Theory of Computation
Week 9

Much of the material on this slides comes from the recommended textbook by Elaine Rich

Announcement

Assignment 2 Due

- Due: 17/05/2020

Timed Exam (Fixed Start) - The exam will be available in the Assessment or Assignment tab in Blackboard at the scheduled date and time on the exam timetable. Students must complete the exam within an allocated period of time advised in Blackboard. All students complete the exam at the same time regardless of their time zone.

Detailed content

Weekly program

- ✓ Week 1 – Background knowledge revision: logic, sets, proof techniques
- ✓ Week 2 – Languages and strings. Hierarchies. Computation. Closure properties
- ✓ Week 3 – Finite State Machines: non-determinism vs. determinism
- ✓ Week 4 – Regular languages: expressions and grammars
- ✓ Week 5 – Non regular languages: pumping lemma. Closure
- ✓ Week 6 – Context-free languages: grammars and parse trees
- ✓ Week 7 – Pushdown automata
- ✓ Week 8 – Non context-free languages: pumping lemma and decidability. Closure

➡ **Week 9 – Decidable languages: Turing Machines**

- ❑ Week 10 – Church-Turing thesis and the unsolvability of the Halting Problem
- ❑ Week 11 – Decidable, semi-decidable and undecidable languages (and proofs)
- ❑ Week 12 – Revision of the hierarchy. Safety-critical systems
- ❑ Week 13 – Extra revision (if needed)

Week 09 Videos

You already know

- ❑ Why PDA can't handle $A^nB^nC^n$
- ❑ What is a Turning Machine
- ❑ Formal definition of TM
- ❑ Example of TM

Videos to watch before lecture

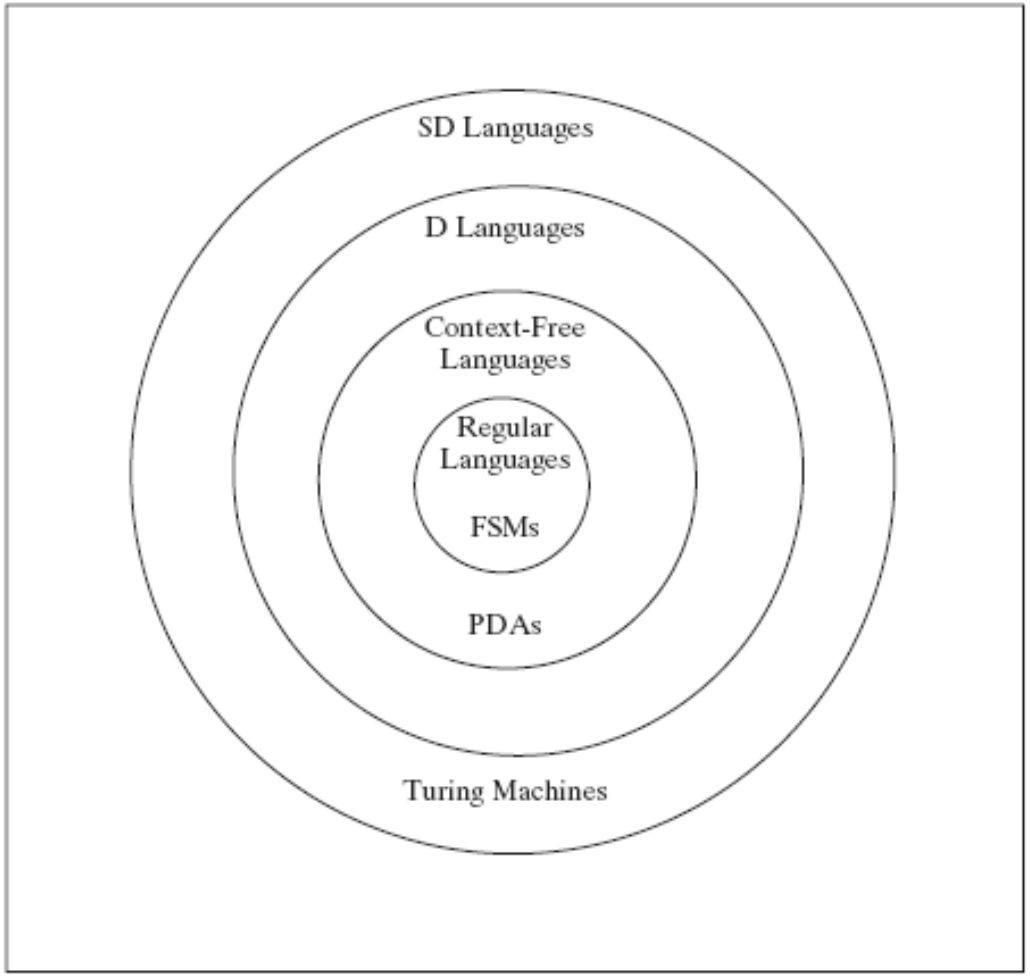
Additional videos to watch for this week

Week 09 Lecture

Turing Machine

- ❑ What are Turing Machines?
- ❑ Example TM
- ❑ TM Macro Notations
- ❑ TM as language recognizer
- ❑ Decidable and Semidecidable TM
- ❑ TM for computing functions
- ❑ Turing Machine Extensions
- ❑ Equivalence of extended TM and standard TM

THE HIERARCHY

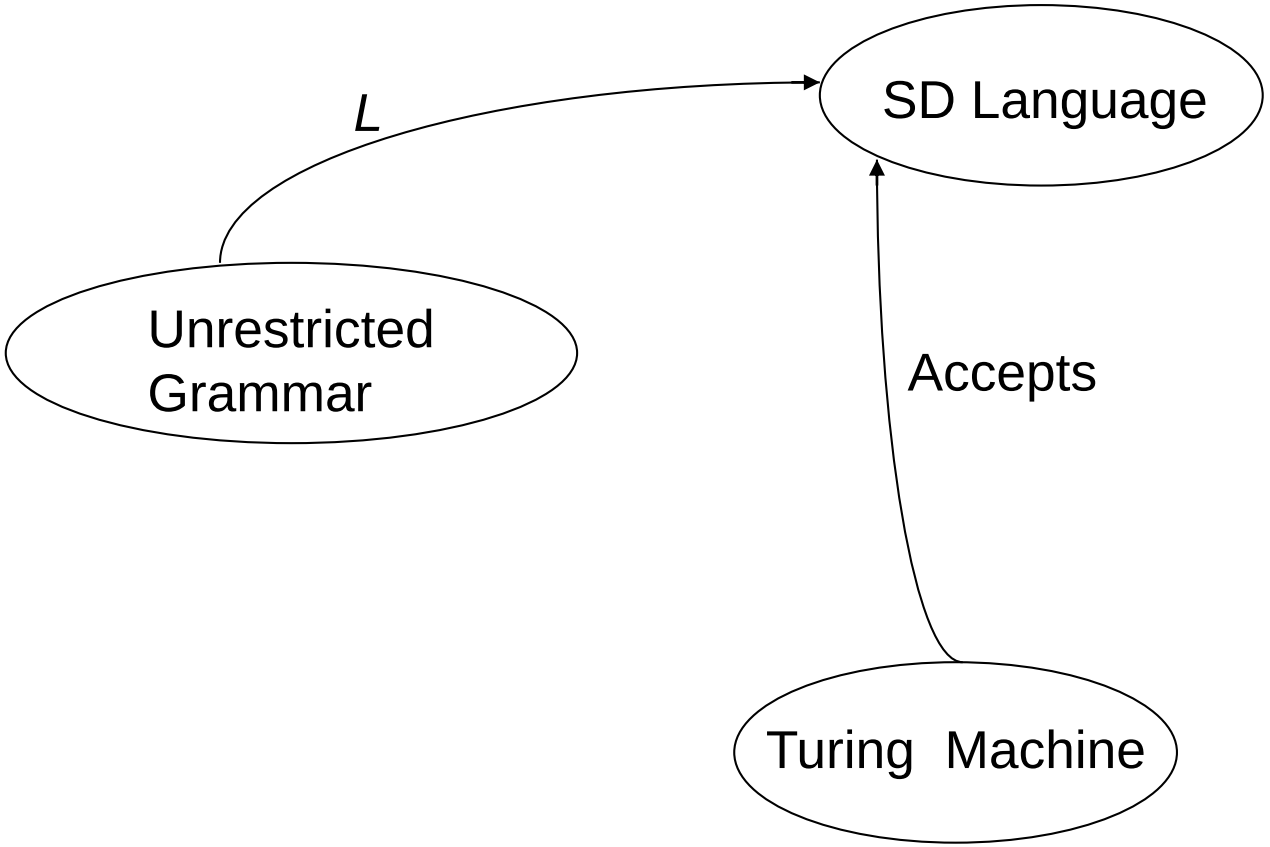


CLASSES OF LANGUAGES SO FAR

A language is ***regular*** iff it is accepted by some FSM

A language *L* is ***context-free*** if and only if it is generated by some context-free grammar *G*.

GRAMMARS, SD LANGUAGES, AND TURING MACHINES



TURING MACHINES

Can we come up with a new kind of automaton that has two properties:

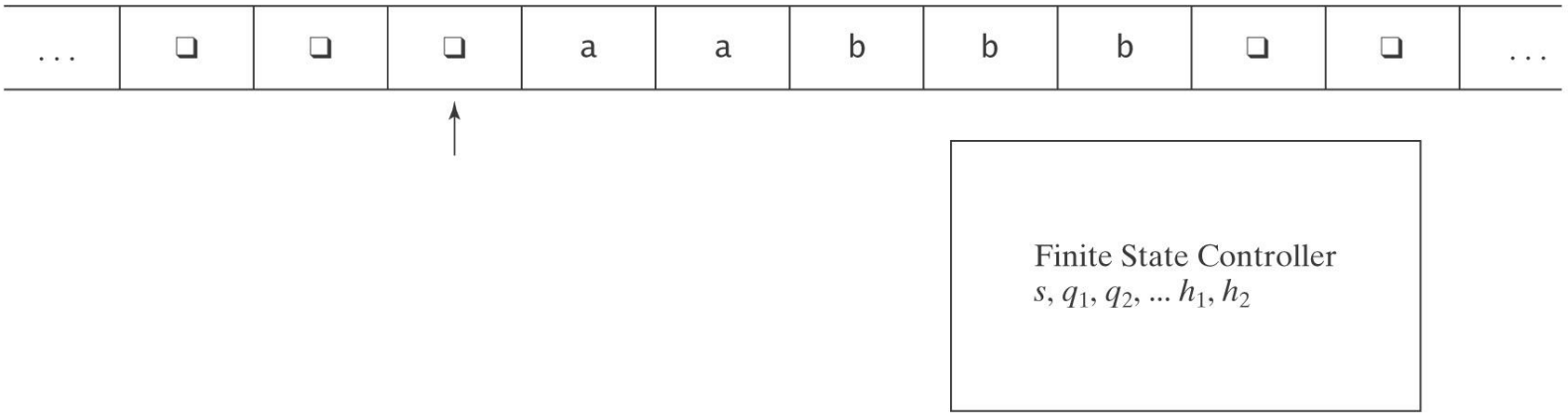
- powerful enough to describe all computable things

  unlike FSMs and PDAs.

- simple enough that we can reason formally about it

  like FSMs and PDAs,
  unlike real computers.

TURING MACHINES



At each step, the machine must:

- choose its next state,
- write on the current square, and
- move left or right.

TURING MACHINES

https://www.youtube.com/watch?v=E3keLeMwfHY

https://www.youtube.com/watch?v=gJQTFhkhwPA

https://www.youtube.com/watch?v=dNRDvLACg5Q

TURING MACHINES

A Turing machine M is a sixtuple $(K, \Sigma, \Gamma, \delta, s, H)$:

- K is a finite set of states;
- Σ is the input alphabet, which does not contain ❑;
- Γ is the tape alphabet, which must contain ❑ and have Σ as a subset.
- $s \in K$ is the initial state;
- $H \subseteq K$ is the set of halting states;
- δ is the transition function:

$$(K - H) \times \Gamma \text{ to } K \times \Gamma \times \{\rightarrow, \leftarrow\}$$

non-halting state	×	tape char		state	×	tape char	×	action (R or L)

TURING MACHINES

$(K - H) \times \Gamma$ to $K \times \Gamma \times \{\rightarrow, \leftarrow\}$

non-halting state × tape char → state × tape char × action (R or L)

If δ contains the transition $((q_0, \texttt{a}),(q_1, \texttt{b}, A))$, then, whenever M is in state q_0, and the character under the read/write head is an $\texttt{a}$,

- M will go to state q_1,
- write a $\texttt{b}$, and
- move the read/write head as specified by A (either one square to the right or one square to the left)

TURING MACHINES

Notice that the tape symbol ❑ is important:

- Initially, all tape squares except those that contain the input string contain ❑,
- The input string may not contain ❑.

When designing a machine be careful when you write ❑. You have to make sure your machine knows whether it is just a part or the end of the string.

. . .	❑	❑	❑	a	a	b	b	b	❑	❑	. . .

TURING MACHINES

1. The input tape is infinite in both directions.

2. δ is a function, not a relation. So this is a definition for deterministic Turing machines.

3. δ must be defined for all (state, input) pairs unless the state is a halting state.

4. Turing machines do not necessarily halt (unlike FSM's and PDAs). Why? To halt, they must enter a halting state. Otherwise they loop.

5. Turing machines generate output so they can compute functions.

TURING MACHINES

- A ***binary relation*** over two sets *A* and *B* is a subset of $A \times B$.
- A ***function*** *f* from a set *A* to a set *B* is a mapping of elements of A to elements of B such that each element of A maps to exactly one element of B.

$$\forall x \in A\ ((((x, y) \in f \wedge (x, z) \in f) \rightarrow y = z) \wedge \exists y \in B\ ((x, y) \in f))$$

TURING MACHINES
Example

Let's assume a Turing machine *M* is (K, Σ, Γ, δ, s, H):

K = {S1, S2}, Σ = {0,1} , Γ = {0,1, ❑}, s = S1, H = {S2}
δ = {S1, 0} , {S2, 1, $\leftarrow$},
{S1, 1} , {S2, 0, $\rightarrow$},
{S1, ❑} , {S1, 0, $\rightarrow$},

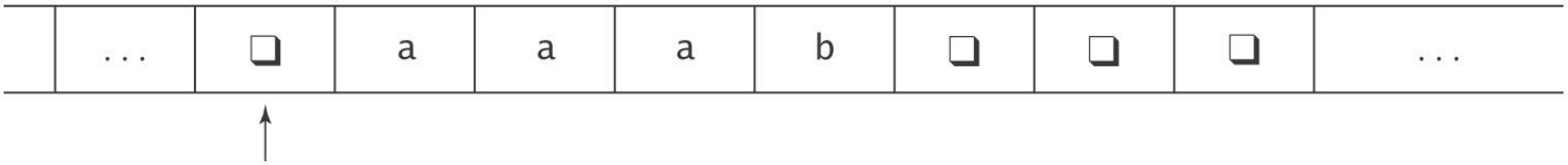
TURING MACHINES
Example

M takes as input a string in the language:

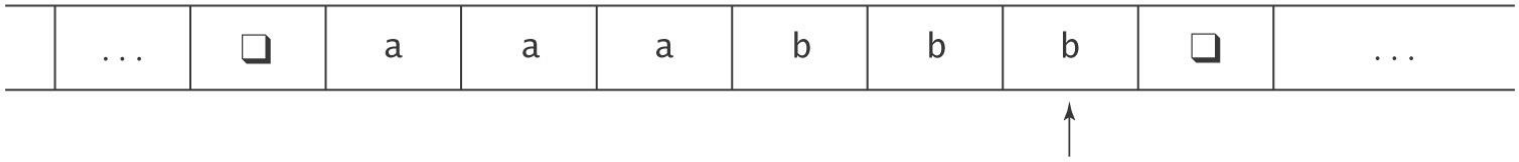
$\{a^i b^j, 0 \leq j \leq i\}$,

and adds b's as required to make the number of b's equal the number of a's.

The input to *M* will look like this:



The output should be:



TURING MACHINES
Example

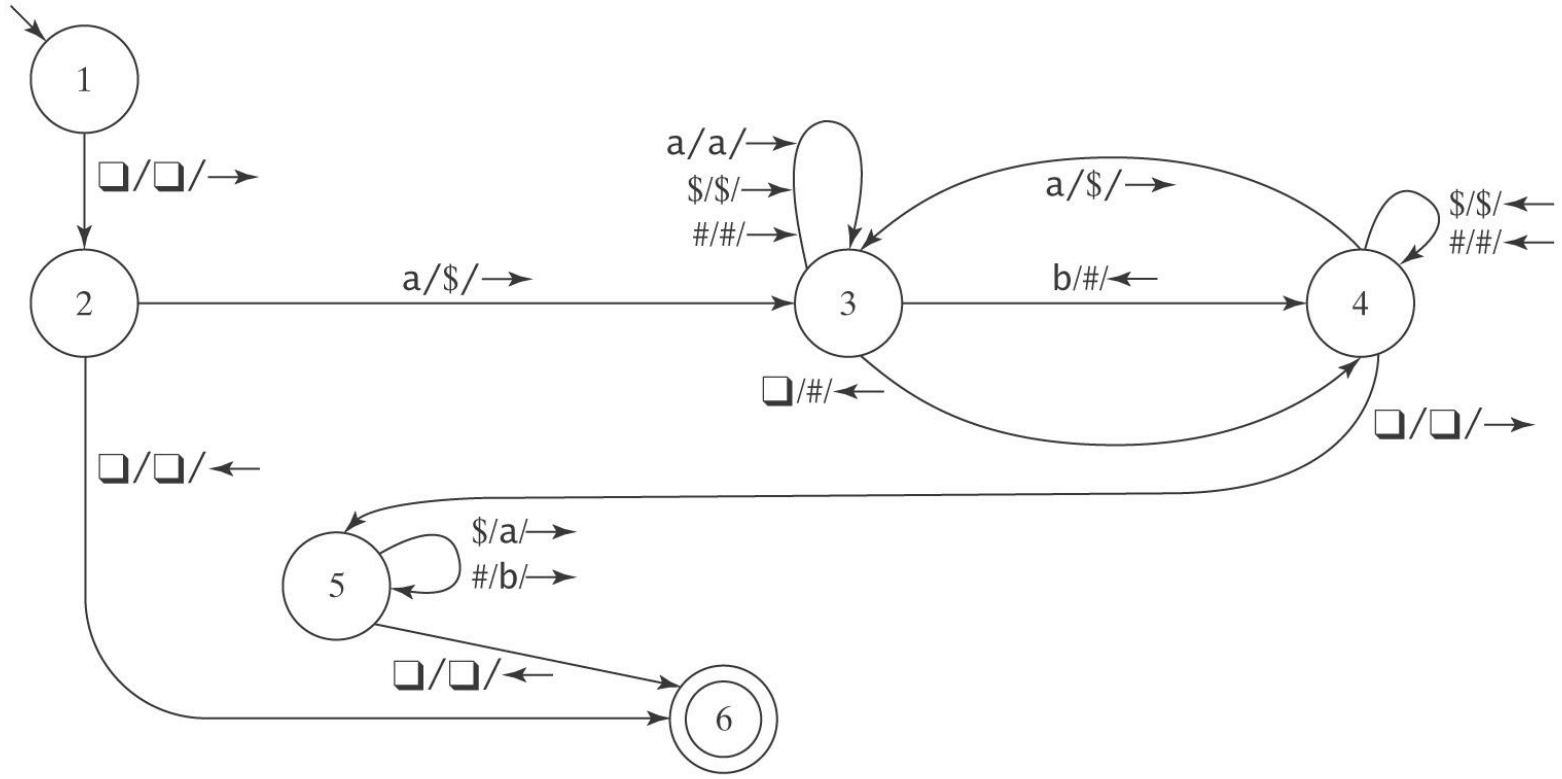
$\{a^i b^j, 0 \leq j \leq i\}$ to $\{a^i b^i, 0 \leq i\}$

M will operate as follows:

1. Move one square to the right. If character is ❑, halt. Otherwise continue
2. Loop:
   a. Mark off an a with a $
   b. Scan rightward to first b or ❑,
      - If b, mark it off with # and get ready to go back and search for next a,b pair
      - If ❑, then there are no more b's but there are a's that need matched. Write # and get ready to go back and search for unmarked a's
   c. Scan back left for an a or ❑. If a, go to 2.a. If ❑, then exit loop.
3. Make one last pass though the non-blank area changing all $ to a's and # to b's
4. Halt

TURING MACHINES
Example

$K = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, ❑, \$, \#\}$,
$s = 1$, $H = \{6\}$, $\delta =$



TURING MACHINES
Example

M = {{1, 2, 3, 4, 5, 6}, {a, b}, {a, b, ❑, $, #}, δ, 1, {6}), where δ =

(((1, ❑), (2, ❑, →)),
  ((1, a), (2, ❑, →)),
  ((1, b), (2, ❑, →)),
  ((1, $), (2, ❑, →)),
  ((1, #), (2, ❑, →)),
  ((2, ❑), (6, ❑, →)),
  ((2, a), (3, $, →)),
  ((2, b), (3, $, →)),
  ((2, $), (3, $, →)),
  ((2, #), (3, $, →)),
  ((3, ❑), (4, # , ←)),
  ((3, a), (3, a, →)),
  ((3, b), (4, #, ←)),
  ((3, $), (3, $, →)),
  ((3, #), (3, #, →)),
  ((4, ❑), (5, ❑, →)),
  ((4, a), (3, $, →)),
  ((4, $), (4, $, ←)),
  ((4, #), (4, # , ←)),
  ((5, ❑), (6, ❑, ←)),
  ((5, $), (5, a, →)),
  ((5, #), (5, b , →)))

{These four transitions are required because M must be defined for every state/ input pair, but since it isn't possible to see anything except ❑ in state 1, it doesn't matter what they do. }

{Three more unusable elements of δ. We'll omit the rest here for clarity.}

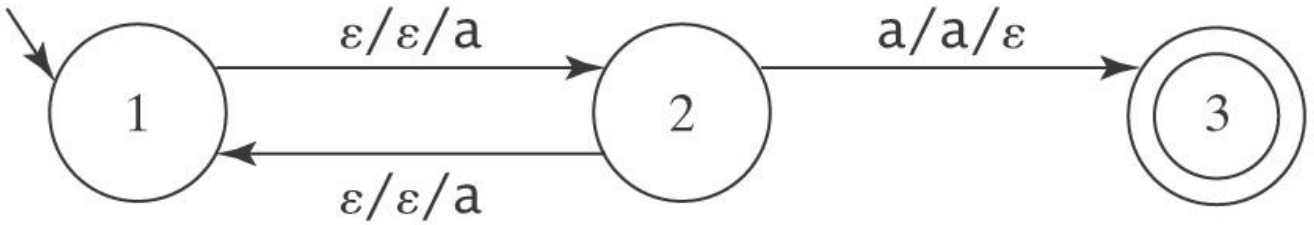
{State 6 is a halting state and so has no transitions out of it.}

TURING MACHINES

- The machine has a strong procedural feel, with one phase coming after another.
- There are common idioms, like scan left until you find a blank
- There are two common ways to scan back and forth marking things off.
- Often there is a final phase to fix up the output.
- Even a very simple machine is a nuisance to write.

HALTING

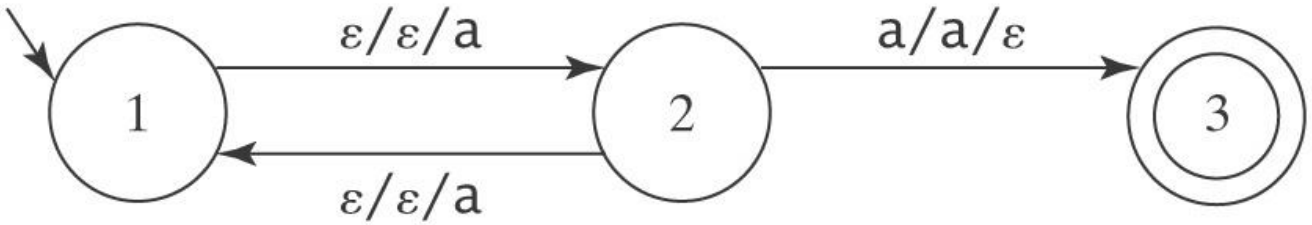
- A DFSM *M*, on input *w*, is guaranteed to halt in |*w*| steps.
- A PDA *M*, on input *w*, is not guaranteed to halt. To see why, consider again *M* =



This is a PDA accepts a the language L(M)= {a}. The computation (1,a,ε) **|-** (2,a,a) **|-** (3, ε,ε) will cause M to accept ‘a’ but any other input will make this PDA loop forever

HALTING

- A DFSM *M*, on input *w*, is guaranteed to halt in |*w*| steps.
- A PDA *M*, on input *w*, is not guaranteed to halt. To see why, consider again *M* =



But there exists an algorithm to construct an equivalent PDA *M′* that is guaranteed to halt.

A TM *M*, on input *w*, is not guaranteed to halt. And there exists no algorithm to construct one that is guaranteed to do so.

FORMALISING TURING MACHINES

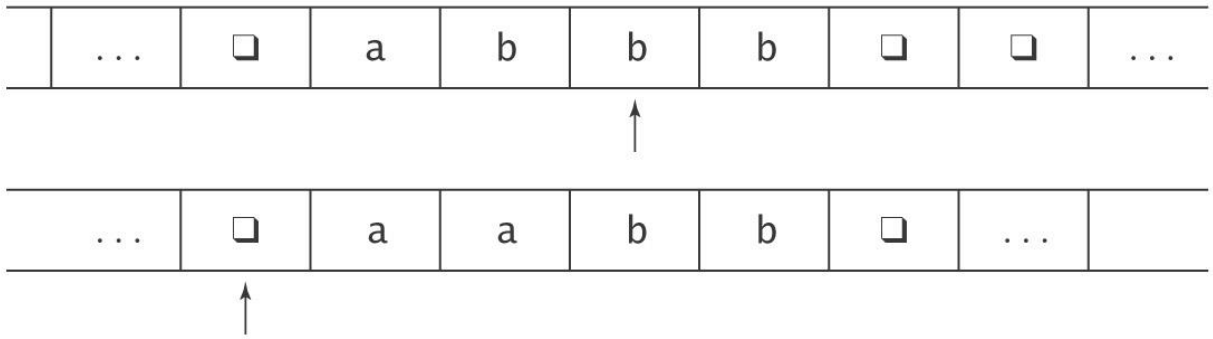
A *configuration* of a Turing machine

$M = (K, \Sigma, \Gamma, s, H)$ is an element of:

$$K \times ((\Gamma - \{\square\})\,\Gamma^*) \cup \{\varepsilon\} \times \Gamma \times (\Gamma^*\,(\Gamma - \{\square\})) \cup \{\varepsilon\}$$

state	all active tape up to scanned square	scanned square	all active tape after scanned square

TURING MACHINES
Example configurations



(1) (*q*, ab, b, b) = (*q*, abb)
(2) (*q*, ε, ❑, aabb) = (*q*, ❑aabb)

Initial configuration is (*s*, ❑*w*).

TURING MACHINES
Yields

(q_1, w_1) $|\text{-}_M$ (q_2, w_2) iff (q_2, w_2) is derivable, via δ, in one step.

For any TM M, let $|\text{-}_M^*$ be the reflexive, transitive closure of $|\text{-}_M$.

Configuration C_1 ***yields*** configuration C_2 if:

$$C_1 \ |\text{-}_M^* \ C_2.$$

TURING MACHINES
Yields

A ***path*** through M is a sequence of configurations $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that C_0 is the initial configuration and:

$$C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n.$$

A **computation** by M is a path that halts.

If a computation is of *length n* or has n steps, we write:

$$C_0 \vdash_M^n C_n$$

MACRO NOTATION FOR TURING MACHINES

- Symbol writing machines

For each $x \in \Gamma$, define M_x, written just x, to be a machine that writes x.

- Head moving machines

R: for each $x \in \Gamma$, $\delta(s, x) = (h, x, \rightarrow)$
L: for each $x \in \Gamma$, $\delta(s, x) = (h, x, \leftarrow)$

- Machines that simply halt

h, which simply halts.
n, which halts and rejects.
y, which halts and accepts.

MACRO NOTATION FOR TURING MACHINES

Next we need to describe how to:

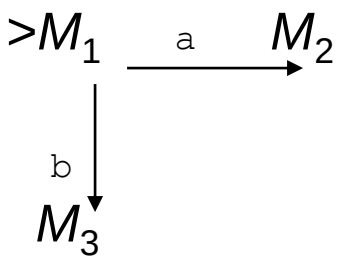
- Check the tape and branch based on what character we see, and
- Combine the basic machines to form larger ones.

To do this, we need two forms:

- $M_1\ M_2$
- $M_1 \xrightarrow{\text{<condition>}} M_2$

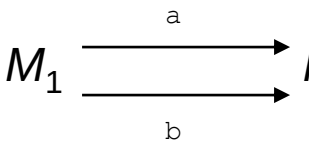
NOTATION FOR TURING MACHINES

Example:

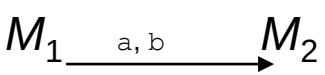


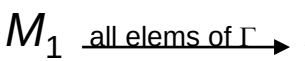
- Start in the start state of M_1.
- Compute until M_1 reaches a halt state.
- Examine the tape and take the appropriate transition.
- Start in the start state of the next machine, etc.
- Halt if any component reaches a halt state and has no place to go.
- If any component fails to halt, then the entire machine may fail to halt.

SHORTHANDS

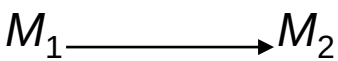


becomes





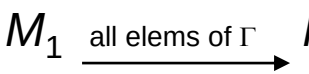
becomes



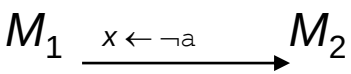
or

M_1M_2

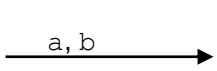
Storing values in variables



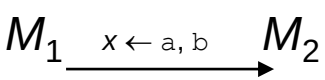
becomes



and *x* takes on the value of the current square



becomes



and *x* takes on the value of the current square

SHORTHANDS

We can use the value of a variable in two ways.

1. The first is a condition on a transition. So we can write:

$$M_1 \xrightarrow{x = y} M_2$$

if $x = y$ then take the transition

Note that we write $\leftarrow$ for assignment and = or $\neq$ for Boolean comparison

2. We can also write the value of a variable

Let M=

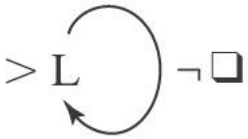
$$> \xrightarrow{x \leftarrow \neg\square} Rx$$

if the current square is not blank, go right and copy it.

SHORTHANDS

Find the first blank square to the right of the current square.

$R_{\square}$



Find the first blank square to the left of the current square.

$L_{\square}$

Find the first nonblank square to the right of the current square.

$R_{\neg\square}$

Find the first nonblank square to the left of the current square

$L_{\neg\square}$

SHORTHANDS

L_{a}

Find the first occurrence of `a` to the left of the current square.

$R_{a,b}$

Find the first occurrence of `a` or `b` to the right of the current square.

$L_{a,b} \xrightarrow{a} M_1$

$\downarrow b$

M_2

Find the first occurrence of `a` or `b` to the left of the current square, then go to M_1 if the detected character is `a`; go to M_2 if the detected character is `b`.

$L_{x \leftarrow a,b}$

Find the first occurrence of `a` or `b` to the left of the current square and set *x* to the value found.

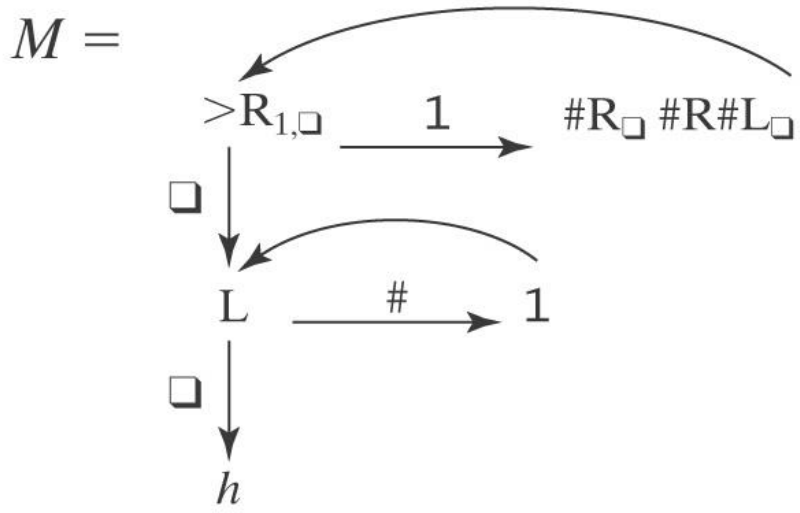
$L_{x \leftarrow a,b} Rx$

Find the first occurrence of `a` or `b` to the left of the current square, set *x* to the value found, move one square to the right, and write *x* (`a` or `b`).

TURING MACHINES
Example

Input: ❑w $w \in \{1\}^*$
Output: ❑w^3

Example: ❑111❑❑❑❑❑❑❑❑❑❑❑❑❑



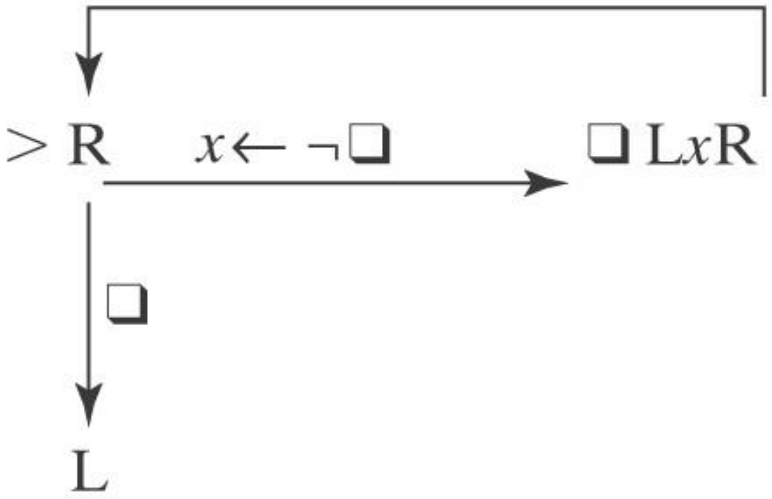
TURING MACHINES

Example: Shifting machine , $S_{\leftarrow}$

Input: ❑*u*❑*w*❑

Output: ❑*uw*❑

Example: ❑ba❑abba❑❑❑❑❑❑❑❑❑❑❑❑



TURING MACHINES AS LANGUAGE RECOGNIZERS

Convention: We will write the input on the tape as:

❑*w*❑, *w* contains no ❑s

The initial configuration of *M* will then be:

(s, $\underline{❑}w$)

Given a language *L*, we would like to design a *TM* that takes as input some string *w* and tells us whether or not $w \in L$. This is possible for many languages, such us all regular, context-free and the others such as $A^nB^nC^n$

However there are languages for which the power or a *TM* is not enough

TURING MACHINES AS LANGUAGE RECOGNIZERS

Let $M = (K, \Sigma, \Gamma, \delta, s, \{y, n\})$.

- *M* ***accepts*** a string *w* iff $(s, \underline{\sqcup} w) \vdash_M^* (y, w')$ for some string *w′*.
  - We call (y, w') an *accepting* configuration
- *M* ***rejects*** a string *w* iff $(s, \underline{\sqcup} w) \vdash_M^* (n, w')$ for some string *w′*.
  - We call (n, w') a *rejecting* configuration

TURING MACHINES AS LANGUAGE RECOGNIZERS

M ***decides*** a language $L \subseteq \Sigma^*$ iff:
For any string $w \in \Sigma^*$ it is true that:

- if $w \in L$ then *M* accepts *w*, and
- if $w \notin L$ then *M* rejects *w*.

A language *L* is ***decidable*** iff there is a Turing machine *M* that decides it. In this case, we will say that *L* is in ***D***.

TURING MACHINES
a DECIDING example

$A^nB^nC^n = \{a^n b^n c^n : n \geq 0\}$

Example: ␣aabbcc␣␣␣␣␣␣␣␣␣

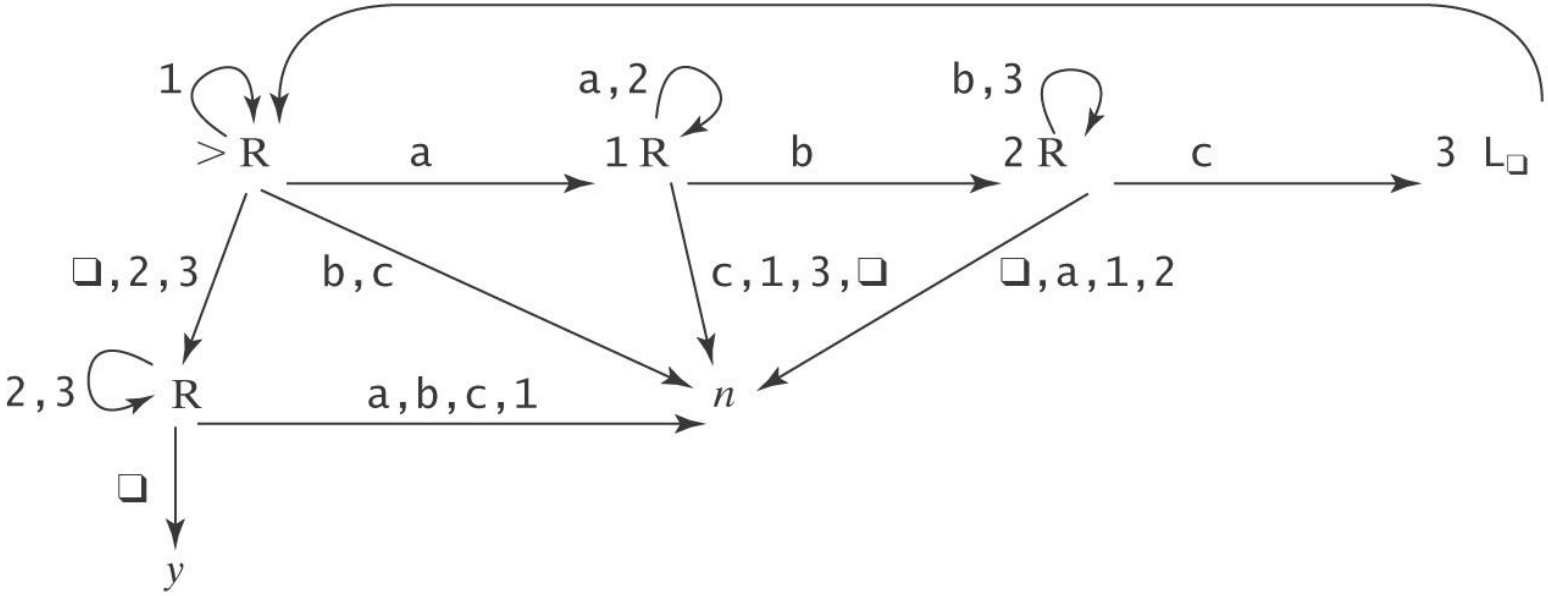
Example: ␣aaccb␣␣␣␣␣␣␣␣␣

TURING MACHINES
a DECIDING example

$A^nB^nC^n = \{a^nb^nc^n : n \geq 0\}$

Example: ❑aabbcc❑❑❑❑❑❑❑❑❑

Example: ❑aaccb❑❑❑❑❑❑❑❑❑



TURING MACHINES
another DECIDING example

WcW = {wcw : $w \in$ {a, b}*}

Example: ❑abbcabb❑❑❑

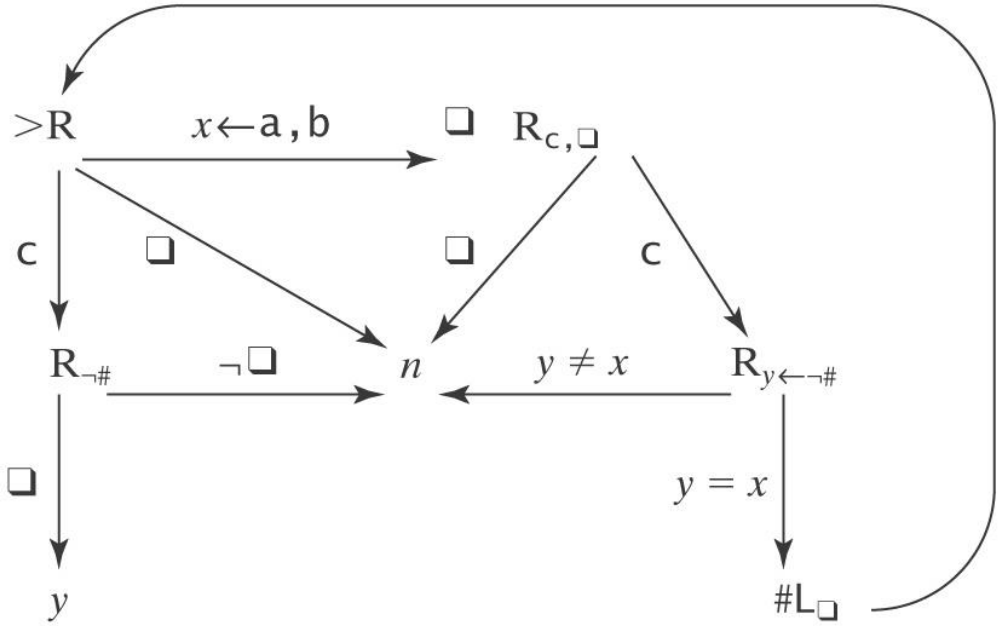
Example: ❑acabb❑❑❑

TURING MACHINES
another DECIDING example

WcW = {*w*c*w* : *w* ∈ {a, b}*}

Example: ❑abbcabb❑❑❑

Example: ❑acabb❑❑❑



SEMIDECIDING A LANGUAGE

Let Σ_M be the input alphabet to a TM M. Let $L \subseteq \Sigma_M^*$.

M ***semidecides*** L iff, for any string $w \in \Sigma_M^*$:

- $w \in L \rightarrow M$ accepts w
- $w \notin L \rightarrow M$ does not accept w.

M may either:
reject or
fail to halt.

A language L is ***semidecidable*** iff there is a Turing machine that semidecides it. We define the set ***SD*** to be the set of all semidecidable languages.

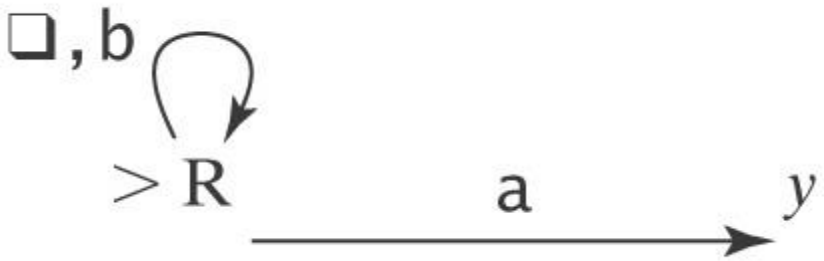
TURING MACHINES
a SEMIDECIDING example

Let L = $\mathtt{b}^*\mathtt{a}(\mathtt{a} \cup \mathtt{b})^*$

We can build M to semidecide L:

1. Loop
    1.1 Move one square to the right. If the character under the read head is an $\mathtt{a}$, halt and accept.

In our macro language, M is:



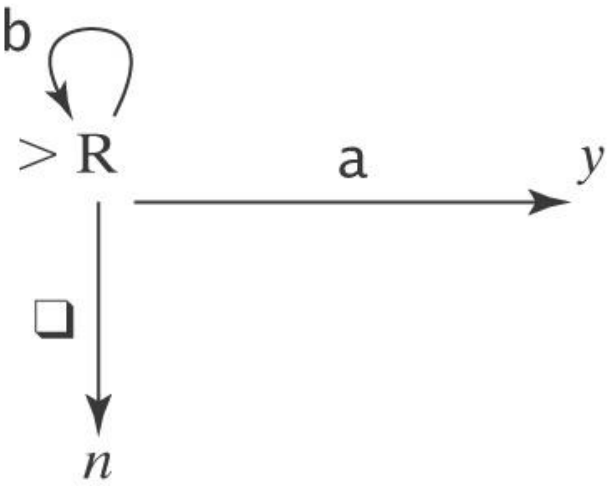
TURING MACHINES
a DECIDING example

L = b*a(a ∪ b)*. We can also decide L:

Loop:

1.1 Move one square to the right.
1.2 If the character under the read/write head is an a, halt and accept.
1.3 If it is ❑, halt and reject.

In our macro language, M is:



COMPUTING FUNCTIONS

- When a TM halts there is a value on its tape.
- When we build deciding or semideciding TMs we ignore the value.
- We don't have to!

- We can use TMs to compute functions using what's left on the tape

COMPUTING FUNCTIONS

Let $M = (K, \Sigma, \Gamma, \delta, s, \{h\})$. Its initial configuration is $(s, \underline{\sqcup} w)$.

Define $M(w) = z$ iff $(s, \underline{\sqcup} w) \vdash_M^* (h, \underline{\sqcup} z)$.

Let $\Sigma' \subseteq \Gamma$ be M's output alphabet.
Let f be any function from Σ^* to Σ'^*.

M ***computes*** f iff, for all $w \in \Sigma^*$:

- If w is an input on which f is defined: $M(w) = f(w)$.
- Otherwise $M(w)$ does not halt.

A function f is ***recursive*** or ***computable*** iff there is a Turing machine M that computes it and that always halts.

EXAMPLE OF COMPUTING A FUNCTION

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Let $f(w) = ww$.

Input: ❑$\underline{w}$❑❑❑❑❑❑ Output: ❑$\underline{ww}$❑

Define the copy machine C:

❑$\underline{w}$❑❑❑❑❑❑ → ❑w❑$\underline{w}$❑

Remember the $S_{\leftarrow}$ machine:

❑u❑$\underline{w}$❑ → ❑$\underline{uw}$❑

Then the machine to compute f is just $>C\ S_{\leftarrow}\ L_{❑}$

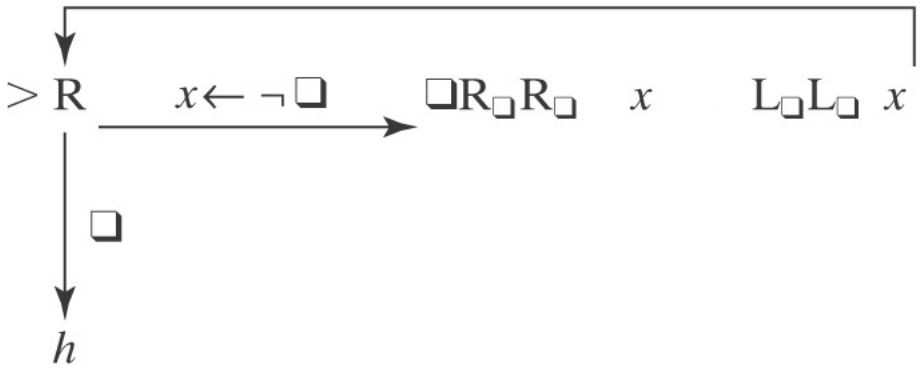
EXAMPLE OF COMPUTING A FUNCTION

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Let $f(w) = ww$.

Input: ❑w❑❑❑❑❑❑ Output: ❑ww❑

Define the copy machine C:

❑w❑❑❑❑❑❑ → ❑w❑w❑



Remember the $S_{\leftarrow}$ machine:

❑u❑w❑ → ❑uw❑

Then the machine to compute f is just $>C\ S_{\leftarrow}\ L$❑

COMPUTING NUMERIC FUNCTIONS

Example: $succ(n) = n + 1$

We will represent n in binary. So $n \in 0 \cup 1\{0, 1\}^*$

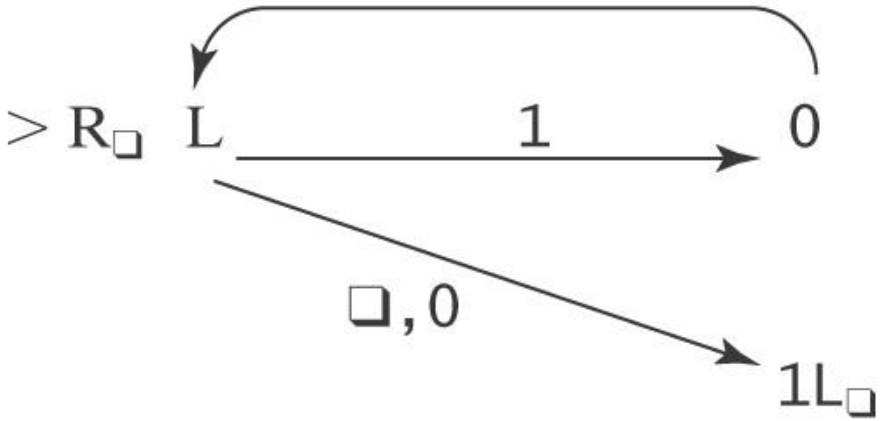
Input: ❑*n*❑❑❑❑❑❑ Output: ❑*n*+1❑
❑1111❑❑❑❑ Output: ❑10000❑

COMPUTING NUMERIC FUNCTIONS

Example: $succ(n) = n + 1$

We will represent n in binary. So $n \in 0 \cup 1\{0, 1\}^*$

Input: ❑n❑❑❑❑❑❑ Output: ❑n+1❑
❑1111❑❑❑❑ Output: ❑10000❑



NOT ALL FUNCTIONS ARE COMPUTABLE

Let *T* be the set of all TMs that:

- Have tape alphabet $\Gamma = \{\square, 1\}$, and
- Halt on a blank tape.

Define the **busy beaver functions** $S(n)$ and $\Sigma(n)$:

- $S(n)$: the maximum number of steps that are executed by any element of *T* with *n*-nonhalting states, when started on a blank tape, before it halts.
- $\Sigma(n)$: the maximum number of 1's that are left on the tape by any element of *T* with *n*-nonhalting states, when it halts.

n	***S*(*n*)**	**$\Sigma(n)$**
1	1	1
2	6	4
3	21	6
4	107	13
5	$\geq$ 47,176,870	$\geq$ 4098
6	$\geq 7.4 \times 10^{36534}$	$\geq 3.5 \times 10^{18267}$

Watch: https://www.youtube.com/watch?v=CE8UhcyJS0I

TURING MACHINES

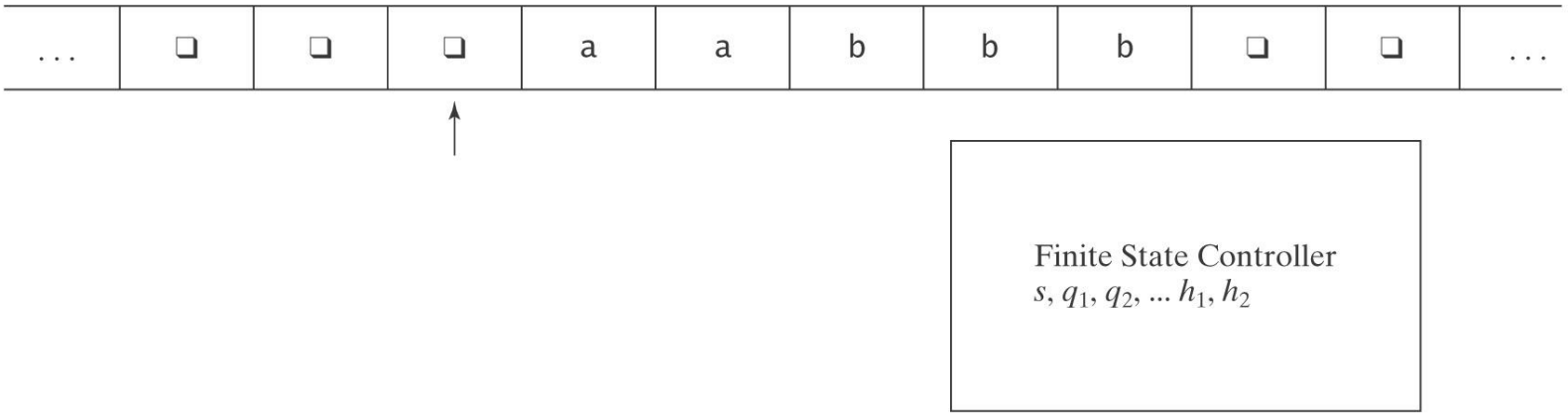
Turing machines	Are more powerful than any of the other formalisms we have studied so far.

Turing machines	Are a **lot** harder to work with than all the real computers we have available.

Why bother?

The very simplicity that makes it hard to program Turing machines makes it possible to reason formally about what they can do. If we can, once, show that anything a real computer can do can be done (albeit clumsily) on a Turing machine, then we have a way to reason about what real computers can do.

TURING MACHINE EXTENTIONS



At each step, the machine must:

- choose its next state,
- write on the current square, and
- move left or right.

Turing Machine Extensions

There are many extensions we might like to make to our basic Turing machine model. But:

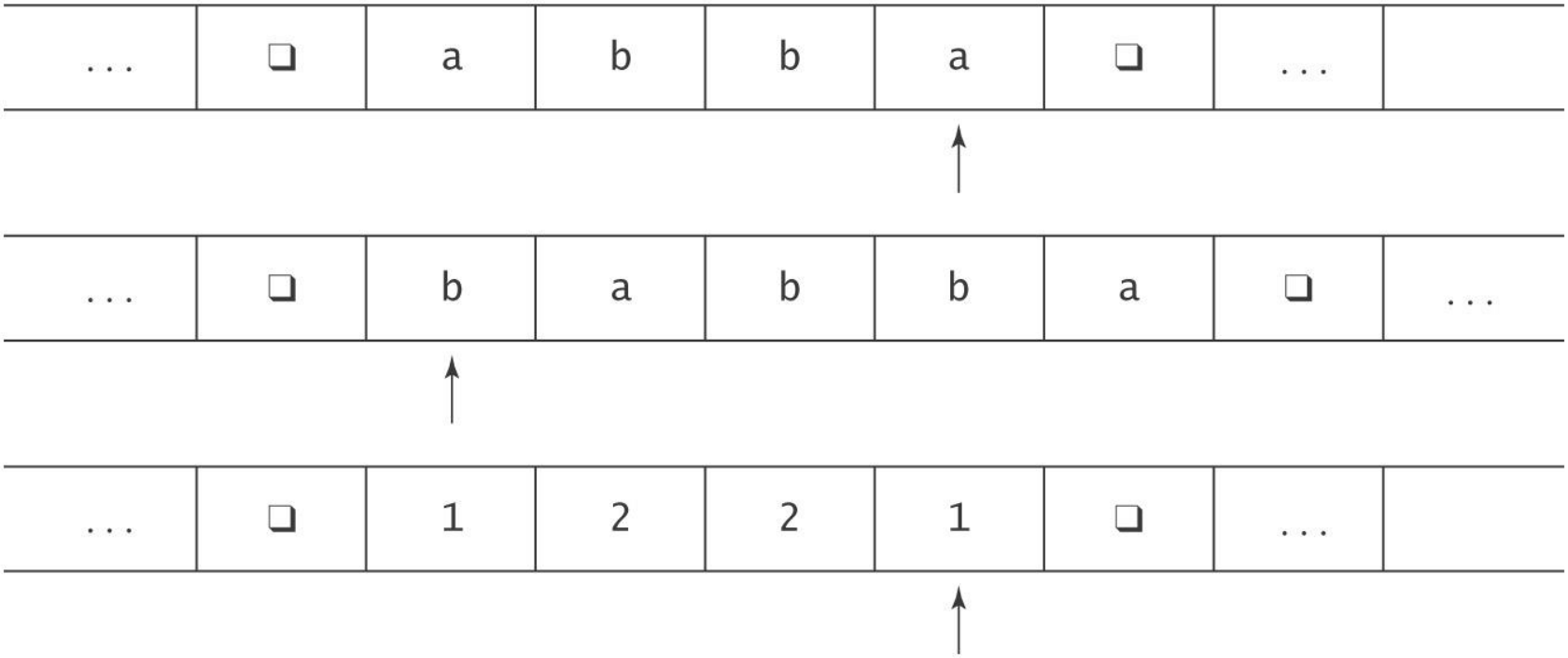
We can show that every extended machine has an equivalent basic machine.

We can also place a bound on any change in the complexity of a solution when we go from an extended machine to a basic machine.

Some possible extensions:

- Multiple tape TMs
- Nondeterministic TMs

MULTIPLE TAPES



A configuration of k-tape Turing Machine is a *k+1* tuple
(*state, tape*$_1$*, tape*$_2$*,…,tape*$_k$)

Initial configuration (*s*, ❑*w,* ❑*,….,*❑,).

MULTIPLE TAPES

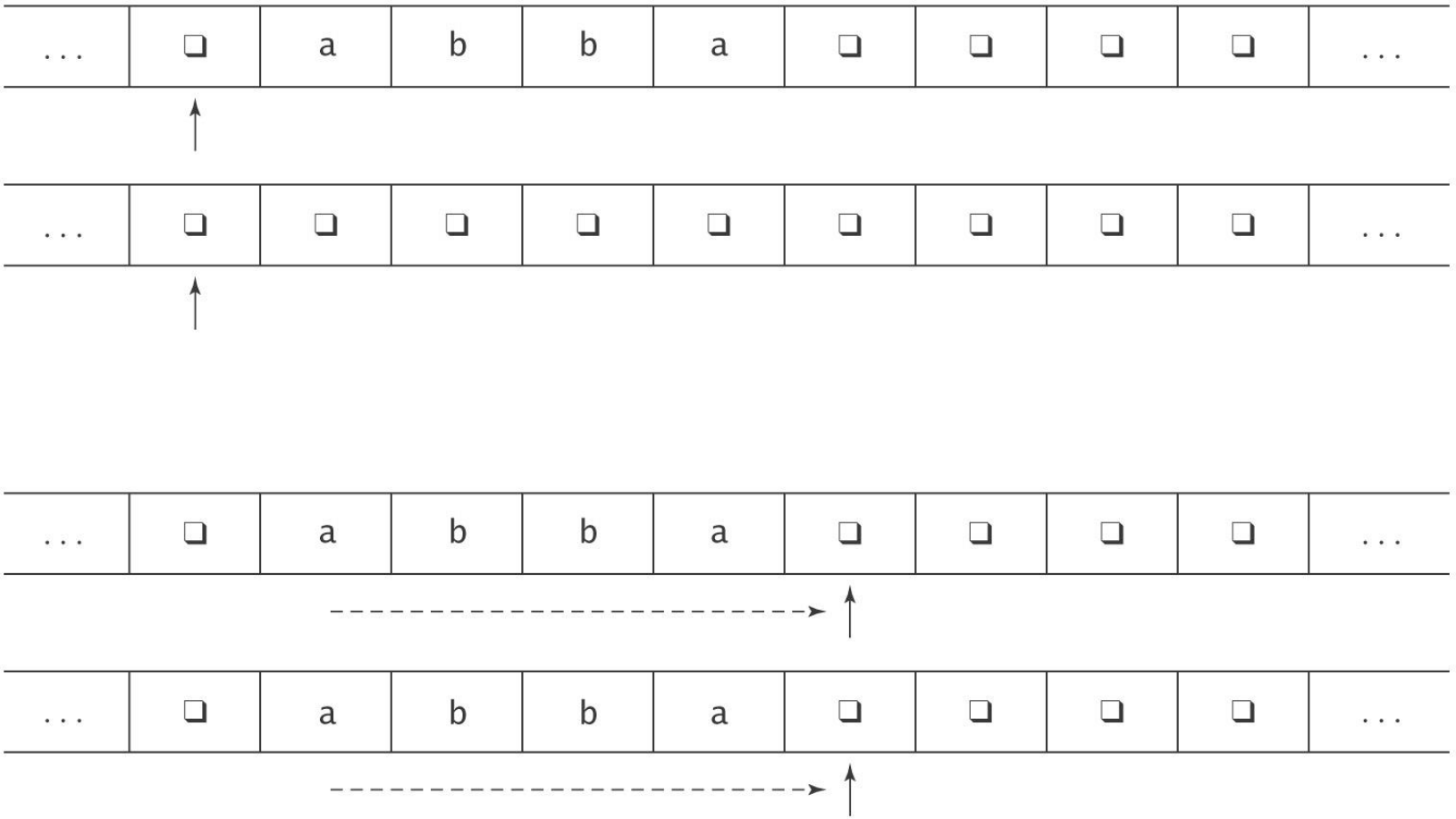
The transition function for a *k*-tape Turing machine:

$$
\begin{array}{lll}
((K\text{-}H) , \Gamma_1 \text{ to} & (K , \Gamma_{1'}, \{\leftarrow, \rightarrow, \uparrow\} \\
\quad , \Gamma_2 & \quad , \Gamma_{2'}, \{\leftarrow, \rightarrow, \uparrow\} \\
\quad , \; . & \quad , \; . \\
\quad , \; . & \quad , \; . \\
\quad , \Gamma_k) & \quad , \Gamma_{k'}, \{\leftarrow, \rightarrow, \uparrow\})
\end{array}
$$

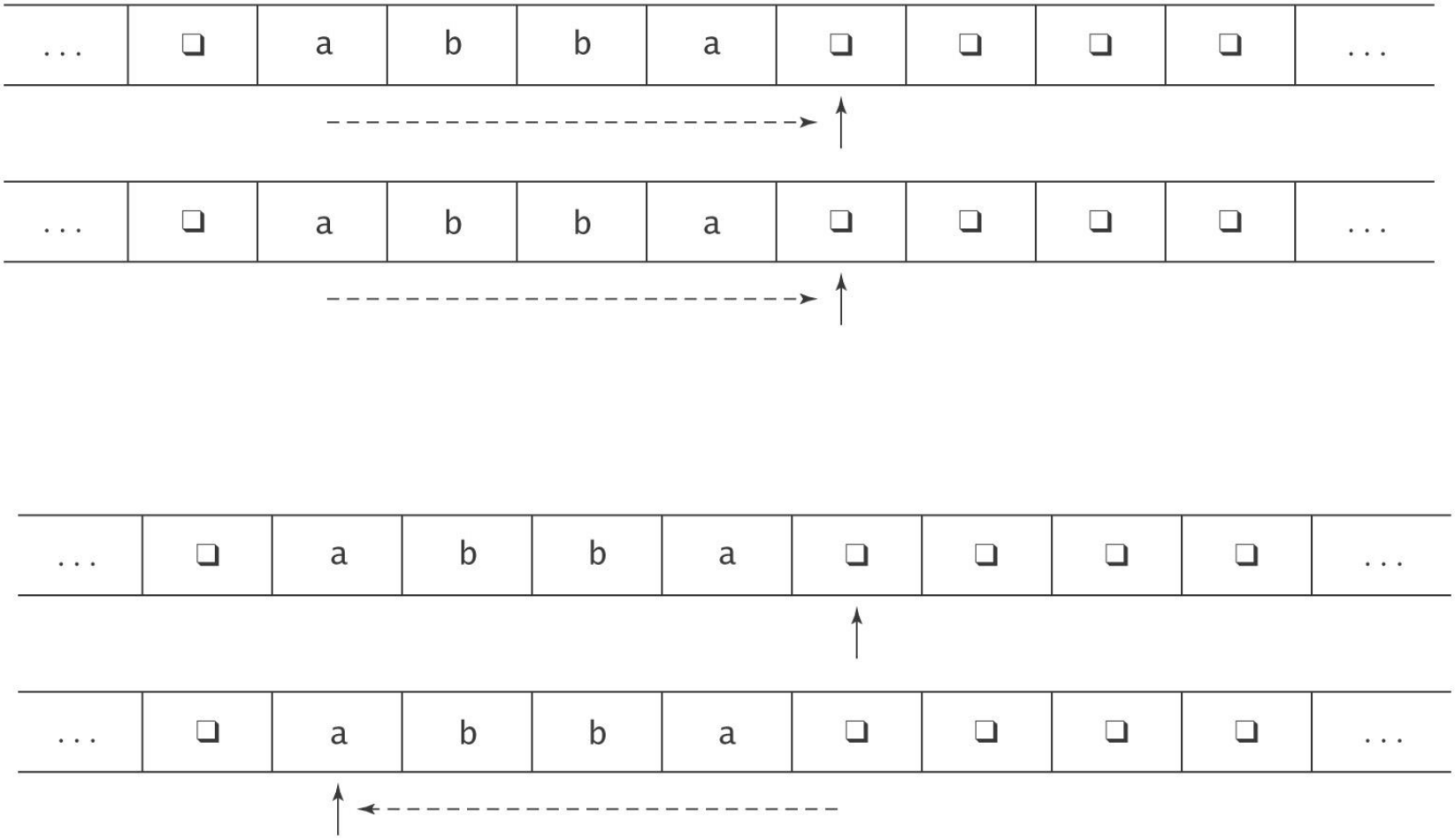
Input: as before on tape 1, others blank.
Output: as before on tape 1, others ignored.

Note: tape head is allowed to stay put.

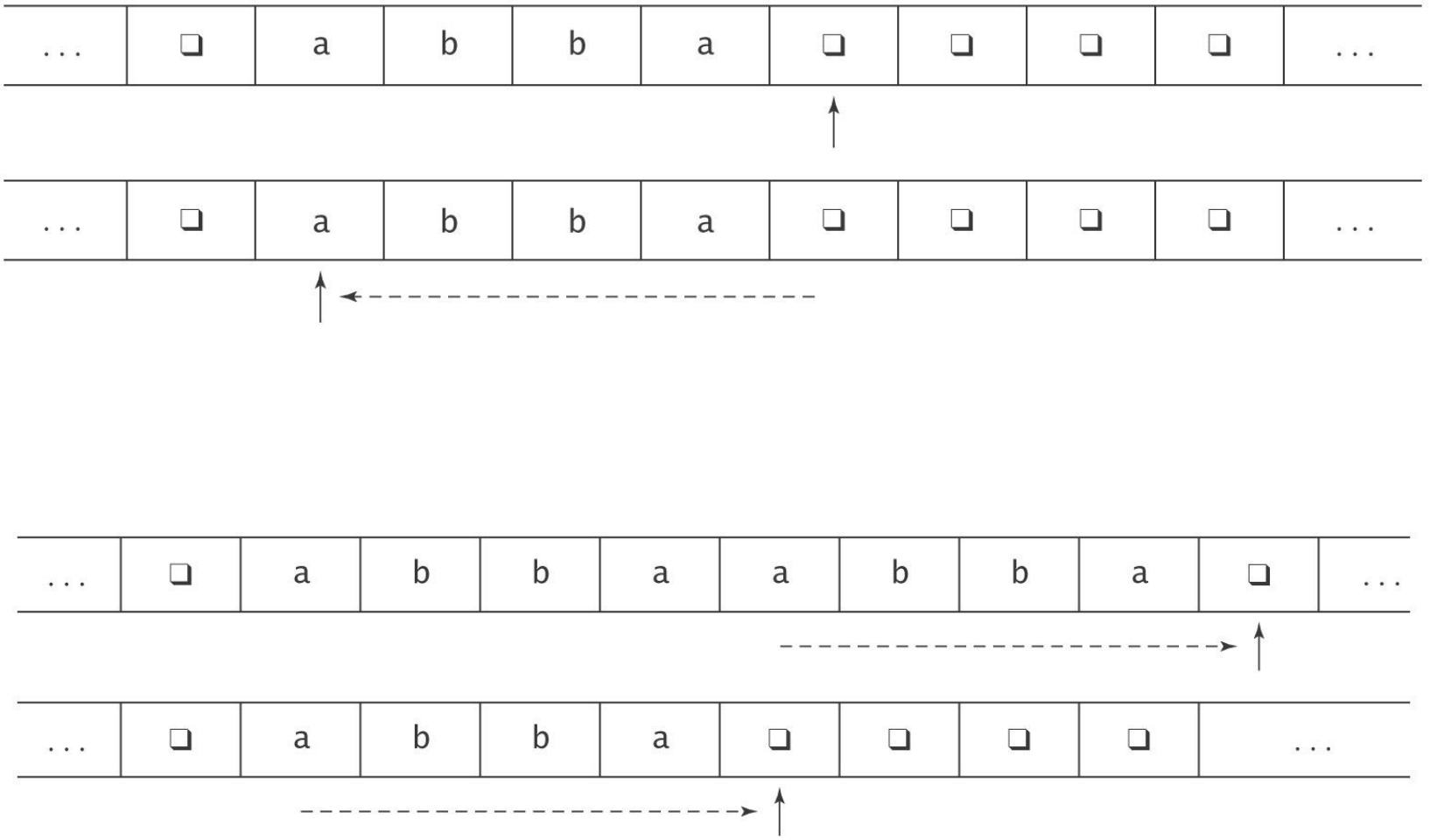
EXAMPLE: COPYING A STRING



EXAMPLE: COPYING A STRING



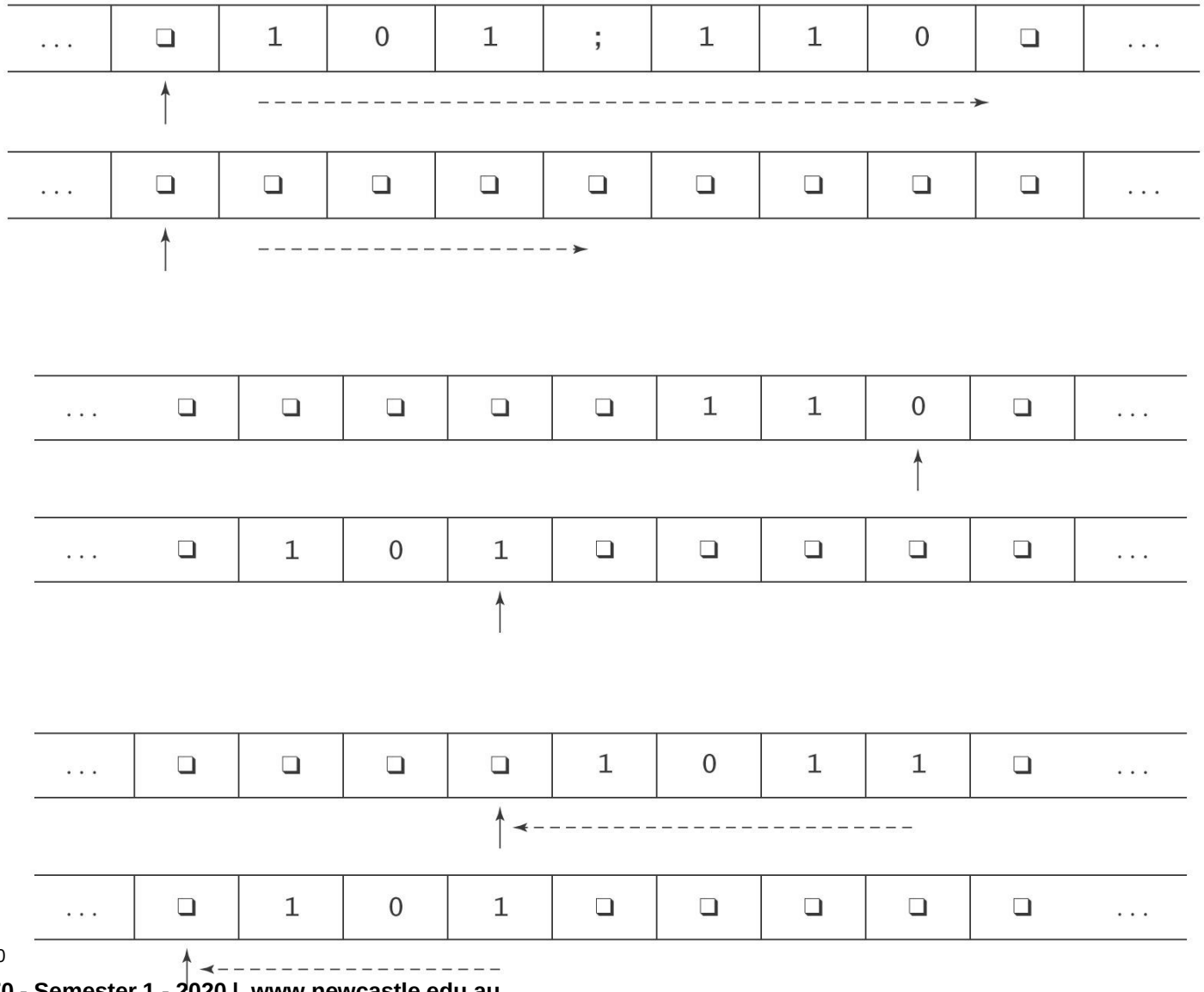
EXAMPLE: COPYING A STRING



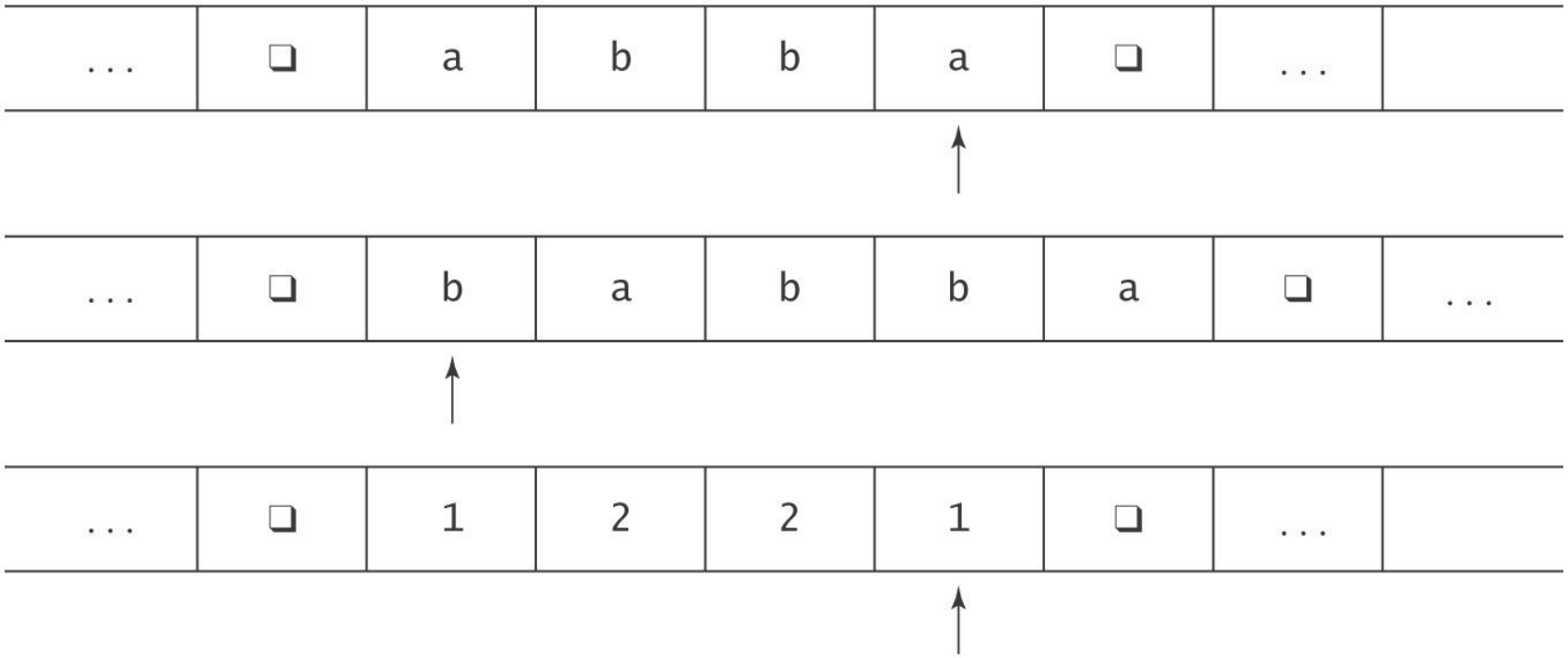
Example: $A^nB^nC^n$

a a a a b b b b c c c c

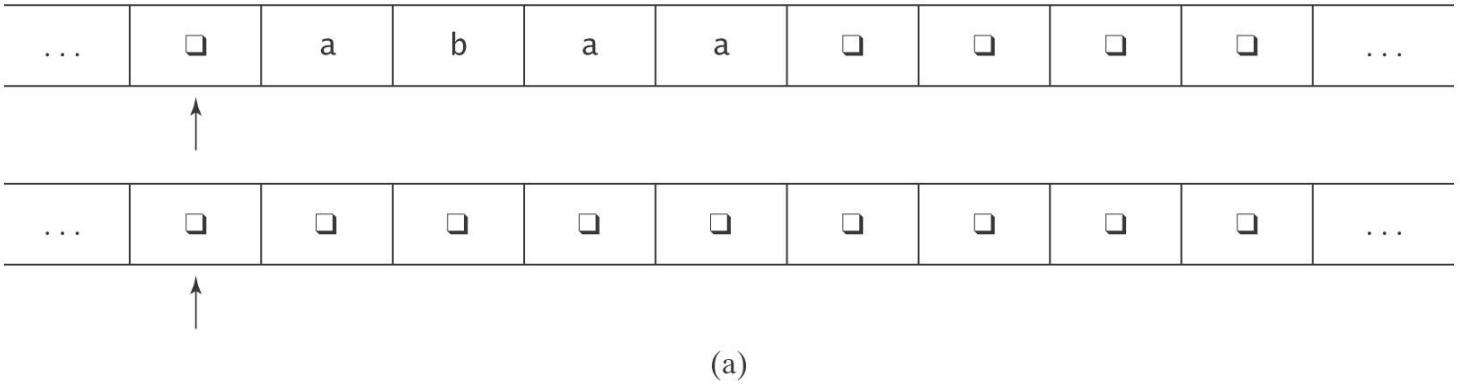
ANOTHER TWO TAPE EXAMPLE: ADDITION



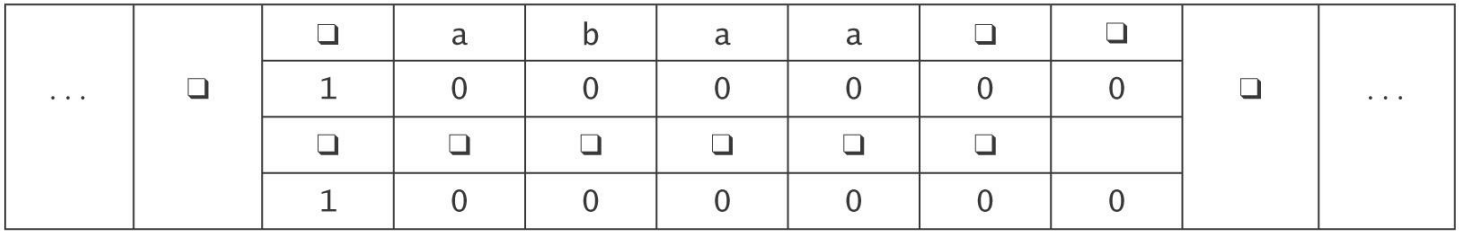
How to Represent Multiple Tapes?



The Representation



(a)



(b)

Alphabet (Σ') of $M' = \Gamma \cup (\Gamma \times \{0, 1\})^k$:

❑, a, b, (❑, 1, ❑, 1), (a, 0, ❑ ,0), (b, 0, ❑, 0), …

The Operation of M'

. . .	❑	❑	a	b	a	a	❑	❑	❑	. . .
		1	0	0	0	0	0	0		
		❑	❑	❑	❑	❑	❑			
		1	0	0	0	0	0	0		

1. Set up the multitrack tape.
2. Simulate the computation of *M* until (if) *M* would halt:
    - 2.1 Scan left and store in the state the *k*-tuple of characters under the read heads. Move back right.
    - 2.2 Scan left and update each track as required by the transitions of *M*. If necessary, subdivide a new square into tracks. Move back right.
3. When *M* would halt, reformat the tape to throw away all but track 1, position the head correctly, then go to *M*'s halt state.

How Many Steps Does M' Take?

Let: w be the input string, and
n be the number of steps it takes M to execute.

Step 1 (initialization): $\mathcal{O}(|w|)$.

Step 2 (computation):
Number of passes = n.
Work at each pass: 2.1 = 2 · (length of tape).
= 2 · ($|w| + n$).
2.2 = 2 · ($|w| + n$).
Total: $\mathcal{O}(n \cdot (|w| + n))$.

Step 3 (clean up): $\mathcal{O}$(length of tape).

Total: $\mathcal{O}(n \cdot (|w| + n)) = \mathcal{O}(n^2)$. *

* assuming that $n \geq w$

ADDING TAPES ADDS NO POWER

Theorem: Let M be a k-tape Turing machine for some $k \geq 1$. Then there is a standard TM M' where $\Sigma \subseteq \Sigma'$, and:

- On input x, M halts with output z on the first tape iff M' halts in the same state with z on its tape.
- On input x, if M halts in n steps, M' halts in $\mathcal{O}(n^2)$ steps.

Proof: By construction.

IMPACT OF NONDETERMINISM

- FSMs
  - Power — NO
  - Complexity
    - Time — NO
    - Space — YES
- PDAs
  - Power — YES
- Turing machines
  - Power — NO
  - Complexity — ?

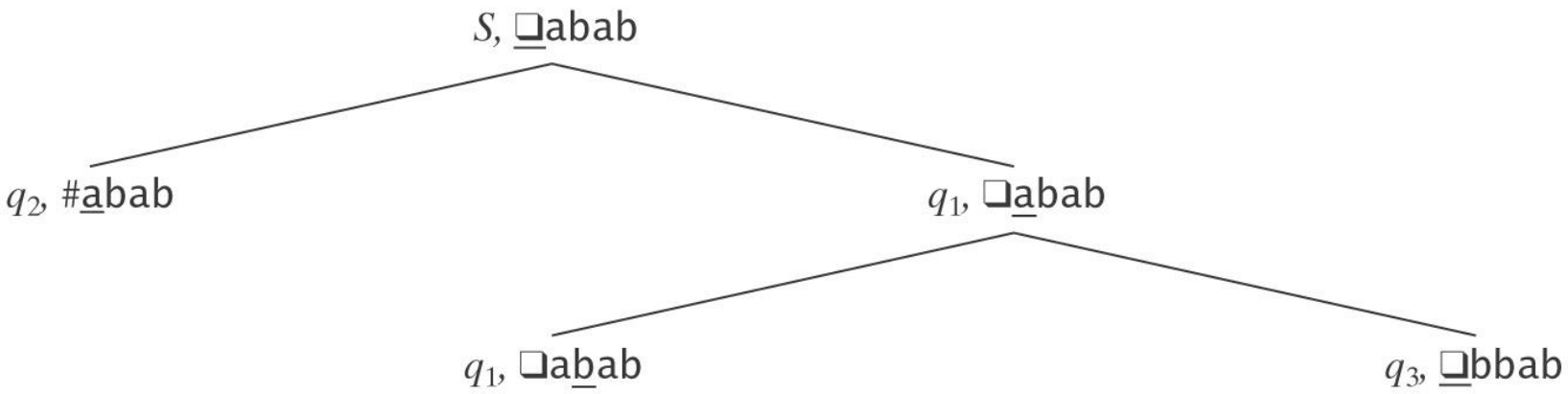
NONDETERMINISTIC TURING MACHINES

A **nondeterministic** TM is a sixtuple $(K, \Sigma, \Gamma, \Delta, s, H)$.

Δ is a ***subset*** of:

$$((K - H) \times \Gamma) \times (K \times \Gamma \times \{\leftarrow, \rightarrow\})$$

NONDETERMINISTIC TURING MACHINES



What does it mean for a nondeterministic Turing machine to:

- Decide a language
- Semidecide a language
- Compute a function

NONDETERMINISTIC DECIDING

Let $M = (K, \Sigma, \Gamma, \Delta, s, \{y, n\})$ be a nondeterministic TM.

Let w be an element of Σ^*.

M ***accepts*** w iff *at least one* of its computations accepts.

M ***rejects*** w iff all of its computations reject.

M ***decides*** a language $L \subseteq \Sigma^*$ iff, $\forall w$:

- There is a finite number of paths that M can follow on input w,
- All of those paths halt, and
- $w \in L$ iff M accepts w.

AN EXAMPLE OF NONDETERMINISTIC DECIDING

$L = \{w \in \{0, 1\}^* : w$ is the binary encoding of a composite number$\}$.

M decides L by doing the following on input w:

1. Nondeterministically choose two positive binary numbers such that:

   $2 \leq |p|$ and $|q| \leq |w|$.

   Write them on the tape, after w, separated by ;

   ❑110011;111;1111❑❑
   w p q

2. Multiply p and q and put the answer, A, on the tape, in place of p and q.

   ❑110011;1011111❑❑
   w A

3. Compare A and w. If equal, go to y. Else go to n.

NONDETERMINISTIC SEMIDECIDING

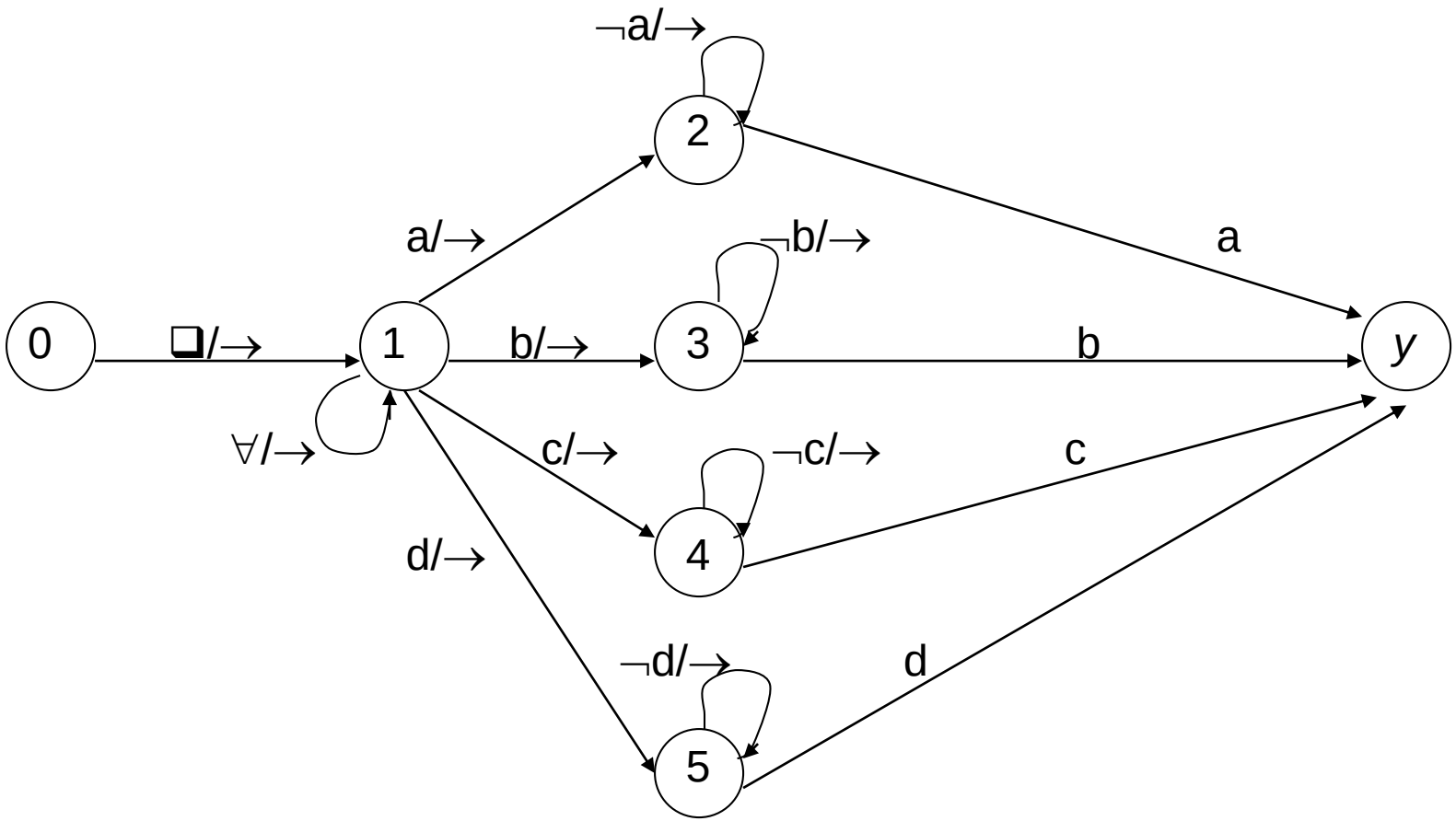
Let $M = (K, \Sigma, \Gamma, \Delta, s, H)$ be a nondeterministic TM.

We say that M ***semidecides*** a language $L \subseteq \Sigma^*$ iff:
for all $w \in \Sigma^*$:

$w \in L$ iff
$(s, \underline{\sqcup} w)$ yields at least one accepting configuration.

AN EXAMPLE OF NONDETERMINISTIC SEMIDECIDING

$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}^* :$ there are two of at least one letter$\}$



AN EXAMPLE OF NONDETERMINISTIC SEMIDECIDING

Let L = {descriptions of TMs that halt on at least one string}.

Let <M> mean the string that describes some TM M.

S semidecides L as follows on input <M>:

1. Nondeterministically choose a string w in Σ_M^* and write it on the tape.
2. Run M on w.
3. Accept.

We'll prove later that, in this case, semideciding is the best we can do.

NONDETERMINISTIC FUNCTION COMPUTATION

M computes a function *f* iff, $\forall w \in \Sigma^*$:

- All of *M*'s computations halt, and
- All of *M*'s computations result in *f*(*w*).

EQUIVALENCE OF DETERMINISTIC AND NONDETERMINISTIC TURING MACHINES

Theorem: If a nondeterministic TM *M* decides or semidecides a language, or computes a function, then there is a standard TM *M'* semideciding or deciding the same language or computing the same function.

Proof: (by construction)

We must do separate constructions for deciding/semideciding and for function computation.

References

- **Automata, Computability and Complexity. Theory and Applications**
  - By Elaine Rich
- Chapter 17:
  - Page : 266-391.