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 $<M>$) 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.
- $<M,w> := <M>111w$

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). □

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. $L_d \notin RE$.

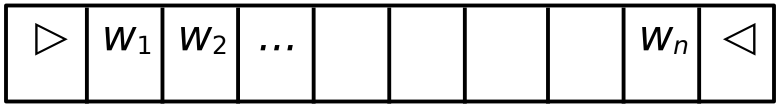
Proof: Georg Cantor's **diagonalization** method.

- 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* . □

T	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\dots$	$\langle D\rangle$	$\dots$
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		$\underline{?}$	
$\vdots$		$\vdots$				$\ddots$

$\bar{z} = 001\dots$

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



- A **linear bounded automaton (LBA)** is a nondeterministic TM, whose
  - input alphabet Σ contains two special symbols ▷ (left endmarker) and ◁ (right endmarker).
  - The inputs are in the form $\triangleright(\Sigma \setminus \{\triangleright,\triangleleft\})^*\triangleleft$,
  - ▷ and ◁ cannot be overwritten
  - The head cannot stand to the left of ▷ or to the right of ◁.
  - The starting position of the head is the right neighbor of the cell containing ▷.
- 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) $S \to \triangleright A(q_{accept},a)A\triangleleft \mid \triangleright A(q_{accept},a)\triangleleft \mid \triangleright(q_{accept},a)A\triangleleft \mid \triangleright(q_{accept},a)\triangleleft$ $\quad(\forall\, a \in \Gamma')$
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) $\triangleright(q_0,a) \to \triangleright a$ $\quad(\forall\, a \in \Gamma')$

- 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 $x \in \Gamma'$, $\alpha,\beta \in (\Gamma')^*$: $\triangleright q_0 aw\triangleleft$ yields $\triangleright\alpha q_{accept} x\beta\triangleleft$ 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 $\mathcal{L}_1$

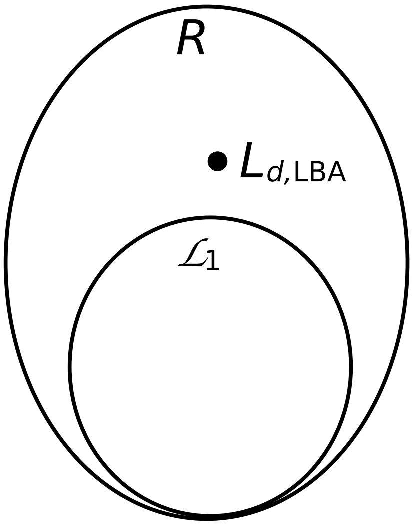
Theorem: If *A* is LBA, then *L*(*A*) is decidable.

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.
- *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 $\mathcal{L}_1$



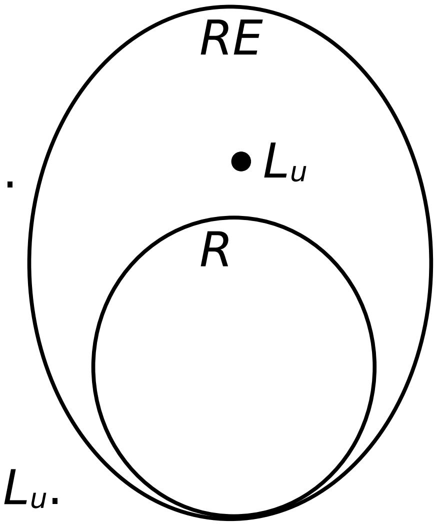
Theorem: $\mathcal{L}_1 \subset R$.

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 ($\Rightarrow L_{d,\text{LBA}} \notin \mathcal{L}_1$)
    - using 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. □

R and *RE* (recursively enumerable languages



- 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.):

- To simulate a transition of *M*,
  U searches tape 1 for a transition on 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 L_u were recursive.
  Then there would exist a TM M that accepts the complement of L_u.
- But 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 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 $\{<M,w> \mid w \text{ is in } H(M)\}$.
- We can show the halting problem is recursively enumerable but not recursive.
- 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.

References

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