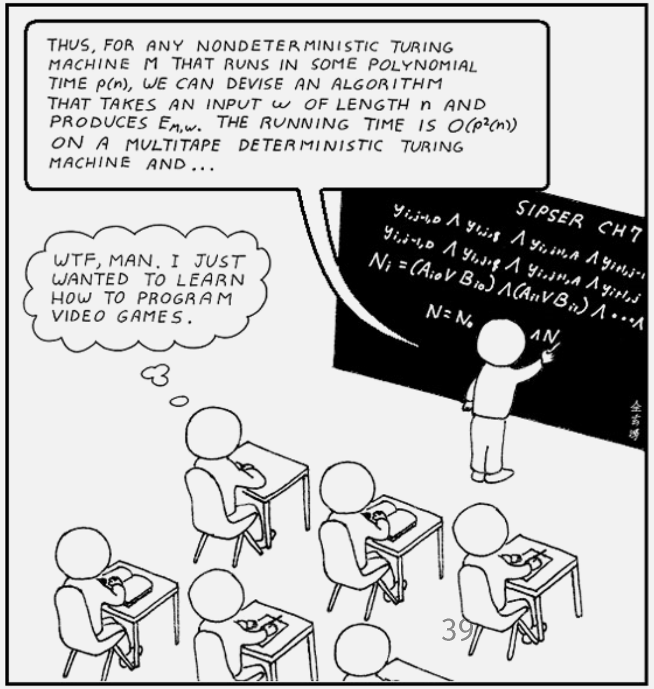
UMB CS420

Nondeterministic TMs

Monday, March 7, 2022

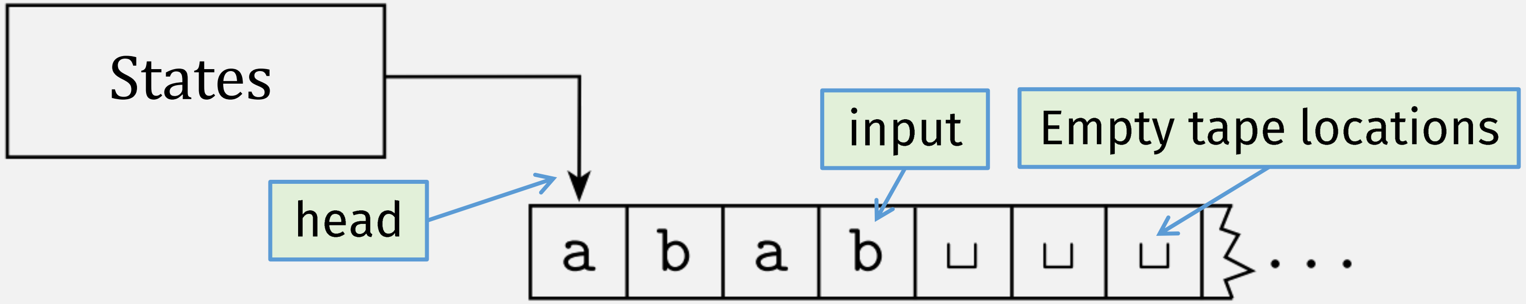


Announcements

- HW 5 in
- HW 6 out
  - Due Sun 3/20 11:59pm EST (2 weeks)
- Reminder: No class next week (Spring Break)

Last Time: Turing Machines

- Turing Machines can read and write to arbitrary "tape" cells
  - Tape initially contains input string
- The tape is infinite
  - (to the right)



- On a transition, "head" can move left or right 1 step

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

Turing Machine: Informal Description

- M_1 accepts if input is in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

M_1 = "On input string w:

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do no … if no # is found, *reject*. Cross off symbols as they … track of which symbols correspond.
2. When all symbols to … n crossed off, check for any remaining … t of the #. If any symbols remain, *reject*; otherwise, … ccept."

We will (mostly) stick to informal descriptions of Turing machines, like this one

Turing Machines: Formal Definition

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. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,

   read — write — move

5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Non-Deterministic Turing Machines?

Flashback: DFAs vs NFAs

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

VS

Nondeterministic transition produces set of possible next states

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Remember: Turing Machine Formal Definition

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

Nondeterministic ~~Remember:~~ Turing Machine Formal Definition

A **Nondeterministic 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. ~~$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$~~ → $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\text{L}, \text{R}\})$
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Thm: Deterministic TM ⇔ Non-det. TM

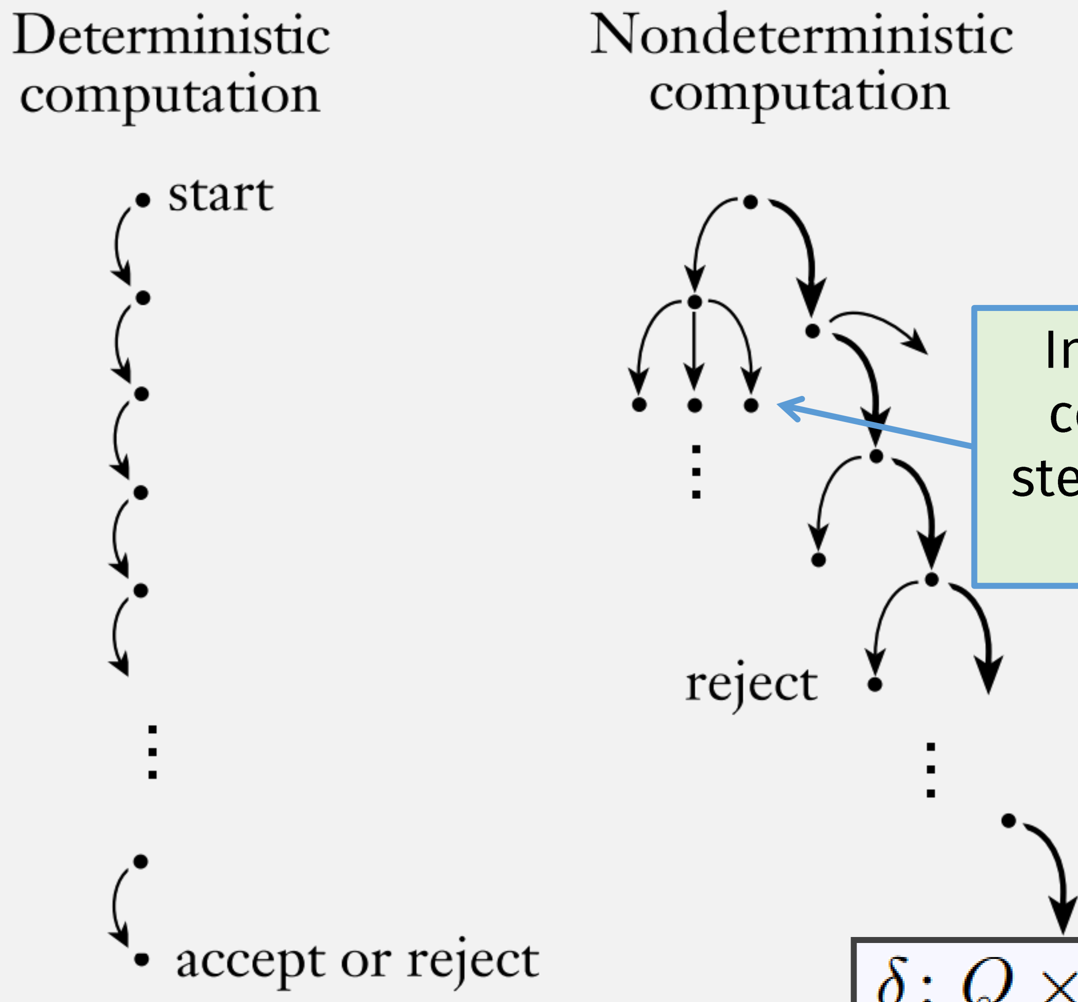
⇒ **If** a deterministic TM recognizes a language,
then a non-deterministic TM recognizes the language

- To convert Deterministic TM → Non-deterministic TM …
- … change Deterministic TM δ fn output to a one-element set
  - (just like conversion of DFA to NFA --- HW 2, Problem 3)
- **DONE!**

⇐ **If** a non-deterministic TM recognizes a language,
then a deterministic TM recognizes the language

- To convert Non-deterministic TM → Deterministic TM …
- … ???

Review: Nondeterminism



In nondeterministic computation, every step can branch into a set of "states"

What is a "state" for a TM?

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

Flashback: PDA Configurations (IDs)

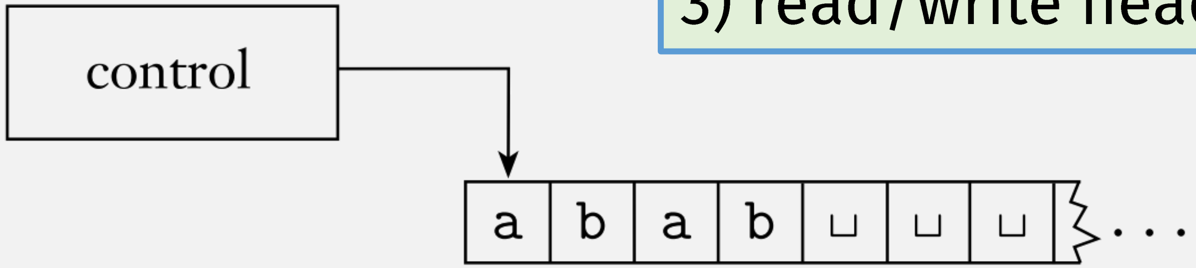
- A **configuration** (or **ID**) is a <u>snapshot</u> of a PDA's computation

- A configuration (or **ID**) (q, w, γ) has three components:
  - q = the current state
  - w = the remaining input string
  - γ = the stack contents

TM Configuration (ID) = ???

1) states

3) read/write head

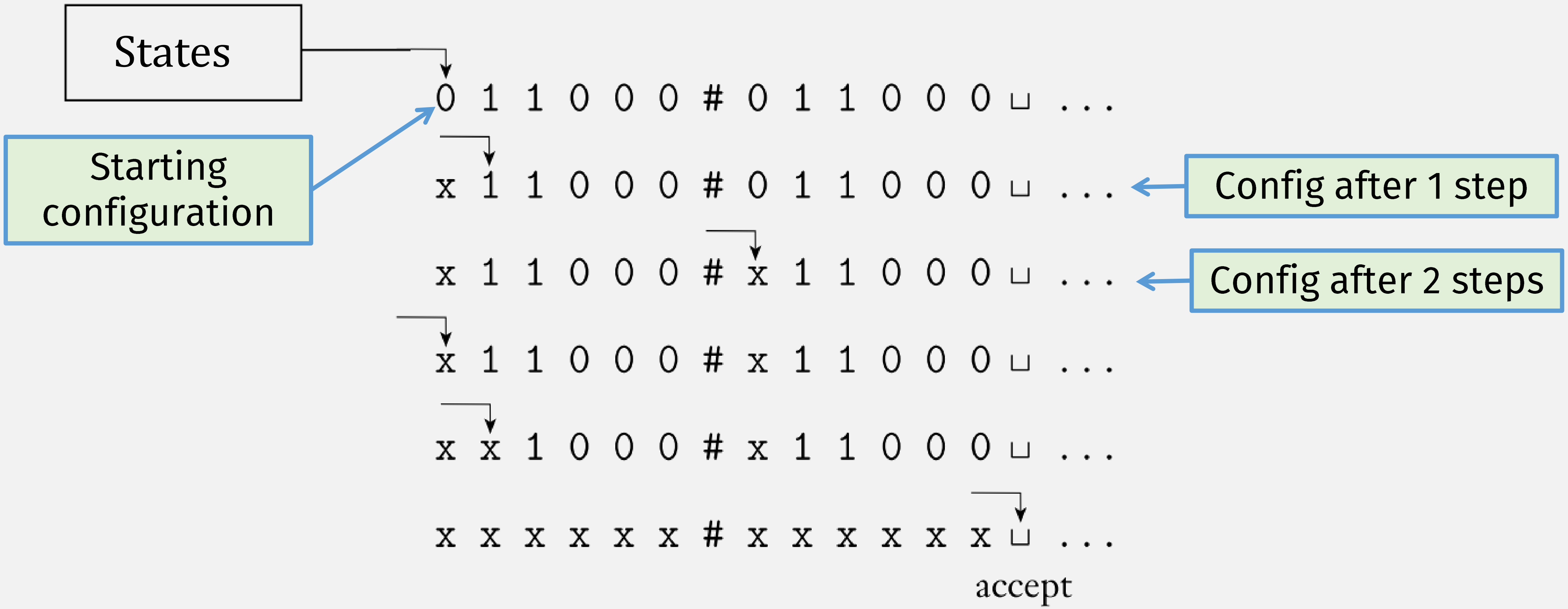


2) Tape contents

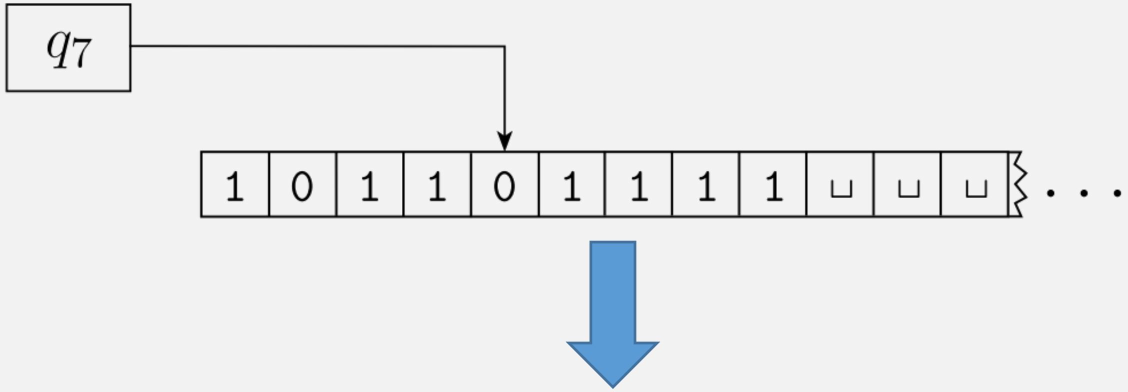
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

TM Configuration = State + Head + Tape



TM Configuration = State + Head + Tape



$1011q_7 01111$

Textual representation of “configuration” (use this in HW)

1st char after state is current head position

TM Computation, Formally

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

Single-step

head

Next config

(Right) $\alpha q_1 \mathbf{a} \beta \vdash \alpha \mathbf{x} q_2 \beta$

write

if $q_1, q_2 \in Q$

$\delta(q_1, \mathbf{a}) = (q_2, \mathbf{x}, \mathrm{R})$

read

$\mathbf{a}, \mathbf{x} \in \Gamma \quad \alpha, \beta \in \Gamma^*$

(Left) $\alpha b q_1 \mathbf{a} \beta \vdash \alpha q_2 b \mathbf{x} \beta$

if $\delta(q_1, \mathbf{a}) = (q_2, \mathbf{x}, \mathrm{L})$

Extended

- Base Case

  $I \overset{*}{\vdash} I$ for any ID I

- Recursive Case

  $I \overset{*}{\vdash} J$ if there exists some ID K such that $I \vdash K$ and $K \overset{*}{\vdash} J$

Edge cases:

$q_1 \mathbf{a} \beta \vdash q_2 \mathbf{x} \beta$ if $\delta(q_1, \mathbf{a}) = (q_2, \mathbf{x}, \mathrm{L})$

Head stays at leftmost cell

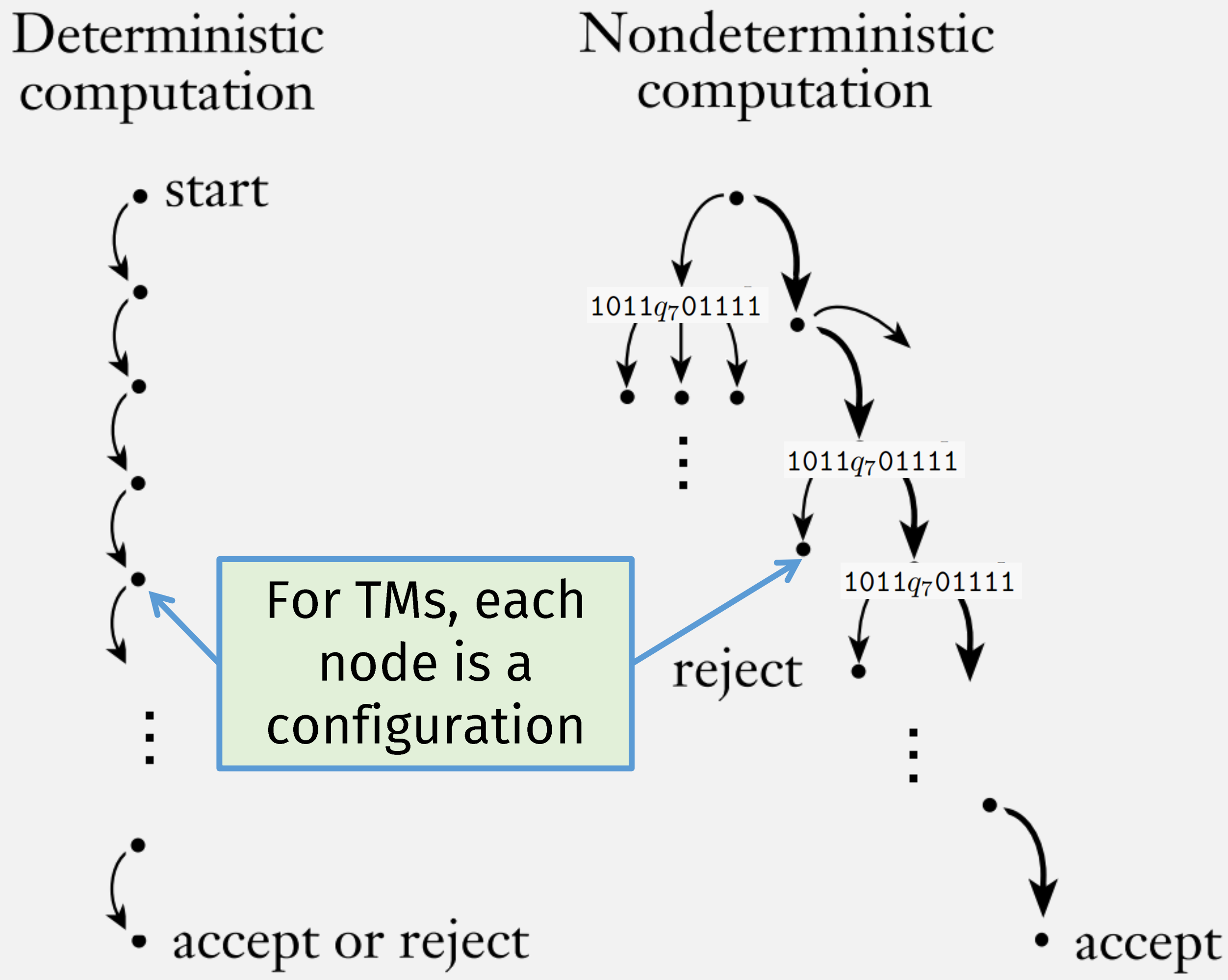
(L move, when already at leftmost cell)

$\alpha q_1 \vdash \alpha \_ q_2$ if $\delta(q_1, \_) = (q_2, \_, \mathrm{R})$

Add blank symbol to config

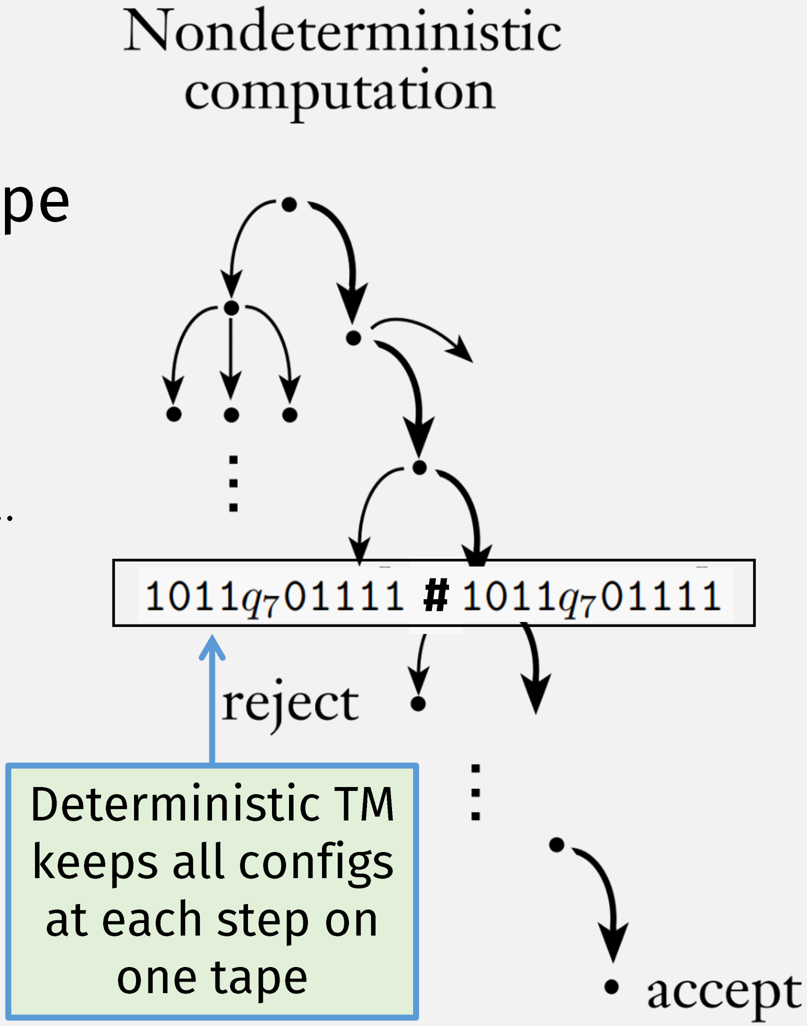
(R move, when at rightmost filled cell)

Nondeterminism in TMs



Nondeterministic TM → Deterministic 1st way

- Simulate NTM with Det. TM:
  - Det. TM keeps multiple configs single tape
    - Like how single-tape TM simulates multi-tape
  - Then run all computations, <u>in parallel</u>
    - I.e., **1 step** on one config, **1 step** on the next, …
  - Accept if any accepting config is found
  - **Important:**
    - Why must we step configs <u>in parallel</u>?



Interlude: Running TMs inside other TMs

"loop" means input string not accepted

Exercise:

- Given TMs M_1 and M_2, create TM M that accepts if either M_1 or M_2 accept

Possible solution #1:

- M = on input x,
  - Run M_1 on x, accept if M_1 accepts
  - Run M_2 on x, accept if M_2 accepts

M_1	M_2	M
reject	accept	accept
accept	reject	accept

Note: This solution would be ok if we knew M_1 and M_2 were **deciders** (which halt on all inputs)

Interlude: Running TMs inside other TMs

Exercise:

- Given TMs M_1 and M_2, create TM M that accepts if either M_1 or M_2 accept

Possible solution #1:

- M = on input x,
  - Run M_1 on x, accept if M_1 accepts
  - Run M_2 on x, accept if M_2 accepts

M_1	M_2	M
reject	accept	accept
accept	reject	accept ☑
accept	loops	accept
loops	accept	loops ✖

Possible solution #2:

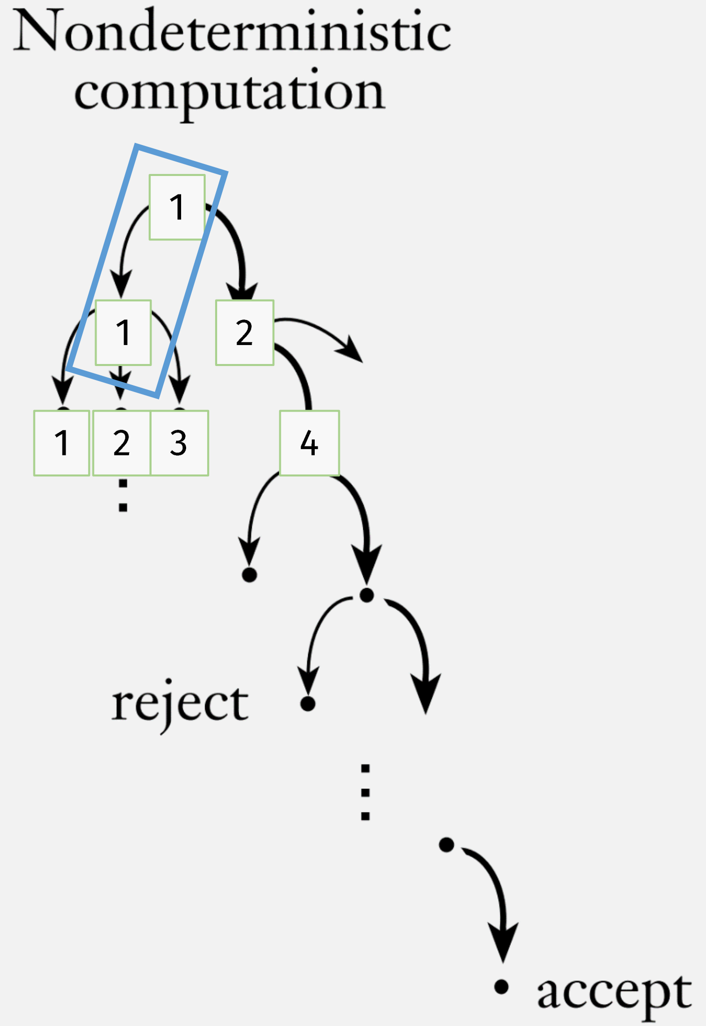
- M = on input x,
  - Run M_1 and M_2 on x in parallel, i.e.,
    - Run M_1 on x for 1 step, accept if M_1 accepts
    - Run M_2 on x for 1 step, accept if M_2 accepts
    - Repeat

M_1	M_2	M
reject	accept	accept
accept	reject	accept ☑
accept	loops	accept
loops	accept	accept ☑

Nondeterministic TM → Deterministic

2nd way
(Sipser)

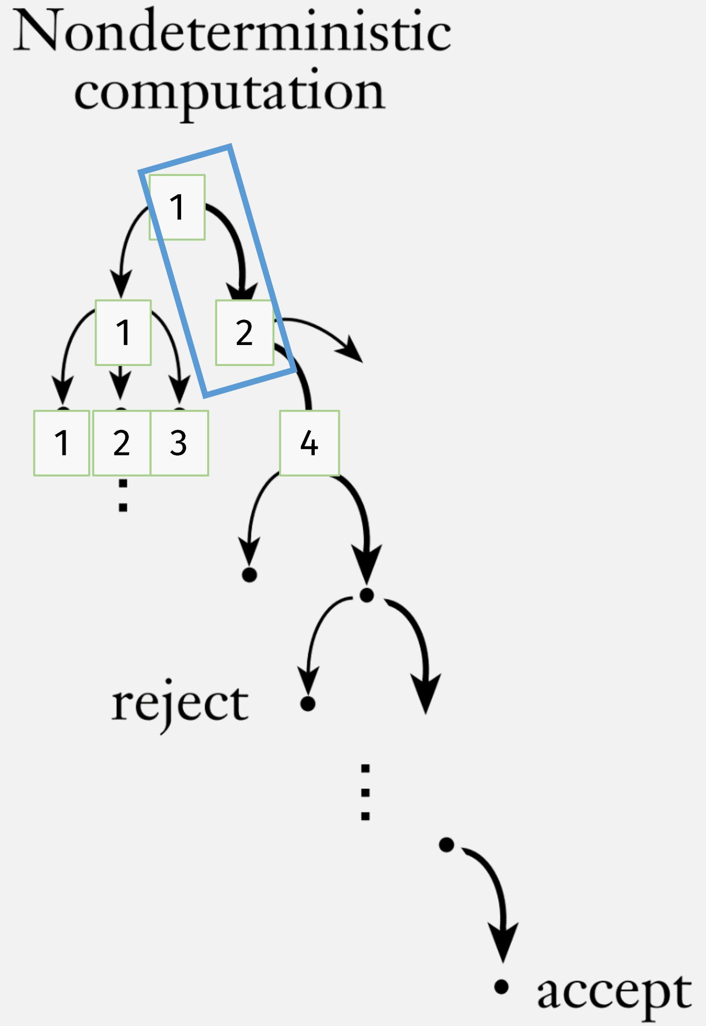
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - **1-1**



Nondeterministic TM → Deterministic

2nd way
(Sipser)

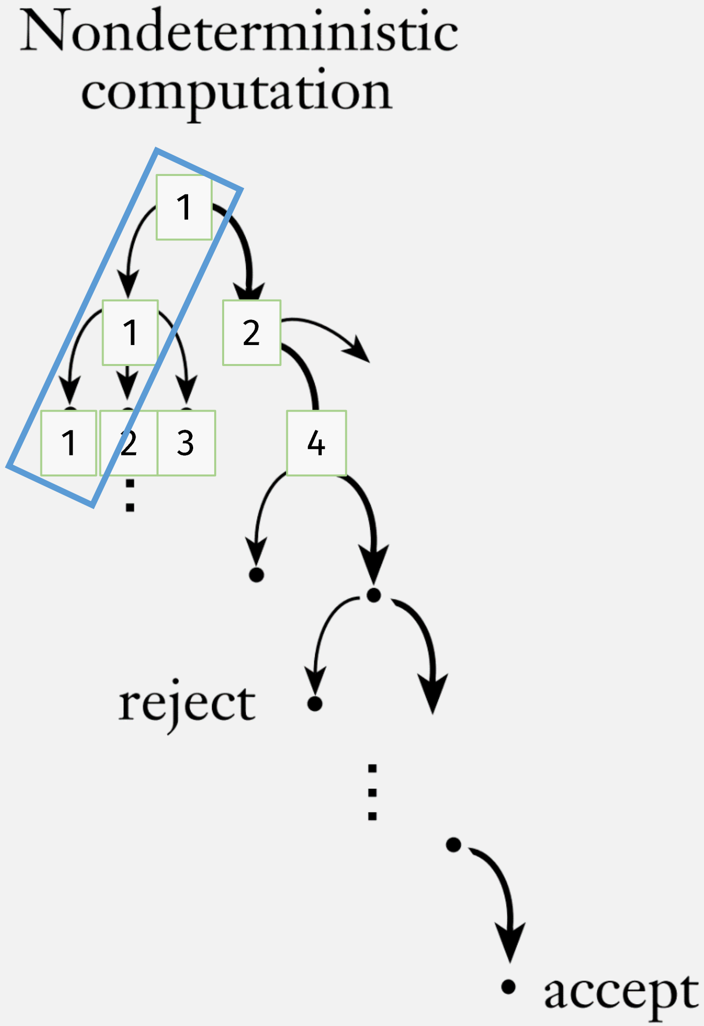
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - **1-2**



Nondeterministic TM → Deterministic

2nd way
(Sipser)

- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - **1-1-1**

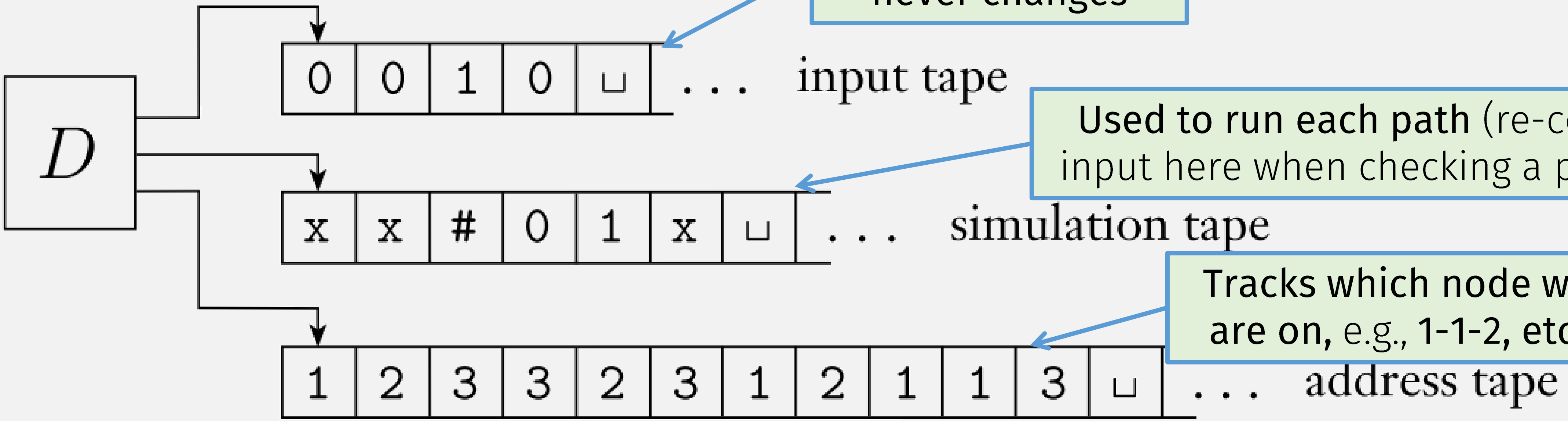


Nondeterministic TM → Deterministic

2nd way
(Sipser)

Needs 3 tapes

Always has input, never changes



Used to run each path (re-copy input here when checking a path)

Tracks which node we are on, e.g., **1-1-2, etc.**

Nondeterministic TM ⇔ Deterministic TM

☑ => **If** a deterministic TM recognizes a language,
then a nondeterministic TM recognizes the language

- To convert **Deterministic TM → Non-deterministic TM** …
- … change **Deterministic TM δ fn output to a one-element set**
  - (just like conversion of DFA to NFA)

☑ <= **If** a nondeterministic TM recognizes a language,
then a deterministic TM recognizes the language

- Convert **Nondeterministic TM → Deterministic TM** ∎

Conclusion: These are All Equivalent TMs!

- Single-tape Turing Machine
- Multi-tape Turing Machine
- Non-deterministic Turing Machine

Turing Machines and Algorithms

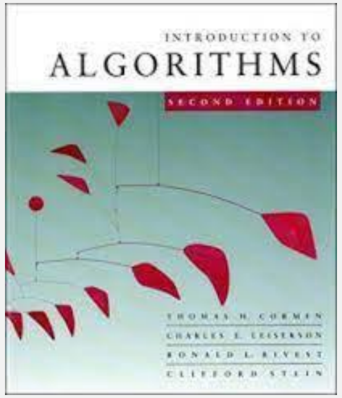
- Turing Machines can express any "computation"
  - I.e., **a Turing Machine models** (Python, Java) **programs**!

- 2 classes of Turing Machines
  - Recognizers may loop forever
  - Deciders always halt

Next

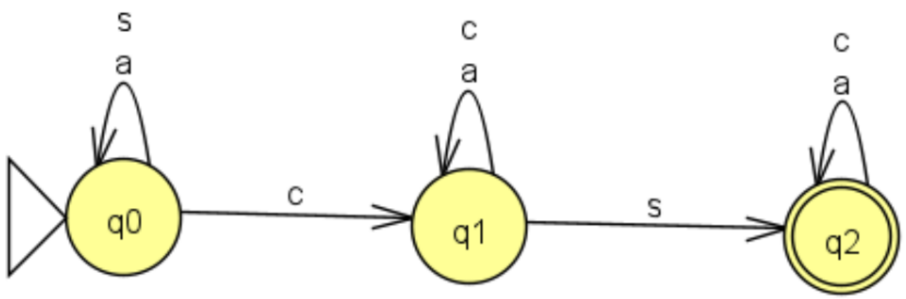
- Deciders = Algorithms
  - I.e., an algorithm is any program that always halts

Remember:
TMs = programs



Flashback: HW 1, Problem 1

1 DFA Formal Description



1. Come up with a formal description for this DFA.

   Recall that a DFA's formal description has five components, e.g. $M = (Q, \Sigma, \delta, q_0, F)$.

   You may assume that the alphabet contains only the symbols from the diagram.

2. Then do the following computations using extended transition function and say whether computation represents an accepting computation (some of these may be tricky so be careful here, you may want to review the definition of an accepting computation):

   a. $\hat{\delta}(q0, \varepsilon)$

   b. $\hat{\delta}(q0, \mathtt{a})$

This represents computation by a DFA

You had to "do" (meta) **computations** (e.g., on paper, in your head), **to "do" this computation!**

$\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***

Flashback: DFA Computations

Define the extended transition function: $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

Base case: $\hat{\delta}(q, \epsilon) = q$

Recursive case: $\hat{\delta}(q, a_1 w_{rest}) = \hat{\delta}(\delta(q, a_1), w_{rest})$

First char

Last chars

Single transition step

Remember:
TMs = programs

Calculating this computation requires (meta) computation!

Could you implement this (meta) computation as an **algorithm?**

A function: **`DFAaccepts(B,w)`** returns TRUE if DFA **B** accepts string `w`

- Define "current" state $q_{current}$ = start state q_0
- For each input char a_i ...
  - Define $q_{next} = \delta(q_{current}, a_i)$
  - Set $q_{current} = q_{next}$
- Return TRUE if $q_{current}$ is an accept state

The language of **`DFAaccepts`**

Function **`DFAaccepts(B,w)`** returns TRUE if DFA **`B`** accepts string **`w`**

$$A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

But a language is a set of strings?

Interlude: Encoding Things into Strings

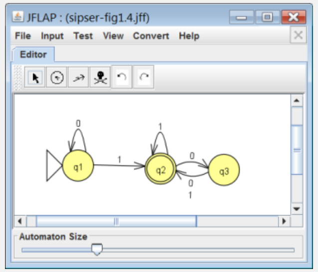
- A Turing machine's input is always a string

- So anything we want to give to TM must be **encoded** as string

Notation: <SOMETHING> = string encoding for SOMETHING

- A tuple combines multiple encodings, e.g., <*B*, *w*> (from prev slide)

Example: Possible string encoding for a DFA?




```
<automaton>
  <!--The list of states.-->
  <state name="q1"><initial/></state>
  <state name="q2"><final/></state>
  <state name="q3"></state>
  <!--The list of transitions.-->
  <transition>
    <from>0</from>
    <to>0</to>
    <read>0</read>
  </transition>
  <transition>
    <from>1</from>
```

Or:

$$(Q, \Sigma, \delta, q_0, F)$$

(written as string)

Interlude: Informal TMs and Encodings

An informal TM description:

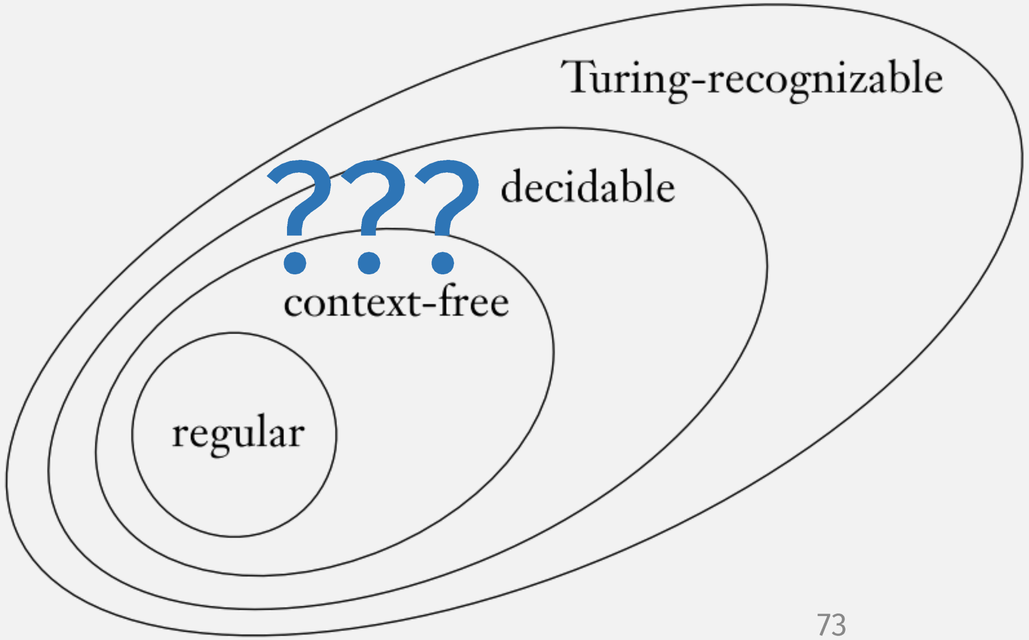
1. Doesn’t need to describe exactly how input string is encoded
2. Assumes input is a “valid” encoding
   - Invalid encodings are automatically rejected

The language of **`DFAaccepts`**

$$A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a } \mathsf{DFA} \text{ that accepts input string } w\}$$

- **`DFAaccepts`** is a Turing machine
- But is it a **decider** or **recognizer**?
  - I.e., is it an **algorithm**?
- To show it's an algo, need to <u>prove</u>:

$$A_{\mathsf{DFA}} \text{ is a decidable language}$$



How to prove that a language is decidable?

- Create a Turing machine that **decides** that language!

Remember:

- A **decider** is Turing Machine that always halts
  - I.e., for any input, it either accepts or rejects it.
  - It must never go into an infinite loop

How to Design Deciders

- If TMs = Programs …

  … then **Creating** a TM = Programm**ing**

- E.g., if HW asks "Show that lang L is decidable" …
  - .. you must create a TM that decides L; to do this …
  - … think of how to write a (halting) program that does what you want

Next Time: A_{DFA} is a decidable language

$A_{\mathsf{DFA}} = \{\langle B, w \rangle \mid B$ is a DFA that accepts input string $w\}$

Decider for A_{DFA} :

Check-in Quiz 3/7

On gradescope