UIT1501 Finite Automata Theory

Turing Machines

Chandrabose Aravindan
<AravindanC@ssn.edu.in>

Professor of Information Technology
SSN College of Engineering

November 21, 2022

Outline

1. What have we seen so far?
   - Context-free Grammar
   - Pushdown Automata
   - Relationship between CFG and PDA
   - Normal Forms
   - Pumping Lemma and Closure Properties

Outline

1. What have we seen so far?
   - Context-free Grammar
   - Pushdown Automata
   - Relationship between CFG and PDA
   - Normal Forms
   - Pumping Lemma and Closure Properties
2. Turing Machines
   - Informal Introduction
   - Formal Definitions
   - Examples
   - Computing integer-valued functions
   - Languages of Turing Machines

Outline

Outline

Context-Free Grammar

- We have discussed about context-free grammars.
- We have seen some examples of constructing CFGs.
- We have also seen the formal definitions of derivations, starred derivations, leftmost derivations and rightmost derivations.
- We have seen the formal definition of the language of a CFG and formal definition of context-free languages.
- We have discussed about inferring strings and parse trees for context-free grammars
- We have looked at examples of ambiguity in CFG and inherently ambiguous languages.

Context-Free Grammar

- We have discussed about context-free grammars.
- We have seen some examples of constructing CFGs.
- We have also seen the formal definitions of derivations, starred derivations, leftmost derivations and rightmost derivations.
- We have seen the formal definition of the language of a CFG and formal definition of context-free languages.
- We have discussed about inferring strings and parse trees for context-free grammars
- We have looked at examples of ambiguity in CFG and inherently ambiguous languages.
- Any Questions?

Pushdown Automata

- We have seen the pushdown automata model — formal definitions, transition diagrams, instantaneous descriptions
- We have discussed two different models of PDA — one that accepts by entering a final state and the other by emptying its stack.
- We have shown that both these models are equivalent.
- We have seen some examples of designing PDA for given languages.

Pushdown Automata

- We have seen the pushdown automata model — formal definitions, transition diagrams, instantaneous descriptions
- We have discussed two different models of PDA — one that accepts by entering a final state and the other by emptying its stack.
- We have shown that both these models are equivalent.
- We have seen some examples of designing PDA for given languages.
- Any Questions?

PDA, CFG, and DPDA

- We have seen how a PDA can be systematically constructed to accept the words of language of a given CFG.
- We have also seen the converse — to systematically construct an equivalent CFG from a given PDA
- Thus the set of languages of PDA is exactly same as the set of languages of CFG — Context-free Languages
- We have seen a deterministic version of PDA (DPDA).
- DPDA do not have the same expressive power as PDA but they are more powerful than FA.
- Thus, we have $\mathcal{R} \subset DCFL \subset CFL$
- It remains to be seen whether $CFL = 2^{\Sigma^*}$ or $CFL \subset 2^{\Sigma^*}$???

PDA, CFG, and DPDA

- We have seen how a PDA can be systematically constructed to accept the words of language of a given CFG.
- We have also seen the converse — to systematically construct an equivalent CFG from a given PDA
- Thus the set of languages of PDA is exactly same as the set of languages of CFG — Context-free Languages
- We have seen a deterministic version of PDA (DPDA).
- DPDA do not have the same expressive power as PDA but they are more powerful than FA.
- Thus, we have $\mathcal{R} \subset DCFL \subset CFL$
- It remains to be seen whether $CFL = 2^{\Sigma^*}$ or $CFL \subset 2^{\Sigma^*}$???
- Any Questions?

Normal forms

- We have seen how a CFG can be simplified by
  - eliminating all ϵ-productions
  - eliminating all unit productions
  - eliminating all useless symbols
- We have seen Chomsky Normal Form (CNF) and how to convert any grammar to its CNF
- We have seen Greibach Normal Form (CNF) and how to convert any grammar to its GNF
- It remains to be seen whether $CFL = 2^{\Sigma^*}$ or $CFL \subset 2^{\Sigma^*}$???

Normal forms

- We have seen how a CFG can be simplified by
  - eliminating all ϵ-productions
  - eliminating all unit productions
  - eliminating all useless symbols
- We have seen Chomsky Normal Form (CNF) and how to convert any grammar to its CNF
- We have seen Greibach Normal Form (CNF) and how to convert any grammar to its GNF
- It remains to be seen whether $CFL = 2^{\Sigma^*}$ or $CFL \subset 2^{\Sigma^*}$???
- Any Questions?

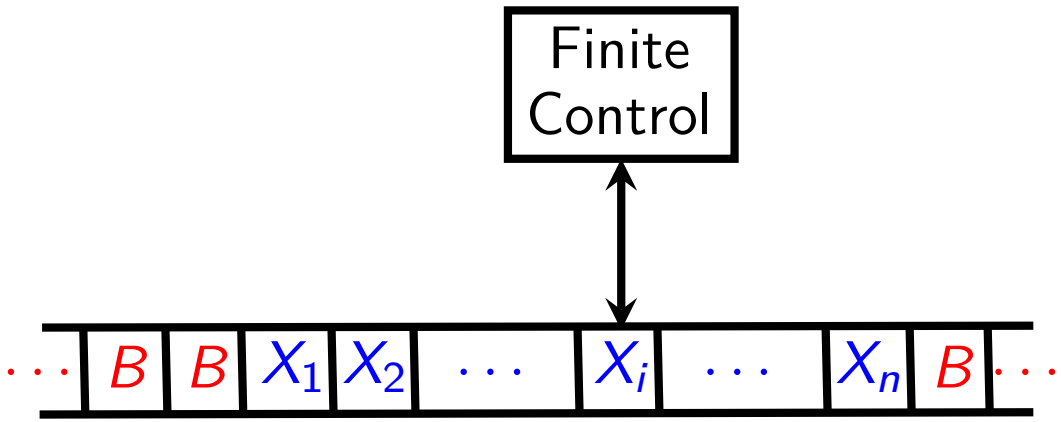
Pumping Lemma and Closure Properties

- We have seen the pumping lemma for context-free languages
- Using the pumping lemma, we have shown the existence of non-context-free languages.
- Thus we have proved that $CFL \subset 2^{\Sigma^*}$.
- We have seen the closure properties of context-free languages.
- We have discussed the decision properties of context-free languages.

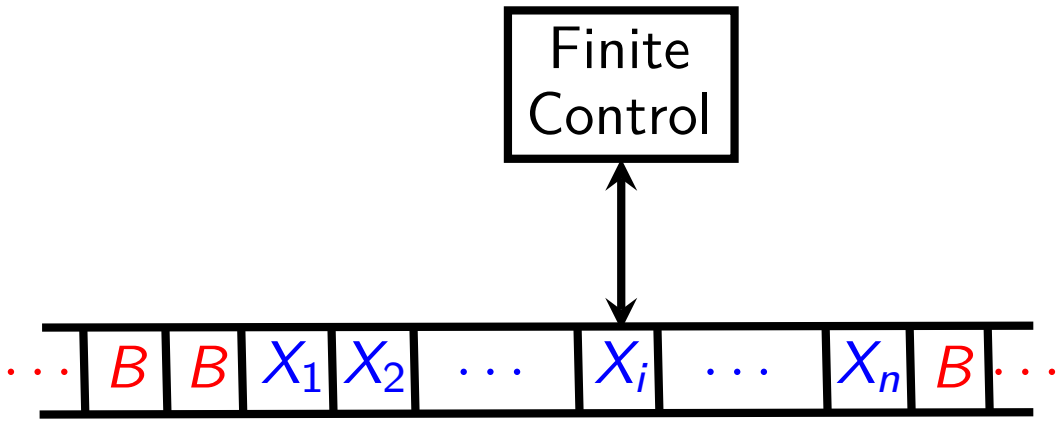
Pumping Lemma and Closure Properties

- We have seen the pumping lemma for context-free languages
- Using the pumping lemma, we have shown the existence of non-context-free languages.
- Thus we have proved that $CFL \subset 2^{\Sigma^*}$.
- We have seen the closure properties of context-free languages.
- We have discussed the decision properties of context-free languages.
- Any Questions?

Informal Introduction

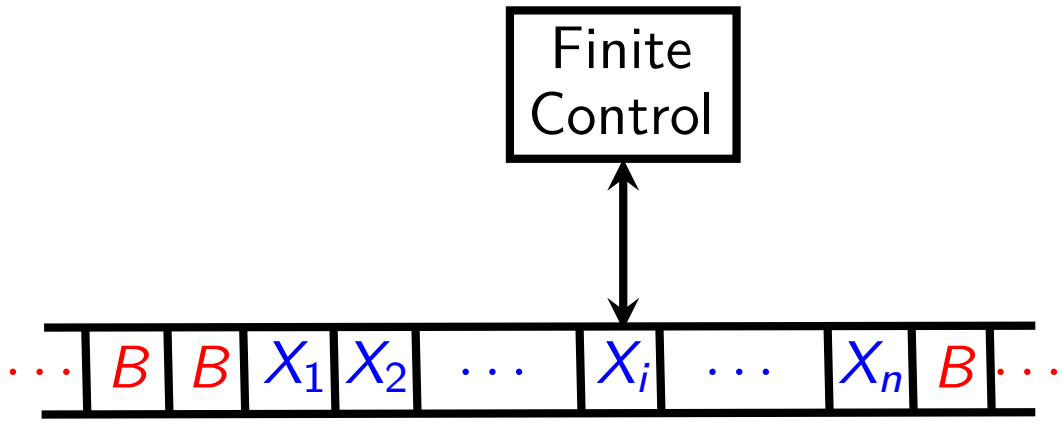


Informal Introduction



- A Turing Machine (TM) can be visualized as a finite control, which is in one of its finite set of possible states, and capable of moving left or right, reading from and writing to memory cells of an infinite tape.

Informal Introduction



- A Turing Machine (TM) can be visualized as a finite control, which is in one of its finite set of possible states, and capable of moving left or right, reading from and writing to memory cells of an infinite tape.
- All the cells in the tape contain a special blank symbol B.

Informal Introduction

- When the machine starts, some input string $X_1X_2 \cdots X_n$ is available on the tape and the finite control points to the cell containing the first symbol X_1. We say that the machine is scanning that cell. The machine starts in its designated start state.

Informal Introduction

- When the machine starts, some input string $X_1 X_2 \cdots X_n$ is available on the tape and the finite control points to the cell containing the first symbol X_1. We say that the machine is scanning that cell. The machine starts in its designated start state.
- Depending the current state, and the current symbol read from the tape, the machine does the following:

Informal Introduction

- When the machine starts, some input string $X_1X_2 \cdots X_n$ is available on the tape and the finite control points to the cell containing the first symbol X_1. We say that the machine is scanning that cell. The machine starts in its designated start state.
- Depending the current state, and the current symbol read from the tape, the machine does the following:
  - Changes its state, which can be same as the current state,

Informal Introduction

- When the machine starts, some input string $X_1 X_2 \cdots X_n$ is available on the tape and the finite control points to the cell containing the first symbol X_1. We say that the machine is scanning that cell. The machine starts in its designated start state.
- Depending the current state, and the current symbol read from the tape, the machine does the following:
  - Changes its state, which can be same as the current state,
  - Write a tape symbol to the current cell being scanned, and

Informal Introduction

- When the machine starts, some input string $X_1 X_2 \cdots X_n$ is available on the tape and the finite control points to the cell containing the first symbol X_1. We say that the machine is scanning that cell. The machine starts in its designated start state.
- Depending the current state, and the current symbol read from the tape, the machine does the following:
  - Changes its state, which can be same as the current state,
  - Write a tape symbol to the current cell being scanned, and
  - Move the finite control to the left or right.

Informal Introduction

- When the machine starts, some input string $X_1 X_2 \cdots X_n$ is available on the tape and the finite control points to the cell containing the first symbol X_1. We say that the machine is scanning that cell. The machine starts in its designated start state.
- Depending the current state, and the current symbol read from the tape, the machine does the following:
  - Changes its state, which can be same as the current state,
  - Write a tape symbol to the current cell being scanned, and
  - Move the finite control to the left or right.
- A "step" of operation of the machine as described above is prescribed by its "program" given by a transition function.

Formal Definition

- A Turing Machine (TM) is formally defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

Formal Definition

- A Turing Machine (TM) is formally defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

- Q is a finite set of **states** of the finite control

Formal Definition

- A Turing Machine (TM) is formally defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

- Q is a finite set of **states** of the finite control
- Σ is the **alphabet** (finite set of **input symbols**)

Formal Definition

- A Turing Machine (TM) is formally defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

- Q is a finite set of **states** of the finite control
- Σ is the **alphabet** (finite set of **input symbols**)
- Γ is the finite set of **tape symbols** that includes all the symbols of Σ. That is, $\Sigma \subset \Gamma$.

Formal Definition

- A Turing Machine (TM) is formally defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

- Q is a finite set of **states** of the finite control
- Σ is the **alphabet** (finite set of **input symbols**)
- Γ is the finite set of **tape symbols** that includes all the symbols of Σ. That is, $\Sigma \subset \Gamma$.
- $q_0 \in Q$ is the **start or initial state** of the machine

Formal Definition

- A Turing Machine (TM) is formally defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

- Q is a finite set of **states** of the finite control
- Σ is the **alphabet** (finite set of **input symbols**)
- Γ is the finite set of **tape symbols** that includes all the symbols of Σ. That is, $\Sigma \subset \Gamma$.
- $q_0 \in Q$ is the **start or initial state** of the machine
- $B \in (\Gamma - \Sigma)$ is the **blank tape symbol**

Formal Definition

- A Turing Machine (TM) is formally defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

- Q is a finite set of **states** of the finite control
- Σ is the **alphabet** (finite set of **input symbols**)
- Γ is the finite set of **tape symbols** that includes all the symbols of Σ. That is, $\Sigma \subset \Gamma$.
- $q_0 \in Q$ is the **start or initial state** of the machine
- $B \in (\Gamma - \Sigma)$ is the **blank tape symbol**
- $F \subseteq Q$ is the finite set of **accepting or final states**.

Formal Definition

- A Turing Machine (TM) is formally defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

- Q is a finite set of **states** of the finite control
- Σ is the **alphabet** (finite set of **input symbols**)
- Γ is the finite set of **tape symbols** that includes all the symbols of Σ. That is, $\Sigma \subset \Gamma$.
- $q_0 \in Q$ is the **start or initial state** of the machine
- $B \in (\Gamma - \Sigma)$ is the **blank tape symbol**
- $F \subseteq Q$ is the finite set of **accepting or final states**.
- $\delta : (Q\ X\ \Gamma) \rightarrow (Q\ X\ \Gamma\ X\ D)$ is the **transition function**, where D is the set of directions $\{L, R\}$.

Instantaneous Description

- A snapshot of a TM is captured by its **instantaneous description** given by

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n$$

Instantaneous Description

- A snapshot of a TM is captured by its **instantaneous description** given by

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n$$

- Here q is the current state, the machine is scanning the input symbol X_i, and X_1 to X_n gives a snapshot of the tape contents.

Instantaneous Description

- A snapshot of a TM is captured by its **instantaneous description** given by

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n$$

- Here q is the current state, the machine is scanning the input symbol X_i, and X_1 to X_n gives a snapshot of the tape contents.
- Suppose $\delta(q, X_i) = (p, Y, L)$. Then the move of the machine is captured as

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash X_1 X_2 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n$$

Instantaneous Description

- A snapshot of a TM is captured by its **instantaneous description** given by

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n$$

- Here q is the current state, the machine is scanning the input symbol X_i, and X_1 to X_n gives a snapshot of the tape contents.
- Suppose $\delta(q, X_i) = (p, Y, L)$. Then the move of the machine is captured as

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash X_1 X_2 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n$$

- As we have seen before, we can capture multiple moves of the machine $\vdash^*$ (zero or more moves) as reflective and transitive closure of $\vdash$.

An Example

- Let us design a TM to accept all the palindromes over the alphabet $\{0, 1\}$.

An Example

- Let us design a TM to accept all the palindromes over the alphabet $\{0, 1\}$.
- The general idea is as follows: Read the first (current) symbol and remember the same (how?).

An Example

- Let us design a TM to accept all the palindromes over the alphabet $\{0, 1\}$.
- The general idea is as follows: Read the first (current) symbol and remember the same (how?). Move to the last symbol (how?) and check if it is same as the remembered symbol.

An Example

- Let us design a TM to accept all the palindromes over the alphabet $\{0, 1\}$.
- The general idea is as follows: Read the first (current) symbol and remember the same (how?). Move to the last symbol (how?) and check if it is same as the remembered symbol. If not same, halt. Else, move left to the second (next) input symbol (how?) and repeat the process.

An Example

- Let us design a TM to accept all the palindromes over the alphabet $\{0, 1\}$.
- The general idea is as follows: Read the first (current) symbol and remember the same (how?). Move to the last symbol (how?) and check if it is same as the remembered symbol. If not same, halt. Else, move left to the second (next) input symbol (how?) and repeat the process.
- This iteration terminates when there are no more input symbols when we move left to find a new symbol (even length palindromes) or when we move right to find a matching symbol (odd length palindromes).

An Example

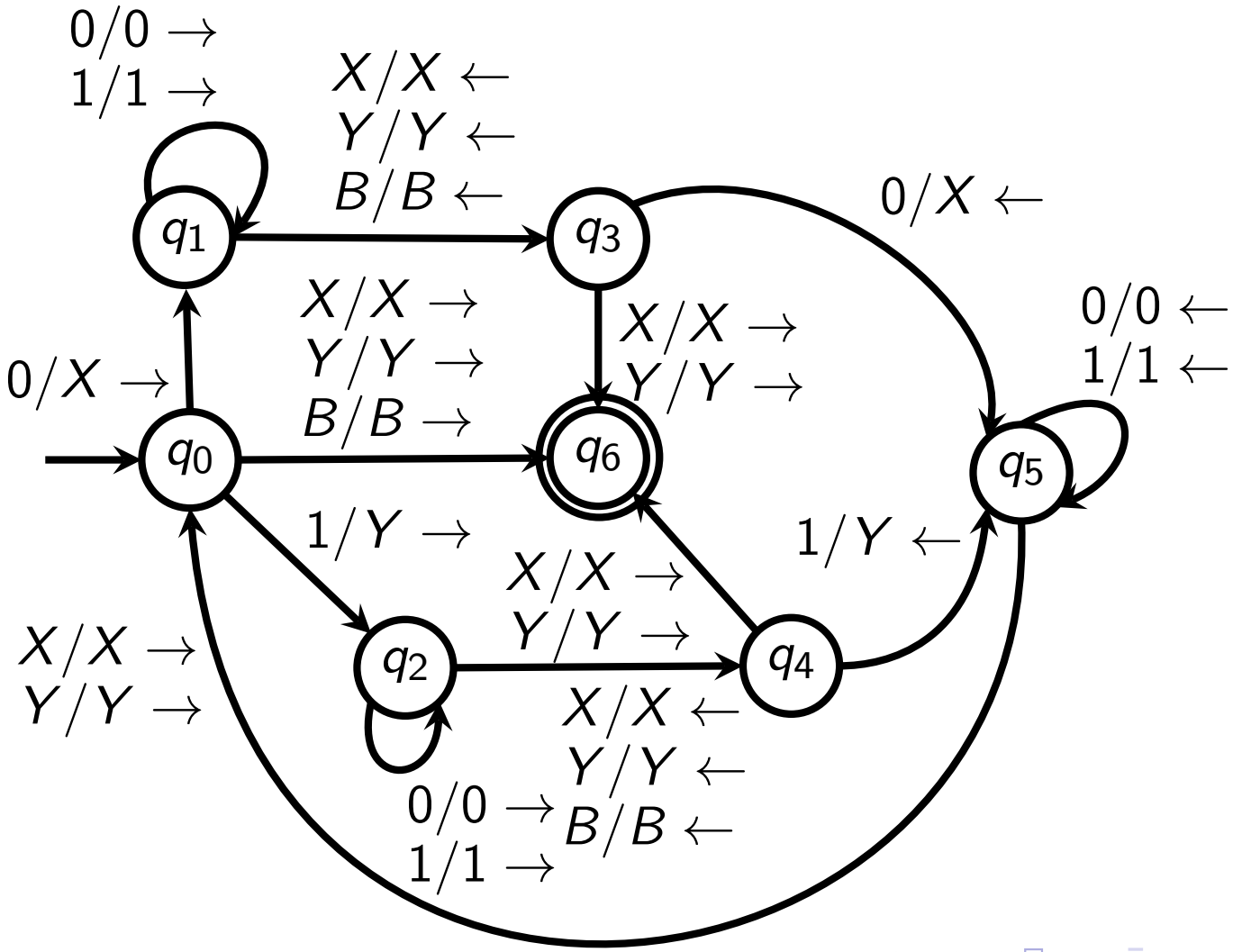
- Let us design a TM to accept all the palindromes over the alphabet $\{0, 1\}$.
- The general idea is as follows: Read the first (current) symbol and remember the same (how?). Move to the last symbol (how?) and check if it is same as the remembered symbol. If not same, halt. Else, move left to the second (next) input symbol (how?) and repeat the process.
- This iteration terminates when there are no more input symbols when we move left to find a new symbol (even length palindromes) or when we move right to find a matching symbol (odd length palindromes).
- Note that ϵ needs to be accepted. So, the machine halts in a final state in the beginning itself if the first symbol read is a blank B.

Transition Table

- Based on this idea, we can write the following "program".

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)	(q_2, Y, R)	(q_6, X, R)	(q_6, Y, R)	(q_6, B, R)
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_3, X, L)	(q_3, Y, L)	(q_3, B, L)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_4, X, L)	(q_4, Y, L)	(q_4, B, L)
q_3	(q_5, X, L)	—	(q_6, X, R)	(q_6, Y, R)	—
q_4	—	(q_5, Y, L)	(q_6, X, R)	(q_6, Y, R)	—
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	(q_0, X, R)	(q_0, Y, R)	—
$*q_6$	—	—	—	—	—

Transition Diagrams



Running the Turing machine

- Let us run our TM on the input 1001.

  $q_0 1001$

Running the Turing machine

- Let us run our TM on the input 1001.

$q_0 1001$
$\vdash Y q_2 001$

Running the Turing machine

- Let us run our TM on the input 1001.

$q_0 1001$
$\vdash Y q_2 001$
$\vdash Y 0 q_2 01 \vdash Y 00 q_2 1 \vdash Y 001 q_2 B$

Running the Turing machine

- Let us run our TM on the input 1001.

$q_0 1001$
$\vdash Y q_2 001$
$\vdash Y 0 q_2 01 \vdash Y 00 q_2 1 \vdash Y 001 q_2 B$
$\vdash Y 00 q_4 1 B$

Running the Turing machine

- Let us run our TM on the input 1001.

$q_0 1001$
$\vdash Y q_2 001$
$\vdash Y 0 q_2 01 \vdash Y 00 q_2 1 \vdash Y 001 q_2 B$
$\vdash Y 00 q_4 1 B$
$\vdash Y 0 q_5 0 Y \vdash Y q_5 00 Y \vdash q_5 Y 00 Y \vdash Y q_0 00 Y$

Running the Turing machine

- Let us run our TM on the input 1001.

$q_0 1001$
$\vdash Y q_2 001$
$\vdash Y 0 q_2 01 \vdash Y 00 q_2 1 \vdash Y 001 q_2 B$
$\vdash Y 00 q_4 1 B$
$\vdash Y 0 q_5 0 Y \vdash Y q_5 00 Y \vdash q_5 Y 00 Y \vdash Y q_0 00 Y$
$\vdash Y X q_1 0 Y \vdash Y X 0 q_1 Y \vdash Y X q_3 0 Y$

Running the Turing machine

- Let us run our TM on the input 1001.

$q_0 1001$
$\vdash Y q_2 001$
$\vdash Y 0 q_2 01 \vdash Y 00 q_2 1 \vdash Y 001 q_2 B$
$\vdash Y 00 q_4 1 B$
$\vdash Y 0 q_5 0 Y \vdash Y q_5 00 Y \vdash q_5 Y 00 Y \vdash Y q_0 00 Y$
$\vdash Y X q_1 0 Y \vdash Y X 0 q_1 Y \vdash Y X q_3 0 Y$
$\vdash Y q_5 X X Y \vdash Y X q_0 X Y$

Running the Turing machine

- Let us run our TM on the input 1001.

$q_0 1001$
$\vdash Y q_2 001$
$\vdash Y 0 q_2 01 \vdash Y 00 q_2 1 \vdash Y 001 q_2 B$
$\vdash Y 00 q_4 1 B$
$\vdash Y 0 q_5 0 Y \vdash Y q_5 00 Y \vdash q_5 Y 00 Y \vdash Y q_0 00 Y$
$\vdash Y X q_1 0 Y \vdash Y X 0 q_1 Y \vdash Y X q_3 0 Y$
$\vdash Y q_5 X X Y \vdash Y X q_0 X Y$
$\vdash Y X X q_6 Y$

Running the Turing machine

- Let us run our TM on the input 1001.

$q_0 1001$
$\vdash Y q_2 001$
$\vdash Y 0 q_2 01 \vdash Y 00 q_2 1 \vdash Y 001 q_2 B$
$\vdash Y 00 q_4 1 B$
$\vdash Y 0 q_5 0 Y \vdash Y q_5 00 Y \vdash q_5 Y 00 Y \vdash Y q_0 00 Y$
$\vdash Y X q_1 0 Y \vdash Y X 0 q_1 Y \vdash Y X q_3 0 Y$
$\vdash Y q_5 X X Y \vdash Y X q_0 X Y$
$\vdash Y X X q_6 Y$

- Since the machine has entered an accepting state, the input string 1001 is accepted. Note that in our example, the machine halts at the accepting state since no moves are defined for the accepting state q_6.

Running the Turing machine

- Let us trace the behaviour of our TM on the input 1011.

  $q_0 1011$

Running the Turing machine

- Let us trace the behaviour of our TM on the input 1011.

  $q_0 1011$
  $\vdash Y q_2 011$

Running the Turing machine

- Let us trace the behaviour of our TM on the input 1011.

$q_0 1011$
$\vdash Y q_2 011$
$\vdash Y 0 q_2 11 \vdash Y 01 q_2 1 \vdash Y 011 q_2 B$

Running the Turing machine

- Let us trace the behaviour of our TM on the input 1011.

$q_0 1011$
$\vdash Y q_2 011$
$\vdash Y 0 q_2 11 \vdash Y 01 q_2 1 \vdash Y 011 q_2 B$
$\vdash Y 01 q_4 1 B$

Running the Turing machine

- Let us trace the behaviour of our TM on the input 1011.

$q_0 1011$
$\vdash Y q_2 011$
$\vdash Y 0 q_2 11 \vdash Y 01 q_2 1 \vdash Y 011 q_2 B$
$\vdash Y 01 q_4 1 B$
$\vdash Y 0 q_5 1 Y \vdash Y q_5 01 Y \vdash q_5 Y 01 Y \vdash Y q_0 01 Y$

Running the Turing machine

- Let us trace the behaviour of our TM on the input 1011.

$q_0 1011$
$\vdash Y q_2 011$
$\vdash Y 0 q_2 11 \vdash Y 01 q_2 1 \vdash Y 011 q_2 B$
$\vdash Y 01 q_4 1 B$
$\vdash Y 0 q_5 1 Y \vdash Y q_5 01 Y \vdash q_5 Y 01 Y \vdash Y q_0 01 Y$
$\vdash Y X q_1 1 Y \vdash Y X 1 q_1 Y \vdash Y X q_3 1 Y$

Running the Turing machine

- Let us trace the behaviour of our TM on the input 1011.

  $q_0 1011$
  $\vdash Y q_2 011$
  $\vdash Y 0 q_2 11 \vdash Y 01 q_2 1 \vdash Y 011 q_2 B$
  $\vdash Y 01 q_4 1 B$
  $\vdash Y 0 q_5 1 Y \vdash Y q_5 01 Y \vdash q_5 Y 01 Y \vdash Y q_0 01 Y$
  $\vdash Y X q_1 1 Y \vdash Y X 1 q_1 Y \vdash Y X q_3 1 Y$

- Since $\delta(q_3, 1)$ is not defined, the machine halts at q_3. Since q_3 is not an accepting state, the input string 1011 is rejected.

Another Example

- Let us look at another example for constructing a Turing machine: Construct a TM to accept the language $\left\{a^n b^n \mid n \geq 1\right\}$.

Another Example

- Let us look at another example for constructing a Turing machine: Construct a TM to accept the language $\left\{a^n b^n \mid n \geq 1\right\}$.

	a	b	X	Y	B
$\rightarrow q_0$	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	(q_1, a, R)	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	(q_2, a, L)	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
$*q_4$	—	—	—	—	—

Computing integer-valued functions

- Turing originally studied these machines in the context of computation of integer-valued functions

Computing integer-valued functions

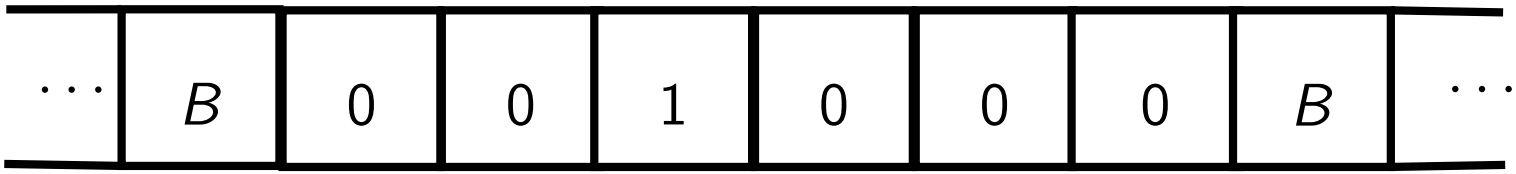
- Turing originally studied these machines in the context of computation of integer-valued functions
- An integer can be represented in unary notation using only one symbol — 0^n to denote the integer n. Note that 0 can be represented by ϵ.

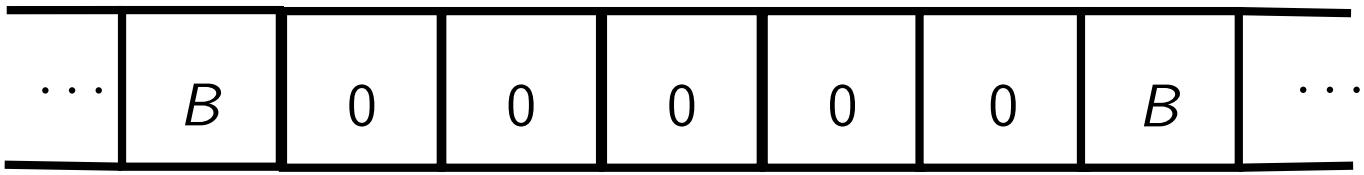
Computing integer-valued functions

- Turing originally studied these machines in the context of computation of integer-valued functions
- An integer can be represented in unary notation using only one symbol — 0^n to denote the integer n. Note that 0 can be represented by ϵ.
- A machine can be designed to read the arguments from the tape (separated by a separator symbol 1), and leave the answer on the tape when it halts.

Computing integer-valued functions

- Turing originally studied these machines in the context of computation of integer-valued functions
- An integer can be represented in unary notation using only one symbol — 0^n to denote the integer n. Note that 0 can be represented by ϵ.
- A machine can be designed to read the arguments from the tape (separated by a separator symbol 1), and leave the answer on the tape when it halts.
- For example, we can let a TM start with 001000 on the tape and halt with 00000 to realize the addition function.





Example of integer computation

- Let us construct a TM to perform $\dot{-}$, the proper subtraction over integers, defined by $m \dot{-} n = max(m - n, 0)$. That is, $m \dot{-} n = m - n$ if $m > n$ and $m \dot{-} n = 0$ if $m \leq n$.

Example of integer computation

- Let us construct a TM to perform $\dot{-}$, the proper subtraction over integers, defined by $m \dot{-} n = max(m - n, 0)$. That is, $m \dot{-} n = m - n$ if $m > n$ and $m \dot{-} n = 0$ if $m \leq n$.
- The general idea is as follows: Replace the first 0 by a blank. Move right, cross the separator, to find a 0 in the second argument. Mark it read by changing it to 1. Move back left to read the next 0 of the first argument and repeat.

Example of integer computation

- Let us construct a TM to perform $\dot{-}$, the proper subtraction over integers, defined by $m \dot{-} n = max(m - n, 0)$. That is, $m \dot{-} n = m - n$ if $m > n$ and $m \dot{-} n = 0$ if $m \leq n$.
- The general idea is as follows: Replace the first 0 by a blank. Move right, cross the separator, to find a 0 in the second argument. Mark it read by changing it to 1. Move back left to read the next 0 of the first argument and repeat.
- This iteration is terminated when
  - Searching for a 0 in the second argument, a blank B is seen. This arises when $m > n$. Now the machine comes back, changes the first 1 to a 0 and erases the remaining 1's by changing them to B's.

Example of integer computation

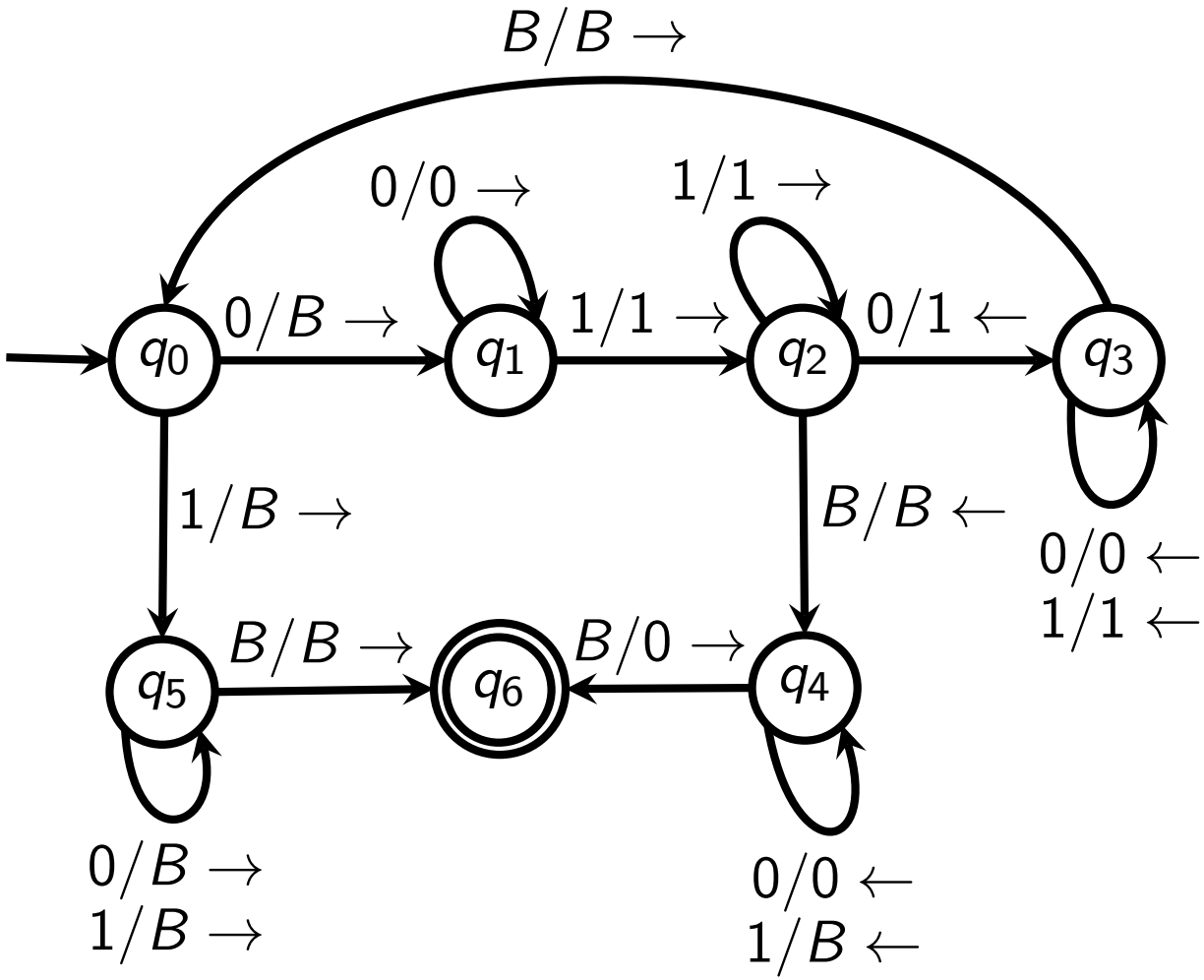
- Let us construct a TM to perform $\dot{-}$, the proper subtraction over integers, defined by $m \dot{-} n = max(m - n, 0)$. That is, $m \dot{-} n = m - n$ if $m > n$ and $m \dot{-} n = 0$ if $m \leq n$.
- The general idea is as follows: Replace the first 0 by a blank. Move right, cross the separator, to find a 0 in the second argument. Mark it read by changing it to 1. Move back left to read the next 0 of the first argument and repeat.
- This iteration is terminated when
  - Searching for a 0 in the second argument, a blank B is seen. This arises when $m > n$. Now the machine comes back, changes the first 1 to a 0 and erases the remaining 1's by changing them to B's.
  - At the beginning of the iteration, the machine can not find any more 0 in the first argument. This arises when $m \leq n$. The machine blanks out all the 1's and 0's to halt with the blank tape.

Transition Table

- Following is a program for proper subtraction.

	0	1	B
$\rightarrow q_0$	(q_1, B, R)	(q_5, B, R)	—
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	—
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
$*q_6$	—	—	—

Transition Diagram



Language of a TM

- When a Turing machine M starts in q_0 scanning the first symbol of w on the tape and eventually enters a final state, then w is accepted by M. Otherwise, w is not accepted.

Language of a TM

- When a Turing machine M starts in q_0 scanning the first symbol of w on the tape and eventually enters a final state, then w is accepted by M. Otherwise, w is not accepted.
- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine. Then $Lang(M)$, the language of the machine M is defined as follows:

$$Lang(M) = \left\{ w \mid w \in \Sigma^* \text{ and } q_0 w \vdash^* \alpha p \beta \text{ for some } p \in F \text{ and } \alpha, \beta \in \Gamma \right.$$

Language of a TM

- When a Turing machine M starts in q_0 scanning the first symbol of w on the tape and eventually enters a final state, then w is accepted by M. Otherwise, w is not accepted.
- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine. Then $Lang(M)$, the language of the machine M is defined as follows:

$$Lang(M) = \left\{ w \mid w \in \Sigma^* \text{ and } q_0 w \vdash^* \alpha p \beta \text{ for some } p \in F \text{ and } \alpha, \beta \in \Gamma \right.$$

- Note that there is no condition that the machine has seen all the input symbols. The machine may accept w just by seeing a prefix of it.

Language of a TM

- When a Turing machine M starts in q_0 scanning the first symbol of w on the tape and eventually enters a final state, then w is accepted by M. Otherwise, w is not accepted.
- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine. Then $Lang(M)$, the language of the machine M is defined as follows:

$$Lang(M) = \left\{ w \mid w \in \Sigma^* \text{ and } q_0 w \vdash^* \alpha p \beta \text{ for some } p \in F \text{ and } \alpha, \beta \in \Gamma \right.$$

- Note that there is no condition that the machine has seen all the input symbols. The machine may accept w just by seeing a prefix of it.
- The class of languages described by Turing machines is termed as **recursively enumerable languages**.

$$RE = \left\{ L \mid \text{ there exists a TM } M \text{ such that } L = Lang(M) \right\}$$

Halting of TM

- A Turing machine M halts when it enters a state for which no transition is defined.

Halting of TM

- A Turing machine M halts when it enters a state for which no transition is defined.
- Usually, M is designed with one single final state, entering which the machine halts. So, without loss of generality, we can assume that M always halts when the input is accepted.

Halting of TM

- A Turing machine M halts when it enters a state for which no transition is defined.
- Usually, M is designed with one single final state, entering which the machine halts. So, without loss of generality, we can assume that M always halts when the input is accepted.
- However, it is not possible to ensure that M always halts when it is not accepting the input string.

Halting of TM

- A Turing machine M halts when it enters a state for which no transition is defined.
- Usually, M is designed with one single final state, entering which the machine halts. So, without loss of generality, we can assume that M always halts when the input is accepted.
- However, it is not possible to ensure that M always halts when it is not accepting the input string.
- Suppose we insist that a Turing machine must halt on all inputs. This is important because such machines stand for "algorithms".

Halting of TM

- A Turing machine M halts when it enters a state for which no transition is defined.
- Usually, M is designed with one single final state, entering which the machine halts. So, without loss of generality, we can assume that M always halts when the input is accepted.
- However, it is not possible to ensure that M always halts when it is not accepting the input string.
- Suppose we insist that a Turing machine must halt on all inputs. This is important because such machines stand for "algorithms". Languages of such always-halting TM's are referred to as **recursive languages**.

$$RC = \{L \mid \text{ there exists TM } M \text{ that halts on all inputs and } L = Lang(M$$

Languages of TM's

- It can be easily shown that a TM can be designed for every context-free grammar.

Languages of TM's

- It can be easily shown that a TM can be designed for every context-free grammar.
- It is also possible to show that TM's are more powerful than PDA's. For example, a TM can be designed to accept the language $\left\{a^n b^n c^n \mid n \geq 1\right\}$.

Languages of TM's

- It can be easily shown that a TM can be designed for every context-free grammar.
- It is also possible to show that TM's are more powerful than PDA's. For example, a TM can be designed to accept the language $\left\{a^n b^n c^n \mid n \geq 1\right\}$.
- Thus we have

$$R \subset DCFL \subset CFL \subset RE$$

Languages of TM's

- It can be easily shown that a TM can be designed for every context-free grammar.
- It is also possible to show that TM's are more powerful than PDA's. For example, a TM can be designed to accept the language $\left\{a^n b^n c^n \mid n \geq 1\right\}$.
- Thus we have

$$R \subset DCFL \subset CFL \subset RE$$

- It remains to be seen if $RC = RE$???

Languages of TM's

- It can be easily shown that a TM can be designed for every context-free grammar.
- It is also possible to show that TM's are more powerful than PDA's. For example, a TM can be designed to accept the language $\left\{a^n b^n c^n \mid n \geq 1\right\}$.
- Thus we have

$$R \subset DCFL \subset CFL \subset RE$$

- It remains to be seen if $RC = RE$???
- And of course the big question: Is $RE = 2^{\Sigma^*}$???

Exercises

8.2.1 Show the ID's of the palindrome checking TM if the input tape contains:

- 00
- 100101
- 01110

8.2.2 Design TMs for the following languages:

- The set of strings with an equal number of 0's and 1's
- $\left\{a^n b^n c^n \mid n \geq 1\right\}$
- The set of all strings of balanced parentheses.

Exercises

8.2.5 Consider the TM

$$M = \left(\{q_0, q_1, q_2, q_3, q_f\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_f\}\right)$$

Informally but clearly describe the language $L(M)$ if δ consists of the following sets of rules:

- $\delta(q_0, 0) = (q_1, 1, R); \delta(q_1, 1) = (q_0, 0, R); \delta(q_1, B) = (q_f, B, R)$.
- $\delta(q_0, 0) = (q_0, B, R); \delta(q_0, 1) = (q_1, B, R); \delta(q_1, 1) = (q_1, B, R); \delta(q_1, B) = (q_f, B, R)$.
- $\delta(q_0, 0) = (q_1, 1, R); \delta(q_1, 1) = (q_2, 0, L); \delta(q_2, 1) = (q_0, 1, R); \delta(q_1, B) = (q_f, B, R)$.

Summary

- We have seen an informal introduction to Turing machines and their formal definitions.
- We have seen some examples of constructing Turing machines.
- Turing machines can be seen as computing integer functions.
- We have seen an example of integer computing with Turing machines.
- We have seen the definition of language of a Turing machine — recursively enumerable languages
- Recursive languges are those for which Turing machines that halt on all inputs can be constructed.
- It remains to be seen if $RC \subset RE$?? and $RE \subset 2^{\Sigma^*}$???

What next?

- Review regular expressions and finite automata
- Review context-free grammars and pushdown automata
- Review pumping lemmas and their applications
- Review closure and decision properties of regular and context-free languages
- Review the Turing Machines
- Work out the exercises given during the lectures!
- Review the basics of complexity analysis of algorithms
- In the next class, we will continue our discussions on Turing Machines