Turing Machine Extensions

Read K & S 4.3.1, 4.4.
Do Homework 19.

Turing Machine Definitions

An alternative definition of a Turing machine:

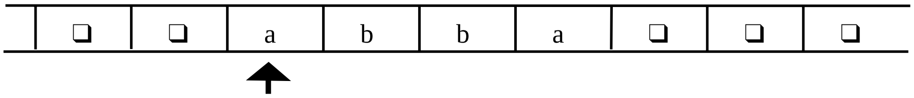
$(K, \Sigma, \Gamma, \delta, s, H)$:

Γ is a finite set of allowable tape symbols. One of these is ❑.

Σ is a subset of Γ not including ❑, the input symbols.

δ is a function from:

$K \times \Gamma$ to $K \times (\Gamma - \{❑\}) \times \{\leftarrow, \rightarrow\}$
state, tape symbol, L or R



Example transition: $((s, a), (s, b, \rightarrow))$

Do these Differences Matter?

Remember the goal:

Define a device that is:

- powerful enough to describe all computable things,
- simple enough that we can reason formally about it

Both definitions are simple enough to work with, although details may make specific arguments easier or harder.

But, do they differ in their power?

Answer: No.

Consider the differences:

- One way or two way infinite tape: we're about to show that we can simulate two way infinite with ours.
- Rewrite and move at the same time: just affects (linearly) the number of moves it takes to solve a problem.

Turing Machine Extensions

In fact, there are lots of extensions we can make to our basic Turing machine model. They may make it easier to write Turing machine programs, but none of them increase the power of the Turing machine because:

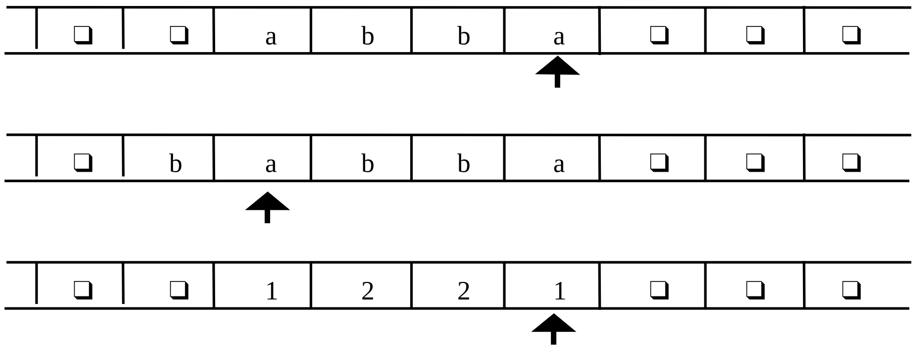
We can show that every extended machine has an equivalent basic machine.

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 tapes
- Two-way infinite tape
- Multiple read heads
- Two dimensional “sheet” instead of a tape
- Random access machine
- Nondeterministic machine

Multiple Tapes



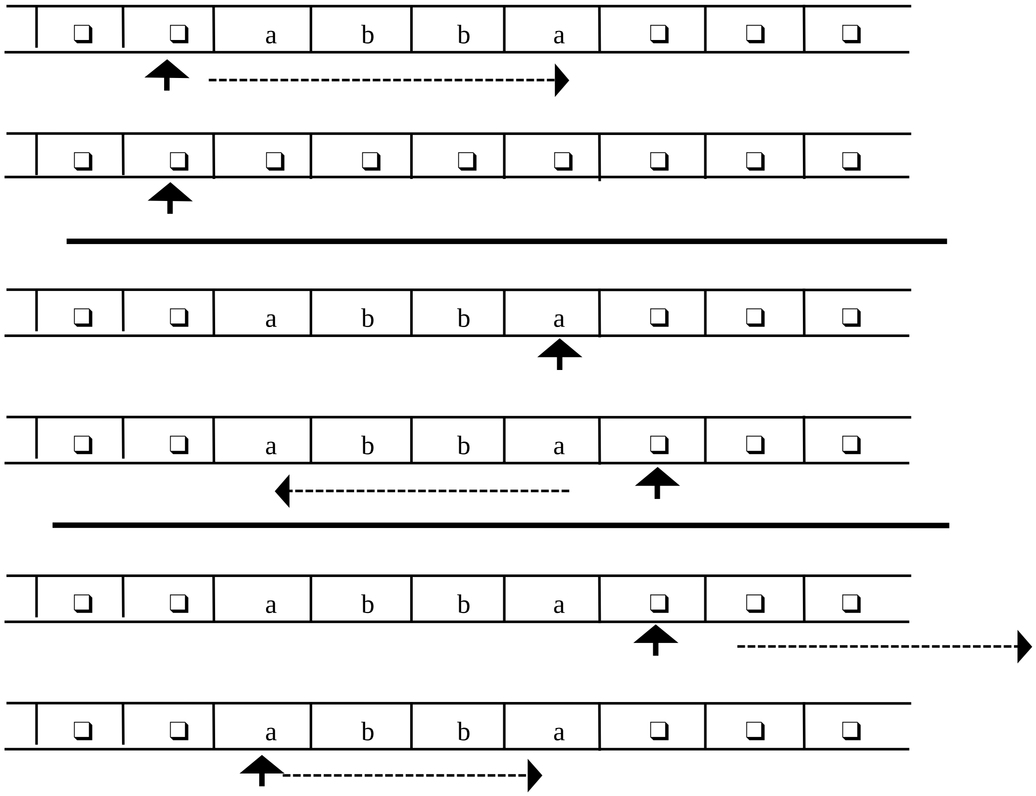
The transition function for a k-tape Turing machine:

$$\begin{array}{llll}
((K\text{-}H) & , \Sigma_1 & \text{to} & (K,\ \Sigma_{1'} \cup \{\leftarrow, \rightarrow\} \\
 & , \Sigma_2 & & , \Sigma_{2'} \cup \{\leftarrow, \rightarrow\} \\
 & , \ . & & , \ . \\
 & , \ . & & , \ . \\
 & , \ \Sigma_k) & & , \Sigma_{k'} \cup \{\leftarrow, \rightarrow\})
\end{array}$$

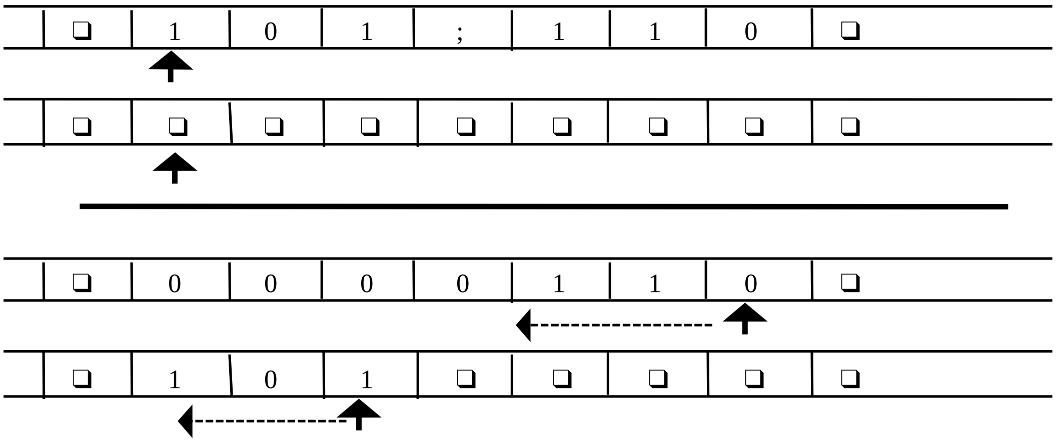
Input: input as before on tape 1, others blank
Output: output as before on tape 1, others ignored

An Example of a Two Tape Machine

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 ≥ 1. Then there is a standard Turing machine M' where $\Sigma \subseteq \Sigma'$, and such that:

- For any input string x, M on input x halts with output y on the first tape iff M' on input x halts at the same halting state and with the same output on its tape.
- If, on input x, M halts after t steps, then M' halts after a number of steps which is O(t · (|x| + t)).

Proof: By construction

◊	◊	❑	a	b	a	❑	❑	❑	❑
	0	0	1	0	0	0	0		
	◊	a	b	b	a	b	a		
	0	1	0	0	0	0	0		

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

e.g., ◊, (◊, 0, ◊, 0), (❑, 0, a, 1)

The Operation of M'

◊	◊	❑	a	b	a	❑	❑	❑	❑
	0	0	1	0	0	0	0		
	◊	a	b	b	a	b	a		
	0	1	0	0	0	0	0		

1. Set up the multitrack tape:
   1) Shift input one square to right, then set up each square appropriately.
2. Simulate the computation of M until (if) M would halt: (start each step to the right of the divided tape)
   1) Scan left and store in the state the k-tuple of characters under the read heads. Move back right.
   2) Scan left and update each track as required by the transitions of M. Move back right.
      i) If necessary, subdivide a new square into tracks.
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: x be the input string, and
t be the number of steps it takes M to execute.

Step 1 (initialization) O(|x|)

Step 2 (computation)

Number of passes = t

Work at each pass: 2.1 = 2 · (length of tape)
= 2 · (|x| + 2 + t)
2.2 = 2 · (|x| + 2 + t)

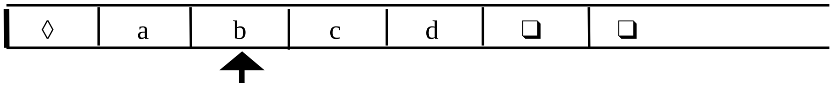
Total = O(t · (|x| + t))

Step 3 (clean up) O(length of tape)

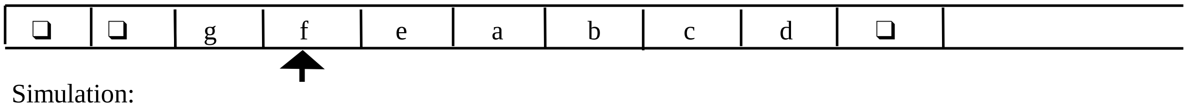
Total = O(t · (|x| + t))

Two-Way Infinite Tape

Our current definition:



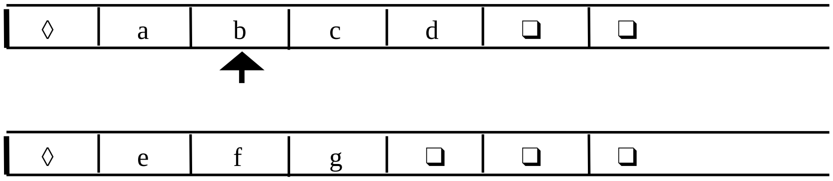
Proposed definition:



Simulation:

Track 1

Track 2



Simulating a PDA

The components of a PDA:

- Finite state controller
- Input tape
- Stack

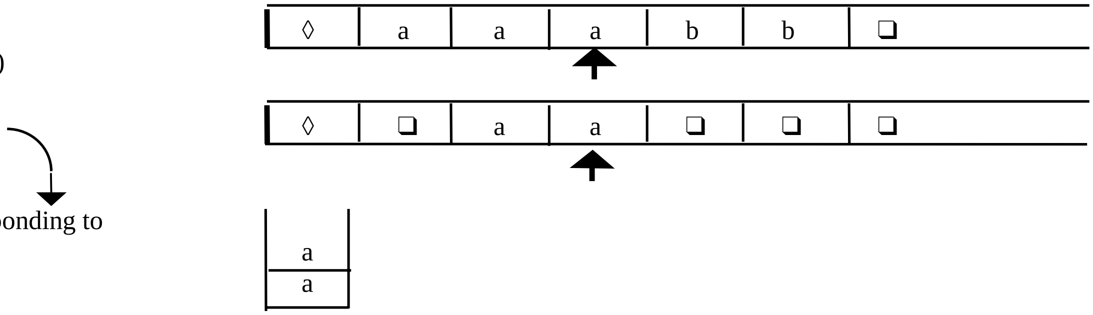
The simulation:

- Finite state controller:
- Input tape:
- Stack:

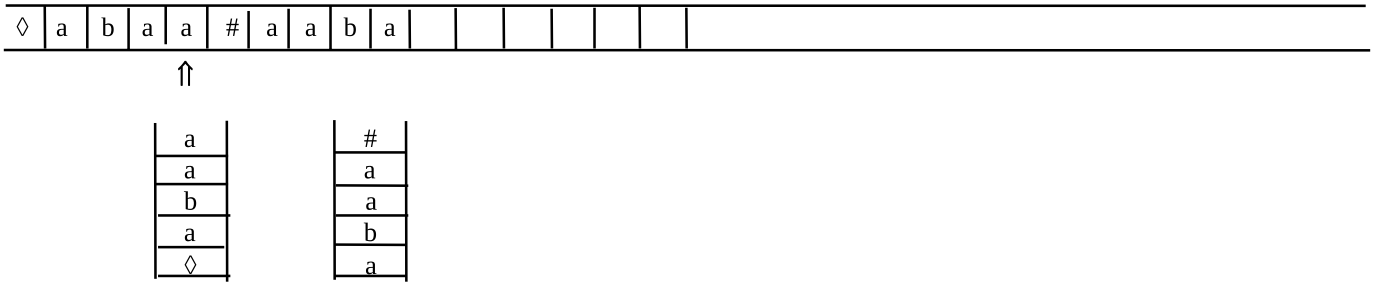
Track 1
(Input)

Track 2

Corresponding to



Simulating a Turing Machine with a PDA with Two Stacks



Random Access Turing Machines

A random access Turing machine has:

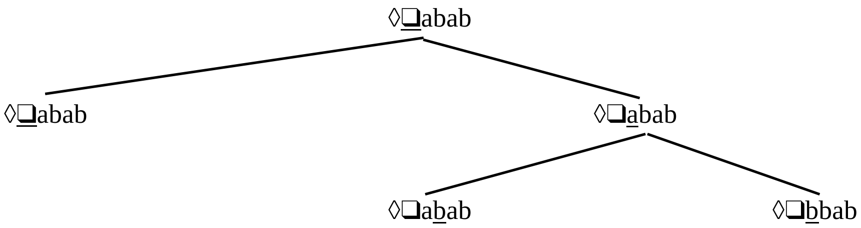
- a fixed number of registers
- a finite length program, composed of instructions with operators such as read, write, load, store, add, sub, jump
- a tape
- a program counter

Theorem: Standard Turing machines and random access Turing machines compute the same things. Furthermore, the number of steps it takes a standard machine is bounded by a polynomial in the number of steps it takes a random access machine.

Nondeterministic Turing Machines

A **nondeterministic** Turing machine is a quintuple $(K, \Sigma, \Delta, s, H)$
where K, Σ, s, and H are as for standard Turing machines, and Δ is a ***subset*** of

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



What does it mean for a nondeterministic Turing machine to compute something?

- Semidecides - at least one halts.
- Decides - ?
- Computes - ?

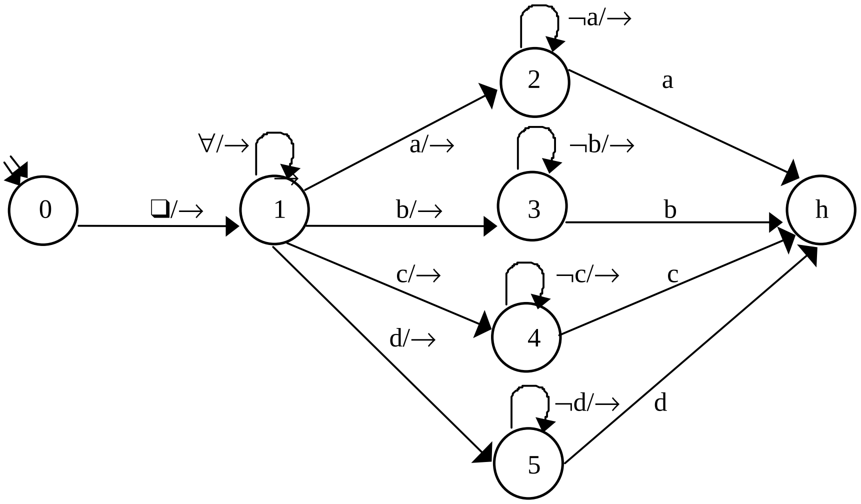
Nondeterministic Semideciding

Let M = (K, Σ, Δ, s, H) be a nondeterministic Turing machine. We say that M **accepts** an input
$w \in (\Sigma - \{◊, ❑\})^*$ iff
(s, ◊❑w) yields a least one accepting configuration.

We say that M **semidecides** a language
$L \subseteq (\Sigma - \{◊, ❑\})^*$ iff
for all $w \in (\Sigma - \{◊, ❑\})^*$:
$w \in L$ iff
(s, ◊❑w) yields a least one halting configuration.

An Example

L = {w ∈ {a, b, c, d}* : there are two of at least one letter}



Nondeterministic Deciding and Computing

M **decides** a language L if, for all $w \in (\Sigma - \{◊, ❑\})^*$:

1. all of M's computations on w halt, and
2. $w \in L$ iff ***at least one*** of M's computations accepts.

M **computes** a function f if, for all $w \in (\Sigma - \{◊, ❑\})^*$:

1. all of M's computations halt, and
2. ***all*** of M's computations result in f(w)

Note that all of M's computations halt iff:

There is a natural number N, depending on M and w, such that there is no configuration C satisfying

$(s, ◊\underline{❑}w) \vdash_M^N C$.

An Example of Nondeterministic Deciding

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

M decides L by doing the following on input w:

1. Nondeterministically choose two binary numbers $1 < p, q$, where $|p|$ and $|q| \leq |w|$, and write them on the tape, after w, separated by ;.

   ◊$\underline{❑}$110011;111;1111❑❑

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

   ◊$\underline{❑}$110011;1011111❑❑

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

Equivalence of Deterministic and Nondeterministic Turing Machines

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

Note that while nondeterminism doesn't change the computational power of a Turing Machine, it can exponentially increase its speed!

Proof: (by construction)
For semideciding: We build M', which runs through all possible computations of M. If one of them halts, M' halts

Recall the way we did this for FSMs: simulate being in a combination of states.

Will this work here?

What about: Try path 1. If it accepts, accept. Else
Try path 2. If it accepts, accept. Else
•
•

The Construction

At any point in the operation of a nondeterministic machine M, the maximum number of branches is

$$r = \underset{\text{states}}{|K|} \cdot \underset{\text{actions}}{(|\Sigma| + 2)}$$

So imagine a table:

	1	2	3		r
(q1,σ1)	(p-,σ-)	(p-,σ-)	(p-,σ-)	(p-,σ-)	(p-,σ-)
(q1,σ2)	(p-,σ-)	(p-,σ-)	(p-,σ-)	(p-,σ-)	(p-,σ-)
(q1,σn)					
(q2,σ1)					
(q|K|,σn)					

Note that if, in some configuration, there are not r different legal things to do, then some of the entries on that row will repeat.

The Construction, Continued

M_d: (suppose r = 6)

Tape 1: Input

Tape 2: 1 3 2 6 5 4 3 6

M_d chooses its 1st move from column 1
M_d chooses its 2nd move from column 3
M_d chooses its 3rd move from column 2

-
-

until there are no more numbers on Tape 2

M_d either:

- discovers that M would accept, or
- comes to the end of Tape 2.

In either case, it halts.

The Construction, Continued

M' (the machine that simulates M):

Tape 1: Input

Tape 2: Copy of Input

Tape 3: 1 3 2 6 5 4 3 6

M_d

Steps of M':

```
write ε on Tape 3
until M_d accepts do
    (1) copy Input from Tape 1 to Tape 2
    (2) run M_d
    (3) if M_d accepts, exit
    (4) otherwise, generate lexicographically next string on Tape 3.
```

Pass	1	2	3		7	8	9		
Tape3	ε	1	2	…	6	11	12	…	2635

Nondeterministic Algorithms

Other Turing Machine Extensions

Multiple heads (on one tape)

Emulation strategy: Use tracks to keep track of tape heads. (See book)

Multiple tapes, multiple heads

Emulation strategy: Use tracks to keep track of tapes and tape heads.

Two-dimensional semi-infinite "tape"

Emulation strategy: Use diagonal enumeration of two-dimensional grid. Use second tape to help you keep track of where the tape head is. (See book)

Two-dimensional infinite "tape" (really a sheet)

Emulation strategy: Use modified diagonal enumeration as with the semi-infinite case.

What About Turing Machine Restrictions?

Can we make Turing machines even more limited and still get all the power?

Example:

We allow a tape alphabet of arbitrary size. What happens if we limit it to:

- One character?
- Two characters?
- Three characters?

Problem Encoding, TM Encoding, and the Universal TM

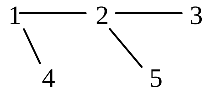
Read K & S 5.1 & 5.2.

Encoding a Problem as a Language

A Turing Machines deciding a language is analogous to the TM solving a **decision problem**.

Problem: Is the number n prime?
Instance of the problem: Is the number 9 prime?
Encoding of the problem, $\langle n \rangle$: n as a binary number. Example: 1001

Problem: Is an undirected graph G connected?
Instance of the problem: Is the following graph connected?



Encoding of the problem, $\langle G \rangle$:

1) |V| as a binary number
2) A list of edges represented by pairs of binary numbers being the vertex numbers that the edge connects
3) All such binary numbers are separated by "/".

Example: 101/1/10/10/11/1/100/10/101

Problem View vs. Language View

Problem View: It is *unsolvable* whether a Turing Machine halts on a given input. This is called the **Halting Problem**.

Language View: Let H = {$\langle M, w \rangle$: TM M halts on input string w}
H is recursively enumerable but not recursive.

The Universal Turing Machine

Problem: All our machines so far are hardwired.

Question: Does it make sense to talk about a programmable Turing machine that accepts as input
program input string
executes the program, and outputs
output string

Yes, it's called the Universal Turing Machine.

Notice that the Universal Turing machine semidecides H = {$\langle M, w \rangle$: TM M halts on input string w} = L(U).

To define the Universal Turing Machine U we need to do two things:

1. Define an encoding operation for Turing machines.
2. Describe the operation of U given an input tape containing two inputs:
   - encoded Turing machine M,
   - encoded input string to be given to M.

Encoding a Turing Machine M

We need to describe M = (K, Σ, δ, s, H) as a string. To do this we must:

1. Encode δ
2. Specify s.
3. Specify H (and y and n, if applicable)

1. To encode δ, we need to:
    1. Encode the states
    2. Encode the tape alphabet
    3. Specify the transitions

1.1 Encode the states as

qs : $s \in \{0, 1\}^+$ and
$|s| = i$ and
i is the smallest integer such that $2^i \geq |K|$

Example: 9 states i = 4
s = q0000,
remaining states: q0001, q0010, q0011,
q0100, q0101, q0110, q0111, q1000

Encoding a Turing Machine M, Continued

1.2 Encode the tape alphabet as

as : $s \in \{0, 1\}^+$ and
$|s| = j$ and
j is the smallest integer such that $2^j \geq |\Sigma| + 2$ (the + 2 allows for ← and →)

Example: Σ = {◊, ❑, a, b} j = 3

❑ =	a000
◊ =	a001
← =	a010
→ =	a011
a =	a100
b =	a101

Encoding a Turing Machine M, Continued

1.3 Specify transitions as (state, input, state, output)

Example: (q00,a000,q11,a000)

2. Specify s as $q0^i$

3. Specify H:

- States with no transitions out are in H.
- If M decides a language, then H = {y, n}, and we will adopt the convention that y is the lexicographically smaller of the two states.
  y = q010 n = q011

Encoding Input Strings

We encode input strings to a machine M using the same character encoding we use for M.
For example, suppose that we are using the following encoding for symbols in M:

symbol	representation
❑	a000
◊	a001
←	a010
→	a011
a	a100

Then we would represent the string s = ◊aa❑a as "s" = ⟨s⟩ = a001a100a100a000a100

An Encoding Example

Consider M = ({s, q, h}, {❑, ◊,a}, δ, s, {h}), where δ =

state	symbol	δ
s	a	(q, ❑)
s	❑	(h, ❑)
s	◊	(s, →)
q	a	(s, a)
q	❑	(s, →)
q	◊	(q, →)

state/symbol	representation
s	q00
q	q01
h	q11
❑	a000
◊	a001
←	a010
→	a011
a	a100

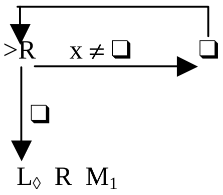
The representation of M, denoted, "M", ⟨M⟩, or sometimes ρ(M) =
(q00,a100,q01,a000), (q00,a000,q11,a000), (q00,a001,q00,a011),
(q01,a100,q00,a100), (q01,a000,q00,a011), (q01,a001,q01,a011)

Another Win of Encoding

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

- It will make sense to talk about operations on programs (Turing machines). In other words, we can talk about some Turing machine T that takes another Turing machine (say M_1) as input and transforms it into a different machine (say M_2) that performs some different, but possibly related task.

Example of a transforming TM T:
Input: a machine M_1 that reads its input tape and performs some operation P on it.
Output: a machine M_2 that performs P on an empty input tape:



The Universal Turing Machine

The specification for U:

U("M" "w") = "M(w)"

◊	"M -------	------------	------------- M"		"w-------	-----------	------w"	❑	❑
	1	0	0	0	0	0	0		
	❑	❑	❑	❑	❑	❑	❑		
	❑	❑	❑	❑	❑	❑	❑		
	❑	❑	❑	❑	❑	❑	❑□		
	❑	❑	❑	❑	❑	❑	❑		

◊	"◊	❑"	"w---------	----------w"		❑	❑	❑	❑
	1	0	0	0	0	0	0		
	"M -------	------------	------------ M"		❑	❑	❑		
	1	0	0	0	0	0	0		
	q	0	0	0	❑	❑	❑		
	1	❑	❑	❑	❑	❑	❑		

Initialization of U:

1. Copy "M" onto tape 2
2. Insert "◊❑" at the left edge of tape 1, then shift w over.
3. Look at "M", figure out what i is, and write the encoding of state s on tape 3.

The Operation of U

◊	a	0	0	1	a	0	0	❑	❑
	1	0	0	0	0	0	0		
	"M ----	--------------	--------------	M"	❑	❑	❑		
	1	0	0	0	0	0	0		
	q	0	0	0	❑	❑	❑		
	1	❑	❑	❑	❑	❑	❑		

Simulate the steps of M:

1. Start with the heads:
   tape 1: the a of the character being scanned,
   tape 2: far left
   tape 3: far left

2. Simulate one step:
   1. Scan tape 2 for a quadruple that matches current state, input pair.
   2. Perform the associated action, by changing tapes 1 and 3. If necessary, extend the tape.
   3. If no quadruple found, halt. Else go back to 2.

An Example

Tape 1: a001a000a100a100a000a100
◊ ❑ a a ❑ a
↑

Tape 2: (q00,a000,q11,a000), (q00,a001,q00,a011),
(q00,a100,q01,a000), (q01,a000,q00,a011),
(q01,a001,q01,a011), (q01,a100,q00,a100)

Tape 3: q01
↑

Result of simulating the next step:

Tape 1: a001a000a100a100a000a100
◊ ❑ a a ❑ a
↑

Tape 3: q00
↑

If A Universal Machine is Such a Good Idea …

Could we define a Universal Finite State Machine?

Such a FSM would accept the language
$L = \{"F"\ "w" : \quad F \text{ is a finite state machine, and } w \in L(F)\ \}$