

Turing Machines

- A Turing machine is similar to a finite automaton with supply of unlimited memory.
- A Turing machine can do everything that any computing device can do.
- There exist problems that even a Turing machine cannot solve.

22c:131 Limits of Computation

Hantao Zhang

<http://www.cs.uiowa.edu/~hzhang/c131>

The University of Iowa
Department of Computer Science

TM versus FA

1. **write:** A TM can both write on the tape and read from it; an FA can only read its input (once).
2. **move:** The read/write head of a TM can move both to the left and to the right and some moves are not defined; an FA can move in one direction only and its next move is always defined (DFA only).
3. **size:** The tape of a TM is infinite; the input of an FA is finite.
4. **accept:** FA accepts a string when it has scanned all the input symbols and enters a final state; TM accepts a string as long as it enters a final state (one suffices).

Tape of a Turing Machine (TM)

- Memory is modeled by a *tape* of symbols.
- Initially the tape contains only the input string and is blank everywhere else.
- If a TM needs to store info, it may write on the tape.
- To read the info that it has written, TM can move its head back over its tape.
- TM continues to move until it enters a state whose next move is not defined.

Example TM computation

Construct a TM M_1 that tests the membership in the language $L_1 = \{w\#w \mid w \in \{0, 1\}^*\}$

- s_0 : if first symbol is 0 or 1, replace it by x , remember it as a ; if it is \sqcup , goto s_5 ; else **reject**;
- $s_1(a)$: move right until $\#$; if no $\#$ before \sqcup , **reject**;
- $s_2(a)$: move right until 0 or 1; if the current symbol is the same as a , then replace it by x ; else **reject**;
- s_3 : move left until $\#$;
- s_4 : move left until x and goto s_0 ;
- s_5 : move right until 0, 1, or \sqcup ; **accept** if \sqcup ; **reject** if 0, 1.

Example TM computation

Construct a TM M_1 that tests the membership in the language $L_1 = \{w\#w \mid w \in \{0, 1\}^*\}$

In other words: we want to design M_1 such that

$$M_1(w) = \text{accept iff } w \in L_1$$

Computations

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ computes as follows:

- M receives as input $w = a_1a_2 \dots a_n \in \Sigma^*$, $a_i \in \Sigma$, written on the leftmost squares of the tape and the rest of the tape is blank (i.e., filled with \sqcup)
- The head starts on the leftmost square of the tape.
- The first blank encountered shows the end of the input.
- Once it starts, it proceeds by the rules defined by δ .
- If M ever tries to move to the left of the leftmost square the head stays in the leftmost square even though δ indicate L .
- The computation continues until M cannot move; if M enters q_{accept} , the input string is **accepted**. M may go on forever as long as δ is defined.

Formal definition

A Turing machine is a 7-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where Q, Σ, Γ are finite sets and

1. Q is a set of states
2. Σ is the input alphabet and $\sqcup \notin \Sigma$
3. Γ is the tape alphabet, $\sqcup \in \Gamma, \Sigma \subset \Gamma$
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the initial state
6. $q_{\text{accept}} \in Q$ is the accept state (sometimes denoted q_a)
7. $q_{\text{reject}} \in Q$ is the reject state (sometimes denoted q_r)

Note: q_{reject} is optional in the definition.

Example TM computation

- s_3 : move left until #;
 $\delta(s_3, a) = \langle s_3, a, L \rangle$, where $a \in \{0, 1, x\}$, $\delta(s_3, \#) = \langle s_4, \#, L \rangle$
- s_4 : move left until x and goto s_0 ;
 $\delta(s_4, a) = \langle s_4, a, L \rangle$, where $a \in \{0, 1\}$, $\delta(s_4, x) = \langle s_0, x, R \rangle$
- s_5 : move right until 0, 1, or \sqcup ; **accept** if \sqcup ; **reject** if 0, 1.
 $\delta(s_5, x) = \langle s_5, x, R \rangle$, $\delta(s_5, \sqcup) = \langle s_a, \sqcup, L \rangle$

Example TM computation

- s_0 : if first symbol is 0 or 1, replace it by x , remember it as a ; if it is \sqcup , goto s_5 ; **else reject**;
 $\delta(s_0, 0) = \langle s_1(0), x, R \rangle$, $\delta(s_0, 1) = \langle s_1(1), x, R \rangle$, $\delta(s_0, \#) = \langle s_5, x, R \rangle$
- $s_1(a)$: move right until #; if no # before \sqcup , **reject**;
 $\delta(s_1(a), 0) = \langle s_1(a), 0, R \rangle$, $\delta(s_1(a), 1) = \langle s_1(a), 1, R \rangle$, $\delta(s_1(a), \#) = \langle s_2(a), \#, R \rangle$
- $s_2(a)$: move right until 0 or 1; if the current symbol is the same as a , then replace it by x ; **else reject**;
 $\delta(s_2(a), x) = \langle s_2(a), x, R \rangle$, $\delta(s_2(0), 0) = \langle s_3, x, L \rangle$, $\delta(s_2(1), 1) = \langle s_3, x, L \rangle$;

Formalizing TM computation

- A configuration C_1 **yields** a configuration C_2 if the TM can legally go from C_1 to C_2 in a single computation step
- **Formally:** suppose $a, b, c \in \Gamma$, $u, v \in \Gamma^*$ and $q_i, q_j \in Q$.
 1. We say that $ua q_i bv$ yields $uac q_j v$ if
 $\delta(q_i, b) = (q_j, c, R)$; (machine moves rightward)
 2. We say that $ua q_i bv$ yields $u q_j acv$ if
 $\delta(q_i, b) = (q_j, c, L)$; (machine moves leftward)

Configuration

A *configuration* C of M is a tuple $C = \langle u, q, v \rangle$, where $q \in Q$, $uv \in \Gamma^*$ is the tape content and the head is pointing to the first symbol of v .

- Configurations are used to formalize machine computation and therefore are represented by special symbols.
- Tape contains only \sqcup following the last symbol of v .

Example 2

M_2 is a Turing machine that decides $A = \{0^{2^n} \mid n \geq 0\}$.
some elementary

M_2 ="On input string w

1. Sweep left to right across the tape, crossing off every other 0; if the number of 0 is odd, **reject**;
2. If in stage 1 the tape contained a single 0, **accept**.
3. Return the head to the left-hand end of the tape.
4. Go to stage 1."

Head at one input end

Given $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- For the left-hand end:
 - the configuration $q_i bv$ yields $q_j cv$ if the transition is left moving, i.e., $\delta(q_i, b) = (q_j, c, L)$
 - the configuration $q_i bv$ yields $c q_j v$ for the right moving transition, i.e., $\delta(q_i, b) = (q_j, c, R)$
- For the right-hand end:
 - the configuration $ua q_i$ is equivalent to $ua q_i \sqcup$ because we assume that blanks follow the part of the tape represented in configuration. Hence we can handle this case as the previous

Example 2

1. Return the head to the left-hand end of the tape.
 $\delta(q_3, \sqcup) = \langle q_5, \sqcup, L \rangle$, $\delta(q_5, a) = \langle q_5, a, L \rangle$ for $a \in \{0, x\}$.
2. Go to stage 1 (b)
 $\delta(q_5, \sqcup) = \langle q_2, \sqcup, R \rangle$.

Example 2

1. Sweep left to right across the tape, crossing off every other 0; if the number of 0 is odd, **reject**;
 - (a) Mark the first 0 by \sqcup
 $\delta(q_1, 0) = \langle q_2, \sqcup, R \rangle$
 - (b) Cross off the next 0 after \sqcup
 $\delta(q_2, 0) = \langle q_3, x, R \rangle$
 - (c) Pass 0 at odd position; cross off 0 at even position.
 $\delta(q_3, 0) = \langle q_4, 0, R \rangle$, $\delta(q_4, 0) = \langle q_3, x, R \rangle$, $\delta(q, x) = \langle q, x, R \rangle$ for $q \in \{q_2, q_3, q_4\}$.
 - (d) If the number of 0 is odd, reject. $\delta(q_4, \sqcup) = \langle q_r, \sqcup, R \rangle$.
2. If in stage 1 the tape contained a single 0, **accept**.
 $\delta(q_2, \sqcup) = \langle q_a, \sqcup, R \rangle$

Accepting an input w

A Turing machine M accepts the input w if a sequence of configurations C_1, C_2, \dots, C_n exists such that:

1. C_1 is the start configuration, $C_1 = (\epsilon, q_0, w)$
2. Each C_i yields $C_{i+1}, i = 1, 2, \dots, n - 1$
3. C_n is an accepting configuration

Special configurations

- If the input of M is w and initial state is q_0 then $q_0 w$ is the *start configuration*
- $ua q_{accept}bv$ is called *accepting configuration*
- $ua q_{reject}bv$ is called *rejecting configuration*
- Accepting and rejecting configurations are also called *halting configurations*

Turing-recognizable language

A language L is Turing-recognizable if there is a Turing machine M that recognizes it

Language of M

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

Fail to accept

- TM fails to accept w by entering q_{reject} and thus rejecting, or by looping
- Sometimes it is difficult to distinguish a machine that fail to accept from one that merely takes long-time to halt.

Note

When we start a TM on an input w three cases can happen:

1. TM may accept w
2. TM may reject w
3. TM may **loop** indefinitely, i.e., TM does not halt.

Note: looping does not mean that machine repeats the same steps over and over again; looping may entail any simple or complex behavior that never leads to a halting state.

Question: is this real? I.e., can you indicate a computation that takes infinite many steps without repetition?

Turing-decidability

- A decider that recognizes some language is also said to *decide* that language
- A language is called *Turing-decidable* or simple *decidable* if some TM decides it.

Decider

A TM that halts on all inputs is called a decider.

Higher Level Descriptions

- We can give a formal description of a particular TM by specifying each of its seven components
- Defining δ can become cumbersome. To avoid this we use higher level descriptions which are precise enough for the purpose of understanding
- We want be sure that every higher level description is actually just a short hand for its formal counterpart.

Note

- Any regular language is Turing-decidable.
- Any context-free language is Turing-decidable.
- Every decidable language is Turing-recognizable (a language is Turing-recognizable if it is recognized by a TM).
- Certain Turing-recognizable languages are not decidable (to be decidable means to be decided by a TM which halts on all inputs)

Analyzing M_3

- In stage 1 M_3 operates as a finite automaton; no writing is necessary as the head moves from left to right:

1. $\delta(q_0, \sqcup) = (q_a, \sqcup, R), \delta(q_0, b) = (q_4, b, R)$
2. $\delta(q_0, a) = (q_1, A, R), \delta(q_1, b) = (q_2, b, R), \delta(q_1, \sqcup) = (q_a, \sqcup, R)$
3. $\delta(q_2, b) = (q_2, b, R), \delta(q_2, c) = (q_3, c, R)$
4. $\delta(q_3, c) = (q_3, c, R)$
5. $\delta(q_4, b) = (q_4, b, R), \delta(q_4, \sqcup) = (q_a, \sqcup, R),$

Example 3

M_3 is a Turing machine that performs some elementary arithmetic. It decides the language

$$C = \{a^i b^j c^k \mid i \times j = k, i, j, k \geq 0\}$$

M_3 ="On input string w

1. Scan the input from left to right to be sure that it is a member of $a^*b^*c^*$; *reject* if it is not; *accept* if it is ϵ, a^+ or b^+ .
2. Set the head pointing at the first a on the tape
3. Cross off an a and scan to the right until a b occurs. Shuttle between the b 's and c 's crossing off one of each until all b 's are gone
4. Restores the crossed off b 's and repeat stage 2 if there is another a to cross off. If all a 's are crossed off, check on whether all c 's are crossed off. If yes *accept*, otherwise *reject*."

Stage 3 cross a

- $\delta(q_5, A) = (q_6, \sqcup, R)$
- $\delta(q_6, a) = (q_7, A, R)$
- $\delta(q_6, b) = (q_8, b, L), \delta(q_8, \sqcup) = (q_7, E, R)$
- $\delta(q_7, a) = (q_7, a, R), \delta(q_7, B) = (q_7, B, R),$
 $\delta(q_7, b) = (q_9, B, R)$
- $\delta(q_9, b) = (q_9, b, R), \delta(q_9, C) = (q_9, C, R),$
 $\delta(q_9, c) = (q_{10}, C, L)$

Stage 2 finding the first a

- $\delta(q_3, \sqcup) = (q_5, \sqcup, L)$
- $\delta(q_5, x) = (q_5, x, L)$ for $x \in \{a, b, c\}$

Example 4

$M_4 = (Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$ is the TM that solves the *element distinctness problem*

M_4 works by comparing x_1 with x_2, \dots, x_k , then by comparing x_2 with x_3, \dots, x_k , and so on

Element distinctness problem

Given a list of strings over $\{0, 1\}$ separated by #, determine if all strings are different.

A TM that solves this problem accepts the language

$$E = \{\#x_1\#x_2\#\dots\#x_k \mid x_i \in \{0, 1\}^*, x_i \neq x_j \text{ for } i \neq j\}$$

Marking tape symbols

- In stage two the machine places a mark above a symbol, # in this case.
- In the actual implementation the machine has two different symbols, # and $\overset{\bullet}{\#}$ in the tape alphabet Σ
- Thus, when machine places a mark above symbol x it actually write the marked symbol of x at that location
- Removing the mark means write the symbol at the location where the marked symbol was.
- Assumption: all symbols of the tape alphabet have marked versions too

Informal description

M_4 ="On input w :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a # continue with the next stage. Otherwise *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only x_k was present, so *accept*
3. By zig-zagging, compare the two strings to the right and to the left of the marked #. If they are equal, *reject*
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time if no # is available for the rightmost mark, all strings have been compared, so *accept*.
5. Go to stage 3"