COL 352 Introduction to Automata and Theory of Computation

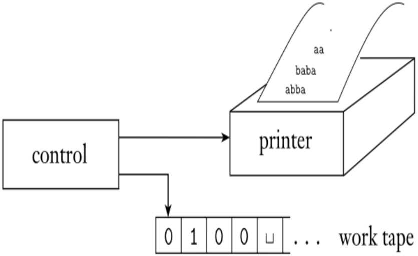
Nikhil Balaji

Bharti 420
Indian Institute of Technology, Delhi
nbalaji@cse.iitd.ac.in

March 22, 2023

Lecture 24: Turing Machines: Variants, CT Thesis (Part 3)

Enumerators



- Turing machine with an attached printer.
- **Exercise:** Formally define it.
- An enumerator E starts with a blank input on its work tape.
- If the enumerator doesn't halt, it may print an infinite list of strings.
- The language enumerated by E is the collection of all the strings that it eventually prints out.
- E may generate the strings of the language in any order, possibly with repetitions.

Enumerators vs Recognizers

Theorem

A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof.

Enumerators vs Recognizers

Theorem

A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof.

($\Rightarrow$) On input w:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept.

($\Leftarrow$)

Enumerators vs Recognizers

Theorem

A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof.

($\Rightarrow$) On input w:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept.

($\Leftarrow$) Ignore the input. Repeat the following for $i = 1, 2, 3, \ldots$.

Enumerators vs Recognizers

Theorem

A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof.

($\Rightarrow$) On input w:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept.

($\Leftarrow$) Ignore the input. Repeat the following for $i = 1, 2, 3, \ldots$.

1. Run M for i steps on each input, $s_1, s_2, \ldots, s_i$.
2. If any computations accepts, print out the corresponding s_j.

$\square$

Remark: Turing Recognizable = Recursively Enumerable languages.

Queue automata

- A DQA is like a push-down automaton except that the stack is replaced by a queue.

Queue automata

- A DQA is like a push-down automaton except that the stack is replaced by a queue.
- A queue is a tape allowing symbols to be written only on the left-hand end and read only at the right hand-end.

Queue automata

- A DQA is like a push-down automaton except that the stack is replaced by a queue.
- A queue is a tape allowing symbols to be written only on the left-hand end and read only at the right hand-end.
- Each write operation (called a push) adds a symbol to the left-hand end of the queue.

Queue automata

- A DQA is like a push-down automaton except that the stack is replaced by a queue.
- A queue is a tape allowing symbols to be written only on the left-hand end and read only at the right hand-end.
- Each write operation (called a push) adds a symbol to the left-hand end of the queue.
- Each read operation (called a pull) reads and removes a symbol at the right-hand end.

Queue automata

- A DQA is like a push-down automaton except that the stack is replaced by a queue.
- A queue is a tape allowing symbols to be written only on the left-hand end and read only at the right hand-end.
- Each write operation (called a push) adds a symbol to the left-hand end of the queue.
- Each read operation (called a pull) reads and removes a symbol at the right-hand end.
- **Initial condition:** the input tape contains a cell with a blank symbol following the input, to detect end of the input.
- **Computation:** Acceptance by entering a special accept state at any time.

Note: As with a PDA, the input of a DQA is placed on a separate read-only input tape, and the head on the input tape can move only from left to right

Queues are more powerful than stacks

Theorem

Language can be recognized by a DQA, iff it is is Turing-recognizable

Proof Sketch.

Idea: Show any DQA Q can be simulated with a 2-tape TM M.

Queues are more powerful than stacks

Theorem

Language can be recognized by a DQA, iff it is is Turing-recognizable

Proof Sketch.

Idea: Show any DQA Q can be simulated with a 2-tape TM M. Show that any single-tape deterministic TM D can be simulated by a DQA Q. □

Simulating a DQA by a TM

- The first tape of M holds the input, second tape holds the queue.
- To simulate reading Q's next input symbol, M reads the symbol under the first head and moves to the right.

Simulating a DQA by a TM

- The first tape of M holds the input, second tape holds the queue.
- To simulate reading Q's next input symbol, M reads the symbol under the first head and moves to the right.
- To simulate a $push\ a$, M writes a on the leftmost blank cell of the second tape.

Simulating a DQA by a TM

- The first tape of M holds the input, second tape holds the queue.
- To simulate reading Q's next input symbol, M reads the symbol under the first head and moves to the right.
- To simulate a $push\ a$, M writes a on the leftmost blank cell of the second tape.
- To simulate a $pull$, M reads the rightmost symbol on the second tape and shifts the tape one symbol leftward.

Simulating a TM by DQA

$$M = (S_M, \Sigma, \Gamma_M, \delta_M, q_0^M, q_a^M, q_r^M)$$

$$Q = (S_Q, \Sigma, \Gamma_M \cup \hat{\Gamma}_M, \delta_Q, q_0^Q, \{q_a^M, q_r^M\})$$

Simulating a TM by DQA

$$M = (S_M, \Sigma, \Gamma_M, \delta_M, q_0^M, q_a^M, q_r^M)$$
$$Q = (S_Q, \Sigma, \Gamma_M \cup \hat{\Gamma}_M, \delta_Q, q_0^Q, \{q_a^M, q_r^M\})$$

- For each symbol $c \in \Gamma_M$, Q also has the corresponding $\hat{c}$ to denote the head.
- Let Q also have an end of tape marker \$.

Simulating a TM by DQA

$$M = (S_M, \Sigma, \Gamma_M, \delta_M, q_0^M, q_a^M, q_r^M)$$
$$Q = (S_Q, \Sigma, \Gamma_M \cup \hat{\Gamma}_M, \delta_Q, q_0^Q, \{q_a^M, q_r^M\})$$

- For each symbol $c \in \Gamma_M$, Q also has the corresponding $\hat{c}$ to denote the head.
- Let Q also have an end of tape marker \$.
- Q simulates M by maintaining a copy of M's tape in the queue.
- Q can scan the tape from right to left by pulling symbols from the right-hand end of the queue and pushing them back on the left-hand end side, until \$ is seen.
- When a $\hat{c}$ symbol is encountered, Q can determine M's next move, because Q can record M's current state in its control.

Computation Simulation

- If M's tape head moves leftwards, the updating of the queue is done by writing the new symbol c instead of the old $\hat{c}$ and moving the $\hat{}$ one symbol left.

Computation Simulation

- If M's tape head moves leftwards, the updating of the queue is done by writing the new symbol c instead of the old $\hat{c}$ and moving the ˆ one symbol left.
- Formally, if current configuration is $ua\hat{b}tv$ and $\delta(q,b) = (q', c, L)$ then the next configuration is $u\hat{a}ctv$ and is obtained by:

Computation Simulation

- If M's tape head moves leftwards, the updating of the queue is done by writing the new symbol c instead of the old $\hat{c}$ and moving the $\hat{}$ one symbol left.
- Formally, if current configuration is $ua\hat{b}tv$ and $\delta(q, b) = (q', c, L)$ then the next configuration is $u\hat{a}ctv$ and is obtained by:
  - $pull\ v$; $push\ v$;
  - $pull\ t$; $push\ t$;
  - $pull\ \hat{b}$; $push\ c$; $pull\ a$; $push\ \hat{a}$;
  - $pull\ u$; $push\ u$;

Computation Simulation

- If M's tape head moves leftwards, the updating of the queue is done by writing the new symbol c instead of the old $\hat{c}$ and moving the $\hat{}$ one symbol left.
- Formally, if current configuration is $ua\hat{b}tv$ and $\delta(q, b) = (q', c, L)$ then the next configuration is $u\hat{a}ctv$ and is obtained by:
  - $pull\ v$; $push\ v$;
  - $pull\ t$; $push\ t$;
  - $pull\ \hat{b}$; $push\ c$; $pull\ a$; $push\ \hat{a}$;
  - $pull\ u$; $push\ u$;
- How about move right? (Exercise!)

2 stacks?

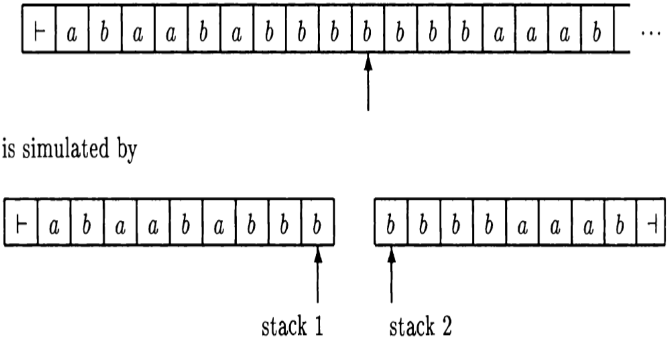
- Same as the regular PDA but now you have two stacks

2 stacks?

- Same as the regular PDA but now you have two stacks
- As powerful as TMs.

2 stacks?

- Same as the regular PDA but now you have two stacks
- As powerful as TMs.



Nondeterministic Turing Machines

A NTM is defined in the expected way: at any point in a computation the machine may proceed according to several possibilities

Nondeterministic Turing Machines

A NTM is defined in the expected way: at any point in a computation the machine may proceed according to several possibilities

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Nondeterministic Turing Machines

A NTM is defined in the expected way: at any point in a computation the machine may proceed according to several possibilities

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

- Computation performed by a NTM is a tree whose branches correspond to different possibilities for the machine

Nondeterministic Turing Machines

A NTM is defined in the expected way: at any point in a computation the machine may proceed according to several possibilities

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

- Computation performed by a NTM is a tree whose branches correspond to different possibilities for the machine
- If some branch of the computation tree leads to the accept state, the machine accepts the input
- Can nondeterministic Turing machines compute more functions than deterministic Turing machines?

Nondeterministic Turing Machines

A NTM is defined in the expected way: at any point in a computation the machine may proceed according to several possibilities

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

- Computation performed by a NTM is a tree whose branches correspond to different possibilities for the machine
- If some branch of the computation tree leads to the accept state, the machine accepts the input
- Can nondeterministic Turing machines compute more functions than deterministic Turing machines?

Theorem

Every nondeterministic Turing machine, N has an equivalent deterministic Turing machine D.

Example: Finding Integer roots of Polynomials

- Given polynomial

$$p(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

where $a_i \in \mathbb{Z}$, find an integer root.

Example: Finding Integer roots of Polynomials

- Given polynomial

$$p(x) = a_1x^n + a_2x^{n-1} + \cdots + a_nx + a_{n+1}$$

where $a_i \in \mathbb{Z}$, find an integer root.

- **Exercise:** Let there be a root at $x = x_0$ and a_{max} be the largest absolute value of a a_i. Show that

$$|x_0| < (n+1)\frac{a_{max}}{|a_1|}$$

Proof of $L(N) = L(D)$

Proof Idea.

Show that a NTM N can be simulated with a DTM D.

Proof of $L(N) = L(D)$

Proof Idea.

Show that a NTM N can be simulated with a DTM D. In this simulation D tries all possible branches of N's computation. If D ever finds the accept state on one of these branches then it accepts. Otherwise D's simulation will not terminate.

- N's computation on an input w is a tree, $N(w)$.
- Each branch of $N(w)$ represents one of the branches of the nondeterminism.

Proof of $L(N) = L(D)$

Proof Idea.

Show that a NTM N can be simulated with a DTM D. In this simulation D tries all possible branches of N's computation. If D ever finds the accept state on one of these branches then it accepts. Otherwise D's simulation will not terminate.

- N's computation on an input w is a tree, $N(w)$.
- Each branch of $N(w)$ represents one of the branches of the nondeterminism.
- Each node of $N(w)$ is a configuration of N.
- The root of $N(w)$ is the start configuration.

Proof of $L(N) = L(D)$

Proof Idea.

Show that a NTM N can be simulated with a DTM D. In this simulation D tries all possible branches of N's computation. If D ever finds the accept state on one of these branches then it accepts. Otherwise D's simulation will not terminate.

- N's computation on an input w is a tree, $N(w)$.
- Each branch of $N(w)$ represents one of the branches of the nondeterminism.
- Each node of $N(w)$ is a configuration of N.
- The root of $N(w)$ is the start configuration.
- D searches $N(w)$ for an accepting configuration.

□

A tempting bad idea

- Design D to explore the tree $N(w)$ using DFS.
-

A tempting bad idea

- Design D to explore the tree $N(w)$ using DFS.
- A depth-first search goes all the way down on one branch before backing up to explore next branch. Hence, D could go forever down on an infinite branch and miss an accepting configuration on an other branch.

A better idea

- Design D to explore the tree by using a breadth-first search
- This strategy explores all branches at the same depth before going to explore any branch at the next depth.
- Hence, this method guarantees that D will visit every node of $N(w)$ until it encounters an accepting configuration.

Proof.

D has three tapes:

- Tape 1 always contains the input and is never altered
- Tape 2 (called the simulation tape) maintains a copy of N's tape on some branch of its nondeterministic computation
- Tape 3 (called address tape) keeps track of D's location in N's nondeterministic computation tree

$\square$

Other variants

- Multi-dimensional tapes.

Other variants

- Multi-dimensional tapes.
- Counter machines

Other variants

- Multi-dimensional tapes.
- Counter machines
- Your favorite programming language ("Turing-completeness")

Other variants

- Multi-dimensional tapes.
- Counter machines
- Your favorite programming language ("Turing-completeness")
- Cellular automata

Other variants

- Multi-dimensional tapes.
- Counter machines
- Your favorite programming language ("Turing-completeness")
- Cellular automata
- ...
- ...

What is computation?

- In 1900, David Hilbert listed out 23 problems as challenges for 20th century at the Int. Cong. of Mathematicians in Paris.

What is computation?

- In 1900, David Hilbert listed out 23 problems as challenges for 20th century at the Int. Cong. of Mathematicians in Paris.
- 10th problem: Devise an algorithm (a process doable using a finite no.of operations) to test if a (multivariate) polynomial has integral roots.

What is computation?

- In 1900, David Hilbert listed out 23 problems as challenges for 20th century at the Int. Cong. of Mathematicians in Paris.
- 10th problem: Devise an algorithm (a process doable using a finite no.of operations) to test if a (multivariate) polynomial has integral roots.

What is computation?

- In 1900, David Hilbert listed out 23 problems as challenges for 20th century at the Int. Cong. of Mathematicians in Paris.
- 10th problem: Devise an algorithm (a process doable using a finite no.of operations) to test if a (multivariate) polynomial has integral roots.
- Now we know that no such algorithm exists. But how to prove this without a mathematical definition of an algorithm?

Church-Turing thesis

Alonso Church
(1903–1995)

Alan Turing
(1912–1954)

Turing's paper

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*,