Automata Theory

CS411-2015S-12

Turing Machine Modifications

David Galles

Department of Computer Science
University of San Francisco
When we added a stack to NFA to get a PDA, we increased computational power.

Can we do the same thing for Turing Machines?

That is, can we add some new "feature" to TMs that will increase their computational power?
12-1: **Multi-Track Tape**

- Instead of each tape location holding a single symbol, we add several “tracks” to the tape
  - Based on contents of all tracks, either move head left, move head right, or write new values to any of the tracks

```
| a | b | b | c | b | a |
```

Read/write head
12-2: Multi-Track Tape

- Can simulate a multi-track machine with a standard TM
  - Increase the size of the tape alphabet
  - $k$ tracks, each with an alphabet of $n$ symbols
  - New alphabet of size $n^k$
12-3: Multi-Track Tape

\[ \begin{array}{cccc}
\text{b} & \text{a} & \text{b} & \text{a} \\
\text{a} & \text{b} & \text{b} & \text{a} \\
\end{array} \ldots \]

\[ \text{Read/write head} \]

\[ a = \begin{bmatrix} a \end{bmatrix}, \quad b = \begin{bmatrix} b \end{bmatrix}, \quad C = \begin{bmatrix} a \end{bmatrix}, \quad D = \begin{bmatrix} a \\ a \end{bmatrix} \]

\[ E = \begin{bmatrix} a \\ b \end{bmatrix}, \quad F = \begin{bmatrix} b \end{bmatrix}, \quad G = \begin{bmatrix} b \\ a \end{bmatrix}, \quad H = \begin{bmatrix} b \\ b \end{bmatrix} \]

\[ \begin{array}{cccc}
\text{G} & \text{E} & \text{F} & \text{E} \\
\text{a} & \text{b} & \text{b} & \text{a} \\
\end{array} \ldots \]

\[ \text{Read/write head} \]
12-4: Multiple Tapes

- Several tapes, with independent read/write heads
- Reach symbol on each tape, and based on contents of all tapes:
  - Write or move each tape independently
  - Transition to new state
Create a 2-Tape Machine that adds two numbers
• Convert $\rightarrow \underline{w}; v$ to $\rightarrow \underline{w} + v$ (leading zeros OK)
• Assume that tape 1 holds input (and output), and tape 2 starts out with blanks
Create a 2-Tape Machine that adds two numbers

- Convert \( w; v \) to \( w + v \) (leading zeros OK)

- Copy first # to second tape (zeroing out first # on first tape)

- Do “standard addition”, keeping track of carries.
12-7: Multiple Tapes

- Create a 2-Tape Machine that adds two numbers
12-8: Multiple Tapes

- Are $k$-tape machines more powerful than 1-tape machines?
12-9: Multiple Tapes

[Diagram showing multiple tapes with symbols and numbering]
12-10: Multiple Tapes

- Each transition from the original, multi-tape machine will require several transitions from the simulated machine – and each state in the multiple-tape machine will be represented by a set of states in the simulation machine
  - First, need to scan tape head to find all “virtual heads”, and remember what symbol is stored at each head location
    - Use state to store this information
  - Next, scan tape to implement the action on each tape (moving head, rewriting symbols, etc)
  - Finally, transition to a new set of states
12-11: 2-Way Infinite Tape

... a b c d e f g h i ...
12-12: 2-Way Infinite Tape
12-13: 2-Way Infinite Tape

- Make 2 copies of states in original machine: One set for top tape, one set for bottom tape
- Top Tape States
  - Use the top track
  - Execute as normal
  - When “Move Left” command, and beginning of tape symbol is on the bottom tape, move Right instead, switch to Bottom Tape States
12-14: 2-Way Infinite Tape

- Make 2 copies of states in original machine: One set for top tape, one set for bottom tape
- **Bottom Tape States**
  - Use the bottom track
  - Move left on a “Move Right” command, move right on a “Move Left” command
  - When the beginning of tape symbol is encountered, switch to Top Tape States
12-15: **Simple Computer**

- **CPU**
- **3 Registers (Instruction Register (IR), Program Counter (PC), Accumulator (ACC))**
- **Memory**
- **Operation:**
  - Set IR $\rightarrow$ MEM[PC]
  - Increment PC
  - Execute instruction in IR
  - Repeat
Simple Computer

CPU

Registers
- Instruction Register
- Program Counter
- Accumulator

Memory

0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015

...
## 12-17: Simple Computer

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>HALT Stop Computation</td>
</tr>
<tr>
<td>01</td>
<td>LOAD x ACC ← MEM[x]</td>
</tr>
<tr>
<td>02</td>
<td>LOADI x ACC ← x</td>
</tr>
<tr>
<td>03</td>
<td>STORE x MEM[x] ← AC</td>
</tr>
<tr>
<td>04</td>
<td>ADD x ACC ← ACC + MEM[x]</td>
</tr>
<tr>
<td>05</td>
<td>ADDI x ACC ← ACC + x</td>
</tr>
<tr>
<td>06</td>
<td>SUB x ACC ← ACC - MEM[x]</td>
</tr>
<tr>
<td>07</td>
<td>SUBI x ACC ← ACC - x</td>
</tr>
<tr>
<td>08</td>
<td>JUMP x IP ← x</td>
</tr>
<tr>
<td>09</td>
<td>JZERO x IP ← x if ACC = 0</td>
</tr>
<tr>
<td>10</td>
<td>JGT x IP ← x if ACC &gt; 0</td>
</tr>
</tbody>
</table>

Write a program that multiplies two numbers (in locations 1000 & 1001), and stores the result in 1002
12-18: **Simple Computer**

<table>
<thead>
<tr>
<th>Memory</th>
<th>Machine Code</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>011000</td>
<td>LOAD 1000</td>
</tr>
<tr>
<td>0002</td>
<td>031003</td>
<td>STORE 1003</td>
</tr>
<tr>
<td>0003</td>
<td>020000</td>
<td>LOADI 0</td>
</tr>
<tr>
<td>0004</td>
<td>031002</td>
<td>STORE 1002</td>
</tr>
<tr>
<td>0005</td>
<td>021003</td>
<td>LOAD 1003</td>
</tr>
<tr>
<td>0006</td>
<td>090012</td>
<td>JZERO 0012</td>
</tr>
<tr>
<td>0007</td>
<td>070001</td>
<td>SUBI 1</td>
</tr>
<tr>
<td>0008</td>
<td>031003</td>
<td>STORE 1003</td>
</tr>
<tr>
<td>0009</td>
<td>011002</td>
<td>LOAD 1002</td>
</tr>
<tr>
<td>0010</td>
<td>041001</td>
<td>ADD 1001</td>
</tr>
<tr>
<td>0011</td>
<td>080004</td>
<td>STORE 1002</td>
</tr>
<tr>
<td>0012</td>
<td>000000</td>
<td>HALT</td>
</tr>
</tbody>
</table>
We can simulate this computer with a multi-tape Turing machine:

- One tape for each register (IR, IP, ACC)
- One tape for the Memory
  - Memory tape will be entries of the form `<address> <contents>`
12-20: Computers & TMs

Memory

| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | & | 0 | 0 | 0 | 2 | 0 | 3 | 1 | 0 | 0 | 3 | & | 0 | 0 | 0 | 3 | & | 0 | 2 | 0 | 0 | 0 | 0 | & | 0 | 0 | 0 | 4 | & | 0 | 3 | 1 | 0 | 0 | 2 | & | 0 | 0 | 0 | 1 | & | 0 | 1 | 1 | 0 | 0 | 0 | 0 | & | 0 | 0 | 0 | 2 | & | 0 | 3 | 1 | 0 | 0 | 3 | & | 0 | 0 | 0 | 3 | & | 0 | 2 | 0 | 0 | 0 | 0 | & | 0 | 0 | 0 | 4 | & | 0 | 3 | 1 | 0 | 0 | 2 | ...

Instruction Pointer

| 0 | 0 | 0 | 1 |

Instruction Register

| 1 | 1 | 0 | 0 | 0 |

Accumulator

| 0 |
12-21: Computers & TMs

- Operation:
  - Scan through memory until reach an address that matches the IP
  - Copy contents of memory at that address to the IR
  - Increment IP
  - Based on the instruction code:
    - Copy value into IP
    - Copy a value into Memory
    - Copy a value into the ACC
    - Do addition/subtraction
Memory

Instruction Pointer

Instruction Register

Accumulator
Memory

Instruction Pointer

Instruction Register

Accumulator
12-24: Computers & TMs

Memory

```
... 0001 & 0111000 & 0002 & 031003 & 0003 & 020000 & 0004 & 031002
   \arrow{0100000}

0005 & 021003 & 0006 & 090012 & 0007 & 070001 & 0008 & 031003

0009 & 011002 & 0010 & 041001 & 0011 & 080004 & 0012 & 000000
   \arrow{0000000}
...
```

Instruction Pointer

```
0001 \arrow{0111000} ... (LOAD 1000)
```

Instruction Register

```
011000 \arrow{0110000} ...
```

Accumulator

```
\arrow{0110000} ...
```
12-25: Computers & TMs

Memory

0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2

0 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3

0 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0

... 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 0 0 ...

Instruction Pointer

0 0 0 2 ...

Instruction Register

0 1 1 0 0 0 ...

(Load 1000)

Accumulator

...
Computers & TMs

Memory

```
0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2
```

```
0 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3
```

```
0 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0
```

... 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 0 0 ...

Instruction Pointer  Instruction Register  Accumulator

```
0 0 0 2
```

```
0 1 1 0 0 0
```

(LOAD 1000)
12-27: Computers & TMs

Memory

```
0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2
0 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3
0 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0
```

... 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 0 0 ...

Instruction Pointer

```
0 0 0 2 ...
```

Instruction Register

```
0 1 1 0 0 0 ...
```

Accumulator

```
0 0 0 0 2 1 ...
```

(LOAD 1000)
Memory

```
0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2

0 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3

0 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0

... 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 0 0 ...```

Instruction Pointer

```
0 0 0 2 ...```

Instruction Register

```
...```

Accumulator

```
0 0 0 0 2 1 ...```
12-29: Computers & TMs

Memory

```
0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2
```

```
0 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3
```

```
0 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0
```

```
... 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 0 0 ...
```

Instruction Pointer  Instruction Register  Accumulator

```
0 0 0 2 ...  
```

```
...  
```

```
0 0 0 0 2 1 ...  
```
12-30: Computers & TMs

Memory

```
0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2
```

```
0 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3
```

```
0 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0
```

... 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 0 0 ...

Instruction Pointer

```
0 0 0 2
```

... (STORE 1003)

Instruction Register

```
0 3 1 0 0 3
```

... 

Accumulator

```
0 0 0 0 2 1
```

...
Memory

```
0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2

0 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3

0 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0
```

... 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 0 0 ...

Instruction Pointer

```
0 0 0 3 ...
```

Instruction Register

```
0 3 1 0 0 3 ...
```

Accumulator

```
0 0 0 0 2 1 ...
```

(STORE 1003)
12-32: Computers & TMs

Memory

0001 & 011000 & 0002 & 031003 & 0003 & 020000 & 0004 & 031002

0005 & 021003 & 0006 & 090012 & 0007 & 070001 & 0008 & 031003

0009 & 011002 & 0010 & 041001 & 0011 & 080004 & 0012 & 000000

... 1000 & 000021 & 1001 & 00005 & 1002 & 000000 & 1003 & 000000 ...

Instruction Pointer

0003 ...

Instruction Register

031003 ...

(STORE 1003)

Accumulator

000021 ...

000000
Memory

0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2

0 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3

0 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0

... 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 2 1 ...

Instruction Pointer

0 0 0 3 ... 0 3 1 0 0 3 ...

Instruction Register

(STORE 1003)

Accumulator

0 0 0 2 1 ...
Memory

| 0 0 0 1 & 0 1 1 0 0 0 & 0 0 0 2 & 0 3 1 0 0 3 & 0 0 0 3 & 0 2 0 0 0 0 & 0 0 0 4 & 0 3 1 0 0 2 |

| 0 0 5 & 0 2 1 0 0 3 & 0 0 0 6 & 0 9 0 0 1 2 & 0 0 0 7 & 0 7 0 0 0 1 & 0 0 0 8 & 0 3 1 0 0 3 |

| 0 0 9 & 0 1 1 0 0 2 & 0 0 1 0 & 0 4 1 0 0 1 & 0 0 1 1 & 0 8 0 0 0 4 & 0 0 1 2 & 0 0 0 0 0 0 |

... | 1 0 0 0 & 0 0 0 0 2 1 & 1 0 0 1 & 0 0 0 0 5 & 1 0 0 2 & 0 0 0 0 0 0 & 1 0 0 3 & 0 0 0 0 2 1 ... 

Instruction Pointer

| 0 0 0 3 | ... 

Instruction Register

| ... |

Accumulator

| 0 0 0 0 2 1 | ... |
“Simple Computer” can be modeled by a Turing Machine

Any current machine can be modeled in the same way by a Turing Machine

If there is an algorithm for it, a Turning Machine can do it
  - Note that at this point, we don’t care *how long* it might take, just that it can be done
A computation formalism is “Turing Complete” if it can simulate a Turing Machine.

Turing Complete $\Rightarrow$ can compute anything

Of course it might not be convenient ...
Final extension to Turing Machines: Non-Determinism

- Just like non-determinism in NFAs, PDAs
- String is accepted by a non-deterministic Turing Machine if there is at least one computational path that accepts
12-38: Non-Determinism

A Non-Deterministic Machine $M$ Decides a language $L$ if:

- All computational paths halt
- For each $w \in L$, at least one computational path for $w$ accepts
- For all $w \notin L$, no computational path accepts
A Non-Deterministic Machine $M$ Semi-Decides a language $L$ if:

- For each $w \in L$, at least one computational path for $w$ halts and accepts
- For all $w \notin L$, no computational path halts and accepts
Non-Determinism

A Non-Deterministic Machine $M$ Computes a Function if:

- All computational paths halt
- Every computational path produces the same result
Non-Determinism

- Non-Deterministic TM for \( L = \{ w \in \{0, 1\} : w \text{ is composite} \} \)
- (semi-decides is OK)
Non-Deterministic TM for \( L = \{w \in \{0, 1\} : w \text{ is composite} \} \)

How could we make this machine decide (instead of semi-decide) \( L \)?
How we can make this machine decide (instead of semi-decide) $L$

- First, transform $w$ into $w \sqcup w; w$
- Non-deterministically modify the second 2 $w$’s
- Multiply the second 2 $w$’s
- Check to see if the resulting string is $w \sqcup w$
Non-Determinism

- Are Non-Deterministic Turing Machines more powerful than Deterministic Turing machines?
  - Is there some $L$ which can be semi-decided by a non-deterministic Turing Machine, which cannot be semi-decided by a Deterministic Turing Machine?
- Non-determinism in Finite Automata didn’t buy us anything
- Non-determinism in Push-Down Automata did
12-45: Non-Determinism

- How to Simulate a Non-Deterministic Turing Machine with a Deterministic Turing Machine
How to Simulate a Non-Deterministic Turing Machine with a Deterministic Turing Machine

- Try one computational path – if it says yes, halt and say yes. Otherwise, try a different computational path. Repeat until success
How to Simulate a Non-Deterministic Turing Machine with a Deterministic Turing Machine

Try one computational path – if it says yes, halt and say yes. Otherwise, try a different computational path. Repeat until success.

But what if the first computational path runs forever . . .
Non-Determinism

- How to Simulate a Non-Deterministic Turing Machine with a Deterministic Turing Machine
  - Try all computational paths of length 1
  - Try all computational paths of length 2
  - Try all computational paths of length 3
  - ...
- If there is a halting configuration, you will find it eventually. Otherwise, run forever.
12-49: Non-Determinism

Original Tape

Work Tape

Control Tape
12-50: Non-Determinism

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>(\square)</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q2</td>
<td>(q2,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
```
12-51: Non-Determinism

Original Tape

| a | b | a | a | ... |

Work Tape

| a | b | a | a | ... |

Control Tape

| 1 | ... |

State: q0

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
12-52: Non-Determinism

Original Tape

Work Tape

Control Tape

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>⊘</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

State: q1
12-53: Non-Determinism

Original Tape

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

...  

Work Tape

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

...  

Control Tape

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...  

State: q0

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
Non-Determinism

Original Tape

|$\triangleright$| a | b | a | a | ... |

Work Tape

|$\triangleright$| a | b | a | a | ... |

Control Tape

|$\triangleright$| 2 | ... |

| State: q1 |

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>□</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
### Non-Determinism

#### Original Tape
```
\[ \begin{array}{|c|c|c|c|}
  \hline
  a & b & a & a \\
  \hline
\end{array} \ldots \]
```

#### Work Tape
```
\[ \begin{array}{|c|c|c|c|}
  \hline
  a & b & a & a \\
  \hline
\end{array} \ldots \]
```

#### Control Tape
```
\[ \begin{array}{|c|c|c|}
  \hline
  1 & 1 & \ldots \\
  \hline
\end{array} \ldots \]
```

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

State: q0
### Non-Determinism

#### Original Tape

```
  ▲
 ▶ a b a a ... ▼
```

#### Work Tape

```
  ▲
 ▶ a b a a ... ▼
```

#### Control Tape

```
  ▲
 ▶ 1 1 ... ▼
```

#### Table: States and Transitions

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>a</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>a</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q2</td>
<td>a</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>a</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>a</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

**State:** q1
12-57: Non-Determinism

Original Tape

| a | b | a | a | ...
|---|---|---|---|---|

Work Tape

| a | b | a | a | ...
|---|---|---|---|---|

Control Tape

| 1 | 1 | ...
|---|---|---|

State: q1

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>Acceptance</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q2</td>
<td>(q2,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
### Non-Determinism

#### Original Tape
```
\[ \begin{array}{cccc}
\rightarrow & a & b & a & a \\
\end{array} \ldots \]
```

#### Work Tape
```
\[ \begin{array}{cccc}
\rightarrow & a & b & a & a \\
\end{array} \ldots \]
```

#### Control Tape
```
\[ \begin{array}{c}
1 & 2 \\
\end{array} \ldots \]
```

#### Control Tape
```
\[ \begin{array}{c}
1 & 2 \\
\end{array} \ldots \]
```

#### Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

State: q0
Non-Determinism

Original Tape

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
</table>

Work Tape

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
</table>

Control Tape

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

State: q1

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
12-60: Non-Determinism

Original Tape

Work Tape

Control Tape

State: q2

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>□</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q2</td>
<td>(q2,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
12-61: Non-Determinism

Original Tape

Work Tape

Control Tape

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

State: q0
**Non-Determinism**

Original Tape

```
> | a | b | a | a | ...
```

Work Tape

```
> | a | b | a | a | ...
```

Control Tape

```
> | 2 | 1 | ...
```

<table>
<thead>
<tr>
<th>State: q1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
### Non-Determinism

#### Original Tape

![Original Tape Diagram](image)

#### Work Tape

![Work Tape Diagram](image)

#### Control Tape

![Control Tape Diagram](image)

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next State</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>a</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>b</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(q2,R)</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>a</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>b</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>a</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(q4,R)</td>
<td></td>
</tr>
</tbody>
</table>

State: q1
### Non-Determinism

#### Original Tape

- **Input:** a b a a

#### Work Tape

- **Input:** a b a a

#### Control Tape

- **Input:** 2 2

#### Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

**State:** q0
# Non-Determinism

## Original Tape
```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
</table>
```

## Work Tape
```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
</table>
```

## Control Tape
```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>
```

### State Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

State: q0
## Non-Determinism

### Original Tape

```
>  a b a a
```

### Work Tape

```
>  a b a a
```

### Control Tape

```
>  1 1 1 2 1 1
```

### State Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td></td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td></td>
</tr>
</tbody>
</table>

State: q1
12-67: Non-Determinism

Original Tape

```
> | a | b | a | a | ...
```

Work Tape

```
> | a | b | a | a | ...
```

Control Tape

```
> | 1 | 1 | 1 | 2 | 1 | ...
```

State: q1

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>□</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

The control tape is used to keep track of the state transitions. The work tape is used to process the input tape. The state transitions are given in the table, where the first column represents the current state, the second column represents the symbol read, the third column represents the symbol to be written (or nothing if □), and the fourth column represents the next state.
Non-Determinism

Original Tape

Work Tape

Control Tape

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>□</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

State: q1
**12-69: Non-Determinism**

### Original Tape
- Tape Content: `a b a a`...
- Current State: `q0`
- Action: `(q1,R)`
- Next State: `q1`

### Work Tape
- Tape Content: `a b a a`...
- Current State: `q1`
- Action: `(q2,R)`
- Next States: `q2, q3, q4`...

### Control Tape
- Tape Content: `1 1 1 2 1 1`...
- Current State: `q2`

### Table

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>

**State:** `q2`
Non-Determinism

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>(\square)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>((q_1,R))</td>
<td>((q_1,R))</td>
<td>((q_1,R))</td>
</tr>
<tr>
<td>(q_1)</td>
<td>((q_1,R))</td>
<td>((q_1,R))</td>
<td>((q_4,R))</td>
</tr>
<tr>
<td></td>
<td>((q_2,R))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(q_2)</td>
<td>((q_3,R))</td>
<td>((q_4,R))</td>
<td>((q_4,R))</td>
</tr>
<tr>
<td>(q_3)</td>
<td>((q_4,R))</td>
<td>((q_4,R))</td>
<td>yes</td>
</tr>
<tr>
<td>(q_4)</td>
<td>((q_4,R))</td>
<td>((q_4,R))</td>
<td>((q_4,R))</td>
</tr>
</tbody>
</table>

State: \(q_3\)
12-71: Non-Determinism

**Original Tape**

```
| a | b | a | a |
```

**Control Tape**

```
| 1 | 1 | 1 | 2 | 1 | 1 |
```

**Work Tape**

```
| a | b | a | a |
```

| State: yes |

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q1,R)</td>
<td>(q1,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td></td>
<td>(q2,R)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>(q3,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>yes</td>
</tr>
<tr>
<td>q4</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
<td>(q4,R)</td>
</tr>
</tbody>
</table>
Some Turing Machine review problems:

Create a Turing Machine that *semi-decides* the language \( L = \) all strings over \( \{a, b\} \) with at least as many a’s as b’s
Create a Turing Machine that semi-decides the language \( L = \text{all strings over \{a, b\} with at least as many a's as b's} \)
Some Turing Machine review problems:

Create a Turing Machine that computes the function $\lceil \lg x \rceil$, where $x$ is a binary number.
Some Turing Machine review problems:

- Create a Turing Machine that computes the function $\lceil \lg x \rceil$, where $x$ is a binary number
  - Set result to 0
  - While $x \leq 2$, divide $x$ by 2, and add one to the result
Create a Turing Machine that computes the function $\lceil \lg x \rceil$, where $x$ is a binary number.
Create a Turing Machine that computes the function $\lceil \lg x \rceil$, where $x$ is a binary number.

- Initialize result to 0
- Eliminate leading zeroes
- Set marker for shifting at end of computation
- Blank out MSB
- Increment Result
- $M_{\text{LeftShift}}$
12-78: Turing Machines

\[ M_{\text{LeftShift}} \]

\[
R \xrightarrow{x=0,1} ZL_Y R \xrightarrow{x=0,1} R_Z R \xrightarrow{\square} L_Y \]

x = 0, 1