INF2080

Turing Machines

Daniel Lupp

Universitetet i Oslo

22nd February 2018

Department of Informatics

University of Oslo

So far

So far, our computational models had limited memory:

So far

So far, our computational models had limited memory:

- DFA/NFA: finite memory represented in the states;

So far

So far, our computational models had limited memory:

- DFA/NFA: finite memory represented in the states; accepted regular languages

So far

So far, our computational models had limited memory:

- DFA/NFA: finite memory represented in the states; accepted regular languages
- PDA: finite memory in states, restricted infinite memory in a stack;

So far

So far, our computational models had limited memory:

- DFA/NFA: finite memory represented in the states; accepted regular languages
- PDA: finite memory in states, restricted infinite memory in a stack; accepted context-free languages

So far

So far, our computational models had limited memory:

- DFA/NFA: finite memory represented in the states; accepted regular languages
- PDA: finite memory in states, restricted infinite memory in a stack; accepted context-free languages

Last week, we saw the pumping lemma for context-free languages

So far

So far, our computational models had limited memory:

- DFA/NFA: finite memory represented in the states; accepted regular languages
- PDA: finite memory in states, restricted infinite memory in a stack; accepted context-free languages

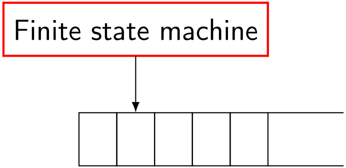
Last week, we saw the pumping lemma for context-free languages
Saw some examples of languages that PDA's do not cover!

Turing Machiens

Today: Introduce computational model underlying most of modern computer science

Turing Machiens

Today: Introduce computational model underlying most of modern computer science



A Turing machine is a finite state machine that has access to an infinite tape

FA vs. TM

Properties of the automata we've seen:

FA vs. TM

Properties of the automata we've seen:

- could only read input once (and never move backwards over the input)

FA vs. TM

Properties of the automata we've seen:

- could only read input once (and never move backwards over the input)
- would only accept after having read the entire input (reject if no computational branches accept)

FA vs. TM

Properties of the automata we've seen:

- could only read input once (and never move backwards over the input)
- would only accept after having read the entire input (reject if no computational branches accept)
- either finite memory (DFA), or restricted access to memory (PDA)

FA vs. TM

Properties of the automata we've seen:

- could only read input once (and never move backwards over the input)
- would only accept after having read the entire input (reject if no computational branches accept)
- either finite memory (DFA), or restricted access to memory (PDA)

Turing machines are a bit different:

FA vs. TM

Properties of the automata we've seen:

- could only read input once (and never move backwards over the input)
- would only accept after having read the entire input (reject if no computational branches accept)
- either finite memory (DFA), or restricted access to memory (PDA)

Turing machines are a bit different:

- can move left and right across it's tape

FA vs. TM

Properties of the automata we've seen:

- could only read input once (and never move backwards over the input)
- would only accept after having read the entire input (reject if no computational branches accept)
- either finite memory (DFA), or restricted access to memory (PDA)

Turing machines are a bit different:

- can move left and right across it's tape
- if enters accept/reject state, immediately stops computing

FA vs. TM

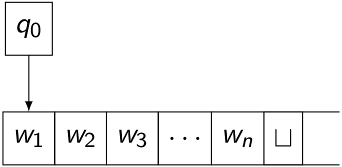
Properties of the automata we've seen:

- could only read input once (and never move backwards over the input)
- would only accept after having read the entire input (reject if no computational branches accept)
- either finite memory (DFA), or restricted access to memory (PDA)

Turing machines are a bit different:

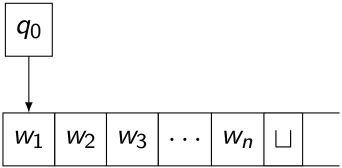
- can move left and right across it's tape
- if enters accept/reject state, immediately stops computing
- unrestricted access to infinite memory

A Turing machine starts in the following configuration



Turing Machines

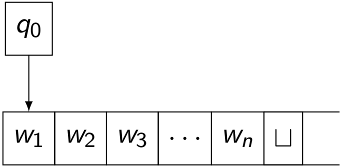
A Turing machine starts in the following configuration



- q_0 is the start state

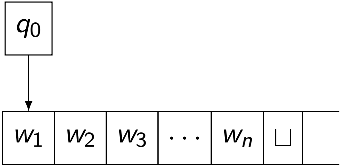
Turing Machines

A Turing machine starts in the following configuration



- q_0 is the start state
- $w_1 \cdots w_n$ is the input string

A Turing machine starts in the following configuration



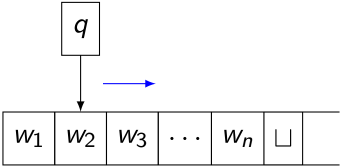
- q_0 is the start state
- $w_1 \cdots w_n$ is the input string
- $\sqcup$ is the *blank symbol*: represents that the cell on tape does not contain any value

Turing Machines

- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right

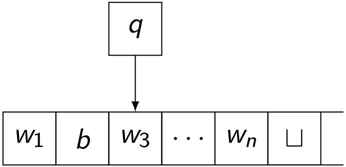
- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right
- Say that if M is in state q and reads w_2 it writes b and goes to the right:

- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right
- Say that if M is in state q and reads w_2 it writes b and goes to the right:



Turing Machines

- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right
- Say that if M is in state q and reads w_2 it writes b and goes to the right:



Turing Machines

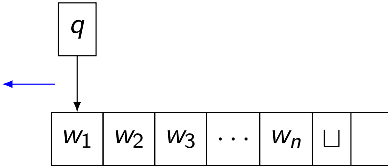
- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right

Turing Machines

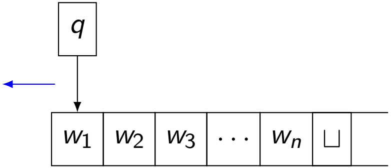
- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right
- Say that if M is in state q and reads w_1 it writes b and goes to the left:

Turing Machines

- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right
- Say that if M is in state q and reads w_1 it writes b and goes to the left:



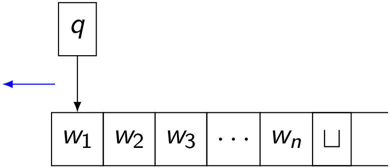
- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right
- Say that if M is in state q and reads w_1 it writes b and goes to the left:



- The machine is at the leftmost cell on the tape!

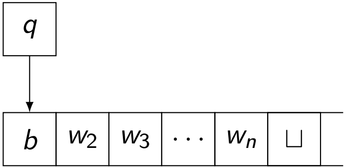
Turing Machines

- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right
- Say that if M is in state q and reads w_1 it writes b and goes to the left:



- The machine is at the leftmost cell on the tape!
- The machine cannot move left; instead, it performs the write operation and stays in the same cell

- Given current state and tape symbol, the Turing machine can write into the current cell, and move left/right
- Say that if M is in state q and reads w_1 it writes b and goes to the left:



- The machine is at the leftmost cell on the tape!
- The machine cannot move left; instead, it performs the write operation and stays in the same cell

Recall language $\{ww \mid w \in \{0,1\}^*\}$. Last week we used the pumping lemma to show this is not context-free.

Turing Machines

Recall language $\{ww \mid w \in \{0,1\}^*\}$. Last week we used the pumping lemma to show this is not context-free. Let's try to describe a Turing machine that accepts the similar language $\{w\#w \mid w \in \{0,1\}^*\}$. How would the tape operations look?

Turing Machines

Recall language $\{ww \mid w \in \{0,1\}^*\}$. Last week we used the pumping lemma to show this is not context-free. Let's try to describe a Turing machine that accepts the similar language $\{w\#w \mid w \in \{0,1\}^*\}$. How would the tape operations look?

- First input symbol must be compared with the first symbol occuring after $\#$.

Turing Machines

Recall language $\{ww \mid w \in \{0,1\}^*\}$. Last week we used the pumping lemma to show this is not context-free. Let's try to describe a Turing machine that accepts the similar language $\{w\#w \mid w \in \{0,1\}^*\}$. How would the tape operations look?

- First input symbol must be compared with the first symbol occuring after $\#$.
- Each following symbol (before and after $\#$) must be compared as well

Turing Machines

Recall language $\{ww \mid w \in \{0,1\}^*\}$. Last week we used the pumping lemma to show this is not context-free. Let's try to describe a Turing machine that accepts the similar language $\{w\#w \mid w \in \{0,1\}^*\}$. How would the tape operations look?

- First input symbol must be compared with the first symbol occuring after $\#$.
- Each following symbol (before and after $\#$) must be compared as well
- If the same symbols occur on both sides, accept. Else, reject.

Example

$L = \{w\#w \mid w \in \{0,1\}^*\}$:
Intuitively, we *zig-zag* across the tape:

Example

$L = \{w\#w \mid w \in \{0,1\}^*\}$:
Intuitively, we *zig-zag* across the tape:

- Cross off first symbol, move to first symbol after $\#$ and cross off if the same symbol occurs

Example

$L = \{w\#w \mid w \in \{0,1\}^*\}$:
Intuitively, we *zig-zag* across the tape:

- Cross off first symbol, move to first symbol after $\#$ and cross off if the same symbol occurs
- Move back to beginning of tape

Example

$L = \{w\#w \mid w \in \{0,1\}^*\}$:
Intuitively, we *zig-zag* across the tape:

- Cross off first symbol, move to first symbol after $\#$ and cross off if the same symbol occurs
- Move back to beginning of tape
- If no non-crossed off symbols remain, move past $\#$ and check if any non-crossed off symbols remain on that side. If yes, reject. If no, accept.

Example

$L = \{w\#w \mid w \in \{0,1\}^*\}$:
Intuitively, we *zig-zag* across the tape:

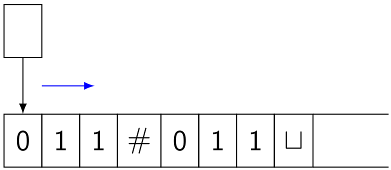
- Cross off first symbol, move to first symbol after $\#$ and cross off if the same symbol occurs
- Move back to beginning of tape
- If no non-crossed off symbols remain, move past $\#$ and check if any non-crossed off symbols remain on that side. If yes, reject. If no, accept.
- Cross off first non-crossed off symbol, move to first non-crossed off symbol after $\#$

Example

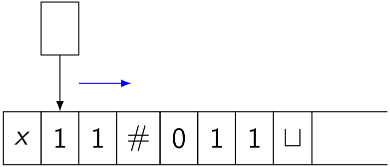
$L = \{w\#w \mid w \in \{0,1\}^*\}$:
Intuitively, we *zig-zag* across the tape:

- Cross off first symbol, move to first symbol after $\#$ and cross off if the same symbol occurs
- Move back to beginning of tape
- If no non-crossed off symbols remain, move past $\#$ and check if any non-crossed off symbols remain on that side. If yes, reject. If no, accept.
- Cross off first non-crossed off symbol, move to first non-crossed off symbol after $\#$
- If no non-crossed off symbols after $\#$ remain or next symbol is not the same, reject. Else, cross off if equal to the last crossed off symbol

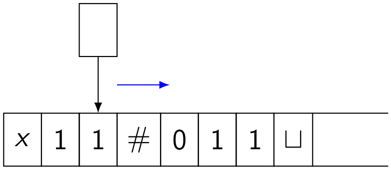
Example



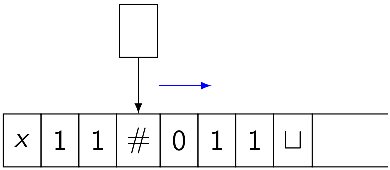
Example



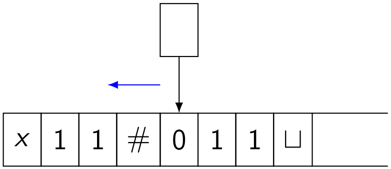
Example



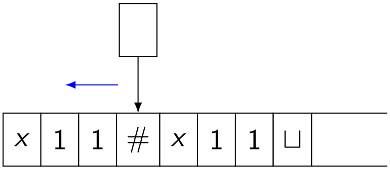
Example



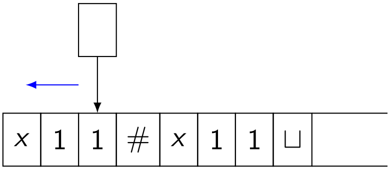
Example



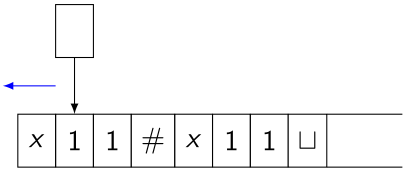
Example



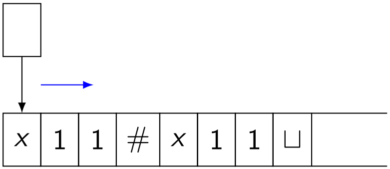
Example



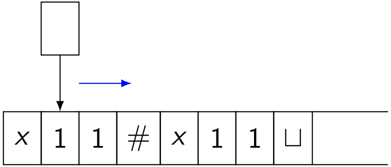
Example



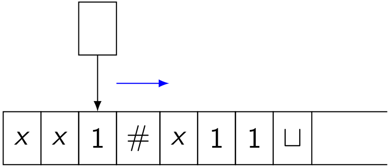
Example



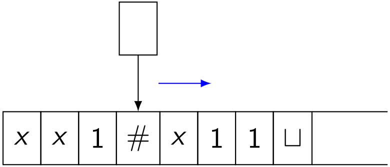
Example



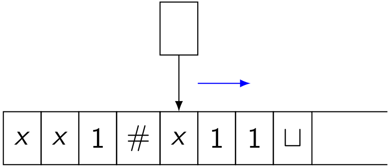
Example



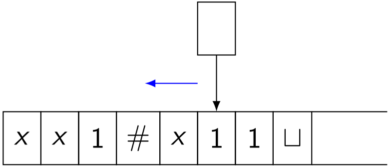
Example



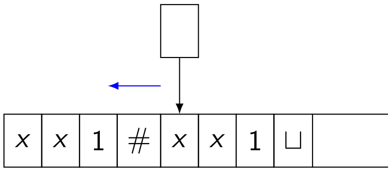
Example



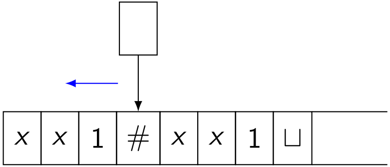
Example



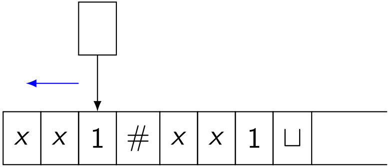
Example



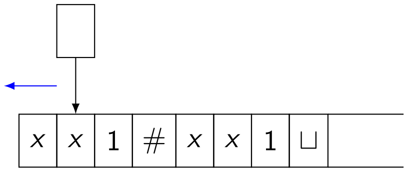
Example



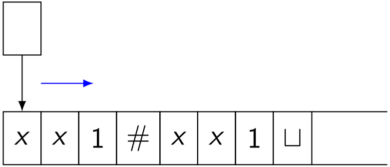
Example



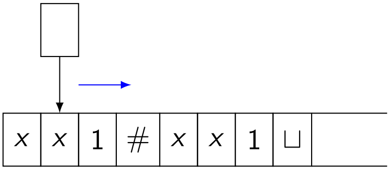
Example



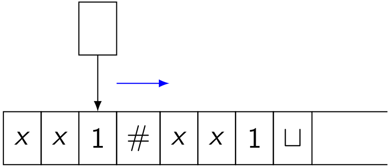
Example



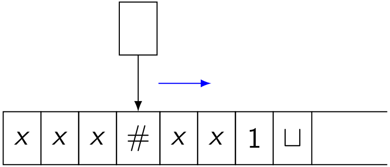
Example



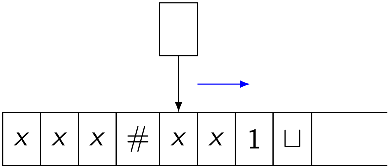
Example



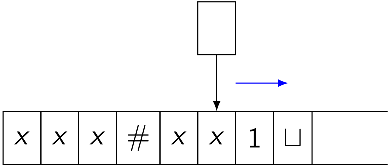
Example



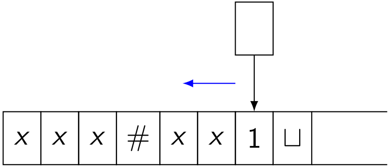
Example



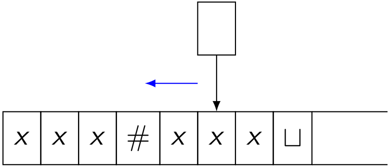
Example



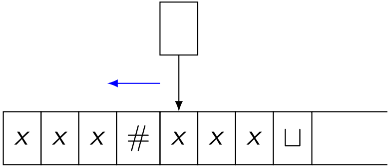
Example



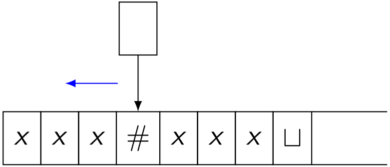
Example



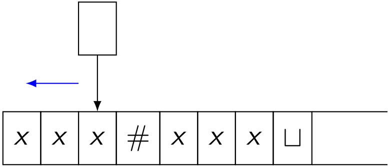
Example



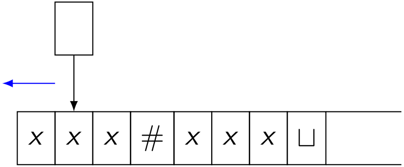
Example



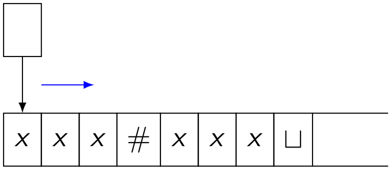
Example



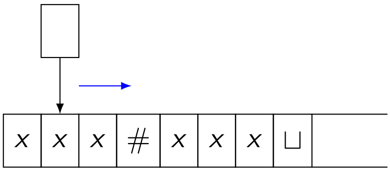
Example



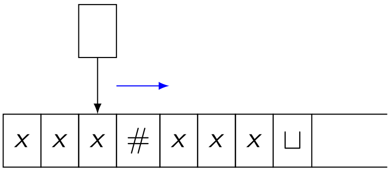
Example



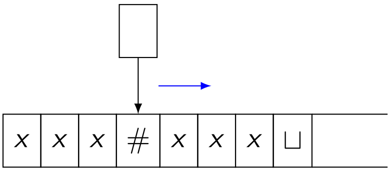
Example



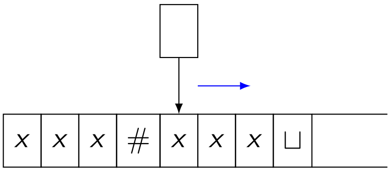
Example



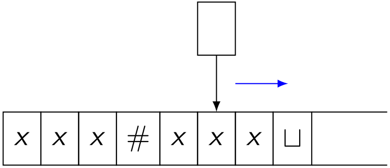
Example



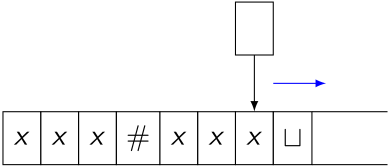
Example



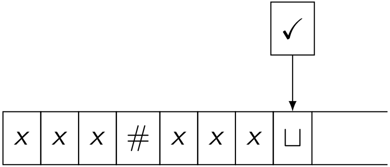
Example



Example



Example

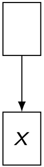


“Move back to beginning”

- We used “move back to beginning of tape”....but how to implement this?

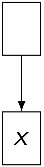
"Move back to beginning"

- We used "move back to beginning of tape"....but how to implement this?
- A Turing machine's head's view:



“Move back to beginning”

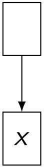
- We used “move back to beginning of tape”....but how to implement this?
- A Turing machine’s head’s view:



- It cannot see anything outside of its current cell

"Move back to beginning"

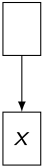
- We used "move back to beginning of tape"....but how to implement this?
- A Turing machine's head's view:



- It cannot see anything outside of its current cell
- If after moving the same symbol appears, how does the machine know whether it *actually* moved or not?

"Move back to beginning"

- We used "move back to beginning of tape"....but how to implement this?
- A Turing machine's head's view:



- It cannot see anything outside of its current cell
- If after moving the same symbol appears, how does the machine know whether it *actually* moved or not?
- Multiple options:
- write a special symbol at the beginning to encode the beginning of the tape

"Move back to beginning"

Other option:

- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.

"Move back to beginning"

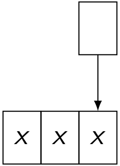
Other option:

- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.

"Move back to beginning"

Other option:

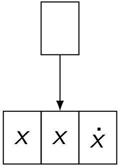
- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



"Move back to beginning"

Other option:

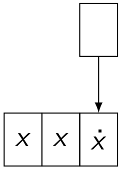
- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



“Move back to beginning”

Other option:

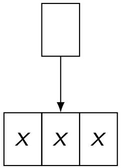
- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it’s at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



“Move back to beginning”

Other option:

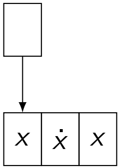
- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



"Move back to beginning"

Other option:

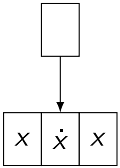
- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



“Move back to beginning”

Other option:

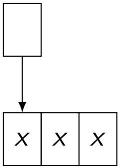
- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



"Move back to beginning"

Other option:

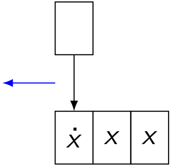
- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



“Move back to beginning”

Other option:

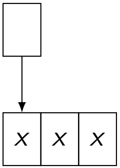
- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



"Move back to beginning"

Other option:

- when looking for beginning of tape: replace current cell contents x with a new symbol $\dot{x}$ to mark where the head was. If after moving left it reads $\dot{x}$, it knows it's at the beginning of the tape.
- Otherwise, go right, replace $\dot{x}$ with x, go left, repeat.



Example

- This was a *low-level description* of a Turing machine, describing in words how the tape operates:

Example

- This was a *low-level description* of a Turing machine, describing in words how the tape operates:
  M= on input w:
  1. Zig-zag across the tape to corresponding positions on either side of $\#$, checking whether these positions contain the same symbol. If they do not, reject. Cross off corresponding symbols if equal
  2. When all symbols before $\#$ have been crossed off, scan symbols after $\#$ to see if any uncrossed symbols remain. If yes, reject, if no, accept.

Example

- This was a *low-level description* of a Turing machine, describing in words how the tape operates:
  M= on input w:
  1. Zig-zag across the tape to corresponding positions on either side of #, checking whether these positions contain the same symbol. If they do not, reject. Cross off corresponding symbols if equal
  2. When all symbols before # have been crossed off, scan symbols after # to see if any uncrossed symbols remain. If yes, reject, if no, accept.
- High-level: "Compare words before and after # and accept if equal, reject if not equal"

Example

- This was a *low-level description* of a Turing machine, describing in words how the tape operates:
  M= on input w:
  1. Zig-zag across the tape to corresponding positions on either side of #, checking whether these positions contain the same symbol. If they do not, reject. Cross off corresponding symbols if equal
  2. When all symbols before # have been crossed off, scan symbols after # to see if any uncrossed symbols remain. If yes, reject, if no, accept.
- High-level: "Compare words before and after # and accept if equal, reject if not equal"
- Implementation level: giving the formal definition of a Turing machine, with all states and transitions (we'll go through some examples tomorrow)

TM—A formal definition

Definition

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

TM—A formal definition

Definition

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,

TM—A formal definition

Definition

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** $\sqcup$

TM—A formal definition

Definition

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 $\Sigma \subseteq \Gamma$,

TM—A formal definition

Definition

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 $\Sigma \subseteq \Gamma$,
4. $\delta : Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition fucntion,

TM—A formal definition

Definition

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 $\Sigma \subseteq \Gamma$,
4. $\delta : Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition fucntion,
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.

Configurations

During each computation step, a TM is in a specific *configuration*:

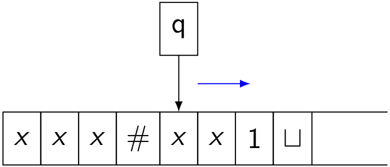
- current tape contents uv;
- current head position;
- current state q.

Configurations

During each computation step, a TM is in a specific *configuration*:

- current tape contents uv;
- current head position;
- current state q.

Notation: uqv encodes that the machine is in state q, pointing at the first symbol in v.
Example:



Notation: $xxx\#qxx1$

Configurations

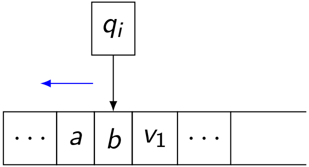
A configuration C_1 *yields* C_2 if TM can legally go from C_1 to C_2 in a single step:

- uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$.

Configurations

A configuration C_1 *yields* C_2 if TM can legally go from C_1 to C_2 in a single step:

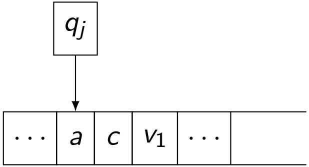
- uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$.



Configurations

A configuration C_1 *yields* C_2 if TM can legally go from C_1 to C_2 in a single step:

- uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$.



Configurations

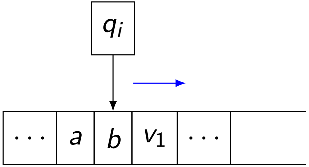
A configuration C_1 *yields* C_2 if TM can legally go from C_1 to C_2 in a single step:

- uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$.

Configurations

A configuration C_1 *yields* C_2 if TM can legally go from C_1 to C_2 in a single step:

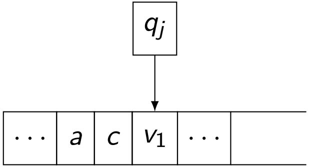
- uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$.



Configurations

A configuration C_1 *yields* C_2 if TM can legally go from C_1 to C_2 in a single step:

- uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$.



Configurations

If the head is at one end of the tape:

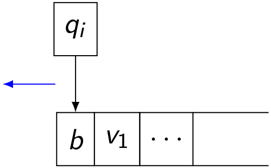
- If the head is at the leftmost side and it moves left:

$$q_i bv \text{ yields } q_j cv \text{ if } \delta(q_i, b) = (q_j, c, L).$$

Configurations

If the head is at one end of the tape:

- If the head is at the leftmost side and it moves left:

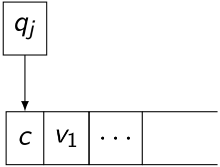


$q_i bv$ yields $q_j cv$ if $\delta(q_i, b) = (q_j, c, L)$.

Configurations

If the head is at one end of the tape:

- If the head is at the leftmost side and it moves left:

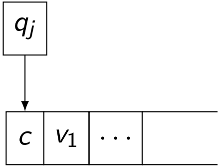


$q_i bv$ yields $q_j cv$ if $\delta(q_i, b) = (q_j, c, L)$.

Configurations

If the head is at one end of the tape:

- If the head is at the leftmost side and it moves left:

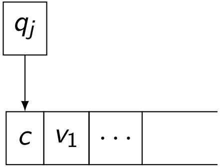
  

  $q_i bv$ yields $q_j cv$ if $\delta(q_i, b) = (q_j, c, L)$.
- If the head is at the rightmost side and it moves right: $uq_i b$ yields $ucq_j = ucq_j\sqcup$ if $\delta(q_i, b) = (q_j, c, R)$.

Configurations

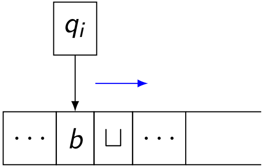
If the head is at one end of the tape:

- If the head is at the leftmost side and it moves left:

  $q_i bv$ yields $q_j cv$ if $\delta(q_i, b) = (q_j, c, L)$.

- If the head is at the rightmost side and it moves right: $uq_i b$ yields $ucq_j = ucq_j\sqcup$ if

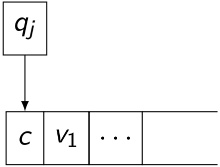
  

  $\delta(q_i, b) = (q_j, c, R)$.

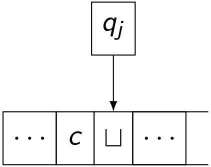
Configurations

If the head is at one end of the tape:

- If the head is at the leftmost side and it moves left:

  $q_i bv$ yields $q_j cv$ if $\delta(q_i, b) = (q_j, c, L)$.
- If the head is at the rightmost side and it moves right: $uq_i b$ yields $ucq_j = ucq_j\sqcup$ if

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

Configurations

- The *starting configuration* is $q_0 w$

Configurations

- The *starting configuration* is $q_0 w$
- An *accepting configuration* is a configuration where the state is q_{accept}

Configurations

- The *starting configuration* is $q_0 w$
- An *accepting configuration* is a configuration where the state is q_{accept}
- A *rejecting configuration* is a configuration where the state is q_{reject}

Configurations

- The *starting configuration* is $q_0 w$
- An *accepting configuration* is a configuration where the state is q_{accept}
- A *rejecting configuration* is a configuration where the state is q_{reject}
- A *halting configuration* is an accepting or a rejecting configuration

Configurations

- The *starting configuration* is $q_0 w$
- An *accepting configuration* is a configuration where the state is q_{accept}
- A *rejecting configuration* is a configuration where the state is q_{reject}
- A *halting configuration* is an accepting or a rejecting configuration

Then a Turing machine M accepts a word w if there exists a sequence of configurations $C_1, \ldots, C_k$ such that

1. C_1 is the start configuration of M on input w,

Configurations

- The *starting configuration* is $q_0 w$
- An *accepting configuration* is a configuration where the state is q_{accept}
- A *rejecting configuration* is a configuration where the state is q_{reject}
- A *halting configuration* is an accepting or a rejecting configuration

Then a Turing machine M accepts a word w if there exists a sequence of configurations $C_1, \ldots, C_k$ such that

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

Configurations

- The *starting configuration* is $q_0 w$
- An *accepting configuration* is a configuration where the state is q_{accept}
- A *rejecting configuration* is a configuration where the state is q_{reject}
- A *halting configuration* is an accepting or a rejecting configuration

Then a Turing machine M accepts a word w if there exists a sequence of configurations $C_1, \ldots, C_k$ such that

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.

Languages

For a Turing machine M, the set of words it accepts is *the language of M* (denoted $L(M)$) or the language recognized by M.

Languages

For a Turing machine M, the set of words it accepts is *the language of M* (denoted $L(M)$) or the language recognized by M.

Definition

A language is called *Turing-recognizable* [recursively enumerable] if some Turing machine recognizes it.

Languages

For a Turing machine M, the set of words it accepts is *the language of M* (denoted $L(M)$) or the language recognized by M.

Definition

A language is called *Turing-recognizable* [recursively enumerable] if some Turing machine recognizes it.

- A Turing machine can either *halt* (accept, reject) on a given input, or *loop* (never accept or reject...for example: head always moves to the right, never enters q_{accept} or q_{reject}).

Languages

For a Turing machine M, the set of words it accepts is *the language of M* (denoted $L(M)$) or the language recognized by M.

Definition

A language is called *Turing-recognizable* [recursively enumerable] if some Turing machine recognizes it.

- A Turing machine can either *halt* (accept, reject) on a given input, or *loop* (never accept or reject...for example: head always moves to the right, never enters q_{accept} or q_{reject}).
- A Turing machine *decides* a language A if it accepts all words $w \in A$ and rejects all words $w \notin A$.

Languages

For a Turing machine M, the set of words it accepts is *the language of M* (denoted $L(M)$) or the language recognized by M.

Definition

A language is called *Turing-recognizable* [recursively enumerable] if some Turing machine recognizes it.

- A Turing machine can either *halt* (accept, reject) on a given input, or *loop* (never accept or reject...for example: head always moves to the right, never enters q_{accept} or q_{reject}).
- A Turing machine *decides* a language A if it accepts all words $w \in A$ and rejects all words $w \notin A$. $\rightarrow$ it halts on all inputs!

Languages

For a Turing machine M, the set of words it accepts is *the language of M* (denoted $L(M)$) or the language recognized by M.

Definition

A language is called *Turing-recognizable* [recursively enumerable] if some Turing machine recognizes it.

- A Turing machine can either *halt* (accept, reject) on a given input, or *loop* (never accept or reject...for example: head always moves to the right, never enters q_{accept} or q_{reject}).
- A Turing machine *decides* a language A if it accepts all words $w \in A$ and rejects all words $w \notin A$. $\rightarrow$ it halts on all inputs!

Definition

A language is *Turing-decidable* [decidable, recursive] if there is a Turing machine that decides it.

Example

Previous example of language $\{w\#w \mid w \in \{0,1\}^*\}$ was recognized by machine:
M= on input w:

1. Zig-zag across the tape to corresponding positions on either side of $\#$, checking whether these positions contain the same symbol. If they do not, reject. Cross off corresponding symbols if equal
2. When all symbols before $\#$ have been crossed off, scan symbols after $\#$ to see if any uncrossed symbols remain. If yes, reject, if no, accept.

Example

Previous example of language $\{w\#w \mid w \in \{0,1\}^*\}$ was recognized by machine:
M= on input w:

1. Zig-zag across the tape to corresponding positions on either side of $\#$, checking whether these positions contain the same symbol. If they do not, reject. Cross off corresponding symbols if equal
2. When all symbols before $\#$ have been crossed off, scan symbols after $\#$ to see if any uncrossed symbols remain. If yes, reject, if no, accept.

It was, in fact, decided by M!

Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.

Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.

- words: 0, 00, 0000, 00000000, etc.

Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.

- words: 0, 00, 0000, 00000000, etc.
- all words except 0 have length divisible by 0.

Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.

- words: 0, 00, 0000, 00000000, etc.
- all words except 0 have length divisible by 0.
- idea: if length=1, accept (if input symbol is correct), otherwise divide length by 2. Repeat

How to divide length by 2?

Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

1. replace first 0 with blank symbol $\sqcup$.

Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.

Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).

Example 2

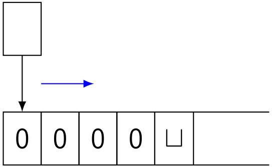
$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.

Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

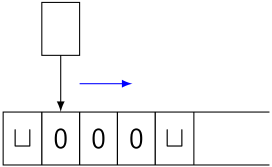
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

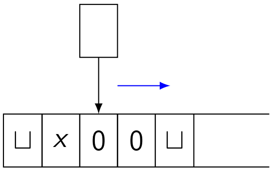
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

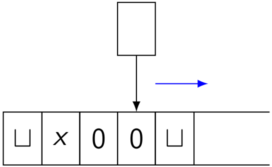
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

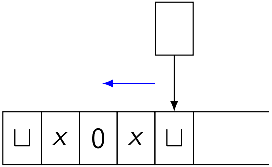
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

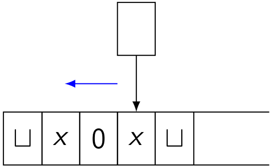
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

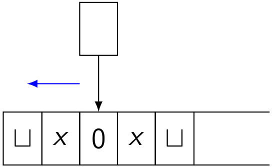
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

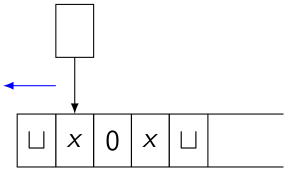
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

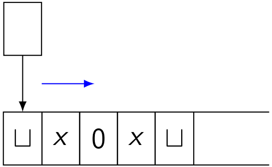
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

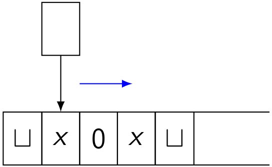
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

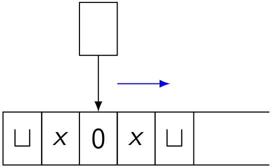
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

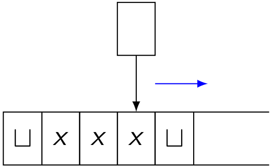
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

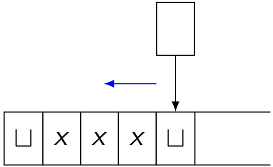
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

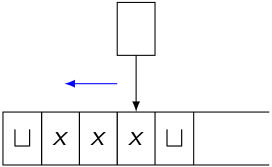
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

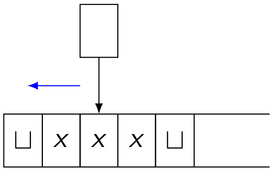
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

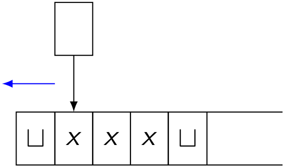
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

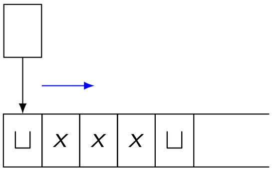
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

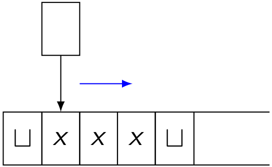
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

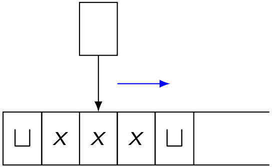
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

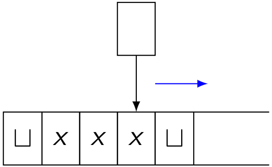
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

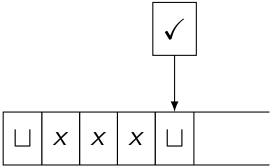
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

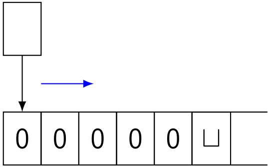
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

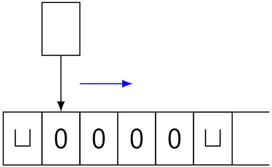
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.

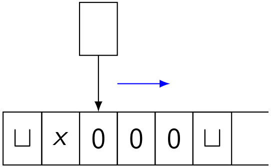


Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.

M=on input w:

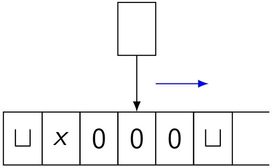
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

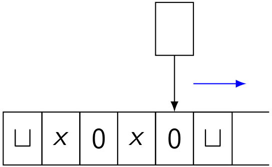
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

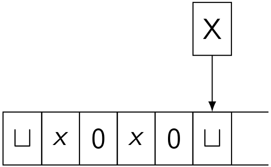
1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.



Example 2

$A = \{0^{2^n} \mid n \geq 0\}$. Want to construct a TM that recognizes (decides?) it.
M=on input w:

1. replace first 0 with blank symbol $\sqcup$.
2. scan input until you read the next 0. If none is found, accept. If one is found, cross it off and cross off every other 0 for the remaining input. If the number of 0's read is odd, reject.
3. Return to head of tape (move left until read $\sqcup$).
4. go to 2.

$\rightarrow$ M decides A!

Alternative Turing machines

- Not surprisingly, Turing machines are *very* powerful!

Alternative Turing machines

- Not surprisingly, Turing machines are *very* powerful!
- Can we make it *even more expressive* by, e.g., allowing the machine to also *stay put* , i.e., not move left or right?

Definition

A *LRS Turing machine* is a Turing machine with a transition function is defined as

$$\delta : Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

LRS TM

We can model *stay put* by adding an extra transition: first move left, then right.

LRS TM

We can model *stay put* by adding an extra transition: first move left, then right.
For each $\delta(q_i, b) = (q_j, c, S)$, we can rewrite this as $\delta(q_i, b) = (q', c, L)$ and $\delta(q', *) = (q_j, *, R)$ where $*$ is any tape symbol.

LRS TM

We can model *stay put* by adding an extra transition: first move left, then right.
For each $\delta(q_i, b) = (q_j, c, S)$, we can rewrite this as $\delta(q_i, b) = (q', c, L)$ and $\delta(q', *) = (q_j, *, R)$ where $*$ is any tape symbol.
So, equally expressive as a normal TM!

Alternative Turing machines

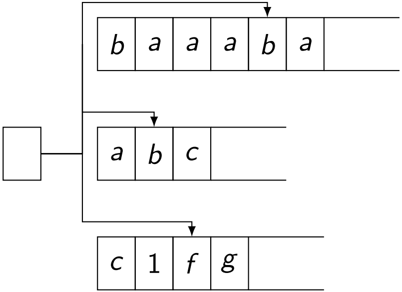
- Can we make it *even more expressive* by, e.g., adding more tapes?

Definition

A *multitape Turing machine* is a turing machine with $k \geq 2$ tapes, i.e., its transition function is defined as

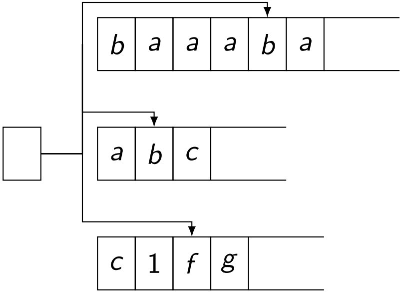
$$\delta : Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}$$

Multitape Turing machines



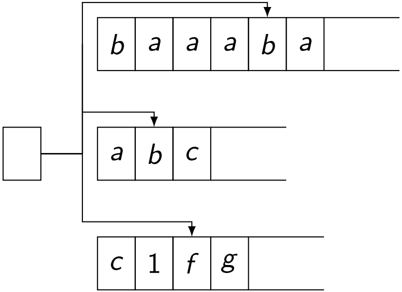
- The machine has access to multiple tapes
- Can this be modelled on a single tape?

Multitape Turing machines



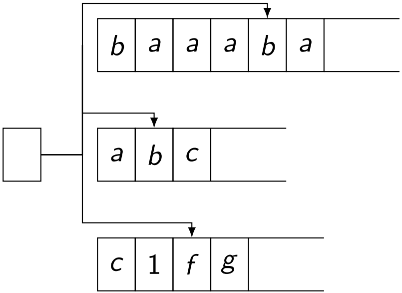
- The machine has access to multiple tapes
- Can this be modelled on a single tape? Yes!

Multitape Turing machines

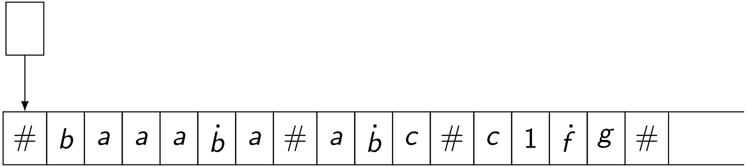


- The machine has access to multiple tapes
- Can this be modelled on a single tape? Yes!
- Idea: encode content of each tape on one tape, using a symbol (#) to separate the different tapes

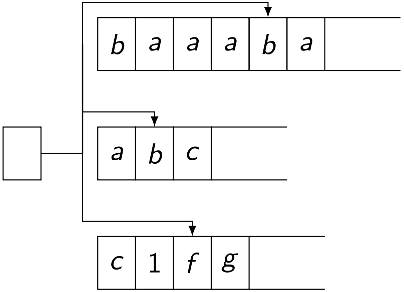
Multitape Turing machines



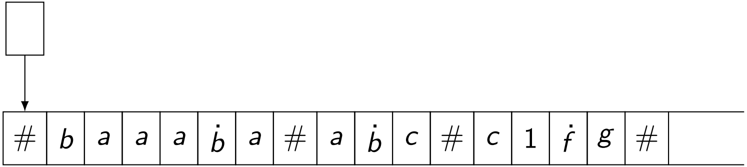
- The machine has access to multiple tapes
- Can this be modelled on a single tape? Yes!
- Idea: encode content of each tape on one tape, using a symbol (#) to separate the different tapes



Multitape Turing machines



- The machine has access to multiple tapes
- Can this be modelled on a single tape? Yes!
- Idea: encode content of each tape on one tape, using a symbol (#) to separate the different tapes
- in each tape block (space between two #), one symbol is marked (represents where the head is on that tape)



Multitape Turing Machines

More formally:

S= on input w:

1. Put S into the following format to model k tapes:

$$\#\dot{w}_1 \ldots w_n\#\dot{\sqcup}\#\dot{\sqcup}\# \ldots \#$$

Multitape Turing Machines

More formally:

S= on input w:

1. Put S into the following format to model k tapes:

$$\#\dot{w}_1 \dots w_n\#\dot{\sqcup}\#\dot{\sqcup}\# \dots \#$$

2. To simulate a single move in the multitape TM, S scans from first $\#$ to $(k+1)$st $\#$ to determine symbols under the virtual heads ("check which cells are dotted"). S then makes a second pass over tape and updates cells according to M's transitions.

Multitape Turing Machines

More formally:
S= on input w:

1. Put S into the following format to model k tapes:

$$\#\dot{w}_1 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$$

2. To simulate a single move in the multitape TM, S scans from first $\#$ to $(k+1)$st $\#$ to determine symbols under the virtual heads ("check which cells are dotted"). S then makes a second pass over tape and updates cells according to M's transitions.
3. If S moves one of virtual heads on to $\#$, this means M has moved on to a previously unread blank symbol on one of the tapes. S then writes blank symbol into that cell, shifts cell contents from this cell to the rightmost $\#$ one to the right. Go to 2.

Multitape Turing Machines

More formally:

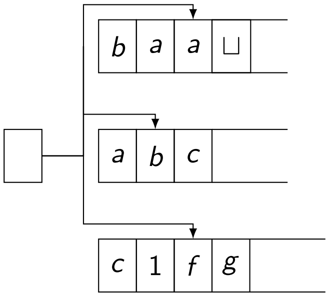
S= on input w:

1. Put S into the following format to model k tapes:

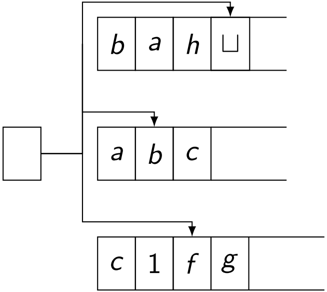
$$\#\dot{w}_1 \ldots w_n\#\dot{\sqcup}\#\dot{\sqcup}\# \ldots \#$$

2. To simulate a single move in the multitape TM, S scans from first $\#$ to $(k+1)$st $\#$ to determine symbols under the virtual heads ("check which cells are dotted"). S then makes a second pass over tape and updates cells according to M's transitions.
3. If S moves one of virtual heads on to $\#$, this means M has moved on to a previously unread blank symbol on one of the tapes. S then writes blank symbol into that cell, shifts cell contents from this cell to the rightmost $\#$ one to the right. Go to 2.

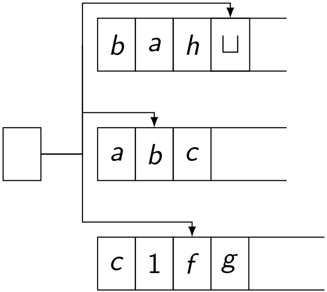
Multitape Turing Machines

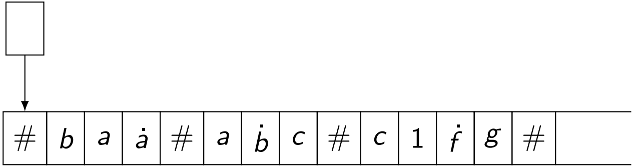


Multitape Turing Machines

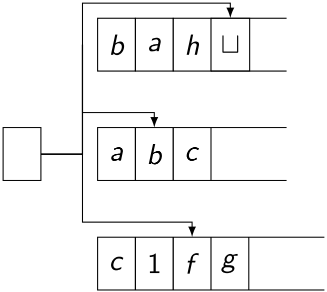


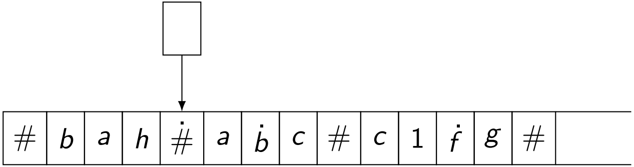
Multitape Turing Machines

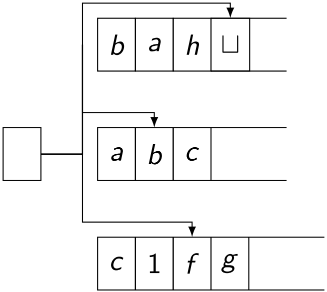


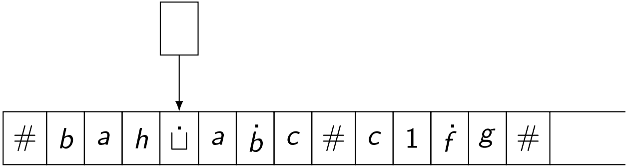


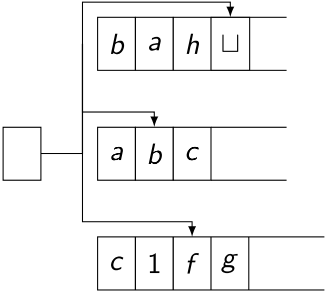
Multitape Turing Machines

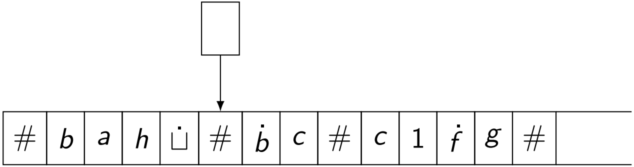




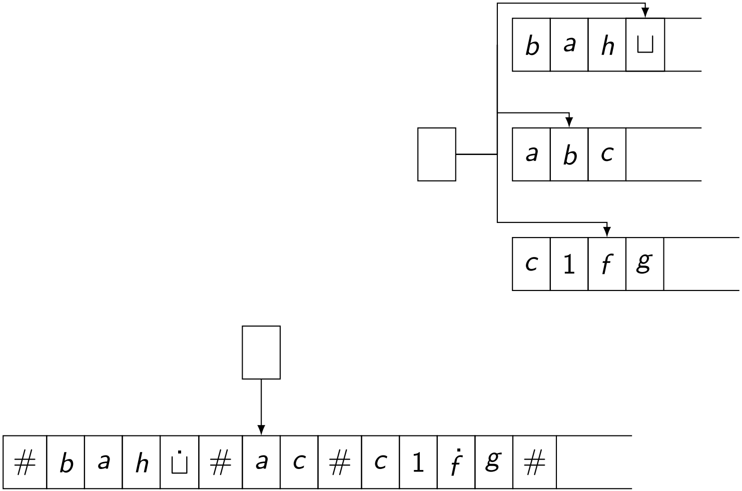

b a h ⊔
a b c
c 1 f g


b a h ⊔̇ a ḃ c # c 1 ḟ g

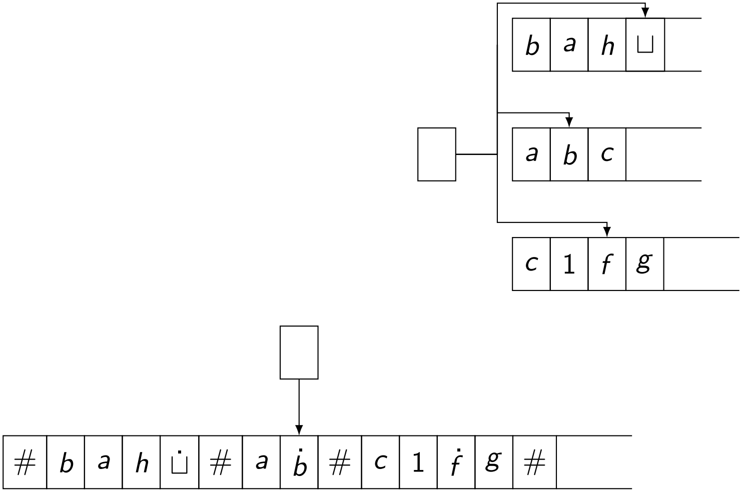

b a h ⊔
a b c
c 1 f g


b a h ⊔̇ # ḃ c # c 1 ḟ g

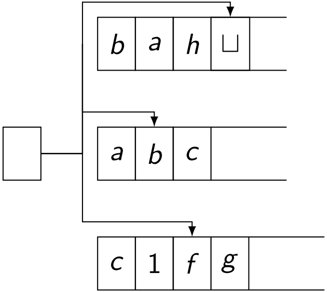
Multitape Turing Machines

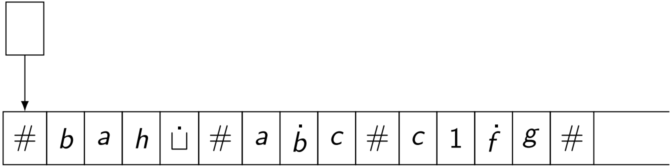


Multitape Turing Machines



Multitape Turing Machines





Multitape Turing machiens

Theorem

A language is Turing recognizable [decidable] if and only if some multitape Turing machine recognizes [decides] it.

Turing Machine alternatives

OK...what about *nondeterministic* Turing machines? Do they let us do anything a deterministic TM cannot?

OK...what about *nondeterministic* Turing machines? Do they let us do anything a deterministic TM cannot?

Definition

A nondeterministic Turing machine has a transition function of the form

$$\delta : Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$$

Nondeterministic TM

We can model a nondeterministic Turing machine N with a deterministic Turing machine M!

Nondeterministic TM

We can model a nondeterministic Turing machine N with a deterministic Turing machine M!
Idea: we use a multitape Turing machine (which we've seen is equivalent to a deterministic, single-tape Turing machine).

Nondeterministic TM

We can model a nondeterministic Turing machine N with a deterministic Turing machine M! Idea: we use a multitape Turing machine (which we've seen is equivalent to a deterministic, single-tape Turing machine).

- We use 3 tapes: *input tape, simulation tape, address tape*

Nondeterministic TM

We can model a nondeterministic Turing machine N with a deterministic Turing machine M!
Idea: we use a multitape Turing machine (which we've seen is equivalent to a deterministic, single-tape Turing machine).

- We use 3 tapes: *input tape, simulation tape, address tape*
- input tape: contains input, will never be altered;

Nondeterministic TM

We can model a nondeterministic Turing machine N with a deterministic Turing machine M! Idea: we use a multitape Turing machine (which we've seen is equivalent to a deterministic, single-tape Turing machine).

- We use 3 tapes: *input tape, simulation tape, address tape*
- input tape: contains input, will never be altered;
- simulation tape: "where the magic happens"...tape simulating current computational branch in N;

Nondeterministic TM

We can model a nondeterministic Turing machine N with a deterministic Turing machine M! Idea: we use a multitape Turing machine (which we've seen is equivalent to a deterministic, single-tape Turing machine).

- We use 3 tapes: *input tape, simulation tape, address tape*
- input tape: contains input, will never be altered;
- simulation tape: "where the magic happens"...tape simulating current computational branch in N;
- address tape: tells us which choices to make in the computational tree of N

Nondeterministic TM

More on the address tape:

- For each Configuration C_i in N, there are multiple possibilities for following configurations.

Nondeterministic TM

More on the address tape:

- For each Configuration C_i in N, there are multiple possibilities for following configurations.
- Let b be the largest number of possible following configurations in δ.

Nondeterministic TM

More on the address tape:

- For each Configuration C_i in N, there are multiple possibilities for following configurations.
- Let b be the largest number of possible following configurations in δ.
- Then a string over $\{1, \ldots b\}$ represents a sequence of choices; e.g., 143 represents that we choose Option 1 after start configuration, then Option 4, then Option 3.

Nondeterministic TM

More on the address tape:

- For each Configuration C_i in N, there are multiple possibilities for following configurations.
- Let b be the largest number of possible following configurations in δ.
- Then a string over $\{1, \ldots b\}$ represents a sequence of choices; e.g., 143 represents that we choose Option 1 after start configuration, then Option 4, then Option 3.
- To ensure that we do not loop before reaching a possible accept configuration, we parse the computational tree with *breadth first search*:

Nondeterministic TM

More on the address tape:

- For each Configuration C_i in N, there are multiple possibilities for following configurations.
- Let b be the largest number of possible following configurations in δ.
- Then a string over $\{1, \dots b\}$ represents a sequence of choices; e.g., 143 represents that we choose Option 1 after start configuration, then Option 4, then Option 3.
- To ensure that we do not loop before reaching a possible accept configuration, we parse the computational tree with *breadth first search*:
- we order the addresses (strings over $\{1, \dots b\}$) as follows (lexicographic ordering): $1, 2, \dots, b, 11, 12, 13, \dots, 1b, 21, 22, \dots, bb, 111, \dots$

Nondeterministic TM

Given NTM N, we construct a 3-tape TM M as follows:
M=on input w:

1. Initiate input tape with input w, simulation and address tapes are empty.

Nondeterministic TM

Given NTM N, we construct a 3-tape TM M as follows:
M=on input w:

1. Initiate input tape with input w, simulation and address tapes are empty.
2. Copy input tape to simulation tape, initialize address tape to ε.

Nondeterministic TM

Given NTM N, we construct a 3-tape TM M as follows:
M=on input w:

1. Initiate input tape with input w, simulation and address tapes are empty.
2. Copy input tape to simulation tape, initialize address tape to ε.
3. Use simulation tape to simulate N with input w. For each computation step, consult next symbol on address for which option to choose. If address is invalid (number corresponds to a nonexistent option), or this leads to a rejecting configuration, go to 4. If accepting configuration, *accept*.

Nondeterministic TM

Given NTM N, we construct a 3-tape TM M as follows:
M=on input w:

1. Initiate input tape with input w, simulation and address tapes are empty.
2. Copy input tape to simulation tape, initialize address tape to ε.
3. Use simulation tape to simulate N with input w. For each computation step, consult next symbol on address for which option to choose. If address is invalid (number corresponds to a nonexistent option), or this leads to a rejecting configuration, go to 4. If accepting configuration, *accept*.
4. Replace address tape's contents with next address according to our ordering. Go to 2.

Nondeterministic TM

Theorem

A language is Turing recognizable [decidable] if and only if some nondeterministic Turing machine recognizes [decides] it.

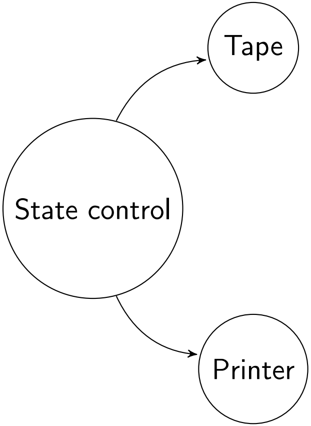
Nondeterministic TM

Theorem

A language is Turing recognizable [decidable] if and only if some nondeterministic Turing machine recognizes [decides] it.

A little extra work needed for decidability result! $\rightarrow$ Exercise 3.3 in book!

Enumerators



- An enumerator is a slightly altered Turing machine:
- It has a working tape and an attached printer
- initializes with an empty working tape, taking no input
- throughout its computation, it can output strings using the printer
- If the enumerator does not halt, it can potentially output infinitely many strings

Enumerators

An enumerator *enumerates* a word iff at some point it prints it.

Enumerators

An enumerator *enumerates* a word iff at some point it prints it.

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Enumerators

An enumerator *enumerates* a word iff at some point it prints it.

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Assume we have an enumerator E. We want to construct a Turing machine A that accepts all the words that E enumerates.

Enumerators

An enumerator *enumerates* a word iff at some point it prints it.

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Assume we have an enumerator E. We want to construct a Turing machine A that accepts all the words that E enumerates.

$A =$ On input w

1. Run E. Every time E prints a string, compare to w.
2. If w appears in the output of E, *accept*.

Enumerators

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Now, let A be a Turing machine. We want to construct an enumerator that enumerates $L(A)$.

Enumerators

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Now, let A be a Turing machine. We want to construct an enumerator that enumerates $L(A)$.
- Le Σ be the alphabet of $L(A)$. Then we can order all strings in Σ^* (first list all strings of length 1, then of length 2, etc).

Enumerators

Theorem

A language is Turing-recognizable iff there exists an enumerator that enumerates it.

Proof:

- Now, let A be a Turing machine. We want to construct an enumerator that enumerates $L(A)$.
- Le Σ be the alphabet of $L(A)$. Then we can order all strings in Σ^* (first list all strings of length 1, then of length 2, etc). Label them $s_1, s_2, s_3, \ldots$. Then we can construct an enumerator:

$E =$ Ingore input

1. Repeat for $i = 1, 2, 3, \ldots$.
2. Run A on $s_1, \ldots, s_i$ for i steps.
3. If any computation accepts, print corresponding s_j.