Computability Theory
TM examples

Haniel Barbosa

THE UNIVERSITY OF IOWA
Chapter 3 of [Sipser 1996], 3rd edition. Section 3.1.
Higher lever descriptions

- Defining a TM formally can be cumbersome, so we will use high level descriptions which are precise enough for understanding.

- We want to be sure that every higher level description directly corresponds to its formal counterpart.
Example: TM that decides $A = \{0^{2^n} \mid n \geq 0\}$

$M$ decides the language of all strings of 0s whose length is a power of 2.

$M =$ “On input string $w$:

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, accept
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject
4. Return the head of the left-hand end of the tape.
5. Go to stage 1.”
Example: TM performing elementary arithmetic

$M$ decides the language $C = \{a^i b^j c^k \mid i \times j = k, \text{ with } i,j,k \geq 0\}$

$M =$ “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 $\epsilon$, $a^+$ or $b^+$

2. Set the head pointing at the first $a$ in the tape

3. Cross off an $a$ and scan to the right until a $b$ occurs. Shuttle between the $b'$s and the $c'$s, crossing off one of each until all $b'$s are gone. If all $c'$s have been crossed off and some $b'$s remain, reject

4. Restore the crossed off $b'$s and repeat stage 3 if there is another $a$ to cross off. If all $a'$s have been crossed off, determine whether all $c'$s also have been crossed off. If yes, accept, otherwise reject.”
Element distinctness problem

Given a list of strings over \{0, 1\} separated by #, determine if all strings are different. A TM $M$ that solves this problem accepts the language

$$E = \{#x_1#x_2# \ldots #x_l \mid \text{each } x_i \in \{0, 1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M$ = “On input string $w$:
1. Place a mark on top of the leftmost tape symbol. If it was a blank, accept. If it 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 $x_1$ was present, so accept
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, reject
4. Move the rightmost of the two markers 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 the strings have been compared, so accept.
5. Go to stage 3.”
Levels of abstraction

What is the right level of detail to give when describing a Turing Machine algorithm?

The three possibilities are:

1. *Formal description*: spelling out the 7-tuple of a TM. This is the most detailed level of description.

2. *Implementation description*: using English prose to describe the way the TM moves its head and the way it stores data on its tape. No details of the transitions are given.

3. *High-level description*: using English prose to describe the algorithm, ignoring the implementation model. No need to mention how TM manages its head and tape.

As one might expect, the "right" level of abstraction depends on the application at hand.
Levels of abstraction

What is the right level of detail to give when describing a Turing Machine algorithm?

The three possibilities are:

1. *Formal description*: spelling out the 7-tuple of a TM. This is the most detailed level of description.

2. *Implementation description*: using English prose to describe the way the TM moves its head and the way it stores data on its tape. No details of the transitions are given.

3. *High-level description*: using English prose to describe the algorithm, ignoring the implementation model. No need to mention how TM manages its head and tape.

As one might expect, the “right” level of abstraction depends on the application at hand.