Turing Machines

COMP 455 – 002, Spring 2019

Turing Machines

- We want to study computable functions, or *algorithms*.
- In particular, we will look at algorithms for answering certain questions.
  - A question is *decidable* if and only if an algorithm exists to answer it.
- Example question: Is the complement of an arbitrary CFL also a CFL?
  - This question is *undecidable*—there is no algorithm that takes an *arbitrary* CFL and outputs "yes" if the complement is a CFL and "no" otherwise.

Such an algorithm does exist for any *specific* CFL.

Undecidability Example

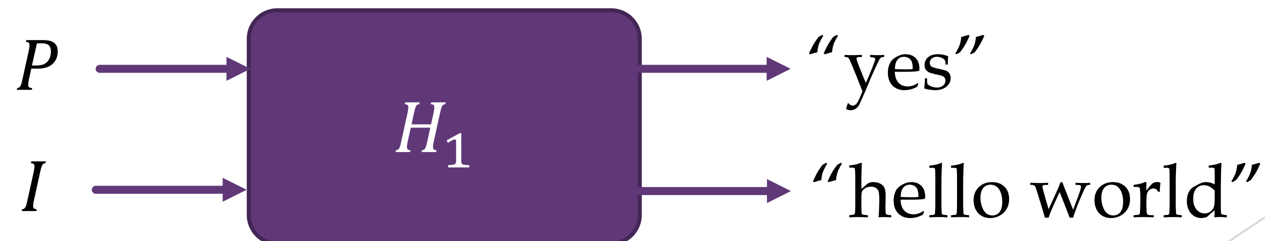
We can give an informal "proof" showing that an undecidable problem exists. Let's consider:

- **The "Hello World" problem**: Given a program *P* and an input to that program *I*, print "yes" if *P* prints "hello world" when run with input *I* and "no" otherwise.
  - *P* can be any program—including one that is very convoluted!

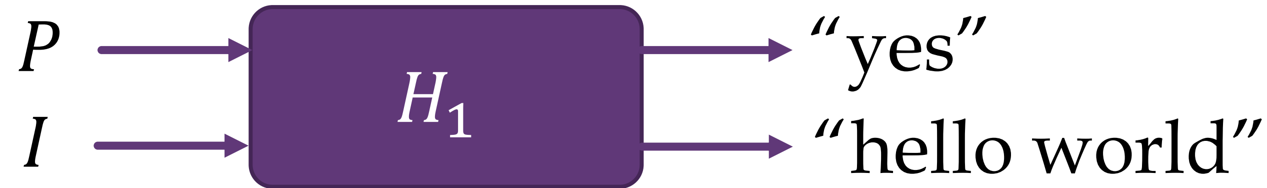
Undecidability Example

Suppose a program, H, exists that solves the "hello world" problem:

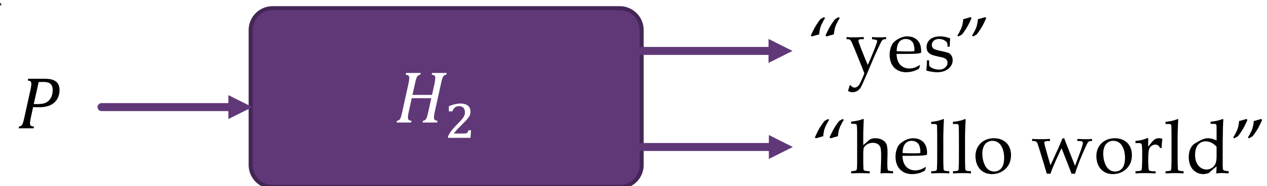
Now consider another program, H_1, which is the same as H, but prints "hello world" instead of "no":



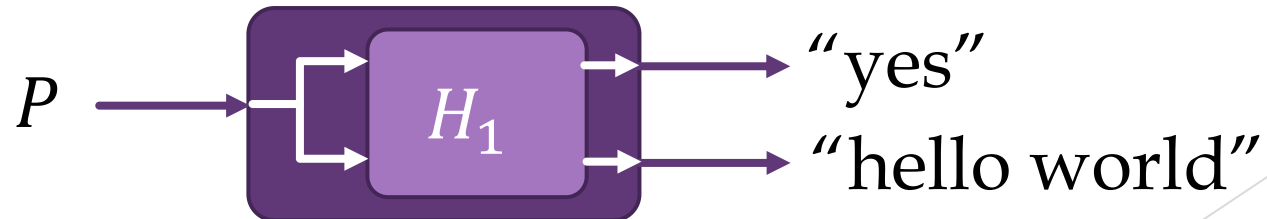
Undecidability Example



Next, construct a program H_2 that is identical to H_1 but only takes a single input, P, that it uses as both P and I in H_1:

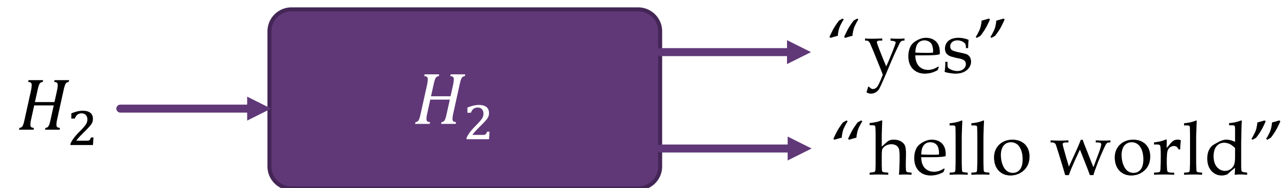


Another way to represent H_2:



Undecidability Example

Now consider what happens when we run H_2 with itself as input:



We have reached a paradox:

- Suppose H_2 prints "hello world" when fed H_2 as input. This means that H_2 must also print "yes" when fed H_2 as input!

Turing Machines

- To be able to rigorously do proofs like the "hello world" example, we need a *formal model for defining computable functions*.
- The **Turing Machine** has become the accepted model for formalizing functions.

Turing Machines

- A TM can be defined by a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$
  - Q: A finite set of states.
  - Γ: The tape alphabet.
  - Σ: The input alphabet. $\Sigma \subset \Gamma$
  - B: The **blank** tape symbol. ($B \in \Gamma$, and $B \notin \Sigma$)
  - δ: The **next move function**: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
  - q_0: The start state.
  - F: The set of final states.

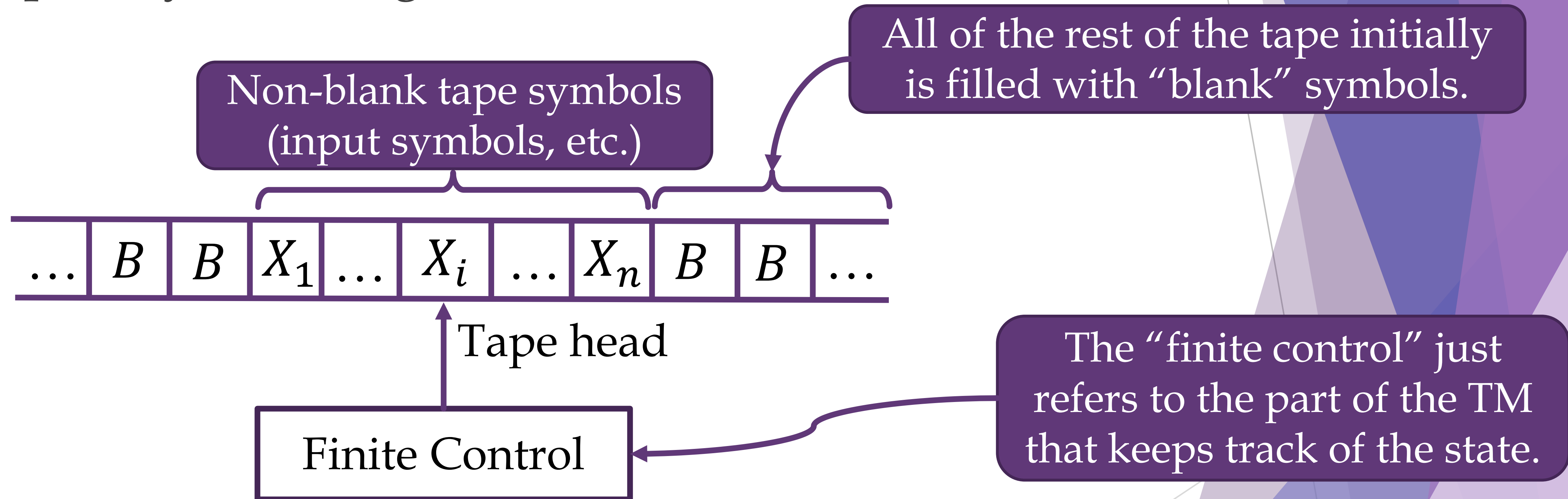
A TM's input starts out on the "tape", so the tape alphabet must include the input alphabet.

δ takes a state and tape symbol

δ returns a new state, a new tape symbol, and whether to move left or right.

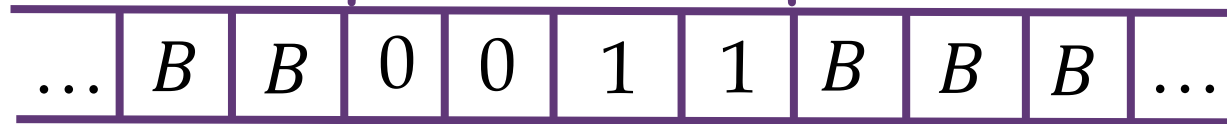
Turing Machines

Conceptually, a Turing Machine looks like this:

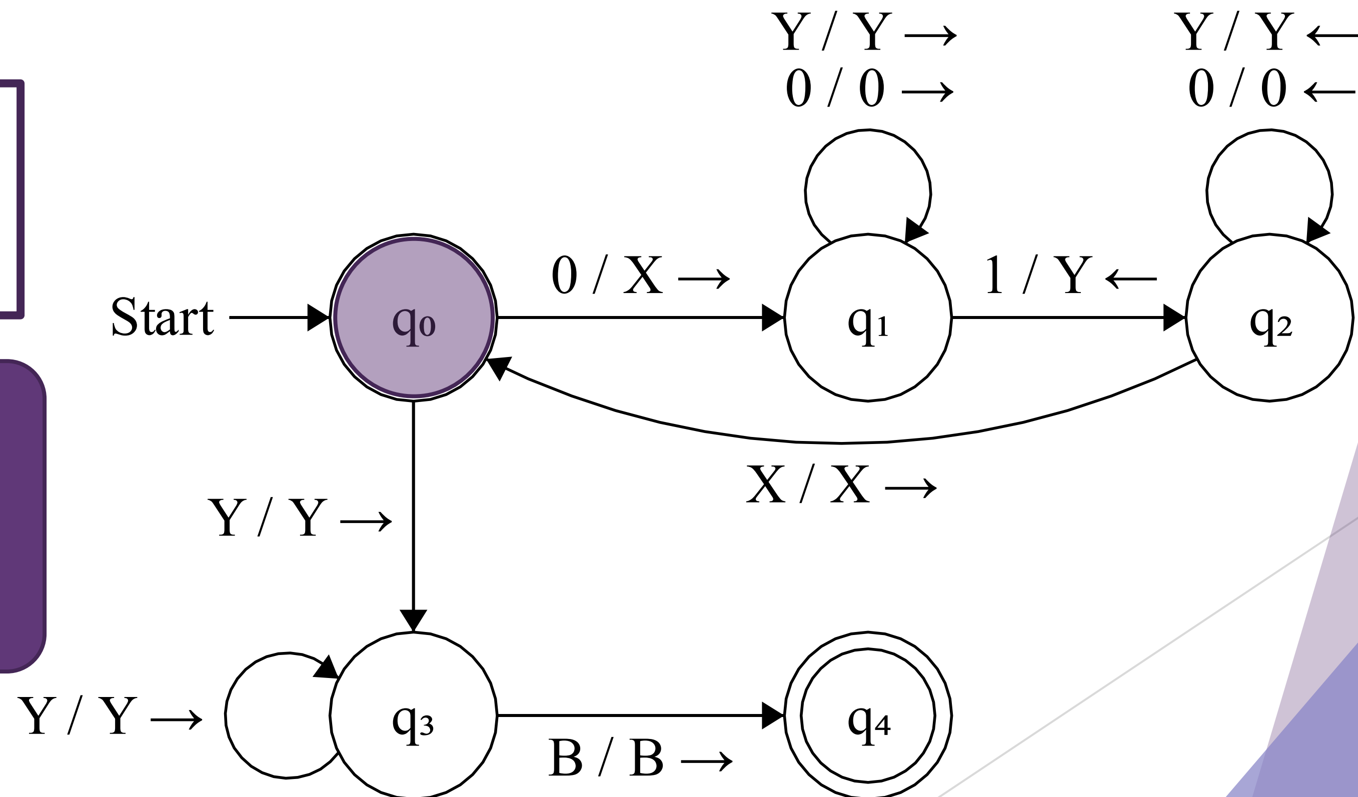


Initially, the input starts out on the tape, and the tape head starts at the leftmost input symbol.

Example: Turing Machine Accepting $0^n 1^n$

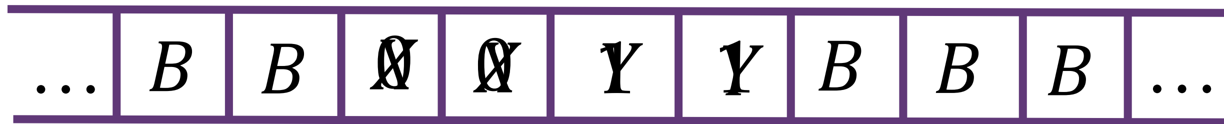


Finite Control
State = q_0

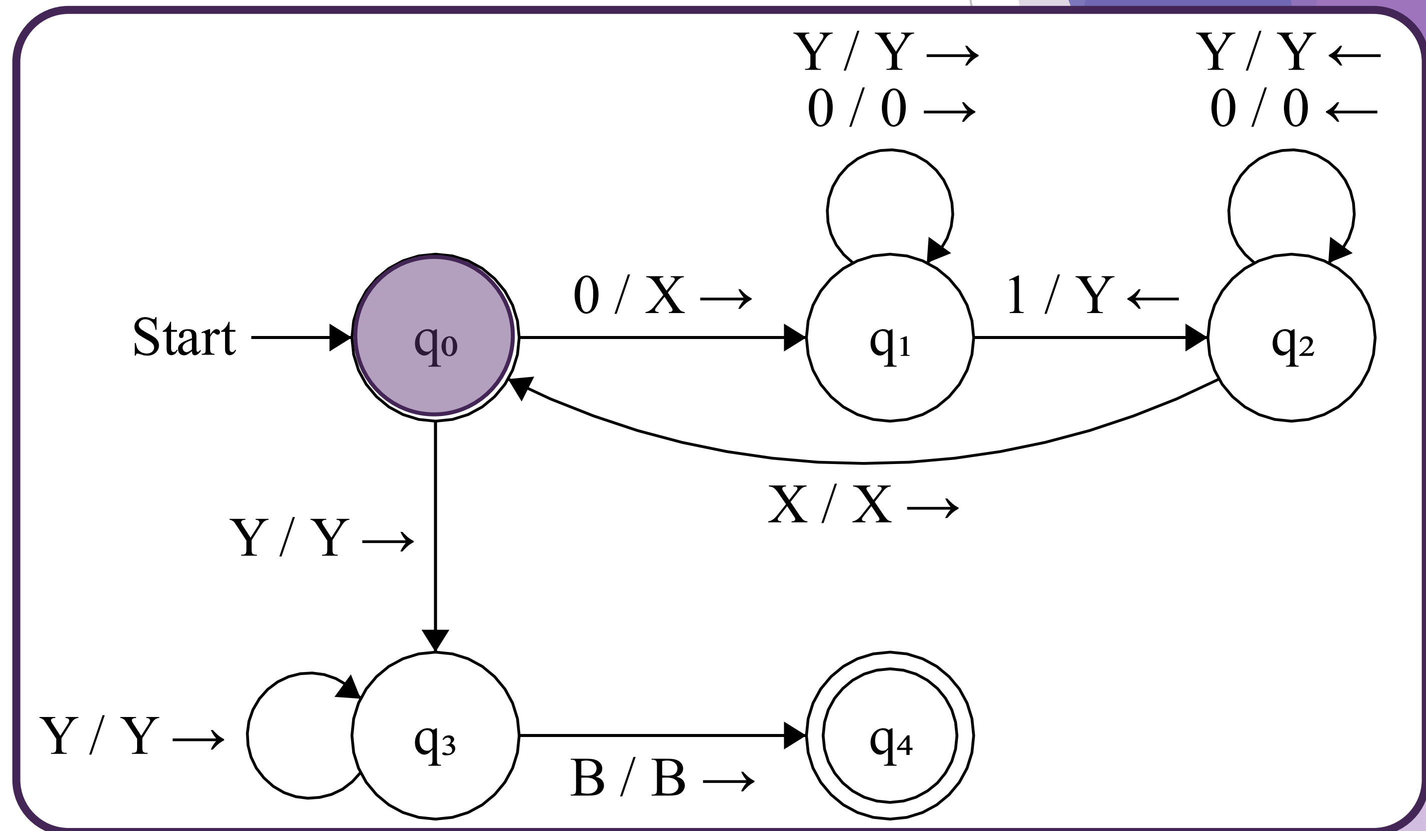


Initial configuration:
In start state, at leftmost input symbol on the tape.

Example: Turing Machine Accepting 0^n1^n



Finite Control
State = q_0 q_1 q_2 q_3 q_4

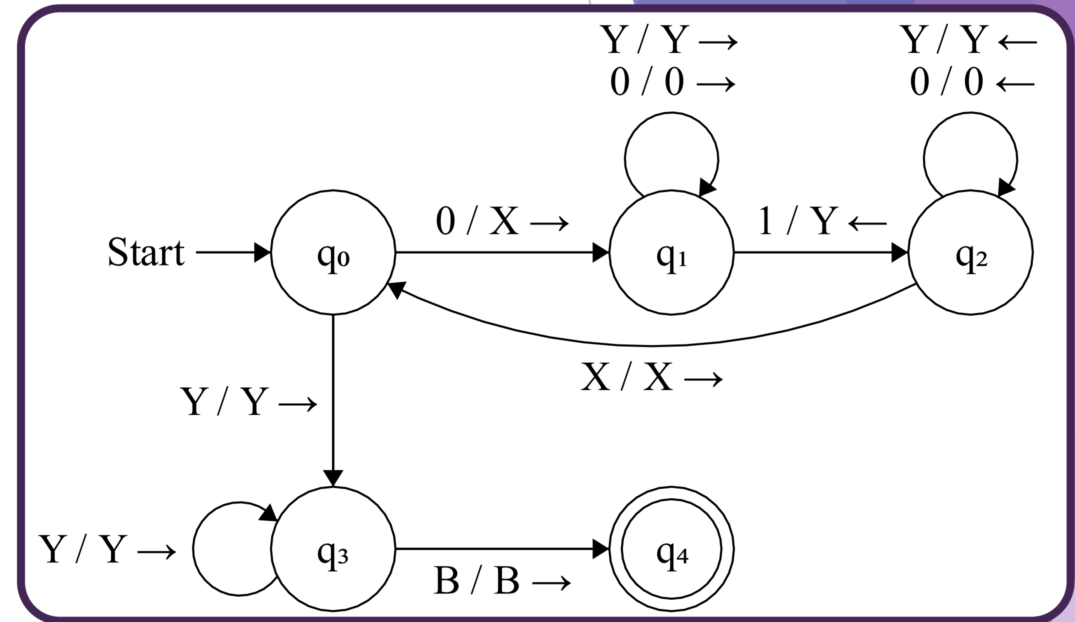


A Turing Machine halts when it can't make any more moves.

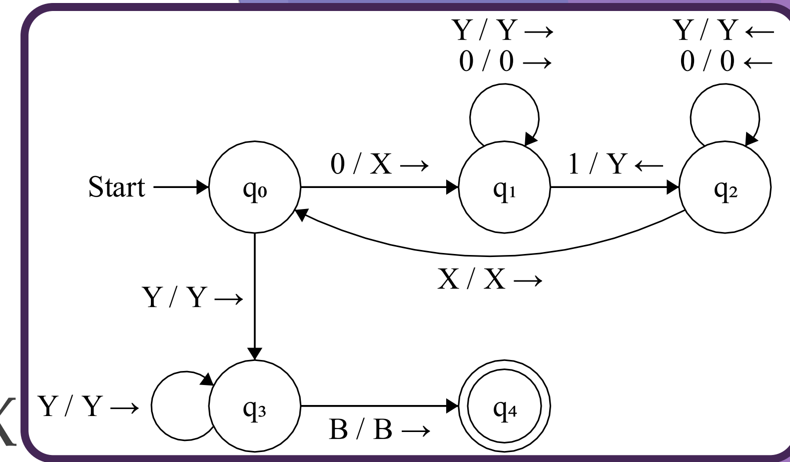
Example: Formal Notation

$(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$,
- $\Sigma = \{0, 1\}$,
- $\Gamma = \{0, 1, X, Y, B\}$,
- $F = \{q_4\}$,
- And δ is defined on the following slide…



Example: Formal Notation



- δ is as follows:
  - $\delta(q_0, 0) = (q_1, X, R)$: Change the leftmost 0 to X
  - $\delta(q_0, Y) = (q_3, Y, R)$: There are no more 0s
  - $\delta(q_1, 0) = (q_1, 0, R)$: Skip over 0s
  - $\delta(q_1, Y) = (q_1, Y, R)$: Skip over Ys
  - $\delta(q_1, 1) = (q_1, Y, L)$: Change the leftmost 1 to Y
  - $\delta(q_2, 0) = (q_2, 0, L)$: Skip over 0s
  - $\delta(q_2, Y) = (q_2, Y, L)$: Skip over Ys
  - $\delta(q_2, X) = (q_0, X, R)$: Move right of the rightmost X
  - $\delta(q_3, Y) = (q_3, Y, R)$: Make sure there are no more 1s
  - $\delta(q_3, B) = (q_4, B, R)$: There were no more 1s, accept.

"Seek" to the leftmost 1.

"Seek" to the leftmost 0.

Instantaneous Descriptions for TMs

- As with a PDA, we can also give an instantaneous description (ID) for a Turing Machine.
- An ID for a TM has the following form: $\alpha_1 q \alpha_2$.
  - q corresponds to both the state of the TM and the position of the tape head (q is written directly before the tape symbol the head is on).
  - $\alpha_1 \alpha_2$ = the tape's current contents, and only contains the non-blank portion, except in cases like $\alpha BBBq$ or $qBBB\alpha$.
- The depicted portion of the tape is always *finite*.

This indicates both the location of the tape head *and* the current state (q).

We can't write an infinite number of cells without either infinite input or an infinite number of moves.

Notation for TM Moves

- **Left moves**. Suppose $\delta(q, X_i) = (p, Y, L)$. Then,
  - $X_1 \dots X_{i-1} q X_i \dots X_n \underset{M}{\vdash} X_1 \dots X_{i-2} p X_{i-1} Y \dots X_n$.
    - The tape head moved from X_i to X_{i-1}.
    - The state changed from q to p.
    - X_i was replaced by Y.
  - Special case where $i = 1$:
    - $q X_1 X_2 \dots X_n \underset{M}{\vdash} p B Y X_2 \dots X_n$
  - Special case where $i = n$ and $Y = B$:
    - $X_1 \dots X_{n-1} q X_n \underset{M}{\vdash} X_1 \dots X_{n-2} p X_{n-1}$

We moved left past the start of the tape content, so we need to show the extra blank symbol (B).

We replaced the rightmost tape symbol (X_n) with a B, so we no longer include it in the ID.

Notation for TM Moves

- Right moves. Suppose $\delta(q, X_i) = (p, Y, R)$. Then,
  - $X_1 \ldots X_{i-1} q X_i \ldots X_n \underset{M}{\vdash} X_1 \ldots X_{i-1} Y p X_{i+1} \ldots X_n$.
    - The tape head moved from X_i to X_{i+1}.
    - The state changed from q to p.
    - X_i was replaced by Y.
  - Special case where $i = n$:
    - $X_1 \ldots X_{n+1} q X_n \underset{M}{\vdash} X_1 \ldots X_{n-1} Y p B$.
  - Special case where $i = 1$ and $Y = B$:
    - $q X_1 \ldots X_n \underset{M}{\vdash} p X_2 \ldots X_n$.

The B is included here just to make it clear what the tape head is pointing at.

The Language Defined by a TM

The language accepted by a Turing Machine M is defined as:

- $L(M) \equiv \{w \mid w \in \Sigma^* \text{ and } q_0 w \overset{*}{\underset{M}{\vdash}} \alpha_1 p \alpha_2 \text{ for some } p \in F, \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$.
- **Assumption**: If M accepts w, M halts.
- Notation used for moves (similar to a PDA):
  - $\overset{i}{\underset{M}{\vdash}}, \overset{*}{\underset{M}{\vdash}}, \vdash, \overset{i}{\vdash}, \overset{*}{\vdash}$.
    - The i indicates "exactly i moves"
    - The * indicates "any number of moves"
    - The (optional) M below the $\vdash$ indicates that TM M made the moves.

Recursively Enumerable Languages

- A language that is accepted by a Turing Machine is called *recursively enumerable* (RE).
- If a string w is in $L(M)$, then M eventually halts and accepts w.
- If $w \notin L(M)$, then one of two things can happen:
  - M halts without accepting w, or
  - M never halts.

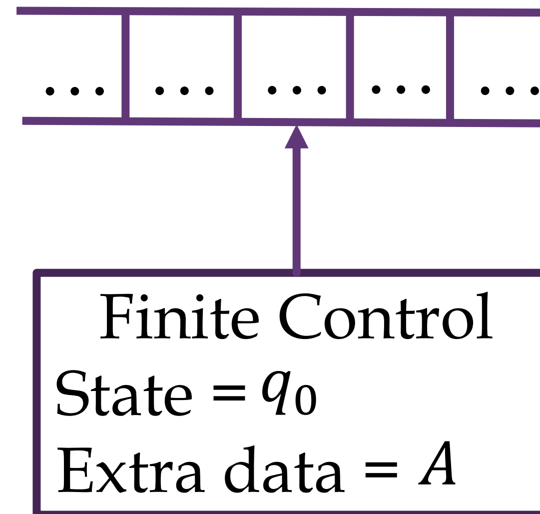
Recursive Languages

- A language that is accepted by a TM that halts on all inputs is called *recursive*.
- This is the accepted formal definition of "algorithm".
  - For a recursive language, we can always algorithmically determine membership—but we can't necessarily do this for RE languages.
- A problem that is solved by a TM that always halts is called *decidable*.
  - We will discuss decidable problems in much more detail later!

TM Construction Technique: Extra Storage

Storage in the finite control:

- You can keep track of a *finite* amount of extra data by incorporating it into the *state* of a TM.
- Example: Keep track of an additional symbol:
- If the "extra data" can be A or B, and the "state" can be q_0 or q_1, then the actual states of the TM can be $\{[q_0, A], [q_1, A], [q_0, B], [q_1, B]\}$.



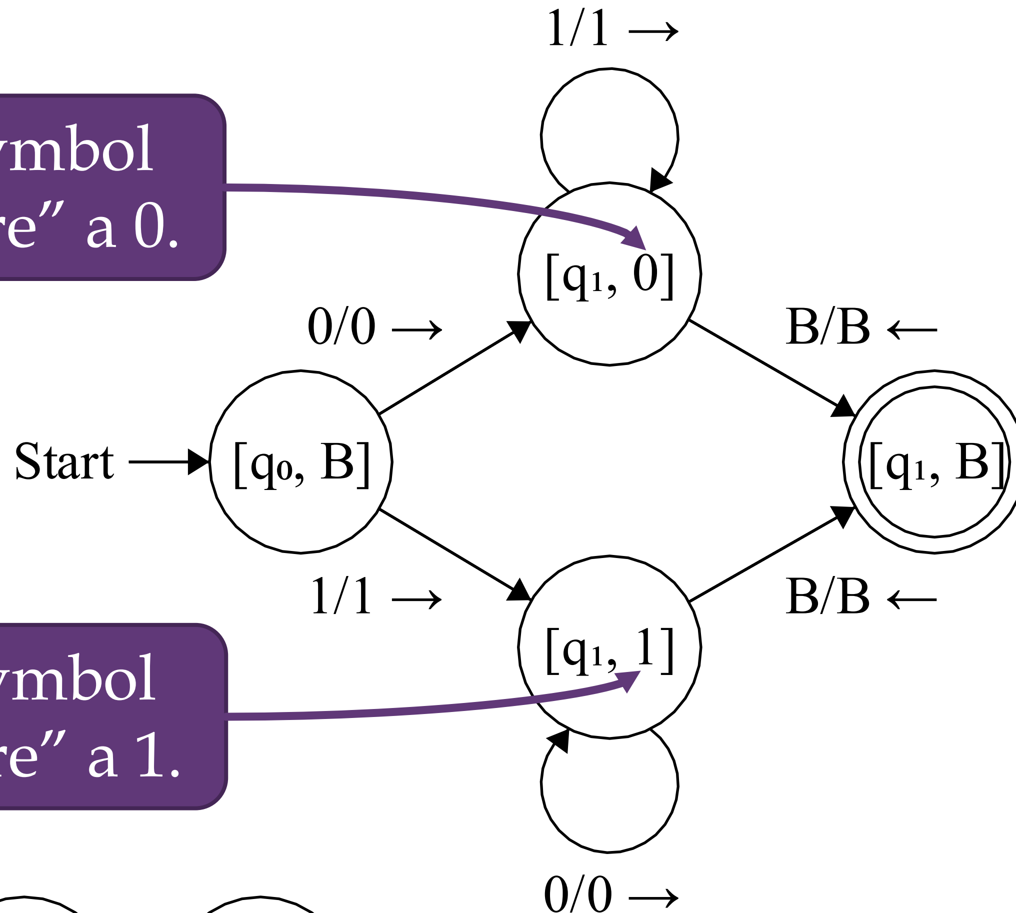
Storage in the Finite Control: Example

- Example: Scan the input and ensure the first symbol doesn't appear a second time.
- $M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], B, F)$
- $Q = \{q_0, q_1\} \times \{0, 1, B\}$
- $F = \{[q_1, B]\}$

- $\delta([q_0, B], 0) = ([q_1, 0], 0, R)$
- $\delta([q_1, 0], 1) = ([q_1, 0], 1, R)$
- $\delta([q_1, 0], B) = ([q_1, B], 0, L)$
- $\delta([q_0, B], 1) = ([q_1, 1], 1, R)$
- $\delta([q_1, 1], 0) = ([q_1, 1], 0, R)$
- $\delta([q_1, 1], B) = ([q_1, B], 0, L)$

- Note: This doesn't change the fact that the TM has a finite number of states!

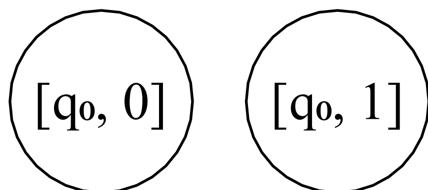
Storage in the Finite Control: Example



If the first symbol was a 0, "store" a 0.

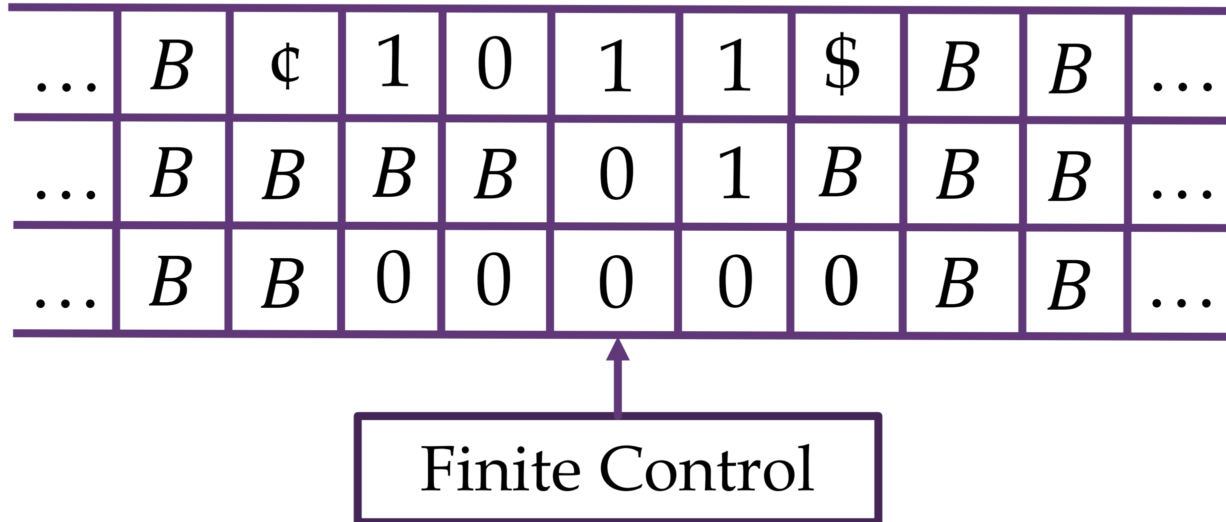
If the first symbol was a 1, "store" a 1.

$\delta\big((q_0, B), 0\big) = \big((q_1, 0), 0, R\big)$
$\delta\big((q_1, 0), 1\big) = \big((q_1, 0), 1, R\big)$
$\delta\big((q_1, 0), B\big) = \big((q_1, B), 0, L\big)$
$\delta\big((q_0, B), 1\big) = \big((q_1, 1), 1, R\big)$
$\delta\big((q_1, 1), 0\big) = \big((q_1, 1), 0, R\big)$
$\delta\big((q_1, 1), B\big) = \big((q_1, B), 0, L\big)$



This construction may result in unreachable states.

TM Construction Technique: Multiple Tracks



- In the above figure, the "real" input is between the ¢ and \$ on the first track; the other tracks are used for scratch storage.
- Tape symbols are tuples, i.e., $[¢, B, B], [1, B, 0], [\$, B, 0]$, etc.
- This doesn't change the fact that there are a *finite* number of possible tape symbols!

Multiple Tracks: Example

- We will use multiple tracks and storage in the finite control to write a TM that accepts the language $L = \{wcw \mid w \in (\mathbf{a} + \mathbf{b})^*\}$
- This TM will work by "checking off" corresponding symbols in each copy of w.

Any string w, followed by a "c", followed by a second copy of w.

Multiple Tracks: Example

Defining a TM accepting $L = \{wcw \mid w \in (\mathbf{a} + \mathbf{b})^*\}$, using two tracks and storing an extra symbol in the finite control.

- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- $Q = \{[q, d] \mid q \in \{q_1, \dots, q_9\}, d \in \{a, b, B\}\}$
- $\Sigma = \{[B, d] \mid d \in \{a, b, c\}\}$
- $\Gamma = \{[X, d] \mid X \in \{B, *\}, d \in \{a, b, c, B\}\}$
- $q_0 = [q_1, B]$
- $F = \{[q_9, B]\}$
- $B = [B, B]$

Multiple Tracks: Example

Defining a TM accepting $L = \{wcw \mid w \in (\mathbf{a} + \mathbf{b})^*\}$, using two tracks and storing an extra symbol in the finite control

- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- $Q = \{[q, d] \mid q \in \{q_1, \ldots, q_9\}, d \in \{a, b, B\}\}$
- $\Sigma = \{[B, d] \mid d \in \{a, b, c\}\}$
- $\Gamma = \{[X, d] \mid X \in \{B, *\}, d \in \{a, b, c, B\}\}$
- $q_0 = [q_1, B]$
- $F = \{[q_9, B]\}$
- $B = [B, B]$

This TM stores one additional symbol, d, in its finite control, along with the "state".

The additional symbol that this TM stores in the finite control can be a, b, or B.

This TM has 9 logical "states"

Multiple Tracks: Example

Defining a TM accepting $L = \{wcw \mid w \in (\mathbf{a} + \mathbf{b})^*\}$, using two tracks and storing an extra symbol in the finite control.

- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- $Q = \{[q, d] \mid q \in \{q_1, \ldots, q_9\}, d \in \{a, b, B\}\}$
- $\Sigma = \{[B, d] \mid d \in \{a, b, c\}\}$
- $\Gamma = \{[X, d] \mid X \in \{B, *\}, d \in \{a, b, c, B\}\}$
- $q_0 = [q_1, B]$
- $F = \{[q_9, B]\}$
- $B = [B, B]$

The actual input string we care about consists of as, bs, or cs. For example, the symbol $[B, a]$ corresponds to a symbol a in an input string.

The tape has two tracks, so the input "symbols" simply initialize corresponding cells in one of the tracks as blank.

Multiple Tracks: Example

Defining a TM accepting $L = \{wcw \mid w \in (\mathbf{a} + \mathbf{b})^*\}$, using two tracks and storing an extra symbol in the finite control.

- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- $Q = \{[q, d] \mid q \in \{q_1, \dots, q_9\}, d \in \{a, b, B\}\}$
- $\Sigma = \{[B, d] \mid d \in \{a, b, c\}\}$
- $\Gamma = \{[X, d] \mid X \in \{B, *\}, d \in \{a, b, c, B\}\}$
- $q_0 = [q_1, B]$
- $F = \{[q_9, B]\}$
- $B = [B, B]$

The second track contains either input symbols or B.

The first track in the tape contains a $*$ to indicate if the input symbol in the second track has been "checked off".

Multiple Tracks: Example

To avoid having to write several similar rules, let $d \in \{a, b\}$ and $e \in \{a, b\}$. $\delta =$

- $\delta([q_1, B], [B, d]) = ([q_2, d], [*, d], R)$

- $\delta([q_2, d], [B, e]) = ([q_2, d], [B, e], R)$
- $\delta([q_2, d], [B, c]) = ([q_3, d], [B, c], R)$

- $\delta([q_3, d], [*, e]) = ([q_3, d], [*, e], R)$
- $\delta([q_3, d], [B, d]) = ([q_4, B], [*, d], L)$

- $\delta([q_4, B], [*, d]) = ([q_4, B], [*, d], L)$
- $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$

- $\delta([q_5, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_5, B], [*, d]) = ([q_7, B], [*, d], R)$

- $\delta([q_6, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_6, B], [*, d]) = ([q_1, B], [*, d], R)$

- $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$

- $\delta([q_8, B], [*, d]) = ([q_8, B], [*, d], R)$
- $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], L)$

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

- $\delta([q_1, B], [B, d]) = ([q_2, d], [*, d], R)$

We're looking at input symbol d in the second track.

Store d into the finite-control storage.

- Start off by storing the symbol you're looking at in the "finite control" storage, checking off the symbol, and starting to move right.
  - The goal will be to make sure the first "unchecked" symbol in the right-hand string matches the symbol stored in the finite control.

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

- $\delta([q_1, B], [B, d]) = ([q_2, d], [*, d], R)$

Previously, this symbol was not checked off…

So we'll check it off.

- Start off by storing the symbol you're looking at in the "finite control" storage, checking off the symbol, and starting to move right.
  - The goal will be to make sure the first "unchecked" symbol in the right-hand string matches the symbol stored in the finite control.

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

- $\delta([q_1, B], [B, d]) = ([q_2, d], [*, d], R)$

- $\delta([q_2, d], [B, e]) = ([q_2, d], [B, e], R)$
- $\delta([q_2, d], [B, c]) = ([q_3, d], [B, c], R)$

- $\delta([q_3, d], [*, e]) = ([q_3, d], [*, e], R)$
- $\delta([q_3, d], [B, d]) = ([q_4, B], [*, d], L)$

- $\delta([q_4, B], [*, d]) = ([q_4, B], [*, d], L)$
- $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$

The point of state q_2 is just to move right until we reach the right-hand string.

We know we've reached the right-hand string after passing the input symbol c.

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

- $\delta([q_1, B], [B, d]) = ([q_2, d], [*, d], R)$

- $\delta([q_2, d], [B, e]) = ([q_2, d], [B, e], R)$
- $\delta([q_2, d], [B, c]) = ([q_3, d], [B, c], R)$

- $\delta([q_3, d], [*, e]) = ([q_3, d], [*, e], R)$
- $\delta([q_3, d], [B, d]) = ([q_4, B], [*, d], L)$

- $\delta([q_4, B], [*, d]) = ([q_4, B], [*, d], L)$
- $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$

The point of state q_3 is to move right until we reach the first unchecked symbol in the right-hand string.

Once this symbol is reached, check it off. (The TM will die here if the symbol doesn't match the one in the finite-control storage.)

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

- $\delta([q_1, B], [B, d]) = ([q_2, d], [*, d], R)$

- $\delta([q_2, d], [B, e]) = ([q_2, d], [B, e], R)$
- $\delta([q_2, d], [B, c]) = ([q_3, d], [B, c], R)$

- $\delta([q_3, d], [*, e]) = ([q_3, d], [*, e], R)$
- $\delta([q_3, d], [B, d]) = ([q_4, B], [*, d], L)$

- $\delta([q_4, B], [*, d]) = ([q_4, B], [*, d], L)$
- $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$

The point of state q_4 is to move left until we reach the left-hand string again.

Enter state q_5 after passing the symbol c.

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

In q_5, we are looking at the last symbol in the left-hand string. If it's *not* checked off, go to state q_6.

If the last symbol in the left-hand string was checked off already, go to q_7 instead.

- $\delta([q_5, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_5, B], [*, d]) = ([q_7, B], [*, d], R)$

- $\delta([q_6, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_6, B], [*, d]) = ([q_1, B], [*, d], R)$

- $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$

- $\delta([q_8, B], [*, d]) = ([q_8, B], [*, d], R)$
- $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], L)$

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

In q_6, keep moving left until we find the leftmost checked-off symbol in the left-hand string.

Move right from this symbol to get to the next unchecked symbol in the left-hand string, and repeat the process from state q_1.

- $\delta([q_5, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_5, B], [*, d]) = ([q_7, B], [*, d], R)$

- $\delta([q_6, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_6, B], [*, d]) = ([q_1, B], [*, d], R)$

- $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$

- $\delta([q_8, B], [*, d]) = ([q_8, B], [*, d], R)$
- $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], L)$

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

In state q_7, everything in the left-hand string has been checked off, so we're getting ready to make sure everything in the right-hand string is checked off, too.
State q_7's main purpose is to move right past the c.

- $\delta([q_5, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_5, B], [*, d]) = ([q_7, B], [*, d], R)$

- $\delta([q_6, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_6, B], [*, d]) = ([q_1, B], [*, d], R)$

- $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$

- $\delta([q_8, B], [*, d]) = ([q_8, B], [*, d], R)$
- $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], L)$

Multiple Tracks: Example

(Let $d \in \{a, b\}$ and $e \in \{a, b\}$.)

In state q_8, move right while checking that everything in the right-hand string is checked off (if any input symbol is not checked off, execution will die).

If we made it past all the input symbols without dying, go to state q_9 (which is accepting).

- $\delta([q_5, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_5, B], [*, d]) = ([q_7, B], [*, d], R)$

- $\delta([q_6, B], [B, d]) = ([q_6, B], [B, d], L)$
- $\delta([q_6, B], [*, d]) = ([q_1, B], [*, d], R)$

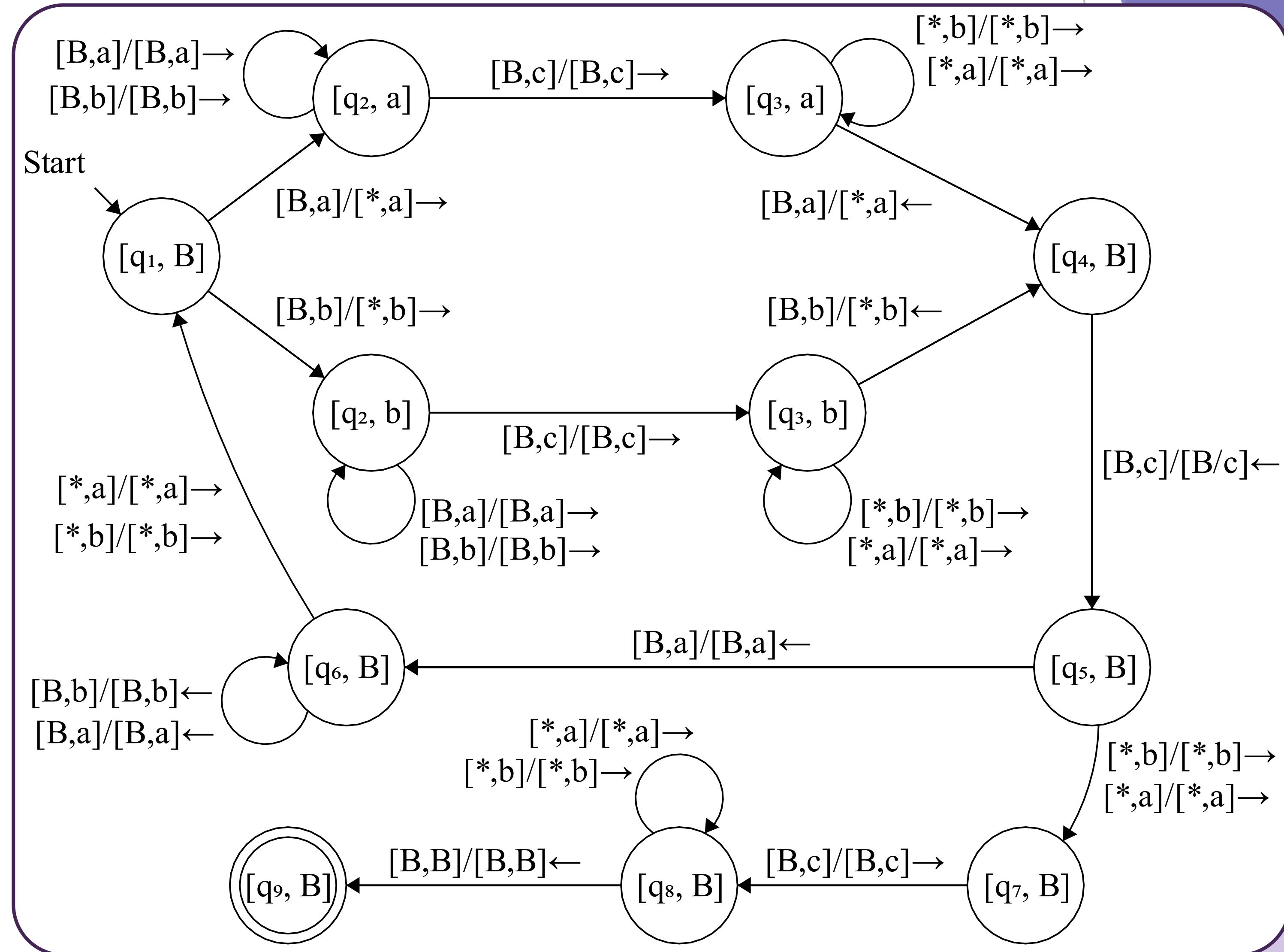
- $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$

- $\delta([q_8, B], [*, d]) = ([q_8, B], [*, d], R)$
- $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], L)$

Side note: we already accept here, so it doesn't matter whether we move L or R now.

Multiple Tracks: Example

Here's the TM as a transition diagram:



TM Technique: Shifting Symbols Over

- You may need to create extra space in the middle of a string of symbols in a TM.
- This can be accomplished by holding a small buffer of symbols in the finite control.
- **Example**: Shifting a string of nonblank symbols over by two spaces.
  - States will be of the form $[q, A_1, A_2]$, where q is q_1 or q_2 and A_1 and A_2 are in Γ.

TM Technique: Shifting Symbols Over

Here are the relevant moves for shifting symbols over by two spaces:

- $\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R)$, $A_1 \in \Gamma - \{B, X\}$
- $\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R)$, A_1 and $A_2 \in \Gamma - \{B, X\}$

- $\delta([q_1, A_1, A_2], A_3) = ([q_1, A_2, A_3], A_1, R)$, $A_1, A_2,$ and $A_3 \in \Gamma - \{B, X\}$

- $\delta([q_1, A_1, A_2], B) = ([q_1, A_2, B], A_1, R)$, A_1 and $A_2 \in \Gamma - \{B, X\}$
- $\delta([q_1, A_1, B], B) = ([q_2, B, B], A_1, L)$, $A_1 \in \Gamma - \{B, X\}$

- $\delta([q_2, B, B], A) = ([q_2, B, B], A, L)$, $A \in \Gamma - \{B, X\}$

TM Technique: Shifting Symbols Over

Here are the relevant moves for shifting symbols over by two spaces:

- $\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R)$, $A_1 \in \Gamma - \{B, X\}$
- $\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R)$, A_1 and $A_2 \in \Gamma - \{B, X\}$

Start by "shifting" the tape symbols A_1 and A_2 into the finite-control buffer.

A_1 and A_2 can't be B. For simplicity, also assume that they can't be X.

We will write the symbol X into the "extra space" we are adding.

TM Technique: Shifting Symbols Over

Here are the relevant moves for shifting symbols over by two spaces:

- $\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R)$, $A_1 \in \Gamma - \{B, X\}$
- $\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R)$, A_1 and $A_2 \in \Gamma - \{B, X\}$

- $\delta([q_1, A_1, A_2], A_3) = ([q_1, A_2, A_3], A_1, R)$, $A_1, A_2,$ and $A_3 \in \Gamma - \{B, X\}$

Once the buffer is full, shift subsequent new symbols into the buffer while replacing them with the oldest symbol in the buffer.

TM Technique: Shifting Symbols Over

Here are the relevant moves for shifting symbols over by two spaces:

- $\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R)$, $A_1 \in \Gamma - \{B, X\}$
- $\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R)$, A_1 and $A_2 \in \Gamma - \{B, X\}$

- $\delta([q_1, A_1, A_2], A_3) = ([q_1, A_2, A_3], A_1, R)$, $A_1, A_2,$ and $A_3 \in \Gamma - \{B, X\}$

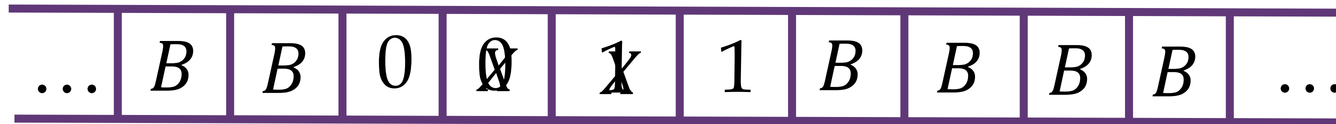
- $\delta([q_1, A_1, A_2], B) = ([q_1, A_2, B], A_1, R)$, A_1 and $A_2 \in \Gamma - \{B, X\}$
- $\delta([q_1, A_1, B], B) = ([q_2, B, B], A_1, L)$, $A_1 \in \Gamma - \{B, X\}$

- $\delta([q_2, B, B], A) = ([q_2, B, B], A, L)$, $A \in \Gamma - \{B, X\}$

After shifting everything, state q_2 causes the TM to move left to the newly added space.

Finally, after you encounter blank symbols, shift the remaining symbols out of the buffer.

TM Technique: Shifting Symbols Over



Finite Control

State = q_2

Buffer = B, B

$A_1, A_2, A_3, A \notin \{X, B\}$:

$\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R)$
$\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R)$
$\delta([q_1, A_1, A_2], A_3) = ([q_1, A_2, A_3], A_1, R)$
$\delta([q_1, A_1, A_2], B) = ([q_1, A_2, B], A_1, R)$
$\delta([q_1, A_1, B], B) = ([q_2, B, B], A_1, L)$
$\delta([q_2, B, B], A) = ([q_2, B, B], A, L)$

Done!

TM Technique: Subroutines

- TMs can simulate subroutines, even including parameter-passing and recursion.
- **Example**: Multiplication using a "copy" subroutine
  - Given $0^m 1 0^n 1$ as input, produce 0^{mn} as output.
  - This can be done by "copying" n 0s m times.

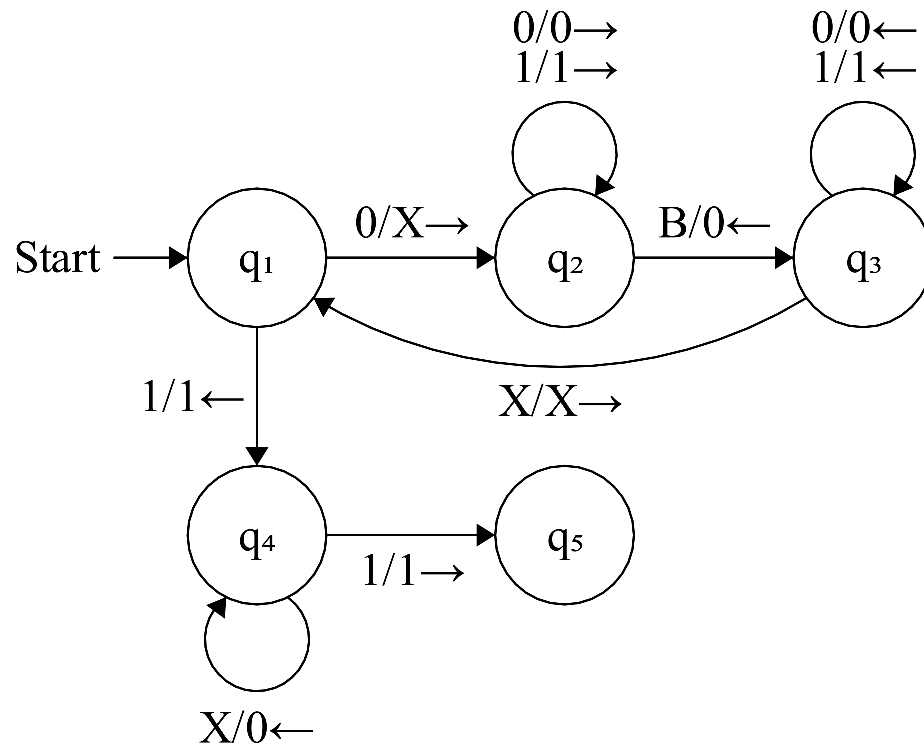
TM Technique: Subroutines

Here's how the overall TM will work:

1. While processing, the tape will contain a string of the form $0^i 1 0^n 0^{kn}$ for some k.
2. In one "iteration", we will change a 0 in the first group of i 0s to B and append n 0s to the last group of 0s.
   - This will require *copy*ing the group of n 0s to the end of the string.
3. Eventually, there will be no more 0s at the start of the string, and the TM can delete the $10^n 1$, leaving only 0^{nm} on the tape.

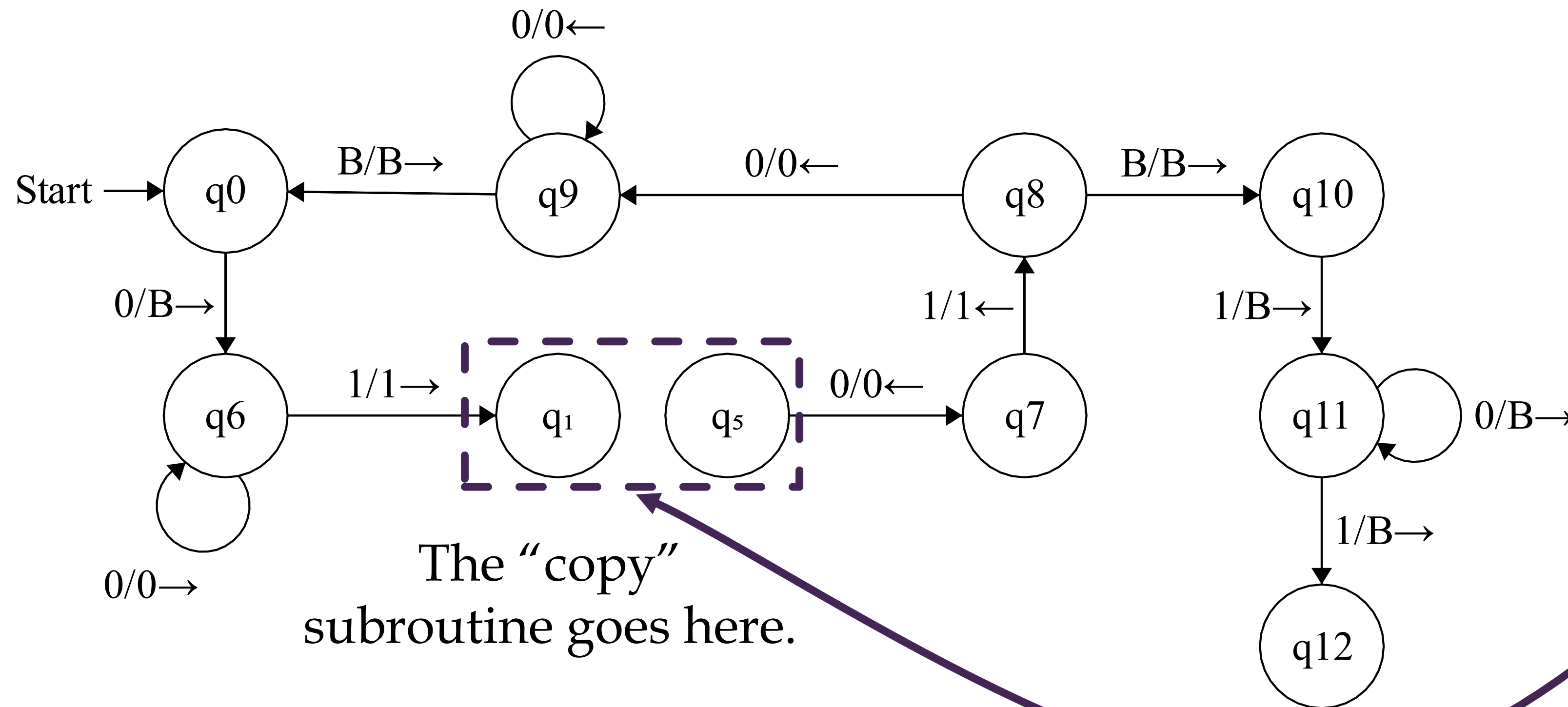
TM Technique: Subroutines

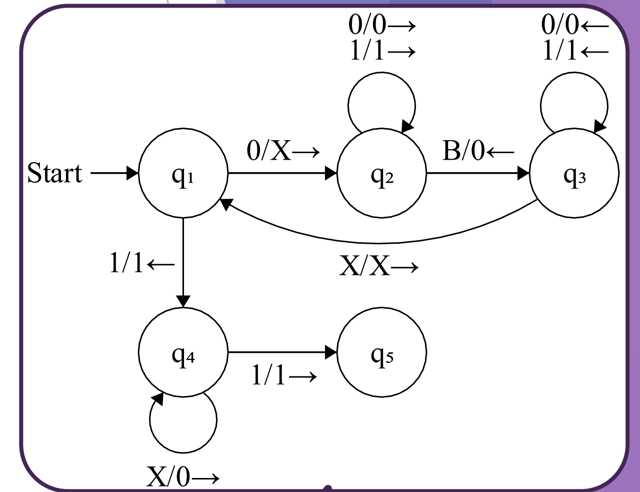
The key component of our "multiplication" TM is this subroutine, *copy*, defined in the following diagram:



TM Technique: Subroutines

We can "plug in" the copy subroutine where it is needed in a larger TM:





TM Technique: Subroutines

Example move sequence for the *copy* routine for an input where $m = 3, n = 2,$ and $i = 4$:

$0^3 1 q_1 0^2 1 0^4$

$\vdash 0^3 1 X q_2 010000$

$\vdash 0^3 1 X 0 q_2 10000$

$\vdash 0^3 1201 q_2 0000$

$\ldots$

$\vdash 0^3 1 X010000 q_2$

$\vdash 0^3 1 X01000 q_3 00$

$\vdash 0^3 1 X0100 q_3 000$

$\ldots$

$\vdash 0^3 1 q_3 X0100000$

$\vdash 0^3 1 X q_1 0100000$

$\ldots$

$\vdash 0^3 1 XX100000 q_2$

$\vdash 0^3 1 XX10000 q_3 00$

$\ldots$

$\vdash 0^3 1 X q_3 X1000000$

$\vdash 0^3 1 XX q_1 1000000$

$\vdash 0^3 1 X q_4 X1000000$

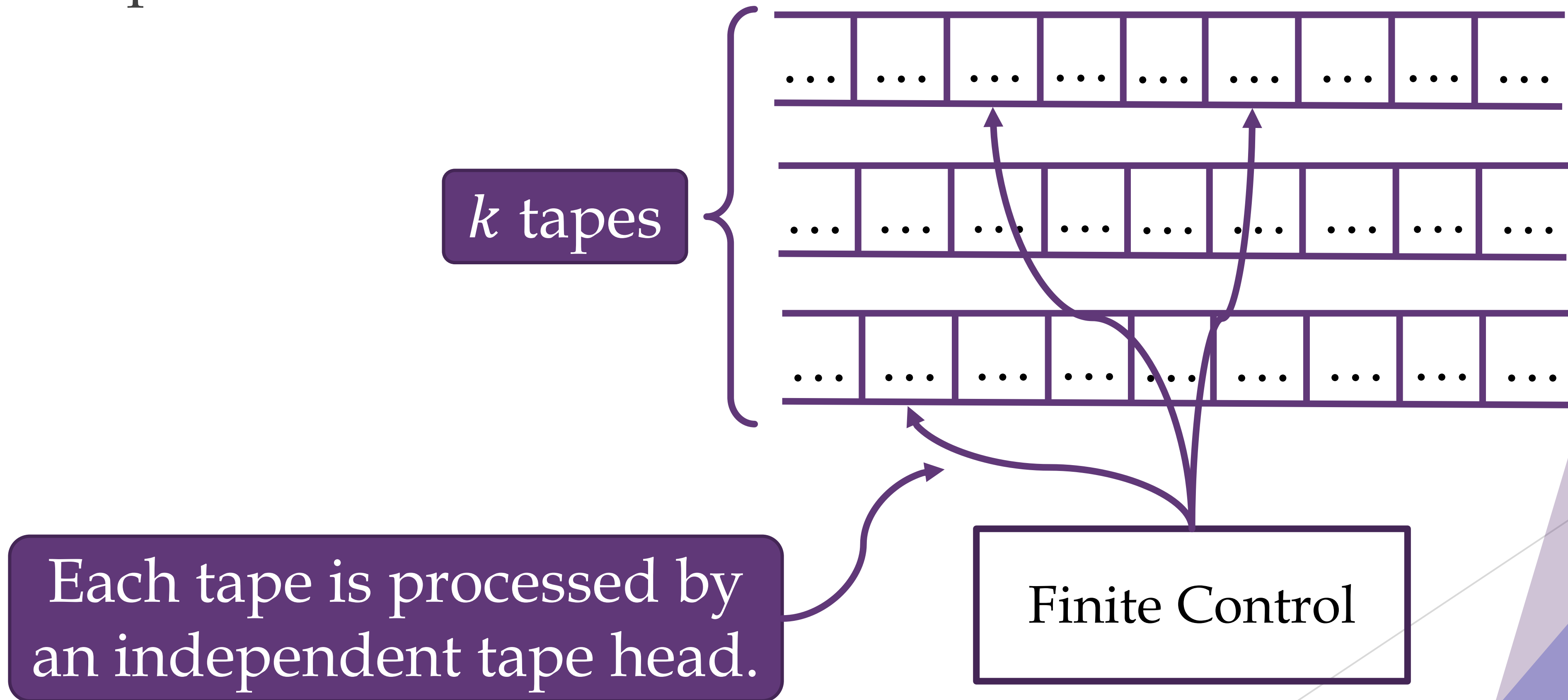
$\vdash 0^3 1 q_4 X01000000$

$\vdash 0^3 q_4 1001000000$

$\vdash 0^3 1 q_5 001000000$

TM Extensions: Multiple Tapes

A multi-tape TM can be conceptualized like this:



TM Extensions: Multiple Tapes

- In a multi-tape TM, each move depends on the state of the finite control and *k* "current" tape symbols.
- During a move, a multi-tape TM can:
  1. Change state.
  2. Print a new symbol on each cell scanned by a tape head.
  3. Move each tape head left, right, or stationary.
- We assume the input is initially on the first tape.

Single-Tape vs. Multi-Tape TMs

Theorem 8.9: If L is accepted by a multi-tape TM, then it is accepted by a single-tape TM.

Proof:

- Let M be a multi-tape TM with k tapes.
- We can construct a single-tape TM, M' with $2k$ *tracks* that accepts $L(M)$.

Single-Tape vs. Multi-Tape TMs

M''s (single) multi-track tape looks like this:

Tape 1 of *M*	⋯	X_{11}	X_{12}	⋯	X_{1i}	⋯		
Head for tape 1	⋯	B	B	⋯	#	⋯		
Tape 2 of *M*	⋯	X_{21}	X_{22}				X_{2j}	⋯
Head for tape 2	⋯	B	B				#	⋯
				⋮				
Tape *k* of *M*	⋯	X_{k1}	X_{k2}	⋯	X_{kn}	⋯		
Head for tape *k*	⋯	B	B	⋯	#	⋯		

We use the # in these tracks to indicate where the tape heads are.

The **state** of M' includes:

- The state of M.
- The number of head markers to the right of the tape head.
- The tape symbols at each head marker.
- The scan direction.

Single-Tape vs. Multi-Tape TMs

Tape 1 of *M*	…	X_{11}	X_{12}	…	X_{1i}	…		
Head for tape 1	…	B	B	…	#	…		
Tape 2 of *M*	…	X_{21}	X_{22}				X_{2j}	…
Head for tape 2	…	B	B				#	…
				⋮				
Tape *k* of *M*	…	X_{k1}	X_{k2}	…	X_{kn}	…		
Head for tape *k*	…	B	B	…	#	…		

Keeping the number of head markers to the left of M''s head makes it possible to know when all of the head markers have been visited.

For M' to simulate a move of M:

1. Starting at the leftmost head marker, move right and store the symbol at each head marker.
2. Determine the move M would make.
3. Move left and update the tape symbols and head markers.
4. Change the state as indicated by the move that M would make.

Single-Tape vs. Multi-Tape TMs

Tape 1 of *M*	⋯	X_{11}	X_{12}	⋯	X_{1i}	⋯			
Head for tape 1	⋯	B	B	⋯	#	⋯			
Tape 2 of *M*	⋯	X_{21}	X_{22}				X_{2j}	⋯	
Head for tape 2	⋯	B	B				#	⋯	
				⋮					
Tape *k* of *M*	⋯	X_{k1}	X_{k2}	⋯	X_{kn}	⋯			
Head for tape *k*	⋯	B	B	⋯	#	⋯			

Proof (abridged):
After n moves in M, the head markers in M' can be at most $2n$ cells apart. This means that simulating a *single* move in M requires $O(n)$ moves in M'.

- **Theorem 8.10**: M' takes $O(n^2)$ moves to simulate n moves of M.
- Since a multi-tape TM is just as powerful as a single-tape TM, we can use whichever is more convenient.

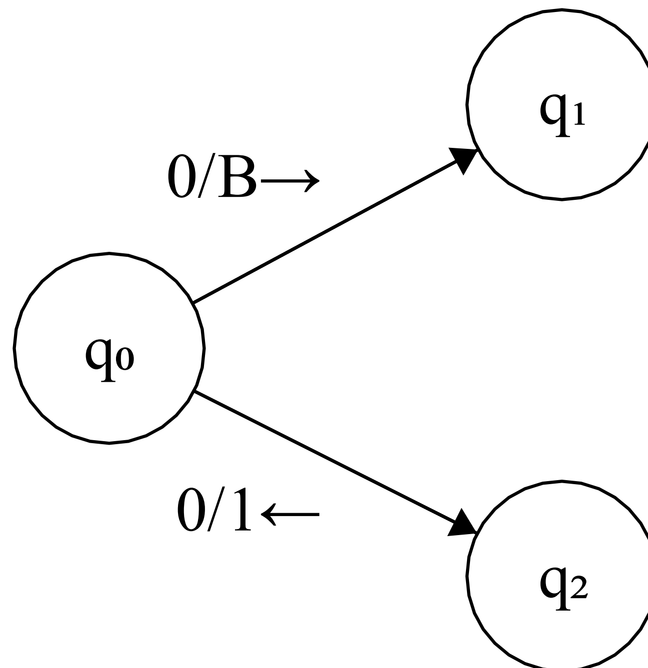
TM Extensions: Nondeterministic TMs

- The original definition we discussed was for a *deterministic* TM (DTM).
- A nondeterministic TM (NDTM) can “choose” between multiple possible moves for any combination of state and tape symbol(s).
- Move function for a nondeterministic TM with k tapes:
  - δ maps $Q \times \Gamma^k$ to *subsets* of $Q \times (\Gamma \times \{L, R, S\})^k$.

TM Extensions: Nondeterministic TMs

Example move function for a nondeterministic single-tape TM, where $\Gamma = \{0, 1, B\}$:

- $\delta(q_0, 0) = \{(q_1, B, R), (q_2, 1, L)\}$



Nondeterministic TMs

Note: The book reasons about a multi-tape NDTM here, instead.

Theorem 8.11: If L is accepted by a single-tape NDTM M_1, then L is accepted by some DTM M_2.

- Let d be the maximum number of nondeterministic choices M_1 can make at any given move.
- M_2 will systematically try all nondeterministic possibilities.
- M_2 will have three tapes.

Nondeterministic TMs

How M_2 works:

- **Tape 1** holds the input.
- **Tape 2** contains a sequence of digits from 1 to d, generated systematically. Each sequence dictates a sequence of choices. e.g.,
  - $[1], [2], [3], \ldots, [d]$,
    $[1,1], [1, 2], [2, 1], \ldots, [d, d]$,
    $[1,1,1], [1,1,2], [1, 2, 1], \ldots, [d, d, d]$,
    …
- **Tape 3** contains a scratch copy of the input.

Nondeterministic TMs

How M_2 works (continued):

1. Generate the next sequence on tape 2
2. Copy tape 1 (input) to tape 3 (scratch copy)
3. Simulate M_1 on tape 3, making choices according to the sequence on tape 2.
4. If M_1 accepts, then accept;
5. Otherwise, go to step 1.

Nondeterminism and Time Complexity

- **Definition**: A NDTM M is of time complexity $T(n)$ if for every accepted string of length n, some sequence of at most $T(n)$ moves leading to an accepting state *exists*.

Theorem: If L is accepted by a single-tape NDTM M_1 with time complexity $T(n)$, then L is accepted by some DTM M_2 with time complexity $O\left(c^{T(n)}\right)$, for some constant c.

- **Proof**: If M_1 in the previous construction can accept in at most $T(n)$ moves, then M_2 can accept in at most $O\left(T(n)(d+1)^{T(n)}\right)$ moves.

Restricted TMs: Semi-infinite Tape

- In a TM with a *semi-infinite tape*, the tape is infinite only to the right of the starting position.
  - Assume that trying to move left from the leftmost cell causes execution to die.

Theorem 8.12: (reworded) L is recognized by a TM with a two-way infinite tape if and only if it is recognized by a TM with a semi-infinite tape.

Restricted TMs: Semi-infinite Tape

Theorem 8.12: *L* is recognized by a TM with a two-way infinite tape if and only if it is recognized by a TM with a semi-infinite tape.

Proof of Theorem 8.12:

"If":

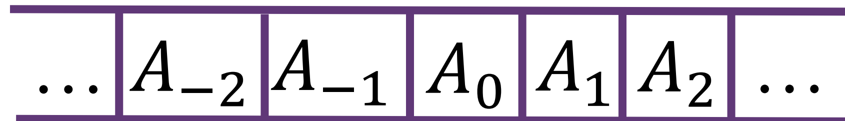
- It's pretty straightforward to simulate a semi-infinite tape TM using one with a two-way infinite tape.
  - Mark the tape cell to the left of the starting position with a special symbol, and halt without accepting if that symbol is ever encountered.

Restricted TMs: Semi-infinite Tape

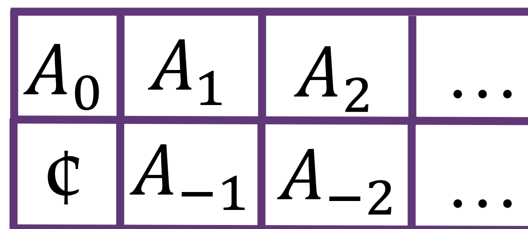
Proof (continued), **"Only if"**:

- Let $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_2, B, F_2)$ be a two-way TM.
- Construct a one-way TM M_1 to simulate M_2.
- The tape of M_1 will have two tracks.

Tape for M_2:



Tape for M_1:



The ¢ helps us identify the leftmost cell.

Restricted TMs: Semi-infinite Tape

Tape for M_2:

$\dots$	A_{-2}	A_{-1}	A_0	A_1	A_2	$\dots$

Tape for M_1:

A_0	A_1	A_2	$\dots$
¢	A_{-1}	A_{-2}	$\dots$

- Formally, $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, B, F_1)$.
  - Q_1 contains all $[q, U]$ and $[q, D]$ for all $q \in Q_2$.
    - We're storing U or D in the finite control to remember whether M_1 is on the upper "U", or lower "D" track.
    - Q_1 also contains q_1 by itself.
  - Γ_1 contains all $[X, Y]$, where $X \in \Gamma_2$ and $Y \in \Gamma_2 \cup \{¢\}$.

Restricted TMs: Semi-infinite Tape

Tape for M_2:

…	A_{-2}	A_{-1}	A_0	A_1	A_2	…

Tape for M_1:

A_0	A_1	A_2	…
¢	A_{-1}	A_{-2}	…

- Formally, $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, B, F_1)$.
  - Σ_1 contains all $[a, B]$, where $a \in \Sigma_2$.
  - F_1 is $\{[q, U], [q, D] \mid q \in F_2\}$.
  - B is $[B, B]$.

Restricted TMs: Semi-infinite Tape

Tape for M_2:

…	A_{-2}	A_{-1}	A_0	A_1	A_2	…

Tape for M_1:

A_0	A_1	A_2	…
¢	A_{-1}	A_{-2}	…

The definition of δ_1:

- $\delta_1(q_1, [a, B]) = ([q, U], [X, ¢], R)$ if $\delta_2(q_2, a) = (q, X, R)$
  - This is the case when the first move of M_2 is *right*.
- $\delta_1(q_1, [a, B]) = ([q, D], [X, ¢], R)$ if $\delta_2(q_2, a) = (q, X, L)$
  - This is the case when the first move of M_2 is *left*.

Restricted TMs: Semi-infinite Tape

Tape for M_2:

…	A_{-2}	A_{-1}	A_0	A_1	A_2	…

Tape for M_1:

A_0	A_1	A_2	…
¢	A_{-1}	A_{-2}	…

The definition of δ_1 (continued):

For all $[X, Y] \in \Gamma_1$, with $Y \neq$ ¢, and $A = L$ or R:

- $\delta_1([q, U], [X, Y]) = ([p, U], [Z, Y], A)$ if $\delta_2(q, X) = (p, Z, A)$.
  - This simulates M_2 on the *upper* track.
- $\delta_1([q, D], [X, Y]) = ([p, D], [X, Z], A)$ if $\delta_2(q, Y) = (p, Z, \bar{A})$
  - This simulates M_2 on the *lower* track.

If we're in the lower track in M_1 we need to move in the *opposite* direction from how M_2 would move.

Restricted TMs: Semi-infinite Tape

Tape for M_2:

…	A_{-2}	A_{-1}	A_0	A_1	A_2	…

Tape for M_1:

A_0	A_1	A_2	…
¢	A_{-1}	A_{-2}	…

The definition of δ_1 (continued):

- $\delta_1([q,U],[X,¢]) = \delta_1([q,D],[X,¢]) = ([p,C],[Y,¢],R)$, if $\delta_2(q,X) = (p,Y,A)$, where

$$C = \begin{cases} U & \text{if } A = R \\ D & \text{if } A = L \end{cases}$$

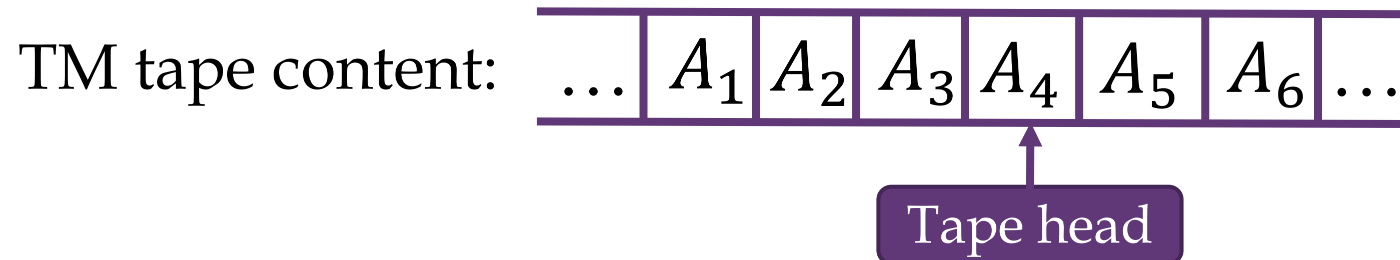
- This allows M_1 to switch tracks if it is currently at the leftmost cell.

Restricted TMs: Two-Stack Machine

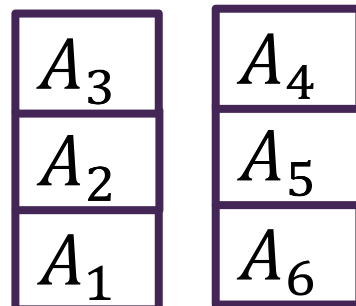
- Imagine a PDA that can make decisions and modify two stacks rather than one. This is called a *two-stack machine*.

Theorem 8.13: A two-stack machine can simulate a TM.

Proof sketch:



Store tape contents in two stacks like this:



See Section 8.5.2 of the book for a much more detailed proof.

Restricted TMs: Counter Machines

- A *counter machine* can be thought of as a multi-stack PDA, with only two stack symbols and some restrictions:
  - Z_0, serving as the "bottom of stack marker"
  - X
  - Initially, only Z_0 is on each stack.
  - Only X's can be pushed or popped from the stack.
- Essentially, each stack is a *counter*. The PDA can only do three things with each stack:
  - Increase the "counter" by pushing more X's.
  - Decrement the "counter" by popping a single X.
  - Check if the "counter" is 0 by seeing if Z_0 is on top of the stack.

Restricted TMs: Three-Counter Machines

Theorem 8.14: Every RE language is accepted by a three-counter machine.

Proof:

We will show that *one stack* can be simulated by *two counters*, the second of which is a "scratch" counter used for calculations.

When simulating *two* stacks, we can then use the same scratch counter, so three counters in total are sufficient.

Restricted TMs: Three-Counter Machines

We now need to show how to simulate one stack using two counters.

- Suppose the stack alphabet contains $r - 1$ symbols.
- We can assume these symbols are denoted $1, \ldots, r - 1$.
- Store the stack contents $X_1 X_2 \ldots X_n$ (where X_1 is the top of the stack) as a base-r number: $X_n r^{n-1} + X_{n-1} r^{n-2} + \ldots + X_2 r + X_1$.
- **Example**: Assume $r = 10$ and the stack contains the symbols 9, 2, 5, and 3 (9 is on top). The counter value associated with this stack is simply 3259.

Restricted TMs: Three-Counter Machines

We now need to show how to simulate stack operations using a counter.

- **Pop** the stack: Replace the counter value i by i/r. The remainder is the old top-of-stack symbol, which can be stored in the finite control.
  - **Example**: popping 3259. $\frac{3259}{r} = \frac{3259}{10} = 325$, with remainder 9.

Restricted TMs: Three-Counter Machines

- **Push** X onto the stack: Replace the counter value i by $ir + X$.
  - **Example**: pushing a 6 onto 3259. $(3259r) + 6 = (3259 * 10) + 6 = 32596$.
- **Replace** X **by** Y on top of the stack: If $Y > X$, then increment i by $Y - X$. If $Y < X$, then decrement it by $X - Y$.

Note that these operations (including pop) only involve constant values (based on the constant r).

Restricted TMs: Three-Counter Machines

But how do we multiply or divide when we can only check if a counter is 0 or nonzero?

To multiply a counter by *r* using a scratch counter:

```
scratch := 0;
while count ≠ 0:
      count := count – 1;
      scratch := scratch + r;
/* If count was initially i, then scratch is now ir.
So now, copy scratch back to count. */
while scratch ≠ 0:
      count := count + 1;
      scratch := scratch – 1;
```

This algorithm only adds or subtracts constant values, and only needs to check if counters are 0.

Restricted TMs: Three-Counter Machines

To divide a counter by r using a scratch counter and a bounded value k ($k \leq r$, and r is constant, so we can keep track of k using a finite number of states):

```
scratch := 0;
while count ≠ 0:
    k := 0;
    while k ≠ r and count ≠ 0:
        count := count – 1;
        k := k + 1;
    if k = r:
        scratch := scratch + 1;
/* If count was initially i, then scratch is now i/r and k is the remainder.*/
while scratch ≠ 0:
    count := count + 1;
    scratch := scratch – 1;
```

Restricted TMs: Two-Counter Machines

We have now shown that we can simulate a single stack using two counters, and two stacks using three counters, but we can go farther!

Theorem 8.15: Every RE language is accepted by a two-counter machine.

Proof: We will show that we can simulate *three counters using two counters*.

If i, j, and k are the three counts we want to track, we can store them in a single integer $n = 2^i 3^j 5^k$.

Restricted TMs: Two-Counter Machines

(From previous slide) If i, j, and k are the three counts we want to track, we can store them in a single integer $n = 2^i 3^j 5^k$.

- To increment i: Replace n by $2n$.
- To decrement i: Replace n by $n/2$.
- To test if i is 0: Check if n is not divisible by 2.
- The other counters are similar (e.g. increment j by replacing n with $3n$ and increment k by setting $n = 5n$, etc.).

We have already seen how to multiply and divide by constant values using a second "scratch" counter.

Note that this construction works because 2, 3, and 5 are *relatively prime*.

Restricted TMs: Summary

- We have shown that the following machines are equally powerful *in terms of languages they can identify*:
  - Nondeterministic TMs
  - Multi-tape TMs
  - Multi-track TMs
  - Single-tape TMs
  - Semi-infinite tape TMs
  - Two-stack machines
  - Three-counter machines
  - Two-counter machines

TMs vs. "Real" Computers

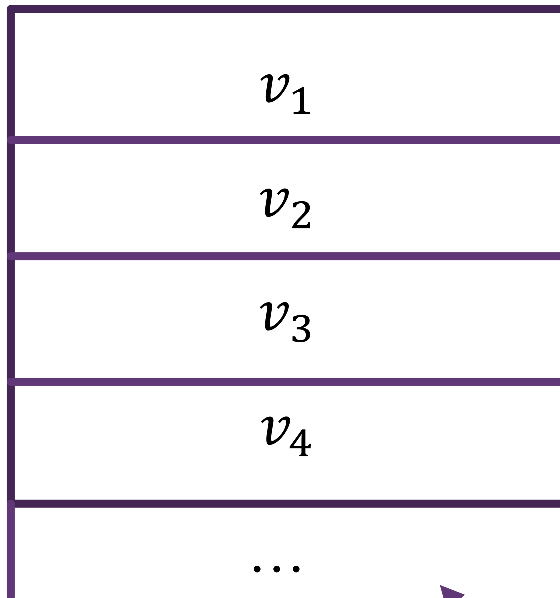
We want to show that anything a TM can do is possible using a computer, and vice versa.

- Designing a computer that simulates a TM is easy: just store the transition function in a table, which tells a program what to do next.
- One "gotcha": *a TM has infinite memory*.
  - The book says we can just continually add new disks to a TM when old ones are full.
  - What if we have TM that uses more tape cells than atoms in the universe? (Clearly the book's approach isn't a *true* Turing Machine.)
  - My take: we can't simulate infinite memory, but we will also never have a practical application that requires it.

Simulating a Computer using a TM

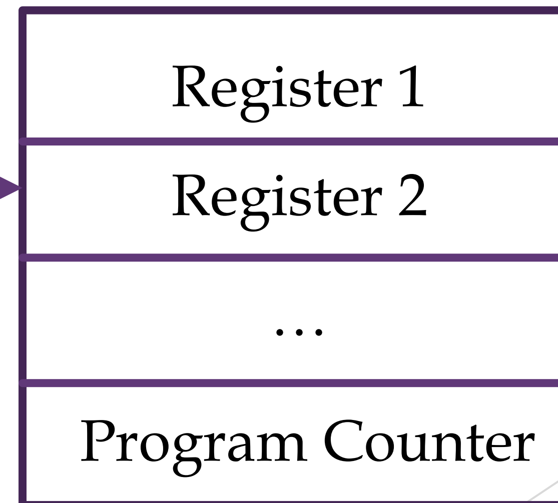
We will consider a simplified model of a computer called a *random access machine* (RAM):

A potentially infinite number of values stored in memory:



Both memory and registers can store any integer; even indefinitely large ones.

A finite number of registers:



Program "code" and data is stored in memory.

Random Access Machines

- In a RAM, a “program” is a list of instructions.
  - Instructions are like what you’d find on a “real” computer, and consist of an opcode and an address.
  - Instructions are encoded as a single integer (e.g. with some bits representing the opcode and the remaining bits representing the argument).
- With this model, you can have basic instructions like LOAD, STORE, ADD, etc., like an assembly language.

Simulating a Random Access Machine

Theorem: A TM can simulate a RAM, provided that the TM can simulate each individual RAM instruction.

- For example, you can't have a RAM instruction that solves the halting problem.

Proof sketch:

We will use multiple tapes to store the RAM's memory and register values in a TM.

Simulating a Random Access Machine

- One tape in our TM holds memory values that have been written.
  - The format of this tape is #0*v_0#1*v_1#10*v_2#...
  - The # and * are simply delimiters around the "address" of each value. Assume that each v_i is stored in binary.
- Use one additional tape for each register.

Simulating a Random Access Machine

Overview of the TM’s behavior for simulating the RAM:

1. Read the current PC value i from the tape holding the “program counter” register.
2. Scan the memory tape for the “address” #i*. Halt if it isn’t found.
3. Decode the instruction opcode and address from the memory value at location i.
4. Perform the operation indicated by the opcode and address.
5. Repeat from step 1.

Simulating a Random Access Machine

Possible behavior for a couple common "instructions":

- If the instruction indicates to "ADD" a value at address *j* to a register:
    1. Scan memory for #*j**, and add the value found afterwards to a register.
    2. Add 1 to the program counter register to advance to the next instruction.
- If the instruction indicates to "GOTO" address *j*:
    1. Copy the address *j* to the program counter register.

(Other instructions can be simulated in a similar manner)

Running Time of the RAM Simulation

The previous simulation will run in *polynomial time* provided that a couple restrictions hold:

- First, the RAM can't have an instruction that takes exponential time to compute.
  - For example, you can't have a "solve the traveling salesman problem" instruction.
  - This isn't a big restriction, because no real computer will have instructions like this anyway.

Running Time of the RAM Simulation

(Continued list of restrictions for this simulation)

- Second, we can't have instructions that "drastically" increase the length of some number.
  - Example: If integer values are unrestricted, then starting with a value 2, applying a hypothetical "multiply a value by itself" instruction n times would lead to a value of 2^{2^n}, which takes 2^{n+1} bits to represent. Just writing these down takes exponential time, so we don't allow instructions like this.
  - In reality, this is also a mild restriction. In real computers, the lengths of individual numbers are limited.

Running Time of the RAM Simulation

Theorem 8.17: If a computer has **(1)** only instructions that increase the length of a number by 1 bit and **(2)** has only instructions that a multi-tape TM can perform on numbers of length k in $O(k^2)$ steps, then a TM exists that can simulate n steps of the computer in $O(n^3)$ moves.

Notes:

- **(1)** rules out multiplication, which can double the length of a number. However, multiplication can be simulated in polynomial time by repeatedly adding, which is allowed.
- The $O(k^2)$ bound in **(2)** was selected because it seems sufficient for most “reasonable” instructions.

Running Time of the RAM Simulation

We won't prove Theorem 8.17 in detail. However,

- If individual instructions can be simulated in polynomial time, we only need to worry about spending an exponential amount of time scanning the "memory" tape.
- Initially, the "memory" tape holds the program to execute and input data, all of which starts at a "constant" length.
- Restriction **(1)** ensures that the tape's length remains polynomial in n (after executing n RAM instructions). See book Section 8.6.3 for the proof.

Final Comment on RAM Simulation

- We know from before that a single-tape TM can simulate a multi-tape TM in polynomial time.
- So, even a single-tape TM can simulate a RAM computation in polynomial time.