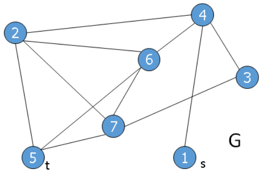
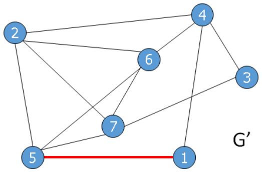
Reducibility

Reducibility

- **Eulerian path (resp., cycle) problem:** Given graph G and two nodes s and t, determine whether there is a path from s to t (resp., cycle from s to s) visiting each edge in G exactly once.
- To answer Eulerian path problem for G ($EP(G)$), construct a graph G' that is identical to G except an additional edge between s and t.
  - If $EC(G')$ returns true, there is a Eulerian path from s to t.
  - If $EC(G')$ returns false, there is no Eulerian path from s to t.
- We use $EC(G')$ as a subroutine.
- We say the Eulerian path problem is reduced to the Eulerian cycle problem, abbrev. as $EP \leq EC$.



1→4 →2→5 →6 → 2 → 7→3→4→6→7→5



1→4 →2→5 →6 → 2 → 7→3→4→6→7→5→1

Reducibility

- Let us say A and B are two problems and A is reduced to B (or equivalently, B is reduced from A).
- Notation-wise, we often write $A \leq B$.
- If we solve B, we solve A as well.
  - B is easy $\to$ A is easy.
  - If we solve the Eulerian cycle problem, we solve the Eulerian path problem.
- If we can't solve A, we can't solve B.
  - A is hard $\to$ B is hard.
- To show a problem P is not decidable, it suffices to reduce A_{TM} to P.
- We will give examples in this chapter.

The Halting Problem for Turing Machines

- The halting problem is to test whether a TM M halts on a string w.
- As usual, we first give a language-theoretic formulation.

$$HALT_{\mathrm{TM}} = \{\langle M, w\rangle : M \text{ is a TM and } M \text{ halts on the input } w\}.$$

Theorem 1

$HALT_{TM}$ is undecidable.

Proof.

Suppose TM R decides $HALT_{\mathrm{TM}}$. Consider TM S using R as subroutine
S = "On input $\langle M, w\rangle$ where M is a TM and w is a string:

1. Run TM R on the input $\langle M, w\rangle$.
2. If R rejects, reject.
3. If R accepts, simulate M on w until it halts.
4. If M accepts, accept; if M rejects, reject." □

Then S decides A_{TM} - a contradiction.

Emptiness Problem for Turing Machines

- Consider $E_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } L(M) = \emptyset\}$.

Theorem 2

E_{TM} is undecidable.

Proof.

Suppose TM R decides E_{TM}. Consider TM S using R as subroutine
$S =$ "On input $\langle M, w \rangle$ where M is a TM and w a string:

1. Use $\langle M, w \rangle$ to construct
   $M_1 =$ "On input x:
   1. If $x \neq w$, reject.
   2. If $x = w$, run M on the input x (=w). If M accepts x, accept."
2. Run R on the input $\langle M_1 \rangle$ to test whether $L(M_1) = \emptyset$.
3. If R accepts [M rejects w], reject; otherwise [M accepts w], accept." □

Then A_{TM} is decidable - a contradiction.

Regularity Problem for Turing Machines

$$REGULAR_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } L(M) \text{ is regular}\}.$$

Theorem 3

$REGULAR_{TM}$ is undecidable.

Proof.

Let R be TM deciding $REGULAR_{\text{TM}}$. Consider S using R as subroutine
$S =$ "On input $\langle M, w \rangle$ where M is a TM and w a string:

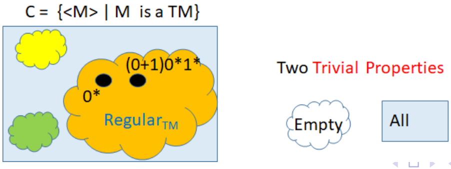
1. Use $\langle M, w \rangle$ to construct
   $M_2 =$ "On input x:
   1. If x is of the form 0^n1^n, accept.
   2. Otherwise, run M on the input w. If M accepts w, accepts." (In this case, $L(M_2) = \Sigma^*$)
2. Run R on the input $\langle M_2 \rangle$.
3. If R accepts $[L(M_2) = \Sigma^*]$, accept; otherwise $[L(M_2) = \{0^n1^n\}]$, reject."

□

Rice's Theorem

Consider the language $\mathbb{C}$ of all TMs, i.e., $\mathbb{C} = \{\langle M \rangle \mid M \text{ is a TM}\}$.

- A property P is a subset of $\mathbb{C}$ such that if $L(M_1) = L(M_2)$ then either $\langle M_1 \rangle \in P \Leftrightarrow \langle M_2 \rangle \in P$.
  - $REGULAR_{TM}$, i.e., the set of all TMs that accept regular languages, is a property.
  - $P' = \{\langle M \rangle \mid M \text{ has more than 100 states}\}$ is NOT a property.
- A property P is trivial if (1) $P = \emptyset$, or (2) P=$\mathbb{C}$.
  - P is non-trivial $\Leftrightarrow \exists M_1, M_2, \langle M_1 \rangle \in P$ and $\langle M_2 \rangle \in \overline{P}$.
  - $\{\langle M \rangle \mid L(M) = \emptyset\}$ is NOT a trivial property.
- **Goal**: given a TM M, decide whether $\langle M \rangle \in P$.
- **Rice's theorem**: Undecidable, unless P is a trivial property.
- Why trivial properties are decidable?



Rice's Theorem

Theorem 4

Let P be a language consisting of TM descriptions such that

1. *P is not trivial ($P \neq \emptyset$ and there is a TM M with $\langle M\rangle \notin P$);*
2. *If $L(M_1) = L(M_2)$, $\langle M_1\rangle \in P$ iff $\langle M_2\rangle \in P$.*

Then P is undecidable.

Proof.

Let R be a TM deciding P. Let $T_\emptyset$ be a TM with $L(T_\emptyset) = \emptyset$. WLOG, assume $\langle T_\emptyset\rangle \notin P$. Moreover, pick a TM T with $\langle T\rangle \in P$. Consider
$S =$ "On input $\langle M, w\rangle$ where M is a TM and w a string:

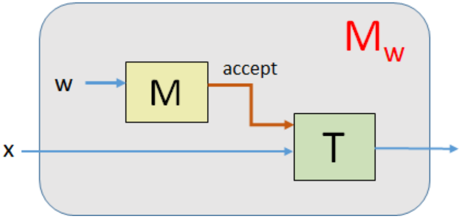
1. Use $\langle M, w\rangle$ to construct

   $M_w =$ "On input x:

   1. Run M on w. If M halts and rejects, reject.
   2. If M accepts w, run T on x."
2. Run R on $\langle M_w\rangle$.
3. If R accepts, accept; otherwise, reject." □

Rice's Theorem

- $\langle T_\emptyset \rangle \notin P$ and $\langle T \rangle \in P$, where $L(T_\emptyset) = \emptyset$.
- M accepts w will "trigger" the execution of T on input x.
- Hence,
  - M accepts $w \Rightarrow L(M_w) = L(T) \in P$
  - M does not accept $w \Rightarrow L(M_w) = L(T_\emptyset) \notin P$
- Does $REGULAR_{TM}$ fit into the above framework?
  How about E_{TM}?



Language Equivalence Problem for Turing Machines

- Consider

$$EQ_{\text{TM}} = \{\langle M_1, M_2 \rangle : M_1 \text{ and } M_2 \text{ are TM's with } L(M_1) = L(M_2)\}.$$

Theorem 5

EQ_{TM} is undecidable.

Proof.

We reduce the emptiness problem to the language equivalence problem this time. Let the TM R decide EQ_{TM} and TM M_1 with $L(M_1) = \emptyset$. Consider
S = "On input $\langle M \rangle$ where M is a TM:

1. Run R on $\langle M, M_1 \rangle$.
2. If R accepts, accept; otherwise, reject." □

Computation History

Definition 6

Let M be a TM and w an input string. An <u>accepting computation history</u> for M on w is a sequence of configurations $C_1, C_2, \ldots, C_l$ where

- C_1 is the start configuration of M on w;
- C_l is an accepting configuration of M; and
- C_i yields C_{i+1} in M for $1 \leq i < l$.

- A deterministic Turing machine has at most one computation history on any given input.
- A nondeterminsitic Turing machine may have several computation histories on an input.

$$\overbrace{q_0 w_1 w_2 \cdots w_n}^{C_1} \; \# \; \overbrace{a q_7 w_2 \cdots w_n}^{C_2} \; \# \; \overbrace{a c q_8 w_3 \cdots w_n}^{C_3} \; \# \; \cdots \; \# \; \overbrace{\cdots \; q_{\text{accept}} \; \cdots}^{C_{accept}}$$

Languages Associated with Computation Histories

Suppose $\alpha \vdash \beta$ is a single step of a TM M.

	left move	right move
α	$abcdqefgh$	$abcdqefgh$
β	$abcq'de'fgh$	$abcde'q'fgh$

Notice that in α and β, at most 3 positions may change.

- Can you check $abc\boxed{\text{dqe}}fgh\#(abc\boxed{\text{q'de'}}fgh)^R$ using a PDA?
  (Keep $(dqe, q'de')$ in finite state, process "abc" "fgh" using stack)
- How about $abc\boxed{\text{dqe}}fgh\#abc\boxed{\text{q'de'}}fgh$?

Consider accepting computation $\alpha_0 \vdash \alpha_1 \vdash \alpha_2 \vdash \alpha_3 \vdash \cdots \vdash \alpha_n$

- CS: $\alpha_0\#\alpha_1\#\alpha_2\#\alpha_3\#\cdots\#\alpha_n$
- CS_R: $\alpha_0\#\alpha_1^R\#\alpha_2\#\alpha_3^R\#\cdots\#\alpha_n$

CS_R is the intersection of two CFL L_{odd} and L_{even}, where

- $L_{odd} = \{\alpha_0\#\alpha_1^R\#\alpha_2\#\alpha_3^R\#\cdots\#\alpha_n \mid \alpha_i \vdash \alpha_{i+1}, i \text{ is odd}\}$
- $L_{even} = \{\alpha_0\#\alpha_1^R\#\alpha_2\#\alpha_3^R\#\cdots\#\alpha_n \mid \alpha_i \vdash \alpha_{i+1}, i \text{ is even}\}$

Linear Bounded Automaton

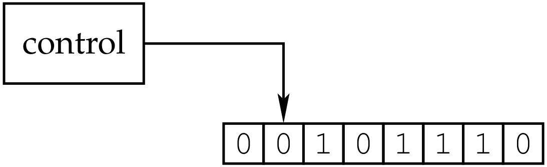


Figure: Schematic of Linear Bounded Automata

Definition 7

A linear bounded automaton is a nondeterministic Turing machine whose tape head is not allowed to move off the portion of its input. If an LBA tries to move its head off the input, the head stays.

- With a larger tape alphabet than its input alphabet, we may allow an LBA to use $c \times |w|$ tape cells on input w, where c is a constant.

Acceptance Problem for Linear Bounded Automata

- Consider

$$A_{\text{LBA}} = \{\langle M, w\rangle : M \text{ is an LBA and } M \text{ accepts } w\}.$$

Lemma 8

Let M be an LBA. There are $|Q| \times n \times |\Gamma|^n$ different configurations of M for a tape of length n.

- An LBA has $|Q| \times n \times |\Gamma|^n$ different configurations on an input of length n. If an LBA runs for longer, it must repeat some configuration and thus will never halt.
- Many langauges can be decided by LBA's.
  - For instance, $A_{\text{DFA}}, A_{\text{CFG}}, E_{\text{DFA}}$, and E_{CFG}.
- Every context-free langauges can be decided by LBA's.

Acceptance Problem for Linear Bounded Automata

Theorem 9

A_{LBA} is decidable.

Proof.

Consider

$L =$ "On input $\langle M, w\rangle$ where M is an LBA and w a string:

1. Simulate M on w for $|Q| \times n \times |\Gamma|^n$ steps or until it halts. ($|Q|$, n, and $|\Gamma|$ are obtained from $\langle M\rangle$ and w.)
2. If M does not halt in $|Q| \times n \times |\Gamma|^n$ steps, reject.
3. If M accepts w, accept; if M rejects w, reject." □

- The acceptance problem for LBA's is decidable. What about the emptiness problem for LBA's?

$$E_{\mathrm{LBA}} = \{\langle M\rangle : M \text{ is an LBA with } L(M) = \emptyset\}.$$

Emptiness Problem for Linear Bounded Automata

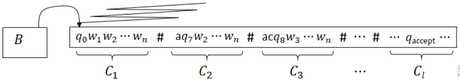
Theorem 10

E_{LBA} is undecidable.

Proof.

Reduce A_{TM} to E_{LBA}. Let R be a TM deciding E_{LBA}. Consider $S =$ "On input $\langle M, w\rangle$ where M is a TM and w a string:

1. Construct
   $B =$ "On input $\langle C_1, C_2, \ldots, C_l\rangle$, C_i's are configurations of M on w:
   1. If C_1 (resp., C_l) is not start (resp. accepting) config., reject.
   2. For each $1 \leq i < l$, if C_i does not yield C_{i+1}, reject.
   3. Otherwise, accept."
2. Run R on $\langle B\rangle$.
3. If R rejects $[L(B) \neq \emptyset]$, accept $[\langle M, w\rangle \in A_{TM}]$; otherwise, reject." □



B
$q_0 w_1 w_2 \cdots w_n$ # $a q_7 w_2 \cdots w_n$ # $ac q_8 w_3 \cdots w_n$ # $\cdots$ # $\cdots q_{\text{accept}} \cdots$
C_1
C_2
C_3
$\cdots$
C_l

Context Sensitive Grammars

- A context sensitive grammar (CSG) is a grammar where all productions are of the form

$$\alpha A\beta \to \alpha\gamma\beta, \ \ \alpha, \beta \in (N \cup \Sigma)^*, \gamma \in (N \cup \Sigma)^+,$$

- During derivation non-terminal A will be replaced by γ only when it is present in context of α and β.
- This definition shows clearly one aspect of this type of grammar; it is noncontracting, in the sense that the length of successive sentential forms can never decrease.
- The production $S \to \epsilon$ is also allowed if S is the start symbol and it does not appear on the right side of any production.
- A language L is said to be context-sensitive if there exists a context-sensitive grammar G, such that $L = L(G)$.
- An alternative definition of CSG:

$$u \to v, \ \ |u| \le |v|, u, v \in (N \cup \Sigma)^+,$$

An Example

$\{a^n b^n c^n \mid n \geq 1\}$ is a CSL.

Consider the following CSG G

$$\begin{array}{lcl} S & \rightarrow & \epsilon \mid abc \mid aTBc \\ T & \rightarrow & abC \mid aTBC \\ CB & \rightarrow & CX; \;\; CX \rightarrow BX; \;\; BX \rightarrow BC \\ bB & \rightarrow & bb \\ Cc & \rightarrow & cc \end{array}$$

E.g., To generate *aaabbbccc*, consider the following derivation:

$S \Rightarrow aTBc \Rightarrow aaTBCBc \Rightarrow aaabCBCBc \Rightarrow aaabBCCBc \Rightarrow aaabBCBCc \Rightarrow aaabBBCCc \Rightarrow aaabbBCCc \Rightarrow aaabbbCCc \Rightarrow aaabbbCcc \Rightarrow aaabbbccc$

More on CSLs

CSLs are closed under

1. Union
2. Intersection
3. Concatenation
4. Kleene closure
5. Complement
   Immerman-Szelepcsenyi theorem (1987).

(1)-(4) can be shown using LBA constructions. (5) follows from ”nondeterministic space being closed under complement,” whose proof is not trivial.

LBA $\equiv$ CSG

Theorem 11

A language is context-sensitive iff it can be accepted by a linear-bounded automaton.

Proof.

($\Rightarrow$) Recall in CSG, if $u \to v$, then $|u| \leq |v|$. Use LBA's tape to keep the current derivation sentence, which never exceeds $|w|$ (Why?)
($\Leftarrow$) Intuitive Idea.
Suppose LBA M has accepting comput. $q_0abcd \overset{*}{\Rightarrow} aeqfd \overset{*}{\Rightarrow} eq_{acc}fgh$. CSG G simulates the above in the following way

$$S \overset{*}{\Rightarrow} V_{(a,q_0a)}V_{(b,b)}V_{(c,c)}V_{(d,d)} \overset{*}{\Rightarrow} V_{(a,a)}V_{(b,e)}V_{(c,qf)}V_{(d,d)}$$

$$\overset{*}{\Rightarrow} V_{(a,e)}V_{(b,q_{acc}f)}V_{(c,g)}V_{(d,h)} \Rightarrow V_{(a,e)}bV_{(c,g)}V_{(d,h)} \Rightarrow V_{(a,e)}bcV_{(d,h)} \overset{*}{\Rightarrow} abcd$$

$\square$

Proof (Cont'd)

Proof.

To realize the above, need rules such as

- $V_{(x,qa)}V_{(y,b)} \to V_{(x,c)}V_{(y,pb)}$ if $\delta(q,a) = (p,c,R)$; $[...qab... \to ...cpb...]$
- $V_{(y,b)}V_{(x,qa)} \to V_{(x,pb)}V_{(y,c)}$ if $\delta(q,a) = (p,c,L)$; $[...bqa... \to ...pbc...]$
- $V_{(x,q_{acc}y)} \to x$; $xV_{(y,z)} \to xy$; $V_{(y,z)}x \to yx$

$\square$

- What rules are needed for $S \overset{*}{\Rightarrow} V_{(a,q_0a)}V_{(b,b)}V_{(c,c)}V_{(d,d)}$? Easy!
- Why does the above grammar construction fail for r.e. languages?
  - A bit tricky! You may generate $S \overset{*}{\Rightarrow} V_{(a,q_0a)}V_{(b,b)}V_{(c,c)}V_{(d,d)}V_{(\sqcup,\sqcup)}...V_{(\sqcup,\sqcup)}$ to "reserve" worktape locations.
  - But then you need rules to "contract" those $\sqcup$'s (rules such as $xV_{(\sqcup,y)} \to x$, which is not allowed in CSG.
  - If contraction rules allowed, we have unrestricted grammars (or called Type-0 grammars)

Universality of Context-Free Grammars

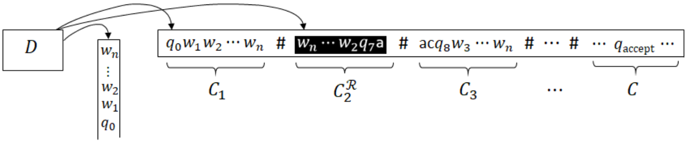
- Consider a problem related to the emptiness problem for CFL's

$$ALL_{\text{CFG}} = \{\langle G\rangle : G \text{ is a CFG and } L(G) = \Sigma^*\}.$$

- Let x be a string. Write x^R for the string x in reverse order.
- Let $C_1, C_2, \ldots, C_l$ be the accepting configuration of M on input w. Consider the following string in the next theorem:

$$\#\langle C_1\rangle\#\langle C_2\rangle^R\#\cdots\#\langle C_{2k-1}\rangle\#\langle C_{2k}\rangle^R\#\cdots\#\langle C_l\rangle\#$$

Consider the following PDA:


D
w_n
$\vdots$
w_2
w_1
q_0
$q_0 w_1 w_2 \cdots w_n$ # $w_n \cdots w_2 q_7 a$ # $a c q_8 w_3 \cdots w_n$ # $\cdots$ # $\cdots q_{\text{accept}} \cdots$
C_1
C_2^R
C_3
$\cdots$
C

(Fig. from M. Sipser's class notes)

Universality of Context-Free Grammars

Theorem 12

ALL_{CFG} is undecidable.

Proof.

We reduce A_{TM} to ALL_{CFG}. We construct a nondeterministic PDA D that accepts all strings if and only if $\underline{M \text{ does not accept } w}$. The input and stack alphabets of D contain symbols to encode M's configurations.

D = "On input $\#x_1\#x_2\#\cdots\#x_l\#$:

1. Do one of the following branches nondeterministically:
    - If $x_1 \neq \langle C_1 \rangle$ where C_1 is the start configuration of M on w, accept.
    - If $x_l \neq \langle C_l \rangle$ where C_l is a rejecting configuration of M, accept.
    - Choose odd i nondeterministically. If $x_i \neq \langle C \rangle$, $x_{i+1}^R \neq \langle C' \rangle$, or C does not yield C' (C, C' are configurations of M), then accept."
    - Choose even i nondeterministically. If $x_i^R \neq \langle C \rangle$, $x_{i+1} \neq \langle C' \rangle$, or C does not yield C' (C, C' are configurations of M), then accept."

M accepts w iff the accepting computation history of M on w is not in $L(D)$ iff $CFG(D) \notin ALL_{\text{CFG}}$. □

Post Correspondence Problem (PCP)

- A domino is a pair of strings: $\left[\frac{t}{b}\right]$
- A match is a sequence of dominos $\left[\frac{t_1}{b_1}\right]\left[\frac{t_2}{b_2}\right]\cdots\left[\frac{t_k}{b_k}\right]$ such that $t_1t_2\cdots t_k = b_1b_2\cdots b_k$.
- The Post correspondence problem is to test whether there is a match for a given set of dominos.

$$PCP = \{\langle P\rangle : P \text{ is an instance of the PCP with a match}\}$$

- Consider

$$P = \left\{ \left[\frac{\texttt{b}}{\texttt{ca}}\right], \left[\frac{\texttt{a}}{\texttt{ab}}\right], \left[\frac{\texttt{ca}}{\texttt{a}}\right], \left[\frac{\texttt{abc}}{\texttt{c}}\right] \right\}$$

- A match in P:

$$\left[\frac{\texttt{a}}{\texttt{ab}}\right]\left[\frac{\texttt{b}}{\texttt{ca}}\right]\left[\frac{\texttt{ca}}{\texttt{a}}\right]\left[\frac{\texttt{a}}{\texttt{ab}}\right]\left[\frac{\texttt{abc}}{\texttt{c}}\right]$$

The Modified Post Correspondence Problem

- The modified Post correspondence problem is a PCP where a match starts with the first domino. That is,

$$MPCP = \{\langle P \rangle : \; P \text{ is an instance of the PCP with a match starting with the first domino}\}$$

Theorem 13

PCP is undecidable.

Proof idea.

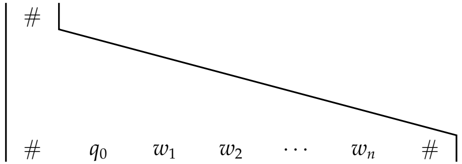
We reduce the acceptance problem for TM's to PCP. Given a TM M and a string w, we first construct an MPCP P' such that $\langle P' \rangle \in MPCP$ if and only if M accepts w. The MPCP P' encodes an accepting computation history of M on w. Finally, we reduce MPCP P' to PCP P.

The Post Correspondence Problem

Proof.

Let the TM R decide $MPCP$. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be the given TM and $w = w_1 w_2 \cdots w_n$ the input. The set P' of dominos has

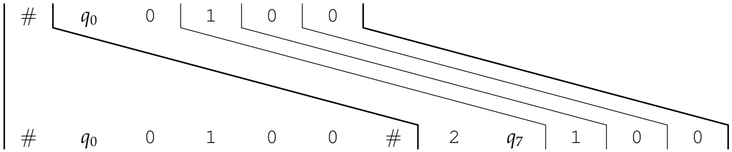
- $\left[\dfrac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right]$ as the first domino. Begin with the start configuration (bottom).



The Post Correspondence Problem

Proof (cont'd).

- $\left[\frac{qa}{br}\right]$ if $\delta(q,a) = (r,b,R)$ with $q \neq q_{\text{reject}}$. Reads a at state q (top); writes b and moves right (bottom).
- $\left[\frac{cqa}{rcb}\right]$ if $\delta(q,a) = (r,b,L)$ with $q \neq q_{\text{reject}}$. Reads a at state q (top); writes b and moves left (bottom).
- $\left[\frac{a}{a}\right]$ if $a \in \Gamma$. Keeps other symbols intact.

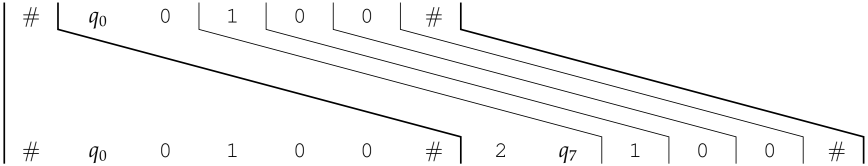


$$\delta(q_0, 0) = (q_7, 2, R)$$

The Post Correspondence Problem

Proof (cont'd).

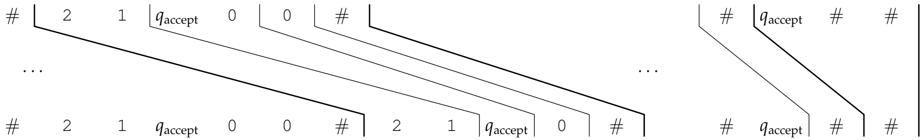
- $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\#}\right]$ Matches previous # (top) with a new # (bottom). Adds $\sqcup$ when M moves out of the right end.



The Post Correspondence Problem

Proof (cont'd).

- $\left[\frac{aq_{\text{accept}}}{q_{\text{accept}}}\right]$ and $\left[\frac{q_{\text{accept}}a}{q_{\text{accept}}}\right]$ if $a \in \Gamma$. Eats up tape symbols around q_{accept}.
- $\left[\frac{q_{\text{accept}}\#\#}{\#}\right]$. Completes the match.



# 2 1 q_{accept} 0 0 #
...
# q_{accept} # #
# 2 1 q_{accept} 0 0 # 2 1 q_{accept} 0 #
# q_{accept} # #

The Post Correspondence Problem

Proof (cont'd).

So far, we have reduced the acceptance problem of TM's to MPCP. To complete the proof, we need to reduce MPCP to PCP.
Let $u = u_1 u_2 \cdots u_n$. Define

$$\begin{array}{lcccccccccc} \star u & = & * & u_1 & * & u_2 & * & \cdots & * & u_n & \\ u\star & = & & u_1 & * & u_2 & * & \cdots & * & u_n & * \\ \star u\star & = & * & u_1 & * & u_2 & * & \cdots & * & u_n & * \end{array}$$

Given a MPCP P':

$$\left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$$

Construct a PCP P:

$$\left\{ \left[\frac{\star t_1}{\star b_1 \star}\right], \left[\frac{\star t_2}{b_2 \star}\right], \ldots, \left[\frac{\star t_k}{b_k \star}\right], \left[\frac{*\Diamond}{\Diamond}\right] \right\}$$

Any match in P must start with the domino $\left[\frac{\star t_1}{\star b_1 \star}\right]$. □

Some Applications of PCP

Theorem 14

Given two CFGs G_1 and G_2, "$L(G_1) \cap L(G_2) = \emptyset$?" is undecidable.

Proof.

For a PCP instance $\left[\frac{t_1}{b_1}\right]\left[\frac{t_2}{b_2}\right]\cdots\left[\frac{t_k}{b_k}\right]$, where $t_i, b_i \in \Sigma^*$, construct

- $G_1 : S_1 \to a_1S_1t_1 \mid a_2S_1t_2 ... \mid a_kS_1t_k \mid a_1t_1 \mid a_2t_2 ... \mid a_kt_k$
- $G_2 : S_2 \to a_1S_2b_1 \mid a_2S_2b_2 ... \mid a_kS_2b_k \mid a_1b_1 \mid a_2b_2 ... \mid a_kb_k$

where $a_i, 1 \leq i \leq k$, are new symbols not in Σ. Clearly $L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow$ PCP has a match. □

- Why do we need $a_1, ..., a_k$?
- Can you modify the above construction to yield the following?

Theorem 15

Given a CFG G, checking whether G is ambiguous is undecidable.

More Undecidability Results for CFLs

Theorem 16

Given two CFGs G_1 and G_2,and a regular language R, the following are undecisable:

1. $L(G_1) = L(G_2)$
2. $L(G_1) \subseteq L(G_2)$
3. $L(G_1) = R$
4. $R \subseteq L(G_1)$

Proof.

For (1) and (2), let $L(G_1) = \Sigma^*$. For (3) and (4), let $R = \Sigma^*$.
Undecidability following from the undecidablity of ALL_{CFG}. □

More on CFLs

Note, in contrast, that checking $L(G_1) \subseteq R$ is decidable.

- Let M be a FA accepting $\overline{R}$.

$$L(G_1) \subseteq R \Leftrightarrow L(G_1) \cap L(M) = \emptyset.$$

  The decidability result follows from $L(G_1) \cap L(M)$ being CFL, and the emptiness problem being decidable for CFLs.
- Why can we use a similar argument for $R \subseteq L(G_1)$?
  - Note that $\overline{L(G_1)}$ may not be a CFL. E.g., $\Sigma^* - \{a^n b^n c^n \mid n \geq 0\}$ is CF. Why?

Computable Functions

Definition 17

$f : \Sigma^* \to \Sigma^*$ is computable if some Turing machine M, on input w, halts with $f(w)$ on its tape.

- Usual arithmetic operations on integers are computable functions. For instance, the addition operation is a computable function mapping $\langle m, n \rangle$ to $\langle m + n \rangle$ where m, n are integers.

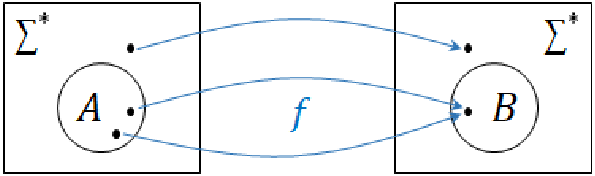
Mapping Reducibility

Definition 18

A language A is mapping reducible (or many-one reducible) to a language B (written $A \leq_m B$) if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that

$$w \in A \text{ if and only if } f(w) \in B, \text{ for every } w \in \Sigma^*.$$

f is called the reduction of A to B.



Properties of Reducibility

Theorem 19

If $A \leq_m B$ and B is decidable, A is decidable.

Proof.

Let the TM M decide B and f the reduction of A to B. Consider
$N =$ "On input w:

1. Construct $f(w)$.
2. Run M on $f(w)$.
3. If M accepts, accept; otherwise reject. □

Corollary 20

If $A \leq_m B$ and A is undecidable (i.e., not recursive), then B is undecidable.

Transitivity of Mapping Reductions

Lemma 21

If $A \leq_m B$ and $B \leq_m C$, $A \leq_m C$.

Proof.

Let f and g be the reductions of A to B and B to C respectively. $g \circ f$ is a reduction of A to C. □

Example 22

Give a mapping reduction from A_{TM} to PCP.

Proof.

The proof of Theorem 13 gives such a reduction. We first show $A_{\text{TM}} \leq_m MPCP$. Then we show $MPCP \leq_m PCP$. □

More Properties about Mapping Reductions

Theorem 23

If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof.

Similar to the proof of Theorem 19 except that M and N are TM's, not deciders. $\square$

Corollary 24

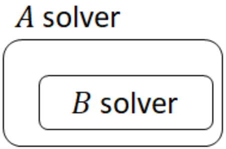
If $A \leq_m B$ and A is not Turing-recognizable (non-r.e.), then B is not Turing-recognizable.

More Properties about Mapping Reductions

- Observe that $A \leq_m B$ if and only if $\overline{A} \leq_m \overline{B}$.
  - The same reduction applies to $\overline{A}$ and $\overline{B}$ as well.
- Recall that $\overline{A_{\text{TM}}}$ is not Turing-recognizable.
- In order to show B is not Turing-recognizable, it suffices to show $A_{\text{TM}} \leq_m \overline{B}$ (or $\overline{A_{\text{TM}}} \leq_m B$).
  - $A_{\text{TM}} \leq_m \overline{B}$ implies $\overline{A_{\text{TM}}} \leq_m \overline{\overline{B}}$. That is, $\overline{A_{\text{TM}}} \leq_m B$.

Mapping vs. General Reducibility

- (General) Reducibility of A to B: Use B solver (as a subroutine) to solve A.
  - Conceptually simpler
  - Useful for proving undecidability



- A is reducible to $\overline{A}$.
- A may not be mapping reducible to $\overline{A}$.
- Note that $\overline{A_{TM}} \not\leq_m A_{TM}$. Why?

Reducibility - General Framework

To prove B is undecidable (i.e., not recursive):

- Show that undecidable A is reducible to B. (e.g., A is A_{TM})
- Approach:
  1. Assume TM R decides B.
  2. Construct TM S deciding A. Contradiction.

To prove B is Turing-unrecognizable (i.e., non-r.e.):

- Show that Turing-unrecognizable A is mapping reducible to B. (e.g., A is $\overline{A_{TM}}$)
- Approach:
  1. Give many-one reduction function f.
     - ★ Show f is computable.
     - ★ Show $w \in A \Leftrightarrow f(w) \in B$.

Examples

Example 25

Give a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

Proof.

We need to show a computable function f such that $\langle M, w \rangle \in A_{\mathrm{TM}}$ if and only if $\langle M', w' \rangle \in HALT_{\mathrm{TM}}$ whenever $\langle M', w' \rangle = f(\langle M, w \rangle)$.

Consider

$F =$ "On input $\langle M, w \rangle$:

1. Use $\langle M \rangle$ and w to construct
   $M' =$ "On input x:
   1. Run M on x.
   2. If M accepts, accept.
   3. If M rejects, loop."
2. Output $\langle M', w \rangle$."

□

Examples

Example 26

Give a mapping reduction of A_{TM} to $Regular_{TM} = \{\langle M \rangle \mid L(M) \text{ is regular}\}$.

- $f(\langle M, w \rangle) = \langle M' \rangle$ described below

M' takes input x:

- if x has form $0^n 1^n$, accept
- else simulate M on w and accept x if M accepts

$M' = \{0^n 1^n\}$ if $w \notin L(M)$
$= \Sigma^*$ if $w \in L(M)$

What would a formal proof of this look like?

- is f computable?
- YES maps to YES?
  $\langle M, w \rangle \in ACC_{TM} \Rightarrow f(M, w) \in REGULAR$
- NO maps to NO?
  $\langle M, w \rangle \notin ACC_{TM} \Rightarrow f(M, w) \notin REGULAR$

Examples

Example 27

Give a mapping reduction from E_{TM} to EQ_{TM}.

Proof.

The proof of Theorem 5 gives such a reduction. The reduction maps the input $\langle M \rangle$ to $\langle M, M_1 \rangle$ where M_1 is a TM with $L(M_1) = \emptyset$. □

E_{TM} is not Turing-recognizable

Theorem 28

E_{TM} is not Turing-recognizable.

Proof.

Show $\overline{A_{TM}} \leq_m E_{TM}$.

$F =$ "On input $\langle M, w\rangle$:

1. Use $\langle M\rangle$ and w to construct
   $M' =$ "On input x:
   1. if $x \neq w$, reject; else run M on w.
   2. If M accepts, accept.
2. Output $\langle M'\rangle$."

$\square$

- F is clearly computable. Furthermore, $\langle M, w\rangle \notin A_{TM} \Leftrightarrow L(M') = \emptyset$
- Is E_{TM} co-Turing-recognizable?
  (Nondeterministically generate a w on its tape, run M on w).

Equivalence Problem for TM's (revisited)

Theorem 29

EQ_{TM} is neither Turing-recognizable nor co-Turing-Recognizable.

Proof.

We first show $A_{\mathrm{TM}} \leq_m \overline{EQ_{\mathrm{TM}}}$. Consider
F = "On input $\langle M, w \rangle$ where M is a TM and w a string:

1. Construct
   - M_1 = "On input x: Reject."
   - M_2 = "On input x:
     1. Run M on w. If M accepts, accept."
2. Output $\langle M_1, M_2 \rangle$."

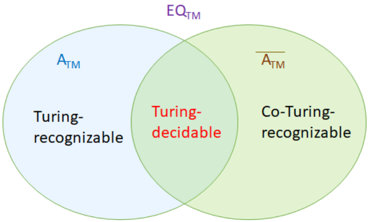
Equivalence Problem for TM's (revisited)

Proof (cont'd).

Next we show $A_{\text{TM}} \leq_m EQ_{\text{TM}}$. Consider

G = "On input $\langle M, w \rangle$ where M is a TM and w a string:

1. Construct
   - M_1 = "On input x: Accept."
   - M_2 = "On input x:
     1. Run M on w.
     2. If M accepts w, accept."
2. Output $\langle M_1, M_2 \rangle$." □



Strong Rice's Theorem

Theorem 30

Let P be a non-trivial property of TM descriptions, and M be a TM s.t. $L(M) = \Sigma^*$. If $\langle M \rangle \notin P$, then P is not Turing-recognizable.

Proof.

If we could show $A_{TM} \leq_m \overline{P}$, which in turn implies $\overline{A_{TM}} \leq_m P$. Pick a TM T with $\langle T \rangle \in P$. Consider
$S =$ "On input $\langle M, w \rangle$:

1. If $\langle M, w \rangle$ does not encode a TM and a string, then accept.
2. Use $\langle M, w \rangle$ to construct

   $M_w =$ "On input x:

   1. Run M on w and T on x in parallel,
   2. If either accepts, accept x
3. Run the "supposed" recognizer for P on $\langle M_w \rangle$. Output what the recognizer says.

- $L(M_w) = \Sigma^*$ ($\langle M_w \rangle \notin P$) iff $\langle M, w \rangle \in A_{TM}$
- $L(M_w) = L(T)$ ($\langle T \rangle \in P$) iff $\langle M, w \rangle \notin A_{TM}$

Applications of Strong Rice's Theorem

- $E_{TM} = \{\langle M \rangle \mid L(M) = \emptyset\}$ is not Turing-recognizable
  - Clearly E_{TM} is a non-trivial property
  - For M with $L(M) = \Sigma^*$, $\langle M \rangle \notin E_{TM}$.
- Can you think of other applications?

Arithmetic Hierarchy

- A language L is in Σ_0 (or Π_0) if it is recursive.
- A language L is in Σ_n, where $n \geq 1$, if there is a recursive relation $R(x, y_1, y_2, ..., y_n)$ such that

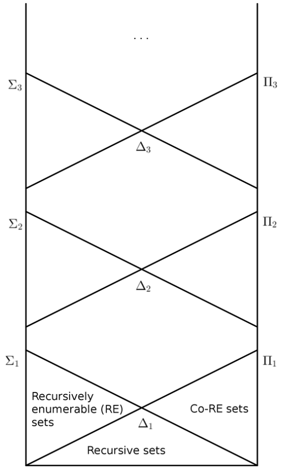
$$x \in L \quad \Leftrightarrow \quad \exists y_1 \forall y_2 \exists y_3 ... Q_n y_n R(x, y_1, y_2, ..., y_n).$$

  where Q_n is $\exists$ (resp., $\forall$) if n is odd (resp., even).
- A language L is in Π_n, where $n \geq 1$, if there is a recursive relation $R(x, y_1, y_2, ..., y_n)$ such that

$$x \in L \quad \Leftrightarrow \quad \forall y_1 \exists y_2 \forall y_3 ... Q_n y_n R(x, y_1, y_2, ..., y_n).$$

  where Q_n is $\exists$ (resp., $\forall$) if n is even (resp., odd).
- $\Delta_n = \Sigma_n \cap \Pi_n$.

Arithmetic Hierarchy



Some Examples in Arithmetic Hierarchy

In what follows, we let $R(M, w, n)$ be a predicate which is true if TM M accepts w in $\leq n$ steps. Clearly, R is a decidable predicate.

- $A_{TM} = \{\langle M, w\rangle \mid \exists n, R(M, w, n)\}$.
  - $A_{TM} \in \Sigma_1$
- $E_{TM} = \{\langle M\rangle \mid \forall_{\langle w,n\rangle}, \neg R(M, w, n)\}$.
  - $E_{TM} \in \Pi_1$
- $ALL_{TM} = \{\langle M\rangle \mid L(M) = \Sigma^*\} = \{\langle M\rangle \mid \forall_w \exists_n, R(M, w, n)\}$.
  - $ALL_{TM} \in \Pi_2$
- $FIN_{TM} = \{\langle M\rangle \mid L(M) \text{ is finite}\} =$ $\{\langle M\rangle \mid \exists_m \forall_{\langle w,n\rangle} (|w| \leq m) \vee \neg R(M, w, n)\}$.
  - $FIN_{TM} \in \Sigma_2$
- $COFIN_{TM} = \{\langle M\rangle \mid \overline{L(M)} \text{ is finite}\} =$ $\{\langle M\rangle \mid \exists_m \forall_w \exists_n (|w| \leq m) \vee R(M, w, n)\}$.
  - $COFIN_{TM} \in \Sigma_3$

Some Examples in Arithmetic Hierarchy

How about the following languages:

- $EQ_{TM} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$
  Note: $\overline{EQ_{TM}} = \{\langle M_1, M_2\rangle \mid$
  $\exists_{\langle w,n\rangle}\forall_m (R(M_1, w, n) \wedge \neg R(M_2, w, m)) \vee (R(M_2, w, n) \wedge \neg R(M_1, w, m))\}$
- $INF_{TM} = \{\langle M\rangle \mid L(M)$ is infinite$\}$
  Note: $INF_{TM} = \{\langle M\rangle \mid \forall_m \exists_{\langle w,n\rangle} (|w| \geq m) \wedge R(M, w, n)\}$
- $REG_{TM} = \{\langle M\rangle \mid L(M)$ is regular$\}$
  Note: $REG_{TM} = \{\langle M\rangle \mid \exists_{\langle M'\rangle}\forall_w \exists_n (R'(M', w) \Leftrightarrow R(M, w, n))\}$, where M' is a FA, and $R'(M', w)$ is true if FA M' accepts w.
- How about
  $CFL_{TM} = \{\langle M\rangle \mid L(M)$ is coontxt-free$\}$?
  $REC_{TM} = \{\langle M\rangle \mid L(M)$ is recursive$\}$?

Gödel Incompleteness Theorem

- A proof system is a collection of axioms.
- A proof system is consistent if the axioms don't contradict each other.
- A proof system is complete if every true statement can be derived from the set of axioms.

Theorem 31

For a proof system within which a certain amount of elementary arithmetic can be carried out,

1. *if the system is consistent, then it is incomplete.*
2. *the system cannot prove "its own consistency".*

Gödel Incompleteness Theorem

Proof.

To prove (1), consider

```
Function S(M, program)
{    for each proof P
         {
         if (P proves S(S) loops) return;
         if (P proves S(S) halts) loop;
         }
}
```

Question: Does $S(S)$ loop or halt?
Note: For the system to be complete, either $S(S)$ loops or halts must hold.

- if there is a P proving $S(S)$ loops, $S(S)$ halts.
- if there is a P proving $S(S)$ halts, $S(S)$ loops.

Both yield inconsistency.
How to prove (2)? □