
19. Nichtdeterministische Turingmaschinen und ihre Komplexität

Bei einem Turingmaschinenprogramm P aus bedingten Anweisungen wird durch die Forderung

$$(i, a, b, B, j) \neq (i', a', b', B', j') \Rightarrow (i, a) \neq (i', a')$$

sichergestellt, dass zu jeder (Nichtstopp-)Konfiguration die Nachfolgekonfiguration eindeutig bestimmt ist, d.h. die Maschine deterministisch ist (vgl. Abschnitt 5). Geben wir diese Forderung auf, so erhalten wir *nichtdeterministische (nd.) Turingmaschinen*. Während eine deterministische Turingmaschine M und eine Eingabe x die M -Rechnung eindeutig festlegen, definieren eine nichtdeterministische Turingmaschine $N = (B, P)$ und eine Eingabe x einen *Rechenbaum*, in dessen Wurzel die Startkonfiguration von N bei Eingabe x steht, bei dem die Söhne mit den möglichen Nachfolgekonfigurationen der Konfiguration im Vaterknoten markiert sind, und die Blätter die erreichbaren Stoppkonfigurationen enthalten¹. Jeder Pfad im Rechenbaum (endlich von der Wurzel zu einem Blatt oder unendlich von der Wurzel ausgehend) ist eine *mögliche Rechnung* von N bei Eingabe x . Da verschiedene Rechnungen zu verschiedenen Ergebnissen führen können, muss man festlegen, welche Sprache von N erkannt bzw. welche Funktion von N berechnet wird. Wir beschränken uns hier auf den Fall von Sprachen und definieren, dass x in der von N erkannten Sprache L_N liegt, falls wenigstens eine Rechnung zu diesem Ergebnis kommt. Es gibt hier also eine Asymmetrie zwischen Akzeptieren ($x \in L_N$) und Verwerfen ($x \notin L_N$): Akzeptiert eine Rechnung (lokal), so wird (global) akzeptiert. Das (lokale) Verwerfen einer Rechnung führt jedoch nicht automatisch zum (globalen) Verwerfen.

Da wir hier nur Sprachen und keine Funktionen betrachten, definieren wir Akzeptanz und Verwerfen dadurch, dass wir die Stoppzustände in akzeptierende und verwerfende Zustände aufteilen (statt die Werte 1 und 0 der charakteristischen Funktion der erkannten Sprache auszugeben). Dies wird im Folgenden ebenso für deterministische Maschinen gemacht.

Ist jede mögliche Rechnung von N bei jeder möglichen Eingabe endlich, so heißt N total. Eine totale nichtdeterministische Turingmaschine N ist $f(n)$ -zeit- bzw. platzbeschränkt, wenn für jede mögliche Rechnung von N bei einer Eingabe x der Länge n , die Länge der Rechnung bzw. deren Platzbedarf durch $f(n)$ beschränkt ist. Hiermit lassen sich die nichtdeterministischen Zeit- und Platzklassen analog zum deterministischen

¹Man beachte, dass der Verzweigungsgrad d des Rechenbaumes, d.h. die maximale Anzahl der Söhne eines Vaters, endlich ist. Hierbei ist d durch die maximale Anzahl von Instruktionen in P mit gleichen ersten beiden Komponenten beschränkt.

Fall definieren:

$$\begin{aligned} \text{NTIME}(f(n)) &= \{L \subseteq \Sigma_2^* : \text{Es gibt eine nd. } f(n)\text{-zeitbeschränkte} \\ &\quad \text{Mehrband-TM } N, \text{ die } L \text{ erkennt}\} \\ \text{NSPACE}(f(n)) &= \{L \subseteq \Sigma_2^* : \text{Es gibt eine nd. } f(n)\text{-platzbeschränkte} \\ &\quad \text{Mehrband-TM } N, \text{ die } L \text{ erkennt}\} \end{aligned}$$

Durch Betrachtung von Familien von Schrankenfunktionen erhalten wir allgemeine nichtdeterministische Komplexitätsklassen wie z.B. die folgenden nichtdeterministischen Gegenstücke zu den deterministischen Komplexitätsklassen LIN, P, LOGSPACE und PSPACE:

$$\begin{aligned} \text{NLIN} &= \text{NTIME}(O(n)) \\ \text{NP} &= \bigcup_p \text{Polynom} \text{NTIME}(p(n)) \\ \text{NLOGSPACE} &= \text{NSPACE}(O(\log(n))) \\ \text{NPSPACE} &= \bigcup_p \text{Polynom} \text{NSPACE}(p(n)) \end{aligned}$$

Die Beweise der meisten in den vorhergehenden Abschnitten gezeigten Ergebnisse benutzen nicht, dass die betrachteten Maschinen deterministisch sind, weshalb sich die Ergebnisse auf nichtdeterministische Maschinen direkt übertragen lassen. Dies gilt insbesondere für die Bandreduktionssätze, Lineare Beschleunigung und Kompression, Normierung und Existenz universeller nichtdeterministischer Turingmaschinen. Weiter sind die nichtdeterministische Zeit- und Platzkomplexität der verschiedenen Turingmaschinenmodelle allgemeine Komplexitätsmaße im Sinne von Abschnitt 16. Die Hierarchiesätze gehören zu den wenigen Ergebnissen, deren Beweise für die nichtdeterministische Komplexität nicht unmittelbar durchgehen. Diese werden durch Diagonalisierung gezeigt, d.h. das Akzeptanzverhalten einer simulierten Maschine verändert. Dies ist bei deterministischen Maschinen lokal möglich, nicht aber – wie schon oben bemerkt – bei nichtdeterministischen Maschinen. So sind die deterministischen Zeit- und Platzkomplexitätsklassen trivialerweise gegen Komplement abgeschlossen. Im Falle nichtdeterministischer Maschinen konnte dagegen der Abschluss der Platzklassen (mit konstruierbaren Schranken) erst vor wenigen Jahren gezeigt werden (Beweis in der Vorlesung „Komplexitätstheorie“).

19.1 SATZ. (SATZ VON IMMERMANN UND SZELEPCSENYI, 1988) *Für platzkonstruierbares* $s(n) \geq \log(n)$ *gilt* $\text{NSPACE}(s(n)) = \text{co-NSPACE}(s(n))$.

Hierbei ist die co-Klasse $\text{co-}C$ einer Komplexitätsklasse C die Menge der Komplemente von Sprachen in C , d.h. $\text{co-}C = \{\bar{A} : A \in C\}$.

Für die nichtdeterministische Zeitkomplexität ist die Frage des Komplementabschlusses offen, und man vermutet, dass für konstruierbare Zeitschranken $t(n) > n$ die Klasse $\text{NTIME}(t(n))$ nicht gegen Komplement abgeschlossen ist. (Aus Korollar 16.10 folgt, dass $\text{DTIME}(t(n)) = \text{NTIME}(t(n))$ – und damit $\text{NTIME}(t(n)) = \text{co-NTIME}(t(n))$ – für geeignet gewählte beliebig große t gilt. Diese Schranken t sind aber nicht konstruierbar.)

Satz 19.1 ermöglicht die Übertragung des Platzhierarchiesatzes auf den nichtdeterministischen Fall. Im Fall der nichtdeterministischen Zeit hat man mit anderen Methoden Hierarchiesätze bewiesen, die aber nicht exakt dem deterministischen Zeithierarchiesatz entsprechen. Zeigen kann man jedoch z.B., dass für jedes $k \geq 1$

$$\text{NTIME}(n^k) \subsetneq \text{NTIME}(n^{k+1}) \subsetneq \text{NTIME}(2^n)$$

gilt.

Über das Verhältnis zwischen den deterministischen und nichtdeterministischen Komplexitätsmaßen sind im Wesentlichen nur die folgenden allgemeinen Ergebnisse bekannt.

Da jede deterministische Maschine auch eine nichtdeterministische Maschine ist, und die Rechenzeit einer Maschine deren Platzbedarf beschränkt, gilt trivialerweise

$$\begin{aligned} \text{DTIME}(f(n)) &\subseteq \text{DSPACE}(f(n)) \\ \cap_1 & \qquad \qquad \cap_1 \\ \text{NTIME}(f(n)) &\subseteq \text{NSPACE}(f(n)) \end{aligned} \quad (19.1)$$

Umgekehrt haben wir schon gesehen, dass sich DSPACE durch DTIME wie folgt beschreiben lässt:

$$\text{DSPACE}(f(n)) \subseteq \text{DTIME}(2^{O(f(n))}) \quad (19.2)$$

(vgl. Korollar 17.11), und analog zeigt man

$$\text{NSPACE}(f(n)) \subseteq \text{NTIME}(2^{O(f(n))}). \quad (19.3)$$

Will man nichtdeterministische durch deterministische Zeit nach oben abschätzen, so kann man

$$\text{NTIME}(f(n)) \subseteq \text{DTIME}(2^{O(f(n))}) \quad (19.4)$$

zeigen. Hierzu genügt es, mit einer Tiefensuche den Rechenbaum einer nichtdeterministischen $f(n)$ -zeitbeschränkten Turingmaschine N deterministisch zu durchlaufen, bis entweder eine akzeptierende Konfiguration gefunden oder der gesamte Baum ergebnislos durchsucht wurde. Da der Rechenbaum von N Tiefe $f(n)$ hat und endlich verzweigend ist, ist die Größe des Baumes von der Ordnung $2^{O(f(n))}$.

Für den Vergleich von nichtdeterministischem und deterministischem Platz liefert ein naives Verfahren ebenfalls einen exponentiellen Overhead. Hier kann man jedoch für konstruierbare Schranken mit einem Divide-and-Conquer-Verfahren diesen Overhead drastisch reduzieren.

19.2 SATZ. (SATZ VON SAVITCH) Für platzkonstruierbares $s(n) \geq \log(n)$ gilt $\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2)$.

(Beweis in der Vorlesung „Komplexitätstheorie“.) Dies zeigt z.B., dass

$$\text{PSPACE} = \text{NPSPACE},$$

da das Quadrat eines Polynoms wiederum ein Polynom ist. Die Beobachtungen (19.2)-(19.4) lassen sich durch den Vergleich von nichtdeterministischem Platz und deterministischer Zeit wie folgt verschärfen:

19.3 SATZ. Für rekursives $f(n) \geq \log(n)$ gilt

$$\text{NSPACE}(f(n)) \subseteq \text{DTIME}(2^{O(f(n))}).$$

BEWEIS. Sei $L \in \text{NSPACE}(f(n))$. Wegen der linearen Beschleunigung genügt es, eine deterministische $O(2^{O(f(n))})$ -zeitbeschränkte Mehrband-Turingmaschine M anzugeben, die L erkennt.

Nach Annahme gibt es eine nichtdeterministische $f(n)$ -platzbeschränkte Turingmaschine N , die L erkennt. Wie im Beweis von Korollar 17.11 (ii) können wir für eine Eingabe x der Länge n die Kardinalität der Menge $\text{KON}_N(x)$ der theoretisch denkbaren Konfigurationen in einer N -Rechnung bei Eingabe x (d.h. der Platzschranke $f(n)$ genügenden Konfigurationen mit x auf dem Eingabeband) durch

$$|\text{KON}_N(x)| \leq 2^{cf(n)} \quad (c \text{ geeignet}) \quad (19.5)$$

(für hinreichend großes n) abschätzen.

Da sich in einer N -Rechnung eine Konfiguration nicht wiederholen kann (sonst würde N unendliche Rechnungen besitzen, im Widerspruch zur Totalität von N), ist die Länge jeder N -Rechnung bei Eingabe x durch $2^{cf(n)}$ beschränkt. Da die Größe des Rechenbaums exponentiell in seiner Tiefe beschränkt ist, kann diese also durch $2^{2^{O(f(n))}}$ beschränkt werden. Ein vollständiges Durchsuchen des Baumes nach einer akzeptierenden Konfiguration (wie zum Nachweis von (19.4) oben) führt also zu einer doppelt exponentiellen und nicht zu der gewünschten einfach exponentiellen Zeitschranke. Um letztere zu erhalten beobachtet man, dass wegen (19.5) viele Knoten im Rechenbaum mit den gleichen Konfigurationen markiert sind. Da die Teilbäume unterhalb von zwei Knoten mit gleicher Markierung identisch sind, kann man die redundanten Teilbäume aus dem Rechenbaum herausschneiden, d.h. in einem Breitensuchdurchlauf durch den Rechenbaum dünnt man diesen aus, indem man Teilbäume mit einer zuvor schon gesehenen Konfiguration in der Wurzel eliminiert. So erhält man einen bzgl. der Akzeptanz äquivalenten Rechenbaum, in dem jede Konfiguration höchstens einmal vorkommt, der also nach (19.5) höchstens die Größe $2^{cf(n)}$ hat.

Am einfachsten lässt sich dieses Verfahren mit Hilfe einer Liste *LISTE* implementieren, in die man die Konfigurationen des Rechenbaums bei ihrem erstmaligen Vorkommen aufnimmt. Gehen wir o.B.d.A. davon aus, dass jede Nichtstopp-Konfiguration C genau zwei Nachfolge-Konfigurationen C' , C'' besitzt, so kann man diese Liste induktiv wie folgt anlegen. (Dabei sei

$$\text{LISTE} = \langle C_0, C_1, \dots, C_n \rangle$$

die jeweils aktuelle Liste und l deren Länge.)

```

LISTE := < StartN(x) >; l := 1; Zeiger := 0;
WHILE zeiger < l DO {
  IF CZeiger keine Stoppkonfiguration THEN {
    IF C'Zeiger ∉ LISTE THEN {
      LISTE := LISTE * < C'Zeiger >;
      l := l + 1;
    }
    IF C''Zeiger ∉ LISTE THEN {
      LISTE := LISTE * < C''Zeiger >;
      l := l + 1;
    }
  }
  IF CZeiger akzeptierende Konfiguration THEN AKZEPTIERE;
  Zeiger := Zeiger + 1;
}
VERWERFE.

```

Da $l \leq 2^{c f(n)}$ und die Länge jeder Konfiguration in *LISTE* durch $O(f(n))$ beschränkt ist, lässt sich dieses Verfahren in $O(2^{O(f(n))})$ Schritten ausführen. \square