite handy in highlighting how nets are related.

**Definition 1.** A Petri net is a 4-tuple $N = (S,T,F,m)$, where $S$ is a set of places, $T$ is a set of transitions (with $S \cap T = \emptyset$), $F = (F_{\text{pre}},F_{\text{post}})$ is a pair of multirelations from $T$ to $S$, and $m \in \mu S$, and it is called the initial marking.

We require that for each $t \in T$, $F_{\text{pre}}(t,s) > 0$ for some place $s \in S$. Subscripts on the net name carry over the names of the net components. As usual, given a finite multiset of transitions $A \in \mu T$ we write $A^*$ for its pre-set $\mu F_{\text{pre}}(A)$ and $A^\bullet$ for its post-set $\mu F_{\text{post}}(A)$. The same notation is used to denote the functions from $S$ to $2^T$ defined as, for $s \in S$, $s^* = \{t \in T \mid F_{\text{post}}(t,s) > 0\}$ and $s^\bullet = \{t \in T \mid F_{\text{pre}}(t,s) > 0\}$.

Let $N$ be a net. A finite multiset of transitions $A$ is enabled at a marking $M$, if $M$ contains the pre-set of $A$ and an additional multiset of tokens which covers the context of $A$. Furthermore no token must be present nor produced by the transitions in the places of the inhibitor set of $A$. Formally, a finite multiset $A \in \mu T$ is enabled at $M$ if $A^* \leq M$. In this case, to indicate that the execution of $A$ in $M$ produces the new marking $M' = M - A^* + A^\bullet$ we write $M[A]M'$.

Step and firing sequences, as well as reachable markings are defined in the usual way. The set of reachable markings of a net $N$ is denoted with $\mathcal{M}_N$.

A net $N = (S,T,F,m)$ is said safe in the case that $F_{\text{pre}}$ and $F_{\text{post}}$ are such that $F_{\text{pre}} = [F_{\text{pre}}]$, $F_{\text{post}} = [F_{\text{post}}]$ and each marking $m \in \mathcal{M}_N$ is such that $m = [m]$.

We recall now the notion of morphism between nets, which we will use to construct an unfolding (and also to relate the various notions of unfolding).

**Definition 2.** Let $N_0$ and $N_1$ be nets. A morphism $h : N_0 \to N_1$ is a pair $h = (\eta,\beta)$, where $\eta : T_0 \to T_1$ is a partial function and $\beta : S_0 \to S_1$ is a multirelation such that (a) $\mu \beta(m_0) = m_1$ and (b) for each $t \in T$, $\mu \beta^*(t) = \eta(t)^*$, and $\mu \beta^*(t^*) = \eta(t)^*$.

The conditions of the above definition are the defining conditions of Winskel’s morphisms on ordinary nets ([1]). In the case two elements are not in multirelation with each others we will occasionally indicate it with $*$ (as it is done with partial functions). The category of net and morphisms is called Petri.

A morphism $h : N_0 \to N_1$ in this setting is a simulation, in the sense that each reachable marking in $N_0$ is also a reachable marking in $N_1$.

**Proposition 1.** Let $N_0$ and $N_1$ be nets, and let $h = (\eta,\beta) : N_0 \to N_1$ be a net morphism. For each $M,M' \in \mathcal{M}_{N_0}$ and $A \in \mu T$, if $M[A]M'$ then $\mu \beta(M)[\mu \eta(A)]\mu \beta(M')$. Therefore net morphisms preserve reachable markings, i.e. if $M_0$ is a reachable marking in $N_0$ then $\mu \beta(M_0)$ is reachable in $N_1$.
Subnet A subnet of a net is a net obtained restricting places and transitions, and correspondingly also the multirelation $F$ and the initial marking.

**Definition 3.** Let $N = \langle S, T, F, m \rangle$ be a Petri net and let $T' \subseteq T$. Then the subnet generated by $T'$ is the net $N|_{T'} = \langle S', T', F', m' \rangle$, where $S' = \{ s \in S \mid F_{\text{pre}}(t, s) > 0 \text{ or } F_{\text{post}}(t, s) > 0 \text{ for } t \in T' \} \cup \{ s \in S \mid m(s) > 0 \}$, $F' = \langle F'_{\text{pre}}, F'_{\text{post}} \rangle$ is the pair of multirelations from $T'$ to $S'$ obtained restricting $\langle F_{\text{pre}}, F_{\text{post}} \rangle$ to $S'$ and $T'$, and $m'$ is the multiset on $S'$ obtained by $m$ restricting to places in $T'$.

It is trivial to observe that, given a net $N = \langle S, T, F, m \rangle$ and a subnet $N|_{T'}$, with $T' \subseteq T$, then the pair $\eta(t) = t$ and $\beta : S \to S'$ is $\beta(s, s) = 1$ if $s \in S'$ and 0 otherwise is a well defined morphism from $N$ to $N|_{T'}$.

**Occurrence Nets** The notion of occurrence net we introduce here is the one called 1-occurrence net and proposed by van Glabbeek and Plotkin in [3]. First we need to introduce the notion of state.

**Definition 4.** Let $N = \langle S, T, F, m \rangle$ be a net, the state is any finite multiset $X$ of transitions with the property that the function $m_X : S \to \mathbb{Z}$ given by $m_X(s) = m(s) + \sum_{t \in T} X(t) \cdot (F_{\text{post}}(t, s) - F_{\text{pre}}(t, s))$, for all $s \in S$, is a reachable marking of the net. With $\mathcal{X}_N$ we denote the states of a net.

**Definition 5.** An occurrence net $O = \langle S, T, F, m \rangle$ is a Petri net where each state is a set, i.e. $\forall X \in \mathcal{X}_N$ it holds that $X = \lfloor X \rfloor$.

Observe that, differently from the notion of causal net we will introduce later, an occurrence net is not required to be a safe net. The only requirement is that each transition in the net is fired at most once. Thus each transition can occur only once, hence the reason for calling them occurrence nets. We will see later that this is also a characteristic of what we will call causal nets.

Another relevant difference is that it is not possible (or at least easy) to obtain directly relationships among occurrences of transitions, as it is with causal nets.

**Causal Nets** The notion of causal net we use here is the classical one, though it is often called occurrence net. The different name is due to the other notion of occurrence net we have introduced above.

As causal nets are safe nets, it is possible to see the multirelations $F_{\text{pre}}$ and $F_{\text{post}}$ as a flow relation $F \subseteq (S \times T) \cup (T \times S)$ by stating $s F t$ iff $F_{\text{pre}}(t, s)$ and $t F s$ iff $F_{\text{post}}(t, s)$. Hence we can use the usual notation for the transitive (and reflexive) closure of this relation. For denoting places and transitions we use the notation $B$ and $E$ (see [12] and [1, 13]).

**Definition 6.** A causal net $C = \langle B, E, F, m \rangle$ is a safe net satisfying the following restrictions: (a) $\forall b \in m$, $\bullet b = \emptyset$, (b) $\forall b \in B$. $\exists b' \in m$ such that $b'F^*b$, (c) $\forall b \in B$. $|\bullet b| \leq 1$, (d) $F^+$ is irreflexive and, for all $e \in E$, the set $\{ e' | e' F^* e \}$ is finite, and (e) $\#$ is irreflexive, where $e#e'$ iff $e, e' \in E$, $e \neq e'$ and $\bullet e \cup \bullet e' \neq \emptyset$, and $x \# x'$ iff $\exists y, y' \in B \cup E$ such that $y\#_1 y'$ and $yF^*x$ and $yF^*x'$. 283
On causal nets it is easy to define a relation expressing concurrency: two elements of the causal net are concurrent if they are neither causally dependent nor in conflict. Formally \( x \mathbin{\text{co}} y \) iff \( \neg (x \# y \text{ or } xF^+ y \text{ or } yF^+ x) \). This relation can be extended to sets of conditions: let \( A \subseteq B \), then \( \mathbf{co}(A) \) iff \( \forall b, b' \in A. \ b \mathbin{\text{co}} b' \) and \( \{ e \in E \mid \exists b \in A. \ eF^* b \} \) is finite.

**Proposition 2.** Let \( C = \langle B, E, F, m \rangle \) be a causal net. Then \( C \) is also an occurrence net.

## 3 Unfoldings

In this section we review the notions of unfolding for the individual and collective token philosophy.

**Individual token philosophy** In the case of the individual token philosophy the unfolding of a net \( N \) is a causal net. It can be defined either as top of a chain of causal nets or as the unique causal net satisfying certain conditions. In the following we will use the same notation both for the multirelations and for the relations between transitions (events) and places (conditions). This can be done safely as on causal nets these coincide.

**Proposition 3.** The \( \mathcal{I} \)-unfolding \( \mathcal{U}_\mathcal{I}(N) = \langle B^\mathcal{I}, E^\mathcal{I}, F^\mathcal{I}, m^\mathcal{I} \rangle \) of the net \( N = \langle S, T, F, m \rangle \) is the unique causal net to satisfy:

\[
B^\mathcal{I} = \{(m, s, i) \mid s \in S \text{ and } 0 \leq i < m(s)\} \cup \{\{(e), \ast, 0\} \mid e \in E^\mathcal{I}\} \\
E^\mathcal{I} = \{(X, t) \mid X \subseteq B^\mathcal{I} \text{ and } \mathbf{co}(X) \text{ and } \ast t = \mu \beta^\mathcal{I}(X)\}
\]

\[
F^\mathcal{I} = F^\mathcal{I}_{\text{pre}}((X, t), b) \iff b \in X \text{ or } b = \{((X, t)), \ast, 0\} \\
F^\mathcal{I}_{\text{post}}((X, t), b) \iff \exists s \in S, i \in \mathbb{N}. \ b = ((X, t), s, i)
\]

\[
m^\mathcal{I} = \{(m, s, i) \mid (m, s, i) \in B^\mathcal{I}\} \cup \{\{(e), \ast, 0\} \mid e \in E^\mathcal{I}\}
\]

where \( \mathbf{co} \) is the concurrency relation obtained by \( F' \) on \( B \) and \( E \). Furthermore \( \eta : E^\mathcal{I} \to T \) defined as \( \eta^\mathcal{I}(X, t) = t \) and \( \beta : B^\mathcal{I} \to S \) defined as \( \beta^\mathcal{I}(X, s, i) = s \) form a net morphism, called the \( \mathcal{I} \)-folding morphism.

We omit the proof of this proposition, the interested reader can consult [1] for the proof of a similar one (the case considered there is the one of safe nets). We stress that there are other alternative characterizations of the \( \mathcal{I} \)-unfolding of a net.

The construction we have introduced has a little difference with respect to the ones introduced in literature (e.g. [1] or [2]), as we introduce a condition for each event (transition). This cause no harm (these conditions are superfluous in a net theoretical sense) but they are useful in relating the unfolding constructions.

A well known consequence of this construction is that each reachable marking of the unfolding is a unique reachable marking of the net (this is due to the folding morphism) and also the vice versa holds, i.e. to each reachable marking
of the net \( N \) a reachable marking in the unfolding corresponds (though this is not necessarily unique, as it depends on the history).

**Collective token philosophy** In the case of the collective token philosophy we adopt a slightly different notion with respect to the original one, as we associate to the unfolding also a folding morphism.

**Proposition 4.** The \( \mathcal{C} \)-unfolding \( \mathcal{U}_\mathcal{C}(N) = (S^\mathcal{C}, T^\mathcal{C}, F^\mathcal{C}, m^\mathcal{C}) \) of the net \( N = (S, T, F, m) \) is the unique occurrence net to satisfy:

\[
S^\mathcal{C} = S \cup (T^\mathcal{C} \times \{\ast\})
\]

\[
T^\mathcal{C} = T \times \mathbb{N}
\]

\[
F^\mathcal{C} = F^\mathcal{C}_{\text{pre}}((t, i), s) = \begin{cases}
F_{\text{pre}}(t, s) & \text{if } s \in S \\
1 & \text{if } s = ((t, i), \ast) \\
0 & \text{otherwise}
\end{cases}
\]

\[
F^\mathcal{C}_{\text{post}}((t, i), s) = \begin{cases}
F_{\text{post}}(t, s) & \text{if } s \in S \\
0 & \text{otherwise}
\end{cases}
\]

\[
m^\mathcal{C} = m^\mathcal{C}(s) = \begin{cases}
m(s) & \text{if } s \in S \\
1 & \text{otherwise}
\end{cases}
\]

Furthermore \( \eta^\mathcal{C} : T^\mathcal{C} \rightarrow T \) defined as \( \eta(t, i) = t \) and \( \beta^\mathcal{C} : S^\mathcal{C} \rightarrow S \) is a multirelation defined as \( \beta(s) = s \) iff \( s \in S \), form a net morphism, called the \( \mathcal{C} \)-folding morphism.

**Proof.** It is easy to see that this is indeed an occurrence net as each state of this net is necessarily a set (each transition can occur at most once). We just verify that \( \langle \beta^\mathcal{C}, \eta^\mathcal{C} \rangle \) is indeed a morphism. Observe that \( \mu\beta(m_0) = m_0 \), consider now \( F^\mathcal{C}_{\text{pre}}((t, i), s) \) with \( s \in S \), then it is \( F^\mathcal{C}_{\text{pre}}(t, s) \) and hence applying the multirelation \( \beta^\mathcal{C} \) to \( F^\mathcal{C}_{\text{pre}}((t, i), -) \) gives the proper result. Similarly for \( F^\mathcal{C}_{\text{post}} \).

4 Relating the notions of unfolding

In this section we relate the two notions introduced so far. The first result we present is the folding of the \( \mathcal{I} \)-unfolding onto the \( \mathcal{C} \)-unfolding. This account to state that history can be forgotten. The folding construction is made explicitly by exhibiting a suitable folding morphism.

**Theorem 1.** Let \( N = (S, T, F, m_0) \) be a Petri net and let \( \mathcal{U}_\mathcal{I}(N) = (B^\mathcal{I}, E^\mathcal{I}, F^\mathcal{I}, m^\mathcal{I}) \) and \( \mathcal{U}_\mathcal{C}(N) = (S^\mathcal{C}, T^\mathcal{C}, F^\mathcal{C}, m^\mathcal{C}) \) be the individual (resp. collective) unfolding of \( N \). Then there exists a folding morphism from \( \mathcal{U}_\mathcal{I} \) to \( \mathcal{U}_\mathcal{C} \).
Proof. Consider the $\mathcal{F}$-unfolding $\mathcal{U}_\mathcal{F}(N)$. Define $\eta : E^\mathcal{F} \to T^\mathcal{G}$ as follows: $\eta(X,t) = (t,j)$, with $j \in \mathbb{N}$ and such that $\eta$ is injective. Define now the multirelation $\beta : E^\mathcal{F} \to S^\mathcal{G}$ as follows: $\beta(X,s,i) = s$ and $\beta((X,t),*,0) = ((t,j),*)$ whenever $\eta(X,t) = (t,j)$. It is routine to check that this is a well defined net morphism.

In fact the morphism in the proof of this theorem is almost the same of the folding morphism in proposition 3.

Clearly, as a consequence of the proposition 1, each reachable marking of $\mathcal{U}_\mathcal{F}(N)$ is a reachable marking of $\mathcal{U}_\mathcal{G}(N)$. Observe that there are many different morphisms from the individual unfolding to the collective one: in the individual unfolding the ordering among different instances of the firing of the same transition is relevant, but in the collective unfolding this is not considered.

The relation between the collective unfolding of a net and the individual one is more subtle, as we have to guess an history to be able to relate the collective unfolding to the individual one. Clearly the only morphism is the one show in the previous theorem. However if we consider the a state of the collective unfolding, we can unfold it onto the individual one. Thus we can fix an order in the execution of different occurrences of the same transition (which is arbitrary) and also keep track of the dependencies among different transitions by exploiting which tokens are produced and consumed.

**Theorem 2.** Let $N = \langle S, T, F, m_0 \rangle$ be a Petri net and let $\mathcal{U}_\mathcal{F}(N) = \langle B^\mathcal{F}, E^\mathcal{F}, F^\mathcal{F}, m^\mathcal{F} \rangle$ and $\mathcal{U}_\mathcal{G}(N) = \langle S^\mathcal{G}, T^\mathcal{G}, F^\mathcal{G}, m^\mathcal{G} \rangle$ be the individual (resp. collective) unfolding of $N$. Let $X$ be a state of $\mathcal{U}_\mathcal{G}(N)$ and $m_X$ be the reached marking. Then there exists a finite subnet of $N' = \langle B', E', F', m' \rangle$ of $\mathcal{U}_\mathcal{F}(N)$ such that (a) each reachable marking of $N'$ is a marking of a firing sequence leading to $m_X$, and (b) there exists an one to one correspondence between $E'$ and $X$. Furthermore $N'$ is a causal net such that $|b^*| \leq 1$ for all $b \in B'$.

**Proof.** Let $X$ be a state of $\mathcal{U}_\mathcal{G}(N)$. We do the proof by induction on the number of elements of $X$ (which is a set). If $X$ is the empty set, then the subnet we are looking for has conditions $B' = \{(m,s,i) \mid s \in S$ and $0 \leq i < m(s)\} \cup \{(e,*,0) \mid e \in E^\mathcal{F}\}$, and with an empty set of events, and the initial marking is $B'$ itself (here we confuse the set with the corresponding multiset). This is trivially a subnet of $\mathcal{U}_\mathcal{F}(N)$ and it is such that $|b^*| \leq 1$ for all $b \in B'$. Furthermore there is a set of conditions $\hat{B} \subseteq B'$ such that $b^* = \emptyset$ for all $b \in \hat{B}$, and $\hat{B}$ is a marking of $N'$.

Assume it holds for $n$. Consider a state $X$ such that $|X| = n + 1$. As $m_X$ is a reachable marking there exists a transition $t \in T^\mathcal{G}$ such that $X - \{t\}$ is a state of $\mathcal{U}^\mathcal{G}(N)$. Hence $m_{X - \{t\}} \{t\} m_X$. By induction hypothesis there is a causal net $N' = \langle B', E', F', m' \rangle$ such that each each reachable marking of $N'$ is a marking of a firing sequence leading to $m_X - \{t\}$, there exists an one to one correspondence between $E'$ and $X - \{t\}$, and $N'$ is a causal net such that $|b^*| \leq 1$ for all $b \in B'$. We know that there is a subset of conditions $\hat{B} \subseteq B'$ such that $b^* = \emptyset$ in $N'$. Consider now, in $\mathcal{U}^\mathcal{F}(N)$ the transitions in $\bigcup_{b \in \hat{B}} b^*$, and take an $e$ such that $\eta^\mathcal{F}(e) = \hat{t} = \eta^\mathcal{G}(t)$, which clearly exists. Take now $N''$ as the subnet
generated by \( E' \cup \{e\} \). This is a causal net, and the conditions \( \{e\}, s, i \) are such that \( \{e\}, s, i \) is empty. Consider now the conditions \( B \subseteq B'' \) (the conditions of the net \( E'' \)) such that their postset is empty. They correspond to the marking \( m_X \). The one to one correspondence between \( X \) and \( E'' \) is obviously the one obtained by \( X - \{t\} \) and \( E' \) adding the association \( e \) and \( t \).

5 Event Automata

We recall the notion of event automata.

**Event Automata and morphisms** Event Automata ([11]) basically consist of a set of states (set of events) together with a transition relation. States are finite subsets of events, though they may have more structure (e.g. events may be equipped with a partial order), making the event automata more informative; whereas the transition relation should be such that to go from a state \( s \) to a state \( s' \) at least an event (not appearing in \( s \)) should be added. The transition relation is such that more events can be added in going from a state to another, and we do not require that such a step must be divisible in smaller ones. Clearly all the added events to the state \( s \) must not appear in \( s \).

**Definition 7.** An event automaton is a triple \( A = (E, St, \sim) \), where \( E \) is a set of events, \( St \subseteq 2^E_{\text{fin}} \) is a set of states and \( \sim \subseteq St \times St \), called the transition relation, satisfies: for all \( X, Y \in St \), if \( X \sim Y \) then \( Y = X \cup E' \) with \( E' \subseteq E \), \( E' \cap X = \emptyset \) and \( E' \neq \emptyset \). An event automaton \( A \) is called simple if \( \emptyset \in St \) and \( \emptyset \sim^* s \) for all \( s \in St \), namely each state is reachable from the empty state.

Clearly the empty state is considered the initial one. In [11] it is shown how the configurations of various brands of event structures (general, prime, flow) give rise to (special kinds of) event automata, thus event automata are well suited as event based model.

Event automata have been introduced to capture phenomena involving causality that are difficult to model in other brands of event structures, like the irrelevance of the history of an event or the so called or-causality. Therefore they can be easily related to the notions of unfolding seen in section 3.

We recall the notion of morphism between event automata.

**Definition 8.** Let \( A_0 \) and \( A_1 \) be event automata. A morphism \( f : A_0 \rightarrow A_1 \) is a partial function \( f : E_0 \rightarrow E_1 \) such that \( f(X) \in St_1 \) if \( X \in St_0 \) and \( f(X) \sim^*_1 Y \) whenever \( X \sim^*_0 Y \).

Event automata and morphism form a category, called **EvAut**.

**Event automata and unfoldings** We show how to associate event automata to both unfoldings. Event automata are easily associated to occurrence nets, as shown in [10]. We first introduce a suitable relation on states of the occurrence net.

**Definition 9.** Let \( N = \langle S, T, F, m \rangle \) be a an occurrence net, and let \( \mathcal{E} \) be the set of states of such a net. With \( \leftrightarrow \) we denote the following relation: \( X \leftrightarrow Y \) iff \( X, Y \in \mathcal{E}, X \neq Y \) and \( X \subseteq Y \).
Proposition 5. Let $N = (S, T, F, m)$ be a an occurrence net, and let $\mathcal{X}$ be the set of states of such net. Then $\mathcal{E}(N) = (T, \mathcal{X}, \rightarrow)$ is an event automaton.

Proposition 6. Let $N_0 = (S_0, T_0, F_0, m_0)$ and $N_1 = (S_1, T_1, F_1, m_1)$ be occurrence nets and let $h = \langle \eta, \beta \rangle$ be a morphism $h : N_0 \rightarrow N_1$. Then $\eta : T_0 \rightarrow T_1$ is an event automata morphism from $\mathcal{E}(N_0)$ to $\mathcal{E}(N_1)$.

Thus we can conclude that $\mathcal{E}$ can be lifted to a functor from occurrence nets to event automata.

As we have already noticed, a causal net is an occurrence net as well, hence we do not need a suitable construction for causal nets.

Theorem 3. Let $N = (S, T, F, m)$ be a net. Then $\mathcal{E}(\mathcal{U}(N))$ and $\mathcal{E}(\mathcal{U}^\mathcal{E}(N))$ are event automata. Furthermore, given a folding morphism $\langle \eta, \beta \rangle : \mathcal{U}(N) \rightarrow \mathcal{U}^\mathcal{E}(N)$, then $\eta$ is an event automata morphism from $\mathcal{E}(\mathcal{U}(N))$ to $\mathcal{E}(\mathcal{U}^\mathcal{E}(N))$.

References

Prime Models of Finite Computable Dimension

Pavel Semukhin *
Department of Mathematics, National University of Singapore, 2 Science Drive 2, Singapore.
pavels@nus.edu.sg

Abstract. We study the following open question in computable model theory: does there exist a structure of computable dimension two which is the prime model of its first-order theory? We construct an example of such a structure by coding a certain family of c.e. sets with exactly two one-to-one computable enumerations into a directed graph. We also show that there are examples of such structures in the classes of undirected graphs, partial orders, lattices, and integral domains.

1 Introduction

Computable Model Theory studies the effective content of typical mathematical concepts, constructions, and theorems, especially from algebra and classical model theory. One of the most fundamental notions here is that of an isomorphism. In algebra and model theory we usually identify isomorphic structures and consider them to be the same. However, when studying computable models, one can see that isomorphic structures might have different computability-theoretic properties. We, therefore, introduce the notion of a computable isomorphism, instead of the classical one, and use it as a tool to distinguish two different computable presentations of the same structure. This approach leads us to the notion of a computable dimension which is defined below.

Definition 1. A structure $\mathcal{A}$ is computable if its domain $A \subseteq \omega$ and its atomic diagram are computable. $\mathcal{A}$ is computably presentable if it is isomorphic to a computable structure, which is called a computable presentation of $\mathcal{A}$.

Definition 2. The computable dimension of a structure $\mathcal{A}$ is the number of its computable presentations up to computable isomorphism. If the computable dimension of $\mathcal{A}$ is 1 we say that $\mathcal{A}$ is computably categorical.

In this paper we will answer the following open question in computable model theory: does there exist a structure of computable dimension two which is the prime model of its own theory? It is easy to give examples of prime models of dimension 1 or $\omega$. For instance, the countable dense linear order without endpoints and one successor structure $(\mathbb{N}, S)$ are computably categorical, while

* This research was partially supported by Marsden Fund grant of New Zealand.
(N, ≤) and \( B_\omega \), the Boolean algebra of finite and cofinite subsets of the naturals, have infinite computable dimensions.

However it is much more difficult to construct a structure of finite computable dimension \( k > 1 \). Goncharov [2, 3] was the first to give an example of such structure. In [2] he constructed a uniformly computably enumerable (u.c.e.) family \( F \) of sets that has exactly two non-equivalent one-to-one computable enumerations. This family is then encoded into a computable graph \( G \) in such a way that the computable dimension of \( G \) is equal to the number of non-equivalent one-to-one computable enumerations of \( F \).

Since then many improvements to the construction have been made to obtain various strengthenings of this result. For example, Cholak/Goncharov/Khoussainov/Shore [1] showed that for each \( k > 1 \), there is a computably categorical structure \( A \) such that any expansion of \( A \) by a single constant has computable dimension \( k \). This construction was further improved by Hirschfeldt/Khoussainov/Shore [6] who showed that it is possible to make the dimension of the expanded structure infinite.

The research on the structures of finite computable dimension is also related to the study of degree spectra of relations on computable models. The degree spectrum of a relation \( R \) on a computable structure \( A \) is the set of Turing degrees of images of \( R \) in all computable presentations of \( A \). Harizanov [4] showed that there exists a relation \( U \) in a structure \( A \) of computable dimension two such that \( DgSp_A(U) = \{0, d\} \), where \( d \leq 0' \) and does not contain a c.e. set. Later on, Khoussainov and Shore [8] proved that there exists a relation \( U \) in a structure \( A \) of dimension two such that \( DgSp_A(U) = \{0, d\} \), where \( d \) is c.e. but not computable. Hirschfeldt [5] further improved this result by showing that \( d \) can be chosen to be any non-computable c.e. Turing degree.

All known examples of the structures of finite computable dimension \( k > 1 \) are not the prime models of their theories. Hence it was an open problem as to whether there exists a prime model of computable dimension two. This question is especially interesting because prime models are relatively simple from a model-theoretic point of view as they are elementarily embeddable into any other model of their theories. So, the problem is whether it is possible to encode enough information into a prime model to construct a structure of dimension two. The main result of this paper is the construction of such a structure.

**Theorem.** There exists a structure of computable dimension two which is the prime model of its own theory.

The construction is based on coding a u.c.e. family of sets \( F \) constructed by Goncharov [2] into a graph. We will use some structural properties of \( F \) to make the coding in such a way that every element of the graph is definable by a first order formula, which implies that the structure is prime.

There are also examples of prime models of finite computable dimension in some well-known classes of algebraic structures.
Theorem. There are prime models of computable dimension two in the following classes of structures:

1) undirected graphs
2) partially ordered sets
3) lattices
4) integral domains expanded by finitely many constants

The construction of these examples follows the technique from Hirschfeldt/Khoussainov/Shore/Slinko [7], where they developed the methods for coding directed graphs into undirected, irreflexive graphs, partial orders, lattices, integral domains, nilpotent groups, etc. These codings preserve the following computability-theoretic properties of the structures:

(a) degree spectra of the structures;
(b) degree spectra of relations on computable structures;
(c) computable dimensions of the structures as well as computable dimensions of their expansions by a single constant.

In addition to the above properties, it can be shown that if in the original structure $A$ every element is definable by a first order formula, then the structure $B$, into which $A$ is encoded, is prime. In fact, every $b \in B$ is also definable by a formula or, in the case of integral domains, there is a formula with finitely many solutions that holds on $b$.

2 Proof of the Main Result

We now give a proof of the following theorem which is the main result of this paper.

Theorem 1. There exists a computable structure $G$ of computable dimension two which is the prime model of its own theory.

The structure $G$ will be a directed graph. The proof is based on coding the u.c.e. family $F$ constructed by Goncharov [2] into a computable graph of dimension two in such a way that every element of $G$ can be defined by a first-order formula without parameters. This implies that $G$ is the prime model of its theory. We now restate the result of S. Goncharov in more detail.

Definition 3. Let $F$ be a u.c.e. family of sets. A computable enumeration $\mu : \omega \rightarrow F$ is a mapping from $\omega$ onto $F$ such that the set $\{(n, k) : n \in \mu(k)\}$ is c.e. We will also use the notation $\{A_i\}_{i \in \omega}$ for an enumeration $\mu$, where $A_i = \mu(i)$.

An enumeration $\mu$ is reducible to $\nu$, denoted $\mu \leq \nu$, if there is a computable function $f$ such that $\mu(i) = \nu(f(i))$ for every $i \in \omega$. Two enumerations $\mu$ and $\nu$ are equivalent, denoted $\mu \equiv \nu$, if $\mu \leq \nu$ and $\nu \leq \mu$.

Theorem 2 (Goncharov [2]). There exists a u.c.e. family $F$ that has exactly two nonequivalent one-to-one computable enumerations. Moreover, $F$ has the following properties:
\((i)\) If \(S \in \mathcal{F}\) is a finite set, then \(S\) contains an element \(n(S)\), called a marker for a finite set \(S\), that does not belong to any other set from \(\mathcal{F}\).

\((ii)\) If \(S \in \mathcal{F}\) is an infinite set, then \(S\) contains an element \(n(S)\), called a marker for an infinite set \(S\), that does not belong to any other infinite set from \(\mathcal{F}\).

**Remark 1.** We may assume that the family \(\mathcal{F}\) contains infinitely many one-element sets. Indeed, we can always take \(\mathcal{F}' = \{2S : S \in \mathcal{F}\} \cup \{\{2k + 1\} : k \in \omega\}\) instead of \(\mathcal{F}\). The family \(\mathcal{F}'\) has exactly two nonequivalent one-to-one computable enumerations. This follows from the fact that the index set of the subfamily \(\{\{2k + 1\} : k \in \omega\}\) is computable in any one-to-one computable enumeration of \(\mathcal{F}'\).

Let \(\{A^0_n\}_{n \in \omega}\) and \(\{A^1_n\}_{n \in \omega}\) be two nonequivalent one-to-one computable enumerations of \(\mathcal{F}\). For each \(i = 0, 1\), fix a computable enumeration of \(\{A^n_i\}_{n \in \omega}\) such that at every step \(s\), exactly one element enters one of the \(A^n_i\)'s.

We build two computable presentations \(G_0\) and \(G_1\) of the directed graph \(G\) using a step-by-step construction. Let \(G^s_i\) be the finite part of \(G_i\) constructed by the end of step \(s\). When we add a new element to \(G^s_i\), we always choose the least element that has not been used so far. At every step \(s\), we will have that \(G^s_i \subseteq G^{s+1}_i\) and \(G_i = \bigcup_{s \in \omega} G^s_i\). We will use the following notations in our construction.

**Definition 4.** Let \(n \in \omega\); the directed graph \([n]\) has \(n+3\) many nodes \(x_0, x_1, \ldots, x_{n+2}\) with an edge from \(x_0\) to itself, an edge from \(x_{n+2}\) to \(x_1\), and an edge from \(x_i\) to \(x_{i+1}\) for \(i \leq n + 1\). We call \(x_0\) the top of \([n]\).

Let \(S \subseteq \omega\); the directed graph \([S]\) consists of one copy of \([s]\) for every \(s \in S\), with all tops identified.

**Definition 5.** Two tops \(n_0\) and \(n_1\) of \(G^s_i\) are **connected** if there is an element \(l \in G^s_i\) such that \((n_0, l), (l, n_0), (n_1, l), (l, n_1)\) are edges in \(G^s_i\). In this case \(l\) is called the linking element.

“To connect two tops \(n_0, n_1\) of \(G^s_i\) using a linking element” means to add one new element \(l\) as well as the edges \((n_0, l), (l, n_0), (n_1, l), (l, n_1)\) to the graph \(G^s_i\).

A **component** is a maximal subgraph isomorphic to \([S]\) for some \(S \subseteq \omega\). Note that this is not necessarily the same as a maximal connected component.

The construction of \(G_0\) and \(G_1\) is now as follows.

**Step 0.** Let \(G^0_0 = G^0_1 = \{2n : n \in \omega\}\) and, for every \(n \in \omega\), connect \(2n\) to itself in both \(G^0_0\) and \(G^0_1\). Thus \(2n\) is a top in \(G_0\) and \(G_1\).

**Step \(s+1\).** For \(i \in \{0, 1\}\), do the following. Let \(k\) be the unique element that enters some \(A^n_i\) at step \(s\). Consider the component of \(G^s_i\) isomorphic to \([A^n_{i,n,s}]\) and containing the top \(2n\). Expand this component to one isomorphic to \([A^n_{i,n,s}] \cup \{k\}] = [A^n_{i,n,s+1}]\). If \(k\) is not the first element that enters \(A^n_i\), then find the least \(m\) such
that $2n$ is not connected to $2m$ in $G^*_i$ and connect $2n$ to $2m$ using one new linking element.

Now, for every pair $u_0$, $v_0$ of tops in $G^*_0$ and every pair $u_1$, $v_1$ of tops in $G^*_1$ such that $u_0$, $u_1$ belong to the components isomorphic to $[S_0]$ and $v_0$, $v_1$ belong to the ones isomorphic to $[S_1]$ for some non-empty sets $S_0$ and $S_1$, do the following. Check if $u_0$, $v_0$ are connected in $G^*_0$, but $u_1$, $v_1$ are not connected in $G^*_1$, or vice versa. If yes, connect those tops $u_i$, $v_i$ using one linking element which are not connected in $G^*_i$.

End of the construction.

**Lemma 1.** $G_0$ and $G_1$ are isomorphic.

**Proof.** According to the construction, each top $2n$ in $G_0$ (resp. $G_1$) belongs to the component isomorphic to $[A^0_n]$ (resp. $[A^1_n]$). Since $\{A^0_n\}_{n \in \omega}$ and $\{A^1_n\}_{n \in \omega}$ are one-to-one enumerations of the same family, $G_0$ and $G_1$ consist of the same collection of components.

To finish the proof, we need to show that for every pair $n_0$, $n_1$ of tops in $G_0$ and every pair $m_0$, $m_1$ of tops in $G_1$, if for $i \in \{0, 1\}$, $n_i$ and $m_i$ belong to the isomorphic components, then $n_0$, $n_1$ are connected in $G_0$ iff $m_0$, $m_1$ are connected in $G_1$. Suppose that $n_0$, $n_1$, $m_0$, $m_1$ is a counterexample to the above statement such that, for instance, $n_0$, $n_1$ are connected in $G_0$ and $m_0$, $m_1$ are not connected in $G_1$.

Let $n_i$, $m_i$ be the tops of the components isomorphic to $[S_i]$, where $i \in \{0, 1\}$. By the construction, if $n$ is the top of an infinite component of $G_i$, then $n$ is connected to all other tops in $G_i$. Therefore, $[S_0]$ and $[S_1]$ are finite. Hence there is a step $s_0$ such that both $G^*_0$ and $G^*_1$ contain the components isomorphic to $[S_0]$, $[S_1]$ with tops $n_0$, $n_1$ and $m_0$, $m_1$ respectively, and $n_0$, $n_1$ are connected in $G^*_0$. Now, if $m_0$ and $m_1$ have not yet been connected, then they will be connected at the next step. This contradiction proves the lemma. \hfill \Box

**Lemma 2.** $G_0$ and $G_1$ are not computably isomorphic.

**Proof.** Let $f : G_0 \rightarrow G_1$ be a computable isomorphism. Note that $f$ maps tops to tops, and the component containing the top $2n$ in $G_0$ is isomorphic to the one containing the top $f(2n)$ in $G_1$. Therefore, the enumerations $\{A^0_n\}_{n \in \omega}$ and $\{A^1_n\}_{n \in \omega}$ are reducible to one another via the computable functions $h_0(n) = f(2n)/2$ and $h_1(n) = f^{-1}(2n)/2$, which contradicts our choice of $\{A^0_n\}_{n \in \omega}$ and $\{A^1_n\}_{n \in \omega}$. \hfill \Box

**Lemma 3.** Let $H$ be a computable graph isomorphic to $G$, then $H$ is computably isomorphic either to $G_0$ or to $G_1$. Thus $G$ has computable dimension two.

**Proof.** Since $H$ is computable, there is a computable list $t_0 < t_1 < t_2 < \ldots$ of the tops in $H$, where $<$ is the natural order on $\omega$. The structure $H$ gives rise to a one-to-one computable enumeration $\{A_n\}_{n \in \omega}$ of $\mathcal{F}$ such that $k \in A_n$ iff there is a subgraph of $H$ isomorphic to $[k]$ containing $t_n$ as its top.
Since $\mathcal{F}$ has exactly two nonequivalent one-to-one computable enumerations, \( \{A_n\}_{n \in \omega} \) is equivalent either to \( \{A^0_n\}_{n \in \omega} \) or \( \{A^1_n\}_{n \in \omega} \). Suppose that \( \{A_n\}_{n \in \omega} \) is equivalent to \( \{A^0_n\}_{n \in \omega} \). We now construct a computable isomorphism $h$ from $H$ to $G_0$.

By our assumption, there is a computable function $f$ such that $A_n = A^0_{f(n)}$ for all $n$. Note that $f$ is a permutation of $\omega$ because \( \{A_n\}_{n \in \omega} \) and \( \{A^0_n\}_{n \in \omega} \) are one-to-one. Take any $k \in H$; the value of $h(k)$ is defined according to the following three cases:

1) If $k = t_n$ for some $n$, then $h(k) = 2f(n)$.
2) If $k$ is the linking element between $t_n$ and $t_m$, then $h(k)$ is the linking element between $2f(n)$ and $2f(m)$ in $G_0$. Note that such a linking element exists since $H \cong G_0$.
3) If $k$ is neither a top nor a linking element, then there are $m$ and $n$ such that $k$ belongs to the subgraph of $H$ isomorphic to $[m]$ with the top $t_n$. Let $l$ be the length of the unique path from $t_n$ to $k$ without repetitions. Now, $h(k)$ is the unique element of $G_0$ belonging to the subgraph isomorphic to $[m]$ with the top $2f(n)$ such that the length of the path from $2f(n)$ to $h(k)$ without repetitions is equal to $l$.

By the construction, $h : H \rightarrow G_0$ is an isomorphism. It is computable since, for a given $k \in H$, one can effectively find out which one of the cases 1), 2) or 3) holds and then effectively find the value of $h(k)$.

To show that $G$ is prime we will use the following model-theoretic fact.

**Proposition 1.** Let $A$ be a model such that for every $a \in A$, there is a formula $\varphi_a(x)$ in the language of $A$ with the property that

$$A \models \forall z (\varphi_a(z) \iff a = z).$$

Then $A$ is the prime model of its theory.

**Lemma 4.** $G$ is prime.

**Proof.** Due to Proposition 1, to prove that $G$ is prime it suffices to show that for every $a \in G$, there is a formula $\varphi_a(x)$ in the language of directed graphs such that $G \models \forall x (\varphi_a(x) \iff a = x)$. Let $E(x,y)$ be the edge relation on $G$.

By the construction, the top of every infinite component is connected to all other tops. On the other hand, the top of every finite component is not connected to all other tops. To see this, let $2n_0$ be the top of a finite component $[A^0_{n_0}]$ in $G_0$, and let $n_1$ be such that $A^0_{n_0} = A^1_{n_1}$. Hence, $2n_1$ is the top of a finite component isomorphic to $[A^0_{n_0}]$ in $G_1$. Consider the step $s_0$ by which we have constructed the components $[A^0_{n_0}]$ and $[A^1_{n_1}]$ in $G^s_0$ and $G^s_1$ respectively. Since $\mathcal{F}$ contains infinitely many singletons, there are $k_0$ and $k_1$, such that $2k_0$ and $2n_0$ are not connected in $G^s_0$, $2k_1$ and $2n_1$ are not connected in $G^s_1$, and $A^0_{k_0}$, $A^1_{k_1}$ are equal one-element sets. Then $2k_0$, $2n_0$ are not connected in $G_0$ as well as $2k_1$, $2n_1$ are not connected in $G_1$ because we do not connect $2k_0$ with any top when the only element of $A^0_{k_0}$ is enumerated in it, and the same is true for $2k_1$ in $G_1$. 

294
First, let us define $\varphi_a(x)$ when $a$ is a top. If $a$ is the top of a finite component $[S]$, then $\varphi_a(x)$ states that $E(x, x)$ and $x$ belongs to a subgraph isomorphic to $[n(S)]$, where $n(S)$ is the marker for the finite set $S \in \mathcal{F}$. If $a$ is the top of an infinite component $[S]$, then $\varphi_a(x)$ states that $E(x, x)$ and $x$ belongs to a subgraph isomorphic to $[n(S)]$, where $n(S)$ is the marker for the infinite set $S \in \mathcal{F}$.

If $a$ is a linking element between two tops $u$ and $v$, then

$$\varphi_a(x) = \exists y \exists z \left( E(x, y) \land E(y, x) \land E(x, z) \land E(z, x) \land \varphi_u(y) \land \varphi_v(z) \right).$$

If $a$ is neither a top nor a linking element, then let $k$, $l$, and $u$ be such that $a$ belongs to the subgraph of $G$ isomorphic to $[k]$ with the top $u$, and $l$ is the length of the unique path from $u$ to $a$ without repetitions. In this case $\varphi_a(x)$ states that

$$\exists z \left( \varphi_u(z) \land \text{“}x \text{ belongs to a subgraph isomorphic to } [k] \text{ with top } z\text{“} \land \text{“}there is a path of length } l \text{ without repetitions from } z \text{ to } x\text{“} \right).$$

Theorem 1 now follows from Lemmas 1, 2, 3, and 4.

References

Generalized Temperley-Lieb Algebras and Quantum Computation

Mario Vélez and Juan Ospina
Logic and Computation Group
Physical Engineering Program
Department of Basic Sciences
School of Sciences and Humanities
Medellin, Colombia
{mvelez,jospina}@eafit.edu.co

Abstract. Generalized Temperley-Lieb (TL) algebras such as the two-boundary TL algebra and the bubble algebra are applied in quantum computation. The diagrammatic and matrix representations of these generalized TL algebras are exploited with the aim to obtain the following results: 1) the generalized TL algebras contain generalizations of the Dirac formalism with kets and bras; 2) via Yang-Baxterization of the representations of the generalized TL algebras, quantum Yang-Baxter operators can be derived and its corresponding Zhang-Kauffman-Ge Hamiltonians can be calculated; 3) GHZ states can be obtained via Yang-Baxter deformation of cohomological states associated with knots, links, graphs and algebraic varieties; 4) a physical system like a quantum spin chain with dynamical symmetry described by a generalized TL algebra can be used as the hardware for a topological quantum computer.

Keywords: Topological Quantum Computation, Generalized TL algebras, Quantum Yang-Baxter Operator, Dirac formalism, Cohomological states, GHZ states.

1. Introduction

Recently certain algebraic structures and geometric topology underlying quantum information and computation [1] were discovered [2,3,4,5,6]. Specifically, algebraic structures like braid groups, Hecke algebras, Temperley-Lieb algebras and Birman-Wenzl-Murakami algebras [7] were founded as the natural mathematics for quantum information and computation, including teleportation and cryptography.

In this work we propose that extended Temperley-Lieb algebras, recently introduced in the mathematical literature [8,9,10], such as the two-boundary Temperley-Lieb algebra [8,9], the bubble algebra [10] and the colored and decorated Temperley-Lieb algebras, are a very strong mathematical framework to describe quantum information and computation involving maximally entangled states and local unitary transformations. More technically, in this work we describe quantum information and computation in the language of the two-boundary Temperley-Lieb algebra.
In this paper we formulate a model of topological quantum computation [11,12,13] using as physical referents certain dilute lattice systems of two-dimensional statistical mechanics for which generalized Temperley-Lieb algebras such as the two-boundary or the two-color Temperley-Lieb algebra (Bubble algebra) are the corresponding dynamical algebras [8,9,10]. To construct the model we exploit the fact that the extended Temperley-Lieb algebras are able to produce solution of the Yang-Baxter equations which can be unitarized until to generate universal quantum gates.

The resulting topological quantum computation model contains the mechanisms of quantum teleportation, contains a generalization of the Dirac formalism with kets and bras and entails deepest relationships with link and knot invariant polynomials including Khovanov homology [14]. Finally, it is conjectured that the Yang-Baxterization of more general Temperley-Lieb algebras such as the multi-color braid-monoid algebras associated with solvable lattice models, could be exploited in quantum information, processing and computation.

2. Mathematical Background

The quantum states for a system of two qubits are represented by [1]

$$
|a_{0,0}> = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix},
|a_{0,1}> = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix},
|a_{1,0}> = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},
|a_{1,1}> = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.
$$

The Bell states are defined as [1,2,3]

$$
|\phi_1> = \frac{1}{2}\sqrt{2} (|a_{0,0}> + |a_{1,1}>),
|\psi_1> = \frac{1}{2}\sqrt{2} (|a_{0,0}> - |a_{1,1}>),
|\phi_2> = \frac{1}{2}\sqrt{2} (|a_{0,1}> + |a_{1,0}>),
|\psi_2> = \frac{1}{2}\sqrt{2} (|a_{0,1}> - |a_{1,0}>).
$$

The standard two-qubits states, in terms of the Bell states are rewritten as

$$
|a_{0,0}> = \frac{1}{2}\sqrt{2} (|\phi_1> + |\psi_2>),
|a_{0,1}> = \frac{1}{2}\sqrt{2} (|\psi_1> + |\phi_2>),
|a_{1,0}> = \frac{1}{2}\sqrt{2} (|\psi_1> - |\psi_2>),
|a_{1,1}> = \frac{1}{2}\sqrt{2} (|\phi_1> - |\phi_2>).
$$

The Bell matrix, denoted B has the following standard properties: [1,2,3]

$$
B = \begin{pmatrix}
\frac{1}{\sqrt{2}} & 0 & 1 & \frac{1}{\sqrt{2}} \\
0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\
1 & 0 & \frac{1}{\sqrt{2}} & 0 \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0
\end{pmatrix},
B^{-1} = \begin{pmatrix}
\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \\
0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\
0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\
\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}}
\end{pmatrix},
B^T = \begin{pmatrix}
\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \\
0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\
0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\
\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}}
\end{pmatrix}.
$$

297
Recently was demonstrated that the Bell matrix satisfy the following Yang-Baxter equations [2,3]:

\[(B \otimes I_2) (I_2 \otimes B) (B \otimes I_2) = (I_2 \otimes B) (B \otimes I_2) (B \otimes B)\]
\[(P \otimes I_2) (I_2 \otimes P) (P \otimes I_2) = (I_2 \otimes P) (P \otimes I_2) (I_2 \otimes P)\]
\[(P \otimes I_2) (I_2 \otimes P) (B \otimes I_2) = (I_2 \otimes I_2) (P \otimes I_2) (B \otimes B)\]

where P is the permutation matrix for two-qubits [1,4].

The Yang-Baxter equations are intimately linked with certain algebraic structures in geometric topology such as the braid group and the Temperley-Lieb algebra. Recently the Temperley-Lieb algebra was discovered as a relevant tool in quantum information and computation. In the next section a generalization of the Temperley-Lieb algebra will be presented.

3. The two-boundary Temperley-Lieb algebra

The q-number is defined by:

\[\left[ n \right] = \frac{q^n - q^{-n}}{q - q^{-1}} \quad [8,9]\]

The two-boundary Temperley-Lieb (2BTL) algebra is an associative algebra given by generators \(e_i\) with \(i=0,\ldots,N\); which satisfy the relations [8,9]:

\[e_j^2 = \left( B_{ij} \right) e_j \quad \text{for} \quad j = 1, N - 1 \quad e_i e_j e_i = e_i \quad e_i e_j = e_j e_i \quad \text{with} \quad |i - j| > 1,\]

\[e_0^2 = \left[ w_1 \right] e_0 \quad e_0^2 = \left[ w_2 \right] e_0 \]

The most simple 1BTL algebra with generators \(e_0\) and \(e_1\) satisfy the relations [8,9]

\[e_0^2 = \frac{\sin(\omega)}{\sin(\omega + \gamma)} e_0 \quad e_1^2 = \left[ 2 \right] e_1 \quad e_0 e_1 e_0 = e_1, e_1 e_0 e_1 = e_1, e_0 e_2 e_0 = e_2, e_2 e_0 e_2 = e_2, e_0 e_1 e_2 = e_2, e_2 e_0 e_1 = e_1, e_2 e_1 e_2 = e_2, e_1 e_2 e_0 = e_2.

A particular case with N=3 is the 2BTL\(_3\)(q) algebra with generators \(e_0\), \(e_1\), \(e_2\) and \(e_3\) with the relations [8,9]

\[e_0^2 = \left[ \frac{\omega_1}{\omega_1 + 1} \right] e_0 \quad e_1^2 = \left[ 2 \right] e_1 \quad e_2^2 = \left[ 2 \right] e_2 \quad e_3^2 = \left[ \frac{\omega_1}{\omega_2 + 1} \right] e_3.

A matrix representation for 2BTL\(_3\)(q) algebra is defined via the relations [8,9]
The 2BTL algebra also admits the diagrammatic representation in figure 1 [9]

\[ \begin{array}{cccccc}
1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
\end{array} \]

\[ \begin{array}{cccccc}
1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
\end{array} \]

\[ \begin{array}{cccccc}
\frac{q}{q^{-1}+w_{1}} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{q}{q^{-1}+w_{1}} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{q}{q^{-1}+w_{1}} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{q}{q^{-1}+w_{1}} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{q}{q^{-1}+w_{1}} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{q}{q^{-1}+w_{1}} \\
\frac{q^{1-w_{1}}}{1-q^{1+w_{1}}} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{q^{1-w_{1}}}{1-q^{1+w_{1}}} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{q^{1-w_{1}}}{1-q^{1+w_{1}}} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{q^{1-w_{1}}}{1-q^{1+w_{1}}} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{q^{1-w_{1}}}{1-q^{1+w_{1}}} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{q^{1-w_{1}}}{1-q^{1+w_{1}}} \\
\frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} \\
\frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} \\
\end{array} \]

\[ \begin{array}{cccccc}
\frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} \\
\frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{q-q^{-2}w_{2}} \\
\end{array} \]

The diagrammatic representation for the 2BTL algebra.

This diagrammatic representation of the 2BTL algebra is called here the two-boundary Brauer diagram or the two-boundary Kauffman diagram or the two-boundary tangle. The 2BTL algebra gives origin to the two-boundary Temperley-Lieb category and to the two-boundary tangle category. Some examples of tangles and their generalizations are illustrated in figure 2 [6, 9].
4. Quantum Computing via the two-boundary Temperley-Lieb algebra

The physical referent is here the one-dimensional anisotropic spin-1/2 Heisenberg model (the XXZ quantum chain) [8,9,10]. This model is a paradigm for integrable systems, an interesting model to describe experimental data and an attractive model for building a quantum computer. The quantum chain has the quantum symmetry \( U_q(SU(2)) \) and recently was proved that an alternative understanding of the properties of the chain can be obtained from an algebraic point of view when the terms appearing in the Hamiltonian are the generators of the Temperley-Lieb (TL) algebra and its generalizations such as the 1BTL algebra (blob algebra) [8], the 2BTL algebra [9] and the bubble algebra [10].

4.1. Extended Dirac formalism in the extended TL algebras

For the 2BTL algebra is possible to design a Dirac representation in terms of bras, kets, brackets and ketbras. When the graphs in figure 1 are considered like Feynman graphs according with the ideas originally proposed by Kauffman and Lomonaco [2,5,6]. For the 2BTL algebra with have two kinds of particles named respectively “bulk” and “boundary” particles. A single -particle is denoted by \( p \) and one bulk-fluid is denoted by \( p \otimes p \otimes \ldots \otimes p \otimes p \). There are two kinds of boundary-particles denoted respectively by \( b_0 \) and \( b_3 \). Apart of the bulk and boundary particles we assume the existence of a vacuum denoted \(*\). The elementary process are: annihilation into the vacuum of a pair of bulk-particles, creation from the vacuum of pair of bulk particles, disintegration of bulk-particles in boundary particles, disintegration of boundary particles in bulk-particles. The Dirac representation from the 2BTL algebra, resulting from the elementary process, is as follows.

\[
\begin{align*}
\Theta &\leftrightarrow \Theta \otimes \Omega = \Theta \otimes \Omega \leftrightarrow * \\
\cap &\leftrightarrow \Omega \otimes p \rightarrow * \\
\cup &\leftrightarrow \Theta \otimes * \rightarrow p \\
- &\leftrightarrow \Omega_{b,0} : p \rightarrow b_0 \\
\wedge &\leftrightarrow \Theta_{b,0} : b_0 \rightarrow p \\
\c &\leftrightarrow \Omega_{b,3} : p \rightarrow b_3 \\
\vee &\leftrightarrow \Theta_{b,3} : b_3 \rightarrow p
\end{align*}
\]

The first column of equations directly gives the elementary process while the second column of equations gives the secondary effects like vacuum-vacuum transition, propagation -dispersion of bulk-particles and virtual propagation of boundary particles. As we observe the flavor of the boundary particles remains unaltered.

When the particles carry on colors, the Dirac formalism assumes the form: [10]
It is direct to very that the Dirac representation satisfies the Temperley-Lieb relations of 2BTL algebra as the following calculations indicate.

\[ U^2 = U = \mathcal{O} = \mathcal{O} = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = U^{(\alpha, \beta)} \]

\[ U_{b,0}^2 = \mathcal{O} = \mathcal{O} = \langle \Omega_{b,0} \rangle \langle \theta_{b,0} \rangle \langle \Omega_{b,0} \rangle \langle \theta_{b,0} \rangle = \langle \Omega_{b,0} \rangle \langle \theta_{b,0} \rangle \langle \Omega_{b,0} \rangle \langle \theta_{b,0} \rangle = \langle \Omega_{b,0} \rangle \langle \theta_{b,0} \rangle \langle \Omega_{b,0} \rangle \langle \theta_{b,0} \rangle = U_{b,0} \]

\[ U_{b,3}^2 = \mathcal{O} = \mathcal{O} = \langle \Omega_{b,3} \rangle \langle \theta_{b,3} \rangle \langle \Omega_{b,3} \rangle \langle \theta_{b,3} \rangle = \langle \Omega_{b,3} \rangle \langle \theta_{b,3} \rangle \langle \Omega_{b,3} \rangle \langle \theta_{b,3} \rangle = \langle \Omega_{b,3} \rangle \langle \theta_{b,3} \rangle \langle \Omega_{b,3} \rangle \langle \theta_{b,3} \rangle = U_{b,3} \]

\[ U^{(\beta, \gamma)} U^{(\alpha, \beta)} = \]

A faithful representation of the 2BTL algebra is obtained from the Dirac formalism, according with the assignations:

\[ U \cdot \mathcal{O} = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \mathcal{O} = \mathcal{O} = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = U^{(\alpha, \beta)} \]

\[ U \cdot \mathcal{O} = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \mathcal{O} = \mathcal{O} = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = \langle \Omega \rangle \langle \theta \rangle \langle \Omega \rangle \langle \theta \rangle = U^{(\alpha, \beta)} \]

4.2. Yang-Baxterization of the extended TL algebras
The standard Temperley-Lieb generators are able to produce Yang-Baxter operators and for hence quantum gates according with Kauffman and Lomonaco. Similarly the generators of the 2BTL algebra are able to produce both Yang-Baxter operators as Reflection operators [8,9].

For a standard Temperley-Lieb generator we have the following interpretation in the Dirac formalism [10]:

\[
 e = \left( \bigcup \right) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & q & 1 & 0 \\ 0 & 1 & q^{-1} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = (\bigcup) \otimes (\cap) = \begin{pmatrix} 0 \\ q^\frac{1}{2} \\ q^{-\frac{1}{2}} \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & q^\frac{1}{2} & q^{-\frac{1}{2}} & 0 \end{pmatrix}
\]

\[
 = |K\rangle\langle B|
\]

Associated with this Temperley-Lieb generator it is possible to define the following quantum Yang-Baxter operator: [10]

\[
 \bar{R}(u) = \frac{\sin(\lambda - \mu)}{\sin(\lambda)} I + \frac{\sin(\mu)}{\sin(\lambda)} e = \frac{\sin(\lambda - \mu)}{\sin(\lambda)} (\bigcup) + \frac{\sin(\mu)}{\sin(\lambda)} (\cap)
\]

\[
 = \frac{\sin(\lambda - \mu)}{\sin(\lambda)} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \frac{\sin(\mu)}{\sin(\lambda)} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & q & 1 & 0 \\ 0 & 1 & q^{-1} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}
\]

\[
 = \begin{pmatrix} \frac{\sin(\lambda - \mu)}{\sin(\lambda)} & 0 & 0 & 0 \\ 0 & \frac{\sin(\lambda - \mu)}{\sin(\lambda)} + q \frac{\sin(\mu)}{\sin(\lambda)} & \frac{\sin(\mu)}{\sin(\lambda)} & 0 \\ 0 & \frac{\sin(\mu)}{\sin(\lambda)} & \frac{\sin(\lambda - \mu)}{\sin(\lambda)} + q^{-1} \frac{\sin(\mu)}{\sin(\lambda)} & 0 \\ 0 & 0 & 0 & \frac{\sin(\lambda - \mu)}{\sin(\lambda)} \end{pmatrix}
\]

where \( u \) is the spectral parameter and \( \lambda \) is a free parameter. This quantum Yang-Baxter operator can be considered as an entangling quantum gate from which is possible to derive the Zhang-Kauffman-Ge Hamiltonian [3,4] for the quantum evolution of entangled states for two qubits.

Now we consider the following 16-dimensional representation of the generalized colored Temperley-Lieb generators [10]:

302
From this representation it is possible to derive the following quantum Yang-Baxter operator [10]:
\[ \tilde{R}(\mu) = \frac{\sin(\lambda - \mu) \sin(3\lambda - \mu)}{\sin(\lambda) \sin(3\lambda)} \begin{pmatrix} (\quad < \quad) + (\quad > \quad) + \frac{\sin(3\lambda - \mu)}{\sin(3\lambda)} \begin{pmatrix} (\quad < \quad) + (\quad > \quad) \end{pmatrix} \\
- \frac{\sin(\mu) \sin(2\lambda - \mu)}{\sin(\lambda) \sin(3\lambda)} \begin{pmatrix} (\quad > \quad) + (\quad < \quad) \end{pmatrix} + \frac{\sin(\mu)}{\sin(3\lambda)} \begin{pmatrix} (\quad > \quad) + (\quad < \quad) \end{pmatrix} \\
+ \frac{\sin(\mu) \sin(3\lambda - \mu)}{\sin(\lambda) \sin(3\lambda)} \begin{pmatrix} (X_1) + (X_2) \end{pmatrix} \end{pmatrix} \]

Again this last quantum Yang-Baxter operator can be considered as an entangling quantum gate from which is possible to derive the Zhang-Kauffman-Ge Hamiltonian [3,4] for the quantum evolution of GHZ for four qubits.

4.3. Yang-Baxterized Cohomological GHZ States

The quantum Yang-Baxter operators can be used to generate GHZ states from cohomological states derived from knots, links, graphs and algebraic varieties. As an example we use here the quantum Yang-Baxter operator given by [3]

\[ R(x) = \begin{bmatrix} q - \frac{x}{q} & 0 & 0 & 0 \\
0 & q - \frac{1}{q} & 1 - x & 0 \\
0 & 1 - x & q - \frac{1}{q} & 0 \\
0 & 0 & 0 & q x - \frac{1}{q} \end{bmatrix} \]

The cohomological states to be used here are the corresponding to the complex projective space denoted CP_3. This space has three complex dimensions or equivalently six real dimensions. The total Todd class for CP_3 has the form

\[ Td(CP_3) = 1 + 2 t + \frac{11}{6} h^2 t^2 + h^3 t^3 \]

and the Chern character for the line bundle O(nh) over CP_3 is given by

\[ ch(O(nh)) = 1 + n h t + \frac{1}{2} n^2 h^2 t^2 + \frac{1}{6} n^3 h^3 t^3 \]

The total Todd class Td(CP_3) and the Chern character ch(O(nh)) can be represented as vectors according with

\[ v(Td(CP_3)) = \begin{bmatrix} 1 \\
2 t \\
11 h^2 t^2 \\
h^3 t^3 \end{bmatrix}, \quad v(O(nh)) = \begin{bmatrix} 1 \\
n h t \\
n^2 h^2 t^2 \\
n^3 h^3 t^3 \end{bmatrix} \]

These vectors can be transformed via the quantum Yang-Baxter operator R(x) according with

\[ v(Td(CP_3), R(x)) = R(x) v(Td(CP_3)) \quad v(O(nh), R(x)) = R(x) v(O(nh)) \]
explicitly we have

\[
\nu(Td(CP_3), R(x)) = \begin{cases} 
2 \left( q - \frac{1}{q} \right) x h t + \frac{11}{6} (1 - x) h^2 r^2 \\
2 (1 - x) h t + \frac{11}{6} \left( q - \frac{1}{q} \right) h^2 r^2 \\
\left( q - \frac{1}{q} \right) h^3 r^3
\end{cases},
\nu(O(nh), R(x)) = \begin{cases} 
\left( q - \frac{1}{q} \right) x h t + \frac{(1 - x) a^2 h^2 r^2}{2} \\
\left( q - \frac{1}{q} \right) x h t + \frac{(1 - x) a^2 h^2 r^2}{2} \\
\left( q - \frac{1}{q} \right) a h^3 r^3
\end{cases}.
\]

These \((x, q)\)-deformed cohomological states can be considered as entangled quantum states for two qubits; and certain quantum algorithm is devised for the computation of integrality theorems of the form

\[
\int \left[ Td(CP_3', R(x)) \, ch(O(nh), R(x)) \right]_{\text{top-form}} dCP_3 = -1 + \frac{11 n q x^2}{6} + \frac{11 n}{6} + n^2 - \frac{n^3}{6} + q^2 x + n^2 q^2 x - n^2 q x^2 - 4 n^2 x + n^2 q - \frac{22 n x}{3} + \frac{11 n q}{6} + \frac{11 n q^2 x}{6} + \frac{11 n x^2}{6} - \frac{n^3 x^2}{6} + \frac{n^3 x^2}{6} + \frac{x^2}{q^2} + \frac{n^3 x}{6 q} + \frac{x^2}{q^2} + \frac{n^3 x}{6 q^2} + \frac{n^3 x}{6 q^2} + \frac{n^3 x}{q^2}
\]

where in particular, we have

\[
\int \left[ Td(CP_3', R(1)) \, ch(O(nh), R(1)) \right]_{\text{top-form}} dCP_3 = \left( q - 1 \right)^2 \left( q + 1 \right) \left( n + 3 \right) \left( n + 2 \right) \left( n + 1 \right)
\]

\[
\frac{1}{6 q^2}.
\]

5. Conclusions

The generalized TL algebras such as the 2BTL algebra and the Bubble algebra actually display enormous relevance for quantum computation. In this work three examples of such relevance were presented, namely, Dirac formalism, Yang-Baxterization and Yang-Baxterized cohomological polynomials. An important conclusion derived here is that the generalized TL algebras contains extended keys for teleportation and it is evident when the Dirac formalism is recovered from the generalized TL algebras. Other important conclusion is that certain Zhang-Kauffman-Ge hamiltonians can be derived from quantum Yang-Baxter operators associated with generalized TL algebras and for hence the generalized TL algebras are dynamical algebras for determined physical referents such as quantum spin chains. A final important conclusion is the generalized TL algebras can provide quantum algorithms for the computation of Euler characteristics associated with homology complex like singular and Khovanov and then will be possible to obtain generalizations of the Aharonov-Jones-Landau [15] and the Aharonov-Arad-Eban-Landau algorithms [16].
References


Random Diffusion and Congestion on Weighted Complex Networks

Xin Jiang\(^1\), Shaoting Tang\(^1\), Lili Ma\(^1\), Zhanli Zhang\(^2\) and Zhiming Zheng\(^1\)

\(^1\) LMIB and Department of Mathematics, Beihang University, 100191, Beijing, China
\(^2\) LMIB and School of Mathematical Sciences, Peking University, 100871, Beijing, China

jiangxin@ss.buaa.edu.cn

Abstract. Here we present an approach to systematically study random diffusion and congestion in the structure of weighted networks. Exact solutions for stationary distributions of diffusing walkers and the mean first passage time (MFPT) are obtained analytically. To characterize the coverage of the space by the diffusion, we study the random walk exploration property, the diffusion coverage \(S(t)\). The experiential equation for \(S(t)\) is obtained, which reveals the time dependence of the diffusion speed. Finally, to address the congestion problem, we intergrade diffusion properties and network capacity, taking into account the weight distribution. A dynamic phase transition is observed in the congestion phenomenon on networks through simulations. Relatively, we propose a new characteristic parameter \(L_{\text{time}}\) to measure the network tolerance to congestion. A possible method for alleviating congestion on weighted networks is raised.

Key words: Complex weighted networks · Random walk · Scale-free network · ER network · Diffusion · Congestion

1 Introduction

Complex systems can emerge from a mass of interdependent and interacting elements [1]. Networks, in forms of vertices joined in pairs by edges, have been extensively used for understanding various artificial or real-world complex systems. Usually, the elements are denoted by nodes and their interactions are represented by edges [2].

Many real-world systems, including Internet, WWW, social networks, and biological networks of various kinds, have been studied intensively [3, 4]. The macroscopic statistical properties of the network topology have attracted significant attention from scientific communities, which have led to many interesting results, such as the scale-free feature [5] and the small-world phenomenon [6]. Idealized network models being studied in the physics literature are binary in

\* This work was partially supported by the National Key Basic Research Project of China Grant No.2005CB321902
nature; that is, the edges between nodes are either present or not. Such a network with \( N \) vertices can be represented by a binary adjacent matrix \((A_{ij})_{N \times N}\), where \( A_{ij} = 1 \) if \( i \) and \( j \) are connected, else, 0. However, in many real networks, the edges may intrinsically have different strengths or capacities, that is, the edges are weighted [7]. For example, for social networks, social ties between individuals may differ from each other. As to real world Internet, the amount of data packets exchanged between two routers may be more or less [8]. Thus, it is improper to represent real-world systems without considering interaction strengths between different pairs of nodes. Generally, these weighted networks tend to exhibit more complex behaviors than the unweighted ones.

Recently, there have been many studies on such weighted complex networks [9, 10]. Many researchers have pointed out that, except for topology, networks also can be specified by the dynamical process taking place on them [11]. For instance, the power-law behavior of the load distribution \( P_L(l) \sim l^{-\delta} \) was observed on scale-free networks, where the load is defined as the level of burden on vertices in the shortest path based on diffusion, with the load exponent \( \delta \). Previous researches show that diffusion is one of the most common phenomenons on networks. Although diffusion on unweighted networks, such as Random Walks [11], Zero-Range Processes [12], has been well studied, the effect of disorder on diffusion on weighted networks is still an open question. As the first step, the diffusion properties can be studied through walking randomly on weighted networks with various routing strategies [13]. On second thoughts, one of the direct results caused by the disorder of diffusion is congestion. For example, Internet may endure a traffic jam while there is a sudden influx of information packets swarming into only a few routers [14]. Whether the diffusions on networks are free, or uncongested, is critical for the basic and efficient functioning of transport network systems such as, the power grids, and transportation networks, etc. Thus, to ensure the free diffusing, congestion control is of great importance.

In this paper, we concentrate on the general random diffusion properties on weighted networks and explore the efficient congestion control strategies. We propose a weight based local transport strategy for diffusion and present the random diffusion model in terms of biased random walk. Stationary distributions for diffusion walkers together with the mean first passage time (MFPT) for this process are derived using the mean field theory. Numerical simulations on scale-free networks confirm our results. Also, we study the diffusion coverage \( S(t) \), which is a measure for diffusion as it characterizes the time dependence of the diffuse speed. We present the experiential equation for \( S(t) \) both in Erdős-Rényi (ER) and scale-free (SF) weighted networks with a typical weight distribution \( w_{ij} \equiv \exp(-ax_{ij}) \). Here, parameter \( a \) controls the broadness "strength" of the disorder and directs us to focus on the phase transition from free diffusion to congestion. Finally, we give a constructive strategy for congestion control on \( \exp(-ax_{ij}) \) weighted networks.

In Sec 2 we propose our diffusion model and present our theoretical and numerical analysis for diffusion properties. Sec 3 describes the Monte Carlo simulation procedure for \( S(t) \) and the quantities that we compute. Sec 4 describes
the congestion phenomenon and gives strategies for congestion control. Results and discussions are reported in Sec 5.

2 Random Diffusion on Weighted Networks

In this paper we consider finite undirected network denoted by \( G = (V, E) \), where \( V = \{1, 2, \ldots, N\} \) is the vertex set and \( E = \{(i, j) | i, j \in V \} \) is the edge set. Each edge \((i, j)\) is supposed to have a corresponding weight \( w_{ij} \). Mathematically, a weighted network can be represented by an adjacency matrix \( W \) with entries \( W_{ij} \) which are equal to the weights on the edges, if \((i, j) \in E\), \( W_{ij} = w_{ij} \), else, \( W_{ij} = 0 \). We assume throughout this paper that all weights are nonnegative. As we only consider diffusion of entities on networks, this assumption is reasonable. We also assume the network to be connected, that is, for any two nodes, we can find a series of adjacent edges to connect them. A main statistical parameter of network topology is the degree distribution \( P_k \) which characterize the probabilities \( p_k \) that a randomly chosen vertex has \( k \) neighbors , i.e., the node has degree \( k \). Our theory is valid for any distribution \( p_k \), so we do not make any assumption about it.

In real-world weighted networks, the weight distribution, besides the topology structure, plays a leading role in guiding a diffusion process. This phenomena is quite common in real networks, for example, a surf on Internet, which can be thought as a random information diffusion, is more likely to visit famous web sites with large bandwidth links. Starting from this point, we propose a local weight based transition strategy for our random diffusion model which is based on random walk process. Then the diffusion, a discrete stochastic process, is defined as a walk along the edges of a given network.

Let us consider a random walker, or an information packet that diffuses at discrete time: a packet at node \( i \) and time \( t \) will choose one of its neighbors \( j \) to hop to with certain transition probability \( t_{ij} \) at time \( t + 1 \). The local weight based transition strategy is defined as:

\[
t_{ij} = \frac{W_{ij}^\alpha}{\sum_{l=1}^{N} A_{il} W_{il}^\alpha}.
\]

Here, for convenient operation, we introduce \( A_{il} \), the element of the adjacency matrix \( A \) as we described above, which is, indeed, redundant in form. \( \alpha \) is a free adjustable parameter and \( W_{ij}^\alpha \) characterizes the tendency based on weight for the packet hops from node \( i \) to node \( j \). For \( \alpha = 0 \), \( t_{ij} \) is just the transition probability for pure random walk regardless of the weights. Suppose the packet starts at node \( i \) at time \( t = 0 \). We define \( P_{ij}(t) \) as the probability that we can find the packet at node \( j \) at time \( t \). Then the master equation for this transition probability is:

\[
P_{ij}(t + 1) = \sum_{l=1}^{N} A_{lj} t_{lj} P_{il}(t).
\]
Let us denote $P_j^\infty = \lim_{t \to \infty} P_{ij}(t)$. For the infinite time limit, stationary solution $P^\infty$ follows:

$$P_j^\infty = \sum_{l = 1}^{N} A_{lj} t_{lj} P_l^\infty .$$

(3)

Here, the stationary distribution of this random diffusion process can be understood by means of a simple dynamic mean-field argument:

$$P_j^\infty \simeq \sum_{l = 1}^{N} \langle A_{lj} \rangle \langle t_{lj} \rangle P_l^\infty .$$

(4)

For uncorrelated complex networks, $\langle A_{lj} \rangle$ has been proposed [15] to be $\langle A_{lj} \rangle = \frac{k_i k_j}{\langle k \rangle N}$, where $\langle k \rangle = \sum_{k=1}^{\infty} k P(k)$ is the average degree of the network. Next, to compute the mean-field transition amplitude $\langle t_{lj} \rangle$ between two randomly selected nodes $l$ and $j$, we progress sums over the index $j$,

$$\langle t_{lj} \rangle = \left( \frac{\sum_{m=1}^{N} W_{lm}^\alpha}{\sum_{m=1}^{N} A_{lm} W_{lm}^\alpha} \right) = \left( \frac{\sum_{j=1}^{N} W_{lj}^\alpha}{\sum_{j=1}^{N} A_{lj} W_{lj}^\alpha} \right) = \frac{\sum_{j=1}^{N} t_{lj}(W_{lj}^\alpha)}{k_i k_j W_{lj}^\alpha}.$$  

(5)

Here, $C_j(\alpha) = \sum_{i=1}^{N} W_{ji}^\alpha$ is some kind of the total link weight a vertex has. By inserting the relations into the simplified master Eq (2.4), we therefore are led to the general expression

$$P_j^\infty = \frac{C_j(\alpha)}{L_0 \langle W^\alpha \rangle},$$

(6)

where $L_0 = \langle k \rangle \cdot N = \sum_i k_i$, is the number of edges. In the special case of $\alpha = 0$, we recover the exact result $P_j^\infty = k_j \cdot (L_0)^{-1}$ [11]. Note that $P_j^\infty \propto C_j(\alpha)$, up to normalization, as $\alpha > 0$, the larger total link weight a vertex has to other ones in the network, the more often it will be visited by a diffusing packet. While $\alpha < 0$, we expect that the effect of the total link weight decrease, vertices with large total weight may no longer be visited quite often. In order to test Eq (2.6), we simulate this diffusion process described above. We randomly displace M noninteracting walkers on a weighted scale-free network, allowing each walk to perform $N_0$ hops according to $t_{ij}$ with parameter $\alpha = 1$ and $\alpha = -1$, and then monitor the total number of visitations $v_i$ for each node $i$. Figure 1 shows that $V_i$ for each node is proportional to $C_i(\alpha)$, which is in good agreements with our results.

Next, to depict how fast the random walk acts, we present an analytical approach for the form of the Mean First Passage Time (MFPT). The first-passage probability $F_{ij}(t)$ from $i$ to $j$ after $t$ steps satisfies the relation

$$P_{ij}(t) = \delta_{t0} \delta_{ij} + \sum_{t' = 0}^{t} P_{jj}(t - t') F_{ij}(t').$$  

(7)

Using the Laplace transform, the MFPT is given by $\langle T_{ij} \rangle = \sum_{t=0}^{\infty} t F_{ij}(t)$ . Following the results showed in [11], after some algebra we get the average return
The total number of visitations $V$ to each node versus $C(\alpha)$ in two scale-free networks generated by the Molloy-Reed algorithm, with $N = 10^3$ nodes, and an average degree of 6. For $\alpha = -1$ and $\alpha = 1$, $w$ is respectively a random number from the range (0,1) and (0,100). We originally set $M = 5$ and $N_s = 10^8$, which can ensure the system reach the equilibrium phase.

The average return time $\langle T_{ii} \rangle$ indirectly shows the frequency the biased random walk visits vertex $i$, which can be considered as a centrality of vertex $i$ with parameter $\alpha$, that is, the extended degree $C(\alpha)$ quantifies how central a vertex is in the random diffusion on weighted networks according to our transition strategy.

3 Monte Carlo Simulation for Diffusion Coverage

As a property of random diffusion processes, the diffusion coverage $S(t)$, which from a practical point of view, characterizes the mean numbers of distinct vertices visited by random walkers after time $t$ [16]. We first look at a modified Erdős-Rényi network with size $N$. To construct it, we randomly connect each pair of nodes with a connecting probability $p$ at first. In order to make this network connected, we repeat the random procedure for nodes which do not connect to the giant component until the network is connected. Secondly, we consider $S(t)$ on SF networks, in which the degree distribution follows a power law. We employ the Molloy-Reed algorithm to construct a SF network.

Here, we model a weighted network by assigning edges with weight of the following form: $w_{ij} = \exp(-ax_{ij})$ [17], where the parameter $a$ controls the broadness "strength" of the disorder, and $x_{ij}$ is a random number taken from a uniform distribution in the range [0,1]. We use this kind of disorder since recent studies [17] show that many types of disorder distributions of network weight lead to the same universal behavior.
In the first series of simulations, we put one random walker onto one of the four initial sites of the random network and the scale-free network, and at each time step $t$ the walker diffuses with probability $t_{ij}$ with $\alpha = 1$. We average for each realization the results over the four starting points, and then over all network realizations. Figure 2 indicates that the mean number of distinct sites visited is given by $S(t) \propto t$ for intermediate times both in the modified ER network and scale-free networks. Similar results can be found in [16]. For infinite time we get $S(t) = N$ as expected that each node of the network will eventually be visited. However, as figure 2 shows, the spread speed of diffusion in modified ER networks is larger than that in SF networks, which indicates that, for a weight preferred diffusion, random connectivity structure is easier to be detected.

Figure 3 shows the results for several values of the weight distribution parameter $a$. We show that $S(t)$ is almost the same for different values $a$ both in modified ER and SF networks. In this sense, for the weight preferred diffusion, weight distribution shows little effect on the speed of diffusion.

4 Congestion in Diffusion Process

In many real transport systems, diffusion is shadowed with congestion or jammy [14]. The fact is that the nodes of real-world networks do not have infinite processing capacity. Once there are too many particles diffusing on the network, some nodes will be overloaded, causing collective behaviors, such as congestion and jamming. For example, in computer networks, each host is restricted to have a finite storage. Once there are overmany information packets swarm into one host in a short period of time, this host will be overloaded. The following problem is that packets diffusing to overloaded nodes will be discarded or just wait in a infinite queue to be disposed. As the number of overloaded nodes reaches a certain finitude, diffusion on the network is congested. In this sight, the system
has come through a phrase transition from a free diffusing state to a congested one which is mainly manifested as the increasing number of overloaded nodes.

Intuitively, we can numerically analyze the image of the number of overloaded nodes as the entry of packets increases, which allows us to understand in details the system’s congestion behavior. Follow a rational line to do some work well, we assume that each node have an finite processing ability which we define as the processing capacity denoted by $N_c$. In a diffusion process, in a small time scale window, or at a time step $t$, we denote the visit number of particles for a node $i$ is $V_i(t)$. If $V_i(t) < N_c$, the node stays in a free state. With the passage of time, when $V_i(t) > N_c$, the node’s free state turns to a overloaded state.

In this paper, as a first approximation, we define congestion is the state that all the network nodes are overloaded. In figure 2 we show the behavior of the overloaded nodes number $N_o$ as a function of the density of particles $\rho$ in both ER and SF networks. Here $\rho$ is defined as $\rho = M/N$, where $M$ is the number of particles diffusing on the network and $N$ is the total number of nodes. Simulation results (red curve) suggest that there exist a dynamic phase transition at a threshold density $\rho_c$. We show that the threshold density $\rho_c$ is always smaller than the capacity of each node $N_c$. Note that the threshold differs with the topological structure of the network, as we show in figure 2. The emerge of the overloaded nodes is ineluctable in our model for the number of particles in network will not decrease. However, the length of the time window $L_{time}$, which is defined as the time from the moment the overloaded nodes emerge to the moment all the nodes are overloaded, can be controlled by redistributing weights on network edges. The primary goal of our congestion control is to increase $L_{time}$, the length of the time window, with respect the weight distribution. Thus we focus on examining $L_{time}$ for random and scale-free networks. In particular, we are interested in networks with the weight distribution $w_{ij} = \exp[-ax_{ij}]$ as mentioned in Sec 3. Here the diffusion strategy is defined as Eq (2.1) with $\alpha = 0$.

First, we give simulation results for random graphs. Figure 4 shows the num-
Fig. 4. Number of overloaded nodes $N_0$ vs $\rho$ with different parameter $a$ in random graphs of size $N = 10^3$ and the connection probability $p = 0.5$.

The number of overloaded nodes $N_0$ versus the density of packets $\rho$ in random networks with different weight distributions as $a$ varies. Here, $a = 0$ is correspond to the pure random diffusion case where all the edge weight are equal to 1. We see that, for all cases considered, $L_{\text{time}}$ for $a = 0$ is minimum. It increases as $|a|$ becomes larger than 0. We also observe that the complete congestion time $t_c$ can be delayed as $a > 0$, $L_{\text{time}}$ for $a = 5$ is larger than that for $a = 10$. While for $a < 0$, $t_c$ is almost the same. As a result, the network can be more tolerant to diffusion congestions for larger density of packets with $a > 0$. Then we point out that, in order to alleviate the congestion on random networks, the parameter $a$ for weight distribution $w(a)$ should be set with a small positive number. Figure 5 also indicates that, in order to prolong $L_{\text{time}}$, a small positive value of the weight parameter $a$ is required for scale-free networks. However, $L_{\text{time}}$ is much larger in scale-free network than it in random ones. This means that scale-free networks are significantly more robust to traffic congestion. This is because, in scale-free networks, the most congested nodes have large number of links, packets tends to congregate on these nodes while other nodes can remain a free state. As an expectation, we propose a diffusion strategy considering the interactions among the diffusion entities. Actually, the transition capacity for a node is neither infinite nor changeless. Take the transaction ability of each node into account, we introduce the interacting particle system. In the interacting particle systems, microscopic interactions influence the macroscopic properties of the steady states of the diffusion processes. However, systems of the interacting kind, cannot be described by random diffusion. We hope to find an optimized transport strategy in interacting systems lead to the maximum critical creation rate of packets with the minimum transition time. This kind of model, defined on a finite, undirected network, can describe a diffusion process in which many packets hoping from node to node with different hop rates and transition probabilities. The hop rate, characterized by parameter $\delta$, represents the transition ability of a node, and the transition probability, characterized by parameter $\alpha$, represents the congestion-aware diffusion strategy that defines the degree-dependent of congestion awareness. We can encapsulate the transition probability matrix $(\omega_{ij})$
as
\[
\omega_{ij} = \begin{cases} 
q_j(m)/m \cdot p_{ij}, & \text{for } i \neq j; \\
1 - q_j(m)/m, & \text{for } i = j.
\end{cases}
\]

where \( p_{ji} = \frac{k_i^\alpha}{\sum_{m} a_{im}^k W_i^\alpha} \), and \( q_i(m) = m^\delta \). By adjusting the two factors we mentioned above in Eq (4.1), we can get an optimized routing strategy which can increase the critical generating rate of packets, decrease the transit time, promote the efficiency of transaction, and reduce the congestion.

![Figure 5](image-url)

**Fig. 5.** Number of overloaded nodes \( N_0 \) vs \( \rho \) with different parameter \( a \) in scale-free networks with size \( N = 10^3 \) and average degree \( \langle k \rangle = 6 \).

5 Conclusion

In this paper we have addressed the topic of dynamics and congestion control on complex weighted networks. Our motivation comes from the desire to understand the influence of both the weight distribution and topological structure on the diffusion processes on networks.

We consider general network structures to couple with a simple diffusion model determined by the weight \( w \) on each edge and a parameter \( \alpha \) to describe the preference of the weight. We present theoretical and numerical analysis and results for the stationary distribution and the average return time of random diffusion process. Moreover, by means of the analysis of the mean return time (MRT), we have been able to recognize the leading role played by the extension of conceptual degree \( C(\alpha) \) of the target vertex in general weighted network structures. We also study the random walk exploration property, the diffusion coverage \( S(t) \), in a representative weighted network. The experiential equation for \( S(t) \) reveals the time dependence of the diffusion speed. To gain deep insight of this property, general weighted cases are expected to be studied.

As a practical application, using our diffusion strategy and weight distribution, we investigate the congestion phenomenon on both ER and SF networks. Simulation results suggest that there exist a dynamic phase transition.
at a threshold density of the number of packets. Accordingly, we propose a new characteristic parameter $L_{time}$ to measure the network tolerance to congestions. In our cases, scale-free networks are more tolerant to congestion than random networks. This is further supported by examining the results of networks with the weight control parameter $a$ changes. All these results suggest a possible method for alleviating congestions on weighted complex networks.

References

The Complexity of Computable Conditional Probability

Nathanael L. Ackerman\textsuperscript{1}, Cameron E. Freer\textsuperscript{2}, and Daniel M. Roy\textsuperscript{3}

\textsuperscript{1} Department of Mathematics
University of Pennsylvania
nate@aleph0.net

\textsuperscript{2} Department of Mathematics
Massachusetts Institute of Technology
freer@math.mit.edu

\textsuperscript{3} Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
droy@csail.mit.edu

Given a computable random variable, one can easily compute its conditional expectation with respect to a discrete random variable (i.e., a \(\sigma\)-algebra of positive measure events). For measure zero events, the situation is much more complicated and constructive methods have remained limited to special cases [Rao88].

Using computable analogues of results from nonstandard analysis [Ros07], we give an upper bound on the computational power needed to compute the conditional expectation of a computable random variable.

We also discuss negative results which code noncomputable reals into conditional expectations with respect to computably defined \(\sigma\)-algebras. Finally, we describe a framework for computable probability theory which uses results from nonstandard analysis via computably saturated real closed fields (as constructed in [DKS09]).


Computably Categorical Boolean Algebras
Enriched by Ideals and Atoms

Pavel Alaev
Novosibirsk, Russia
alaev@math.nsc.ru

A computable structure $\mathfrak{A}$ is computably categorical (autostable) if for every its computable copy $\mathfrak{B}$, there is a computable isomorphism $f : \mathfrak{A} \rightarrow \mathfrak{B}$. Here we formulate an algebraic description for computably categorical structures of the form

$$\mathfrak{A} = (\mathfrak{A}^*, I_1, \ldots, I_\lambda, At_{I_1}, \ldots, At_{I_\mu}),$$

where $\mathfrak{A}^*$ is a Boolean algebra, $I_1, \ldots, I_\lambda$ are predicates for some ideals in $\mathfrak{A}^*$, and $At_{I_j}$ are predicates for the sets $\{x \in \mathfrak{A}^* \mid x/I_j \text{ is an atom of } \mathfrak{A}^*/I_j\}$. We will call them $I_\lambda,\mu$-algebras. Earlier, such descriptions were found for many partial cases of $I_\lambda,\mu$-algebras.

For every $\lambda, \mu \in \omega$, $\mu \leq \lambda$, we define a finite set of $I_\lambda,\mu$-algebras $\mathfrak{A}_1, \ldots, \mathfrak{A}_n$, which will be called stable, so that the following theorem holds.

**Theorem.** A computable $I_\lambda,\mu$-algebra is computably categorical if and only if it is isomorphic to a finite direct product of algebras in the set $\{\mathfrak{A}_1, \ldots, \mathfrak{A}_n\}$.

To define the notion of a stable $I_\lambda,\mu$-algebra $\mathfrak{A}$, we need a few additional terms. Let $x \in \mathfrak{A}$. We define its characteristic $P_x : \{1, \ldots, \lambda\} \rightarrow \{0, 1, 2\}$ so that

$$P_x(j) = \begin{cases} 0, & \text{if } x \in I_j, \\ 1, & \text{if } x \in At_{I_j} \text{ and } j \leq \mu, \\ 2, & \text{if } x \not\in I_j, At_{I_j}. \end{cases}$$

We say that $P_x \leq P_y$ if $P_x(j) \leq P_y(j)$ for all $j \leq \lambda$. An element $x$ is decomposable if $x = 0$ or $x = x_1 + \ldots + x_n$, where $P_{x_i} < P_x$ for all $i \leq n$. If $P', P''$ are two characteristics then their sum $P' + P''$ is defined so that $P' + P''(j) = \min\{2, P'(j) + P''(j)\}$. By $L_P$ we denote the ideal $\{x_1 + \ldots + x_n \mid P_x < P\}$.

An $I_\lambda,\mu$-algebra $\mathfrak{A}$ is stable if the following hold:

1. $1$ is an indecomposable element in $\mathfrak{A}$;
2. for each characteristic $P$, one of the following three cases holds:
   (i) every element of characteristic $P$ is decomposable in $\mathfrak{A}$;
   (ii) every non-zero element of characteristic $P$ is indecomposable in $\mathfrak{A}$;
   (iii) $P_1 = P$ and $1/L_P$ is an atom.
3. if $P, Q$ are characteristics such that $P + Q = Q$, and $\mathfrak{A}$ contains indecomposable elements of characteristic $P$, then for each indecomposable element of characteristic $Q$, there is an indecomposable element of characteristic $P$ under it.
4. if $P$ is a characteristic such that $P + P = P$, then one of the following holds:
   (i) $x/L_P$ is atomless for every element $x$ of characteristic $P$;
   (ii) $P_1 = P$ and $1/L_P$ is an atom.
Comparing Nontriviality for the Exponential Time Classes E and EXP

Timur Bakibayev
University of Heidelberg
Institut für Informatik
Im Neuenheimer Feld 294
D-69120 Heidelberg
Germany

Lutz (1995) calls a set $A$ \emph{weakly complete} for a complexity class $C$ if a \emph{nonnegligible} part of $C$ can be reduced to $A$ (by a polynomial-time many-one reduction). For the exponential-time classes $E = \text{DTIME}(2^{O(n)})$ and $\text{EXP} = \text{DTIME}(2^{\text{poly}(n)})$, Lutz formalized this idea by introducing resource bounded measures on these classes and by saying that a subclass of $E$ ($\text{EXP}$) is negligible if it has measure 0 in $E$ ($\text{EXP}$).

We generalize Lutz's weak completeness notions for the exponential-time classes by calling a set $A$ \emph{E-nontrivial} if, for any $k \geq 1$, there is a set $B \in E \setminus \text{DTIME}(2^{kn})$ such that $B \leq^p_m A$, and by calling a set $A$ \emph{EXP-nontrivial} if, for any $k \geq 1$, there is a set $B \in \text{EXP} \setminus \text{DTIME}(2^{n^k})$ such that $B \leq^p_m A$.

As one can easily show, any E-complete set is weakly E-complete, any weakly E-complete set is E-nontrivial, and any E-nontrivial set is intractable but none of these implications can be reversed (and, similarly, for EXP in place of E).

While, for sets in E, E-completeness and EXP-completeness coincide, weak E-completeness is strictly stronger than weak EXP-completeness (Juedes and Lutz, Ambos-Spies, Terwijn and Zheng).

In case of the still weaker nontriviality notions we get the following independence result: For sets in E, neither E-nontriviality implies EXP-nontriviality nor EXP-nontriviality implies E-nontriviality. Moreover, there is a weakly EXP-complete set which is not E-nontrivial.

(This is joint work with Klaus Ambos-Spies.)
Abstract. A novel degree structure – the detour degrees – defined by Kristiansen & Voda in [K&V08] is generalised – or relativised – spawning interesting new structures. These structures are related to sub-recursive hierarchies, both intra-structurally and – by the virtue of the relativisation – inter-structurally. The main results presented pertain to inter-structural results. We also present an improved version of an important lemma from [K&V08], and outline directions for further research.

References

The Limits of Decidability and Tractability in Growth-Rate Analysis for Programs

Amir ben-Amram¹
Lars Kristiansen²

¹ Tel-Aviv Academic College, Tel-Aviv
   benamram.amir@gmail.com
² Department of Mathematics, University of Oslo
   larsk@math.uio.no

In our work on static growth-rate analysis of programs, we have encountered a family of decision problems that we find interesting and amusing. These problems regard first-order imperative programs manipulating natural numbers stored in registers. A program will work on a fixed numbers of registers $x_1, \ldots, x_n$. The numbers $a_1, \ldots, a_n$ stored in the registers when the execution of the program starts are regarded as inputs, and the numbers $b_1, \ldots, b_n$ stored in the registers when the execution terminates are regarded as outputs. For each language $L$, we have the following decision problem.

(The feasibility problem of $L$) Input to the problem: a program $p$ (written in the language $L$). Question: Is every output $b_i$ (for $i = 1, \ldots, n$) of the program $p$ bounded by a polynomial in the inputs $a_1, \ldots, a_n$? Answers: Yes or No.

When the language $L$ yields full Turing computability, the feasibility problem of $L$ is of course undecidable. So is the feasibility problem of $L$ when $L$ is capable of computing every primitive recursive function, every Kalmár elementary function, etc.

However, a number of languages which are weaker, but sufficiently expressive to be of interest for growth-rate analysis, have decidable feasibility problems. In CiE 2008, we presented (with Neil Jones) such a language where the problem was even Ptime-decidable. This result was surprising to us, because the problem is in some sense close to the frontier of decidability. Since then, we have continued to explore the limits of decidability and tractability by trying to characterise the complexity of similar problems; in this talk we will survey our current state of knowledge on this topic.
Initial Segments of C.E. ibT-Degrees

Philipp Bodewig

University of Heidelberg
Department of Mathematics and Computer Science
Im Neuenheimer Feld 294
D-69120 Heidelberg
Germany

Paul Fischer has shown that some initial segments of the c.e. wtt-degrees form a lattice.¹ We examine the case of c.e. ibT-degrees, where the use of the reduction is bounded by the identity function instead of any computable function. We show that, for every c.e. ibT-degree \( c > 0 \), there exists a pair of c.e. ibT-degrees \( a, b \) below \( c \) such that \( a \) and \( b \) do not have an infimum.

Thus Fischer’s result does not hold in the c.e. ibT-degrees.

Weihrauch Degrees, Omniscience Principles and Weak Computability

Vasko Brattka\(^1\) and Guido Gherardi\(^2\)

\(^1\) Laboratory of Foundational Aspects of Computer Science  
Department of Mathematics & Applied Mathematics  
University of Cape Town, Rondebosch 7701, South Africa  
\texttt{Vasco.Brattka@uct.ac.za}

\(^2\) Dipartimento di Filosofia, Università di Bologna, Italy  
\texttt{Guido.Gherardi@unibo.it}

In this paper we study a reducibility that has been introduced by Klaus Weihrauch or, more precisely, a natural extension of this reducibility for multi-valued functions on represented spaces. We call the corresponding equivalence classes Weihrauch degrees and we show that the corresponding partial order induces a lower semi-lattice with the disjoint union of multi-valued functions as greatest lower bound operation. We prove that parallelization is a closure operator for this semi-lattice and the parallelized Weihrauch degrees even form a lattice with the product of multi-valued functions as greatest lower bound operation. We show that the Medvedev lattice can be embedded into the parallelized Weihrauch lattice in a natural way, even into the sublattice of total continuous multi-valued functions on Baire space and such that greatest lower bounds and least upper bounds are preserved. As a consequence we obtain that Turing degrees can be embedded into the single-valued part of this sublattice. The importance of Weihrauch degrees is based on the fact that multi-valued functions on represented spaces can be considered as realizers of mathematical theorems in a very natural way and studying the Weihrauch reductions between theorems in this sense means to ask which theorems can be transformed continuously or computably into each other. This allows a new purely topological or computational approach to metamathematics that sheds new light on the nature of theorems. As crucial corner points of this classification scheme we study the limited principle of omniscience LPO, the lesser limited principle of omniscience LLPO and their parallelizations. We recall that the parallelized version of LPO is complete for limit computable functions (which are exactly the effectively \(\Sigma^0_2\)-measurable functions in the Borel hierarchy). We prove that parallelized LLPO is equivalent to Weak König’s Lemma and hence to the Hahn-Banach Theorem in this new and very strong sense. We call a multi-valued function weakly computable if it is reducible to the Weihrauch degree of parallelized LLPO and we present a new proof that the class of weakly computable operations is closed under composition. This proof is based on a computational version of Kleene’s ternary logic. Moreover, we characterize weakly computable operations on computable metric spaces as operations that admit upper semi-computable compact-valued selectors and we prove that any single-valued weakly computable operation is already computable in the ordinary sense.
Independence of Axioms in Non-Classical Logics and ATP

Karel Chvalovský

Charles University, Prague
Institute of Computer Science
Academy of Sciences of the Czech Republic

We study the independence of axioms in Hilbert-style calculi for some propositional non-classical logics, especially for substructural and fuzzy logics, by automated theorem proving methods. The standard approach to solve these problems is encoding in first-order classical logic. We discuss some theoretical as well as practical issues of this approach and present several results e.g. the complete solution for prominent fuzzy logics MTL (Monoidal T-norm based Logic) and Hájek’s BL (Basic Logic).
The Reverse Mathematics of Artinian Rings

Chris Conidis
University of Chicago
Department of Mathematics and Computer Science

We will examine the reverse mathematical strengths of the standard theorems from classical algebra which say that every Artinian ring is Noetherian, and every Artinian ring is of finite length. In particular, we will show that, over RCA\textsubscript{0} (i.e. recursive comprehension axiom) the former lies between ACA\textsubscript{0} (i.e. arithmetic comprehension axiom) and WKL\textsubscript{0} (weak König’s Lemma), while the latter is equivalent to ACA\textsubscript{0} over RCA\textsubscript{0} + B\Sigma\textsubscript{2} (i.e. recursive comprehension, plus \Sigma\textsubscript{2} bounding).

Time permitting, we shall also examine the complexity of radical ideals in noncommutative rings. In particular, we will construct noncommutative rings \( R_1, R_2 \) such that the prime radical of \( R_1 \) is \( \Pi_{1}^{1} \)-complete, and the Levitzki radical of \( R_2 \) is \( \Pi_{2}^{0} \)-complete, thus achieving the best possible upper bounds on the complexity of these radicals.
Modelling Quantum Information by Chu Spaces

Graziana Conte

Università degli Studi di Milano

Category theory is widely applied in computer science, it provides a setting in which elegant functorial techniques became available in order to investigate systems and processes, respectively interpreted as objects of the category and their morphisms. For a symmetric monoidal closed category $\mathcal{A}$ and any object $K$ in $\mathcal{A}$, in an appendix to Barr[1], Chu described a $*$-autonomous category whose objects are known as Chu spaces. The Chu construction was originally motivated by the theory of pairs of topological vector spaces consisting of a vector space and of the space of functionals on it, therefore it is a privileged context for the study of concrete duality. The spaces arising from the study of axiomatic foundations of quantum mechanics for the description of physical systems are partially ordered Banach spaces and their duals, more specifically there is an embedding of the set $E$ of ensembles and the set $L$ of effects of a quantum system into a pair of real ordered Banach spaces $B \times B'$, being $B'$ the dual of $B$. Our approach is to consider the construction of the corresponding category using the technique of Chu spaces and it will be discussed in the light of the potential applications in quantum information theory. In particular, the objective of the talk is to make explicit some of the structural aspects involved during the flow of quantum information in a quantum circuit or during a communication protocol.

References

2. Chu P.H.: Constructing $*$-autonomous categories. appendix to [1]
Phase Transitions Related to the Pigeonhole Principle

Michiel De Smet* and Andreas Weiermann

Ghent University
mmdesmet@cage.ugent.be
weierman@cage.ugent.be

Keywords Ackermann function, pigeonhole principle, Ramsey theory, phase transitions.

Brief description of the talk

The pigeonhole principle is one of the most well-know combinatorial principles, due to both its simplicity and usefulness. It is attributed to Dirichlet in 1834 and also known as the chest-of-drawers principle or Schubfachprinzip. Although known for a long time now, the principle still has interesting properties to reveal. So it is not surprising it gained the attention of several mathematicians lately. Terence Tao, for example, uses the example of the so called “finitary” infinite pigeonhole principle in an essay on his blog. Gaspar and Kohlenbach commented on his ideas and wrote a paper about it.

The pigeonhole principle is an instance of the finite Ramsey theorem for 1-tuples and is interesting with regard to the classical Ramsey for pairs problem in reverse mathematics. Motivated by their simplicity and surprising strength we investigate the recursion-theoretic complexity of two assertions which are related to the pigeonhole principle. We show that resulting density principles give rise to Ackermannian growth. After parameterizing these assertions with respect to a number-theoretic function $f$, we investigate for which functions $f$ Ackermannian growth is still preserved. It will turn out that there is an interesting connection between the threshold functions of both density principles.

* Aspirant Fonds Wetenschappelijk Onderzoek - Flanders
On Least Enumerations of Partial Structures

Angel V. Ditchev

Faculty of Mathematics and Computer Science
Sofia University
5 James Bourchier Blvd.
1164 Sofia, Bulgaria
ditchev@fmi.uni-sofia.bg

Abstract: It will be considered arbitrary partial structures. It is not supposed that either the equality or the inequality is among the predicates of the structure. It will be given a characterization of arbitrary partial structures which have a least (T- and e-) enumerations, i.e. the structures which have (T- and e-) degrees. First it will be done for structures with unary functions and predicates. The characterization will be in the terms of so called types and $\exists$-types of an elements of a structure. Roughly speaking, $\exists$-type of an element is the codes of all existential formulae which are true on that element in the structure. Then, using ideas for unary structures and predicates and so called Moschovakis’ extension and extended enumeration on it, it will be given characterization in the general case, as well. It will be given some corollaries, concerning the spectrum of a given structure. It will be given several examples, among which of structures which don’t have degree, but having quasi-degree, as well.
Combinatorics, Chromosomal Rearrangements and Reconstruction of Ancestral Genomes

Guénola Drillon

Laboratoire de Génomique des Microorganismes
Équipe Génomique Analytique
Université Pierre et Marie Curie
91, Bd de l’Hopital, 75013 PARIS
guenoladrillon@gmail.com

New technologies of sequencing have revolutionized genomics and especially our ability to understand evolution. The reconstruction of ancestral genomic sequences from available genomes is one of the challenges of bioinformatics today. Different species can have many genes in common, inherited from their common ancestors, but not necessarily ordered in the same way. This is due to chromosomal rearrangements, such as reversals or translocations, whose effects have not been yet completely understood. Many algorithms/models have been proposed, by computer scientists and by mathematicians, to reconstruct the history of rearrangements from one genome to another. Most models are based on parsimony, which expects the most direct transformation to be the one chosen by nature. The problem is usually stated in the language of signed permutations, where genes or groups of physically close genes are represented by numbers. Given a permutation, one asks how many reversals are required to transform a permutation in its identity. But the real challenge of bioinformatics today, is to study the underlying combinatorics of chromosomal rearrangements taking into account several genomes in once and several chromosomes partitioning a genome of a species. This demands the exploration of a large combinatorial space of chromosomal arrangements and asks for rather sophisticated algorithmic approaches. In this context, the comparison between what biologists know and what models reveal, bring to light interesting methodological questions and new approaches to the problem. For instance, is the phylogenetic tree of species necessary to reconstruct ancestral genomic sequences, or can we rather hope all needed information be coded in available genomes? How can information on pairs of genomes be of some use when we want to reconstruct the ancestor of several species? Do we need a combination of global information instead? And if it is the case, do we also need the precise information that can be obtained by comparing pairs of close genomes? Is the reconstruction of the ancestral genome from genomes A and B symmetric when we start from A and compare it to B, and the other way around? In the talk we shall try to present some mathematical framework where to reason about different strategies for the reconstruction of ancestral sequences.
Quantum and Classical Analogue Computation Using Microwaves

Mark S Everitt, Martin L. Jones, Vivien M Kendon
Neil B Lovett, Robert C Wagner

School of Physics and Astronomy
University of Leeds
Leeds, LS2 9JT, UK

The quantum version of analogue computation – usually known as continuous variable quantum computing (CVQC) – is relatively unexplored compared to digital quantum computation. We know that universal quantum computation is possible in an analogue setting [Lloyd+Braunstein PRL 82 1784 1999], with the same caveats as classical analogue computation where the resources scale unfavourably with precision due to the lack of binary encoding of the data. Little else is known about the theoretical underpinning and practical application. We are developing the required theoretical details for quantum analogue computation in the experimental setting of the micro maser (one atom maser), a very clean cavity quantum electrodynamics system. We have specified the experimental operations that are necessary and sufficient, and we are now comparing with classical analogue computation, using microwaves but no cavity system, to understand what provides the key quantum enhancement over Shannon’s original GPAC (general purpose analogue computer).

Our classical analogue studies are both theoretical and experimental, using a standard microwave synthesizer, waveguide and co-axial cable and suitable devices such as phase shifters and attenuators. We will then turn to the development of algorithms for the CVQC system, identifying the most suitable problems for such computers. We already know that integer problems such as factoring are not suitable, due to lack of binary encoding and hence poor scaling for the high precision required for handling large integers. However, simulation of quantum systems may be efficient and practical in an analogue setting since the digital quantum computer version also does not binary encode the data [Brown et al, PRL 97 050504 2006].

This is very much work in progress, but we have one paper accepted: Analogue computation using microwaves, Everitt et al, to appear in J. Unconventional Comp., and one close to submission: Universal continuous variable quantum computation in the micromaser, Wagner et al., on the basic operations required in the micromaser setting. We would welcome the opportunity for feedback and input from the CiE community into the theoretical side of this project.
Fixed-Parameter Tractability, Relative Kernelization and the Effectivization of Structural Connections

Michael Fellows, Juraj Hromkovič, Frances Rosamond and Monika Steinová*

The central concept of parameterized complexity is the generalization of P, fixed parameter tractability (FPT), solvability in time $f(k)n^c$ where $n$ is the input size, $k$ is the parameter, $c$ is a constant, and $f$ is an arbitrary function. Another view is based on the Kernel Lemma: A parameterized problem is FPT if and only if there is a P-time algorithm to transform $(x, k)$ into an equivalent kernel $(x', k')$, where: (1) $k' \leq k$, (2) $|x'| \leq g(k)$, and (3) $(x, k)$ is a yes-instance iff $(x', k')$ is a yes-instance. In recent years there has been an explosion of research in FPT kernelization, because this view of FPT allows the systematic investigation of pre-processing, something that is difficult to formulate in the classical framework. Such investigations have revealed that the subject has unexpected mathematical depth and strong practical applications. We investigate a generalization.

We initiate a systematic study of relative kernelization and the algorithmic effectivization of structural connections. Letting $\Gamma(G)$ denote the maximum length of a cycle in $G$, and $tw(G)$ the treewidth of $G$, then we have the connection: $\Gamma(G) \leq k$ implies $tw(G) \leq k$. A effectivization is given by the “win/win” algorithm of Fellows and Langston that in P-time either determines that $\Gamma(G) > k$ or produces a tree decomposition of width at most $k$. Win/win’s play an important role in many FPT algorithms. Mathematics offers many such existential (extremal) connections between various parameters of objects such as graphs. We initiate the study of complexity issues of effectivization. This leads to multivariate generalizations of FPT by varying (2), replacing the bound on the size of $x'$ by a bound on $\mu(x')$ for some other measure $\mu$ (for example, treewidth). We explore relative kernelization and its limits.

* This work was partially supported by SNF grant 200021-109252/1.
Computing over Non-Linear Structures

Christophe Fouqueré
LIPN, Université Paris 13

Computation over non-linear structures has not been seriously studied as far as one searches for a real correspondence with logic. We propose a presentation of a logic calculus (and variants) that is a conservative extension of Linear Logic. Linear Logic is a good framework for interpreting and computing over linear structures. We suggest to extend it by using a (formal) symmetric Fock algebra as a semantical framework, built on top of an algebraic semantics for Linear Logic, and together with a comultiplication. Note that comultiplication allows for a distribution of the semantics of a formula/program to several spaces, each of which denoting a sequent/universe of computation. For that purpose, we basically shift the logical syntax from a sequent view to a multisequent view where sharing of formula occurrences between such sequents is allowed, i.e. universes of computation may share programs or data. We introduce new connectives (hence data structures and operations) that allow for computing over non-linear structures, keeping logical properties available (the model is adequate w.r.t. the logic).

This research was first motivated by a study of lazy evaluation in logic programming. Since works of Andreoli, we know that full Linear Logic may be used as a logical programming language thanks to focalization and works have been done on lazy evaluation in this case. However a naive calculus taking laziness as a principle does not even satisfy cut-elimination. A second motivation concerns the control of true concurrency and interferences (or dually process independence) in proof search. As a last consideration, Hopf algebras have been used as a framework for investigating the semantics of Linear Logic or Noncommutative Logic. In these works, comultiplication is the operation dual to the multiplication (hence a Hopf structure). To our knowledge, the general case where the two operations are not algebraically dual has not been studied with respect to logic and the computation they can allow for.
Hierarchies below the Halting Problem for Additive Machines

Christine Gaßner

Institut für Mathematik und Informatik, Ernst-Moritz-Arndt-Universität,
F.-L.-Jahn-Straße 15 a, 17487 Greifswald, Germany
gassnerc@uni-greifswald.de

We deal with a question posed by Klaus Meer and Martin Ziegler in [1]. We consider the additive machines over the real numbers which can perform $+$ and $-$, and tests of the form $x \geq 0$. The oracle machines can also check whether a tuple of real numbers belongs to a given oracle set $O$ or not. The non-deterministic machines are able to guess arbitrary real numbers. In order to answer the question whether the set of rational numbers is strictly easier than the Halting problem $\mathbb{H}_{\text{add}} = \{(x, \text{code}(M)) \mid M \text{ is an additive machine and } M \text{ halts on input } x\}$, a first hierarchy was constructed in [2]. The problems $\mathbb{L}_n = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid (\exists (q_0, \ldots, q_{n-1}) \in \mathbb{Q}^n)(q_0 + \sum_{i=1}^{n-1} q_i x_i = x_n)\}$ form a hierarchy below the Halting Problem such that the set of rational numbers is strictly easier than the new problems. Moreover, every problem of the hierarchy is strictly easier than its successor. We have $\mathbb{Q} = \mathbb{L}_1 \preceq \mathbb{L}_2 \preceq \cdots \preceq \mathbb{L} = \bigcup_{n \geq 1} \mathbb{L}_n \preceq \mathbb{H}_{\text{add}}$ and $\mathbb{L}_{i+1} \npreceq \mathbb{L}_i$ where $A \preceq B$ means that $A$ can be decided by an additive oracle machine using only the constants 0 and 1 and the set $B$ as oracle.

Here, we want to answer the question posed in [2] by proving that $\mathbb{L}$ is strictly easier than $\mathbb{H}_{\text{add}}$. It is easy to see that $\mathbb{L} \preceq \mathbb{H}_{\text{add}}^1 = \mathbb{H}_{\text{add}}^1 \preceq \mathbb{H}_{\text{add}}^1 \preceq \mathbb{H}_{\text{add}}$ where $\mathbb{H}_{\text{add}}^1$ and $\mathbb{H}_{\text{add}}^1$ are halting problems restricted to additive machines whose only constants are 0 and 1 and, moreover, $\mathbb{H}_{\text{add}}^1$ is restricted to machines performing tests of the form $x = 0$ only. We will present hierarchies between $\mathbb{H}_{\text{add}}^1$ and $\mathbb{H}_{\text{add}}^1$ as well as hierarchies between $\mathbb{H}_{\text{add}}^1$ and $\mathbb{H}_{\text{add}}$. Since the additive machines can use a finite number of real constants, the construction of the latter is also dependent on the encoding technique used to encode the real constants.

Furthermore, we will shortly characterize the relationship between these problems and halting problems defined for non-deterministic additive machines.

References

$n$-Tardy Sets

Peter M. Gerdes

Department of Mathematics
University of Notre Dame, Indiana

In [1] Harrington and Soare introduced the $n$-tardy sets as a type of very tardy computably enumerable (c.e.) set, a c.e. set whose elements enter very slowly. They used the 2-tardy sets to prove several results about the automorphisms of $\mathcal{E}$, the c.e. sets under containment. However, the behavior of $n$-tardy sets for $n > 2$ was left largely unexplored. I will present results from joint work with Peter Cholak and Karen Lange about the $n$-tardy sets including the existence of a 3-tardy set Turing incomparable with any 2-tardy set. This answers negatively the question posed by Harrington and Soare: is every 3-tardy set codable, i.e., computed by some element of every orbit of $\mathcal{E}$. If time permits I will discuss the broader application of $n$-tardy sets for $n > 2$ to automorphisms of $\mathcal{E}$.

References

Empty Intervals of Enumeration Degrees and Strong Minimal Covers

Matthew B. Giorgi *, Thomas F. Kent**1, Andrew E. M. Lewis ***2, and Andrea Sorbi3

1 Department of Mathematics, Marywood University, Scranton, PA USA
tfkent@marywood.edu
2 Department of Pure Mathematics, University of Leeds aemlewis@aemlewis.co.uk
3 Dipartimento di Scienze Matematiche e Informatiche “Roberto Magari, University of Siena sorbi@unisi.it

In 1996, Calhoun and Slaman [1] answered an open question of Cooper by constructing a minimal cover in the \( \Pi^0_2 \)-enumeration degrees. We extend the forcing argument of Calhoun and Slaman to show that any finite distributive lattice \( L \) with 0 and 1 embeds into the \( \Pi^0_2 \)-enumeration degrees in the following manner: for any \( a, b \in L \), if \( b \) is a minimal cover over \( a \), then \( b \) is a minimal cover over \( a \), where \( a \) and \( b \) are the images of \( a \) and \( b \) respectively.

Additionally, we give an effective construction that demonstrates the existence of a strong minimal cover in the \( \Pi^0_2 \)-enumeration degrees. Finally, considering s-reducibility, a strong form of enumeration reducibility, we show that there is a strong minimal cover in the \( \Pi^0_2 \)-s degrees.

References


* The first author was supported by a Marie Curie Fellowship of the European Community Fifth Framework programme under contract number HPMF-CT-2002-018281.
** The second author has been partially supported by a Marie Curie Incoming International Fellowship of the European Community FP6 Program under contract number MIFI-CT-2006-021702
*** The third author was partially supported by a Marie-Curie Fellowship of the European Community Sixth Framework programme under contract number MEIF-CT-2005-023657 and by a Royal Society University Research Fellowship. The fourth author was partially supported by NSFC Grand International Joint Project New Directions in Theory and Applications of Models of Computations, No.60310213.
Elaborating Critical Points of Phase Transitions

L. Gordeev and A. Weiermann

Tübingen University and Utrecht University

We consider Higman- and Kruskal style well-quasi-orderings $\preceq$ on finite sequences of natural numbers and binary trees, respectively. It turns out that in order to analyze the critical point (threshold) of the corresponding Friedman-Weiermann style basic phase transition theorem it suffices to estimate the cardinality of the sets

$$\{\sigma \mid \sigma_K \not\preceq \sigma \land |\sigma| = n\}$$

where $\sigma_K$ denotes a suitable regular object (a sequence of natural numbers or a binary tree) determined by a sufficiently large natural number $K$. This can be done by Flajolet style analytic approach by investigating the OGFs of the corresponding classes

$$\{\sigma \mid \sigma_K \not\preceq \sigma\}$$

Surprisingly enough these OGFs are quite elementary rational functions, which by classical methods yields the desired critical point refinements.
In the local structure of enumeration degrees, the noncuppable degrees form a subclass of the properly $\Sigma^0_2$ degrees. This follows from the result that every $\Delta^0_2$ degree cups [CSY96]. In fact every noncuppable degree $a$ is downward properly $\Sigma^0_2$ in the sense that every non-zero degree below $a$ is also properly $\Sigma^0_2$. Accordingly the study of noncuppable degrees can be seen as a way forward in the investigation of the distribution of the properly $\Sigma^0_2$ degrees. A central question in this regard is that of the distribution of such degrees relative to the high/low jump hierarchy. Cooper and Copestake, who pioneered research in this area, showed in [CC88] that there exists a high properly $\Sigma^0_2$ enumeration degree. More recently Giorgi proved in [Gio08] that there exists a high noncuppable degree. On the other hand, another recent result by Giorgi, Sorbi and Yang has shown that every total $\Sigma^0_2$ enumeration degree bounds a noncuppable degree [GSY]. Therefore, since the standard embedding of the Turing degrees into the enumeration degrees preserves the jump operation, we can deduce, via standard results [Sac63] on the high/low jump hierarchy in the context of the local Turing degrees, that there exists a (relative to $\leq_e$) low$_2$ $\Sigma^0_2$ noncuppable enumeration degree. Now, constructive proofs of high properly $\Sigma^0_2$ and noncuppable enumeration degrees in the literature involve $\emptyset'$ priority tree arguments whereas the proof in [GSY] of the existence of a low$_2$ noncuppable degree is carried out using a derivative construction. With this in mind, I will show, using jump inversion results [Gri03,Har09a], and a simple observation on the relationship between enumeration reducibility and relative computable enumerability, that both results can be obtained using straightforward finite injury constructions [Har09b]. Essentially this is because, instead of using the standard method of defining a $\Sigma^0_2$ approximation $\{\hat{A}_s\}_{s \in \omega}$ to construct a $\Sigma^0_2$ set $A$, one can enumerate $A$ using the halting set $\emptyset'$ as Turing oracle, thus obtaining an approximation $\{A_s\}_{s \in \omega}$, that is c.e. in $\emptyset'$. Accordingly the availability of the information inherent in $\emptyset'$ during the construction allows the latter to recognise $\Sigma^0_1/\Pi^0_1$ outcomes in a finitary manner thus eliminating the need to process requirements infinitely often. I will also briefly explain how these proofs lay the foundation for the construction of noncuppable $\Sigma^0_2$ enumeration degrees lying at every possible level of the high/low jump hierarchy as carried out in [Har09c].
References


On the Form of Witness Terms

Stefan Hetzl

INRIA Saclay – Ile-de-France
École Polytechnique – LIX
91128 Palaiseau, France
hetzl@lix.polytechnique.fr

Cut-elimination is an inherently non-deterministic process. At each stage one has the choice between different cuts to reduce and – for a single cut – there are different ways of reducing it. This formal non-determinism has the effect that depending on the chosen cut-elimination procedure, mathematically different normal forms may be produced. Understanding the possible span of the results of cut-elimination procedures is therefore of fundamental importance for judging proof analyses based on these methods.

The mathematical content of a cut-free proof is fully contained in the information how to instantiate which quantifiers, i.e., in the witness terms. They therefore provide adequate means for relating different cut-free proofs.

We analyze the development of terms during cut-elimination in first-order logic and Peano arithmetic. The main result is a characterization of the form of witness terms in cut-free proofs in terms of structured combinations of basic substitutions that are read off from the proof with cuts.

Based on this result, it is shown that each proof with cuts induces a regular tree grammar s.t. every witness term computable by cut-elimination can also be computed by the grammar. As a second application of this result, a class of proofs in first-order logic is shown to have only elementary cut-elimination (while cut-elimination in the worst case is non-elementary). From the algorithmic point of view, we obtain a method for computing witness terms that circumvents cut-elimination and has several advantages, it allows for example to find the shortest witness term. All of the above results also apply to proofs of $\Sigma^0_1$-formulas in Peano arithmetic.
Parallel Term Rewriting in Neural Networks

Ekaterina Komendantskaya

School of Computer Science, University of St Andrews, UK

ek@cs.st-andrews.ac.uk

Term rewriting is a major area of research in theoretical computer science, and has found numerous applications in lambda calculus, higher-order logics and functional programming. To establish confluence of a given term-rewriting system, a parallel rewriting step is needed. Traditionally, this parallel step is defined sequentially - as a sequence of disjoint rewriting steps.

Neural networks, however, allow us to perform a parallel rewriting step in a truly parallel fashion. So, we believe that the area of parallel term-rewriting has a big potential to become the common ground for Neuro-Symbolic Networks and Computational Logic, where the parallelism evolves and occurs naturally.

We will show how parallel term rewriting can be implemented in neural networks. That is, we define a network with suitable architecture and processing/learning functions to perform parallel term-rewriting steps. We apply the network both to abstract rewriting and first-order term-rewriting systems. The resulting neural network performs parallel rewriting steps by unsupervised learning. We will present a library of functions and examples written in MATLAB Neural Network Toolbox, which implements the network we describe.
Definability of Combinatorial Functions

Extended Abstract

T. Kotek * and J.A. Makowsky **

Department of Computer Science
Technion–Israel Institute of Technology
32000 Haifa, Israel
e-mail: {tkotek,janos}@cs.technion.ac.il

Abstract. We consider functions of natural numbers and sequences of polynomials which allow a combinatorial interpretation, such as Fibonacci numbers and polynomials, Bell numbers and Touchard polynomials, Catalan numbers and the like. Many of these functions satisfy a linear recurrence relation over $\mathbb{Z}$ or $\mathbb{Z}_m$ and allow an interpretation as counting the number of relations satisfying a property expressible in Monadic Second Order Logic (MSOL).

C. Blatter and E. Specker (1981) showed that if such a function $f$ counts the number of binary relations satisfying a property expressible in MSOL then $f$ satisfies for every $m \in \mathbb{N}$ a linear recurrence relation over $\mathbb{Z}_m$.

In this paper we give a complete characterization in terms of definability in MSOL of the combinatorial functions which satisfy a linear recurrence relation over $\mathbb{Z}$, and discuss various extensions and limitations of the Specker-Blatter theorem.

* Partially supported by a grant of the Graduate School of the Technion–Israel Institute of Technology
** Partially supported by a grant of the Fund for Promotion of Research of the Technion–Israel Institute of Technology and grant ISF 1392/07 of the Israel Science Foundation (2007-2010)
The Semantics and Complexity of Successor-free Nondeterministic Gödel’s T and PCF

Lars Kristiansen¹
Bedeho Mesghina Wolde Mender²

¹ Department of Mathematics, University of Oslo
larsk@math.uio.no WWW home page: http://folk.uio.no/larsk/
² Institute of Informatics, University of Oslo
bmmender@ifi.uio.no

Neat and natural successor-free fragments of Gödel’s system T and Plotkin’s PCF capture a family of well-known complexity classes, see e.g. [2], [4] and [3]. Nondeterminism is a pivotal notion of complexity theory. Hence it seems interesting to study successor-free fragments of nondeterministic versions of system T and PCF. In this paper we introduce nondeterminism in a naive and straightforward way. We simply extend our deterministic calculi by

\[(M\mid N)\] is a term of type \(\sigma\) if \(M\) and \(N\) are terms of type \(\sigma\)

and the two accompanying reduction rules \(M\mid N \rightarrow M\) and \(M\mid N \rightarrow N\). The theorems found in this paper show that such an extension makes sense. We also prove some results on the computational power of the nondeterministic calculi versus the computational power of the deterministic calculi.

The paper is based on the second author’s Master thesis [1].

References

Computably Well-Founded $\omega$-c.e. Linear Extensions

Kyung Il Lee
University of Leeds

Using a priority construction, we will prove a strong version of a theorem in Rosenstein [2]: every computably well-founded partial order has a computably well-founded $\omega$-c.e. linear extension. Note that Rosenstein’s theorem provides a construction of a computably well-founded $\Delta^0_2$ linear extension under the same condition, using an oracle for $\mathcal{O}'$. On the other hand, Rosenstein [2] gives a counterexample to show that there is a computably well-founded computable partial order with no computably wellfounded computable linear extension. We will discuss the possibility of extending this counterexample to that of a computably well-founded d-c.e. linear extension.

(Joint work with S. B. Cooper and A. Morphett.)

References

Random Reference-Switching Functions

Chung-Chih Li

School of Information Technology
Illinois State University
Normal, IL 61790, USA

This idea of RRS (Random Reference-Switching) functions emerged from our previous effort in developing VPF (Virtual Password Functions) to avoid some common password thief attacks. After some speculation, I learned that, if properly defined, some RRS functions were much stronger than we originally expected in the context of light-weighted cryptosystem in which the computational power was limited. What I have proven is the reverse functions of such RRS are NP-complete. However, a probabilistic algorithm can resolve the hidden random key in a polynomial time. Here I give the definitions and a theorem.

Let $Z_m = \{0, 1, \ldots, m - 1\}$, and let $X, R, V \in Z_m^n$ be vectors as follows:

$$X = (x_0, x_1, x_2, \ldots, x_{n-1}), R = (r_0, r_1, r_2, \ldots, r_{n-1}), V = (v_0, v_1, v_2, \ldots, v_{n-1})$$

Definitions:

- An RRS function, $f$, is a function of type $Z_m^n \times Z_m^n \rightarrow Z_m^n$ such that, it is intractable to find $X$ from any given $R$ and $V$ such that $f(X, R) = V$.
- An strong RRS function, $f$, is an RRS function such that, it is intractable to find $X$ from any chosen $R$ and $V$ such that $f(X, R) = V$.

An RRS function is said to be weak if it is not strong.

**Theorem 1.** Given any nonzero $n, m \in \mathbb{N}$ and $V \in Z_m^n$, to decide whether there exists $X \in Z_m^n$ such that (1) hold is an NP-complete problem.

$$v_0 \equiv x_0 x_0 \mod n \mod m$$
$$v_1 \equiv x_1 x_1 \mod n \mod m$$
$$\vdots$$
$$v_{n-1} \equiv x_{n-1} x_{n-1} \mod n \mod m$$

In this informal presentation, I would like to expose the concept of RRS and use the theorem to define strong RRS functions.
Dynamic Algorithms For Reachability Games

Jiamou Liu

Department of Computer Science, University of Auckland, New Zealand
jliu036@aucklanduni.ac.nz

In this talk I will present new results in the investigation of the dynamic content of game determinacy when games are played on finite graphs [2]. A two-person reachability game $G$ is a pair $(G, T)$ where $G$ is a finite directed graph whose nodes are partitioned into two subsets $V_0$ and $V_1$ and $T \subseteq V_0 \cup V_1$ is the set of target nodes. The two players, Player 0 and Player 1, start playing the game by placing a token on some node $v \in V$ and then move the token in rounds. At each round, if the token is placed at $u \in V_\sigma$, $\sigma \in \{0, 1\}$, then it is moved by Player $\sigma$ along an edge by respecting the direction of the edge. Player 0 wins whenever the token is placed on a target node, and Player 1 wins otherwise. To solve the game means to design an (efficient) algorithm that tells us from which nodes a given player wins the game. On a graph with $n$ vertices and $m$ edges, the reachability game is solve in time $O(n + m)$ and is PTIME-complete [1].

The problem of dynamic game determinacy calls for finding efficient algorithms that solve the game when the underlying graph undergoes repeated modifications. The possible modifications include insertion/deletion of nodes/edges and modifying the set of target nodes $T$. An algorithm is fully-dynamic if it supports all of the mentioned modifications, and is partially-dynamic if it only supports some of these modifications. In this talk, I will present the following algorithms that solve the dynamic game determinacy problem:

1. a fully-dynamic algorithm for games played on trees with amortized time complexity $O(\log n)$, where $n$ is the number of nodes in the current graph.
2. a partially-dynamic algorithm for games played on any directed graphs with amortized time complexity $O(1)$.

References

Some Properties of Traces as a Lattice under the Set Theoretic Inclusion

Gunther Mainhardt
Department of Mathematics and Computer Science
University of Heidelberg, Germany

Let a trace be a computably enumerable set $V$ of pairs of natural numbers, such that each row of $V$ is finite. After looking at some basic properties of traces, like, that there is no uniform enumeration of all traces, we turn to lattices

$$L_{tr}(V) = (\{W | V \subseteq W \text{ and } W \text{ a trace}\}, \subseteq),$$

$V$ a trace. We study the close relationship between $L_{tr}(\emptyset)$ and $(\{W | W \subseteq \text{N c.e.}\}, \subseteq)$. We further study isomorphisms from $L_{tr}(\emptyset)$ to $L_{tr}(V)$, $V$ a trace. We will also see, that if $W$ and $V$ are traces, then $L_{tr}(W)$ can isomorphically be embedded into $L_{tr}(V)$.

Of course, we will use methods typical for the theory of computably enumerable sets, however, some of our considerations are also of interest from a lattice theoretic point of view.
Sequential Functionals at Type Level 2

Dag Normann

Department of Mathematics
The University of Oslo

In 2004 D.N. showed that the sequential functionals of pure type 3, with the extentional ordering, is not a directed complete partial ordering (dcpo). In 2008 D.N. and Vladimir Sazonov joined efforts to reveal further order theoretical properties of the sequential functionals of various types. In this presentation we will give an example of a strictly increasing sequence of sequential functionals of type

$$(\iota, \iota \to \iota) \to \iota$$

with no sequential upper bound.

We conclude that it is only for types at level $\leq 1$ and pure type 2 that the sequential functionals form a dcpo.

If time permits, we will discuss further results from this joint project.
Real Hypercomputation with Infinite Oracle Queries

Arno Pauly

Computer Laboratory
University of Cambridge
United Kingdom
Arno.Pauly@cl.cam.ac.uk

We define Oracle-Type-2-Machine capable of writing infinite oracle queries. In contrast to finite oracle queries, this extends the realm of oracle-computable function into the discontinuous realm. Our definition is conservative; access to a computable oracle does not increase the computational power.

Other models of real hypercomputation such as Ziegler’s (finitely) revising computation ([4], [5]) are shown to be special cases of Oracle-Type-2-Machines. Our approach offers an intuitive definition of the weakest machine model capable to simulate both Type-2-Machines ([3]) and BSS machines ([1]). As an example for applications, we can derive some results on functions definable by arithmetic circuits similar to [2].

References

Index Sets and Scott Sentences

Sara Quinn

Northwestern University, 2033 Sheridan Road Evanston, IL 60208-2730, USA.

In 1974, Vaught showed that there is a tight connection in the Borel hierarchy between the complexity of a class of structures and the complexity of a sentence axiomatizing the class. In the setting of computable structures, this connection suggests a strategy for determining the complexity of the index set for a structure: if we first write down a sentence giving an optimal description of the structure (a Scott sentence) then we have an upper bound for the complexity of the index set. In this talk, I will present results where, as is often the case, this upper bound is achieved. I will also describe a structure whose index set must be less complicated than any of its Scott sentences, and describe how our strategy for determining index set complexity might be revised.
A Crypto System Based on Symmetric Key Block Cipher

N. Radha

Kalasalingam University
Krishnankoil,
Srivilliputhur (via) Virudhunagar Dt,
Tamilnadu, India

Intellectual property owners must continually exploit new ways of reproducing, distributing, and marketing their products. However, the threat of piracy looms as a major problem with digital distribution and storage technologies. Multimedia encryption and authentication techniques cover current and future trends in the design of modern systems that use encryption and authentication to protect multimedia content.

Problem definition: We propose a new block cipher based on generalized shuffling operator $S$. The one-dimensional plain-text of size $n$ undergoes $n$ iterations under $S$. Every $n$ iteration generates the pseudo cipher text. The block size may be 8, 16, 32, 64, 128, 256, 512, 1024 bits. The encryption process generates the cipher pairs $(C, K')$ where $C$ is the cipher text and $K'$ is the encrypted version of the key $K$. The shuffling operator $S^{-1}$ when applied to $K'$ gives back the key $K$. The decryption process undergoes $n$ iterations using $S^{-1}$ and $K'$, and yields the plain text.
Algorithmic Reducibilities and Borel Equivalence Relations

Jan Reimann
Department of Mathematics, University of California, 705 Evans Hall, Berkeley reimann@math.berkeley.edu

The degrees of unsolvability of the various algorithmic reducibilities present natural examples of countable Borel equivalence relations, i.e. an equivalence relation that is Borel (as a subset of \( \mathbb{N}^\mathbb{N} \times \mathbb{N}^\mathbb{N} \)) and so that every equivalence class is countable.

Borel equivalence relations can be partially ordered using Borel reductions. This partial ordering has a rich structure. There exists a universal countable equivalence relation (induced by the shift action of the free group \( F_2 \) on two generators on \( 2^{F_2} \)) to which all other countable Borel equivalence relations can be reduced.

For most algorithmic reducibilities, the exact strength of the induced equivalence relation is not known, for instance whether they are universal.

We give a short overview of the known results and present a new partial result concerning the universality of Turing equivalence.
We will apply the methods developed in the field of ‘proof mining’ (see mainly [1]) to the Bolzano-Weierstraß theorem \(\text{BW}\) and calibrate the computational contribution of using this theorem in a proof of a combinatorial statement. Using a simple proof based on weak König’s lemma (analyzed by W. A. Howard, [2]) and arithmetical comprehension (analyzed by C. Spector, [3]) we provide an explicit solution of the Gödel functional interpretation as well as the monotone functional interpretation of \(\text{BW}\) for the product space \(\prod_{i \in \mathbb{N}} [-k_i, k_i]\) for a given sequence of rational numbers \((k_i) \subseteq \mathbb{Q}\) (with the standard product metric). In fact, we will use it to get optimal program and bound extraction theorems for proofs based on fixed instances of \(\text{BW}\), i.e. for \(\text{BW}\) applied to fixed sequences \(\prod_{i \in \mathbb{N}} [-k_i, k_i]\).

References


* This is joint work with Ulrich Kohlenbach. The author gratefully acknowledges the support by the German Science Foundation (DFG Project KO 1737/5-1).
An Impact of Infinity: Comparison of Two Reverse Mathematics

Kentaro Sato
Kobe University, Japan
textttkensato@umich.edu

In mathematics, it happens quite often that the allowance of infinity changes the picture drastically, e.g., the dimension of vector spaces. Logic is not an exception: infinity allows sets to be isomorphic to their proper subsets. However, there seems to be few results that clearly show how infinity changes the situation, or “controlled experiments” for infinity. In this talk, a “controlled experiment” is given.

The tentative “methodology” is as follows. On the infinite side second-order arithmetic, which treats infinite (and finite) sets of natural numbers, has been widely investigated (classical reverse mathematics). On the finite side two-sorted bounded arithmetic, which treats only finite sets, has recently been investigated (bounded reverse mathematics). There is an analogy between the complexities of formulae of these two frameworks: $\Sigma^1_n$ cares about (unbounded) set quantifiers and does not care about (unbounded) number quantifiers; $\Sigma^B_n$ cares about bounded set quantifiers and does not care about bounded number quantifiers, where set quantifiers are bounded in the sense of the lengths of binary expressions. Thus, via this analogy, we can define unbounded and bounded versions of several principles, e.g., number recursion ($NR$), comparability of well-orderings ($CWO$), decidability of well-foundedness ($DWF$), decidability of bisimilarity ($BISIM$), and transfinite recursion schemata ($\Gamma\text{-}TR$) (though the bounded version of $DWF$ is just checking acyclicity and orders in that of $\Gamma\text{-}TR$ is not transcendental over finite any more!).

The results are as in the following table, where a system and a principle in the same column are equivalent over a suitable base theory ($\text{RCA}_0$ for unbounded versions, and $\text{V}^0$ for bounded ones). One can notice that the order among the strengths is completely changed by the allowance of infinity.

<table>
<thead>
<tr>
<th>System</th>
<th>bounded versions (in bounded RM)</th>
<th>unbounded versions (in classical RM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>principle</td>
<td>$\text{VTC}^0$</td>
<td>$\text{VL}$</td>
</tr>
<tr>
<td>CWO</td>
<td>NR</td>
<td>$\text{DWF}$</td>
</tr>
</tbody>
</table>

There remains a problem: two versions denoted by the same letters are not the rigorously same, and the frameworks are different. In this sense, the “experiment” is not properly “controlled”. To solve this, in the final ”methodology”, we translate the principles and the base theories into set theory. Then the two versions are translated into exactly the same formulae, e.g., $\Delta^1_0$ or $\Delta^B_0\text{-TR}$ into $\Delta_0\text{-TR}$, $NR$ into weak depending choice and $DWF$ into the axiom of regularity. The difference of base theories is “controlled” only by the axiom of infinity. This could be called the unification of the two reverse mathematicis.

References

A Characterization of the Join-Irreducible Medvedev Degrees

Paul Shafer

Cornell University

For sets $A, B \subseteq \omega^\omega$, we say $A$ Medvedev reduces to $B$ ($A \leq_M B$) if there is a Turing functional $\Phi$ such that $\Phi^f$ is total and in $A$ for all $f$ in $B$, and we say $A, B \subseteq \omega^\omega$ are Medvedev equivalent ($A \equiv_M B$) if $A \leq_M B$ and $B \leq_M A$. The Medvedev lattice is the degree structure $(P(\omega^\omega)/\equiv_M, \leq)$, where $\leq$ is the ordering $\leq_M$ induces on equivalence classes. An element $a$ of a lattice is called join-irreducible if there are no $b, c < a$ with $b \lor c = a$. We show that the Medvedev degree of $A \subset \omega^\omega$ is join-irreducible if and only if $A$ is Medvedev equivalent to the complement of an ideal in the Turing degrees. This answers a question of Sorbi from [1]. Let $0'$ denote the degree of all non-recursive functions. We show that there is a Medvedev degree $a > 0'$ that bounds no join-irreducible Medvedev degree $> 0'$. This answers a question of Sorbi and Terwijn from [2].

References

Some Applications of the Jump Inversion Theorem for the Degree Spectra

Alexandra A. Soskova and Ivan N. Soskov *

Faculty of Mathematics and Informatics,
Sofia University,
5 James Bourchier blvd.,
1164 Sofia, Bulgaria,
asoskova@fmi.uni-sofia.bg
soskov@fmi.uni-sofia.bg

We will present some applications of the Jump inversion theorem for the
degree spectra [3], which says that every jump spectrum is also a spectrum and
that if a spectrum \( A \) is contained in the set of the jumps of the degrees in some
spectrum \( B \) then there exists a spectrum \( C \) such that \( C \subseteq B \) and \( A \) is equal to
the set of the jumps of the degrees in \( C \).

We will give a method of constructing a structure, possessing an \( n \)-th jump
degree equal to \( 0^{(n)} \) and which has no \( k \)-th jump degree for \( k < n \) (known result
[1]).

The next application is a generalization of results of SLAMAN [2] and WEHNER
[4]. We construct a structure whose \( n \)-th jump spectrum contains all degrees
above an arbitrary fixed degree.

**Theorem 1.** For each \( n \in \mathbb{N} \) and every Turing degree \( b \geq 0^{(n)} \) there exists a
structure \( \mathcal{C} \), whose \( n \)-th jump spectrum \( DS_n(\mathcal{C}) = \{ x \mid x >_T b \} \).

We would like to point out that the Jump inversion theorem gives a method
to lift some interesting results for degree spectra to the \( n \)-th jump spectra.

**References**

1. Downey, R. G., Knight, J. F. : Orderings with \( \alpha \)-th jump degree \( 0^{(\alpha)} \). Proc. Amer.
4. Wehner St. : Enumerations, Countable Structures and Turing Degrees. Proc. of

* This work was partially supported by Bulgarian National Science Fund under con-
tract D002-258/18.12.08
Embedding Countable Partial Orderings in Degree Structures

Mariya I. Soskova and Ivan N. Soskov *

Faculty of Mathematics and Informatics, Sofia University, 5 James Bourchier blvd., 1164 Sofia

One of the most basic measures of the complexity of a given partially ordered structure is the quantity of partial orderings embeddable in this structure. The study of this problem in the Turing degrees has lead to the discovery of the embeddability method via independent sequences of sets. We study the embeddability problem further in the context of three different structures. We start with a slight generalization of the known embeddability results for the structure of the enumeration degrees.

**Theorem 1.** If $B <_e A$ are sets of natural numbers such that $A$ has a good approximation, then there is an embedding of every countable partial ordering in the interval $[d_e(B), d_e(A)]$.

Next we focus on the structure of the $\omega$-enumeration degrees, $D_\omega$. This structure is an upper semi-lattice with jump operation, where the building blocks of the degrees are of a higher type - sequences of sets of natural numbers, introduced by Soskov and investigated in the works of Ganchev and Soskov. Our second result is strong generalization of Soskov’s density theorem for the substructure of the $\Sigma^0_2$ $\omega$-enumeration degrees, $G_\omega$.

**Theorem 2.** Let $b <_\omega a <_\omega 0'_{\omega}$ be two $\Sigma^0_2$ $\omega$-enumeration degrees. There is an embedding of every countable partial ordering in the interval $[b, a]$.

Finally consider the structure of the $\Sigma^0_2$ $\omega$-enumeration degrees modulo iterated jumps, $G_\omega / \sim_\infty$, the structure induced by the preorder $\leq_\infty$ on $G_\omega$, defined by $a \leq_\infty b$ if there is a natural number $n$ such that $a^{(n)} \leq_\omega b^{(n)}$. The structure of the c.e. Turing degrees modulo iterated jump, $R/ \sim_\infty$, was studied by Jockusch, Lerman, Soare and Solovay and Lempp. They prove that this is a dense structure and that there is a splitting of the highest $\infty$-degree and a minimal pair of $\infty$-degrees. We show that $G_\omega / \sim_\infty$ is a proper extension of the structure $R/ \sim_\infty$. The main result is an application of Theorem 2.

**Theorem 3.** If $a$ and $b$ are $\omega$-enumeration degrees such that $[b] \sim_\infty < [a] \sim_\infty$, then there is an embedding of every countable partial ordering in the $\infty$-degrees between $[b] \sim_\infty$ and $[a] \sim_\infty$.

* This research was supported by an BNSF Grant No. D002-258/18.12.08 and by a Marie Curie European Reintegration Grant No. 239193 within the 7th European Community Framework Programme.
On Semilattices of \( \Sigma \)-Degrees of Structures

Alexey Stukachev

Sobolev Institute of Mathematics, Novosibirsk, Russia

aistu@math.nsc.ru

We consider computability on abstract structures via the formalism based on the notion of \( \Sigma \)-definability in admissible sets and, in particular, in HF-superstructures. Providing a natural generalization of the classical computable model theory, this approach can be used for measuring the complexity of abstract structures, no matter countable or not.

We present some new results about the semilattices \( S_\Sigma(\alpha) \) of \( \Sigma \)-degrees of structures with cardinality less or equal \( \alpha \), where \( \alpha \) is an infinite cardinal. First, concerning the global structure of the semilattices of \( \Sigma \)-degrees, we prove the jump inversion theorem for \( S_\Sigma(\alpha) \). For \( \alpha = \omega \), this can be regarded as a generalization (in view of [4, 5]) of the similar result by I.N.Soskov and A.A.Soskova [2].

Second, concerning the conjecture stated by Yu.L.Ershov in [1], we study some (local) properties of \( \Sigma \)-degrees of uncountable structures. Namely, we prove that any \( c \)-simple theory with an additional property of discreteness has an uncountable model \( \Sigma \)-definable in \( \text{HF}(\mathcal{L}) \), where \( \mathcal{L} \) is a dense linear order. As a corollary, this is true for any submodel complete \( c \)-simple theory with a finite signature, while in the class of submodel complete \( c \)-simple theories with infinite signatures there is a counterexample [3].

References

Solving a PSPACE-Complete Problem
with DNA Computing

Thomas Zerjatke

Institut für Mathematik und Informatik,
Ernst-Moritz-Arndt-Universität Greifswald, Germany

Since Adleman’s presentation of the first practical implementation of a DNA
based algorithm in 1994 [1] there have been quite a number of suggestions for
solving NP-complete problems with DNA computing, some of which have been
practically realised for small instances. In contrast there have been only a few
papers about solutions of PSPACE-complete problems. Wolpert and Dantsin [2]
made a proposal for solving such a problem, the problem of Quantified Boolean
Formulas (QBF). In their approach all assignments of the Boolean variables are
coded in DNA strands, building the initial pool. Now in a first step those strands
are filtered out of this pool that code for a truth assignment of the unquantified
Boolean formula. In a second step it is checked whether all quantifications are
fulfilled by using the property of DNA complementarity. The main drawback of
this approach is the need of $O(2^n)$ DNA strands coding for all possible assign-
ments where $n$ is the number of Boolean variables in the given QBF. This leads
to impractical amounts of DNA even for relatively small numbers of variables.

I will present an algorithm decreasing the number of strands needed. My
approach is based on a breadth first search method suggested by Yoshida and
Suyama [3] for solving the NP-complete problem SAT. The truth assignments of
the unquantified formula are successively built so that there is no need to create
an initial pool of DNA strands coding for all $2^n$ assignments. Yet an additional
step has to be carried out to reorder the strands before checking whether the
quantifications are fulfilled.

I will also discuss limitations that would occur in a practical implementation.

References

1. L. Adleman: Molecular computation of solutions to combinatorial problems. Science,
2. A. Wolpert and E. Dantsin: A robust DNA computation model that captures
3. H. Yoshida and A. Suyama: Solution to 3-SAT by breadth first search. In E. Winfree
   and D. Gifford, editors, DNA Based Computers V, DIMACS, pages 9–22. American

* This work is based on my diploma thesis that was supervised by Christine Gaßner
   (University of Greifswald) and Monika Sturm (TU Dresden).
On the Role of Quantum Metalanguage in Brain Quantum Computation

P. Zizzi* and E. Pessa**

*Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Trieste, 63 35121 Padova, Italy
*Dipartimento di Psicologia, Università di Pavia, Piazza Botta, 6 27100 Pavia, Italy

A classical metalanguage reflects into an object language, via the so-called definitional equation of Basic logic, which turns metalinguistic links between assertions into logical connectives between propositions. However, a classical metalanguage does not reflect properly into a quantum object language (the quantum logic of a quantum computer, for example) when a deductive calculus is taken into account. Then, a quantum metalanguage is required, and, in this way, it is possible to introduce the new logical connective of quantum superposition, which is similar to the classical conjunction, but depends on two parameters, which, in the interpretation, correspond to the probability amplitudes. In this case, atomic propositions are interpreted as weak measurement operators acting on the basis states of the Hilbert space. As this quantum metalanguage is obviously a product of a (meta) thought process, and weak measurements implement quantum control on quantum systems, then this quantum metalanguage acts as a quantum control on the thought process itself. Moreover, the eigenstates of the weak measurement operators are generalized coherent states. This suggests that high-level mental states are quantum generalized coherent states, produced in quantum dissipative brain processes. In general, therefore, we are in the framework of Quantum Field Theory. However, when one considers a particular kind of superposition of generalized coherent states, then those compound propositions correspond to unitary operators, which indicate the occurrence of quantum computation. This means that, in this case, quantum metalanguage is a quantum control on quantum computation taking place in the brain.
<table>
<thead>
<tr>
<th>Author</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abeshev, Kuanysh</td>
<td>23</td>
</tr>
<tr>
<td>Ackerman, Nathanael L.</td>
<td>317</td>
</tr>
<tr>
<td>Agrawal, Manindra</td>
<td>2</td>
</tr>
<tr>
<td>Alaev, Pavel</td>
<td>318</td>
</tr>
<tr>
<td>Angadi, Ulavappa B.</td>
<td>28</td>
</tr>
<tr>
<td>Avigad, Jeremy</td>
<td>3</td>
</tr>
<tr>
<td>Babenyshev, Sergey</td>
<td>38</td>
</tr>
<tr>
<td>Badaev, Serikzhan</td>
<td>23</td>
</tr>
<tr>
<td>Bakibayev, Timur</td>
<td>319</td>
</tr>
<tr>
<td>Barmpalias, George</td>
<td>9</td>
</tr>
<tr>
<td>Barra, Mathias</td>
<td>320</td>
</tr>
<tr>
<td>Bayer, Hannah</td>
<td>48</td>
</tr>
<tr>
<td>ben-Amram, Amir</td>
<td>321</td>
</tr>
<tr>
<td>Berger, Josef</td>
<td>58</td>
</tr>
<tr>
<td>Bienvenu, Laurent</td>
<td>11</td>
</tr>
<tr>
<td>Blockeel, Hendrik</td>
<td>60</td>
</tr>
<tr>
<td>Bodewig, Philipp</td>
<td>322</td>
</tr>
<tr>
<td>Brattka, Vasko</td>
<td>323</td>
</tr>
<tr>
<td>Brijder, Robert</td>
<td>60</td>
</tr>
<tr>
<td>Brodhead, Paul</td>
<td>70</td>
</tr>
<tr>
<td>Candela, Roberto</td>
<td>79</td>
</tr>
<tr>
<td>Cenzer, Douglas</td>
<td>70</td>
</tr>
<tr>
<td>Chen, Jingchao</td>
<td>89</td>
</tr>
<tr>
<td>Chen, Yixiang</td>
<td>246</td>
</tr>
<tr>
<td>Chvalovský, Karel</td>
<td>324</td>
</tr>
<tr>
<td>Collins, Pieter</td>
<td>99, 110</td>
</tr>
<tr>
<td>Conidis, Chris</td>
<td>325</td>
</tr>
<tr>
<td>Conte, Graziana</td>
<td>326</td>
</tr>
<tr>
<td>Costa, Antônio Carlos da Rocha</td>
<td>120</td>
</tr>
<tr>
<td>Cottone, Giulio</td>
<td>79</td>
</tr>
<tr>
<td>De Smet, Michiel</td>
<td>327</td>
</tr>
<tr>
<td>Dimuro, Graçaliz Pereira</td>
<td>120</td>
</tr>
<tr>
<td>Ditchev, Angel V.</td>
<td>328</td>
</tr>
<tr>
<td>Drillon, Guénola</td>
<td>329</td>
</tr>
<tr>
<td>Edmunds, Mike</td>
<td>1</td>
</tr>
<tr>
<td>Eisenbrand, Friedrich</td>
<td>12</td>
</tr>
<tr>
<td>Escuela, G.</td>
<td>186</td>
</tr>
<tr>
<td>Everitt, Mark S.</td>
<td>330</td>
</tr>
<tr>
<td>Faßler, R.</td>
<td>186</td>
</tr>
<tr>
<td>Fellows, Michael</td>
<td>331</td>
</tr>
<tr>
<td>Fileccia Scimemi, Giuseppe</td>
<td>79</td>
</tr>
<tr>
<td>Fiorino, Guido</td>
<td>130</td>
</tr>
<tr>
<td>Fouqueré, Christophe</td>
<td>332</td>
</tr>
<tr>
<td>Freer, Cameron E.</td>
<td>317</td>
</tr>
<tr>
<td>Gaßner, Christine</td>
<td>333</td>
</tr>
<tr>
<td>Gerdes, Peter M.</td>
<td>334</td>
</tr>
<tr>
<td>Gherardi, Guido</td>
<td>323</td>
</tr>
<tr>
<td>Giorgi, Matthew B.</td>
<td>335</td>
</tr>
<tr>
<td>Goncharov, Sergey</td>
<td>13</td>
</tr>
<tr>
<td>Gordeev, L.</td>
<td>336</td>
</tr>
<tr>
<td>Guo, Binghui</td>
<td>140</td>
</tr>
<tr>
<td>Harris, Charles M.</td>
<td>337</td>
</tr>
<tr>
<td>Hemmerling, Armin</td>
<td>150</td>
</tr>
<tr>
<td>Herman, Grzegorz</td>
<td>162</td>
</tr>
<tr>
<td>Hernández, J. A.</td>
<td>256</td>
</tr>
<tr>
<td>Hetzl, Stefan</td>
<td>339</td>
</tr>
<tr>
<td>Higuchi, Kojiro</td>
<td>176</td>
</tr>
<tr>
<td>Hinze, T.</td>
<td>186</td>
</tr>
<tr>
<td>Hromković, Juraj</td>
<td>331</td>
</tr>
<tr>
<td>Ibrahim, B.</td>
<td>186</td>
</tr>
<tr>
<td>Jones, Martin L.</td>
<td>330</td>
</tr>
<tr>
<td>Kľuka, Ján</td>
<td>206</td>
</tr>
<tr>
<td>Kamide, Norihiro</td>
<td>196</td>
</tr>
<tr>
<td>Kendon, Vivien M.</td>
<td>330</td>
</tr>
<tr>
<td>Kent, Thomas F.</td>
<td>335</td>
</tr>
<tr>
<td>Kihara, Takayuki</td>
<td>176</td>
</tr>
<tr>
<td>Author Name</td>
<td>Page Numbers</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Koepke, Peter</td>
<td>4</td>
</tr>
<tr>
<td>Kolaitis, Phokion G.</td>
<td>5</td>
</tr>
<tr>
<td>Komendantskaya, Ekaterina</td>
<td>340</td>
</tr>
<tr>
<td>Kotek, T.</td>
<td>341</td>
</tr>
<tr>
<td>Kristiansen, Lars</td>
<td>321, 342</td>
</tr>
<tr>
<td>Kudinov, Oleg V.</td>
<td>216</td>
</tr>
<tr>
<td>Lambert, James</td>
<td>226</td>
</tr>
<tr>
<td>Lee, Kyung Il</td>
<td>343</td>
</tr>
<tr>
<td>Lewis, Andrew E. M.</td>
<td>335</td>
</tr>
<tr>
<td>Li, Chung-Chih</td>
<td>236, 344</td>
</tr>
<tr>
<td>Linares, David</td>
<td>256</td>
</tr>
<tr>
<td>Liu, Jiamou</td>
<td>345</td>
</tr>
<tr>
<td>Lovett, Neil B.</td>
<td>330</td>
</tr>
<tr>
<td>Ma, Lili</td>
<td>307</td>
</tr>
<tr>
<td>Ma, Yanfang</td>
<td>246</td>
</tr>
<tr>
<td>Mainhardt, Gunther</td>
<td>346</td>
</tr>
<tr>
<td>Makowsky, J. A.</td>
<td>341</td>
</tr>
<tr>
<td>Marcial-Romero, Raymundo</td>
<td>256</td>
</tr>
<tr>
<td>Mender, Bedeho Mesghina Wolde</td>
<td>342</td>
</tr>
<tr>
<td>Montes-Venegas, Héctor A.</td>
<td>256</td>
</tr>
<tr>
<td>Nagy, Benedek</td>
<td>266</td>
</tr>
<tr>
<td>Nebel, Markus</td>
<td>48</td>
</tr>
<tr>
<td>Ng, Keng Meng</td>
<td>15</td>
</tr>
<tr>
<td>Normann, Dag</td>
<td>347</td>
</tr>
<tr>
<td>Ospina, Juan</td>
<td>296</td>
</tr>
<tr>
<td>Pauly, Arno</td>
<td>348</td>
</tr>
<tr>
<td>Pelupessy, Florian</td>
<td>276</td>
</tr>
<tr>
<td>Pessa, E.</td>
<td>359</td>
</tr>
<tr>
<td>Petre, Ion</td>
<td>16</td>
</tr>
<tr>
<td>Pinna, G. Michele</td>
<td>279</td>
</tr>
<tr>
<td>Policriti, Alberto</td>
<td>17</td>
</tr>
<tr>
<td>Quinn, Sara</td>
<td>349</td>
</tr>
<tr>
<td>Radha, N.</td>
<td>350</td>
</tr>
<tr>
<td>Reimann, Jan</td>
<td>351</td>
</tr>
<tr>
<td>Rosamond, Frances</td>
<td>331</td>
</tr>
<tr>
<td>Roy, Daniel M.</td>
<td>317</td>
</tr>
<tr>
<td>Rybakov, Vladimir</td>
<td>38</td>
</tr>
<tr>
<td>Safarik, Pavol</td>
<td>352</td>
</tr>
<tr>
<td>Sanseverino, Eleonora Riva</td>
<td>79</td>
</tr>
<tr>
<td>Sato, Kentaro</td>
<td>353</td>
</tr>
<tr>
<td>Schmidt, Renate</td>
<td>38</td>
</tr>
<tr>
<td>Schuster, S.</td>
<td>186</td>
</tr>
<tr>
<td>Selivanov, Victor L.</td>
<td>216</td>
</tr>
<tr>
<td>Semukhin, Pavel</td>
<td>289</td>
</tr>
<tr>
<td>Shafer, Paul</td>
<td>354</td>
</tr>
<tr>
<td>Shagrir, Oron</td>
<td>18</td>
</tr>
<tr>
<td>Shore, Richard A.</td>
<td>19</td>
</tr>
<tr>
<td>Soltys, Michael</td>
<td>162</td>
</tr>
<tr>
<td>Sorbi, Andrea</td>
<td>6, 335</td>
</tr>
<tr>
<td>Sorkin, Rafael</td>
<td>7</td>
</tr>
<tr>
<td>Soskov, Ivan N.</td>
<td>355, 356</td>
</tr>
<tr>
<td>Soskova, Alexandra A.</td>
<td>355</td>
</tr>
<tr>
<td>Soskova, Mariya I.</td>
<td>356</td>
</tr>
<tr>
<td>Steinová, Monika</td>
<td>331</td>
</tr>
<tr>
<td>Stukachev, Alexey</td>
<td>357</td>
</tr>
<tr>
<td>Tang, Shaoting</td>
<td>307</td>
</tr>
<tr>
<td>Uetz, Marc</td>
<td>20</td>
</tr>
<tr>
<td>Vélez, Mario</td>
<td>296</td>
</tr>
<tr>
<td>Vazirani, Vijay V.</td>
<td>8</td>
</tr>
<tr>
<td>Venkatesulu, M.</td>
<td>28</td>
</tr>
<tr>
<td>Voda, Paul J.</td>
<td>206</td>
</tr>
<tr>
<td>Wagner, Robert C.</td>
<td>330</td>
</tr>
<tr>
<td>Wei, Wei</td>
<td>140</td>
</tr>
<tr>
<td>Weiermann, A.</td>
<td>336</td>
</tr>
<tr>
<td>Weiermann, Andreas</td>
<td>276, 327</td>
</tr>
<tr>
<td>Westhead, David R.</td>
<td>22</td>
</tr>
<tr>
<td>Xin, Jiang</td>
<td>307</td>
</tr>
<tr>
<td>Zapreev, Ivan S.</td>
<td>110</td>
</tr>
<tr>
<td>Zepeda, Claudia</td>
<td>256</td>
</tr>
<tr>
<td>Zerjatke, Thomas</td>
<td>358</td>
</tr>
<tr>
<td>Zhang, Zhanli</td>
<td>307</td>
</tr>
<tr>
<td>Zheng, Zhiming</td>
<td>140, 307</td>
</tr>
<tr>
<td>Zhukov, Anton V.</td>
<td>216</td>
</tr>
<tr>
<td>Zizzi, P.</td>
<td>359</td>
</tr>
</tbody>
</table>