Week 12 - Turing Machines & Decidability

Billy Price

TURING MACHINES

Formal Definition

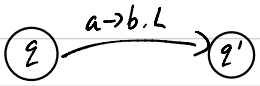
A Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where

- Q is a finite set of states,
- Γ is a finite tape alphabet, which includes the blank character, $\sqcup$,
- $\Sigma \subseteq \Gamma \setminus \{\sqcup\}$ is the input alphabet,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
- q_0 is the initial state,
- q_a is the accept state, and
- q_r ($\neq q_a$) is the reject state.

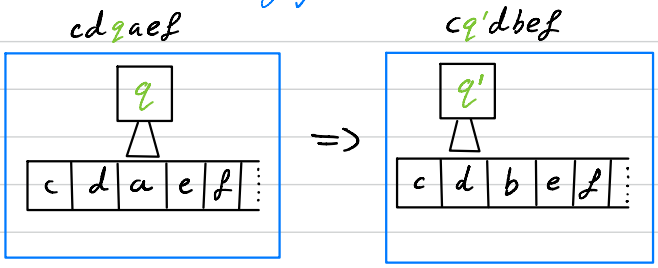
Transitions

If $\delta(q,a) = (q', b, L)$

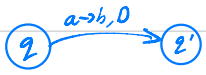
Graph Rep



Configurations



How To "Run" a TM

1. Write the input string at the start of the tape (the rest of the tape is filled with $\sqcup$).
2. Start in state q_0, with the tape head over the first cell.
3. If $\delta(q,a) = (q', b, D)$ 

   - Replace the a under the tape head with b
   - Shift the tape head once in direction D
   - Change to state q'
4. Repeat 3
5. If the state ever becomes q_a, halt & accept
6. If the state ever becomes q_r, halt & reject
7. Otherwise run forever... This counts as both not accepting & not rejecting.

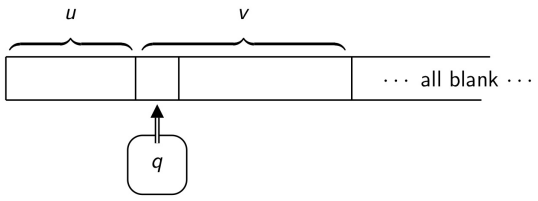
Languages

Given a TM M & language A we say...

- M recognises A if M accepts every string in A and doesn't accept any string not in A.
- M decides A if M accepts every string in A and rejects every string not in A

Configurations

We write uqv for this configuration:



On input aba, the example machine goes through 5 configurations:

$\epsilon q_0 aba$ (or just $q_0 aba$)
$\Rightarrow a q_1 ba$
$\Rightarrow ab q_0 a$
$\Rightarrow aba q_1 \sqcup$
$\Rightarrow aba \sqcup q_r$

Formal Acceptance

For all $q_i, q_j \in Q$, $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$, we have

$$
\begin{array}{ll}
uq_ibv \Rightarrow ucq_jv & \text{if } \delta(q_i, b) = (q_j, c, R) \\
q_ibv \Rightarrow q_jcv & \text{if } \delta(q_i, b) = (q_j, c, L) \\
uaq_ibv \Rightarrow uq_jacv & \text{if } \delta(q_i, b) = (q_j, c, L)
\end{array}
$$

The start configuration of M on input w is $q_0 w$.

M accepts w iff there is a sequence of configurations $C_1, C_2, \ldots, C_k$ such that

1. C_1 is the start configuration $q_0 w$,
2. $C_i \Rightarrow C_{i+1}$ for all $i \in \{1 \ldots k-1\}$, and
3. The state of C_k is q_a.

FACTS

- TMs are deterministic
- The "scanner" must always move left or right, unless you attempt to move left when the tape head is over the leftmost cell, in which case it stays in the same position
- The input doesn't need to be "consumed", if you reach the unique accept state, accept.
- TM doesn't accept $\not\equiv$ TM rejects.
- The tape is infinite to the right, not the left.
- When drawing a TM, any omitted transitions are assumed to be a transition to the reject state.

Big Fact

- Any algorithm written in any programming language with a single string input/output can be programmed as a TM.

DECIDABILITY

Billy Price

Encodings All of the "machines" we have discussed in this subject can each be encoded as a string (we wrote them down didn't we?). This is because they the definitions involve only a finite amount of information.

Notation Given a machine M, we write $\langle M \rangle$ to denote a string which encodes the data of M. Sometimes we pair other things, like strings in the encoding, like $\langle M, w \rangle$

Examples $\{ \langle D, w \rangle \mid D \text{ is a DFA that accepts } w \}$, $\{ \langle P \rangle \mid P \text{ is a PDA with 3 states} \}$

A language A is **Turing Recognisable** iff there is some TM that recognises A

Equivalently, can you describe (in python, haskell, C, pseudocode) a mechanistic algorithm that takes a string as input (probably an encoding of something) which halts and returns True if and only if the string is in A? If the string is not in A, the algorithm must either return False or never halt.
Such an algorithm/TM is called a **recogniser** for A.

A language A is **Decidable** iff there is some TM that decides A

Equivalently, can you describe (in python, haskell, C, pseudocode) a mechanistic algorithm that takes a string as input (probably an encoding of something) which halts and returns True if the string is in A, and halts and returns False if the string is not in A? There cannot be any input for which the algorithm does not halt.
Such an algorithm/TM is called a **decider** for A.

A language A is **Undecidable** iff it is not Decidable (no TM decides A)

Reducibility Let A, B, be languages. If a (hypothetical) decider for A can be used to construct a decider for B, we say B reduces to A. In this situation, we have two implications.

A decidable $\Rightarrow$ B decidable
B undecidable $\Rightarrow$ A undecidable

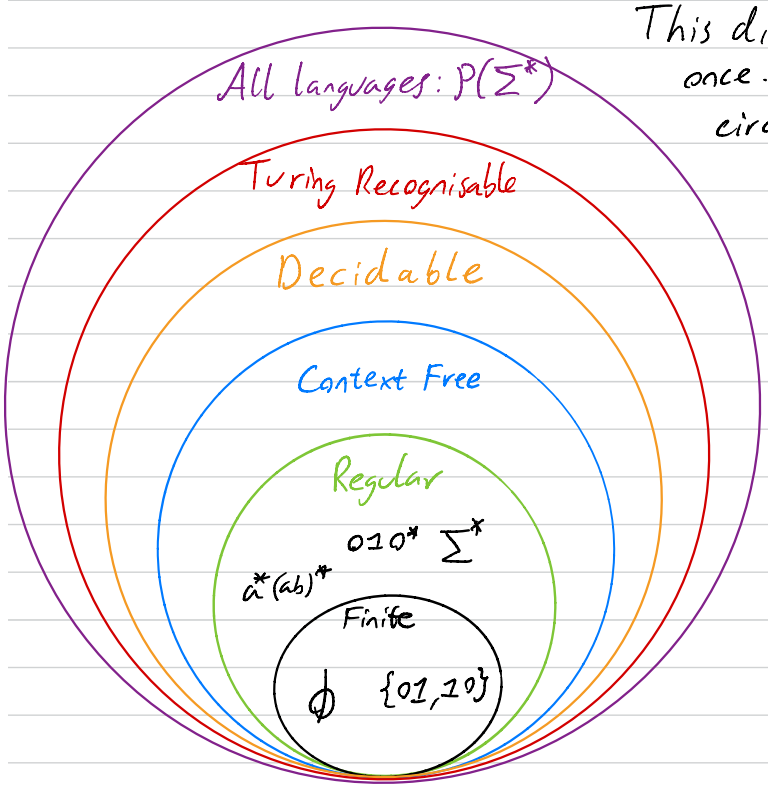
Decidability Proofs "Since A is decidable, it has a decider D_A. Using D_A, here is a decider, D_B, for the language B. Therefore B is decidable."

Undecidability Proofs "Suppose A was decidable. Then there is a decider, D_A, for A. Using D_A, here is a decider, D_B, for B. But we already know B is undecidable, so this is a contradiction. Therefore, A cannot be decidable, so it's undecidable."

LANGUAGES

Billy Price

DESCRIBER	CLASS	MACHINE
The set written as a list	Finite	— (use any below)
Regular Expressions	Regular	DFA / NFA
Context Free Grammar	Context-Free	PDA
—	Decidable	TM that always halts
—	Turing Recognisable	TM



This diagram states $2 \times \binom{6}{2}$ theorems at once. In particular, choose any 2 of these circles/classes of languages.

- Every language in the smaller class is also in the larger class.
- There is at least one language both in the larger class & <u>not</u> in the smaller class. Actually, there are always infinitely many like this