Turing Machines, continued

Lecture 19
19 November 2019
Exam 2 due now

Assignments 1–5 graded!

Assignment 6 graded by next class
Review from last time
To picture a Turing machine, imagine a mathematician performing calculations on a scroll of paper.

So that we don’t need to worry about running out of places to write things down, imagine the scroll is infinitely long.

The mathematician will be able to solve any computational problem that’s solvable, no matter how many operations are involved – though it might take him a very long time!
Alan Turing showed that any calculation that can be performed by a *smart mathematician* can also be performed by a *stupid but meticulous clerk* who follows a simple set of rules for reading and writing the information on the scroll.
In fact, Turing showed that the human clerk can be replaced by a *finite state machine*.

The finite state machine looks at only one symbol on the scroll at a time, so it’s best thought of as a narrow paper tape, with a single symbol on each line.

Today we call the combination of a finite-state machine with an infinitely long tape a *Turing machine*. 
Our first Turing machine

This is the Turing machine’s finite state control. It issues commands that drive the operation of the machine.
Our first Turing machine

This is the TM’s infinite tape. Each tape cell holds a tape symbol. Initially all tape symbols are blank.
Our first Turing machine

The machine is started with the input string written somewhere on the tape. The tape head initially points to the first symbol of the input string.
Our first Turing machine

Like other automata, TMs begin execution in their start state
Our first Turing machine

At each step, the TM only looks at the symbol immediately under the tape head.
Our first Turing machine

These two transitions originate at the current state. We’re going to choose one of them to follow.
Our first Turing machine

Each transition has the form
\( \langle \text{read} \rangle \rightarrow \langle \text{write} \rangle, \langle \text{direction} \rangle \)
and means “if symbol read is under the tape head, replace it with write and move the tape head in direction (left or right)”. The \( \square \) symbol denotes a blank cell.
Our first Turing machine

Each transition has the form

\[ \langle \text{read} \rangle \rightarrow \langle \text{write} \rangle, \langle \text{direction} \rangle \]

and means “if symbol read is under the tape head, replace it with write and move the tape head in direction (left or right)”. The □ symbol denotes a blank cell.
Our first Turing machine

Each transition has the form
\[ \langle \text{read} \rangle \rightarrow \langle \text{write} \rangle, \langle \text{direction} \rangle \]
and means “if symbol \text{read} is under the tape head, replace it with \text{write} and move the tape head in \text{direction} (left or right)”. The \( \square \) symbol denotes a blank cell.
Our first Turing machine

Each transition has the form
\[\langle \text{read} \rangle \rightarrow \langle \text{write} \rangle, \langle \text{direction} \rangle\]
and means “if symbol read is under the tape head, replace it with write and move the tape head in direction (left or right)”. The □ symbol denotes a blank cell.
Our first Turing machine

Each transition has the form
\[ \langle \text{read} \rangle \rightarrow \langle \text{write} \rangle, \langle \text{direction} \rangle \]
and means “if symbol read is under the tape head, replace it with write and move the tape head in direction (left or right).”
The □ symbol denotes a blank cell.
Unlike a DFA or NFA, a TM doesn’t stop after reading all the input characters. We keep running until the machine explicitly says to stop.
Our first Turing machine

This special state is an **accepting state**. When a TM enters an accepting state, it immediately stops running and accepts whatever the original input string was.
Our first Turing machine

This special state is a rejecting state. When a TM enters a rejecting state, it immediately stops running and rejects whatever the original input string was.
Input and tape alphabets

A Turing machine has two alphabets:

An *input alphabet*, $\Sigma$. All input strings are written in the input alphabet.

A *tape alphabet*, $\Gamma$, where $\Sigma \subset \Gamma$. The tape alphabet contains all symbols that can be written onto the tape.

The tape alphabet $\Gamma$ can contain any number of symbols but always includes at least one *blank symbol*, $\square \notin \Sigma$.

The Turing machine begins with an infinite tape of $\square$ symbols with the input written beginning in the cell under the tape head.
Designing Turing machines
Main question for today: *Just how powerful are Turing machines?*

We’ll move toward an answer to this by seeing how we can use Turing machines to solve different kinds of problems.
Let $\Sigma = \{0, 1\}$ and consider the language
$L = \{0^n1^n \mid n \in \mathbb{N}_0\}$

We know that $L$ is context-free.

How could we build a Turing machine for it?
\[ L = \{0^n1^n \mid n \in \mathbb{N}_0\} \]
A recursive approach

The string $\varepsilon$ is in $L$.

The string $0w1$ is in $L$ iff $w$ is in $L$.

Any string starting with $1$ is not in $L$.

Any string ending with $0$ is not in $L$. 
A sketch of the TM
A sketch of the TM
A sketch of the TM

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

arrow pointing to a specific position.
A sketch of the TM

0 0 1 1 1
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM

0 1 1
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
A sketch of the TM
start → check for 0 → 0 → □, R → □

0 0 0 1 1 1
start

\[
\text{check for } 0 \quad 0 \rightarrow \square, R
\]
start \rightarrow \text{check for } 0 \rightarrow 0 \rightarrow \square, R \rightarrow \text{end, R}

0 \rightarrow 0, R
1 \rightarrow 1, R
start → \text{check for 0} \rightarrow 0 \rightarrow \square, \ R \rightarrow \text{go to end} \rightarrow 0 \rightarrow 0, \ R \rightarrow 1 \rightarrow 1, \ R
start \rightarrow check for 0 \rightarrow 0 \rightarrow \square, R \rightarrow go to end

0 \rightarrow 0, R
1 \rightarrow 1, R

0 0 1 1 1
start $\rightarrow$ \textit{check for 0} $\rightarrow$ $\square$, R $\rightarrow$ \textit{go to end} $\rightarrow$ $\square$, R

$0 \rightarrow 0$, R

$1 \rightarrow 1$, R
start

check for 0

0 → □, R

go to end

0 → 0, R

1 → 1, R

...
start

check for 0

0 → □, R

go to end

□ → □, L

0 → 0, R

1 → 1, R

0 0 1 1 1
start

check for 0

\(0 \rightarrow \square, R\)

go to end

\(\square \rightarrow \square, L\)

\(0 \rightarrow 0, R\)

\(1 \rightarrow 1, R\)
start

check for 0

0 → □, R

go to end

1 → 1, R

□ → □, L

clear a 1

0 → 0, R

0 0 1 1 1 1
start

check for 0

0 → □, R

go to end

□ → □, L

1 → □, L

clear a

1

0 → 0, R

1 → 1, R

0 0 1 1 1

0
0 → 0, L
1 → 1, L

start → check for 0
0 → □, R

go to start
1 → □, L

clear a
□ → □, L

go to end
0 → 0, R
1 → 1, R

[Table]
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
$0 \rightarrow 0, L$
$1 \rightarrow 1, L$

go to
start

$1 \rightarrow \square, L$

clear a
1

$\square \rightarrow \square, L$

start

check
for 0

$0 \rightarrow \square, R$

go to
end

$0 \rightarrow 0, R$

$1 \rightarrow 1, R$

\begin{array}{c}
\text{0 0 1 1}
\end{array}$
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R

go to start

1 → □, L

clear a

□ → □, R

□ → □, L

go to end

0 → 0, R

1 → 1, R

0 1 1
0 → 0, L
1 → 1, L

go to start

1 → □, L
□ → □, R

start

check for 0

□ → □, R

0 → □, R

go to end

0 → 0, R
1 → 1, R

clear a 1

□ → □, L

□ → □, L

1 → 1, R

0 1 1
0 → 0, L
1 → 1, L

0 → □, R
□ → □, R
□ → □, L
□ → □, L

start

check for 0

go to start

1 → □, L
go to end

clear a

0 → 0, R
1 → 1, R

0 1
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R

go to start

1 → □, L

clear a 1

□ → □, R

□ → □, L

go to end

1 → □, R

0 → □, R

0 → □, L

1 → □, R

end
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R
□ → □, R
□ → □, L

go to start

1 → □, L
go to end

clear a 1

0 → □, R
□ → □, L
0 → 0, R
1 → 1, R

1
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R

1 → □, R

go to start

clear a

1 → □, L

□ → □, R

□ → □, L

go to end

0 → 0, R

1 → 1, R

clear a

1 → □, L

□ → □, R

□ → □, L

go to start

0 → 0, L
1 → 1, L

start
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R

1 → □, L

go to start

clear a 1

□ → □, R
□ → □, L

0 → □, L

go to end

0 → 0, R
1 → 1, R

go to end
0 → 0, L
1 → 1, L

go to start

0 → □, R
□ → □, R

check for 0

1 → □, L
□ → □, L

clear a 1

go to end

0 → 0, R
1 → 1, R

start
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R

□ → □, R

go to start

1 → □, L

□ → □, R

clear a 1

□ → □, L

□ → □, R

go to end

0 → 0, R

1 → 1, R

q_{acc}
0 → 0, L
1 → 1, L

0 → □, L
1 → □, L

□ → □, R
□ → □, L
□ → □, R

0 → □, R
1 → □, R

△ → △, R

q_{acc}

start

go to start

clear a 1

go to end

check for 0
0 → 0, L
1 → 1, L

check for 0

□ → □, R

0 → □, R

□ → □, R

0 → □, R

go to start

1 → □, L

clear a 1

□ → □, L

□ → □, L

go to end

0 → 0, R

1 → 1, R

q_{acc}
\[
\begin{align*}
0 & \rightarrow 0, L \\
1 & \rightarrow 1, L
\end{align*}
\]

\[
\begin{align*}
0 & \rightarrow \square, R \\
1 & \rightarrow \square, R \\
\square & \rightarrow \square, R
\end{align*}
\]

\[
\begin{align*}
0 & \rightarrow 0, R \\
1 & \rightarrow 1, R
\end{align*}
\]
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R
□ → □, R

□ → □, L

go to start

1 → □, L

clear a 1

□ → □, L

go to end

0 → 0, R
1 → 1, R

q_{acc}

0 1 1 1
0 → 0, L
1 → 1, L

go to start

1 → □, L

clear a

□ → □, R
□ → □, L

check for 0

0 → □, R
□ → □, R

start

0 → □, R

go to end

0 → 0, R
1 → 1, R

q_{acc}

0 1 1 1
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R

□ → □, R

q_{acc}

go to start

1 → □, L

□ → □, R

clear a 1

□ → □, L

go to end

0 → 0, R
1 → 1, R

0 1 1 1
0 → 0, L
1 → 1, L

start

check for 0

0 → □, R
□ → □, R

q_{acc}

go to start

1 → □, L
□ → □, R

clear a 1

□ → □, L

go to end

0 → 0, R
1 → 1, R

0 → □, R
\[ 0 \rightarrow 0, L \]
\[ 1 \rightarrow 1, L \]

\[ \square \rightarrow \square, R \]

\[ 0 \rightarrow \square, R \]

\[ \square \rightarrow \square, R \]

\[ \square \rightarrow \square, L \]

\[ 0 \rightarrow 0, R \]

\[ 1 \rightarrow 1, R \]

\[ q_{\text{acc}} \]

\[ \text{start} \]

\[ \text{go to start} \]

\[ 1 \rightarrow \square, L \]

\[ \text{check for } 0 \]

\[ \text{clear a} 1 \]

\[ \text{go to end} \]
\[
\begin{align*}
0 & \rightarrow 0, L \\
1 & \rightarrow 1, L \\
\text{go to start} & \rightarrow 1, □, L \\
\text{clear a 1} & \rightarrow □, □, R \\
\text{check for 0} & \rightarrow □, □, R \\
\text{go to end} & \rightarrow 0, 0, R \\
q_{\text{acc}} & \rightarrow 0, 0, R
\end{align*}
\]
0 → 0, L
1 → 1, L

0 → 0, L
1 → 1, L

check for 0

0 → 0, R
□ → □, R
□ → □, R

□ → □, R

□ → □, L

□ → □, L

go to start

goto end

clear a 1

1 → □, L

start

q_{acc}

0 → 0, R
1 → 1, R
0 → 0, L
1 → 1, L

0 → □, R

check for 0

1 → □, L

go to start

clear a 1

□ → □, R

□ → □, L

start

go to end

□ → □, R

0 → □, R

q_{acc}

1 → □, R

1 → 1, R

end
0 → 0, L
1 → 1, L

start

0 → □, R
1 → □, R

check for 0

□ → □, R

□ → □, L

go to end

0 → 0, R
1 → 1, R

clear a

1 → □, L

1 → □, R

goto start

□ → □, R

□ → □, R

q_{acc}

0 → □, R

11
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a
1
□ → □, R
□ → □, L

start
check for 0
0 → □, R
□ → □, R
□ → □, R

q_{acc}

0 → □, R
go to end
0 → 0, R
1 → 1, R

1 1
0 → 0, L
1 → 1, L

$\text{go to start}$

0 → □, R

$\text{check for 0}$

$\text{clear a}$

clear a

$\text{go to end}$

1 → 1, R

$q_{\text{acc}}$

$\text{start}$
Check for 0

Q̄̄→ Q̄̄, L
1→ 1, L
Q̄̄→ Q̄̄, R

Start

0→ Q̄̄, R
1→ Q̄̄, R

Q̄̄→ Q̄̄, R

Q̄̄→ Q̄̄, L

Go to start

1→ Q̄̄, L

Clear a 1

Q̄̄→ Q̄̄, L

Go to end

0→ 0, R
1→ 1, R

Accept

Q̂→ Q̂, R

Reject

Q̂→ Q̂, R
0 → 0, L
1 → 1, L

go to start

1 → □, L
clear a
1

□ → □, R
□ → □, L

check for 0

0 → □, R
go to end

0 → 0, R
1 → 1, R

1 → □, R
□ → □, R

start

q_{acc}

q_{rej}
0 → 0, L
1 → 1, L

0 → □, R
1 → □, R

q_{rej}

q_{acc}

start

check for 0

1 → □, L
go to end

clear a

0 → 0, R
1 → 1, R

0 → □, L

□ → □, L

□ → □, R

□ → □, R

□ → □, R
Start

Check for 0:

- $0 \to 0, L$
- $1 \to 1, L$

- $\square \to \square, R$

Go to start:

- $1 \to \square, L$

Clear a 1:

- $\square \to \square, L$

Go to end:

- $0 \to 0, R$
- $1 \to 1, R$

Qacc

Qrej

$q_{acc} \rightarrow q_{rej}$
start

check for 0

go to start

1 → □, L

clear a 1

□ → □, R

□ → □, L

go to end

0 → □, R

1 → 1, R

start

check for 0

0 → □, R

□ → □, R

□ → □, R

q_{rej}

q_{acc}

q_{acc}
\[
\begin{align*}
0 &\rightarrow 0, \text{L} \\
1 &\rightarrow 1, \text{L} \\
\square &\rightarrow \square, \text{R} \\
1 &\rightarrow \square, \text{L} \\
\text{go to start} &\rightarrow \text{check for } 0 \\
0 &\rightarrow \square, \text{R} \\
1 &\rightarrow \square, \text{R} \\
\square &\rightarrow \square, \text{L} \\
\text{go to end} &\rightarrow 0, \text{R} \\
0 &\rightarrow 0, \text{R} \\
1 &\rightarrow 1, \text{R} \\
\square &\rightarrow \square, \text{R} \\
\end{align*}
\]
\[ \square \rightarrow \square, R \]

\[ 0 \rightarrow 0, L \]
\[ 1 \rightarrow 1, L \]

\[ go \ to \ start \]

\[ 1 \rightarrow \square, L \]

\[ clear \ a \ 1 \]

\[ \square \rightarrow \square, R \]

\[ start \]

\[ check \ for \ \square \]

\[ 0 \rightarrow \square, R \]

\[ go \ to \ end \]

\[ 0 \rightarrow 0, R \]
\[ 1 \rightarrow 1, R \]

\[ q_{\text{acc}} \]

\[ q_{\text{rej}} \]
\[ \square \rightarrow \square, R \]

\[ \begin{align*}
0 & \rightarrow 0, L \\
1 & \rightarrow 1, L
\end{align*} \]

**go to start**

\[ 1 \rightarrow \square, L \]

**clear a 1**

\[ \square \rightarrow \square, R \]

\[ \square \rightarrow \square, L \]

**check for 0**

\[ \square \rightarrow \square, R \]

\[ \square \rightarrow \square, R \]

\[ 0 \rightarrow \square, R \]

\[ 1 \rightarrow \square, R \]

**start**

\[ q_{\text{rej}} \]

\[ q_{\text{acc}} \]

**go to end**

\[ \begin{align*}
0 & \rightarrow 0, R \\
1 & \rightarrow 1, R
\end{align*} \]

\[ \begin{align*}
0 & \rightarrow 0, R \\
1 & \rightarrow 1, R
\end{align*} \]
\[ \Box \rightarrow \Box, \text{R} \]
\[ 0 \rightarrow 0, \text{R} \]

\[ 0 \rightarrow 0, \text{L} \]
\[ 1 \rightarrow 1, \text{L} \]

\[ \text{go to start} \]

\[ 1 \rightarrow \Box, \text{L} \]

\[ \text{clear a} \]
\[ 1 \]

\[ \Box \rightarrow \Box, \text{R} \]
\[ \Box \rightarrow \Box, \text{L} \]

\[ \text{check for 0} \]

\[ 0 \rightarrow \Box, \text{R} \]

\[ \text{go to end} \]

\[ 0 \rightarrow 0, \text{R} \]
\[ 1 \rightarrow 1, \text{R} \]

\[ \text{qrej} \]

\[ \text{qacc} \]
0 → 0, L
1 → 1, L

0 → 0, L
1 → □, L

□ → □, R
□ → □, L

□ → □, R

start → check for 0 → □, R → go to end

q_{acc}
Another Turing machine design

We designed a Turing machine for \( \{0^n1^n \mid n \in \mathbb{N}_0\} \).

Let’s consider how we could design a Turing machine for a related context-free language over \( \Sigma = \{0, 1\} \):

\[
L = \{w \in \Sigma^* \mid w \text{ has the same number of } 0\text{s and } 1\text{s}\}
\]
A caveat
A caveat
A caveat
A caveat
A caveat

0 0 1 1 1 0
A caveat
A caveat
A caveat
A caveat
A caveat

How do we know that this blank isn’t one of the infinitely many blanks after our input string?
A caveat
A caveat
A caveat

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A caveat
A caveat
A caveat

How do we know that this blank isn’t one of the infinitely many blanks after our input string?
A caveat
A caveat

How do we know that this blank isn’t one of the infinitely many blanks after our input string?
One solution
One solution

\[
x \times 0 0 1 1 1 1 1 0
\]
One solution

x 0 0 1 1 1 1 1 0
One solution

\[
x \times 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0
\]
One solution

\[ x \ 0 \ 0 \ x \ 1 \ 1 \ 1 \ 1 \ 0 \]
One solution

$x \times 0 \times 0 \times 1 \times 1 \times 1 \times 0$
One solution

\[ x \times 0 \times 0 \times 1 \times 1 \times 1 \times 1 \times 0 \]
One solution
One solution
One solution

\[ \times 000 \times 11110 \]
One solution

x x 0 x 1 1 1 1 0
One solution

\[ \begin{array}{ccccccc}
  & & & x & x & 0 & x & 1 & 1 & 1 & 1 & 0 & \\
\end{array} \]
One solution
One solution

\[ x \ x \ 0 \ x \ x \ 1 \ 1 \ 0 \]
One solution

x x 0 x x 1 1 0
One solution

\[
x \times 0 \times x 1 1 0
\]
One solution
One solution
One solution
One solution

```
x x 0 x x 1 1 0
```
One solution

x x 0 x x 1 1 0
One solution

x x x x x x 1 1 0
One solution
One solution
One solution
One solution
One solution
One solution

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
One solution
One solution
One solution
One solution

\[ \begin{array}{ccccccccccc}
\times & \times & \times & \times & \times & \times & \times & 1 & 0 & \end{array} \]
One solution
One solution

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
One solution
One solution
One solution

Now the first non-x character we encounter is a 1, so we want to cross it off but remember that we're looking for a matching 0
start
start

find 0/1

0 \rightarrow x, R

0 0 1 1 1 1 1 1 0 0
$\begin{array}{l}
\text{start} \rightarrow \text{find 0/1} \\
0 \rightarrow x, R
\end{array}$
start \rightarrow find 0/1

0 \rightarrow x, R \rightarrow find 1

x 0 1 1 1 1 1 1 1 0 0
start → find 0/1

0 → x, R

find 1

0 → 0, R

x 0 1 1 1 1 1 0 0
start → find $0/1$

$0 \rightarrow x, R$

find $1$

$1 \rightarrow x, R$

$0 \rightarrow 0, R$

---

$x \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0$
The diagram illustrates a process starting from the 'start' node, moving through a series of states labeled 'find 0/1' and 'find 1'.

- From 'start', if the input is 0, it transitions to 'find 0', moving right (R) and setting the output to 0.
- From 'find 0', if the next input is 0, it stays in 'find 0' moving right (R) and setting the output to 0.
- From 'find 0', if the next input is 1, it transitions to 'find 1', moving left (L) and setting the output to 1.
- From 'find 1', if the next input is 1, it stays in 'find 1' moving left (L) and setting the output to 1.

The sequence of inputs and outputs is: x 0 x 1 1 1 1 0 0.
start → find 0/1

find 1 → 1 → x, L

0 → x, R

0 → 0, R

go home
start \quad \rightarrow \quad find \ 0/1

0 \quad \rightarrow \quad x, \ R

find \ 1

1 \quad \rightarrow \quad x, \ L

0 \quad \rightarrow \quad 0, \ R

0 \quad \rightarrow \quad 0, \ L

1 \quad \rightarrow \quad 1, \ L

x \quad \rightarrow \quad x, \ L

go \ home
Find 0/1

0 → x, R

Find 1

1 → x, L

Go home

0 → 0, L
1 → 1, L
x → x, L
\[
x \rightarrow x, R
\]

**start**

\[
\text{find } 0/1
\]

\[
0 \rightarrow x, R
\]

\[
\square \rightarrow \square, R
\]

**find 1**

\[
0 \rightarrow 0, R
\]

\[
1 \rightarrow x, L
\]

**go home**

\[
0 \rightarrow 0, L
\]

\[
1 \rightarrow 1, L
\]

\[
x \rightarrow x, L
\]

\[
\downarrow
\]

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
x & 0 & x & 1 & 1 & 1 & 0 & 0
\end{array}
\]
\begin{align*}
x &\rightarrow x, \text{R} \\
\text{start} &\rightarrow \text{find } 0/1 \\
\text{find } 0/1 &\rightarrow \square \rightarrow \square, \text{R} \\
\text{find } 1 &\rightarrow 1 \rightarrow x, \text{L} \\
\text{go home} &\rightarrow 0 \rightarrow 0, \text{L} \\
&\rightarrow 1 \rightarrow 1, \text{L} \\
&\rightarrow x \rightarrow x, \text{L} \\
\end{align*}
\( x \rightarrow x, R \)

\( \text{start} \rightarrow \text{find 0/1} \)

\( \text{□} \rightarrow \text{□}, R \)

\( \text{find 1} \rightarrow 0 \rightarrow x, R \)

\( 1 \rightarrow x, L \)

\( 0 \rightarrow x, R \)

\( x \rightarrow x, R \)

\( 0 \rightarrow 0, L \)

\( 1 \rightarrow 1, L \)

\( x \rightarrow x, L \)

\( \)
\[ \begin{align*}
\text{x} & \rightarrow \text{x}, \text{R} \\
& \text{start} \\
\text{find 0/1} & \rightarrow \text{□} \rightarrow \text{□}, \text{R} \\
\text{go home} & \rightarrow 0 \rightarrow 0, \text{L} \\
& \text{1} \rightarrow 1, \text{L} \\
& \text{x} \rightarrow \text{x}, \text{L} \\
\text{find 1} & \rightarrow 0 \rightarrow 0, \text{R} \\
& \text{x} \rightarrow \text{x}, \text{R} \\
\end{align*} \]
\[
\begin{align*}
&\text{start} \quad \text{find } 0/1 \\
&\quad \text{go home} \\
&\quad \text{find } 1 \\
\end{align*}
\]
\[ \begin{array}{l}
x \to x, \text{R}
\\
\text{start} \to \text{find 0/1}
\\
0 \to x, \text{R}
\\
\square \to \square, \text{R}
\\
\text{find 1} \to \text{find 1}
\\
1 \to x, \text{L}
\\
0 \to 0, \text{R}
\\
x \to x, \text{R}
\\
\text{go home} \to \text{go home}
\\
0 \to 0, \text{L}
\\
1 \to 1, \text{L}
\\
x \to x, \text{L}
\\
g \to g, \text{R}
\\
\end{array} \]
start

find 0/1

find 0

find 1

go home

0 → 0, L
1 → 1, L
0 → 0, R
1 → 1, R
x → x, L
x → x, R
□ → □, R
x → x, R
1 → x, R
0 → x, R
x → x, R

x x x x x x 1 0 0
\[
\begin{align*}
&\text{start} \\
&\text{find 0/1} \\
&\text{find 0} \\
&\text{find 1} \\
&\text{go home}
\end{align*}
\]
0 \rightarrow 0, \text{L}
1 \rightarrow 1, \text{L}
x \rightarrow x, \text{L}

\text{find 0/1}

\text{find 0}

\text{find 1}

\text{go home}

\text{start}

\text{x} \rightarrow x, \text{R}
\text{x} \rightarrow x, \text{L}
\text{□} \rightarrow □, \text{R}

\text{1} \rightarrow x, \text{R}
\text{x} \rightarrow x, \text{R}
\text{0} \rightarrow x, \text{R}
\text{1} \rightarrow x, \text{L}
\text{0} \rightarrow x, \text{L}
0 \rightarrow 0, \text{L}
1 \rightarrow 1, \text{L}
x \rightarrow x, \text{L}
start

find 0/1

find 0

find 1

go home

0 → 0, L
1 → 1, L
x → x, L

□ → □, R

0 → x, R
1 → x, R
x → x, R

0 → x, R

1 → x, R
x → x, R

 abolition
Start in the 'find 0/1' state.

- From 'find 0/1', if you see a '0', move to the 'find 0' state.
- From 'find 0/1', if you see a '1', move to the 'find 1' state.
- From 'find 0/1', if you see an 'x', stay in the 'find 0/1' state.

In the 'find 0' state:
- If you see a '0', move to the 'go home' state.
- If you see a '1', stay in the 'find 0' state.
- If you see an 'x', stay in the 'find 0' state.

In the 'find 1' state:
- If you see a '0', move to the 'go home' state.
- If you see a '1', move to the 'find 1' state.
- If you see an 'x', stay in the 'find 1' state.

In the 'go home' state:
- If you see a '0', move to the 'go home' state.
- If you see a '1', move to the 'go home' state.
- If you see an 'x', stay in the 'go home' state.

You can always move right (R) regardless of the symbol you see.

You can move left (L) only when you see a '0' or '1' and are in the 'find 0' or 'find 1' states, respectively.

The bottom row represents the tape with 'x' symbols indicating the current position of the read/write head.
start \rightarrow find 0/1

\begin{align*}
1 &\rightarrow 1, R \\
x &\rightarrow x, R \\
\end{align*}

\begin{align*}
0 &\rightarrow x, R \\
x &\rightarrow x, R \\
\end{align*}

\begin{align*}
\square &\rightarrow \square, R
\end{align*}

find 0

\begin{align*}
1 &\rightarrow x, R \\
x &\rightarrow x, R \\
0 &\rightarrow x, L
\end{align*}

find 1

\begin{align*}
0 &\rightarrow 0, R \\
x &\rightarrow x, R \\
\end{align*}

\begin{align*}
1 &\rightarrow 1, R \\
\end{align*}

\begin{align*}
0 &\rightarrow 0, L \\
x &\rightarrow x, L
\end{align*}

go home

\begin{align*}
1 &\rightarrow 1, L \\
\end{align*}

\begin{align*}
0 &\rightarrow 0, L \\
x &\rightarrow x, L
\end{align*}

\begin{align*}
\end{align*}

\begin{align*}
\end{align*}
start

\( \text{find } 0/1 \)

\( x \rightarrow x, R \)

\( 0 \rightarrow x, R \)

\( 1 \rightarrow x, R \)

\( \square \rightarrow \square, R \)

\( x \rightarrow x, R \)

\( \text{go home} \)

\( 0 \rightarrow 0, L \)

\( 1 \rightarrow 1, L \)

\( x \rightarrow x, L \)

\( 0 \rightarrow x, L \)

\( 1 \rightarrow x, L \)

\( 0 \rightarrow x, R \)

\( 1 \rightarrow x, R \)

\( 0 \rightarrow 0, R \)

\( x \rightarrow x, R \)
Remember that all missing transitions implicitly reject.
These parts of the Turing machine are two “cases” it handles, though the tapes they operate on are indistinguishable.
Constant storage

Sometimes a Turing machine needs to remember some additional information that can’t (at least conveniently) be put on the tape.

In this case, you can use the same techniques you used in designing DFAs and introduce extra states into the Turing machine’s finite-state control.

The finite-state control can only remember one of finitely many things, but that might be all you need.
An alternative approach

Clearly the first language we built a TM to recognize,

\[ \{0^n1^n \mid n \in \mathbb{N}_0\} \]

and this language,

\[ \{w \in \Sigma^* \mid w \text{ has the same number of } 0 \text{s and } 1 \text{s}\} \]

are related.

Could we use our Turing machine for the first language to recognize the second?
An alternative approach

To check if a string is in \{ w ∈ Σ^* | w has the same number of 0s and 1s \}:

1. Sort the string so all 0s are before 1s.
2. Run the Turing machine for \{0^n1^n | n ∈ ℕ₀ \} on the resulting tape.
3. **Accept** if it accepts; **reject** if it rejects.

You can think of this approach as using **subroutines**.
Turing machine subroutines

A **TM subroutine** is a Turing machine that, instead of accepting or rejecting an input, does some sort of processing job.

TM subroutines let us compose larger TMs out of smaller TMs, just as you’d use a variety of helper functions to write a complicated program.

Here, we could write a TM subroutine to sort a sequence of 0s and 1s into ascending order.
Turing machine subroutines

Note that subroutines are not a special part of the model for Turing machines.

They are a convenient way to *think* about building a single complex Turing machine out of simpler pieces.
Turing machine subroutines

Typically, when a subroutine is done running, you have it enter a state marked “done” with a dashed line around it.

When we’re composing multiple subroutines together, the idea is that we’ll snap in a real state for the “done” state.
We won’t go through implementing a Turing machine subroutine to sort the input string.

You can do it in very much the same way you would implement a sorting algorithm in a programming language.
Using subroutines to solve one problem using the solution to another is also a move toward the last proof technique we’ll cover in the course, reductions.

More on them later!
Example: Turing machine arithmetic
Let’s design a Turing machine that, given a tape that looks like this:

ends up having the tape look like this:

In other words, a Turing machine that can add two numbers.
There are many ways we could design this Turing machine.

Here’s one approach:

- Build a Turing machine to *increment* a number.
- Build a Turing machine to *decrement* a number.

*Combine them*, repeatedly decrementing the second number and incrementing the first.
def add(num1, num2):
    while num2 > 0:
        increment(num1)
        decrement(num2)
def add(num1, num2):
    while num2 > 0:
        increment(num1)  
        decrement(num2)  
        Let’s write this subroutine first.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

the tape head points at the start of a number
there are at least two blanks to the left of the number
there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

the tape head points at the start of a number
there are at least two blanks to the left of the number
there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

the tape head points at the start of a number
there are at least two blanks to the left of the number
there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.

```
9 0 0
```
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

the tape head points at the start of a number
there are at least two blanks to the left of the number
there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

Let’s begin by building a TM that increments a number. We’ll assume that

- the tape head points at the start of a number
- there are at least two blanks to the left of the number
- there’s at least one blank at the end of the number

The tape head will end at the start of the number after incrementing it.
Incrementing numbers

def increment(num):
    # go to the end of the number
    while current_digit == 9:
        current_digit = 0
        # go left one digit
        current_digit += 1
    # go to the start of the number
start
\begin{verbatim}
start \rightarrow to end
0 \rightarrow 0, R
1 \rightarrow 1, R
\ldots
9 \rightarrow 9, R
\end{verbatim}
\begin{align*}
0 \rightarrow &\ 0, R \\
1 \rightarrow &\ 1, R \\
\ldots &\ \\
9 \rightarrow &\ 9, R
\end{align*}
0 \rightarrow 0, R
1 \rightarrow 1, R
\ldots
9 \rightarrow 9, R
0 → 0, R
1 → 1, R
... 
9 → 9, R

start to end
start → to end

0 → 0, R
1 → 1, R
...
9 → 9, R

□ → □, L
0 → 0, R
1 → 1, R
...
9 → 9, R
0 → 0, R
1 → 1, R
... 9 → 9, R

start to end

□ → □, L

wrap 9s

? → ?, ?
\begin{align*}
\text{start} & \rightarrow \text{to end} & \square \rightarrow \square, \text{ L} \\
0 & \rightarrow 0, \text{ R} \\
1 & \rightarrow 1, \text{ R} \\
\vdots & \\
9 & \rightarrow 9, \text{ R} \\
\text{wrap 9s} & \rightarrow 9 \rightarrow 0, \text{ L}
\end{align*}
The diagram shows a transition graph with states labeled as 'start', 'to end', and 'wrap 9s'. The transitions are as follows:

- From 'start', the machine moves to 'to end' on input □, with direction L.
- From 'to end', the machine moves back to itself on input □, with direction L.
- From 'to end', the machine moves to 'wrap 9s' on input 9, with direction L.
- In 'wrap 9s', the machine wraps 9s on input 9, with direction R.

The transitions for inputs 0, 1, and 9 are as follows:

- Input 0: Move to 0, direction R.
- Input 1: Move to 1, direction R.
- Input 9: Move to 9, direction R.

The diagram also shows a wrapped pattern of numbers from 1 to 0, with each number having an arrow pointing to the next number in the sequence.
start

to end

□ → □, L

0 → 0, R
1 → 1, R
...
9 → 9, R

wrap 9s

9 → 0, L

0 → 1, L
1 → 2, L
...
8 → 9, L

done!

□ → □, R

back home

0 → 0, L
1 → 1, L
...
9 → 9, L

1 0 0 2
start → to end
0 → 0, R
1 → 1, R
... 
9 → 9, R
to end → wrap 9s
9 → 0, L
wrap 9s → to end
0 → 1, L
1 → 2, L
... 
8 → 9, L
to end → done!
done! → back home
1 → 1, L
... 
9 → 9, L
back home → to end
0 → 0, L
1 → 1, L
... 
9 → 9, L
done! → 1
1 0 0 2
- **start**
  - \( \square \rightarrow \square, L \)
- **to end**
  - \( 0 \rightarrow 0, R \)
  - \( 1 \rightarrow 1, R \)
  - \( \ldots \)
  - \( 9 \rightarrow 9, R \)
- **wrap 9s**
  - \( 9 \rightarrow 0, L \)
    - \( 0 \rightarrow 1, L \)
    - \( 1 \rightarrow 2, L \)
    - \( 2 \rightarrow 3, L \)
    - \( \ldots \)
    - \( 8 \rightarrow 9, L \)
  - \( 9 \rightarrow 9, L \)
- **back home**
  - \( \square \rightarrow \square, R \)
  - \( 0 \rightarrow 0, L \)
  - \( 1 \rightarrow 1, L \)
  - \( \ldots \)
  - \( 9 \rightarrow 9, L \)

**Board:**

```
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
| 1 | 0 | 0 | 3 |   |   |
+---+---+---+---+---+---+
```
start → to end
0 → 0, R
1 → 1, R
...
9 → 9, R

□ → □, L

wrap 9s
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
...
8 → 9, L

back home

□ → □, R

done!
0 → 0, L
1 → 1, L
...
9 → 9, L

1 0 0 3
The diagram shows a state transition system with three states: `start`, `to end`, and `back home`. The transitions include:

- From `start` to `to end`: □ → □, L
- From `to end` to `wrap 9s`: 9 → 0, L
- From `wrap 9s` to `back home`: 0 → 1, L
- From `back home` to `to end`: □ → □, R
- From `to end` to `back home`: 0 → 0, L
- From `back home` to `back home`: 1 → 1, L
- From `back home` to `back home`: 9 → 9, L

The diagram includes loops for transitions 0 → 0, 1 → 1, 8 → 9, and so on, indicating that the system can wrap around from 9 back to 0.
\[ \begin{align*}
&\textit{start} \\
&\textit{to end} \\
&0 \rightarrow 0, \text{ R} \\
&1 \rightarrow 1, \text{ R} \\
&\ldots \\
&9 \rightarrow 9, \text{ R} \\
&\square \rightarrow \square, \text{ L} \\
&\textit{wrap 9s} \\
&9 \rightarrow 0, \text{ L} \\
&0 \rightarrow 1, \text{ L} \\
&1 \rightarrow 2, \text{ L} \\
&\ldots \\
&8 \rightarrow 9, \text{ L} \\
&\textit{done!} \\
&\square \rightarrow \square, \text{ R} \\
&\textit{back home} \\
&0 \rightarrow 0, \text{ L} \\
&1 \rightarrow 1, \text{ L} \\
&\ldots \\
&9 \rightarrow 9, \text{ L}
\end{align*} \]
start → to end
0 → 0, R
1 → 1, R
... 
9 → 9, R

□ → □, L

wrap 9s
9 → 0, L
0 → 1, L
1 → 2, L
... 
8 → 9, L

□ → □, R
done!

back home
0 → 0, L
1 → 1, L
... 
9 → 9, L

9 9 9
start

to end

□ → □, L

0 → 0, R
1 → 1, R
...
9 → 9, R

wrap 9s

9 → 0, L

0 → 1, L
1 → 2, L
...
8 → 9, L

back home

□ → □, R

0 → 0, L
1 → 1, L
...
9 → 9, L

done!

9 9 9
start

\[\text{to end} \rightarrow [\square \rightarrow \square, \text{L}] \rightarrow \text{wrap 9s} \rightarrow [9 \rightarrow 0, \text{L}] \rightarrow \text{back home} \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [8 \rightarrow 9, \text{L}] \rightarrow \text{done!} \rightarrow [\square \rightarrow \square, \text{R}] \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [9 \rightarrow 9, \text{L}] \rightarrow \text{back home} \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [8 \rightarrow 9, \text{L}] \rightarrow \text{done!} \rightarrow [\square \rightarrow \square, \text{R}] \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [9 \rightarrow 9, \text{L}] \rightarrow \text{back home} \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [8 \rightarrow 9, \text{L}] \rightarrow \text{done!} \rightarrow [\square \rightarrow \square, \text{R}] \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [9 \rightarrow 9, \text{L}] \rightarrow \text{back home} \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [8 \rightarrow 9, \text{L}] \rightarrow \text{done!} \rightarrow [\square \rightarrow \square, \text{R}] \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [9 \rightarrow 9, \text{L}] \rightarrow \text{back home} \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [8 \rightarrow 9, \text{L}] \rightarrow \text{done!} \rightarrow [\square \rightarrow \square, \text{R}] \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [9 \rightarrow 9, \text{L}] \rightarrow \text{back home} \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [8 \rightarrow 9, \text{L}] \rightarrow \text{done!} \rightarrow [\square \rightarrow \square, \text{R}] \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]

\[\vdots \rightarrow [9 \rightarrow 9, \text{L}] \rightarrow \text{back home} \rightarrow [0 \rightarrow 0, \text{L}], [1 \rightarrow 1, \text{L}] \]
Start: to end

0 → 0, R
1 → 1, R
...
9 → 9, R

Wrap 9s:

9 → 0, L
0 → 1, L
1 → 2, L
...
8 → 9, L

Done:

□ → □, R

Back home:

0 → 0, L
1 → 1, L
...
9 → 9, L

9 9 9
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to end

□ → □, L

wrap 9s

9 → 0, L

\[ \text{0 → 1, L} \]
\[ \text{1 → 2, L} \]
\[ \text{...} \]
\[ \text{8 → 9, L} \]
\[ \text{□ → 1, L} \]

\[ \text{0 → 0, L} \]
\[ \text{1 → 1, L} \]
\[ \text{...} \]
\[ \text{9 → 9, L} \]

\[ \text{back home} \]

\[ \text{0 → 0, L} \]
\[ \text{1 → 1, L} \]
\[ \text{...} \]
\[ \text{9 → 9, L} \]

done!
def add(num1, num2):
    while num2 > 0:
        increment(num1)
        decrement(num2)    Next!
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

the tape head points at the start of a number
there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

the tape head points at the start of a number
there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

the tape head points at the start of a number
there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

the tape head points at the start of a number
there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

the tape head points at the start of a number
there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
### Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

the tape head points at the start of a number
there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.

1 0 9
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

the tape head points at the start of a number
there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

the tape head points at the start of a number
there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

Now let’s build a TM that decrements a number. We’ll assume that

- the tape head points at the start of a number
- there’s at least one blank on each side of the number

The tape head will end at the start of the number after decrementing it.

If the number is 0, the subroutine should signal this rather than go negative.
Decrementing numbers

def decrement(num):
    go to the end of the number
    if (every digit was 0):
        signal that we’re done
    while current_digit == 0:
        current_digit = 9
        go left one digit
    current_digit -= 1
    go to the start of the number
$0 \rightarrow 0, R$

Start

non-zero?
0 \rightarrow 0, R

\textit{start}

\textit{non-zero?}
0 → 0, R  
start  

1 → 1, R  
2 → 2, R  
...  
9 → 9, R  

Non-zero?
0 → 0, R
1 → 1, R
2 → 2, R
...
9 → 9, R

start

non-zero?
non-zero?

0 → 0, R
1 → 1, R
2 → 2, R
... 
9 → 9, R

to end

start

0 2 0 0
0 → 0, R
1 → 1, R
... 
9 → 9, R

start

non-zero?

to end

0 → 0, R
1 → 1, R
2 → 2, R
... 
9 → 9, R

0 | 2 | 0 | 0
...
non-zero?

start

to end

0 \rightarrow 0, R
1 \rightarrow 1, R
\ldots
9 \rightarrow 9, R

1 \rightarrow 1, R
2 \rightarrow 2, R
\ldots
9 \rightarrow 9, R

0 \rightarrow 0, R

0 2 0 0
0 → 0, R
1 → 1, R
... 
9 → 9, R

non-zero?

start

□ → □, L

0 → 0, R
to end

... 

... 

9 → 9, R

□ → □, L
non-zero?

start

to end

wrap zeroes

0 \rightarrow 0, R
1 \rightarrow 1, R
\vdots
9 \rightarrow 9, R

\square \rightarrow \square, L

0 \rightarrow 9, L

\begin{array}{cccc}
0 & 2 & 9 & 9 \\
\end{array}
non-zero?

start

0 → 0, R
1 → 1, R
...
9 → 9, R

1 → 1, R
2 → 2, R
...
9 → 9, R

0 → 0, R

to end

wrap zeroes

done!

back home

0 → 0, L
1 → 1, L
...
9 → 8, L
9 → 9, L
non-zero?  

wrap zeroes

done!

back home

0 → 0, R
1 → 1, R
...
9 → 9, R

0 → 0, R
1 → 1, R
2 → 2, R
...
9 → 9, R

0 → 9, L
1 → 0, L
2 → 1, L
...
9 → 8, L

0 → 9, L
1 → 0, L
2 → 1, L
...
9 → 9, L

0 1 9 9
non-zero?  

0 → 0, R  
1 → 1, R  
...  
9 → 9, R  

0 → 0, R  
1 → 1, R  
2 → 2, R  
...  
9 → 9, R  

start  

0 → 0, R  
1 → 1, R  
2 → 2, R  
...  
9 → 9, R  

to end  

wrap zeroes  

0 → 9, L  

0 → 9, R  

□ → □, L  

done!  

back home  

1 → 0, L  
2 → 1, L  
...  
9 → 8, L  

1 → 0, L  
2 → 1, L  
...  
9 → 8, L  

□ → □, R  

0 → 0, L  
1 → 1, L  
...  
9 → 9, L
non-zero? 

to end

wrap zeroes

start 0 → 0, R 
1 → 1, R 
... 
9 → 9, R 

non-zero? 1 → 1, R 
2 → 2, R 
... 
9 → 9, R 

done! □ → □, R 

back home 1 → 0, L 
2 → 1, L 
... 
9 → 8, L 

0 → 0, L 
1 → 1, L 
... 
9 → 9, L 

0 1 9 9
non-zero?

start

end

wrap zeroes

done!

back home

The diagram represents a sequence of transitions between states, with arrows indicating the movement from one state to another. The states and transitions are as follows:

- **Start** (0 -> 0, R)
- **Non-zero?** (1 -> 1, R, 2 -> 2, R, 9 -> 9, R)
- **To end**
- **Wrap zeroes** (0 -> 9, L)
- **Back home**

The sequence of inputs includes:

- 0 -> 0, R
- 1 -> 1, R
- 2 -> 2, R
- 9 -> 9, R
- 1 -> 0, L
- 2 -> 1, L
- 9 -> 8, L
- 0 -> 0, L
- 1 -> 1, L
- 9 -> 9, L

The sequence ends with **done!**
non-zero?

start

1 → 1, R
2 → 2, R
... 
9 → 9, R
0 → 0, R

to end

wrap zeroes

1 → 0, L
2 → 1, L
... 
9 → 8, L

□ → □, L

done!

back home

0 → 0, L
1 → 1, L
... 
9 → 9, L

0 1 9 8
0 → 0, R
1 → 1, R
...  
9 → 9, R

start

non-zero?
to end

□ → □, L

wrap zeroes

1 → 0, L
2 → 1, L
...  
9 → 8, L

done!

back home

□ → □, R

0 → 0, L
1 → 1, L
...  
9 → 9, L

0 1 9 8
non-zero?

start

0 \rightarrow 0, R
1 \rightarrow 1, R
...
9 \rightarrow 9, R

0 \rightarrow 0, R
1 \rightarrow 1, R
2 \rightarrow 2, R
...
9 \rightarrow 9, R

to end

\[ \square \rightarrow \square, L \]

wrap zeroes

1 \rightarrow 0, L
2 \rightarrow 1, L
...
9 \rightarrow 8, L

done!

\[ \square \rightarrow \square, R \]

back home

0 \rightarrow 0, L
1 \rightarrow 1, L
...
9 \rightarrow 9, L

0 1 9 8
start

non-zero?

to end

0 \rightarrow 0, R
1 \rightarrow 1, R
2 \rightarrow 2, R
9 \rightarrow 9, R

\square \rightarrow \square, L

wrap zeroes

1 \rightarrow 0, L
2 \rightarrow 1, L
9 \rightarrow 8, L

\square \rightarrow \square, R

back home

0 \rightarrow 0, L
1 \rightarrow 1, L
9 \rightarrow 9, L

done!

0 1 9 8
non-zero?

start

0 → 0, R
1 → 1, R
... 9 → 9, R

to end

wrap zeroes

□ → □, L

1 → 0, L
2 → 1, L
... 9 → 8, L

done!

back home

□ → □, R

0 → 0, L
1 → 1, L
... 9 → 9, L
0 \rightarrow 0, R
1 \rightarrow 1, R
\ldots
9 \rightarrow 9, R

0 \rightarrow 0, R
1 \rightarrow 1, R
2 \rightarrow 2, R
\ldots
9 \rightarrow 9, R

\text{start}
\text{non-zero?}

\text{to end}
\square \rightarrow \square, L
\text{wrap zeroes}

1 \rightarrow 0, L
2 \rightarrow 1, L
\ldots
9 \rightarrow 8, L

\text{back home}
0 \rightarrow 0, L
1 \rightarrow 1, L
\ldots
9 \rightarrow 9, L

\text{done!}
\square \rightarrow \square, R
0 → 0, R
1 → 1, R
... 
9 → 9, R

0 → 0, R
1 → 1, R
2 → 2, R
... 
9 → 9, R

non-zero?

start

to end

0 → 9, L

□ → □, L

wrap zeroes

1 → 0, L
2 → 1, L
... 
9 → 8, L

1 → 0, L
2 → 1, L
... 
9 → 9, L

back home

done!

□ → □, R

0 → 0, L
0 \rightarrow 0, R
1 \rightarrow 1, R
...
9 \rightarrow 9, R

start

0 \rightarrow 0, R

non-zero?

to end

\square \rightarrow \square, L

wrap zeroes

1 \rightarrow 0, L
2 \rightarrow 1, L
...
9 \rightarrow 8, L

back home

done!

\square \rightarrow \square, R

\square \rightarrow \square, L
0 → 0, R
1 → 1, R  
...  
9 → 9, R

0 → 0, R
1 → 1, R  
2 → 2, R  
...  
9 → 9, R

0 → 9, L

Non-zero?
to end

□ → □, L

Wrap zeroes

1 → 0, L
2 → 1, L  
...  
9 → 8, L

Start

□ → □, L

Done!

□ → □, R

Back home

0 → 0, L
1 → 1, L  
...  
9 → 9, L

...
0 → 0, R
1 → 1, R
...  
9 → 9, R

非零？

to end

□ → □, L

wrap zeroes

0 → 9, L

1 → 0, L
2 → 1, L
...  
9 → 8, L

1 → 1, R
2 → 2, R
...  
9 → 9, R

0 → 0, R

启动

非零？

到结束

□ → □, L

back home

0 → 0, L

done!

□ → □, R

back home

0 → 0, L
1 → 1, L
...  
9 → 9, L

0 0 0 0 0
non-zero?

0 → 0, R
1 → 1, R
... 9 → 9, R

to end

0 → 9, L

wrap zeroes

1 → 0, L
2 → 1, L
... 9 → 8, L

start

0 → 0, R

back home

0 → 0, L

done!

□ → □, L

□ → □, R

□ → □, L

back home

0 → 0, L
1 → 1, L
... 9 → 9, L

0 0 0 0 0
0 \rightarrow 0, R
1 \rightarrow 1, R
\ldots
9 \rightarrow 9, R

\text{non-zero?}

1 \rightarrow 1, R
2 \rightarrow 2, R
\ldots
9 \rightarrow 9, R

0 \rightarrow 0, R

\text{start}

0 \rightarrow 0, L
\square \rightarrow \square, R

\text{to end}

0 \rightarrow 9, L

\text{wrap zeroes}

1 \rightarrow 0, L
2 \rightarrow 1, L
\ldots
9 \rightarrow 8, L

\text{back home}

1 \rightarrow 0, L
2 \rightarrow 1, L
\ldots
9 \rightarrow 9, L

\text{back home}

\text{done!}

\square \rightarrow \square, R
non-zero? start → 0, R
1 → 1, R
... 9 → 9, R

to end → □, L
□ → □, L
0 → 9, L
wrap zeroes

1 → 0, L
2 → 1, L
... 9 → 8, L
back home

□ → □, L
□ → □, R

0 → 0, L
0 → 0, R

back home

0 → 0, L
1 → 1, L
... 9 → 9, L

done!
The diagram shows a state transition graph with states labeled as follows:

- **Start**
- **Non-zero?**
- **To end**
- **Wrap zeroes**
- **Back home**
- **Done!**

The transitions are as follows:

- **Start** to **Non-zero?** with actions: 0 → 0, R
- **Non-zero?** to **To end** with actions: 0 → 0, L
- **To end** to **Wrap zeroes** with actions: □ → □, L
- **Wrap zeroes** to **Back home** with actions: 0 → 9, L
- **Back home** to **Done!** with actions: 0 → 0, L
- **Back home** to **Back home** with actions: 1 → 0, L
- **Back home** to **Non-zero?** with actions: 2 → 2, R
- **Non-zero?** to **To end** with actions: □ → □, L
- **To end** to **Wrap zeroes** with actions: □ → □, R
- **Wrap zeroes** to **Back home** with actions: □ → □, L
- **Back home** to **Done!** with actions: □ → □, R
- **Back home** to **Back home** with actions: 9 → 9, L

The transitions are numbered from 0 to 9, indicating the order of operations.
Turing machine subroutines

Sometimes a subroutine needs to report back some information about what happened.

Just as a function can return multiple different values, we can have subroutines to have different “done” states.

Each state can then be wired to a different state, so a Turing machine using the subroutine can control what happens next.
Putting it all together

Our goal is to build a Turing machine that, given two numbers, adds those numbers together.
def add(num1, num2):
    while num2 > 0:  # Need to make the outer loop
        increment(num1)
        decrement(num2)
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines
We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
Using our subroutines

We’ll build our new machine using our existing increment and decrement subroutines:
start

1 3 7 4 2
start

1 3 7 4 2
to 2nd num.

\[
\begin{align*}
0 & \rightarrow 0, \text{ R} \\
1 & \rightarrow 1, \text{ R} \\
\ldots & \\
9 & \rightarrow 9, \text{ R}
\end{align*}
\]
start

$0 \rightarrow 0, \text{R}$
$1 \rightarrow 1, \text{R}$
$\vdots$
$9 \rightarrow 9, \text{R}$
\textit{start} → \textit{to 2nd num.}

0 → 0, R
1 → 1, R
\ldots
9 → 9, R
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>
to 2nd num.

0 → 0, R
1 → 1, R
...
9 → 9, R
\(0 \rightarrow 0, R\)
\(1 \rightarrow 1, R\)
\(\ldots\)
\(9 \rightarrow 9, R\)
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

□ → □, R
decr.
start

to 2nd num.

$0 \rightarrow 0$, R

$1 \rightarrow 1$, R

...$

$9 \rightarrow 9$, R

deer.
start

to 2nd num.

□ → □, R

decr.

0 → 0, R
1 → 1, R
 ...
9 → 9, R
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num. □ → □, R

decr.
start

0 → 0, R
1 → 1, R
...
9 → 9, R

0 → 0, R
1 → 1, R

to 2nd num.

□ → □, R

decr.
\[\begin{align*}
0 & \rightarrow 0, \ R \\
1 & \rightarrow 1, \ R \\
\vdots
9 & \rightarrow 9, \ R
\end{align*}\]
to 2nd num. $ightarrow$ decr. 

0 $\rightarrow$ 0, R 
1 $\rightarrow$ 1, R 
... 
9 $\rightarrow$ 9, R
\begin{itemize}
\item 0 \rightarrow 0, R
\item 1 \rightarrow 1, R
\item \ldots
\item 9 \rightarrow 9, R
\end{itemize}
start → to 2nd num. (□) → (□), R → decr.

0 → 0, R
1 → 1, R
...
9 → 9, R
to 2nd num.

0 → 0, R
1 → 1, R
... 9 → 9, R
start

0 → 0, R
1 → 1, R
...
9 → 9, R

□ → □, R

decr.

done

to 1st num.

to 2nd num.

1 3 7 4 1
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

□ → □, R
decr.

done

0 → 0, L
1 → 1, L
...
9 → 9, L

to 1st num.

1 3 7 4 1
Start

to 2nd num.

□ → □, R

0 → 0, R
1 → 1, R
...
9 → 9, R

decr.

□ → □, R

done

0 → 0, L
1 → 1, L
...
9 → 9, L

to 1st num.

□ → □, L

1 3 7 4 1
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

□ → □, R

decr.

0 → 0, L
1 → 1, L
...
9 → 9, L
to 1st num.

done

□ → □, L

go home

0 → 0, L
1 → 1, L
...
9 → 9, L

go home
Start to 2nd num. □ → □, R

0 → 0, R

1 → 1, R

... 9 → 9, R

to 2nd num. □ → □, R

decr. □ → □, R
done 0 → 0, L

1 → 1, L

... 9 → 9, L
to 1st num. 0 → 0, L

1 → 1, L

... 9 → 9, L

go home 0 → 0, L

1 → 1, L

... 9 → 9, L
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

□ → □, R

decr.

0 → 0, L
1 → 1, L
...
9 → 9, L
done

to 1st num.

□ → □, L

go home

0 → 0, L
1 → 1, L
...
9 → 9, L
start

0 → 0, R
1 → 1, R
... 
9 → 9, R

to 2nd num.

□ → □, R

decr.

0 → 0, L
1 → 1, L
... 
9 → 9, L

to 1st num.

□ → □, L

done

□ → □, L

go home

0 → 0, L
1 → 1, L
... 
9 → 9, L
Start

to 2nd num.

0 → 0, R
1 → 1, R
...
9 → 9, R

to 1st num.

0 → 0, L
1 → 1, L
...
9 → 9, L

done

to 1st num.

0 → 0, L
1 → 1, L
...
9 → 9, L

ingcr.

□ → □, R

□ → □, L

go home

□ → □, R

1 3 7 4 1
The diagram illustrates a sequence of operations on a tape "go home → incr. → □ → □, R → to 2nd num. → □ → □, R → decr. → done → 0 → 0, L → 1 → 1, L → ... → 9 → 9, L → 0 → 0, L → 1 → 1, L → ... → 9 → 9, L → go home → ... → 1 → 3 → 7 → ... → 4 → 1 → ... → 0 → 0, R → 1 → 1, R → ... → 9 → 9, R".
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

□ → □, R
decr.

0 → 0, L
1 → 1, L
...
9 → 9, L
to 1st num.

□ → □, L
done

□ → □, R
go home

1 → 1, L
...
9 → 9, L

1 3 7 4 1
start → 0, R
0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num. □ → □, R

decr.

0 → 0, L
1 → 1, L
...
9 → 9, L
done
to 1st num.

□ → □, L

incr.

□ → □, R

go home

0 → 0, L
1 → 1, L
...
9 → 9, L

start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

□ → □, R
decr.

0 → 0, L
1 → 1, L
...
9 → 9, L
to 1st num.

□ → □, L

done

0 → 0, L
1 → 1, L
...
9 → 9, L

ingr.

□ → □, R
go home

1 3 8 4 1
start

0 → 0, R
1 → 1, R
...
9 → 9, R
to 2nd num.

□ → □, R
decr.

0 → 0, L
1 → 1, L
...
9 → 9, L
done
to 1st num.

□ → □, L

□ → □, R

incr.
go home

0 → 0, L
1 → 1, L
...
9 → 9, L

△

1 3 8 4 1
start → to 2nd num.
0 → 0, R
1 → 1, R
... 
9 → 9, R

done → to 1st num.
0 → 0, L
1 → 1, L
... 
9 → 9, L

decl. → incr.
□ → □, R

incr. → go home
□ → □, L

go home → start
□ → □, R
start

to 2nd num.

0 → 0, R
1 → 1, R
...
9 → 9, R
done

 incr.

□ → □, L

□ → □, R

decr.

done

to 1st num.

0 → 0, L
1 → 1, L
...
9 → 9, L

go home

0 → 0, L
1 → 1, L
...
9 → 9, L

1 3 8 4 1
to 2nd num.  \( \square \rightarrow \square, \text{R} \)  decr.

\( 0 \rightarrow 0, \text{R} \)
\( 1 \rightarrow 1, \text{R} \)
\( \cdots \)
\( 9 \rightarrow 9, \text{R} \)

done

depr.  \( \square \rightarrow \square, \text{L} \)  to 1st num.

\( 0 \rightarrow 0, \text{L} \)
\( 1 \rightarrow 1, \text{L} \)
\( \cdots \)
\( 9 \rightarrow 9, \text{L} \)

incr.  \( \square \rightarrow \square, \text{R} \)  go home

\( 0 \rightarrow 0, \text{L} \)
\( 1 \rightarrow 1, \text{L} \)
\( \cdots \)
\( 9 \rightarrow 9, \text{L} \)
\[\begin{array}{c}
0 \rightarrow 0, \text{R} \\
1 \rightarrow 1, \text{R} \\
\cdots \\
9 \rightarrow 9, \text{R}
\end{array}\]
\textbf{start} \quad \textbf{to 2nd num.} \quad \Box \rightarrow \Box, \text{R} \quad \textbf{decr.} \quad 0 \rightarrow 0, \text{L} \quad 1 \rightarrow 1, \text{L} \quad \ldots \quad 9 \rightarrow 9, \text{L} \quad \textbf{done} \quad \Box \rightarrow \Box, \text{L} \quad \textbf{to 1st num.} \quad \Box \rightarrow \Box, \text{R} \quad \textbf{go home} \quad 0 \rightarrow 0, \text{L} \quad 1 \rightarrow 1, \text{L} \quad \ldots \quad 9 \rightarrow 9, \text{L}
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

0 → 0, R
1 → 1, R
...
9 → 9, R

done

to 1st num.

0 → 0, L
1 → 1, L
...
9 → 9, L

done
start

0 → 0, R
1 → 1, R
...
9 → 9, R
to 2nd num.

\[\text{□} \rightarrow \text{□}, \ R\]
decr.

\[\text{0} \rightarrow \text{0}, \ L\]
\[\text{1} \rightarrow \text{1}, \ L\]
\[\text{9} \rightarrow \text{9}, \ L\]
done

to 1st num.

\[\text{□} \rightarrow \text{□}, \ L\]
incr.

\[\text{0} \rightarrow \text{0}, \ L\]
\[\text{1} \rightarrow \text{1}, \ L\]
\[\text{9} \rightarrow \text{9}, \ L\]
done
go home

\[\text{□} \rightarrow \text{□}, \ R\]
start

0 → 0, R
1 → 1, R
...
9 → 9, R

→ 2nd num.

done

deck

0 → 0, L
1 → 1, L
...
9 → 9, L

→ 1st num.

done

goe home

→ □, R

→ □, L

→ □, □, R
start

```
0 → 0, R
1 → 1, R
...
9 → 9, R
```

```
to 2nd num.
```

```
□ → □, R
```

```
decr.
```

```
done
```

```
to 1st num.
```

```
□ → □, L
```

```
incr.
```

```
go home
```

```
□ → □, R
```

```
done
```

```
0 → 0, L
1 → 1, L
...
9 → 9, L
```

```
0 → 0, L
1 → 1, L
...
9 → 9, L
```

```
1 3 9 4 0
```
start

```
0 → 0, R
1 → 1, R
...
9 → 9, R
```
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

to 1st num.

incr.

decr.

done

go home

0 → 0, L
1 → 1, L
...
9 → 9, L

0 → 0, L
1 → 1, L
...
9 → 9, L

1 3 9 4 0
start

0 \rightarrow 0, R
1 \rightarrow 1, R
...
9 \rightarrow 9, R

to 2nd num.

\[
\begin{array}{c}
\square \rightarrow \square, R \\
\end{array}
\]

decr.

\[
\begin{array}{c}
0 \rightarrow 0, L \\
1 \rightarrow 1, L \\
9 \rightarrow 9, L \\
\end{array}
\]

done

\[
\begin{array}{c}
\square \rightarrow \square, L \\
\end{array}
\]

to 1st num.

\[
\begin{array}{c}
0 \rightarrow 0, L \\
1 \rightarrow 1, L \\
9 \rightarrow 9, L \\
\end{array}
\]

go home

\[
\begin{array}{c}
\square \rightarrow \square, R \\
\end{array}
\]

done

1 3 9 3 9
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

decl.
done

to 1st num.

go home

incr.
done

0 → 0, L
1 → 1, L
...
9 → 9, L

0 → 0, L
1 → 1, L
...
9 → 9, L

1 3 9 3 9
0 → 0, R
1 → 1, R
...
9 → 9, R

start

0 → 0, L
1 → 1, L
...
9 → 9, L

to 2nd num.

decr.

done

to 1st num.

ingr.

go home

done

1 4 0 3 9
Many transitions later…

- Start: $0 \rightarrow 0, R$.
- $1 \rightarrow 1, R$.
- ...$9 \rightarrow 9, R$.

- Go home: $\square \rightarrow \square, L$.
- Increment: $\square \rightarrow \square, R$.

- To 2nd num.: $0 \rightarrow 0, R$.
- $1 \rightarrow 1, R$.
- ...$9 \rightarrow 9, R$.

- To 1st num.: $0 \rightarrow 0, L$.
- $1 \rightarrow 1, L$.
- ...$9 \rightarrow 9, L$.
start

0 → 0, R
1 → 1, R
...
9 → 9, R

to 2nd num.

to 1st num.

incr.

done

□ → □, R
□ → □, L

decr.

done

□ → □, R
□ → □, L

0 → 0, L
1 → 1, L
...
9 → 9, L

go home

0 → 0, L
1 → 1, L
...
9 → 9, L

1 7 8 0 1
The diagram illustrates a sequence of operations that start from an initial state, labeled as 'start'. The process involves moving between states such as 'to 2nd num.', 'decr.', 'done', 'to 1st num.', 'incr.', and 'go home'. The arrows indicate the direction of movement between these states, with operations like '□ → □, R' and '□ → □, L' showing the movement of symbols or markers.

The sequence begins with '0 → 0, R', '1 → 1, R', and continues with '9 → 9, R', followed by loops back to 'done'. The diagram includes additional states and operations, such as '0 → 0, L', '1 → 1, L', '9 → 9, L', indicating further transitions and loops.

The bottom row of the diagram shows a sequence of 17800, indicating the final state or output of the process.
0 → 0, R
1 → 1, R
... 
9 → 9, R

to 2nd num.

to 1st num.

start

□ → □, R

decr.

done

go home

□ → □, L

□ → □, R

incr.

0 → 0, L
1 → 1, L
... 
9 → 9, L

0 → 0, L
1 → 1, L
... 
9 → 9, L

1 7 8 0 0
start

to 2nd num. □ → □, R
to 1st num. □ → □, L

done 0 → 0, L

decl. 0 → 0, R

done 1 → 1, L

to 1st num. 1 → 1, R

incr. □ → □, R

done 9 → 9, L

go home 9 → 9, L

1 7 9 0 0
\begin{enumerate}
\item Start at the "start" node.
\item Move to the "to 2nd num." node.
\item If the symbol is 0, move right to 0.
\item If the symbol is 1, move right to 1.
\item For any other symbol, move right.
\item Move to the "decr." node.
\item Move right to the "incr." node.
\item Move left back to the "to 2nd num." node.
\item Move right back to the "to 1st num." node.
\item Move left back to the "done" node.
\end{enumerate}

The final state is "done" with the symbols 1, 7, 9, 0, 0.
start

0 → 0, R
1 → 1, R
... 
9 → 9, R

to 2nd num.

□ → □, R

n = 0

done!

done

done

to 1st num.

□ → □, L

□ → □, R

go home

□ → □, R

0 → 0, L
1 → 1, L
... 
9 → 9, L

0 → 0, L
1 → 1, L
... 
9 → 9, L
Using subroutines

Once you’ve built a subroutine, you can wire it into another Turing machine with something that, schematically, looks like this:

Intuitively, this corresponds to transitioning to the start state of the subroutine, then replacing the “done” state of the subroutine with the state at the end of the transition.
How does this relate to languages?

Every computation can be encoded as a language.

We could, for instance, have the language of all strings \( \{x\#y\#z \mid x + y = z\} \).

How do we check whether an input is in the language?

Add \( x \) and \( y \) and then compare it with \( z \).
Turing machines from the textbook
Many of the languages we’ve recognized with Turing machines have been context-free; we could have used a pushdown automaton to recognize them.

But Turing machines can also recognize languages that PDAs cannot.
Be sure you understand the example Turing machines in the textbook for these languages:

\[ \{w#w \mid w \in \{0, 1\}^*\} \text{, pages 166–7} \]
\[ \{0^{2^n} \mid n \geq 0\}, \text{pages 171–2} \]
\[ \{a^ib^jic^k \mid i \times j = k \text{ and } i, j, k \geq 1\}, \text{page 174} \]
\[ \{#x_1#x_2#…x_k \mid \text{each } x_i \in \{0, 1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}, \text{page 175} \]

All of these are non-context-free languages.
Acknowledgments

This lecture incorporates material from:

Nancy Ide
Keith Schwarz
Michael Sipser