A Turing acceptor, or TA for short, is a sixtupel
\[ M = (Q, \Sigma, \Gamma, \Delta, s, F), \]
where
- \( Q \) is a finite set of states,
- \( \Sigma \) is the input alphabet,
- \( \Gamma \) is the tape alphabet, where \( \Sigma \subseteq \Gamma \) and \( \Box \in \Gamma \setminus \Sigma \),
- \( \Delta \) is the transition relation or program of \( M \), where
  \( \Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \text{Move}) \) and \( \text{Move} = \{ L, R, S \} \),
- \( s \in Q \) is the initial state,
- \( F \subseteq Q \) is the set of accepting states.

Each such tuple in \( \Delta \) is referred to as an instruction of \( M \).

Turing acceptors and r.e. languages

Let \( (Q, \Sigma, \Gamma, \Delta, s, F) \) be a TA with transition relation
\[ \Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \text{Move}). \]

A single instruction \((q, a, q', a', Z)\) in \( \Delta \) has the meaning that
- the instruction may be executed whenever \( q \) is the current state and \( a \in \Gamma \) is the symbol currently read on the tape,
- the instruction is executed
  by going from state \( q \) into state \( q' \),
  by replacing the currently read tape symbol \( a \) by \( a' \), and
  by finally performing the move \( Z \).

TAs are in general nondeterministic, i.e., for any given pair \((q, a)\) the program \( \Delta \) may contain several (or no) instructions of the form \((q, a, q', a', Z)\).

The computation model TA is equivalent to the computation model obtained from LBAs by removing the space restrictions.

Turing acceptors and r.e. languages

A Turing acceptor
- has a single tape, which is infinite in both directions,
- the tape has a single two-way read-write head, that is, the head can move in both directions and symbols on the tape can be overwritten,
- initially the symbols of the input \( w \) are written from left to right into \(|w|\) consecutive tape cells, while all other tape cells are marked with the blank symbol \( \Box \),
- initially \( M \) is in the initial state \( s \) and the head is positioned on the first symbol of the input,
- each instruction specifies a subsequent state, a symbol to be written at the current tape position, and a move,
- a move is either to stay (S) at the current position, or to move one position left (L) or right (R).

Remark

During the computation of a TA, after any given number of steps almost all tape cells are marked with the blank symbol.

For a proof, observe that the condition under consideration holds initially and that during each step of the computation at most one blank symbol may be overwritten.

The preceding remark suggests to natural ways to represent tape contents of TAs as finite objects i.e.,
- A as functions from the integers to the tape alphabet where almost all integers are mapped to \( \Box \),
- B as a word that is equal to the inscription of a block of finitely many subsequent tape cells where the block comprises all cells that are not marked with \( \Box \).

In the sequel we will mainly work with representation \( B \).
Definition (Relevant tape content)

Let \( M \) be a TA with working alphabet \( \Gamma \). A tape content of \( M \) is a function from \( f : \mathbb{Z} \to \Gamma \) such that \( f(i) \neq \square \) for almost all integers \( i \) (i.e., for all but finitely many \( i \in \mathbb{Z} \)).

Given a tape content \( f \), let \( I_f = \{ i \in \mathbb{Z} : f(i) \neq \square \} \) be the set of integers that \( f \) does not map to \( \square \), and let the relevant part of \( f \) be the word

\[
u_f = f(\min I_f) \cdots f(\max I_f)\]

over \( \Gamma \)

where we let \( u_f = \lambda \) in case \( f \) is constant with value \( \square \).

A tape content \( f \) is represented by any word \( u \) that has the form \( u = \square^{r_1} u_f \square^{r_2} \) for natural numbers \( r_1 \) and \( r_2 \), i.e., \( u \) is equal to the relevant part of \( f \) plus possibly finitely many leading or trailing blank symbols.

Definition (Transition relation of a TA)

Let \( M = (Q, \Sigma, \Gamma, \Delta, s, F) \) be a TA. For all states \( q \) and \( q' \) in \( Q \), for all words \( u \) and \( u' \) over \( \Gamma \), and for all \( i \) where \( 1 \leq i \leq |u| \), let

\[
(q, u, i) \xrightarrow{M} (q', u', i'),
\]

in case \( u = a_1 \cdots a_n \) and there is an instruction \((q, a_i, q', a', \Delta)\)

in \( \Delta \) such that for \( u^1 = a_0 \cdots a_{i-1} \cdot a_i \cdot a_{i+1} \cdots a_n \) we have

\[
u' = \begin{cases}
\square \bar{u} & \text{in case } Z = L \text{ and } i = 1,
\bar{u} \square & \text{in case } Z = R \text{ and } i = n,
\bar{u} & \text{otherwise,}
\end{cases}
\]

and

\[
i' = \begin{cases}
1 & \text{in case } Z = L \text{ and } i = 1,
i - 1 & \text{in case } Z = L \text{ and } i \neq 1,
i & \text{in case } Z = S,
i + 1 & \text{in case } Z = R.
\end{cases}
\]

Definition (Configuration of a Turing acceptor)

Let \( M = (Q, \Sigma, \Gamma, \Delta, s, F) \) be a TA. A configuration of \( M \) is a triple

\[
(q, u, i) \in Q \times \Gamma^* \times \{1, \ldots, |u|\}
\]

The initial configuration on input \( w \) is \((s, w, 1)\)

A configuration \((q, u, i)\) represents a situation where

\( q \) is the current state of the computation,

\( u \) is a representation of the relevant tape content,

\( i \) is the position of the head within \( u \)

(e.g., \( i = 1 \) means that the head is on the first symbol of \( u \)).

Definition (Computations of a TA)

Let \( M = (Q, \Sigma, \Gamma, \Delta, s, F) \) be a TA.

A computation of \( M \) of length \( t \) is a sequence \( C_0, \ldots, C_t \) of configurations such that

\[
C_0 \xrightarrow{M} C_1 \xrightarrow{M} \cdots \xrightarrow{M} C_t.
\]

We write \( C \xrightarrow{M, t} C' \) if there is such a computation where \( C = C_0 \) and \( C' = C_t \).

We write \( C \xrightarrow{M, s} C' \) if \( C \xrightarrow{M, t} C' \) for some \( t \geq 0 \).

A stop configuration of \( M \) on inputs of length \( n \) is a configuration \((q, u, i)\) such that for \( u = a_1 \cdots a_n \) there is no instruction of the form \((q, a_i, \ldots)\) in \( \Delta \).
Definition (Language recognized by a TA)

Let $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ be a TA.

A terminating computation of $M$ is a computation that starts with the initial configuration $(s, w, 1)$ of some input $w$ and ends with a stop configuration.

An accepting computation is a terminating computation where the state of the last configuration, the stop configuration, is an accepting state.

The TA $M$ accepts an input $w$ if there is some accepting computation that starts with the initial configuration of input $w$.

The language recognized by the TA $M$ is

$$L(M) = \{ w \in \Sigma^* : M \text{ accepts } w \}.$$
Turing acceptors and r.e. languages

A grammar \( G \) generating \( L(M) \) works as follows.

The rules of \( G \) allow to generate arbitrary initial configuration with doubled input, followed by simulations of arbitrary steps of \( M \).

Each sentential form derived by \( G \) is equal to the relevant part of the tape content of the corresponding computation of \( M \).

Since the tape of \( M \) contains the information about current head position and state, the rules of \( G \) can be chosen such that exactly the computations steps of \( M \) as simulated by \( M \) can be performed.

On simulating an accepting computation, finally the tape content can be transformed into the input \( w \) preserved on the first track. \( \square \)

Context-sensitive languages and r.e. languages

Theorem (Context-sensitive and decidable languages)

Every context-sensitive language is decidable.

Proof. For a given context-sensitive language \( L \) pick a length-increasing grammar \( G = (N, T, P, S) \) where \( L = L(G) \).

If we let \( k = |N| + |T| \geq 2 \), then the number of sentential forms of \( G \) of length at most \( n \) is \( 1 + k + k^2 + \cdots + k^n \leq k^{n+1} \).

For every derivation of a word of length \( n \) in \( G \), it holds that
- all occurring sentential forms have length of at most \( n \),
- in case the derivation has minimum length among all derivations of this word, no sentential form occurs twice.

So \( L \) is decidable because in order to check whether a given word of length \( n \) can be derived over \( G \), it suffices to simulate all derivations of length at most \( k^{n+1} \), \( \square \)

Context-sensitive languages and r.e. languages

Proof, cont.: Fix a listing \( G_0, G_1, \ldots \) of length-increasing grammars with terminal alphabet \( \{0,1\} \) that is
- universal, i.e., for every grammar \( G \) of the latter type there is some index \( e \) such that \( L(G) = L(G_e) \),
- effective in the sense that there is a total deterministic Turing acceptor \( M \) that recognizes the language

\[
L = \{0^e1w : w \in L(G_e)\} \subseteq \{0,1\}^*.
\]

In particular, the language \( L \), as well as the language
\[
D = \{0^e : 0^e \notin L(G_e)\} = \{0^e : 0^e10^e \notin L\}
\]
is again decidable but differs by construction from each language \( L(G_e) \) at the word \( 0^e \).

A total deterministic Turing acceptor that recognizes \( D \) will accept an input of the form \( 0^e \) if and only if \( M \) rejects \( 0^e10^e \), and will reject all other inputs. \( \square \)
Chomsky hierarchy

Definition (Chomsky hierarchy)
The *Chomsky hierarchy* consists of the following four classes

- REG = CH(3) = the class of all regular languages,
- CF = CH(2) = the class of all context-free languages,
- CS = CH(1) = the class of all context-sensitive languages,
- RA = CH(0) = the class of all r.e. languages.

Furthermore, we define the two related classes

- LIN = the class of all linear languages,
- REC = the class of all decidable languages.

Context-sensitive languages and r.e. languages

Theorem (Chomsky hierarchy)

The classes of the Chomsky hierarchy form a strict hierarchy, we have

\[ \text{REG} \subsetneq \text{CF} \subsetneq \text{CS} \subsetneq \text{RA}. \]

Proof. The first and third inclusion are immediate by the characterizations of the involved classes in terms of grammars.

The second inclusion holds because every context-free language is generated by a context-free grammar in Chomsky normal form, which is \( \lambda \)-separated and hence length-increasing.

The last noninclusion holds by the theorem above, the first two noninclusions hold by the previously verified counterexamples

\[
\{0^n1^n : n > 0\} \in \text{CF} \setminus \text{REG} \quad \text{and} \quad \{0^n1^n0^n : n > 0\} \in \text{CF} \setminus \text{CS}.
\]

Closure properties of r.e. languages

Theorem (Closure under union and intersection)

The class of r.e. languages is closed under union and intersection.

Proof. Given two r.e. languages \( L_1 \) over \( T_1 \) and \( L_2 \) over \( T_2 \), fix grammars

\[
G_1 = (N_1, T_1, S_1, P_1) \quad \text{and} \quad G_2 = (N_2, T_2, S_2, P_2)
\]

that generate \( L_1 \) and \( L_2 \), respectively. We can assume \( N_1 \cap N_2 = \emptyset \).

Then the grammar \( G = (N, T_1 \cup T_2, S, P) \) where \( S \notin N_1 \cup N_2 \),

\[
N = N_1 \cup N_2 \cup \{S\} \quad \text{and} \quad P = P_1 \cup P_2 \cup \{S \rightarrow S_1|S_2\}
\]
generates the language \( L_1 \cup L_2 \).

In connection with the latter assertion, recall our convention that the left-hand side of each rule must contain some variable symbol, hence each derivation in \( G \) uses only rules from either \( P_1 \) or \( P_2 \).
Lemma (Closure under complementation)

A language $L$ is decidable if and only if $L$ and the complement $\overline{L}$ of $L$ are both recursively enumerable.

Proof: If $L$ is decidable, then so is $\overline{L}$, hence both languages are r.e. Conversely, if $L$ and $\overline{L}$ are r.e., we obtain a decision procedure for $L$. For a given input $w$, run an enumeration procedure for $L$ and for $\overline{L}$ in parallel until $w$ occurs in either of them. Accept if $w$ occurs in the enumeration of $L$, otherwise reject. $\square$

Theorem (Closure under complementation)

The class of r.e. languages is not closed under complementation.

Proof. The halting problem $H$ is r.e. but its complement $\overline{H}$ is not. If $H$ were r.e., then $H$ would be decidable, a contradiction. $\square$

Closure properties of r.e. languages

Proof cont. More precisely, given a recursively enumerable language $L$ pick some TA $M$ such that $L = L(M)$.

As outlined above, we construct a TA $M^*$ that recognizes $L^*$.

On input $w$, first $M^*$ scans $w$ and guesses nondeterministically subwords $w_1, \ldots, w_t$ of $w$ such that $w = w_1 \cdots w_t$.

Then $M^*$ successively simulates $M$ on the inputs $w_1, \ldots, w_t$, using a separate track of the tape.

The input $w$ is accepted by some computation of $M^*$ if and only if in this computation all simulated computations accept.

By construction, we have $L(G) = L^*$ because $M^*$ will on input $w \in L^*$ accept for some guessed partition of $w$, $w \notin L^*$ accept for no partition of $w$. $\square$

Theorem (Closure under concatenation and Kleene closure)

The class of recursively enumerable languages is closed under concatenation, and Kleene closure, i.e.,

(i) if $L_1$ and $L_2$ are r.e., then $L_1L_2$ is r.e.,
(ii) if $L$ is r.e., then $L^*$ is r.e.,

Proof. We give the proof of assertion (ii) and omit the essentially identical considerations for assertion (i).

We construct a TA that recognizes $L^*$, which

first guesses nondeterministically a partition of the input $w$ of the form $w = w_1 \cdots w_t$,

then simulates successively for inputs $w_1, \ldots, w_t$ a TA that recognizes $L$, moving to $w_{i+1}$ only in case the simulation with input $w_i$ has been accepting, and accepts $w$ if all simulations were accepting.

$\square$

Recall that given a homomorphism $h: \Sigma_1^* \rightarrow \Sigma_2^*$, the inverse homomorphism $h^{-1}$ maps any language $L$ over $\Sigma_2$ to the language

$h^{-1}[L] = \{w \in \Sigma_1^* : h(w) \in L\}$.

Theorem (Closure under inverse homomorphisms)

The class of context-sensitive languages is closed under homomorphisms and under inverse homomorphisms.

Proof. Let $h: T_1 \rightarrow T_2$ be a homomorphism.

Let $L_1$ be a recursively enumerable language over alphabet $T_1$. Fix some enumeration procedure for $L_1$ with output $w_0w_1, \ldots$. Then $h(w_0), h(w_1), \ldots$ is an enumeration procedure for $h[L_1]$.

Conversely, let $L_2$ be a recursively enumerable language that is accepted by a Turing acceptor $M_2$.

A Turing acceptor $M_1$ that accepts $h^{-1}[L_2]$ works as follows. On input $w$, $M_1$ simulates $M_2$ on input $h(w)$ and copies the result of this simulation. $\square$