形式语言与计算复杂性

第 3 章 Computability Theory

3.1 The Church-Turing Thesis

黄文超

https://faculty.ustc.edu.cn/huangwenchao

$\longrightarrow$ 教学课程 $\longrightarrow$ 形式语言与计算复杂性

3.1 The Church-Turing Thesis

Outline

3.1 The Church-Turing Thesis

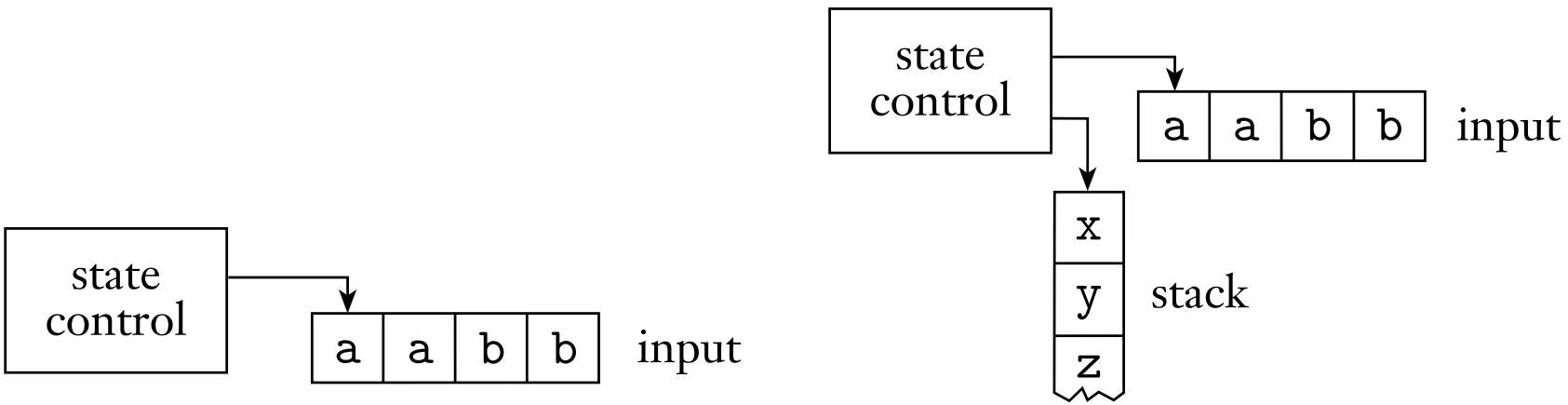
Outline

3.1 The Church-Turing Thesis

Introduction

回顾:

- *Finite automata* are good models for devices that have a *small amount of memory*
- *Pushdown automata* are good models for devices that have an *unlimited memory* that is usable only in the last in, first out manner of a *stack*



待解决问题: Some very *simple tasks* are beyond the capabilities of these models

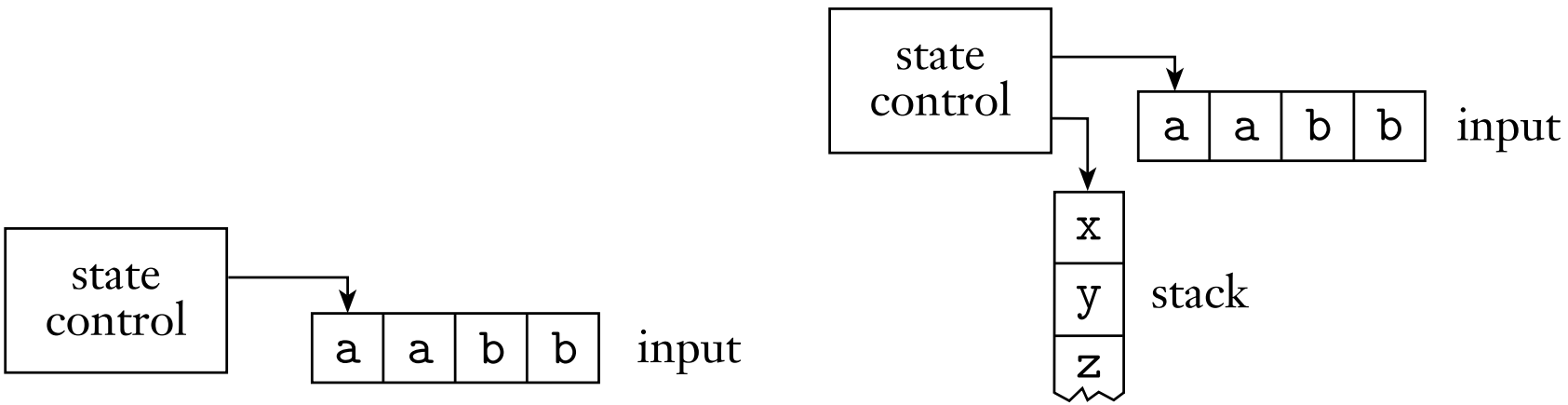
解决方案: Define new models: *Turing Machines*

3.1 The Church-Turing Thesis

Introduction

回顾:

- *Finite automata* are good models for devices that have a *small amount of memory*
- *Pushdown automata* are good models for devices that have an *unlimited memory* that is usable only in the last in, first out manner of a *stack*



待解决问题: Some very *simple tasks* are beyond the capabilities of these models

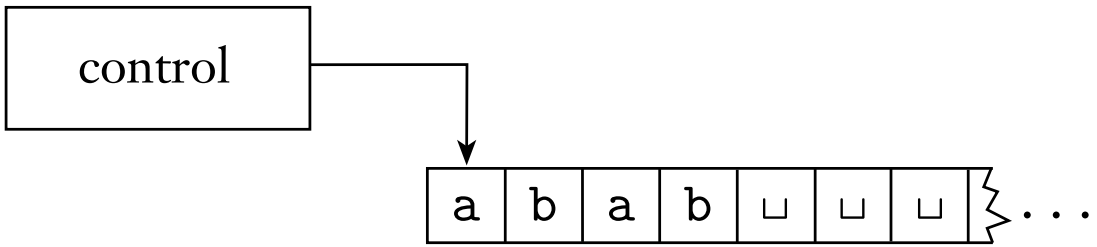
解决方案: Define new models: *Turing Machines*

3.1 The Church-Turing Thesis

Outline

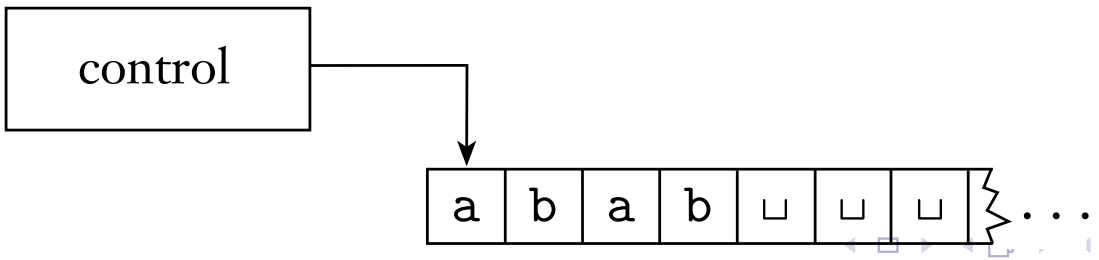
问: What is a Turing machine?
答:

- a much more powerful *model*, first proposed by *Alan Turing* in 1936
- uses an *infinite tape* as its unlimited memory
- *can* do everything that a *real computer* can do
- *cannot* solve certain problems



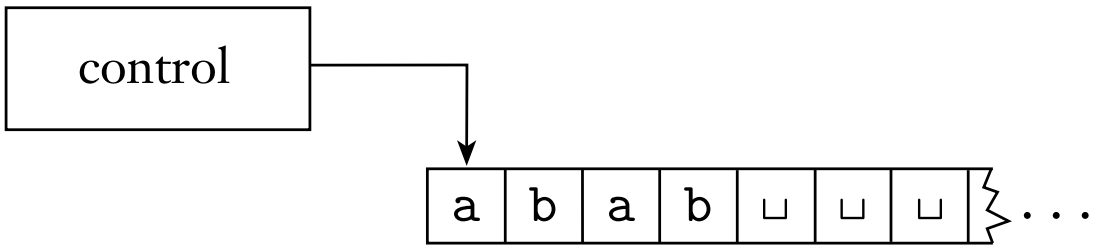
问: How does a Turing machine process input string?
答:

- *Initially*, the *tape* contains only the *input string* and is *blank everywhere* else
- If the machine needs to store information, it may *write* this information on the tape.
- To *read* the information that it has written, the machine can *move its head* back over it
- The outputs *accept* and *reject* are obtained by *entering designated accepting and rejecting states*.
- If it doesn′ t enter an accepting or a rejecting state, it will *go on forever*, never halting.



问: *Differences* with finite automata?
答:

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



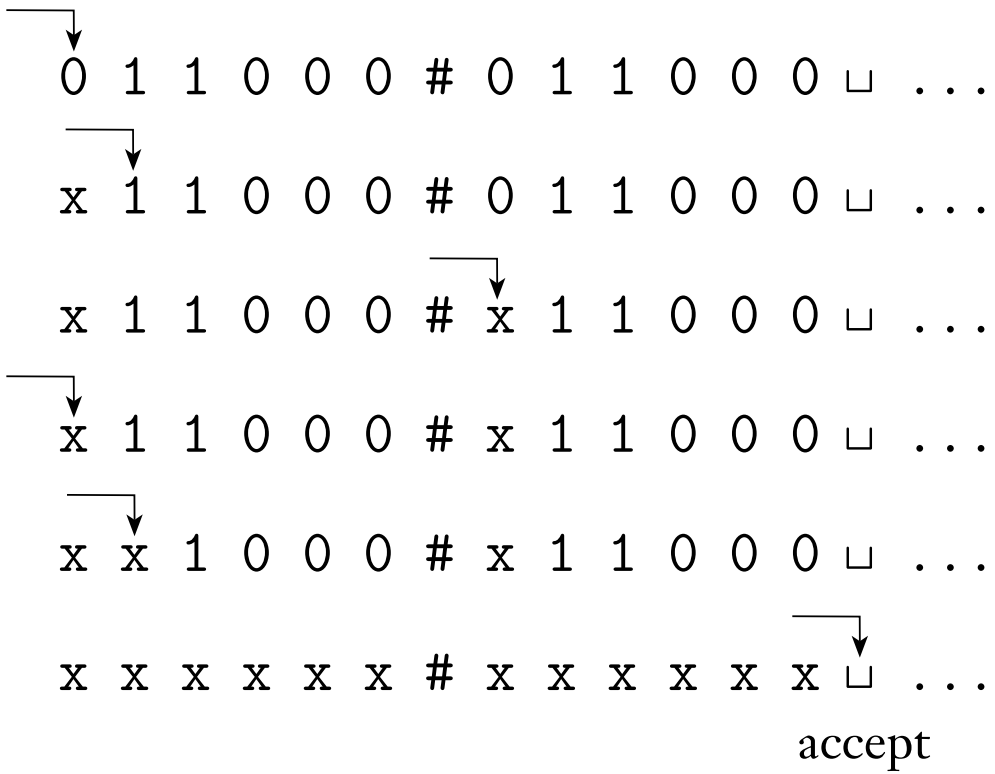
例: a Turing machine M_1 for testing membership in the language $B = \{w\#w \mid w \in \{0,1\}^*\}$

M_1 = "On input string w:

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

例: a Turing machine M_1 for testing membership in the language $B = \{w\#w \mid w \in \{0,1\}^*\}$

M_1 = "On input string w:



3.1 The Church-Turing Thesis

Outline

Turing Machine

A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all 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. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.
5. $q_0 \in Q$ is the start state
6. $q_{\text{accept}} \in Q$ is the accept state
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$

定义: Configuration

As a Turing machine computes, changes occur in

- the *current state*
- the *current tape contents*
- the *current head location*

A setting of *these three items* is called a *configuration* of the Turing machine.

例: $1011q_701111$ represents the configuration when the *tape* is 101101111, the *current state* is q_7, and the *head* is currently on the second 0.

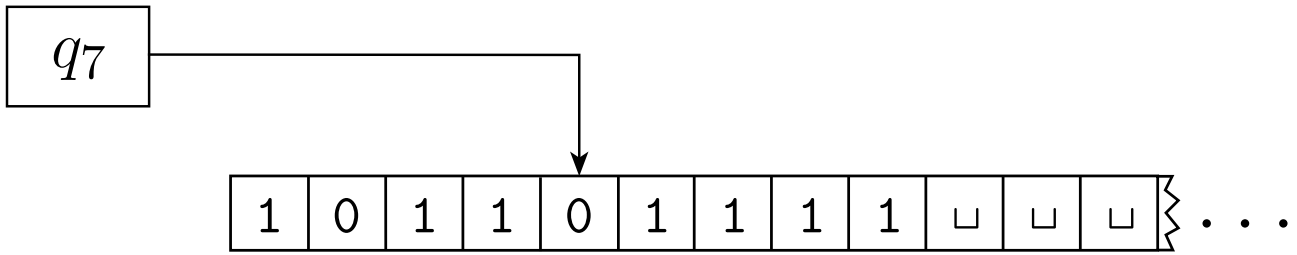
定义: Configuration

As a Turing machine computes, changes occur in

- the *current state*
- the *current tape contents*
- the *current head location*

A setting of *these three items* is called a *configuration* of the Turing machine.

例: $1011q_701111$ represents the configuration when the *tape* is 101101111, the *current state* is q_7, and the *head* is currently on the second 0.



定义: Yields

Say that configuration C_1 *yields* configuration C_2 if the Turing Machine can legally go from C_1 to C_2 in a *single step*.

例: Suppose that we have a, b, and c in Γ,

- as well as u and v in Γ^* and states q_i and q_j.
- In that case, $ua\ q_i\ bv$ and $u\ q_j\ acv$ are two configurations.

Moves leftward:

$ua\ q_i\ bv$ *yields* $u\ q_j\ acv$, if in the transition function $\delta(q_i, b) = (q_j, c, L)$

Moves rightward:

$ua\ q_i\ bv$ *yields* $uac\ q_j\ v$, if $\delta(q_i, b) = (q_j, c, R)$

定义: Yields

Say that configuration C_1 *yields* configuration C_2 if the Turing Machine can legally go from C_1 to C_2 in a *single step*.

例: Suppose that we have a, b, and c in Γ,

- as well as u and v in Γ^* and states q_i and q_j.
- In that case, $ua\ q_i\ bv$ and $u\ q_j\ acv$ are two configurations.

Moves leftward:
$ua\ q_i\ bv$ *yields* $u\ q_j\ acv$, if in the transition function $\delta(q_i, b) = (q_j, c, L)$

Moves rightward:
$ua\ q_i\ bv$ *yields* $uac\ q_j\ v$, if $\delta(q_i, b) = (q_j, c, R)$

定义: Yields

Say that configuration C_1 *yields* configuration C_2 if the Turing Machine can legally go from C_1 to C_2 in a *single step*.

例: Suppose that we have a, b, and c in Γ,

- as well as u and v in Γ^* and states q_i and q_j.
- In that case, $ua\ q_i\ bv$ and $u\ q_j\ acv$ are two configurations.

Moves leftward:
$ua\ q_i\ bv$ *yields* $u\ q_j\ acv$, if in the transition function $\delta(q_i, b) = (q_j, c, L)$

Moves rightward:
$ua\ q_i\ bv$ *yields* $uac\ q_j\ v$, if $\delta(q_i, b) = (q_j, c, R)$

定义: Yields

Say that configuration C_1 *yields* configuration C_2 if the Turing Machine can legally go from C_1 to C_2 in a *single step*.

例: Suppose that we have a, b, and c in Γ,

- as well as u and v in Γ^* and states q_i and q_j.
- In that case, $ua\ q_i\ bv$ and $u\ q_j\ acv$ are two configurations.

Moves leftward:
$ua\ q_i\ bv$ *yields* $u\ q_j\ acv$, if in the transition function $\delta(q_i, b) = (q_j, c, L)$

Moves rightward:
$ua\ q_i\ bv$ *yields* $uac\ q_j\ v$, if $\delta(q_i, b) = (q_j, c, R)$

定义: Start/Accepting/Rejecting/Halting Configuration

- *start configuration* of M on input w: is the configuration $q_0\ w$, which indicates that the machine is in the *start state* q_0 with its head at the *leftmost position* on the tape
- *accepting configuration*: the state of the configuration is q_{accept}
- *rejecting configuration*: the state of the configuration is q_{reject}
- *halting configurations*: accepting and rejecting configurations

定义: Accept

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}
3. C_k is an *accepting configuration*

定义: Start/Accepting/Rejecting/Halting Configuration

- *start configuration* of M on input w: is the configuration $q_0\ w$, which indicates that the machine is in the *start state* q_0 with its head at the *leftmost position* on the tape
- *accepting configuration*: the state of the configuration is q_{accept}
- *rejecting configuration*: the state of the configuration is q_{reject}
- *halting configurations*: accepting and rejecting configurations

定义: Accept

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}
3. C_k is an *accepting configuration*

定义: Language, Recognize

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

定义: Turing-recognizable

Call a *language Turing-recognizable* if some Turing machine *recognizes* it

问: What are the outcomes of a Turing machine?
答: The machine may *accept*, *reject*, or *loop*.

问: What does *loop* mean?
答: the machine simply *does not halt*

问: Why introducing *loop*?
答: We prefer that a Turing machine does not loop

问: How? 答: 见下页

定义: Language, Recognize

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

定义: Turing-recognizable

Call a *language Turing-recognizable* if some Turing machine *recognizes* it

问: What are the outcomes of a Turing machine?
答: The machine may *accept*, *reject*, or *loop*.

问: What does *loop* mean?
答: the machine simply *does not halt*

问: Why introducing *loop*?
答: We prefer that a Turing machine does not loop

问: How? 答: 见下页

定义: Language, Recognize

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

定义: Turing-recognizable

Call a *language Turing-recognizable* if some Turing machine *recognizes* it

问: What are the outcomes of a Turing machine?

答: The machine may *accept*, *reject*, or *loop*.

问: What does *loop* mean?

答: the machine simply *does not halt*

问: Why introducing *loop*?

答: We prefer that a Turing machine does not loop

问: How? 答: 见下页

定义: Language, Recognize

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

定义: Turing-recognizable

Call a *language Turing-recognizable* if some Turing machine *recognizes* it

问: What are the outcomes of a Turing machine?
答: The machine may *accept*, *reject*, or *loop*.

问: What does *loop* mean?
答: the machine simply *does not halt*

问: Why introducing *loop*?
答: We prefer that a Turing machine does not loop

问: How? 答: 见下页

定义: Language, Recognize

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

定义: Turing-recognizable

Call a *language Turing-recognizable* if some Turing machine *recognizes* it

问: What are the outcomes of a Turing machine?
答: The machine may *accept*, *reject*, or *loop*.

问: What does *loop* mean?
答: the machine simply *does not halt*

问: Why introducing *loop*?
答: We prefer that a Turing machine does not loop

问: How? 答: 见下页

定义: Language, Recognize

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

定义: Turing-recognizable

Call a *language Turing-recognizable* if some Turing machine *recognizes* it

问: What are the outcomes of a Turing machine?
答: The machine may *accept*, *reject*, or *loop*.

问: What does *loop* mean?
答: the machine simply *does not halt*

问: Why introducing *loop*?
答: We prefer that a Turing machine does not loop

问: How? 答: 见下页

定义: Decide

- *Deciders* are Turing machines that *always* make a decision to *accept* or *reject*.
- A *decider* that *recognizes* some language also is said to *decide* that language.

定义: Turing-decidable

Call a language *Turing-decidable* or simply *decidable* if some Turing machine *decides* it

问: Then, how?
答: Answer whether a language is Turing-decidable

定义: Decide

- *Deciders* are Turing machines that *always* make a decision to *accept* or *reject*.
- A *decider* that *recognizes* some language also is said to *decide* that language.

定义: Turing-decidable

Call a language *Turing-decidable* or simply *decidable* if some Turing machine *decides* it

问: Then, how?
答: Answer whether a language is Turing-decidable

定义: Decide

- *Deciders* are Turing machines that *always* make a decision to *accept* or *reject*.
- A *decider* that *recognizes* some language also is said to *decide* that language.

定义: Turing-decidable

Call a language *Turing-decidable* or simply *decidable* if some Turing machine *decides* it

问: Then, how?
答: Answer whether a language is Turing-decidable

3.1 The Church-Turing Thesis

Outline

例 1

A Turing Machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$

例 1

A Turing Machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$

M_2 = "On input string w:

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, *reject*.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1."

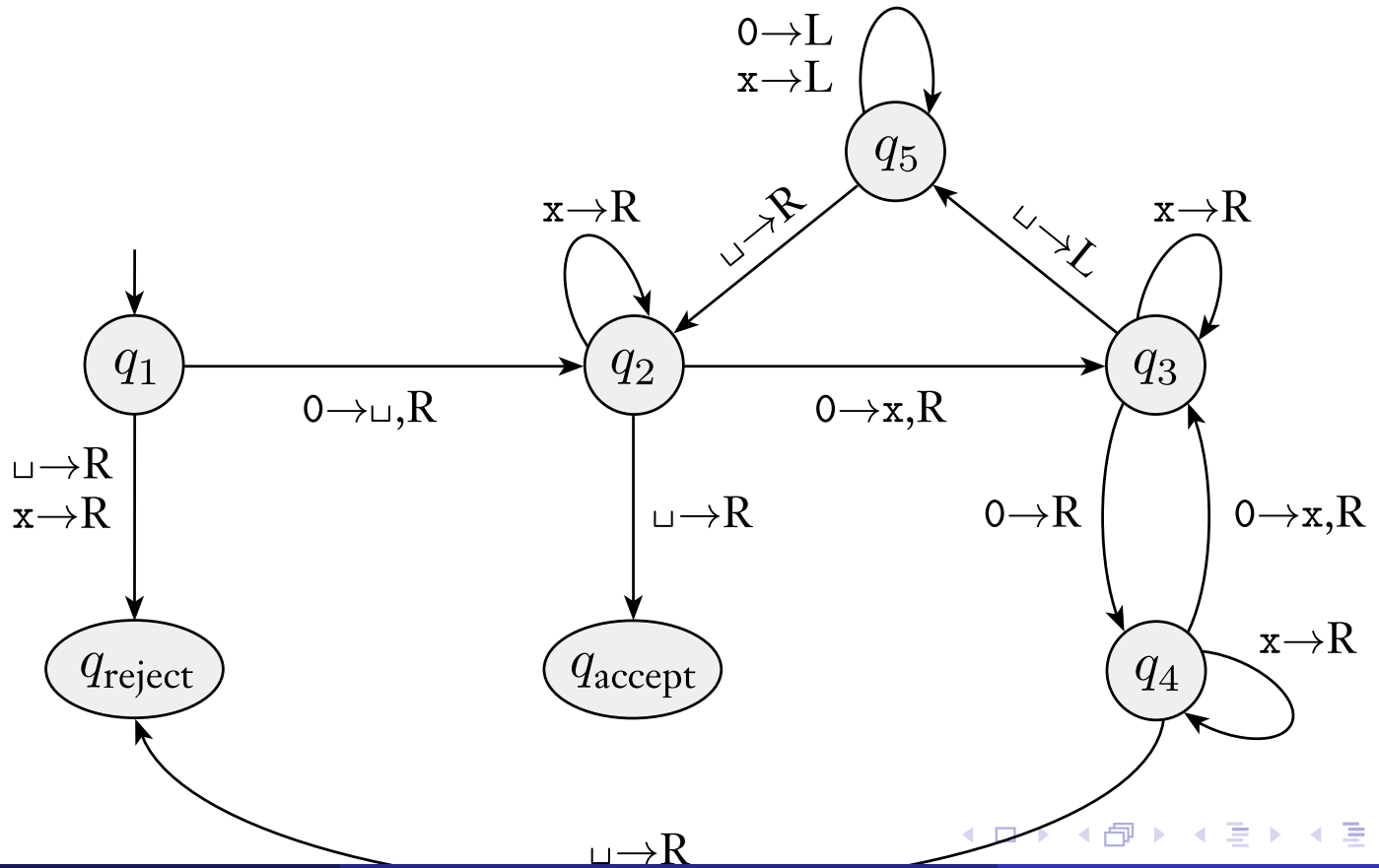
例 1

A Turing Machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$

$q_1 0000$
$\sqcup q_2 000$
$\sqcup \mathrm{x} q_3 00$
$\sqcup \mathrm{x} 0 q_4 0$
$\sqcup \mathrm{x} 0 \mathrm{x} q_3 \sqcup$
$\sqcup \mathrm{x} 0 q_5 \mathrm{x} \sqcup$
$\sqcup \mathrm{x} q_5 0 \mathrm{x} \sqcup$
$\sqcup q_5 \mathrm{x} 0 \mathrm{x} \sqcup$
$q_5 \sqcup \mathrm{x} 0 \mathrm{x} \sqcup$
$\sqcup q_2 \mathrm{x} 0 \mathrm{x} \sqcup$
$\sqcup \mathrm{x} q_2 0 \mathrm{x} \sqcup$
$\sqcup \mathrm{xx} q_3 \mathrm{x} \sqcup$
$\sqcup \mathrm{xxx} q_3 \sqcup$
$\sqcup \mathrm{xx} q_5 \mathrm{x} \sqcup$
$\sqcup \mathrm{x} q_5 \mathrm{xx} \sqcup$
$\sqcup q_5 \mathrm{xxx} \sqcup$
$q_5 \sqcup \mathrm{xxx} \sqcup$
$\sqcup q_2 \mathrm{xxx} \sqcup$
$\sqcup \mathrm{x} q_2 \mathrm{xx} \sqcup$
$\sqcup \mathrm{xx} q_2 \mathrm{x} \sqcup$
$\sqcup \mathrm{xxx} q_2 \sqcup$
$\sqcup \mathrm{xxx} \sqcup q_{\text{accept}}$

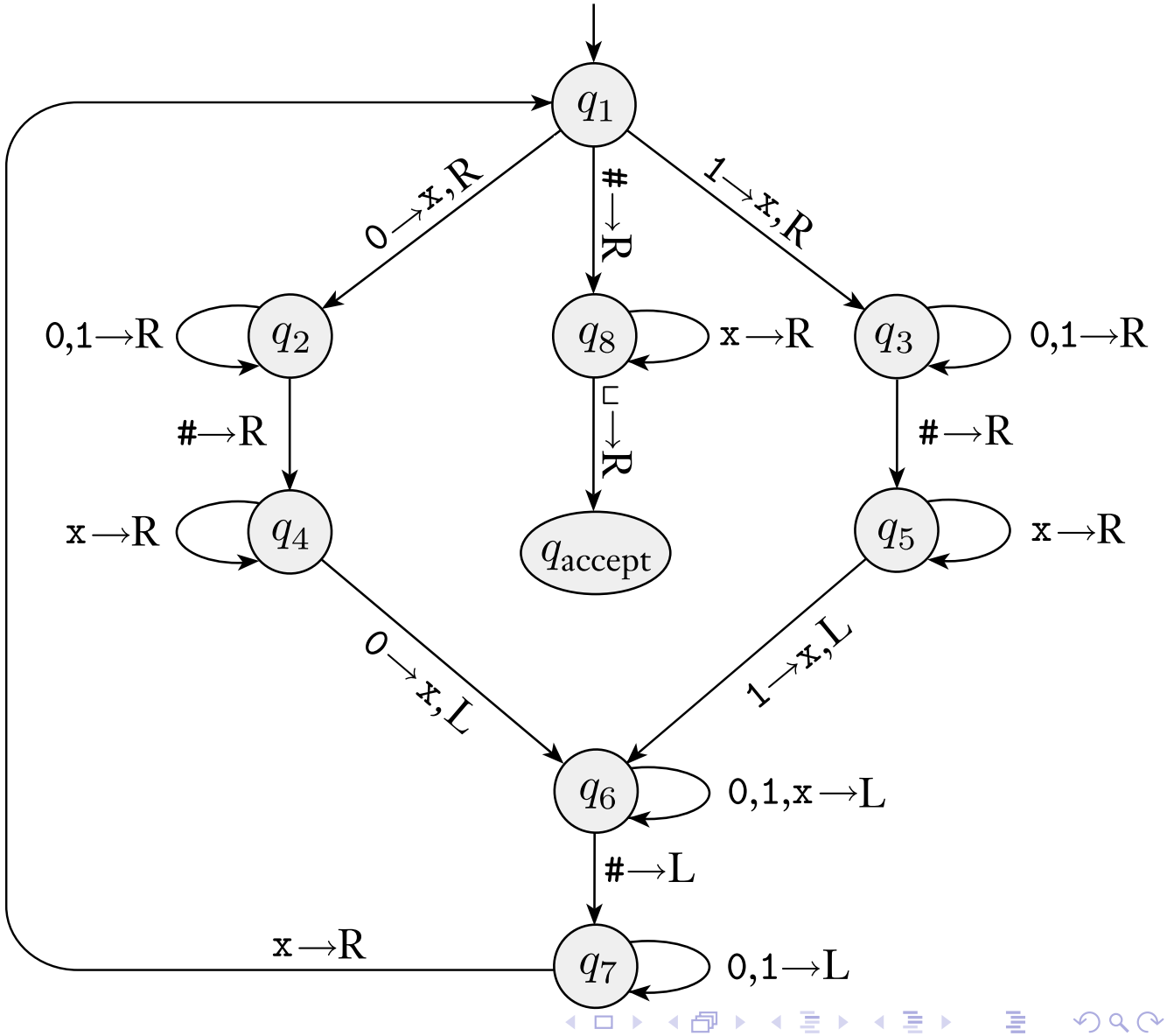
例 1

A Turing Machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$



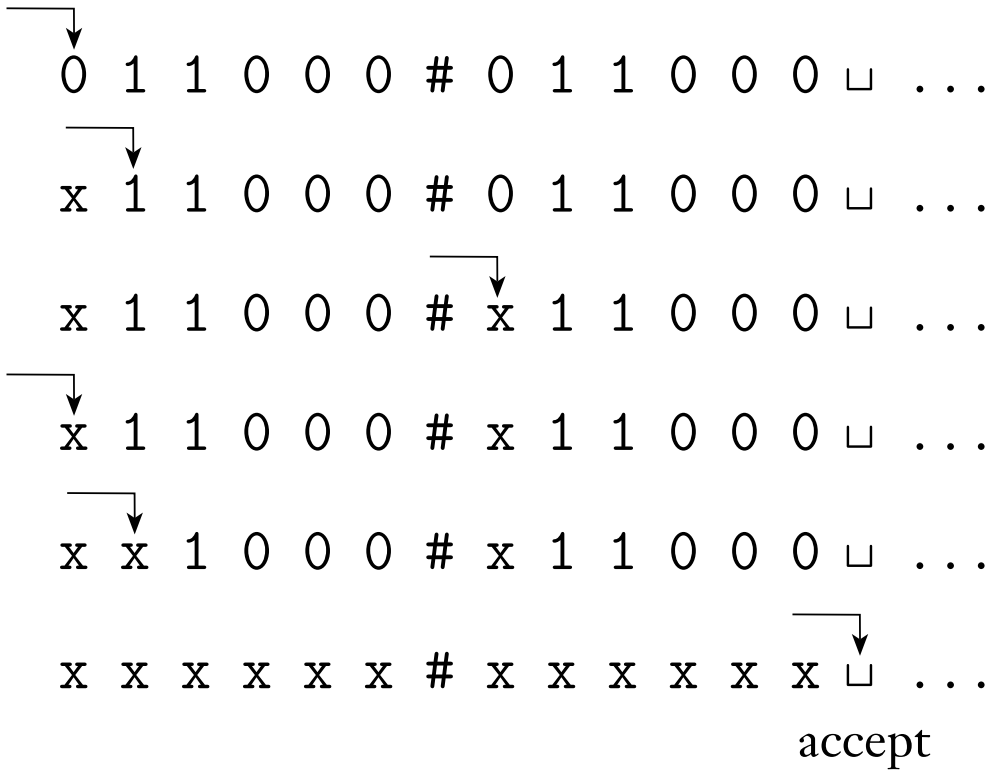
例 2

A Turing Machine M_1 for testing membership in the language $B = \{w\#w \mid w \in \{0,1\}^*\}$



例 2

A Turing Machine M_1 for testing membership in the language $B = \{w\#w \mid w \in \{0,1\}^*\}$



3.1 The Church-Turing Thesis

Outline

回顾: Equivalence

- DFAs and NFAs *are equivalent* in language recognition power
- DPDAs and PDAs *are not equivalent* in language recognition power

问: What are the variants of a Turing machine?
答: Multitape Turing Machine, Nondeterministic Turing Machine, Enumerator, ...

问: Do the original model and its reasonable variants all have the same power?
答: Yes

问: Why?
答: *Robustness*, i.e., invariance to certain changes

问: How?
答: See the following...

回顾: Equivalence

- DFAs and NFAs *are equivalent* in language recognition power
- DPDAs and PDAs *are not equivalent* in language recognition power

问: What are the variants of a Turing machine?
答: Multitape Turing Machine, Nondeterministic Turing Machine, Enumerator, ...

问: Do the original model and its reasonable variants all have the same power?
答: Yes

问: Why?
答: *Robustness*, i.e., invariance to certain changes

问: How?
答: See the following...

回顾: Equivalence

- DFAs and NFAs *are equivalent* in language recognition power
- DPDAs and PDAs *are not equivalent* in language recognition power

问: What are the variants of a Turing machine?
答: Multitape Turing Machine, Nondeterministic Turing Machine, Enumerator, ...

问: Do the original model and its reasonable variants all have the same power?
答: Yes

问: Why?
答: *Robustness*, i.e., invariance to certain changes

问: How?
答: See the following...

回顾: Equivalence

- DFAs and NFAs *are equivalent* in language recognition power
- DPDAs and PDAs *are not equivalent* in language recognition power

问: What are the variants of a Turing machine?
答: Multitape Turing Machine, Nondeterministic Turing Machine, Enumerator, ...

问: Do the original model and its reasonable variants all have the same power?
答: Yes

问: Why?
答: *Robustness*, i.e., invariance to certain changes

问: How?
答: See the following...

回顾: Equivalence

- DFAs and NFAs *are equivalent* in language recognition power
- DPDAs and PDAs *are not equivalent* in language recognition power

问: What are the variants of a Turing machine?
答: Multitape Turing Machine, Nondeterministic Turing Machine, Enumerator, ...

问: Do the original model and its reasonable variants all have the same power?
答: Yes

问: Why?
答: *Robustness*, i.e., invariance to certain changes

问: How?
答: See the following...

Outline

问: What is a *Multitape* Turing Machine?

答: like an ordinary Turing Machine with *several tapes*

- *Each tape* has its own head for *reading* and *writing*.
- *Initially* the *input* appears on *tape 1*, and *the others* start out *blank*.
- The transition function is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

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

where k is the number of tapes

The expression $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$

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

问: What is a *Multitape* Turing Machine?

答: like an ordinary Turing Machine with *several tapes*

- *Each tape* has its own head for *reading* and *writing*.
- *Initially* the *input* appears on *tape 1*, and *the others* start out *blank*.
- The transition function is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

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

where k is the number of tapes

The expression $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$

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

问: What is a *Multitape* Turing Machine?

答: like an ordinary Turing Machine with *several tapes*

- *Each tape* has its own head for *reading* and *writing*.
- *Initially* the *input* appears on *tape 1*, and *the others* start out *blank*.
- The transition function is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

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

where k is the number of tapes

The expression $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$

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

定理

Every multitape Turing machine *has* an *equivalent single-tape* TM

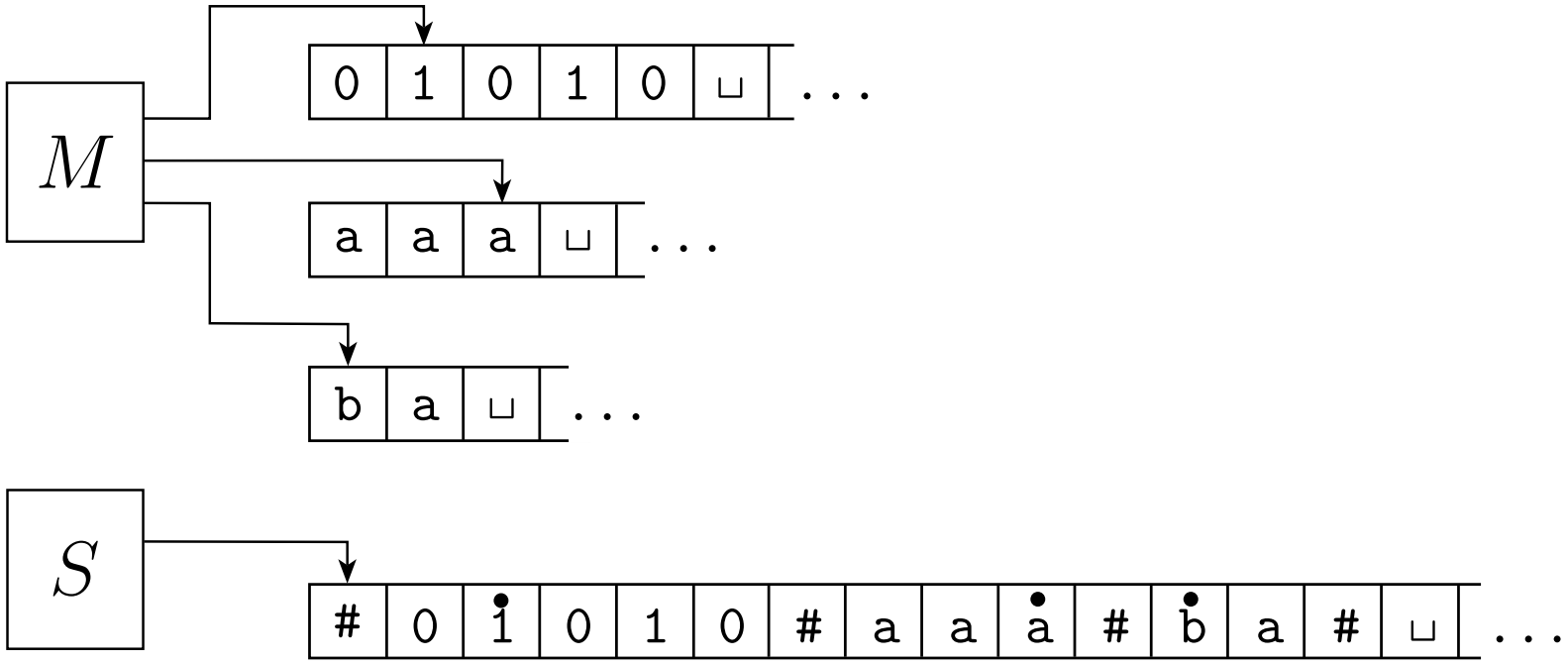
证明思路: (Proof by Construction)

S *simulates* the effect of k tapes by storing their information on its single tape

- *Separate* the contents of the different *tapes*: uses the new symbol # as a delimiter to
- Keep track of the *locations* of the *heads*: writing a *tape symbol* with a *dot above* it to mark the place where the head on that tape would be.

定理

Every multitape Turing machine *has* an *equivalent single-tape* TM



定理

Every multitape Turing machine *has* an *equivalent single-tape* TM

推论

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

3.1 The Church-Turing Thesis

Outline

问: What is a *Nondeterministic* Turing Machine?
答: At any point in a computation, the machine may proceed according to *several possibilities*.

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

If *some* branch of the computation leads to the *accept state*, the machine *accepts* its input.

定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

证明思路: (Proof by Construction)

Simulate any nondeterministic TM N with a deterministic TM D

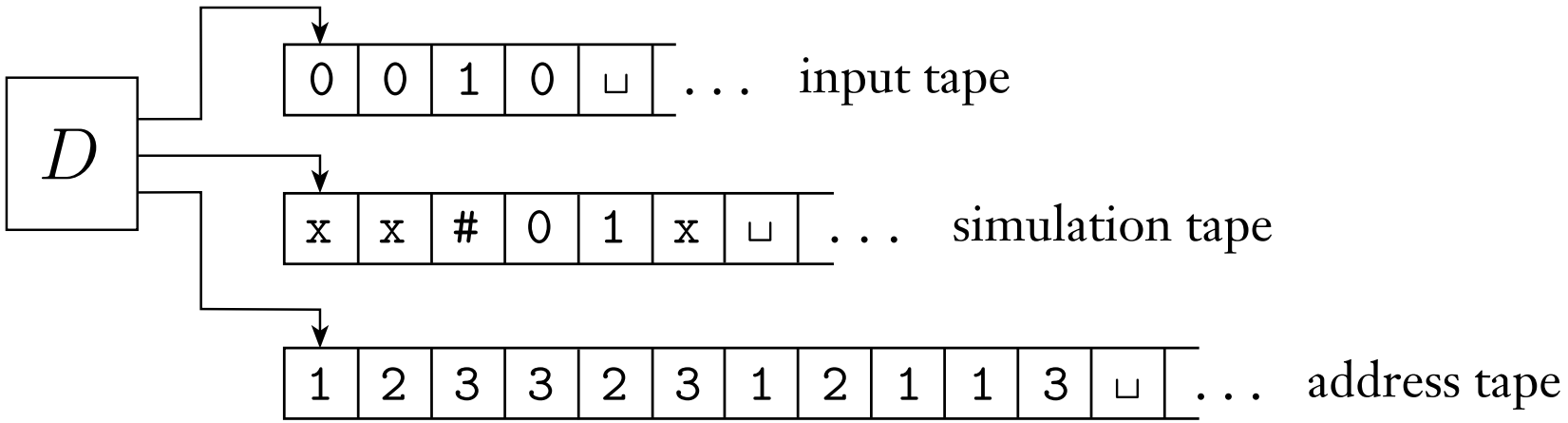
- have D *try all possible branches* of N' s nondeterministic computation
  - If D ever finds the *accept state* on one of these branches, D accepts.
  - Otherwise, D' s simulation will not terminate.

定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

证明思路: (Proof by Construction)

The machine D uses its three tapes

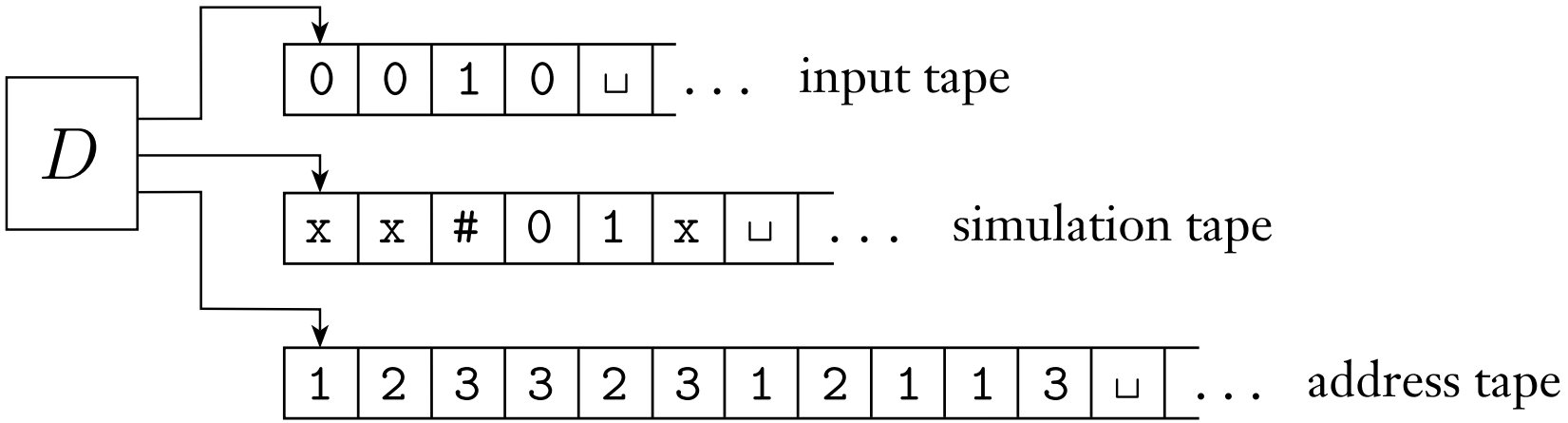


定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

证明思路: (Proof by Construction)

Tape 1 always contains the *input string* and is *never altered*.

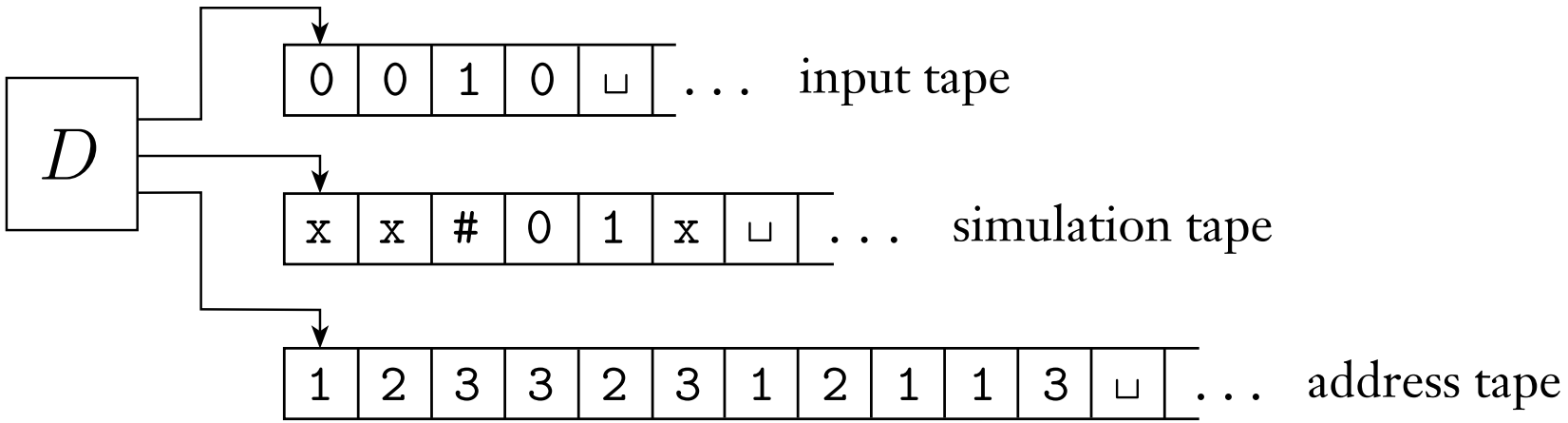


定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

证明思路: (Proof by Construction)

Tape 2 maintains a copy of N' s tape on *some branch* of its nondeterministic computation.

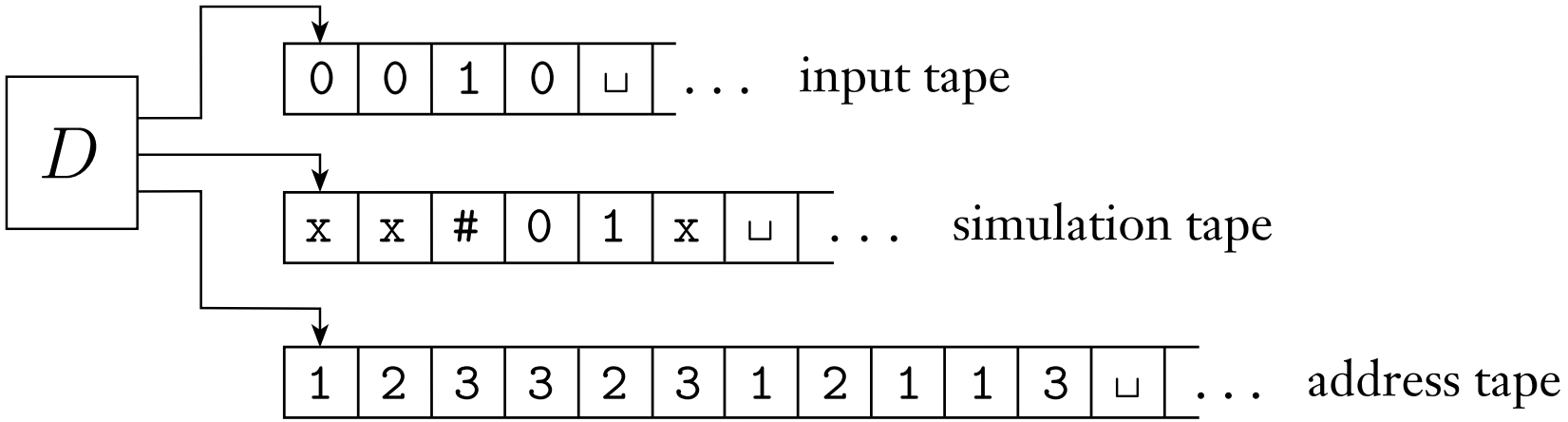


定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

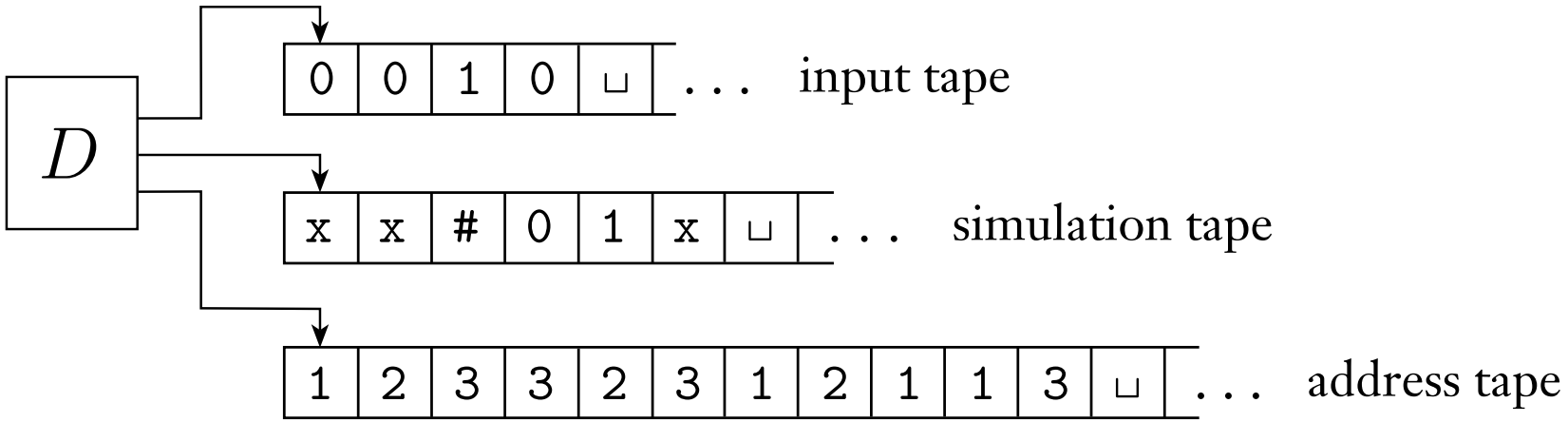
证明思路: (Proof by Construction)

Tape 3 keeps track of D' s location in N' s nondeterministic *computation tree*. The string represents the *branch* of N' s computation.



定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

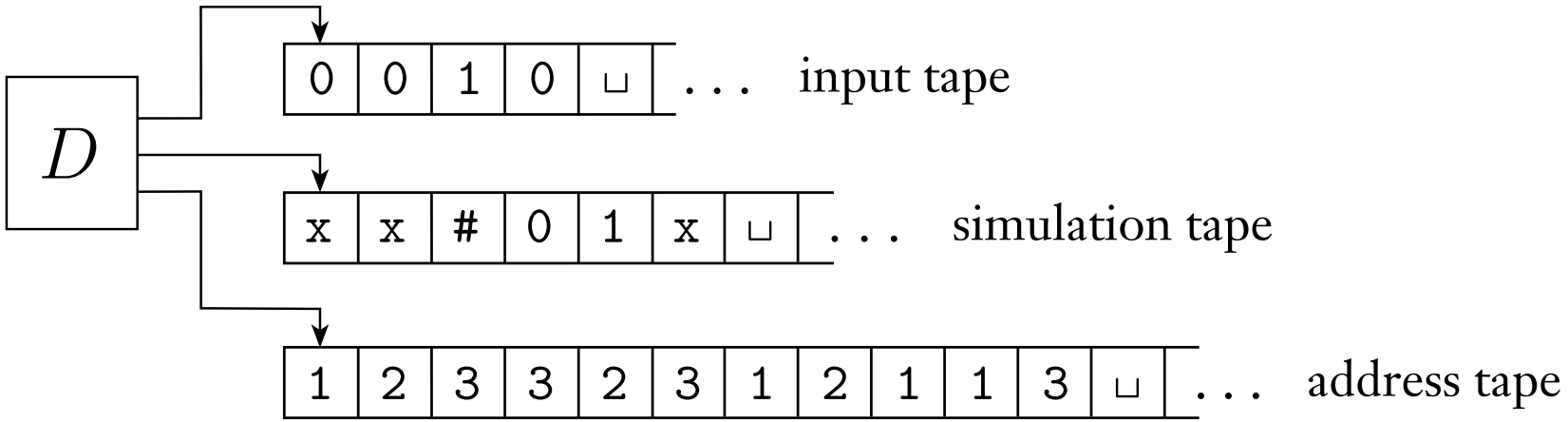


证明: (Proof by Construction)

1. Initially, tape 1 contains the input w, and tapes 2 and 3 are empty.

定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

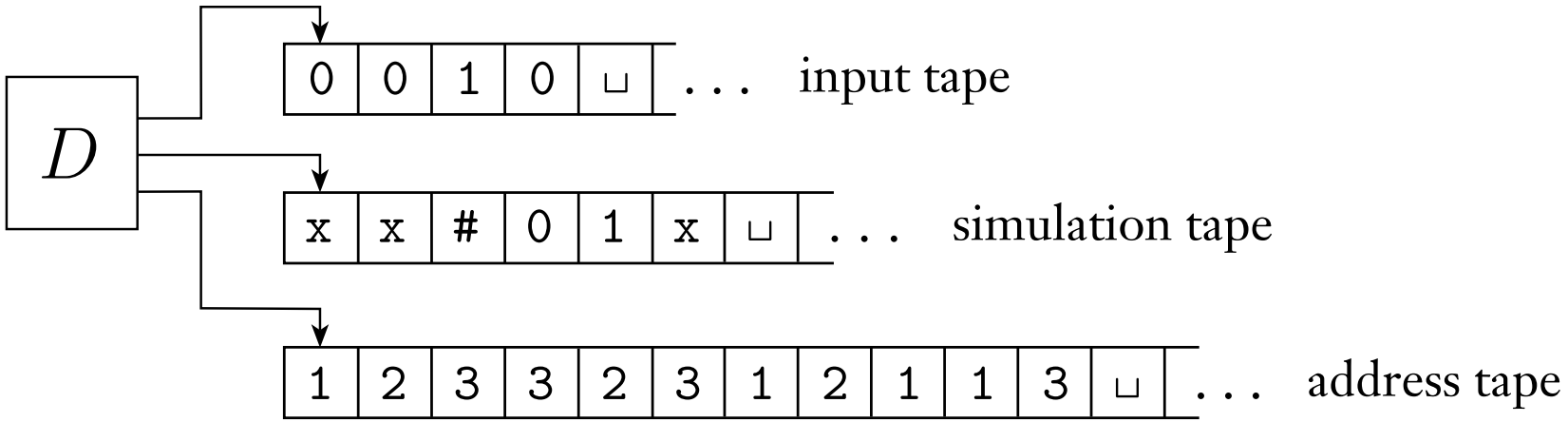


证明: (Proof by Construction)

2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be ε.

定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

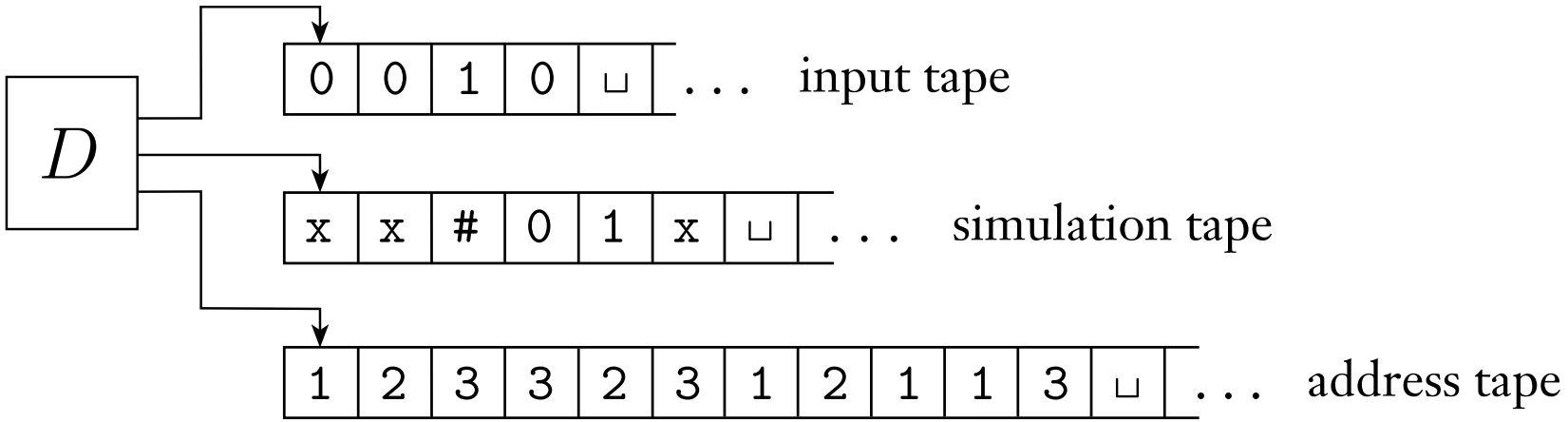


证明: (Proof by Construction)

3. Use tape 2 to simulate N with input w on *one branch* of its nondeterministic computation.

定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine



证明: (Proof by Construction)

4. *Replace* the *string on tape 3* with *the next string* in the string ordering. Simulate the next branch of N' s computation by going to stage 2

定理

Every nondeterministic Turing machine *has* an *equivalent* deterministic Turing Machine

推论 1

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

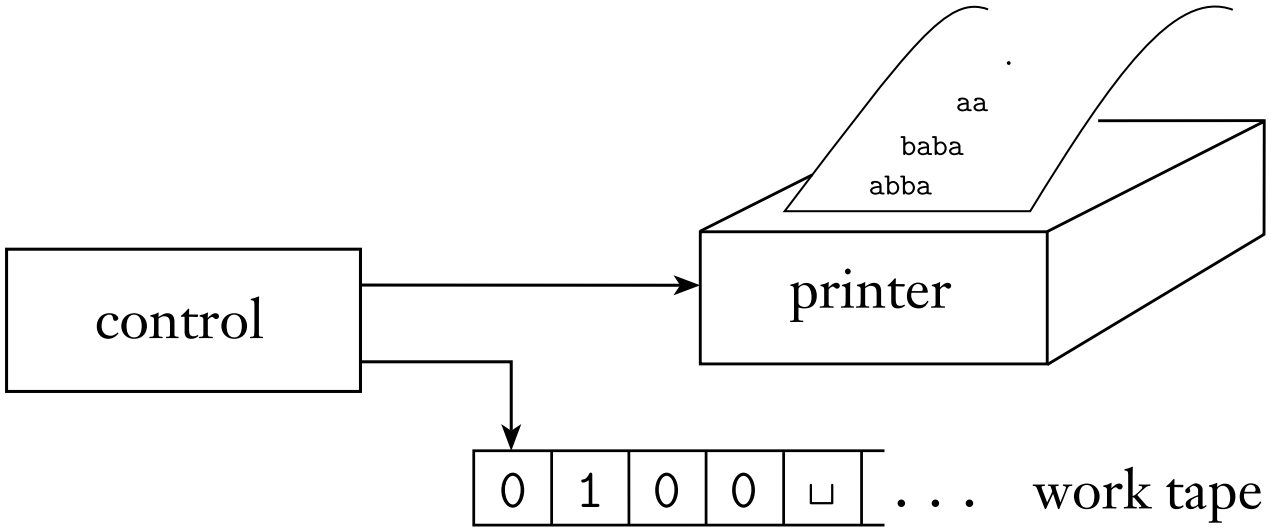
定义: Decider: A special nondeterministic TM

We call a nondeterministic Turing machine a *decider* if *all branches halt on all inputs.*

推论 2

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

Outline



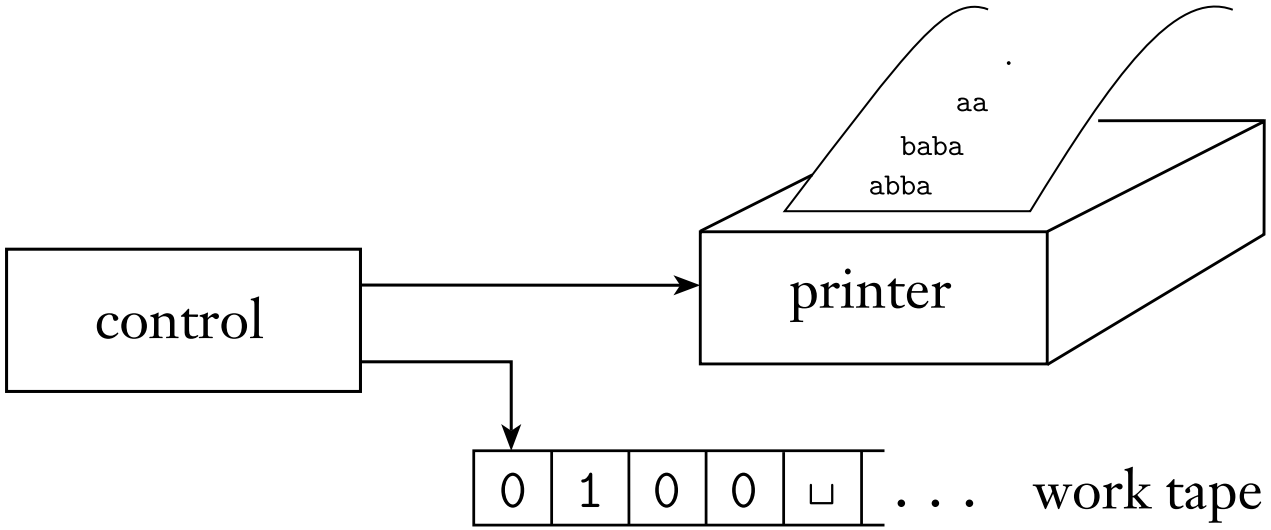
问: What is an Enumerator?

答: An enumerator is a Turing machine with an *attached printer*

问: *How* to use the printer?

答: Every time the Turing machine wants to add a string to the list, it *sends the string* to the printer

- If the enumerator *doesn't halt*, it may print an *infinite* list of strings.
- The *language* enumerated by E is the collection of all the strings that it eventually prints out.



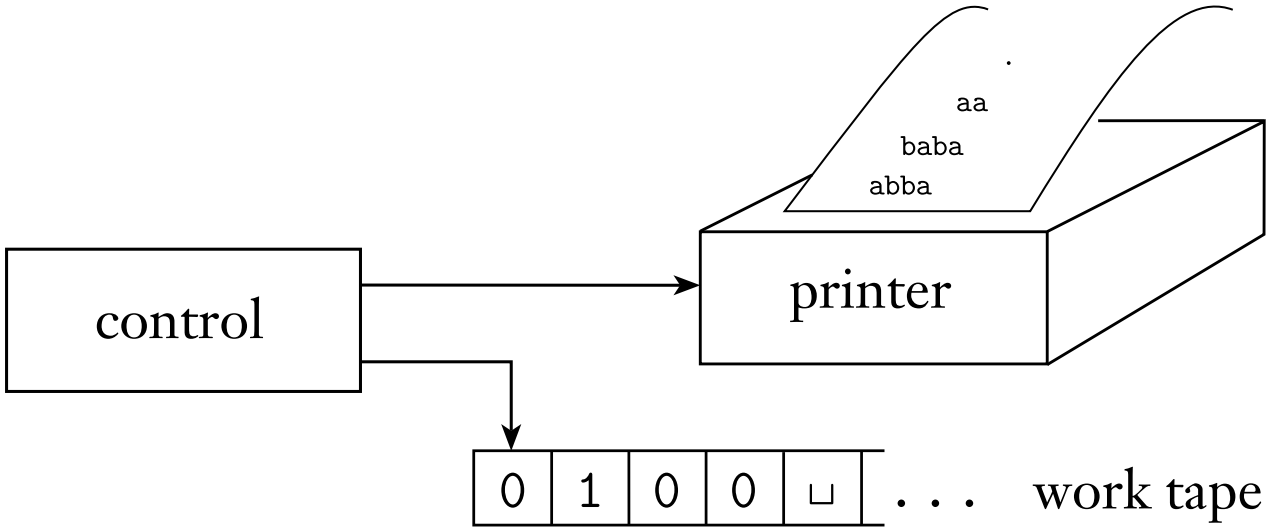
问: What is an Enumerator?

答: An enumerator is a Turing machine with an *attached printer*

问: *How* to use the printer?

答: Every time the Turing machine wants to add a string to the list, it *sends the string* to the printer

- If the enumerator *doesn't halt*, it may print an *infinite* list of strings.
- The *language* enumerated by E is the collection of all the strings that it eventually prints out.



问: What is an Enumerator?

答: An enumerator is a Turing machine with an *attached printer*

问: *How* to use the printer?

答: Every time the Turing machine wants to add a string to the list, it *sends the string* to the printer

- If the enumerator *doesn't halt*, it may print an *infinite* list of strings.
- The *language* enumerated by E is the collection of all the strings that it eventually prints out.

定理

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

定理

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

证明: (Prove by Construction)

Part 1: if we have an enumerator E that enumerates a language A, prove that a TM M recognizes A

定理

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

证明: (Prove by Construction)

Part 1: if we have an enumerator E that enumerates a language A, prove that a TM M recognizes A

M="on input w:

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

定理

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

证明: (Prove by Construction)

Part 2: if TM M recognizes a language A, we can construct the following enumerator E for A.

定理

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

证明: (Prove by Construction)

Part 2: if TM M recognizes a language A, we can construct the following enumerator E for A.

Say that $s_1, s_2, s_3, \ldots$ is a list of all possible strings in Σ^*

定理

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

证明: (Prove by Construction)

Part 2: if TM M recognizes a language A, we can construct the following enumerator E for A.
Say that $s_1, s_2, s_3, \ldots$ is a list of all possible strings in Σ^*

$E =$ "Ignore the input.

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

3.1 The Church-Turing Thesis

Outline

回顾: Several variants of the Turing machine model are equivalent in power.

问: Why are they equivalent?
答: All *share* the essential *feature* of Turing machines—namely, *unrestricted access to unlimited memory*

问: Any examples for understanding the phenomenon?
答: Pascal, LISP, C, C++...

问: What can be implied from the phenomenon?
答: 见下页 (*ALgorithm*)

回顾: Several variants of the Turing machine model are equivalent in power.

问: Why are they equivalent?
答: All *share* the essential *feature* of Turing machines—namely, *unrestricted access to unlimited memory*

问: Any examples for understanding the phenomenon?
答: Pascal, LISP, C, C++...

问: What can be implied from the phenomenon?
答: 见下页 (*ALgorithm*)

回顾: Several variants of the Turing machine model are equivalent in power.

问: Why are they equivalent?
答: All *share* the essential *feature* of Turing machines—namely, *unrestricted access to unlimited memory*

问: Any examples for understanding the phenomenon?
答: Pascal, LISP, C, C++...

问: What can be implied from the phenomenon?
答: 见下页 (*ALgorithm*)

回顾: Several variants of the Turing machine model are equivalent in power.

问: Why are they equivalent?
答: All *share* the essential *feature* of Turing machines—namely, *unrestricted access to unlimited memory*

问: Any examples for understanding the phenomenon?
答: Pascal, LISP, C, C++...

问: What can be implied from the phenomenon?
答: 见下页 (*ALgorithm*)

3.1 The Church-Turing Thesis

Outline

问: What can a Turing machine do?
猜想: It seems to achieve an *algorithm*

问: How to define an algorithm?
答: A story to tell before definition

问: Summary of the story?
答: The notion of algorithm itself was not defined *precisely* until the *twentieth century*

问: Why do we need *precise definition* of *algorithm*?
答: It is a *crucial step* before solving certain problems, e.g., the *10th Hilbert's Problem*.

问: What can a Turing machine do?
猜想: It seems to achieve an *algorithm*

问: How to define an algorithm?
答: A story to tell before definition

问: Summary of the story?
答: The notion of algorithm itself was not defined *precisely* until the *twentieth century*

问: Why do we need *precise definition* of *algorithm*?
答: It is a *crucial step* before solving certain problems, e.g., the *10th Hilbert's Problem*.

问: What can a Turing machine do?
猜想: It seems to achieve an *algorithm*

问: How to define an algorithm?
答: A story to tell before definition

问: Summary of the story?
答: The notion of algorithm itself was not defined *precisely* until the *twentieth century*

问: Why do we need *precise definition* of *algorithm*?
答: It is a *crucial step* before solving certain problems, e.g., the *10th Hilbert's Problem*.

问: What can a Turing machine do?
猜想: It seems to achieve an *algorithm*

问: How to define an algorithm?
答: A story to tell before definition

问: Summary of the story?
答: The notion of algorithm itself was not defined *precisely* until the *twentieth century*

问: Why do we need *precise definition* of *algorithm*?
答: It is a *crucial step* before solving certain problems, e.g., the *10th Hilbert's Problem*.

3.1 The Church-Turing Thesis

Outline

问: What are Hilbert's Problems?

答: 1900, International Congress of Mathematicians

- delivered by mathematician David Hilbert
- 23 mathematical problems
- a challenge for the coming century
- The 10th problem concerned *algorithms*

问: What is the 10th problem?

答: Devise an "algorithm" that tests whether a *polynomial* has an *integral root*.

- $6x^3yz^2 + 3xy^2 - x^3 - 10$ has a root at $x = 5, y = 3$, and $z = 0$
- note: He *did not use* the term *algorithm* but rather "a process according to which it can be determined by a *finite* number of operations."

问: What are Hilbert's Problems?
答: 1900, International Congress of Mathematicians

- delivered by mathematician David Hilbert
- 23 mathematical problems
- a challenge for the coming century
- The 10th problem concerned *algorithms*

问: What is the 10th problem?
答: Devise an "algorithm" that tests whether a *polynomial* has an *integral root*.

- $6x^3yz^2 + 3xy^2 - x^3 - 10$ has a root at $x = 5, y = 3$, and $z = 0$
- note: He *did not use* the term *algorithm* but rather "a process according to which it can be determined by a *finite* number of operations."

问: Does the process/algorithm exists for the 10th problem?
答: No

问: Why?
答: 2 steps:

1. Define algorithm
   - proposed by Alonzo Church and Alan Turing in 1936
2. Prove it by using the definition
   - Matijasevič's theorem

问: Does the process/algorithm exists for the 10th problem?
答: No

问: Why?
答: 2 steps:

1. Define algorithm
   - proposed by Alonzo Church and Alan Turing in 1936
2. Prove it by using the definition
   - Matijasevič's theorem

问: Does the process/algorithm exists for the 10th problem?
答: No

问: Why?
答: 2 steps:

1. Define algorithm
   - proposed by Alonzo Church and Alan Turing in 1936
2. Prove it by using the definition
   - Matijasevič's theorem

问: Does the process/algorithm exists for the 10th problem?
答: No

问: Why?
答: 2 steps:

1. Define algorithm
   - proposed by Alonzo Church and Alan Turing in 1936
2. Prove it by using the definition
   - Matijasevič's theorem

问: How to define algorithm?
答: 2 ways

- *Church* used a *notational system* called the λ-calculus to define algorithms
- *Turing* did it with his "*machines*"

问: What are the connections between the 2 definitions?
答: They are *equivalent*, which is called the ***Church-Turing Thesis***

Intuitive notion of algorithms	equals	*Turing machine algorithms*

问: How to define 10th problem using Turing machine?
答: 见下页

问: How to define algorithm?
答: 2 ways

- *Church* used a *notational system* called the λ-calculus to define algorithms
- *Turing* did it with his "*machines*"

问: What are the connections between the 2 definitions?
答: They are *equivalent*, which is called the ***Church-Turing Thesis***

Intuitive notion of algorithms	equals	*Turing machine algorithms*

问: How to define 10th problem using Turing machine?
答: 见下页

问: How to define algorithm?
答: 2 ways

- *Church* used a *notational system* called the λ-calculus to define algorithms
- *Turing* did it with his "*machines*"

问: What are the connections between the 2 definitions?
答: They are *equivalent*, which is called the ***Church-Turing Thesis***

Intuitive notion of algorithms	equals	*Turing machine algorithms*

问: How to define 10th problem using Turing machine?
答: 见下页

定义: Hilbert's 10th problem

Let $D = \{p \mid p \text{ is a polynomial with an integral root}\}$, whether the set D is *decidable* ?

答: No, D is not decidable.

问: Another problem: is it *recognizable*?

答: Yes

问: Why?

答: Consider a simpler problem:

- Let $D_1 = \{p \mid p \text{ is a polynomial } \textit{over } x \text{ with an integral root}\}$, whether the set D_1 is *recognizable*?

M_1 = "On input $\langle p \rangle$: where p is a polynomial over the variable x.

1. Evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \ldots$. If at any point the polynomial evaluates to 0, *accept*."

定义: Hilbert's 10th problem

Let $D = \{p \mid p \text{ is a polynomial with an integral root}\}$, whether the set D is *decidable* ?

答: No, D is not decidable.

问: Another problem: is it *recognizable*?
答: Yes

问: Why?
答: Consider a simpler problem:

- Let $D_1 = \{p \mid p \text{ is a polynomial } \textit{over } x \text{ with an integral root}\}$, whether the set D_1 is *recognizable*?

M_1 = "On input $\langle p \rangle$: where p is a polynomial over the variable x.

1. Evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \ldots$. If at any point the polynomial evaluates to 0, *accept*."

定义: Hilbert's 10th problem

Let $D = \{p \mid p \text{ is a polynomial with an integral root}\}$, whether the set D is *decidable* ?

答: No, D is not decidable.

问: Another problem: is it *recognizable*?
答: Yes

问: Why?
答: Consider a simpler problem:

- Let $D_1 = \{p \mid p \text{ is a polynomial } \textit{over } x \text{ with an integral root}\}$, whether the set D_1 is *recognizable*?

M_1 = "On input $\langle p \rangle$: where p is a polynomial over the variable x.

1. Evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \ldots$. If at any point the polynomial evaluates to 0, *accept*."

问: Whether D_1 is *decidable*?
答: Yes, because we can *calculate bounds* within which the roots of a *single variable* polynomial must lie, and *restrict the search* to these bounds

问: Whether D is *decidable*?
答: No, because calculating such *bounds* for multivariable polynomials is impossible, by *Matijasevič's theorem*

问: Whether D_1 is *decidable*?
答: Yes, because we can *calculate bounds* within which the roots of a *single variable* polynomial must lie, and *restrict the search* to these bounds

问: Whether D is *decidable*?
答: No, because calculating such *bounds* for multivariable polynomials is impossible, by *Matijasevič's theorem*

3.1 The Church-Turing Thesis

Outline

问: Why introducing Algorithm?
答: We have come to a turning point:

- Focus on *algorithms*
- instead of *Turing Machines*

问: Why?
答: Now, we prefer *Expressiveness* (higher-level description), to *Precision* (lower-level description), in describing algorithms.

问: What is the right level of detail to give when describing such algorithms?
答: *Standardize* the way we describe Turing machine algorithms

问: How?
答: 见下页

问: Why introducing Algorithm?
答: We have come to a turning point:

- Focus on *algorithms*
- instead of *Turing Machines*

问: Why?
答: Now, we prefer *Expressiveness* (higher-level description), to *Precision* (lower-level description), in describing algorithms.

问: What is the right level of detail to give when describing such algorithms?
答: *Standardize* the way we describe Turing machine algorithms

问: How?
答: 见下页

问: Why introducing Algorithm?
答: We have come to a turning point:

- Focus on *algorithms*
- instead of *Turing Machines*

问: Why?
答: Now, we prefer *Expressiveness* (higher-level description), to *Precision* (lower-level description), in describing algorithms.

问: What is the right level of detail to give when describing such algorithms?
答: *Standardize* the way we describe Turing machine algorithms

问: How?
答: 见下页

问: Why introducing Algorithm?
答: We have come to a turning point:

- Focus on *algorithms*
- instead of *Turing Machines*

问: Why?
答: Now, we prefer *Expressiveness* (higher-level description), to *Precision* (lower-level description), in describing algorithms.

问: What is the right level of detail to give when describing such algorithms?
答: *Standardize* the way we describe Turing machine algorithms

问: How?
答: 见下页

问: How?
答: Let′ s entertain three possibilities:

- *formal* description: spells out in full the Turing machine′ s states, *transition function*, and so on
- *implementation* description: describe the way that the Turing machine moves its *head* and the way that it stores data on its *tape*
- *high-level* description: describe an algorithm, ignoring the implementation details.

问: Then?
答: Practice with the possibilities...and be confident....

问: Then??
答: Set up a format and higher-level notation.

问: How? 答: 见下页

问: How?
答: Let′ s entertain three possibilities:

- *formal* description: spells out in full the Turing machine′ s states, *transition function*, and so on
- *implementation* description: describe the way that the Turing machine moves its *head* and the way that it stores data on its *tape*
- *high-level* description: describe an algorithm, ignoring the implementation details.

问: Then?
答: Practice with the possibilities...and be confident....

问: Then??
答: Set up a format and higher-level notation.

问: How? 答: 见下页

问: How?
答: Let′ s entertain three possibilities:

- *formal* description: spells out in full the Turing machine′ s states, *transition function*, and so on
- *implementation* description: describe the way that the Turing machine moves its *head* and the way that it stores data on its *tape*
- *high-level* description: describe an algorithm, ignoring the implementation details.

问: Then?
答: Practice with the possibilities...and be confident....

问: Then??
答: Set up a format and higher-level notation.

问: How? 答: 见下页

问: How?
答: Let′ s entertain three possibilities:

- *formal* description: spells out in full the Turing machine′ s states, *transition function*, and so on
- *implementation* description: describe the way that the Turing machine moves its *head* and the way that it stores data on its *tape*
- *high-level* description: describe an algorithm, ignoring the implementation details.

问: Then?
答: Practice with the possibilities...and be confident....

问: Then??
答: Set up a format and higher-level notation.

问: How? 答: 见下页

问: How?

答: set up the notation of input, and the format of describing algorithms

- types of input:
  - string w
  - object $\langle O \rangle$ as a string: Strings can easily represent polynomials, graphs, grammars, automata, and any combination of those objects.
  - objects $\langle O_1, O_2, \ldots, O_k \rangle$ as a string
- format:
  - describe algorithms with an *indented segment* of text
  - break the algorithm into *stages*
  - The *first line* of the algorithm describes the *input* to the machine.

问: How?
答: set up the notation of input, and the format of describing algorithms

- types of input:
  - string w
  - object $\langle O \rangle$ as a string: Strings can easily represent polynomials, graphs, grammars, automata, and any combination of those objects.
  - objects $\langle O_1, O_2, \ldots, O_k \rangle$ as a string
- format:
  - describe algorithms with an *indented segment* of text
  - break the algorithm into *stages*
  - The *first line* of the algorithm describes the *input* to the machine.

问: How?
答: set up the notation of input, and the format of describing algorithms

- types of input:
  - string w
  - object $\langle O \rangle$ as a string: Strings can easily represent polynomials, graphs, grammars, automata, and any combination of those objects.
  - objects $\langle O_1, O_2, \ldots, O_k \rangle$ as a string
- format:
  - describe algorithms with an *indented segment* of text
  - break the algorithm into *stages*
  - The *first line* of the algorithm describes the *input* to the machine.

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (1) The following is a *high-level description* of a TM M that decides A.

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

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

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*:

例

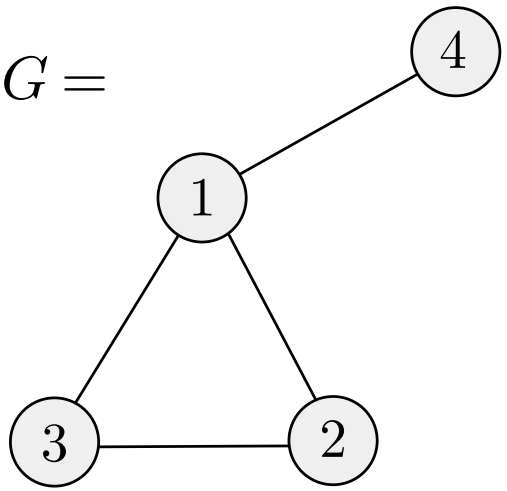
Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*:

- How $\langle G \rangle$ encodes the graph G as a string
  - Each node is a decimal number
  - each edge is the pair of decimal numbers

$G =$



$\langle G \rangle =$

(1,2,3,4)((1,2),(2,3),(3,1),(1,4))

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*:

- How to determine whether the *input* is the *proper* encoding of some graph?
  - the first list should be a list of distinct decimal numbers
  - the second should be a list of pairs of decimal numbers.
  - the node list should contain *no repetitions*
  - every node appearing on the *edge list* should also appear on the *node list*

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 1:

- M *marks* the first node *with a dot* on the leftmost digit

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
- M scans the list of nodes: find an *dotted node* n_2 and *underline* it

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
- M scans the list of nodes: find an *dotted node* n_2 and *underline* it
- M scans the list of *edges*: test whether the two *underlined nodes* n_1 and n_2 are the ones appearing in that *edge*

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
- M scans the list of nodes: find an *dotted node* n_2 and *underline* it
- M scans the list of *edges*: test whether the two *underlined nodes* n_1 and n_2 are the ones appearing in that *edge*
  - If they are, M dots n_1, removes the *underlines*, goes on from the *beginning of stage 2*

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
- M scans the list of nodes: find an *dotted node* n_2 and *underline* it
- M scans the list of *edges*: test whether the two *underlined nodes* n_1 and n_2 are the ones appearing in that *edge*
  - If they aren′ t, M checks the next edge on the list

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
- M scans the list of nodes: find an *dotted node* n_2 and *underline* it
- M scans the list of *edges*: test whether the two *underlined nodes* n_1 and n_2 are the ones appearing in that *edge*
  - If there are no more edges, $\{n_1, n_2\}$ *is not an edge of* G

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
- M scans the list of nodes: find an *dotted node* n_2 and *underline* it
  - Then M moves the underline on n_2 to the next dotted node and now calls this node n_2.

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
- M scans the list of nodes: find an *dotted node* n_2 and *underline* it
  - Then M moves the underline on n_2 to the next dotted node and now calls this node n_2.
  - check, as before, whether the new pair $\{n_1, n_2\}$ is an edge ...

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
- M scans the list of nodes: find an *dotted node* n_2 and *underline* it
  - Then M moves the underline on n_2 to the next dotted node and now calls this node n_2.
  - check, as before, whether the new pair $\{n_1, n_2\}$ is an edge ...
  - If there are no more dotted nodes, n_1 *is not attached to any dotted nodes*

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
  - Then M sets the underlines so that n_1 is the next undotted node and repeats

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 2:

- M scans the list of nodes: find an *undotted node* n_1 and *underline* it
  - Then M sets the underlines so that n_1 is the next undotted node and repeats
  - If there are no more undotted nodes, M has not been able to find any new nodes to dot, so it moves on to *stage 4*

例

Design a TM M that decides A, where

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

答: (2) examine some *implementation-level details*: stage 4:

- M scans the list of nodes to determine whether *all are dotted*
  - If they are, it enters the *accept state*
  - otherwise, it enters the *reject state*

3.1 The Church-Turing Thesis

Outline

总结

- 定义
  - Turing Machine
  - configuration , yields
  - Start/Accepting/Rejecting/Halting Configuration
  - accept , recognize , Turing-recognizable
  - decide , Turing-decidable
  - multitape Turing machine , nondeterministic Turing machine
  - enumerator
  - algorithm , Church-Turing Thesis
  - Hilbert's 10th problem
- 定理
  - multitape Turing machine has $\equiv$ single-tape Turing machine
  - nondeterministic Turing machine has $\equiv$ deterministic Turing machine
  - Turing-recognizable $\sim$ enumerator
- 推论
  - Turing-recognizable $\sim$ multitape Turing machine
  - Turing-recognizable $\sim$ nondeterministic Turing machine
  - Turing-decidable $\sim$ nondeterministic Turing machine

作业

3.1 This exercise concerns TM M_2, whose description and state diagram appear in Example 3.7. In each of the parts, give the sequence of configurations that M_2 enters when started on the indicated input string.

a. 0.

[A] **b.** 00.

c. 000.

d. 000000.

3.2 This exercise concerns TM M_1, whose description and state diagram appear in Example 3.9. In each of the parts, give the sequence of configurations that M_1 enters when started on the indicated input string.

[A] **a.** 11.

b. 1#1.

c. 1##1.

d. 10#11.

e. 10#10.

作业

3.7 Explain why the following is not a description of a legitimate Turing machine.

M_{bad} = "On input $\langle p \rangle$, a polynomial over variables $x_1, \ldots, x_k$:

1. Try all possible settings of $x_1, \ldots, x_k$ to integer values.
2. Evaluate p on all of these settings.
3. If any of these settings evaluates to 0, *accept*; otherwise, *reject*."

3.8 Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet {0,1}.

A **a.** $\{w \mid w$ contains an equal number of 0s and 1s$\}$

b. $\{w \mid w$ contains twice as many 0s as 1s$\}$

c. $\{w \mid w$ does not contain twice as many 0s as 1s$\}$