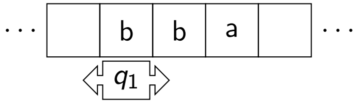
Turing Machines (Section 12.1)

Turing Machines

- A Turing Machine (TM) is a simple computer that has an infinite amount of storage in the form of cells on an infinite tape. There is a control unit that contains a finite set of state transition instructions.



Input/Output Tape

TM Operation

- A instruction reads the symbol in the current tape cell, writes a symbol to the same cell, and then moves the tape head one cell left or right or remains at the same cell, after which the machine enters a new state.
- Assumptions:
  - The symbol Λ denotes a blank cell.
  - The tape head is initially at the left end of a nonempty input string (unless specified otherwise).
  - There is a designated start state.
  - There is one “halt” state.
  - The moves are denoted by L, S, R for move left, stay and move right.

TM Operations continued

- Instructions are represented in graphical form or as 5-tuples, as shown:

$$\begin{tikzcd} \boxed{i} \arrow[r, "\frac{a}{b,R}"] & \boxed{j} \end{tikzcd}$$

or (i, a, b, R, j)

Acceptance

- An input string on the tape is *accepted* by the TM if the machine enters the halt state. The *language* of a TM is the set of strings accepted by the machine.
- *Example:* What is the language of the following simple TM:
  - $(0, a, a, R, \text{halt})$
- *Answer:* If the input string begins with the letter a, then the TM executes the instruction and halts. So the string is accepted. Therefore the language of the TM is the set of all strings that begin with the letter a.

Example

- Construct a TM to accept the language $\{ab^n | n \in \mathbb{N}\}$

Example

- Construct a TM to accept the language $\{ab^n | n \in \mathbb{N}\}$
- *Solution:* Let the start state be state 0. The idea is to see whether a is in the current cell. If so, move right to scan b's until Λ is found.
    - $(0, a, a, R, 1)$
    - $(1, \Lambda, \Lambda, S, \text{halt})$
    - $(1, b, b, R, 1)$

Another Example

- Contruct a TM to accept the language $\{ab^n a | n \in \mathbb{N}\}$

Another Example

- Contruct a TM to accept the language $\{ab^na | n \in \mathbb{N}\}$
- *Solution:* Let the start state be 0. The idea is to see whether a is in the current cell. If so, move right to scan b's until a is found. Then make sure there is Λ to the right.
  - $(0, a, a, R, 1)$
  - $(1, a, a, R, 2)$
  - $(1, b, b, R, 1)$
  - $(2, \Lambda, \Lambda, S, \text{halt})$

Yet Another Example

- Construct a TM to accept the language of the regular expression $a(a+b)^*$.

Yet Another Example

- Construct a TM to accept the language of the regular expression $a(a+b)^*$.
- *Solution:* Let the start state be state 0. The idea is to see whether a is in the current cell. If so, move right to scan any a's or b's until Λ is found.
  - $(0, a, a, R, 1)$
  - $(1, a, a, R, 1)$
  - $(1, b, b, R, 1)$
  - $(1, \Lambda, \Lambda, S, \text{halt})$

Context-free language

- Find a TM to accept $\{a^n b^n | n \in \mathbb{N}\}$.

Context-free language

- Find a TM to accept $\{a^n b^n | n \in \mathbb{N}\}$.
- *Solution:* Let the start state be state 0. The idea is to see whether a is in the current cell. If so, write X and scan right looking for a b to replace by Y.
  - $(0, \Lambda, \Lambda, S,\text{halt})$ accept Λ
  - $(0, a, X, R, 1)$ mark a with X
  - $(0, Y, Y, R, 3)$ no more a's
- Scan right looking for b to pair with a:
  - $(1, a, a, R, 1)$
  - $(1, Y, Y, R, 1)$
  - $(1, b, Y, L, 2)$ mark b with Y

Continued

- Scan back left looking for X:
  - $(2, a, a, L, 2)$
  - $(2, Y, Y, L, 2)$
  - $(2, X, X, R, 0)$
- Scan right looking for Λ and halt:
  - $(3, Y, Y, R, 3)$
  - $(3, \Lambda, \Lambda, S, \text{halt})$

TMs are Powerful

- We can generalize the preceding example to a TM that accepts the non-context-free language $\{a^n b^n c^n | n \in \mathbb{N}\}$. A complete example of this is in the book on page 837.
- *An informal description of the algorithm:* If the current cell is empty, then halt with success. Otherwise, if the current cell contains a, then write an X in the cell and scan right, looking for a corresonding b to the right of any a's and replace it with Y. Then continue scanning to the right, looking for a corresponding c to the right of any b's and replace it by Z. Now scan left to the X and see whether there is an a to its right. If so, then start the process again. If there are no a's, then scan right to make sure there are no b's or c's.

Turing Machines with Output

- The big idea is that we use the tape for the output. So we have to specify the form of the output on the tape when the machine halts.
- *Example:* Find a TM to add 4 to a natural number represented in binary. Start with the tape head at the right end of the input string and halt with the tape head at the left end of the output string.

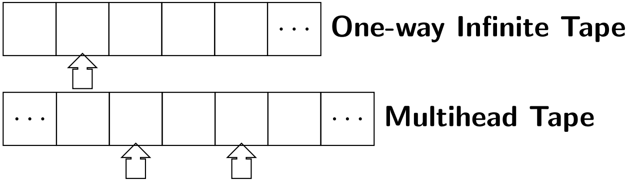
Turing Machines with Output

- The big idea is that we use the tape for the output. So we have to specify the form of the output on the tape when the machine halts.
- *Example:* Find a TM to add 4 to a natural number represented in binary. Start with the tape head at the right end of the input string and halt with the tape head at the left end of the output string.
- *Solution:* Let the start state be 0.
- next slide . . .

TM program

- Move two cells left:
  - $(0, 0, 0, L, 1)$
  - $(0, 1, 1, L, 1)$
  - $(1, 0, 0, L, 2)$
  - $(1, 1, 1, L, 2)$
  - $(1, \Lambda, 0, L, 2)$
- Add 1
  - $(2, 0, 1, L, 3)$ Move left
  - $(2, 1, 0, L, 2)$ Carry
  - $(2, \Lambda, 1, S, \text{halt})$ Done
- Find left end of the string
  - $(3, 0, 0, L, 3)$
  - $(3, 1, 1, L, 3)$
  - $(3, \Lambda, \Lambda, R, \text{halt})$ Done

Alternative TM Definitions



- Or we can have a machine with multiple tapes.
- An *n*-head or *n*-tape instruction contains *n*-tuples for the cell contents and the move operations.
- *Example:* A typical instruction for a 2-head or a 2-tape TM is:
  - $(0, (a, b), (c, d), (L, R), 1)$

Example 2-tape instructions

- Implement some sample PDA instructions as TM instructions for a 2-tape TM. Assume one tape holds the input string and the other tape holds the stack. Assume the input is scanned left to right and the stack grows from right to left.

PDA Instruction	TM Instruction
1. $(i, a, X, \mathrm{nop}, j)$	1. $(i, (a, X), (a, X), (R, S), j)$
2. $(i, a, X, \mathrm{pop}, j)$	2. $(i, (a, X), (a, \Lambda), (R, R), j)$
3. $(i, a, X, \mathrm{push(Y)}, j)$	3. $(i, (a, X), (a, X), (S, L), k)$ and $(k, (a, \Lambda), (a, Y), (R, S), j)$
4. $(i, \Lambda, X, \mathrm{pop}, j)$	4. For each letter a: $(i, (a, X), (a, \Lambda), (S, R), j)$

Nondeterministic Turing Machines

- *Example:* Generate an arbitrary string over $\{a, b\}$ of length n.

Nondeterministic Turing Machines

- *Example:* Generate an arbitrary string over $\{a, b\}$ of length n.
- *Solution:* Assume the input is any string over $\{a, b\}$ of length n. The tape head is at the right end of the output string.
  - $(0, a, a, R, 0)$
  - $(0, a, b, R, 0)$
  - $(0, b, a, R, 0)$
  - $(0, b, b, R, 0)$
  - $(0, \Lambda, \Lambda, L, \text{halt})$

Same Power

- Nondeterministic TMs have the same power as deterministic TMs.
- *The idea:* Any nondeterministic TM can be simulated by a deterministic TM that simulates all 1-step computations, then all 2-step computations and so on. The process stops if some computation enters the halt state.

Universal Turing Machine

- So far, each problem we have solved involved constructing a different Turing machine specific to that problem. Is there a more general Turing machine that acts like a general-purpose computer? Yes.
- Imagine a Turing machine that takes as input an arbitrary Turing machine, M, together with an arbitrary input for M and performs the execution of M on the input.
- Such a machine is called a universal Turing machine. I am using one right now (more or less) to show you these slides.

Universal TM description

- Describe a universal TM, call it U. Since U can have only a finite number of instructions and a finite alphabet of tape cell symbols, we have to discuss the representation of any TM in terms of the fixed symbols of U.
- We assume a fixed alphabet of input symbols and a fixed set of states in our control. We can have a unique symbol for each of these input symbols and states. We have two strings for input, one for the Turing machine (which is an encoding of the finite number of 5-tuples that make up the Turing machine program) and one for the input (which is encoded in the alphabet of input symbols that our Turing machine recognizes (say, for example, ASCII)).

Universal TM details

- Now, imagine a 3-tape machine. One tape is set up with the TM program. One tape is set up with the input string. The third tape holds a representation of the start state. Now *U* repeatedly performs the following actions:
  - If the state on tape 3 is the halt state, then halt.
  - Otherwise, get the current state from tape 3 and the current input symbol from tape 2. With this information, find the proper instruction on tape 1 .
  - Write the next state at the beginning of tape 3, then perform the indicated write and move operations on tape 2.
- Tada.