Models of Computation

8: Decision problems, undecidability

Encoding objects into strings

- If O is some object (e.g., automaton, TM, polynomial, graph, etc.), we write $<O>$ to be an encoding of O into a string.
- If $O_1, O_2,\ldots,O_k$ is a list of objects then we write $<O_1, O_2,\ldots,O_k>$ to be an encoding of them together into a single string.

- Notation for writing Turing machines
- We will use English descriptions of algorithms when we describe TMs, knowing that we could (in principle) convert those descriptions into states, transition function, etc.
- M = “On input w:
- [English description of the algorithm]”

Example

- TM *M* recognizing $L = \{a^k b^k c^k : k \geq 0\}$.
- *M* = "On input *w*
  1) Check if, $w \in a^*b^*c^*$, reject if not.
  2) Count the number of *a*'s, *b*'s, and *c*'s in *w*.
  3) Accept if all counts are equal; reject if not."

- High-level description is ok.
- We do not need to manage tapes, states, etc…

Encoding of TMs

- Assumed that $\Sigma = \{0,1\}$.
- The **code** of a TM M (denoted by $<M>$) is the following:
- Let $M = (Q, \{0,1\}, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where
    - $Q = \{p_1,\ldots,p_k\}$, $\Gamma = \{X_1,\ldots,X_m\}$, $D_1 = R$, $D_2 = S$, $D_3 = L$,
    - $k \geq 3$, $p_1 = q_0$, $p_{k-1} = q_{accept}$, $p_k = q_{reject}$,
    - $m \geq 3$, $X_1 = 0$, $X_2 = 1$, $X_3 =$ _.
    - The code of a transition $\delta(p_i, X_j) = (p_r, X_s, D_t)$ is $0^i10^j10^r10^s10^t$.
    - $<M>$ is list of transition codes separated by 11.
- Note: $<M>$ starts and ends with 0, does not contain the substring 111.
- $<M,w> := <M>111w$

Acceptance Problem for DFAs

- Let $A_{DFA} = \{<B,w> \mid B \text{ is a DFA and } B \text{ accepts } w\}$.

Theorem: A_{DFA} is decidable.

Proof: Give TM $M_{A\text{-}DFA}$ that decides A_{DFA}.

- $M_{A\text{-}DFA}$ = "On input s
  - check that s has the form $<B,w>$ where B is a DFA and w is a string; reject if not.
  - Simulate the computation of B on w.
  - If ends in an accept state then accept.
    If not then reject."

Acceptance Problem for NFAs

- Let A_{NFA} = {<*B*,*w*> | *B* is a NFA and *B* accepts *w*}.

Theorem: A_{DFA} is decidable.

Proof: Give TM $M_{\text{A-NFA}}$ that decides A_{NFA}.

- $M_{\text{A-NFA}}$ = "On input <*B*,*w*>
  - Convert NFA *B* to equivalent DFA *B'*.
  - Run TM $M_{\text{A-DFA}}$ on input <*B'*,*w*>.   [$M_{\text{A-DFA}}$ decides A_{DFA}]
  - Accept if $M_{\text{A-DFA}}$ accepts.
  - Reject if not."

- New element: Use conversion construction and previously constructed TM as a subroutine.

Emptiness Problem for DFAs

- Let $E_{\text{DFA}} = \{<B> \mid B$ is a DFA and $L(B) = \varnothing\}$.

Theorem: E_{DFA} is decidable.

Proof: Give TM $M_{\text{E-DFA}}$ that decides E_{DFA}.

- $M_{\text{E-DFA}}$ = "On input $<B>$
  [Idea: Check for a path from start to accept.]
  - Mark start state.
  - Repeat until no new state is marked:
    - Mark every state that has an incoming arrow from a previously marked state.
  - Accept if no accept state is marked.
  - Reject if some accept state is marked."

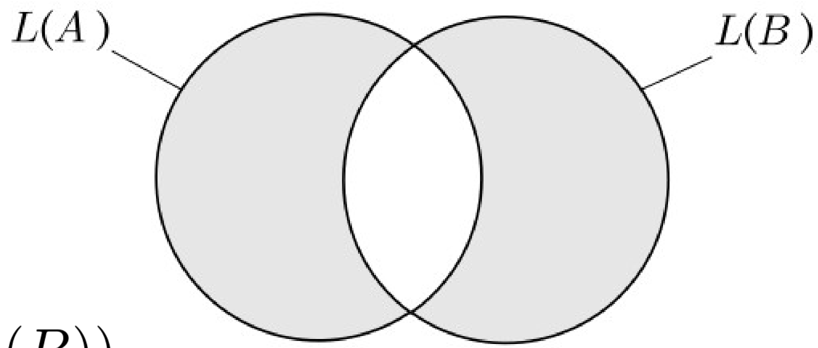
Equivalence Problem for DFAs

- Let $EQ_{\text{DFA}} = \{\langle A,B\rangle \mid A,B \text{ are DFAs and } L(A) = L(B)\}$.

Theorem: EQ_{DFA} is decidable.

Proof: Give TM $M_{\text{EQ-DFA}}$ that decides EQ_{DFA}.

- $M_{\text{EQ-DFA}}$ = “On input $\langle A,B\rangle$
  [Idea: Make DFA C that accepts w where A and B disagree.]
    - Construct DFA C where
      $$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$
    - Run $M_{\text{E-DFA}}$ on C.
    - Accept if $M_{\text{E-DFA}}$ accepts.
    - Reject if $M_{\text{E-DFA}}$ rejects.”



Acceptance Problem for CFGs

- Let $A_{\text{CFG}} = \{<G,w> \mid G \text{ is a CFG and } G \text{ generates } w\}$.

Theorem: A_{CFG} is decidable.

Proof: Give TM $M_{\text{A-CFG}}$ that decides A_{CFG}.

- $M_{\text{A-CFG}}$ = "On input $<G,w>$
  - Convert into CNF.
  - Try all derivations of length max(1, 2|w| - 1).
  - Accept if any generate *w*.
  - Reject if not.

Corollary: Every CFL is decidable.

Proof: Let *L* be a CFL, generated by CFG *G*.

- Construct TM M_G = "on input *w*
  - Run $M_{\text{A-CFG}}$ on $<G,w>$.
  - Accept if $M_{\text{A-CFG}}$ accepts
  - Reject if $M_{\text{A-CFG}}$ rejects."

- Chomsky Normal Form (CNF) only allows rules:
  - $A \rightarrow BC$
  - $B \rightarrow b$
  - $S \rightarrow \varepsilon$
- Lemma 1: Every CFG can be converted CFG into CNF.
- Lemma 2: If *G* is in CNF and $w \in L(G)$, then every derivation of *w* has max(1, 2|*w*| - 1) steps.

Emptiness Problem for CFGs

- Let $E_{\text{CFG}} = \{<G> \mid G \text{ is a CFG and } L(G) = \emptyset\}$.

Theorem: E_{CFG} is decidable.

Proof: Give TM $M_{\text{E-CFG}}$ that decides E_{CFG}.

- $M_{\text{E-CFG}}$ = “On input $<G>$
  [Idea: work backwards from terminals]
    - Mark all occurrences of terminals in G.
    - Repeat until no new variables get marked
        - Mark all occurrences of variable A if $A \rightarrow B_1B_2 \ldots B_k$ is a rule and all B_i were already marked.
    - Reject if the start variable is marked.
    - Accept if not.”

Equivalence Problem for CFGs

- Let $EQ_{\text{CFG}} = \{<G,H> \mid A,B$ are CFGs and $L(G) = L(H)\}$.

Theorem: EQ_{DFA} is <u>not</u> decidable.

Remark: CF languages is not closed under complementation or intersection.

Proof: s. Sipser, 5.1. exercise

Existence of non-Turing-recognizable languages

- For all $i \geq 1$, let w_i be the i-th element of the set $\{0,1\}^*$ ordered by length and lexicograpically, i.e. $\{\varepsilon,0,1,00,01,10,11,000,001,\ldots\}$.
- Let M_i denote the TM encoded by w_i (if w_i does not encode a TM, then M_i is an arbitrary TM that does not accept anything)

Theorem: There is a non-Turing-recognizable language.

Proof:

- Two different languages cannot be recognized by the same TM.
- The number of TMs is countably infinite (the encoding of TMs is an injection into $\{0,1\}^*$, whose cardinality is countably infinite).
- The set of languages over $\{0,1\}$ (i.e. $\{L \subseteq \{0,1\}^*\}$) is uncountable (cardinality of continuum). $\square$

A non-Turing-recognizable language

Theorem: Let $L_d = \{w_i : w_i \notin L(M_i)\}$. L_d is not Turing-recognizable, i.e. $L_d \notin RE$.

Proof: Georg Cantor's **diagonalization** method.

- Consider the bit table T, for which $T(i,j) = 1 \Leftrightarrow w_j \in L(M_i)$ $(i,j \geq 1)$.
- Let z be an infinitely long bit string in the diagonal of T and $\bar{z}$ be the bitwise complement of z.
- For all $i \geq 1$, the i-th row of T is the characteristic vector of language $L(M_i)$.
- $\bar{z}$ is the characteristic vector of L_d.
- If L_d could be recognized by a TM D, the characteristic vector of D would be a row in T.
- $\bar{z}$ differs from every row of T, so L_d differs from all languages in RE. $\square$

$$
\begin{array}{c|cccccc}
T & \langle M_1\rangle & \langle M_2\rangle & \langle M_3\rangle & \dots & \langle D\rangle & \dots \\
\hline
M_1 & \underline{1} & 0 & 1 & & 1 & \\
M_2 & 0 & \underline{1} & 1 & \dots & 0 & \dots \\
M_3 & 1 & 0 & \underline{0} & & 1 & \\
\vdots & & \vdots & & \ddots & & \\
D & 1 & 0 & 1 & & \underline{?} & \\
\vdots & & \vdots & & & & \ddots
\end{array}
$$

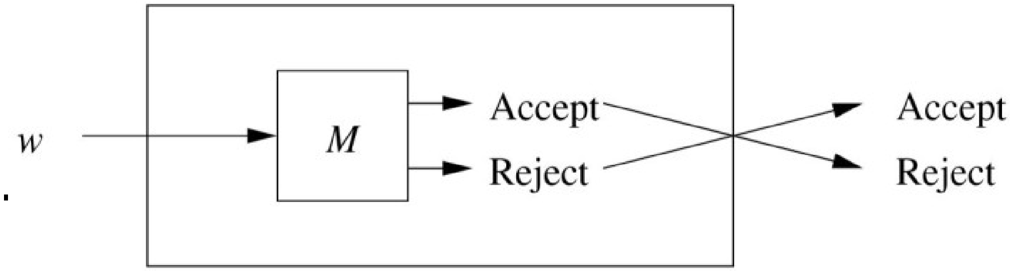
$\bar{z} = 001\dots$

Recursive languages *R*

- A language L is **recursive** if $L = L(M)$ for a **decider** TM M.

Theorem: If a L is recursive, then $\overline{L}$ is also reccursive.

Proof:

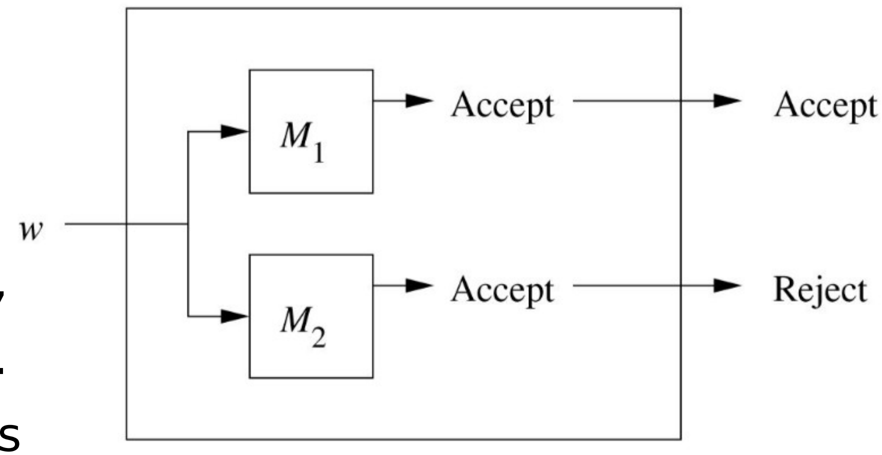


- Let $L = L(M)$ for some TM M which halts for every input. We construct a TM M' with $\overline{L} = L(M')$.
  - The accepting states of M will be the rejecting states of M' (halts without acceptance)
  - M' has a new accepting state r (there is no transition from r).
  - Consider all pairs (q,a) of non accepting states q of M and input symbol a, for which there is no transition in M (M halts without acceptance). For all such pairs (q,a) add a transition to state r.
- Since M is halts with every input word, M' also halts with every input word.
- M' accepts exatly the words that are not accepted by M. M' accepts $\overline{L}$. □

Complements of recursively enumerable (*RE*) languages

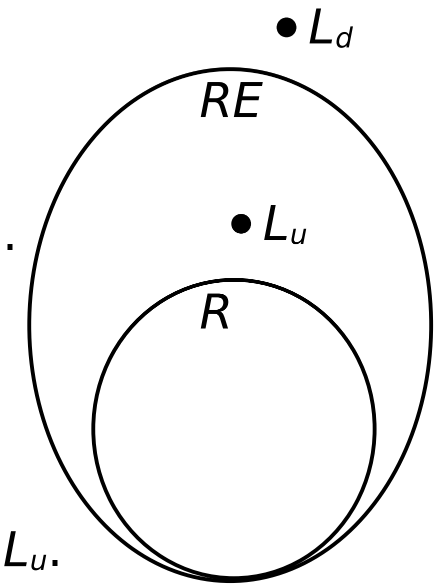
Theorem: If $L \in RE$ and $\overline{L} \in RE$, then $L \in R$ (and $\overline{L} \in RE$).

Proof:



- Let $L = L(M_1)$ and $\overline{L} = L(M_2)$. M_1 and M_2 are simulated in parallel with a TM *M.*
- Let *M* be a 2-tape TM.
  - Tape-1 of *M* simulates the tape of M_1,
  - Tape-2 of *M* simulates the tape of M_2.
  - The states of *M* correspod to the pairs $Q_1 \times Q_2$ (pairs of states of M_1 and M_2).
- If the input *w* of *M* is in *L*, then M_1 accepts it and halts. Then *M* accepts *w* and halts.
- If the input *w* of *M* is in $\overline{L}$, then M_2 accepts it and halts. Then *M* rejects *w* and halts. So *M* halts with all inputs and $L(M) = L$. □

R and RE



- Universal language: $L_u = \{<M,w> \mid M \text{ is TM and } w \in L(M)\}$.

Theorem: $L_u \in RE \setminus R$.

Proof:

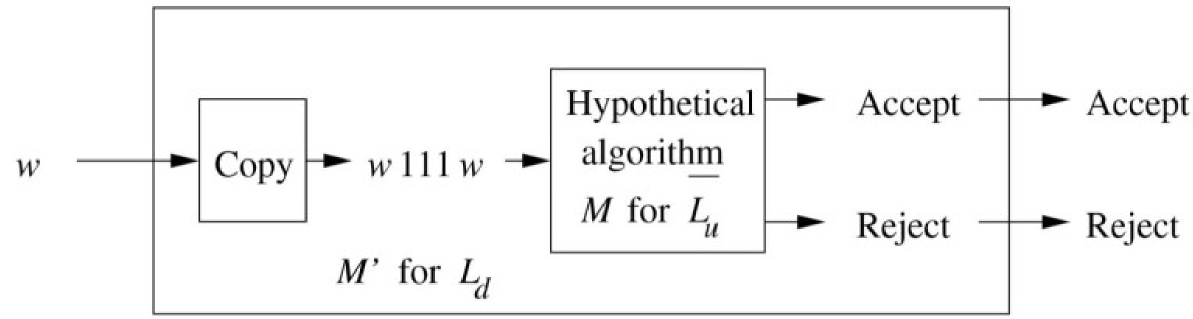
- L_u is recursively enumerable (Turing-recognizable)
- We construct a TM U, called the universal TM, to recognize L_u.
- Let U be a multitape TM s.t.
  - 1st tape holds the input with the encodings of M and w.
    We use the encoding of TMs and binary strings from this lecture.
  - 2nd tape is used to simulate M's input tape.
    We initialize the 2nd tape with w.
    We move the head on the 2nd tape to the first simulated cell.
  - 3rd tape is used to store M's state.
    We initialize the 3rd tape with the start state of M.
  - 4th tape is used as a work tape.

R and *RE*

Proof (cont.):

- Simulating a transition of *M:*
  - *U* searches tape 1 for a transition for the current state of *M* (stored on tape 3) and the current tape symbol of *M* (stored on tape 2).
  - Then *U* stores the new state on tape 3, *U* changes the tape symbol on tape 2, *U* moves *M*'s tape head left or right on tape 2 as specified by the transition.
  - If *M* enters its final state signaling that *M* accepts *w*, then *U* accepts $<M,w>$ and halts.
- Thus, $L(U) = L_u$.
- $\Rightarrow L_u \in RE$

R and *RE*



Proof (cont.):

- L_u is not recursive:
- Suppose for contradiction, L_u were recursive. Then there would exist a TM *M* that accepts $\overline{L_u}$ the complement of L_u.
- Then we can transform *M* into a TM *M*' that accepts L_d as follows:
    - *M*' transforms its input string *w* into a pair <*w*,*w*>.
    - *M*' simulates *M* on <*w*,*w*> assuming the first *w* is an encoding of a TM M_i and the second *w* is an encoding of a binary string w_i. Since *M* accepts the complement of L_u, *M* will accept <*w*,*w*> if and only if M_i does not accept w_i.
- Thus, *M*' accepts *w* if and only if *w* is in L_d. But we have previously shown that there does not exist a TM that recognizes L_d. Consequently, *M* does not exist.
- $\Rightarrow L_u \notin R$. □

Halting Problem

- In Alan Turing's original formulation of Turing machines acceptance was just by halting not necessarily by halting in a final state.
- We define $H(M)$ for a TM M to be the set of input strings w on which M halts in either a final or a nonfinal state.
- The **halting problem** is to he set of pairs $HALT = \{<M,w> \mid w \text{ is in } H(M)\}$.
- **Theorem**: $HALT \in RE \setminus R$.
  Proof: Similar to the proof of $L_u \in RE \setminus R$.
- A similar argument can be used to show that many practical problems associated with software verification are undecidable. For example, the problem of determining whether a program will ever go into an infinite loop is undecidable.

Reducibility – Concept

- If we have two languages (or problems) A and B, then A is reducible to B means that we can use B to solve A.
- If A is reducible to B then solving B gives a solution to A.
  - B is easy → A is easy.
  - A is hard → B is hard.
    this is the form we will use

Reducibility

- If we know that some problem is undecidable, we can use that to show other problems are undecidable.
- $HALT = \{<M,w> \mid M$ halts on input $w\}$.

Theorem: *HALT* is undecidable.

Proof: Showing that L_u is reducible to *HALT*.

- Assume that *HALT* is decidable and show that L_u is decidable.
- Let *R* be a TM deciding *HALT*.
- Construct TM *S* deciding L_u.
- S = “On input $<M,w>$

  1. Use *R* to test if *M* on *w* halts. If not, reject.
  2. Simulate *M* on *w* until it halts (as guaranteed by *R*).
  3. If *M* has accepted then accept.
     If *M* has rejected then reject.

- TM *S* decides L_u is a contradiction. Therefore, *HALT* is undecidable. □

Recursive (Turing-decidable) languages
R and $\mathcal{L}_1$ languages

$\triangleright$	w_1	w_2	$\dots$					w_n	$\triangleleft$

- A **linear bounded automaton (LBA)** is a nondeterministic TM, whose
  - input alphabet Σ contains two special symbols $\triangleright$ (left endmarker) and $\triangleleft$ (right endmarker).
  - The inputs are in the form $\triangleright(\Sigma \setminus \{\triangleright,\triangleleft\})^*\triangleleft$,
  - $\triangleright$ and $\triangleleft$ cannot be overwritten
  - The head cannot stand to the left of $\triangleright$ or to the right of $\triangleleft$.
  - The starting position of the head is the right neighbor of the cell containing $\triangleright$.
- An LBA is an NTM that has a limited working area.
- Named after an equivalent model in which the available storage is bounded by a constant multiple of the length of the input.

R and $\mathcal{L}_1$

Theorem:

- (1) For every type-1 grammar G, a LBA A can be given, s.t. $L(A) = L(G)$.
- (2) For every LBA A, a type-1 grammar G can be specified, s.t. $L(G) = L(A)$.

Proof:

- (1) In the previous lecture, we saw that all type-0 grammar G an NTM can be constructed recognizing $L(G)$.
- The construction simulates a derivation in G non-deterministically on tape 3. At the end of the iterations the NTM checks if the sentence on tape 3 is equal to the the input word w on tape 1.
- If G is a type-1 grammar, the length of strings during the derivation are non-decreasing. Therefore, the length of the string on tape 2 never exceeds $|w|$, so this NTM is an LBA.

R and $\mathcal{L}_1$

Proof (cont.):

- (2) For every LBA A, a type-1 grammar G can be specified, s.t. $L(G) = L(A)$.
- We sightly modify the construction of the last lecture.
- Let $\Gamma' := \Gamma \setminus \{\triangleright,\triangleleft\}$ and $G = ((\Gamma \setminus \Sigma) \cup Q \times \Gamma' \cup \{S,A\}, \Sigma, P, S)$.
  1) $S \to \triangleright A(q_{accept},a)A\triangleleft \mid \triangleright A(q_{accept},a)\triangleleft \mid \triangleright(q_{accept},a)A\triangleleft \mid \triangleright(q_{accept},a)\triangleleft$ $\quad (\forall\, a\in\Gamma')$
  2) $A \to aA \mid a$ $\quad (\forall\, a\in\Gamma')$
  3) $b(q',c) \to (q,a)c$ if $(q',b,R) \in \delta(q,a)$ $\quad (\forall\, c\in\Gamma')$
  4) $(q',b) \to (q,a)$ if $(q',b,S) \in \delta(q,a)$
  5) $(q',c)b \to c(q,a)$ if $(q',b,L) \in \delta(q,a)$ $\quad (\forall\, c\in\Gamma')$
  6) $\triangleright(q_0,a) \to \triangleright a$ $\quad (\forall\, a\in\Gamma')$
- 1-2. we generate an arbitrary accepting configuration. Since A is an LBA, for accepting a word u, it is enough to generate a configuration of length of at most $|u|$. After this the length of sentence is fixed.
- 3-5. configuration transitions are simulated in reverse order in the grammar.

R and $\mathcal{L}_1$

Proof (cont.):

1) $S \to \triangleright A(q_{accept},a)A\triangleleft \mid \triangleright A(q_{accept},a)\triangleleft \mid \triangleright(q_{accept},a)A\triangleleft \mid \triangleright(q_{accept},a)\triangleleft$ $(\ \forall\ a\in\Gamma'\ )$
2) $A \to aA \mid a$ $(\ \forall\ a\in\Gamma')$
3) $b(q',c) \to (q,a)c$ if $(q',b,R) \in \delta(q,a)$ $(\ \forall\ c\in\Gamma'\ )$
4) $(q',b) \to (q,a)$ if $(q',b,S) \in \delta(q,a)$
5) $(q',c)b \to c(q,a)$ if $(q',b,L) \in \delta(q,a)$ $(\ \forall\ c\in\Gamma'\ )$
6) $\triangleright(q_0,a) \to \triangleright a$ $(\ \forall\ a\in\Gamma'\ )$

- 6. Since the grammar does not decrease the length, technically we need symbols from $Q \times \Gamma'$. Until the last step, the sentence contains exactly one of that symbols.
- For all $a \in \Sigma \setminus \{\triangleright,\triangleleft\}$, $w \in (\Sigma \setminus \{\triangleright,\triangleleft\})^*$ or $a = \_$, $w = \varepsilon$, it can be shown by induction on the length of the derivation that
  - for $x \in \Gamma'$, $\alpha,\beta \in (\Gamma')^*$: $\triangleright q_0 aw\triangleleft \vdash^* \triangleright\alpha q_{accept} x\beta\triangleleft$ if and only if $S \Rightarrow^* \triangleright\alpha(q_{accept},x)\beta\triangleleft \Rightarrow^* \triangleright(q_0,a)w\triangleleft \Rightarrow \triangleright aw\triangleleft$. □

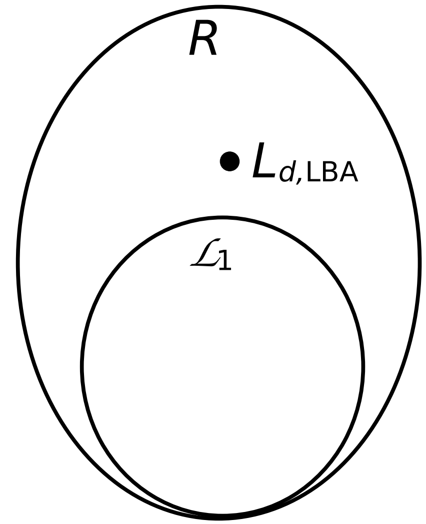
R and $\mathcal{L}_1$

Theorem: If *A* is LBA, then *L*(*A*) is decidable.

Proof:

- Let *w* be an input word, $|w|=n$. Due to the linear bound, the number of possible configurations of *A* for an input *w* is at most $m(w) = |Q| \cdot n \cdot |\Gamma|^n$.
- Every computation longer than *m*(*w*) leads to an infinite loop.
- Let *M'* be the TM, s.t. on input <*A,w*>, where *A* is an LBA and *w* a string
  1) Run *A* on *w* for $\leq m(w)+1$ transitions
  2) If *A* accepts/rejects before this point, accept/reject as *A*.
  3) Otherwise, reject.
- Obviously, *L*(*M'*) = *L*(*A*) and *M'* decides *L*(*A*). □

R and $\mathcal{L}_1$



Theorem: $\mathcal{L}_1 \subset R$.

Proof:

- Based on the previous 2 theorems, $\mathcal{L}_1 \subseteq R$.
- Let $L_{d,\text{LBA}} = \{<A> : A$ is a LBA and $<A> \notin L(A)\}$.
- $L_{d,\text{LBA}}$ can be decided as follows:
    - For LBA A, let S be a TM which goes in state
        - q_{accept} if $<A> \notin L(A)$ and
        - q_{reject} if $<A> \in L(A)$.

  Since $L(A)$ decidable, S always halts. $\Rightarrow L_{d,\text{LBA}} \in R$.
- $L_{d,\text{LBA}}$ is not recognizable with LBA ($\Rightarrow L_{d,\text{LBA}} \notin \mathcal{L}_1$)
    - By Cantor's diagonalization method.
    - For contradiction, assume that $L_{d,\text{LBA}}$ is recognized by an LBA S.
        - if $<S> \in L_{d,\text{LBA}}$, then S recognizes $<S>$, so $<S> \notin L_{d,\text{LBA}}$, contradiction,
        - if $<S> \notin L_{d,\text{LBA}}$, then S does not recognizes $<S>$, so $<S> \in L_{d,\text{LBA}}$, contradiction. $\square$

References

- Michael Sipser: Introduction to the Theory of Computation. 3rd edition, 2012.