Turing machines

Informally:

- A one-way infinite tape of finite memory cells
- A read/write tape head
- A finite program, with instructions:
  - read/write one tape cell
  - move tape head one cell left or right
  - branch, according to contents of current tape cell
  - halt
- Program’s input is placed on the tape initially
- Program’s output is the contents of the tape upon termination
Formal definition of a Turing Machine (TM):

- An input/output alphabet $\Sigma$ (usually $\{0, 1\}$)
- A tape alphabet $\Gamma$, where $\Sigma \subseteq \Gamma$, and $\_ \in \Gamma \setminus \Sigma$ is a special “blank symbol”
- A finite set of states $Q$, with two distinguished states $q_0$ (the start state) and $q_{\text{halt}}$ (the halt state)
- A transition function
  $$\delta : (Q \setminus q_{\text{halt}}) \times \Gamma \to \Gamma \times \{\text{Left, Right}\} \times Q$$
- Semantics: if $\delta(q, a) = (b, d, r)$, then when the machine is in state $q$ and the tape head is reading $a$, the machine writes $b$ to the tape, moves the tape one cell in direction $d$, and goes to state $r$
Details of execution:

- The tape consists of cells $c_1, c_2, c_3, \ldots$ (think of $c_1$ as being on the left end of the tape)
- If the input is $x = x_1 \cdots x_n$, then the tape is initialized so that $c_i = x_i$ for $i = 1 \ldots n$, and $c_i = \_\_$ for $i > n$
- The tape head is initially scanning $c_1$
- When the machine halts, and $m$ is the least $m \geq 0$ such that $c_{m+1} \notin \Sigma$, the output is defined to be the string $c_1 \cdots c_m$
A basic fact:

- Any TM can be simulated on a RAM
- Any RAM can be simulated on a TM

Definitions of computability, decidability, acceptance, and recognition carry over verbatim from RAM’s to TM’s

Consequence:

- A language is decidable/recursively enumerable on a RAM

  $\iff$

  it is decidable/recursively enumerable on a TM
Turing Machine Configurations:

Encode as a string:
\[ C = a_1 a_2 q a_3 \cdots a_m \]
Define a “follows” relation on configuration encodings:

\[ C \vdash C' \]

A computation on input \( x \) can be represented as

\[ \#C_0\#C_1\#\cdots\#C_k\# , \]

where

- \( C_0 = q_0x \) — initial configuration with input \( x \)
- \( C_{i-1} \vdash C_i \) for \( i = 1 \ldots k \)
- \( C_k \in \Gamma^* q_{\text{halt}} \Gamma^* \) — a “halting configuration”
Undecidable problems related to context-free languages

Some decidable languages (G is a CFG, x ∈ Σ*)

- \( A_{\text{CFG}} := \{ \langle G, x \rangle : x \in L(G) \} \)
- \( E_{\text{CFG}} := \{ \langle G \rangle : L(G) = \emptyset \} \)

An undecidable language:

- \( EQ_{\text{CFG}} := \{ \langle G_1, G_2 \rangle : L(G_1) = L(G_2) \} \)
- \( ALL_{\text{CFG}} := \{ \langle G \rangle : L(G) = \Sigma^* \} \)

Reduction: \( ALL_{\text{CFG}} \leq EQ_{\text{CFG}} \)
To show $\text{ALL}_{\text{CFG}}$ is undecidable, we give a reduction $\text{HALT}_{\text{TM}} \leq \text{ALL}_{\text{CFG}}$, where

$$\text{HALT}_{\text{TM}} := \{(M, x) : M \text{ is a TM that halts on input } x\}$$

We want to map $(M, x)$ to $(G_{M,x})$, with the property that

- if $M$ halts on input $x$, then $L(G_{M,x}) \not\subseteq \Sigma^*$
- if $M$ does not halt on input $x$, then $L(G_{M,x}) = \Sigma^*$
The reduction:

- Define $L_{M,x}$ to be the language of halting computations of $M$ on input $x$:
  
  \[
  \#C_0\#C_1\#\cdots\#C_k\#
  \]

- $L_{M,x}$ is either empty, or contains a single string

- For a string $w$, define $w^{-1}$ to be its reverse

- For a string $\alpha = \#\alpha_0\#\alpha_1\#\cdots\#\alpha_k\#$, define its "twist"
  
  \[
  \tilde{\alpha} := \#\alpha_0\#\alpha_1^{-1}\#\cdots\#\alpha_i^{(-1)^i}\#\cdots\#\alpha_k^{(-1)^k}\#
  \]

- Define $L'_{M,x} := \{\tilde{\alpha} : \alpha \in L_{M,x}\}$
The reduction (cont’d):

- If $M$ accepts $x$, then $L'_{M,x} \subseteq \Sigma^*$
- If $M$ does not accept $x$, then $L'_{M,x} = \Sigma^*$
- The goal is now to show that $L'_{M,x}$ is context free, and that we can effectively construct a grammar for $L'_{M,x}$, given $\langle M, x \rangle$
The reduction (cont’d):

- $\overline{L}_{M,x}'$ is the union of several regular languages
  - all strings that do not start and end with #
  - all strings of the form $\#\alpha_0\#\cdots$, where $\alpha_0$ is not the initial configuration of $M$ on input $x$
  - all strings that do not contain $q_{\text{halt}}$

and a context-free language:

- all strings of the form $\#\alpha_0\#\alpha_1^{-1}\#\cdots\#\alpha_k^{(-1)^k}\#$, with $k \geq 1$, such that for some $i = 1 \ldots k$, we have $\alpha_{i-1} \not\vdash \alpha_i$

PDA: use nondeterminism to guess $i$, and use the stack to compare $\alpha_{i-1}$ and $\alpha_i$
Other applications of the same idea:

- Define a 2PDA to be a push-down automaton with 2 stacks
  - The language acceptance problem for 2PDA’s is undecidable
    
    Idea: we can simulate a TM using two stacks

- One can define a notion of context-sensitive grammars, where the rewrite rules may have several symbols on LHS
  
  - The language acceptance problem for these types of grammars is undecidable

  Idea: we can use the rewrite rules to simulate a TM