Turing Machines and their variants

204213 Theory of Computation

Jittat Fakcharoenphol

Kasetsart University

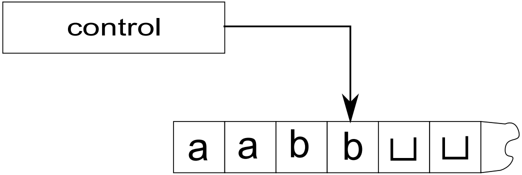
January 6, 2009

Outline

Turing Machines: Components

- An infinite **tape**.
- A tape head that can
  - **read and write** to the tape, and
  - **move** around the tape.

Schematic



How Turing machines work

- The tape initialy contains an input string.

How Turing machines work

- The tape initialy contains an input string.
- The rest of the tape is blank (denoted by $\sqcup$).

How Turing machines work

- The tape initialy contains an input string.
- The rest of the tape is blank (denoted by $\sqcup$).
- The machine reads a symbol from of the tape where its head is at.

How Turing machines work

- The tape initialy contains an input string.
- The rest of the tape is blank (denoted by $\sqcup$).
- The machine reads a symbol from of the tape where its head is at.
- It can write a symbol back and move left or right.

How Turing machines work

- The tape initialy contains an input string.
- The rest of the tape is blank (denoted by $\sqcup$).
- The machine reads a symbol from of the tape where its head is at.
- It can write a symbol back and move left or right.
- At the end of the computation, the machine outputs accept or reject, by entering accept state of reject state. (After changing, it halts.)

How Turing machines work

- The tape initialy contains an input string.
- The rest of the tape is blank (denoted by $\sqcup$).
- The machine reads a symbol from of the tape where its head is at.
- It can write a symbol back and move left or right.
- At the end of the computation, the machine outputs accept or reject, by entering accept state of reject state. (After changing, it halts.)
- It can go on forever (not entering any accept or reject states).

Definition

Definition (Turing Machine)

A **Turing machine** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are finite sets and

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

Defining how TM computes

We need two key concepts:

- Configurations: the “states” of the TM
- Transition: how the TM moves.

Configuration

- At any point of the computation, the TM can be in some state, and at some position on the tape.

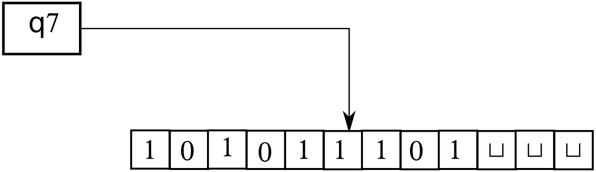
Configuration

- At any point of the computation, the TM can be in some state, and at some position on the tape.
- A **configuration** of the Turing machine, "the current computing status" can be defined with the current state, the current position of the tape head, and the content of the tape.

Configuration

- At any point of the computation, the TM can be in some state, and at some position on the tape.
- A **configuration** of the Turing machine, "the current computing status" can be defined with the current state, the current position of the tape head, and the content of the tape.
- We usually write configurate as: $u\ q\ v$, where q is the state, uv is the current content of the tape, and the TM is at the first symbol of v.

Configuration: $10101q_7 1101$



One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.

One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.
- Let $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$.

One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.
- Let $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$. If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $ua\ q_i\ bv$ yields

One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.
- Let $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$. If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $ua\ q_i\ bv$ yields $u\ q_j\ acv$.

One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.
- Let $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$. If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $ua\ q_i\ bv$ yields $u\ q_j\ acv$. (This is a left-move.)

One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.
- Let $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$. If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $ua\ q_i\ bv$ yields $u\ q_j\ acv$. (This is a left-move.)
- If $\delta(q_i, b) = (q_j, c, \mathrm{R})$, then $ua\ q_i\ bv$ yields

One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.
- Let $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$. If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $ua\ q_i\ bv$ yields $u\ q_j\ acv$. (This is a left-move.)
- If $\delta(q_i, b) = (q_j, c, \mathrm{R})$, then $ua\ q_i\ bv$ yields $uac\ q_j\ v$.

One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.
- Let $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$. If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $ua\ q_i\ bv$ yields $u\ q_j\ acv$. (This is a left-move.)
- If $\delta(q_i, b) = (q_j, c, \mathrm{R})$, then $ua\ q_i\ bv$ yields $uac\ q_j\ v$.
- Are these all case?

One-step move

- Configuration C_1 **yields** configuration C_2 if the Turing machine can move from C_1 to C_2 in one step.
- Let $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$. If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $ua\ q_i\ bv$ yields $u\ q_j\ acv$. (This is a left-move.)
- If $\delta(q_i, b) = (q_j, c, \mathrm{R})$, then $ua\ q_i\ bv$ yields $uac\ q_j\ v$.
- Are these all case? No. We'll have to deal with the case when the head are a the end of the tape on both sides.

One-step move: at the ends of the tape

- When at the left end, trying to move left:

One-step move: at the ends of the tape

- When at the left end, trying to move left: the TM must stay at the same place.

One-step move: at the ends of the tape

- When at the left end, trying to move left: the TM must stay at the same place.
  - If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $q_i\ bv$ yields

One-step move: at the ends of the tape

- When at the left end, trying to move left: the TM must stay at the same place.
  - If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $q_i\ bv$ yields $q_j\ cv$.

One-step move: at the ends of the tape

- When at the left end, trying to move left: the TM must stay at the same place.
  - If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $q_i\ bv$ yields $q_j\ cv$.
- When at the right end, trying to move right:

One-step move: at the ends of the tape

- When at the left end, trying to move left: the TM must stay at the same place.
  - If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $q_i\ bv$ yields $q_j\ cv$.
- When at the right end, trying to move right: just goes over the empty symble.

One-step move: at the ends of the tape

- When at the left end, trying to move left: the TM must stay at the same place.
  - If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $q_i\ bv$ yields $q_j\ cv$.
- When at the right end, trying to move right: just goes over the empty symble.
  - We can treat the configuration $ua\ q_i$ as $ua\ q_i\ \sqcup$, and this case is not a special case anymore.

One-step move: at the ends of the tape

- When at the left end, trying to move left: the TM must stay at the same place.
  - If $\delta(q_i, b) = (q_j, c, \mathrm{L})$, then $q_i\ bv$ yields $q_j\ cv$.
- When at the right end, trying to move right: just goes over the empty symble.
  - We can treat the configuration $ua\ q_i$ as $ua\ q_i\ \sqcup$, and this case is not a special case anymore.

Configurations

3 types of special configurations:

- the start configuration,

Configurations

3 types of special configurations:

- the start configuration,
- the accepting configurations, and

Configurations

3 types of special configurations:

- the start configuration,
- the accepting configurations, and
- the rejecting configurations.

Configurations

3 types of special configurations:

- the start configuration,
- the accepting configurations, and
- the rejecting configurations.
- The accepting and rejecting configurations are called halting configurations.

Computation with TM

A Turing machine M accepts input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where

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

- A collection of strings that a Turing machine M accepts is the language of M, denoted by $L(M)$.

- A collection of strings that a Turing machine M accepts is the language of M, denoted by $L(M)$.
- We say that M recognizes $L(M)$.

- A collection of strings that a Turing machine M accepts is the language of M, denoted by $L(M)$.
- We say that M recognizes $L(M)$.

Definition

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

Output of a TM

- The output of a TM can be accept, reject, or loop.

Output of a TM

- The output of a TM can be accept, reject, or loop.
- A Turing machine may not accept or reject a string.

Output of a TM

- The output of a TM can be accept, reject, or loop.
- A Turing machine may not accept or reject a string.
- We are interested particularly in TM thats halts (does not loop).

Output of a TM

- The output of a TM can be accept, reject, or loop.
- A Turing machine may not accept or reject a string.
- We are interested particularly in TM thats halts (does not loop). We call them **decider**.

Output of a TM

- The output of a TM can be accept, reject, or loop.
- A Turing machine may not accept or reject a string.
- We are interested particularly in TM thats halts (does not loop). We call them **decider**.
- A decider that recognizes some language also is said to **decides** that langauge.

Definition

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

Arithmetics

- Design a TM M_3 that decides the language

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

TM: M_3

M_3 = "On input string w:

1. Scan the input from left to right to check if it is $a^+b^+c^+$ and **reject** if it isn't.

TM: M_3

M_3 = "On input string w:

1. Scan the input from left to right to check if it is $a^+b^+c^+$ and **reject** if it isn't.
2. Move the head back to the left-hand end of the tape.

TM: M_3

M_3 = "On input string w:

1. Scan the input from left to right to check if it is $\mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$ and **reject** if it isn't.
2. Move the head back to the left-hand end of the tape.
3. Cross off an $\mathtt{a}$ and scan to the right until $\mathtt{b}$ occurs. Cross each pair of $\mathtt{b}$'s and $\mathtt{c}$'s until all $\mathtt{b}$'s are crossed off.

TM: M_3

M_3 = "On input string w:

1. Scan the input from left to right to check if it is $\mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$ and **reject** if it isn't.
2. Move the head back to the left-hand end of the tape.
3. Cross off an a and scan to the right until b occurs. Cross each pair of b's and c's until all b's are crossed off.
    - If c's are gone before b, **reject**.

TM: M_3

M_3 = "On input string w:

1. Scan the input from left to right to check if it is $\mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$ and **reject** if it isn't.
2. Move the head back to the left-hand end of the tape.
3. Cross off an a and scan to the right until b occurs. Cross each pair of b's and c's until all b's are crossed off.
   - If c's are gone before b, **reject**.
4. Restore all b's and go back to step 3 if there're more a's.

TM: M_3

M_3 = "On input string w:

1. Scan the input from left to right to check if it is $a^+b^+c^+$ and **reject** if it isn't.
2. Move the head back to the left-hand end of the tape.
3. Cross off an a and scan to the right until b occurs. Cross each pair of b's and c's until all b's are crossed off.
    - If c's are gone before b, **reject**.
4. Restore all b's and go back to step 3 if there're more a's.
    - If all a's are crossed off, check if there is no c's left. If that's the case, **accept**, otherwise **reject**."

Element distinctness

- Given a list of strings over $\{0, 1\}$ separated by #'s, design a TM that accepts if all strings in the list are different.

Element distinctness

- Given a list of strings over $\{0,1\}$ separated by #'s, design a TM that accepts if all strings in the list are different.
- I.e., design a TM that decides the language

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

Variants of Turing Machines

- There are many alternative definitions of TM's.

Variants of Turing Machines

- There are many alternative definitions of TM's.
- They are called variants of TM's.

Variants of Turing Machines

- There are many alternative definitions of TM's.
- They are called variants of TM's.
- We'll see that they all have the same power.

Variants of Turing Machines

- There are many alternative definitions of TM's.
- They are called variants of TM's.
- We'll see that they all have the same power. This demonstrates the robustness in the definition of TM's.

Variants of Turing Machines

- There are many alternative definitions of TM's.
- They are called variants of TM's.
- We'll see that they all have the same power. This demonstrates the robustness in the definition of TM's. Also, this is an evidence that TM's "capture" the idea of computation (because whatever computing machine we can think of they are all equivalent to TM's).

TM with "stay put"

- Let's start with an easy variant. Suppose we allow additional head movement: "stay put (S)".

TM with "stay put"

- Let's start with an easy variant. Suppose we allow additional head movement: "stay put (S)".
- The transition function will be of the form

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}.$$

- Does this give TM's more power?

TM with "stay put"

- Let's start with an easy variant. Suppose we allow additional head movement: "stay put (S)".
- The transition function will be of the form

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}.$$

- Does this give TM's more power?
- Not really. We can convert a TM with "stay put" to a standard TM as follows.

TM with "stay put"

- Let's start with an easy variant. Suppose we allow additional head movement: "stay put (S)".
- The transition function will be of the form

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}.$$

- Does this give TM's more power?
- Not really. We can convert a TM with "stay put" to a standard TM as follows.
  - For any "stay put" transition, we replace with two transitions: "right" and "left".

Multitape Turing Machines

- A **multitape Turing machines** has many tapes.

Multitape Turing Machines

- A **multitape Turing machines** has many tapes.
- For each tape, the machine has a head for reading and writing it.

Multitape Turing Machines

- A **multitape Turing machines** has many tapes.
- For each tape, the machine has a head for reading and writing it.
- The input appears on tape 1; all other tapes contain blanks.

Multitape Turing Machines

- A **multitape Turing machines** has many tapes.
- For each tape, the machine has a head for reading and writing it.
- The input appears on tape 1; all other tapes contain blanks.
- Let k be the number of tapes. The transition function can be defined as

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k.$$

Multitape Turing Machines

- A **multitape Turing machines** has many tapes.
- For each tape, the machine has a head for reading and writing it.
- The input appears on tape 1; all other tapes contain blanks.
- Let k be the number of tapes. The transition function can be defined as

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k.$$

- E.g., if $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \mathrm{L}, \mathrm{R}, \ldots, \mathrm{L})$ then if the machine is at state q_i and each head on tape i reads symbol a_i, it'll write b_i on each tape i, change state to q_j and move each head accordingly.

Theorem 1 (equivalence between multitape TM's and TM's)

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

Proof (1)

- We'll simulate a multitape TM M to a single-tape TM S.

Proof (1)

- We'll simulate a multitape TM M to a single-tape TM S. Suppose that M has k tapes.

Proof (1)

- We'll simulate a multitape TM M to a single-tape TM S. Suppose that M has k tapes.
- We store all information on the k tapes on S's tape. What should we keep?

Proof (1)

- We'll simulate a multitape TM M to a single-tape TM S. Suppose that M has k tapes.
- We store all information on the k tapes on S's tape. What should we keep?
  - The contents of the tapes.

Proof (1)

- We'll simulate a multitape TM M to a single-tape TM S. Suppose that M has k tapes.
- We store all information on the k tapes on S's tape. What should we keep?
  - The contents of the tapes.
  - The position of the heads

Proof (1)

- We'll simulate a multitape TM M to a single-tape TM S. Suppose that M has k tapes.
- We store all information on the k tapes on S's tape. What should we keep?
  - The contents of the tapes.
  - The position of the heads
- How?

Proof (2)

- Combine all tapes into one tape, using # as delimiters.
- Use "marked" symbols to identify the head position on each tape.
- Example:
  Multitape TM:
  - Tape 1: 10101$\boxed{1}$10
  - Tape 2: 00$\boxed{0}$01
  - Tape 3: 011$\boxed{1}$111

Proof (2)

- Combine all tapes into one tape, using # as delimiters.
- Use "marked" symbols to identify the head position on each tape.
- Example:
  Multitape TM:
  - Tape 1: 10101$\boxed{1}$10
  - Tape 2: 00$\boxed{0}$01
  - Tape 3: 011$\boxed{1}$111

  Simulated TM:
  - Tape: #10101$\dot{1}$10#00$\dot{0}$01#011$\dot{1}$111#

Proof (3): Definign S

$S =$ "On input $w = w_1 \cdots w_n$:

1. Put all tape into the format defined previously:

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

Proof (3): Definign S

$S =$ "On input $w = w_1 \cdots w_n$:

1. Put all tape into the format defined previously:

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

2. For each move, S scans its tape once to determine the symbols under all M's heads. It than proceeds to do the update according to M's transition function.

Proof (3): Definign S

$S =$ "On input $w = w_1 \cdots w_n$:

1. Put all tape into the format defined previously:

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

2. For each move, S scans its tape once to determine the symbols under all M's heads. It than proceeds to do the update according to M's transition function.
3. If the content of any simulated tape is overflow (i.e., S has to write over some # on the right), S writes a blank symbol to that delimiter, and shifts all the tape contents to the right."

Corollary 2

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

Nondeterministic Turing Machines

- A nondeterministic Turing machine can make "nondeterministic" move.

Nondeterministic Turing Machines

- A nondeterministic Turing machine can make "nondeterministic" move.
- As expected, its transition function has the form

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

Nondeterministic Turing Machines

- A nondeterministic Turing machine can make "nondeterministic" move.
- As expected, its transition function has the form

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

- Again, we view the computation of a nondeterministic Turing machine as a tree, where each branching corresponds to the place where the TM can make different moves.

Nondeterministic Turing Machines

- A nondeterministic Turing machine can make "nondeterministic" move.
- As expected, its transition function has the form

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

- Again, we view the computation of a nondeterministic Turing machine as a tree, where each branching corresponds to the place where the TM can make different moves.
- Can nondeterminism help?

Theorem 3

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea

- We will simulate a nondeterministic TM N with a deterministic TM D.
- We can try having D visits all the branch in the computation tree of N

Proof idea

- We will simulate a nondeterministic TM N with a deterministic TM D.
- We can try having D visits all the branch in the computation tree of N
  What's wrong with that?

Proof idea

- We will simulate a nondeterministic TM N with a deterministic TM D.
- We can try having D visits all the branch in the computation tree of N
  What's wrong with that? (hint: halting?)

Proof idea

- We will simulate a nondeterministic TM N with a deterministic TM D.
- We can try having D visits all the branch in the computation tree of N
  What's wrong with that? (hint: halting?)
  - If on some branch N loops forever, D will get stuck in that branch and will never find the accept state.

Proof idea

- We will simulate a nondeterministic TM N with a deterministic TM D.
- We can try having D visits all the branch in the computation tree of N
  What's wrong with that? (hint: halting?)
  - If on some branch N loops forever, D will get stuck in that branch and will never find the accept state.
- The idea is to let D explore the computation tree carefully.

Proof idea

- We will simulate a nondeterministic TM N with a deterministic TM D.
- We can try having D visits all the branch in the computation tree of N
  What's wrong with that? (hint: halting?)
  - If on some branch N loops forever, D will get stuck in that branch and will never find the accept state.
- The idea is to let D explore the computation tree carefully.
  - D explore all branches at some fixed depth. After exploring all branches, it start exploring at the next depth.

Proof idea

- We will simulate a nondeterministic TM N with a deterministic TM D.
- We can try having D visits all the branch in the computation tree of N
  What's wrong with that? (hint: halting?)
  - If on some branch N loops forever, D will get stuck in that branch and will never find the accept state.
- The idea is to let D explore the computation tree carefully.
  - D explore all branches at some fixed depth. After exploring all branches, it start exploring at the next depth.
  - This way of exploring the tree is called breadth-first search.