MA/CSSE 474
Theory of Computation

Turing Machine Notation,
Programming,
Extensions

A Macro language for Turing Machines

(1) Define some basic machines

- Symbol writing machines

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

- Head moving machines

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

- Machines that simply halt:

  h, which simply halts (don't care whether it accepts).
  n, which halts and rejects.
  y, which halts and accepts.

Checking Inputs and Combining Machines

Next we need to describe how to:

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

To do this, we need two forms:

- $M_1\ M_2$

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

Turing Machines Macros Cont'd

Example:

$$\begin{tikzcd} >M_1 \arrow[r, "a"] \arrow[d, "b"'] & M_2 \\ M_3 \end{tikzcd}$$

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

More macros

$M_1 \overset{a}{\underset{b}{\rightrightarrows}} M_2$ becomes $M_1 \xrightarrow{a,\ b} M_2$

$M_1 \xrightarrow{\text{all elems of } \Gamma} M_2$ becomes $M_1 \longrightarrow M_2$ or $M_1 M_2$

Variables

$M_1 \xrightarrow[\text{except } a]{\text{all elems of } \Gamma} M_2$ becomes $M_1 \xrightarrow{x \leftarrow \neg a} M_2$

and x takes on the value of the current square

$M_1 \xrightarrow{a,\ b} M_2$ becomes $M_1 \xrightarrow{x \leftarrow a,\ b} M_2$

and x takes on the value of the current square

$M_1 \xrightarrow{x = y} M_2$

if $x = y$ then take the transition

e.g., $> \xrightarrow{x \leftarrow \neg \square} Rx$ if the current square is not blank, go right and copy it.

Blank/Non-blank Search Machines

$$\begin{tikzcd} > R \arrow[loop right, "\neg\square"] \end{tikzcd}$$
Find the first blank square to the right of the current square. $R_{\square}$

$$\begin{tikzcd} > L \arrow[loop right, "\neg\square"] \end{tikzcd}$$
Find the first blank square to the left of the current square. $L_{\square}$

$$\begin{tikzcd} > R \arrow[loop right, "\square"] \end{tikzcd}$$
Find the first nonblank square to the right of the current square. $R_{\neg\square}$

$$\begin{tikzcd} > L \arrow[loop right, "\square"] \end{tikzcd}$$
Find the first nonblank square to the left of the current square $L_{\neg\square}$

More Search Machines

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

$R_{a,b}$ — Find the first occurrence of `a` or `b` to the right of the current square.

$$\begin{tikzcd} L_{a,b} \arrow[r, "a"] \arrow[d, "b"] & M_1 \\ M_2 & \end{tikzcd}$$

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

$L_{x \leftarrow a,b}$ — Find the first occurrence of `a` or `b` to the left of the current square and set x to the value found.

$L_{x \leftarrow a,b}Rx$ — Find the first occurrence of `a` or `b` to the left of the current square, set x to the value found, move one square to the right, and write x (`a` or `b`).

An Example

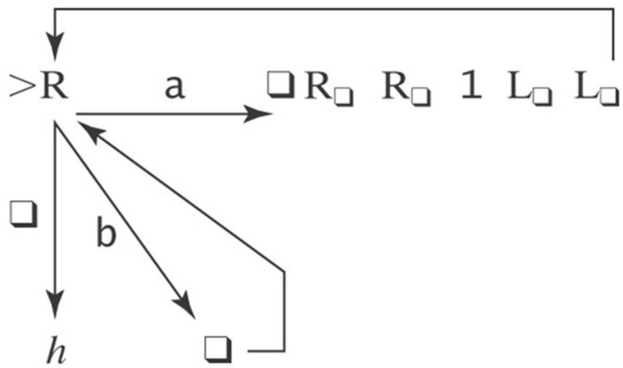
Input: $\underline{\square}w \quad w \in \{1\}^*$

Output: $\underline{\square}w^3$

Example: $\underline{\square}111\square\square\square\square\square\square\square\square\square\square\square\square\square\square$

$$M = \begin{tikzcd} >R_{1,\square} \arrow[r, "1"] \arrow[d, "\square"'] & \#R_{\square}\#R\#L_{\square} \arrow[l, bend right] \\ L \arrow[r, "\#"] \arrow[d, "\square"'] & 1 \arrow[l, bend right] \\ h & \end{tikzcd}$$

What does this machine do?



Two Useful Kinds of TMs

1. Recognize a language
2. Compute a function

Turing Machines as Language Recognizers

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

- M ***accepts*** a string w iff $(s, \underline{\square}w) \mid\!-_M^* (y, w')$ for some string w'.
- M ***rejects*** a string w iff $(s, \underline{\square}w) \mid\!-_M^* (n, w')$ for some string w'.

M ***decides*** a language $L \subseteq \Sigma^*$ iff:
For any string $w \in \Sigma^*$ it is true that:
if $w \in L$ then M accepts w, and
if $w \notin L$ then M rejects w.

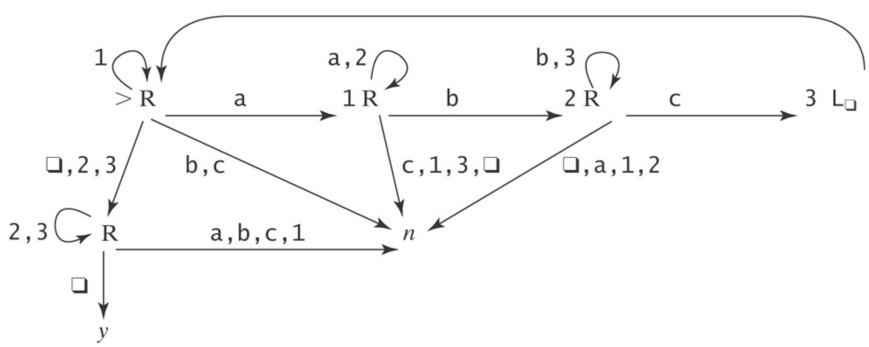
A language L is ***decidable*** iff there is a Turing machine M that decides it. In this case, we will say that L is in ***D***.

A Deciding Example

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

Example: $\underline{\square}$aabbcc❑❑❑❑❑❑❑❑❑

Example: $\underline{\square}$aaccb❑❑❑❑❑❑❑❑❑



Semideciding a Language

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

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

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

M may either:
reject or
fail to halt.

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

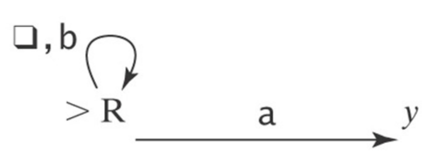
Example of Semideciding

Let L = `b`*`a`(`a` ∪ `b`)*

We can build M to semidecide L:

1. Loop
    1.1 Move one square to the right. If the character under the read head is an `a`, halt and accept.

In our macro language, M is:

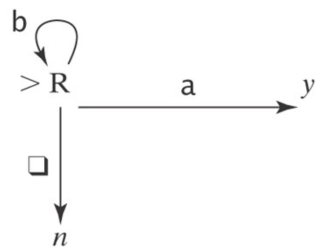


Example of Semideciding

$L = \mathtt{b}^*\mathtt{a}(\mathtt{a} \cup \mathtt{b})^*$. We can also decide L:

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

In our macro language, M is:



Computing Functions

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

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

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

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

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

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

Example of Computing a Function

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

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

Define the copy machine C:

$\underline{\square}w\square\square\square\square\square\square \rightarrow \square w\underline{\square}w\square$

Also use the $S_{\leftarrow}$ machine:

$\square u\underline{\square}w\square \rightarrow \square uw\underline{\square}$

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

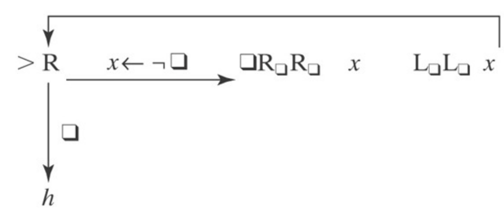
Example of Computing a Function

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

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

Define the copy machine C:

$\underline{\square}w\square\square\square\square\square\square \rightarrow \square w\underline{\square}w\square$



We skip the details in class; you can look at them later.

Then use the the $S_{\leftarrow}$ machine:

$\square u\underline{\square}w\square \rightarrow \square uw\underline{\square}$

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

Computing Numeric Functions

For any positive integer k, $value_k(n)$ returns the nonnegative integer that is encoded, base k, by the string n.

For example:

- $value_2(\texttt{101}) = 5$.
- $value_8(\texttt{101}) = 65$.

TM M computes a function f from $\mathbb{N}^m$ to $\mathbb{N}$ iff, for some k:

$$value_k(M(n_1;n_2;\ldots n_m)) = f(value_k(n_1), \ldots\ value_k(n_m)).$$

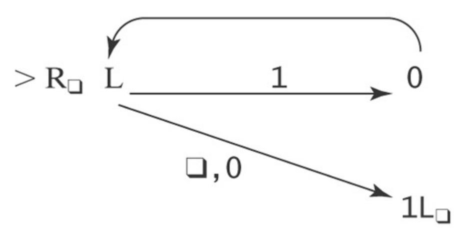
Computing Numeric Functions

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

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

Input: ❑n❑❑❑❑❑❑ Output: ❑n+1❑

❑1111❑❑❑❑ Output: ❑10000❑



Why Are We Working with Our Hands Tied Behind Our Backs?

Turing machines	Are more powerful than any of the other formalisms we have studied so far. ☺
Turing machines	Are a **lot** harder to work with than all the real computers we have available. ☹

Why bother?

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

Turing Machine Extensions

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

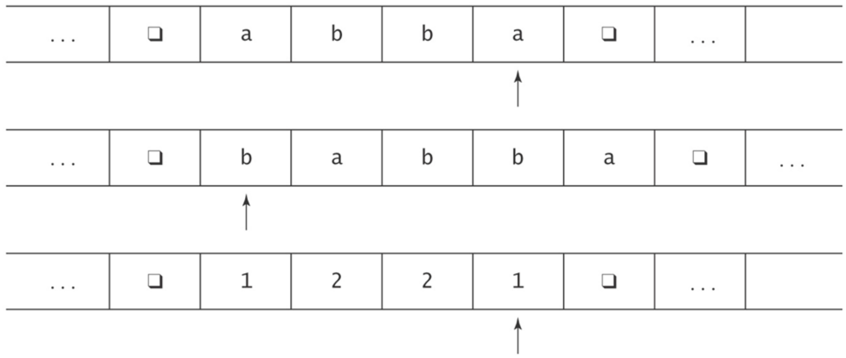
We can show that every extended machine has an equivalent basic machine.

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

Some possible extensions:

- Multiple tape TMs
- Nondeterministic TMs

Multiple Tapes



Multiple Tapes

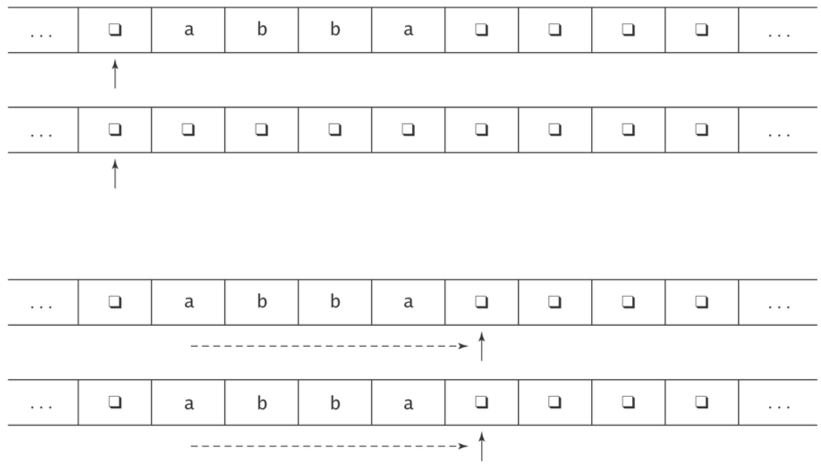
The transition function for a k-tape Turing machine:

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

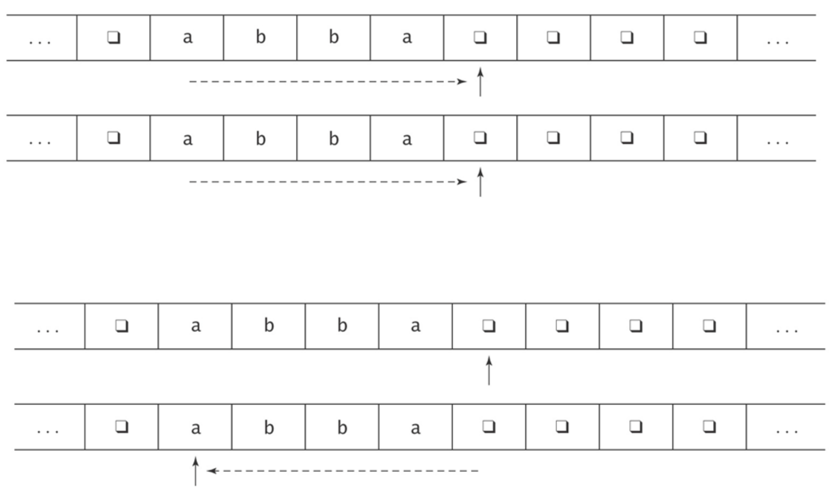
Input: as before on tape 1, others blank.
Output: as before on tape 1, others ignored.

Note: each tape head is allowed to stay where it is.

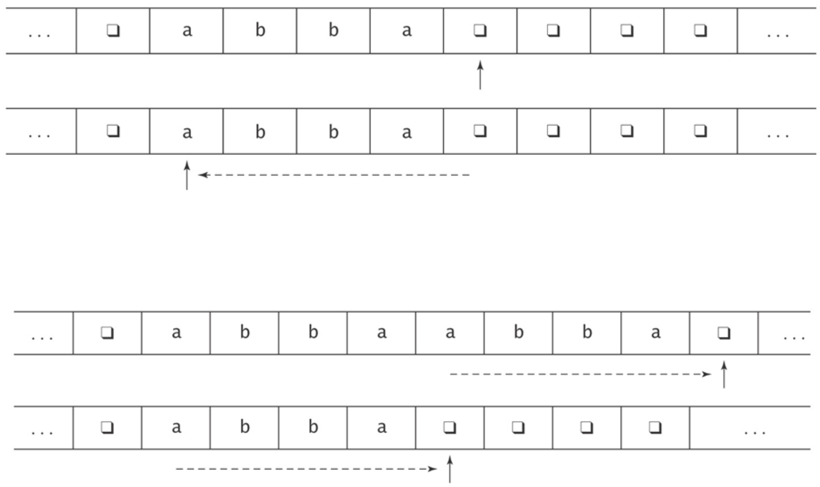
Example: Copying a String



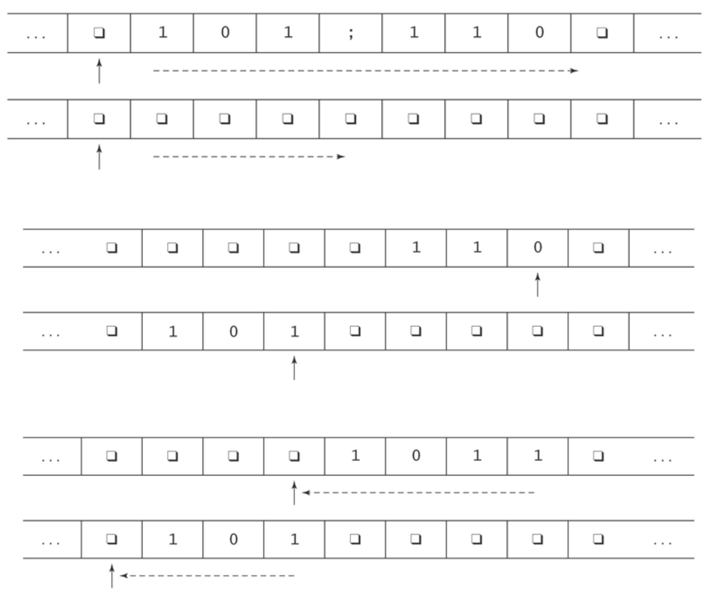
Example: Copying a String



Example: Copying a String



Another Two Tape Example: Addition



Adding Tapes Adds No Power

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

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

Proof: By construction.

The Representation

…	❑	a	b	a	a	❑	❑	❑	❑	…

…	❑	❑	❑	❑	❑	❑	❑	❑	❑	…

(a)

…	❑	❑	a	b	a	a	❑	❑	❑	…
		1	0	0	0	0	0	0		
		❑	❑	❑	❑	❑	❑			
		1	0	0	0	0	0	0		

(b)

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

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

The Operation of *M'*

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

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

How Many Steps Does *M'* Take?

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

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

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

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

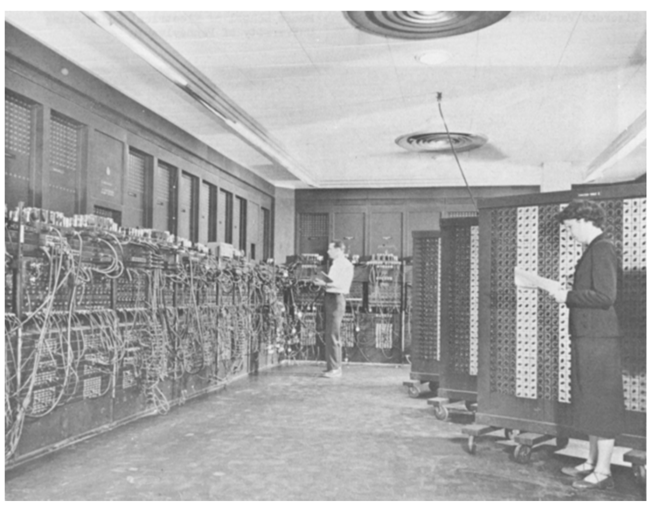
Total: $\mathcal{O}(n \cdot (|w| + n))$.
$= \mathcal{O}(n^2)$. *

* assuming that $n \geq w$

Universal Turing Machine

The Universal Turing Machine

Problem: All our machines so far are hardwired.



ENIAC - 1945

The Universal Turing Machine

Problem: All our machines so far are hardwired.

Question: Can we build a programmable TM that accepts as input:

program *input string*

executes the program, and outputs:

output string

The Universal Turing Machine

Yes, it's called the ***Universal Turing Machine***.

To define the Universal Turing Machine *U* we need to:

1. Define an encoding operation for TMs.
2. Describe the operation of *U* given input <*M*, *w*>, the encoding of:
    - a TM *M*, and
    - an input string *w*.

Encoding a Turing Machine *M*

We need to describe $M = (K, \Sigma, \Gamma, \delta, s, H)$ as a string:

- The states
- The tape alphabet
- The transitions

Encoding the States

- Let i be $\lceil \log_2(|K|) \rceil$.
- Number the states from 0 to $|K|$-1 in binary:
  - Number s, the start state, 0.
  - Number the others in any order.
- If t' is the binary number assigned to state t, then:
  - If t is the halting state y, assign it the string `y`t'.
  - If t is the halting state n, assign it the string `n`t'.
  - If t is any other state, assign it the string `q`t'.

Example of Encoding the States

Suppose *M* has 9 states.

$i = 4$

$s =$ `q0000`,

Remaining states (where *y* is 3 and *n* is 4):

`q0001`, `q0010`, `y0011`, `n0100`,
`q0101`, `q0110`, `q0111`, `q1000`

Encoding a Turing Machine *M*, Continued

The tape alphabet:

ay : $y \in \{0, 1\}^+$,
$|y| = j$, and
j is the smallest integer such that $2^j \geq |\Gamma|$.

Example: $\Sigma = \{$❑, a, b, c$\}$. $j = 2$.

❑ = a00
a = a01
b = a10
c = a11

Encoding a Turing Machine *M*, Continued

The transitions: (state, input, state, output, move)

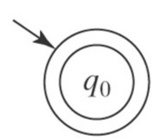
Example: (`q000`,`a000`,`q110`,`a000`,→)

Specify *s* as `q000`.

Specify *H*.

A Special Case

We will treat this as a special case:



An Encoding Example

Consider M = ({s, q, h}, {a, b, c}, {❑, a, b, c}, δ, s, {h}):

state	symbol	δ
s	❑	(q,❑,□ →)
s	a	(s,b,→)
s	b	(q,a, ←)
s	c	(q,b, ←)
q	❑	(s,a,□ →)
q	a	(q,b,→)
q	b	(q,b, ←)
q	c	(h,a, ←)

state/symbol	representation
s	q00
q	q01
h	h10
❑	a00
a	a01
b	a10
c	a11

<M> = (q00,a00,q01,a00,→), (q00,a01,q00,a10,→),
(q00,a10,q01,a01, ←), (q00,a11,q01,a10,←),
(q01,a00,q00,a01,→), (q01,a01,q01,a10,→),
(q01,a10,q01,a11,←), (q01,a11,h10,a01,←)

Enumerating Turing Machines

Theorem: There exists an infinite lexicographic enumeration of:

(a) All syntactically valid TMs.

(b) All syntactically valid TMs with specific input alphabet Σ.

(c) All syntactically valid TMs with specific input alphabet Σ and specific tape alphabet Γ.

Enumerating Turing Machines

Proof: Fix Σ = {`(`, `)`, `a`, `q`, `y`, `n`, `0`, `1`, comma, $\rightarrow$, $\leftarrow$}, ordered as listed. Then:

1. Lexicographically enumerate the strings in Σ^*.
2. As each string *s* is generated, check to see whether it is a syntactically valid Turing machine description. If it is, output it.

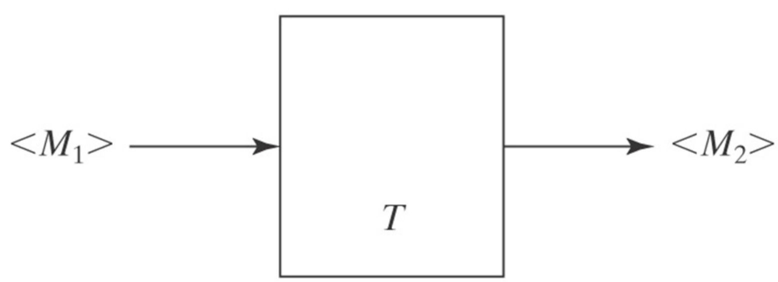
To restrict the enumeration to symbols in sets Σ and Γ, check, in step 2, that only alphabets of the appropriate sizes are allowed.

We can now talk about the i^{th} Turing machine.

Another Win of Encoding

One big win of defining a way to encode any Turing machine *M*:

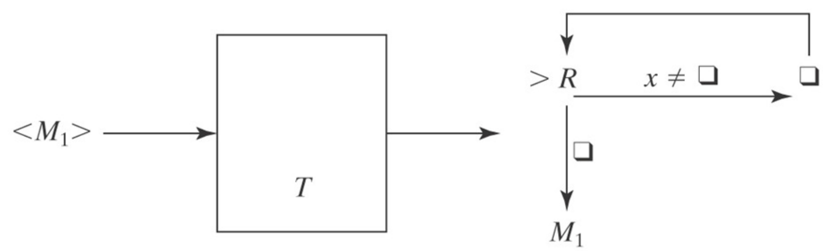
- We can talk about operations on programs (TMs).



Example of a Transforming TM *T*:

Input: a TM M_1 that reads its input tape and performs some operation *P* on it.

Output: a TM M_2 that performs *P* on an empty input tape.



Encoding Multiple Inputs

Let:

$<x_1, x_2, \ldots x_n>$

mean a single string that encodes the sequence of individual values:

$x_1, x_2, \ldots x_n$.

The Specification of the Universal TM

On input <*M*, *w*>, *U* must:

- Halt iff *M* halts on *w*.
- If *M* is a deciding or semideciding machine, then:
  - If *M* accepts, accept.
  - If *M* rejects, reject.
- If *M* computes a function, then *U*(<*M*, *w*>) must equal *M*(*w*).

How *U* Works

U will use 3 tapes:

- Tape 1: *M*'s tape.
- Tape 2: <*M*>, the "program" that *U* is running.
- Tape 3: *M*'s state.

The Universal TM

	<M---------------------------------M,				w--------------------w>				
	1	0	0	0	0	0	0		
❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
	1	0	0	0	0	0	0		
	❑	❑	❑	❑	❑	❑	❑		
	1	0	0	0	0	0	0		

Initialization of *U*:

1. Copy <*M*> onto tape 2.
2. Look at <*M*>, figure out what *i* is, and write the encoding of state *s* on tape 3.

After initialization:

	❑	❑	❑	❑	<w--------------------w>				
	0	0	0	0	1	0	0		
❑	<M---------------------------------M>				❑	❑	❑	❑	❑
	1	0	0	0	0	0	0		
	q	0	0	0	❑	❑	❑		
	1	❑	❑	❑	❑	❑	❑		

The Operation of *U*

	❑	❑	❑	❑	<w--------------------w>				
	0	0	0	0	1	0	0		
❑	<M---------------------------------M>				❑	❑	❑	❑	❑
	1	0	0	0	0	0	0		
	q	0	0	0	❑	❑	❑		
	1	❑	❑	❑	❑	❑	❑		

Simulate the steps of *M* :

1. Until *M* would halt do:
    1.1 Scan tape 2 for a quintuple that matches the current state, input pair.
    1.2 Perform the associated action, by changing tapes 1 and 3. If necessary, extend the tape.
    1.3 If no matching quintuple found, halt. Else loop.
2. Report the same result *M* would report.

How long does *U* take?

If A Universal Machine is Such a Good Idea …

Could we define a Universal Finite State Machine?

Such a FSM would accept the language:

$L = \{<F, w> : F \text{ is a FSM, and } w \in L(F)\}$