Turing Machines (1)

- A Turing Machine (TM) is a mathematical model of a general-purpose computer.
- A TM is a generalisation of a PDA: $TM = FA +$ infinite tape
- Mainly used to study the notion of computation: what (exactly!) can computers do (given sufficient time and memory) and what can they not do.
- There are other notions of computation, e.g. the $\lambda$-calculus introduced by Alonzo Church (G54FOP!).

Turing Machines (2)

- All suggested notions of computation have so far proved to be equivalent.
- The Church-Turing Thesis: “Every function which would naturally be regarded as ‘computable’ can be computed by a TM”.
- At first, given how simple TMs are, it may seem surprising they can do much at all. E.g. how can they even add or multiply?
- We will see that a TM at least is more expressive than a PDA.

Definition of a Turing Machine

A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where
- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet
- $\Gamma$ is the tape alphabet, $\Sigma \subset \Gamma$ (finite)
- $\delta \in Q \times \Gamma \rightarrow \{\text{stop}\} \cup Q \times \Gamma \times \{L, R\}$ is the transition function
- $q_0 \in Q$ is the initial state
- $B$ is the blank symbol, $B \in \Gamma, B \notin \Sigma$
- $F \subseteq Q$ are the accepting (final) states
Instantaneous Description (ID)

Instantaneous Descriptions (ID) describe the state of a TM computation:

\[ \text{ID} = \Gamma^* \times Q \times \Gamma^* \]

\((\gamma_L, q, \gamma_R) \in \text{ID}\) means:

- TM is in state \(q\)
- \(\gamma_L\) is the non-blank part of the tape to the left of the head.
- \(\gamma_R\) is the non-blank part of the tape to the right of the head, including the current position.

The Next State Relation (1)

The next state relation on ID:

\[ \vdash_M \subseteq \text{ID} \times \text{ID} \]

Read

\[ id_1 \vdash_M id_2 \]

“TM \(M\) moves in one step from \(id_1\) to \(id_2\).”

The Language of a TM (1)

Let \(q, q' \in Q, x, y, z \in \Gamma, \gamma_L, \gamma_R \in \Gamma^*\)

1. \((\gamma_L, q, x\gamma_R) \vdash_M (\gamma_Ly, q', \gamma_R)\) if \(\delta(q, x) = (q', y, R)\)
2. \((\gamma_Lz, q, x\gamma_R) \vdash_M (\gamma_L, q', zy\gamma_R)\) if \(\delta(q, x) = (q', y, L)\)
3. \((\epsilon, q, x\gamma_R) \vdash_M (\gamma_L, q', By\gamma_R)\) if \(\delta(q, x) = (q', y, L)\)
4. \((\gamma_L, q, \epsilon) \vdash_M (\gamma_Ly, q', \epsilon)\) if \(\delta(q, B) = (q', y, R)\)
5. \((\gamma_Lz, q, \epsilon) \vdash_M (\gamma_L, q', zy)\) if \(\delta(q, B) = (q', y, L)\)
6. \((\epsilon, q, \epsilon) \vdash_M (\gamma_L, q', By)\) if \(\delta(q, B) = (q', y, L)\)

\[ L(M) = \{ w \in \Sigma^* \mid (\epsilon, q_0, w) \vdash_{M}^k (\gamma_L, q, \gamma_R) \land q \in F \} \]

A TM stops if it reaches an accepting state.

A TM stops in a non-accepting state if the transition function returns \text{stop} for that state and current tape input.

However, it may also never stop!

This is unlike the machines we have encountered before.
The Language of a TM (2)

If a particular TM $M$ always stops, either in an accepting or a non-accepting state, then $M$ decides $L(M)$.

Given that TMs model general purpose computers, it should not come as a surprise that they can loop. Consider e.g.

```
input x; while (x<10);
```

What may come as a surprise is that there are languages for which a TM necessarily cannot decide membership; i.e., will loop on some inputs.

Example

Construct a TM that accepts the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.

This is a language that cannot be defined by a CFG or recognized by a PDA.

On the whiteboard.

There are many TM simulators on-line. Try this (or some other) example with one of those. E.g.:

http://ironphoenix.org/tm

Recursive Language

$L$ is recursive if $L = L(M)$ for a TM $M$ such that

1. if $w \in L$, then $M$ accepts $w$ (and thus halts)
2. if $w \notin L$, then $M$ eventually halts without ever entering an accepting state.

Such a TM corresponds to an algorithm: a well-defined sequence of steps that always produces an answer in finite space and time.

We also say that $M$ decides $L$.

Recursively Enumerable (RE) Language

$L$ is recursively enumerable (RE) if $L = L(M)$ for a TM $M$.

I.e., $M$ is not required to halt for $w \notin L$.

Such a TM corresponds to a semi-algorithm.

Why “recursively enumerable”?

Because it is possible to construct a TM that enumerates all strings in such a language in some order. (But it will necessarily keep trying to enumerate strings forever.)
Decidable and Undecidable

There are even languages that have no TM! The non-RE languages.

- Decidable: a language or problem (encoded as a language) that is recursive.
- Undecidable: a language or problem that is RE but not recursive, or non-RE.

Halting Problem

Famous example of a RE language that is not recursive; i.e. an undecidable language.

On the whiteboard.

Other Undecidable Problems

- Whether two programs (computable functions) are equal
- Whether a CFG is ambiguous
- Whether two CFGs are equivalent
- Rice's Theorem: Whether the language of a given TM has some particular non-trivial property.

Consequence: There are lots of really useful programs that cannot be implemented perfectly. E.g. virus detection.