

CS244: THEORY OF COMPUTATION

Fu Song
ShanghaiTech University

Fall 2020

- ▶ Finite state automata: computation models for devices that have a small amount (finite) of memory
- ▶ Pushdown automata: computations models for devices that use memory in last-in-first-out manner
- ▶ They are too restricted to serve as models for general purpose computers

PART TWO.
COMPUTABILITY THEORY

Outline

The Church – Turing Thesis

- Turing machines

 - Examples of Turing machines

- Variants of Turing machines

 - Multitape Turing Machines

 - Nondeterministic Turing Machines

 - Enumerators

- Relation with Recursively Enumerable Grammar and TM

- The Definition of Algorithm

 - Hilbert's Problems

- Context-Sensitive Grammar and Linear-bounded TM

Outline

The Church – Turing Thesis

Turing machines

- Examples of Turing machines

Variants of Turing machines

- Multitape Turing Machines

- Nondeterministic Turing Machines

- Enumerators

Relation with Recursively Enumerable Grammar and TM

The Definition of Algorithm

- Hilbert's Problems

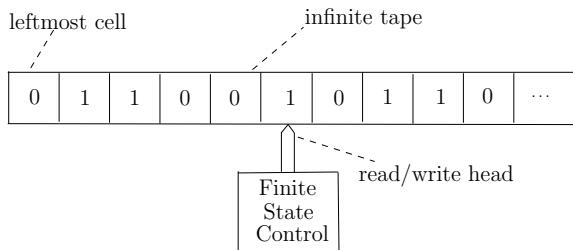
Context-Sensitive Grammar and Linear-bounded TM

Turing machines

Alan Turing in 1936 proposed Turing machines M :

- ▶ M uses an **infinite** tape as its **unlimited memory**, with a tape head **reading and writing symbols and moving around** on the tape.
The tape **initially** contains **only the input string** and is **blank everywhere else**.
- ▶ If M needs to store information, it may **write** this information on the tape. To **read** the information that it has written, M can move its head back over it.
- ▶ M continues computing until it decides to produce an output. The outputs **accept** and **reject** are obtained by entering designated accepting and rejecting states.
- ▶ If M doesn't enter an **accepting** or a **rejecting** state, it will go on forever, never **halting**.

Schematic of a Turing machine



Informally, a Turing machine consists of
an **infinite** tape
with a **read/write** head
controlled by
a **finite state** device.

The difference between finite automata and Turing machines

1. A Turing machine can both **write** on the tape and **read** from it.
2. The read-write head can **move** both to the left and to the right.
3. The tape is **infinite**.
4. The special states for rejecting and accepting take effect **immediately**.

$$B = \{ w\#w \mid w \in \{0,1\}^* \}$$

M_1 on input string w :

1. **Zig-zag across** the tape to corresponding positions on either side of the $\#$ symbol to check whether these positions contain the **same** symbol.
2. If they do **not**, or if **no $\#$** is found, **reject**. Cross off symbols as they are checked to keep track of which symbols correspond.
3. When all symbols to the left of the $\#$ have been **crossed off**, check for any remaining symbols to the right of the $\#$. If any symbols remain, **reject**; otherwise, **accept**.

$$B = \{w\#w \mid w \in \{0, 1\}^*\} \text{ (cont'd)}$$

Diagram illustrating the step-by-step construction of a string in a Turing machine. The string is built row by row, with arrows indicating the head's position at each step. The string grows from left to right, starting with '0' and ending with 'x' and a blank symbol '□'. The final string is '0 1 1 0 0 0 # 0 1 1 0 0 0 □ ...'.

```

0 1 1 0 0 0 # 0 1 1 0 0 0 □ ...
x 1 1 0 0 0 # 0 1 1 0 0 0 □ ...
x 1 1 0 0 0 # x 1 1 0 0 0 □ ...
x 1 1 0 0 0 # x 1 1 0 0 0 □ ...
x x 1 0 0 0 # x 1 1 0 0 0 □ ...
x x x x x x # x x x x x x □ ...

```

accept

Formal definition of a Turing machine

Definition

A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where

1. Q is finite set of states,
2. Σ is a finite nonempty input alphabet not containing the blank symbol \sqcup ,
3. Γ is finite nonempty tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Computation by M

- ▶ Initially, M receives its input $w = w_1 w_2 \dots w_n \in \Sigma^*$ on the leftmost n squares of the tape, and the rest of the tape is blank (i.e., filled \sqcup).
- ▶ The head starts on the leftmost square of the tape.
- ▶ As Σ does not contain \sqcup , so the first blank appearing on the tape marks the end of the input.
- ▶ Once M has started, the computation proceeds according to the rules described by the transition function.
- ▶ If M ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function indicates L .
- ▶ The computation continues until it enters either the accept or reject states, at which point it halts. If neither occurs, M goes on forever.

Configurations

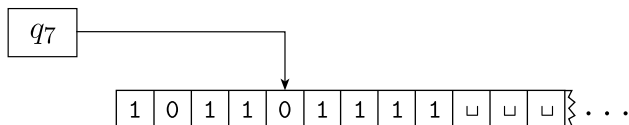
A **configuration** of a Turing machine consists of

- ▶ the current **state**,
- ▶ the current **tape contents**, and
- ▶ the current **head location**.

By $u q v$ we mean the configuration where

- ▶ the current **state** is q ,
- ▶ the current **tape contents** is uv , and
- ▶ the current **head location** is the **leftmost symbol** of v .
- ▶ The tape contains only **blanks** following the rightmost symbol of v .

Configurations (cont'd)



A Turing machine with configuration **1011** q_7 **01111**

Formal definition of computation

Let $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, and $q_i, q_j \in Q$.

1. If $\delta(q_i, b) = (q_j, c, L)$, then

$ua q_i bv$ yields $u q_j acv$.

2. If $\delta(q_i, b) = (q_j, c, R)$, then

$ua q_i bv$ yields $uac q_j v$.

Special cases occur when the head is at one of the ends of the configuration:

1. For the **left-hand end**, the configuration $q_i bv$ yields $q_j cv$ if the transition is **left moving** (because we prevent the machine from going off the left-hand end of the tape), and it yields $c q_j v$ for the right-moving transition.
2. 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 the configuration.

Special configurations

- ▶ The **start configuration** of M on input w is the configuration q_0w .
- ▶ In an **accepting configuration**, the state of the configuration is q_{accept} .
- ▶ In a **rejecting configuration**, the state of the configuration is q_{reject} .
- ▶ Accepting and rejecting configurations are **halting configurations** and do not yield further configurations.

Formal definition of computation (cont'd)

M **accepts** w if there are sequence of configurations C_1, C_2, \dots, C_k such that

1. C_1 the start configuration of M on w .
2. Each C_i yields C_{i+1} , and
3. C_k is an accepting configuration.

The collection of strings that M accepts is the **language of M** , or the **language recognized by M** , denoted $L(M)$.

Definition

A language is **Turing-recognizable**, if some Turing machine recognizes it.

On an input, the machine M may **accept**, **reject**, or **loop**. By **loop** we mean that the machine simply does not halt.

If M always halt, then it is a **decider**. A decider that recognizes some language is said to **decide** that language.

Definition

A language is **Turing-decidable** or simply **decidable** if some Turing machine decides it.

- ▶ Turing-**recognizable** language = recursive **enumerable** language
- ▶ Turing-**decidable** language = recursive language

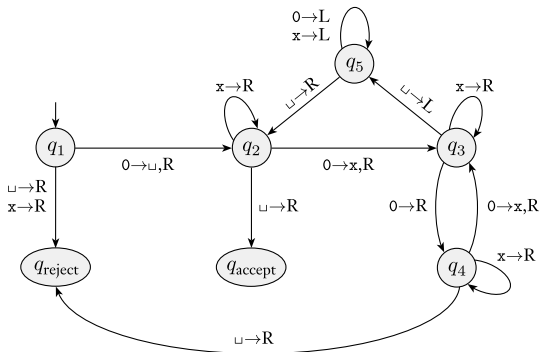
$$A = \{0^{2^n} \mid n \geq 0\}$$

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 to the left-hand end of the tape.
5. Go to stage 1.

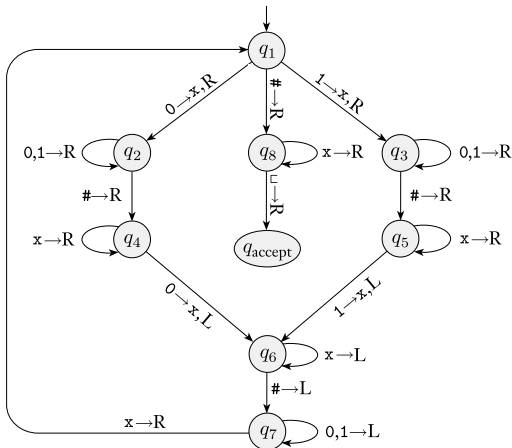
$$A = \{0^{2^n} \mid n \geq 0\} \text{ (cont'd)}$$

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\}$, where q_1 is the start state.
- $\Sigma = \{0\}$ and $\Gamma = \{0, x, \sqcup\}$.
- The transition function δ :



$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

- $Q = \{q_1, \dots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$, where q_1 is the start state.
- $\Sigma = \{0, 1, \#\}$ and $\Gamma = \{0, 1, \#, x, \sqcup\}$.
- The transition function δ :



$$C = \{ a^i b^j c^k \mid i, j, k \geq 1 \text{ and } i \times j = k \}$$

On input string w :

1. Scan the input from left to right to determine whether it is a member of $a^+b^+c^+$ and **reject** if it is **not**.
2. **Return** the head to the **left-hand end** of 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**.

$E =$

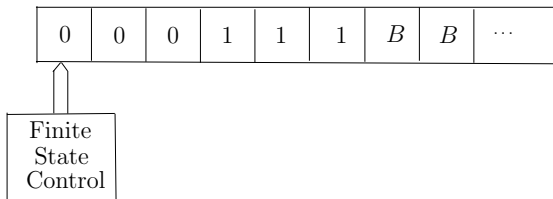
$\{\#x_1\#\cdots\#x_\ell \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$

On input string 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_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 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 the strings have been compared, so **accept**.
5. Go to stage 3.

Discussion

TM M accepting the language $\{0^n 1^n \mid n \geq 1\}$:



Outline

The Church – Turing Thesis

Turing machines

Examples of Turing machines

Variants of Turing machines

Multitape Turing Machines

Nondeterministic Turing Machines

Enumerators

Relation with Recursively Enumerable Grammar and TM

The Definition of Algorithm

Hilbert's Problems

Context-Sensitive Grammar and Linear-bounded TM

Variants of Turing Machines

Multitape Turing Machines

A multitape Turing machine M has several tapes:

1. Each tape has its own head for reading and writing.
2. The input is initially on **tape 1**, with all the other tapes being blank.
3. The transition function is

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, \textcolor{red}{S}\}^k,$$

where k is the number of tapes.

$$\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, R, \dots, L)$$

means that if M is in state q_i and heads 1 through k are reading symbols a_1 through a_k , the machine goes to state q_j , writes symbols b_1 through b_k , and directs each head to move left or right, or to **stay put**, as specified.

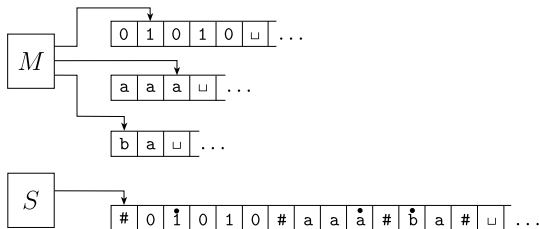
Theorem

Every multitape Turing machine has an equivalent single-tape Turing machine.

Proof (1)

We simulate an M with k tapes by a single-tape S .

- ▶ S uses $\#$ to separate the contents of the different tapes.
- ▶ S keeps track of the locations of the heads by writing a tape symbol with a dot above it to mark the place where the head on that tape would be.



Proof (2)

On input $w = w_1 \cdots w_n$:

1. First S puts its tape into the format that represents all k tapes of M :

$\# \dot{w}_1 w_2 \cdots w_n \# \sqcup \# \sqcup \# \cdots \#$.

2. To determine the symbols under the virtual heads, S scans its tape from the first $\#$, which marks the left-hand end, to the $(k + 1)$ st $\#$, which marks the right-hand end.
3. Then S makes a second pass to update the tapes according to the way that M 's transition function dictates.
4. If S moves one of the virtual heads to the right onto a $\#$, i.e., M has moved the corresponding head onto the previously unread blank portion of that tape. So S writes \sqcup on this tape cell and shifts the tape contents, from this cell until the rightmost $\#$, one unit to the right.
5. Go back to 2.

Corollary

A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

Discussion: Any alternative encoding?

Theorem

Every multitape Turing machine has an equivalent single-tape Turing machine.

Nondeterministic Turing Machines

1. The transition function for a nondeterministic Turing machine has the form

$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \{L, R\}).$$

2. The computation of a nondeterministic Turing machine is a **tree** whose branches correspond to different possibilities for the machine.
3. If **some branch of the computation leads to the accept state**, the machine accepts its input.

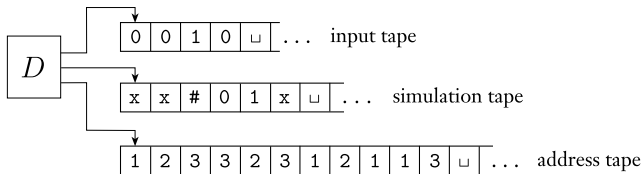
Theorem

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof (1)

We simulate a nondeterministic N by a deterministic D .

1. D try all possible branches of N 's nondeterministic computation.
2. If D ever finds the **accept state** on one of these branches, it **accepts**.
3. Otherwise, D 's simulation will **not** terminate.



Proof (2)

1. Initially, tape 1 contains the input w , and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2.
3. Use tape 2 to simulate N with input w on one branch of its nondeterministic computation. Before each step of N , consult the next symbol on tape 3 to determine which choice to make among those allowed by N 's transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, accept the input.
4. Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of N 's computation by going to stage 2.

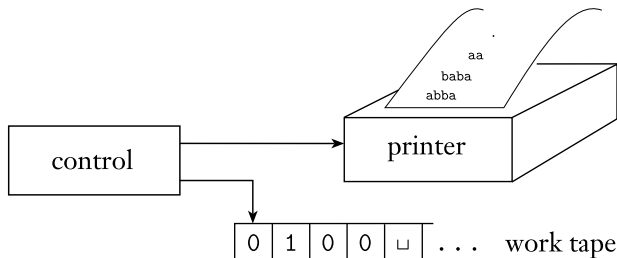
Corollary

A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.

Corollary

A language is decidable if and only if some nondeterministic Turing machine decides it.

Enumerators



1. An **enumerator** is a Turing machine with an attached **printer**.
2. The Turing machine can use that printer as an **output device** to print strings.
3. Every time the Turing machine wants to add a string to the list, it sends the string to the printer.

Theorem

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

Proof (1)

Theorem

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

(\Rightarrow) Let E be an enumerator E that enumerates a language A .

The desired M 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 , then **accept**.

Proof (2)

Theorem

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

(\Rightarrow) If M recognizes a language A , we can construct the following enumerator E for A . Let s_1, s_2, s_3, \dots , be a list of all possible strings in Σ^* .

1. Repeat the following for $i = 1, 2, 3, \dots$
2. Run M for i steps on each input, s_1, s_2, \dots, s_i .
3. If any computations accept, print out the corresponding s_j .

Outline

The Church – Turing Thesis

Turing machines

Examples of Turing machines

Variants of Turing machines

Multitape Turing Machines

Nondeterministic Turing Machines

Enumerators

Relation with Recursively Enumerable Grammar and TM

The Definition of Algorithm

Hilbert's Problems

Context-Sensitive Grammar and Linear-bounded TM

Theorem

Recursively Enumerable Grammar (Type-0) and TM are equivalent.

From TM to Recursively Enumerable Grammar

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. The Type-0 grammar G

- ▶ first nondet. generates a finite word

$$\begin{pmatrix} a_1 \\ a_1 \end{pmatrix} \cdots \begin{pmatrix} a_n \\ a_n \end{pmatrix} \begin{pmatrix} \varepsilon \\ \sqcup \end{pmatrix} \cdots \begin{pmatrix} \varepsilon \\ \sqcup \end{pmatrix},$$

with the intention that

- ▶ $a_1 \dots a_n$ is the input of M ,
 - ▶ the blanks in the second component denote the space used by M .
-
- ▶ then G simulates the computation of M over $a_1 \dots a_n$,
by rewriting the second components of the generated word.

From TM to Recursively Enumerable Grammar

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. The Type-0 grammar G

$$\begin{pmatrix} a_1 \\ a_1 \end{pmatrix} \cdots \begin{pmatrix} a_n \\ a_n \end{pmatrix} \begin{pmatrix} \varepsilon \\ \sqcup \end{pmatrix} \cdots \begin{pmatrix} \varepsilon \\ \sqcup \end{pmatrix},$$

Formally, $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ is defined as follows.

- ▶ $\mathcal{N} = Q \cup (\Sigma \cup \{\varepsilon\}) \times \Gamma \cup \{A_1, A_2, A_3\}$, $S = A_1$,
- ▶ \mathcal{P} includes the following rules,
 - ▶ $A_1 \rightarrow q_0 A_2$, $A_2 \rightarrow [a, a] A_2$, $A_2 \rightarrow A_3$, $A_3 \rightarrow [\varepsilon, \sqcup] A_3$, $A_3 \rightarrow \varepsilon$,
 - ▶ $q[a, X] \rightarrow [a, Y]p$
for each $a \in \Sigma \cup \{\varepsilon\}$ and $q, p, X, Y : \delta(q, X) = (p, Y, R)$,
 - ▶ $[a_1, Z]q[a_2, X] \rightarrow p[a_1, Z][a_2, Y]$
for each $a_1, a_2 \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, and
 $q, p, X, Y : \delta(q, X) = (p, Y, L)$,
 - ▶ $[a, X]q \rightarrow qaq$, $q[a, X] \rightarrow qaq$, $q \rightarrow \varepsilon$
for each $a \in \Sigma$, $X \in \Gamma$, and $q \in F$.

TM for $\{0^n 1^n \mid n \in \mathbb{N}\}$

δ	0	1	X	Y	\sqcup
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, \sqcup, L)

The grammar:

- $A_1 \rightarrow q_0 A_2, A_2 \rightarrow [a, a] A_2, A_2 \rightarrow A_3,$
 $A_3 \rightarrow [\varepsilon, \sqcup] A_3, A_3 \rightarrow \varepsilon,$

$$\begin{aligned}
 A_1 &\models q_0 A_2 \models q_0[0, 0] A_2 \models \dots \\
 &\models q_0[0, 0][0, 0][0, 0][1, 1][1, 1][1, 1] A_2 \\
 &\models q_0[0, 0][0, 0][0, 0][1, 1][1, 1][1, 1] A_3 \\
 &\models q_0[0, 0][0, 0][0, 0][1, 1][1, 1][1, 1][\varepsilon, \sqcup] A_3 \\
 &\models q_0[0, 0][0, 0][0, 0][1, 1][1, 1][1, 1][\varepsilon, \sqcup]
 \end{aligned}$$

TM for $\{0^n 1^n \mid n \in \mathbb{N}\}$

δ	0	1	X	Y	\sqcup
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, \sqcup, L)

The grammar:

- ▶ $A_1 \rightarrow q_0 A_2, A_2 \rightarrow [a, a] A_2, A_2 \rightarrow A_3, A_3 \rightarrow [\varepsilon, \sqcup] A_3, A_3 \rightarrow \varepsilon,$
- ▶ $q[a, X] \rightarrow [a, Y] p$
for each $a \in \Sigma \cup \{\varepsilon\}$ and $q, p, X, Y : \delta(q, X) = (p, Y, R),$
- ▶ $[a_1, Z] q[a_2, X] \rightarrow p[a_1, Z][a_2, Y]$
for each $a_1, a_2 \in \Sigma \cup \{\varepsilon\}, Z \in \Gamma,$ and $q, p, X, Y : \delta(q, X) = (p, Y, L),$
- ▶ $[a, X] q \rightarrow q a q, q[a, X] \rightarrow q a q, q \rightarrow \varepsilon$
for each $a \in \Sigma, X \in \Gamma,$ and $q \in F.$

$A_1 \models q_0 A_2 \models q_0[0, 0] A_2 \models \dots$
 $\models q_0[0, 0][0, 0][0, 0][1, 1][1, 1][1, 1] A_2$
 $\models q_0[0, 0][0, 0][0, 0][1, 1][1, 1][1, 1] A_3$
 $\models q_0[0, 0][0, 0][0, 0][1, 1][1, 1][1, 1][\varepsilon, \sqcup] A_3$
 $\models q_0[0, 0][0, 0][0, 0][1, 1][1, 1][1, 1][\varepsilon, \sqcup]$
 $\models [0, X] q_1[0, 0][0, 0][1, 1][1, 1][1, 1][\varepsilon, \sqcup]$
 $\models \dots$
 $\models [0, X][0, 0][0, 0] q_1[1, 1][1, 1][1, 1][\varepsilon, \sqcup]$
 $\models [0, X][0, 0] q_2[0, 0][1, Y][1, 1][1, 1][\varepsilon, \sqcup]$
 $\models \dots$
 $\models [0, X][0, X][0, X] q_0[1, Y][1, Y][1, Y][\varepsilon, \sqcup]$
 $\models [0, X][0, X][0, X][1, Y] q_3[1, Y][1, Y][\varepsilon, \sqcup]$
 $\models [0, X][0, X][0, X][1, Y][1, Y] q_3[1, Y][\varepsilon, \sqcup]$
 $\models [0, X][0, X][0, X][1, Y][1, Y][1, Y] q_3[\varepsilon, \sqcup]$
 $\models [0, X][0, X][0, X][1, Y][1, Y][1, Y][\varepsilon, \sqcup] q_4$
 $\models [0, X][0, X][0, X][1, Y][1, Y][1, Y] q_4 q_4$
 $\models [0, X][0, X][0, X][1, Y][1, Y] q_4 1 q_4 q_4$
 $\models \dots$
 $\models q_4 0 q_4 0 q_4 0 q_4 1 q_4 1 q_4 1 q_4 q_4$
 $\models \dots$
 $\models 000111$

From Type-0 grammar to TM

Let $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ be a Type-0 grammar.

Construct a nondet. TM M to recognize the language $L(G)$.

- ▶ M has two tapes.
- ▶ The input of M (say w) is in tape 1.
- ▶ M simulates the derivation relation of G in tape 2 by repeating the following procedure.
 1. It nondet. chooses a position i in tape 2 and a production rule $\alpha \rightarrow \beta$.
If α appears from position i in tape 2, then α is replaced by β in tape 2. Some shifting over of the symbols on tape 2 should be done if $|\alpha| \neq |\beta|$.
 2. M compares the sequence of symbols in tape 2 with the sequence in tape 1, to see whether w has been generated by G . If so, then M accepts.

Basic properties of r.e. and recursive languages

Theorem. The following closure properties hold.

1. R.e. languages are closed under union and intersection.
2. Recursive languages are closed under all Boolean operations.

Proof sketch.

- ▶ Union and intersection:
Simulate simultaneously the two TMs by a **two-tape** TM.
- ▶ Complementation for recursive languages: Replace F by $Q \setminus F$.

Theorem. Let $L \subseteq \Sigma^*$. If L and $\Sigma^* \setminus L$ are both r.e., then L is recursive.
Proof sketch.

Let $M_1, M_2 : L = L(M_1)$ and $\Sigma^* \setminus L = L(M_2)$.

Then a two-tape TM M is constructed to simulate simultaneously M_1 and M_2 .

If M_1 *accepts*, then M *accepts*.

If M_2 *accepts*, then M *rejects*.

The

termination of M over all inputs is guaranteed by the following fact.

For any $w \in \Sigma^*$,

either the computation of M_1 accepts,
or the computation of M_2 accepts,
but not both.

Outline

The Church – Turing Thesis

Turing machines

Examples of Turing machines

Variants of Turing machines

Multitape Turing Machines

Nondeterministic Turing Machines

Enumerators

Relation with Recursively Enumerable Grammar and TM

The Definition of Algorithm

Hilbert's Problems

Context-Sensitive Grammar and Linear-bounded TM

The Definition of Algorithm

Polynomials and their roots

A polynomial is a sum of terms, where each term is a product of certain variables and a constant, i.e., **coefficient**. For example,

$$6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z = 6x^3yz^2$$

is a term with coefficient 6, and

$$6x^3yz^2 + 3xy^2 - x^3 - 10$$

is a polynomial with four terms, over the variables x , y , and z .

A **root** of a polynomial is an assignment of values to its variables so that the value of the polynomial is 0. This root is an **integral root** because all the variables are assigned integer values. Some polynomials have an integral root and some do not.

Hilbert's Problems

Hilbert's **tenth problem** (out of 23 mathematical problems in 1900) was to **devise an algorithm** that tests whether a polynomial has an integral root. He did not use the term algorithm but rather

a process according to which it can be determined by a finite number of operations.

Hilbert used the term “devise an algorithm”, like such an algorithm must exist.

Church-Turing Thesis

In 1936 to formalize the definition of an algorithm:

1. Alonzo Church proposed λ -calculus;
2. Alan Turing proposed Turing machines,

which were shown to be equivalent.

So we have the Church-Turing Thesis:

Intuitive notion of algorithms = Turing machine algorithms.

Hilbert's Tenth Problem

$D = \{p \mid p \text{ is a polynomial with integer coefficients}$
 $\text{and with an integral root}\}.$

Theorem (Yuri Matijasevič, Martin Davis, Hilary Putnam, and Julia Robinson, 1970)

D is not *decidable*.

A simple variant

$D_1 = \{p \mid p \text{ is a polynomial on a single variable } x \text{ with integer coefficients and with an integral root}\}.$

For instance, $x^2 - 2x + 1$

Lemma

Both D and D_1 are Turing-recognizable.

Proof.

On input $p(x)$

evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \dots$. If at any point the polynomial evaluates to 0 , then accept.



A simple variant (con'd)

Lemma

Let $p(x) = c_1x^n + c_2x^{n-1} + \dots + c_nx + c_{n+1}$ with $c_1 \neq 0$, $n \geq 1$ and $p(x_0) = 0$. Define $c_{\max} = \max \{|c_i|\}_{i \in [n+1]}$.

Then $|x_0| < \frac{c_{\max} \cdot (n+1)}{|c_1|}$.

Corollary

D_1 is *decidable*. But it is impossible to compute such bounds for D (Matijasevič Theorem)

Describing TM

- ▶ **Formal description:** Turing machine's states, transition function, and so on
- ▶ **Implementation description:** use English prose to describe the way that the Turing machine moves its head and the way that it stores data on its tape
- ▶ **High-level description:** use English prose to describe an algorithm, ignoring the implementation details. At this level we do not need to mention how the machine manages its tape or head.
- ▶ We have given formal and implementation-level descriptions of various examples of Turing machines
- ▶ Practicing with lower level Turing machine descriptions helps you understand Turing machines and gain confidence in using them
- ▶ Once you feel confident, high-level descriptions are sufficient unless explicitly requested

Example:

$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$, where $\langle G \rangle$ is a string encoding of the graph G ,

A **high-level description** of a TM M that decides A .

$M =$ “On input $\langle G \rangle$, the encoding of a graph G :

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
 - ▶ For each node in G , mark it if it is attached by an edge to a node that is already marked
3. Scan all the nodes of G to determine whether they all are marked. If they are, **accept**; otherwise, **reject**.”

Outline

The Church – Turing Thesis

- Turing machines

 - Examples of Turing machines

- Variants of Turing machines

 - Multitape Turing Machines

 - Nondeterministic Turing Machines

 - Enumerators

- Relation with Recursively Enumerable Grammar and TM

- The Definition of Algorithm

 - Hilbert's Problems

- Context-Sensitive Grammar and Linear-bounded TM

Context-sensitive grammar and Linear-bounded TM

Context-sensitive grammar (CSG)

A formal grammar $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ of the production rules of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$ such that $\gamma \neq \varepsilon$ and $A \rightarrow \varepsilon$ iff $A = S$ and A does not appear in the right-side of any rules.

Intuitively, a Linear-bounded TM (a.k.a., linear bounded automata) is a (single-tape) nondeterministic TM using linear space.

Linear-bounded TM (LBA)

A LBA is a **nondeterministic** TM s.t.

- ▶ Its input alphabet includes two special symbols $\#$ and $\$$, the **left and right endmarkers**,
- ▶ The LBA has no moves left from $\#$ or right from $\$$, nor may print another symbol over $\#$ or $\$$.

The language accepted by a linear-bounded TM M ,

$$\{w \mid w \in (\Sigma - \{\#, \$\})^*, q_0 \# w \$ \vdash_M^* \alpha q \beta \text{ for some } q \in F\}$$

From LBA to CSG

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a LBA. The CSG G

- ▶ first nondet. generates a finite word

$$\begin{bmatrix} a_1 \\ q_0 \# a_1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_2 \end{bmatrix} \cdots \begin{bmatrix} a_{n-1} \\ a_{n-1} \end{bmatrix} \begin{bmatrix} a_n \\ a_n \$ \end{bmatrix},$$

with the intention that $a_1 \dots a_n$ is the input of M .

- ▶ then G simulates the computation of M over $a_1 \dots a_n$,
by rewriting the second components of the generated word.

Formally, $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ is defined as follows. $S = A_1$, and \mathcal{P} includes the following rules,

- ▶ $A_1 \rightarrow [a, q_0 \# a] A_2$, $A_1 \rightarrow [a, q_0 \# a \$]$, $A_2 \rightarrow [a, a] A_2$, $A_2 \rightarrow [a, a \$]$.
- ▶ If $\delta(q, \#) = (p, \#, R)$, then $[a, q \# \alpha] \rightarrow [a, \# p \alpha]$.
- ▶ If $\delta(q, X) = (p, Y, R)$ for $X, Y \in \Sigma \setminus \{\#, \$\}$, then
for each $a, b \in \Sigma \setminus \{\#, \$\}$, $\beta \in \Sigma \cup \# \Sigma \cup \Sigma \$ \cup \# \Sigma \$$,
 $[a, q X][b, \beta] \rightarrow [a, Y][b, p \beta]$, $[a, \# q X][b, \beta] \rightarrow [a, \# Y][b, p \beta]$,
 $[a, \# q X \$] \rightarrow [a, \# Y p \$]$.
- ▶ Symmetrically for the transition $\delta(q, \$) = (p, \$, L)$ and
 $\delta(q, X) = (p, Y, L)$.
- ▶ For each $a, b \in \Sigma \setminus \{\#, \$\}$, $q \in F$, $\alpha_1, \alpha_2, \alpha \in \Sigma \cup \# \Sigma \cup \Sigma \$ \cup \# \Sigma \$$,
 $[a, \alpha_1 q \alpha_2] \rightarrow a$, $[a, \alpha] b \rightarrow ab$, $b[a, \alpha] \rightarrow ba$.

From CSG to LBA

Recalling: From Type-0 grammar to TM

Let $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$ be a Type-0 grammar.

Construct a nondet. TM M to recognize the language $L(G)$.

- ▶ M has two tapes.
- ▶ The input of M (say w) is in tape 1.
- ▶ M simulates the derivation relation of G in tape 2 by repeating the following procedure.
 1. It nondet. chooses a position i in tape 2 and a production rule $\alpha \rightarrow \beta$.
If α appears from position i in tape 2, then α is replaced by β in tape 2.
Some shifting over of the symbols on tape 2 should be done if $|\alpha| \neq |\beta|$.
 2. M compares the sequence of symbols in tape 2 with the sequence in tape 1, to see whether w has been generated by G . If so, then M accepts.

From CSG to LBA

G is a CSG $\Rightarrow G$ is noncontracting ($\alpha \rightarrow \beta$ s.t. $|\alpha| \leq |\beta|$) \Rightarrow

The space of tape 2 used by M is **bounded by $|w|$** .

Therefore, M can be simulated by

a **single-tape** nondeterministic TM M' using linear space.

Properties of CSL: A bird's-eye view

Closure properties

Theorem. CSL are closed under all Boolean operations.