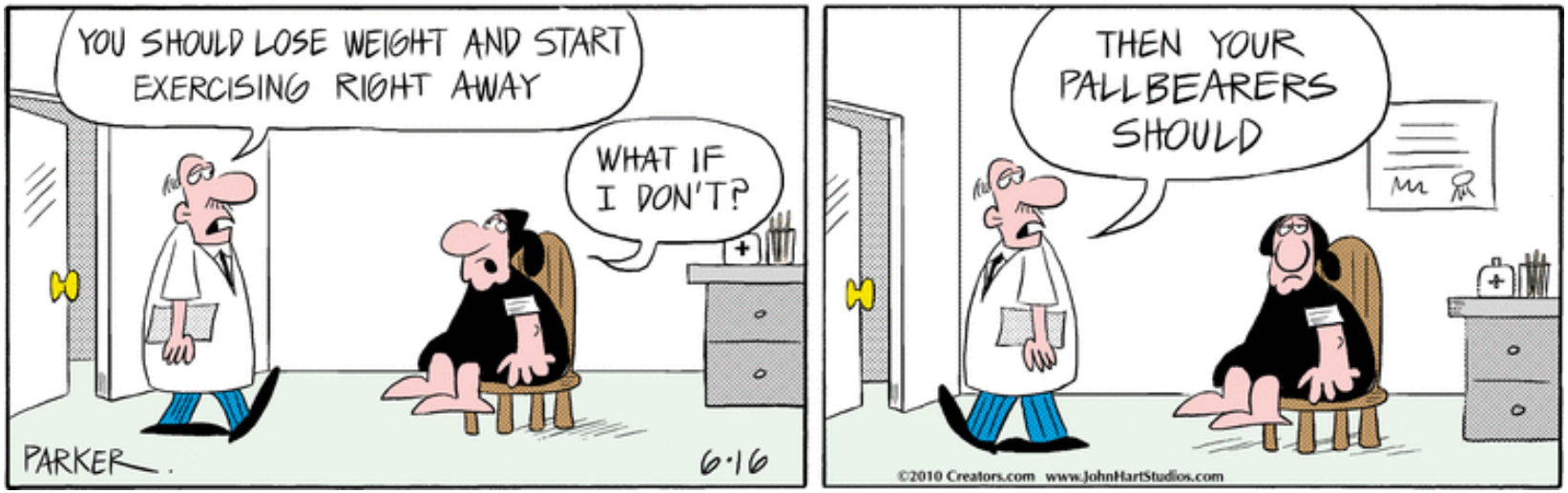
MA/CSSE 474
Theory of Computation

TM Variations
Encoding a TM
(Universal Turing Machine)

Your Questions?

- Previous class days' material
- Reading Assignments
- HW 14b problems
- Tuesday's exam
- Anything else



Two Flavors 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 $(s, \underline{\text{Æ}}\, w) \vdash_M^* (y, w')$ for some string *w′* (that includes an underlined character).
- *M* ***rejects*** a string *w* iff $(s, \underline{\text{Æ}}\, w) \vdash_M^* (n, w')$ 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*.

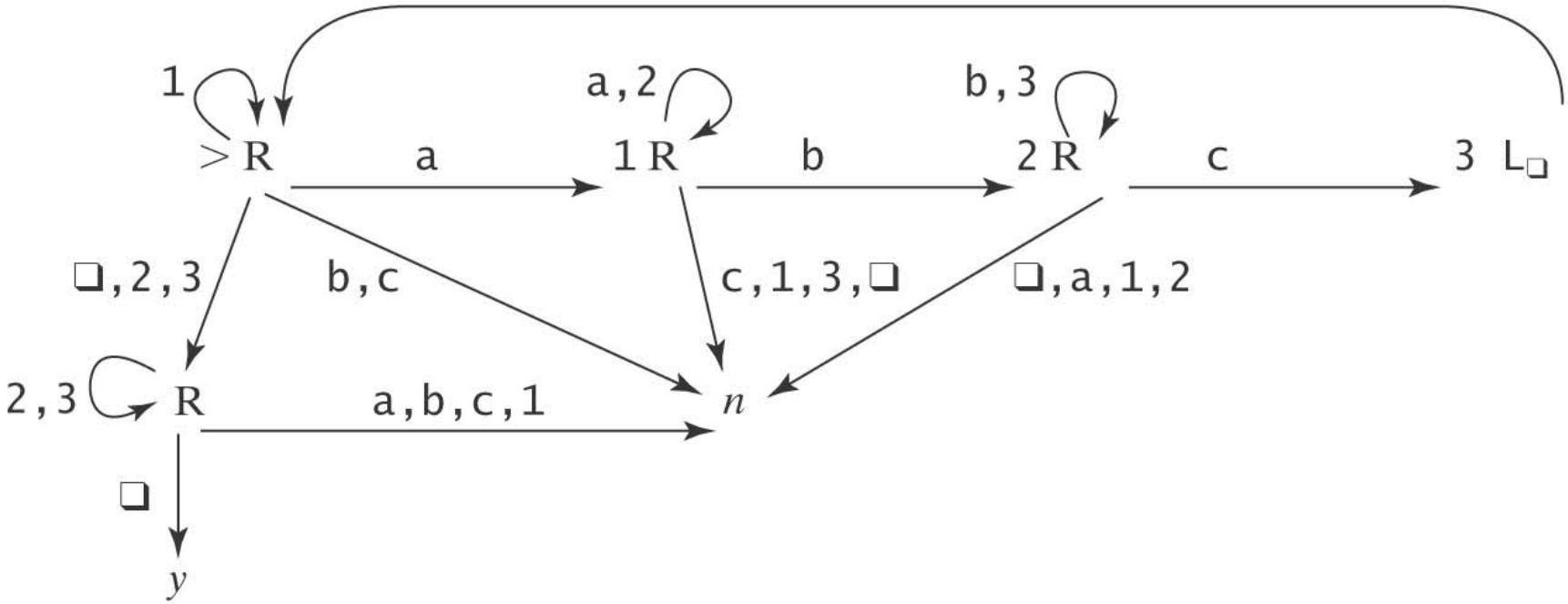
A language *L* is ***decidable*** iff there is a Turing machine *M* that decides it. In this case, we will say that *L* is in ***D***.

A Deciding Example

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

Example: Æ aabbccÆ Æ Æ Æ Æ Æ Æ Æ Æ

Example: Æ aaccbÆ Æ Æ Æ Æ Æ Æ Æ Æ



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.

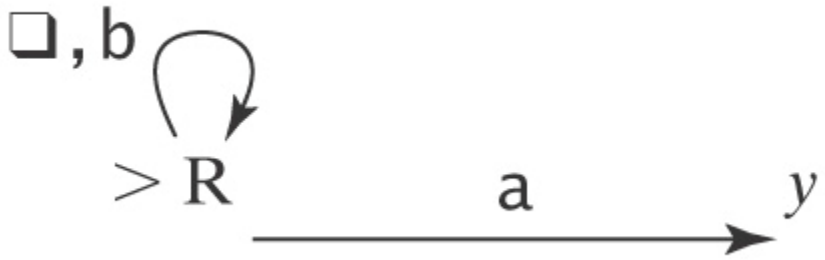
Example of Semideciding

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:



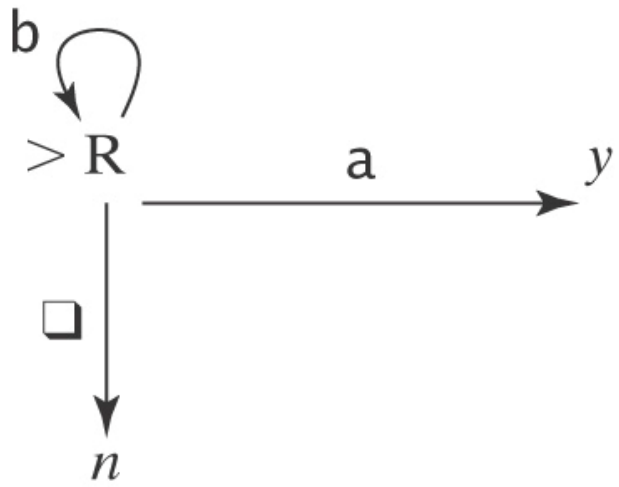
Example of Deciding the same Language

L = b*a(a ∪ 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 Æ , halt and reject.

In our macro language, *M* is:



TM that Computes a Function

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

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

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.

Note that this is different than our common use of *recursive.*

Notice that the TM's function computes with strings (Σ^* to Σ'^*), not directly with numbers.

Example of Computing a Function

Let Σ = {a, b}. Let $f(w) = ww$.

Input: $\underline{Æ}\,wÆ\,Æ\,Æ\,Æ\,Æ\,Æ$ Output: $\underline{Æ}\,wwÆ$

Define the copy machine C:

$\underline{Æ}\,wÆ\,Æ\,Æ\,Æ\,Æ\,Æ \rightarrow Æ\,w\underline{Æ}\,wÆ$

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

$Æ\,u\underline{Æ}\,wÆ \rightarrow Æ\,uw\underline{Æ}$

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

More details next slide

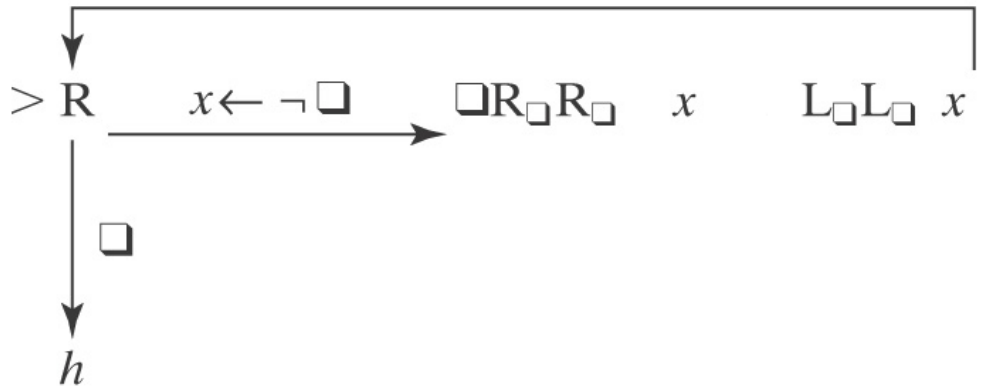
Example of Computing a Function

Let Σ = {a, b}. Let $f(w) = ww$.

Input: $\underline{Æ}\,wÆ\,Æ\,Æ\,Æ\,Æ\,Æ$ Output: $\underline{Æ}\,wwÆ$

Define **the copy machine C**:

$\underline{Æ}\,wÆ\,Æ\,Æ\,Æ\,Æ\,Æ \rightarrow Æ\,w\underline{Æ}\,wÆ$



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

$Æ\,u\underline{Æ}\,wÆ \rightarrow Æ\,uw\underline{Æ}$

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

Computing Numeric Functions

For any positive integer k, $\boldsymbol{value_k(n)}$ 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:

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

Note that the semicolon serves to separate the representations of the arguments

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 that are available to us.

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 *everything* 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.

Multiple tracks

Multiple tapes

Non-deterministic

TURING MACHINE VARIATIONS

Turing Machine Variations

There are many extensions we might like to make to our basic Turing machine model. We can do this 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:

- Multi-track tape.
- Multi-tape TM
- Nondeterministic TM

Recall that *equivalent* means "accepts the same language," or "computes the same function."

Multiple-track tape

We would like to be able to have TM with a multiple-track tape. On an n-track tape, Track i has input alphabet Σ_i and tape alphabet Γ_i.

1st track	….	1	B	B	B	B	B	B	B	..
2nd track	….	B	B	1	B	B	B	B	B	..
3rd track	..	B	B	B	B	1	B	B	B	..
4th track	..	B	B	B	B	B	B	1	B	.
5th track	..	a	b	c	d	e	f	g	h	.

Multiple-track tape

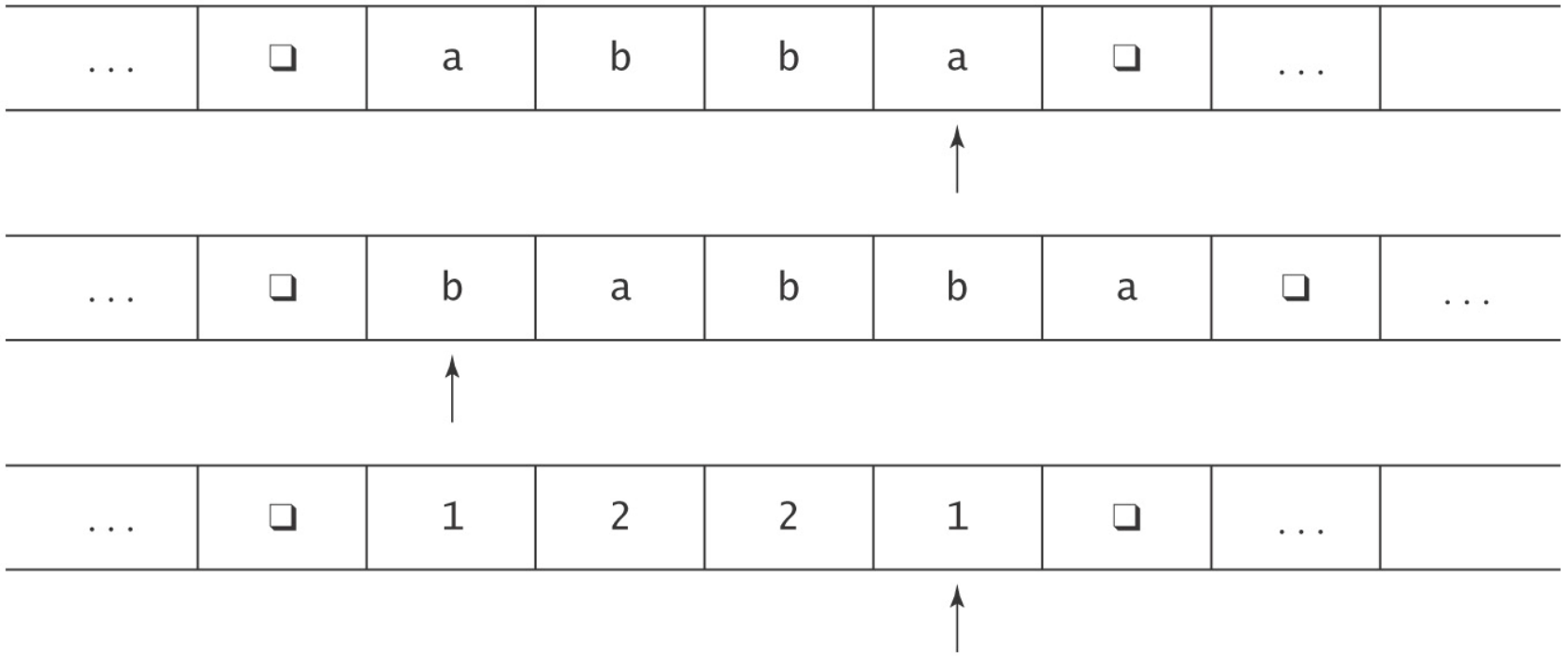
We would like to be able to have a TM with a multiple-track tape. On an n-track tape, Track i has input alphabet Σ_i and tape alphabet Γ_i.

We can simulate this with an ordinary TM.

A transition is based on the current state and the combination of all of the symbols on all of the tracks of the current "column".

Then Γ is the set of n-tuples of the form $[\gamma_1, \ldots, \gamma_n]$, where $\gamma_1 \in \Gamma_i$. Σ is similar. The "blank" symbol is the n-tuple $[\square, \ldots, \square]$. Each transition reads an n-tuple from Γ, and then writes an n-tuple from Γ on the same "square" before the head moves right or left.

Multiple Tapes



Multiple Tapes

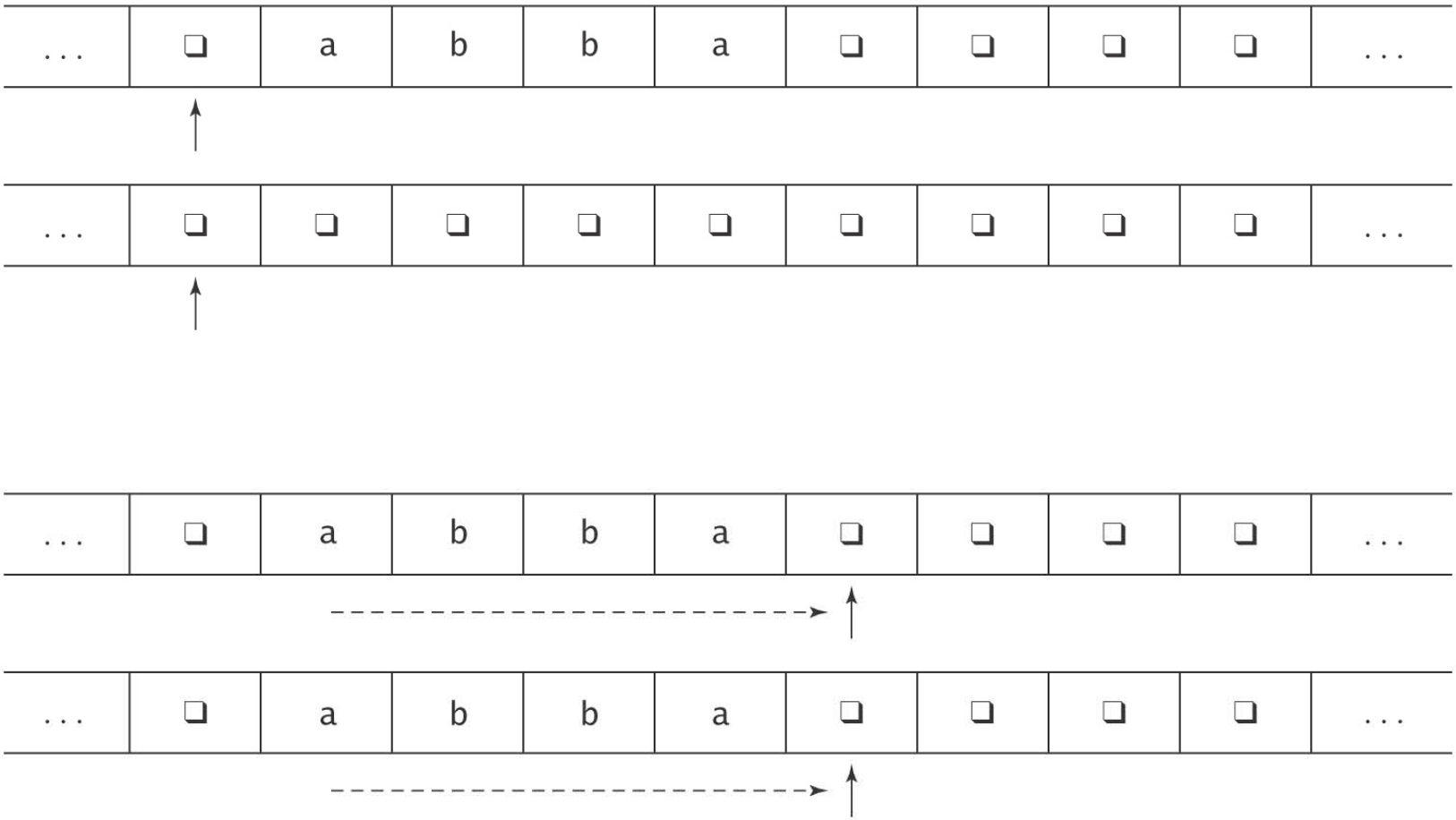
The transition function for a *k*-tape Turing machine:

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

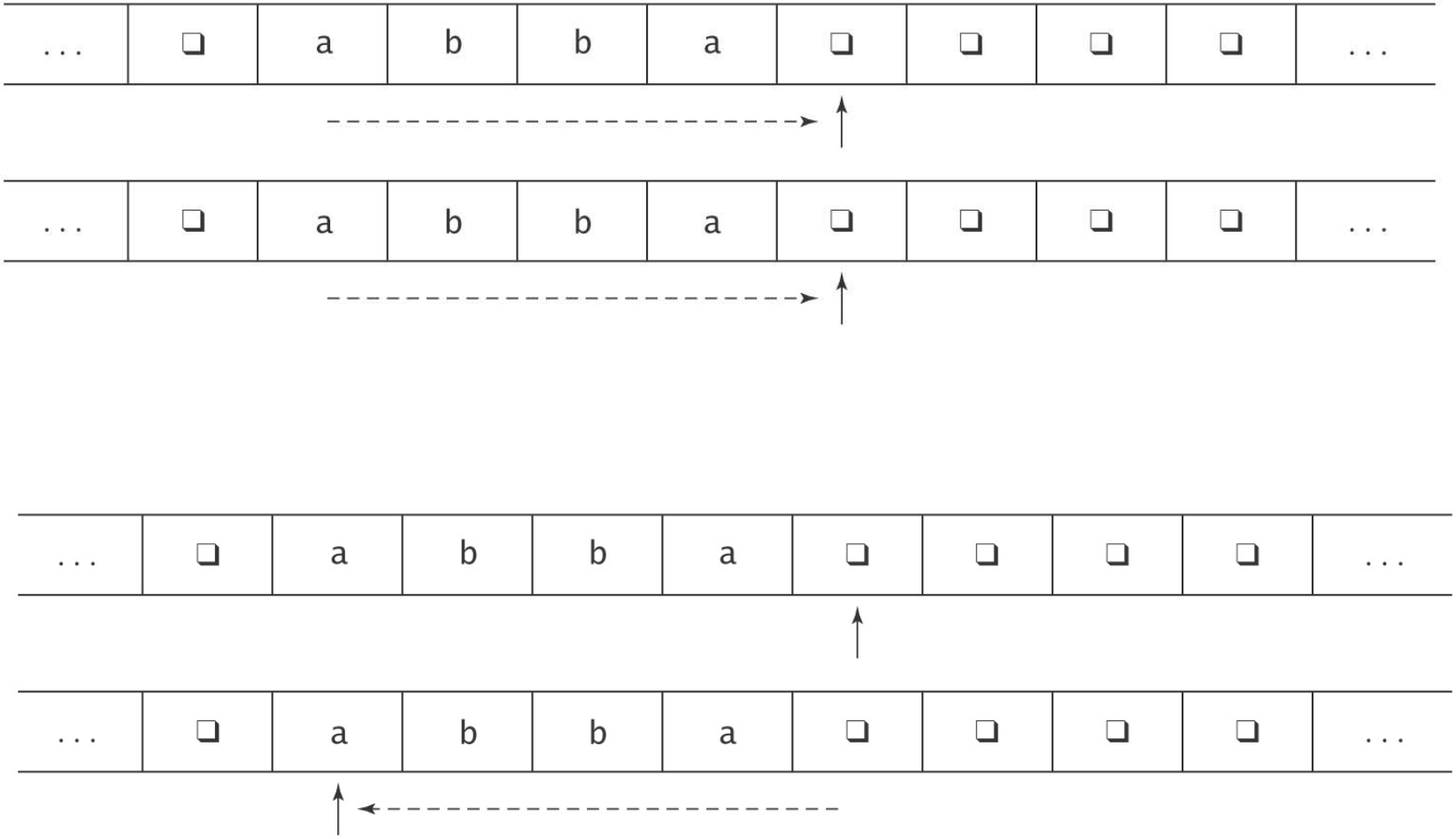
Input: initially all on tape 1, other tapes blank.
Output: what's left on tape 1, other tapes ignored.

Note: On each transition, any tape head is allowed to stay where it is.

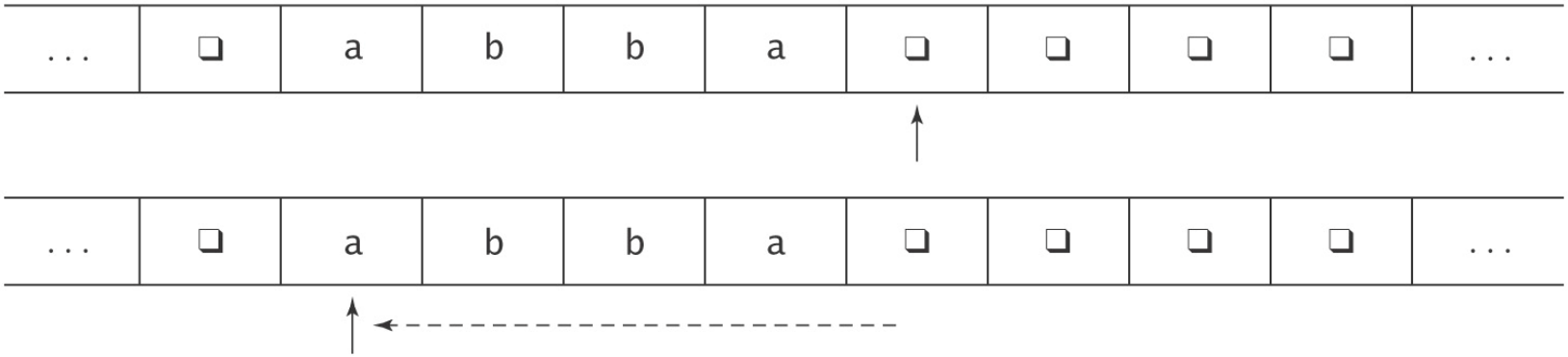
Example: Copying a String

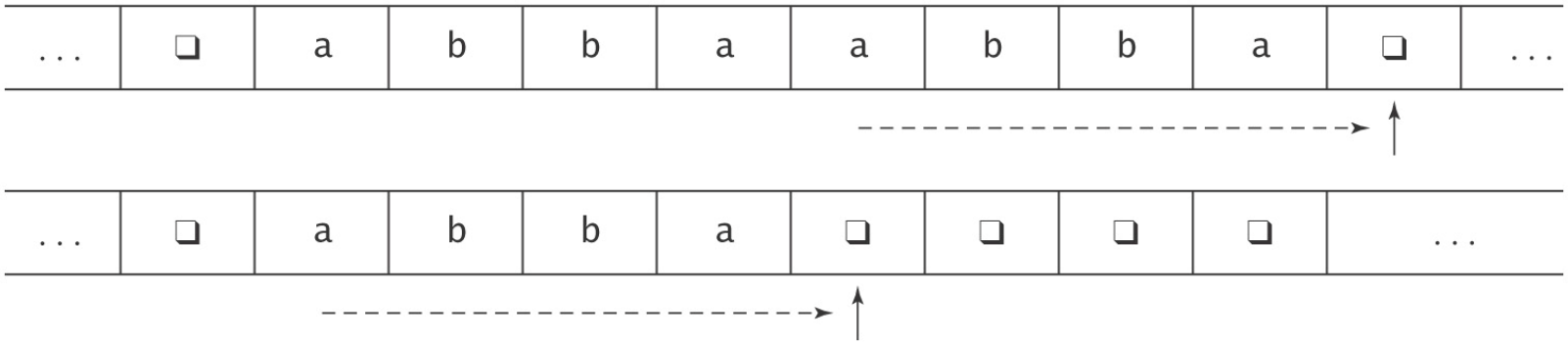


Example: Copying a String

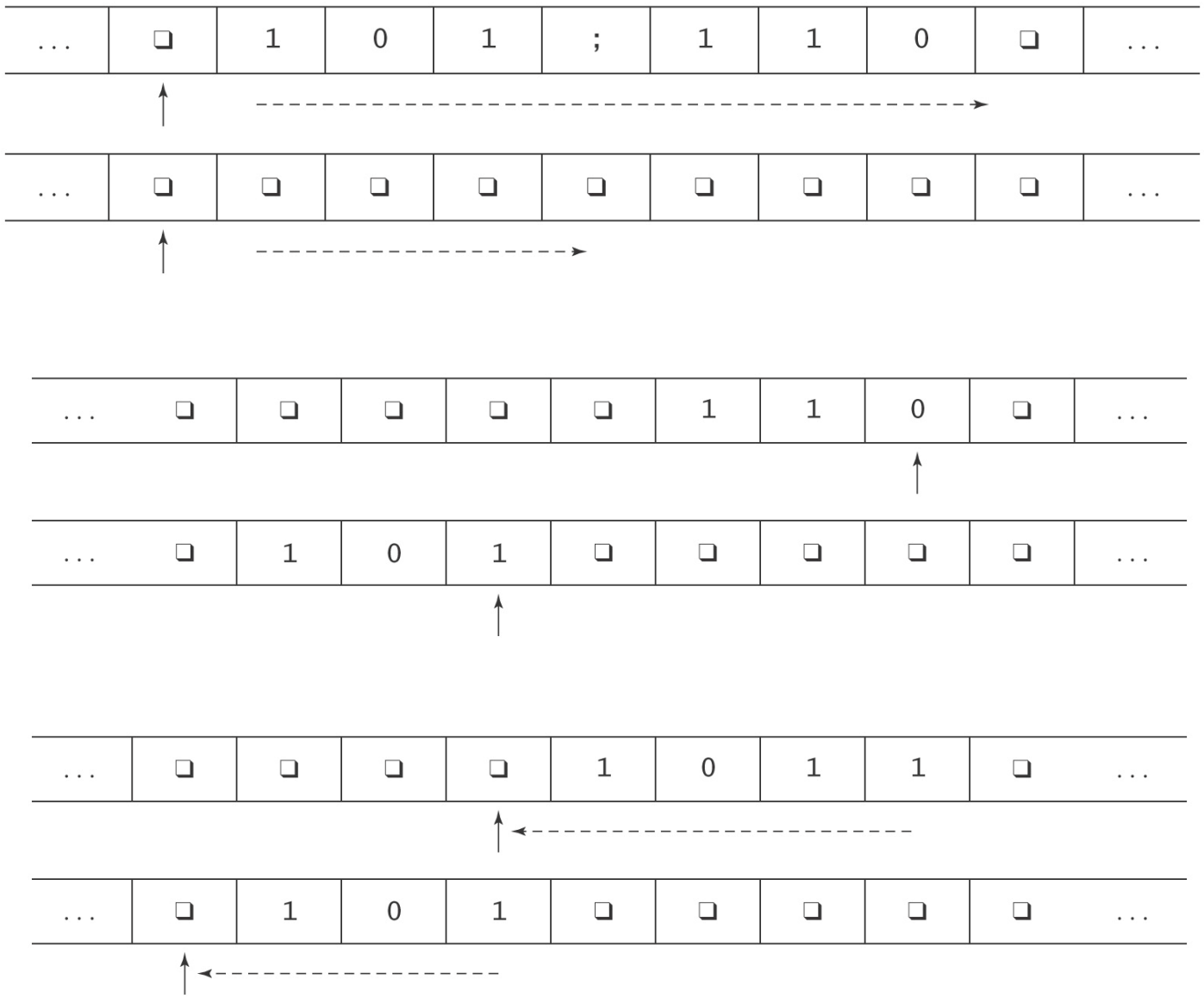


Example: Copying a String





Another Two Tape Example: Addition



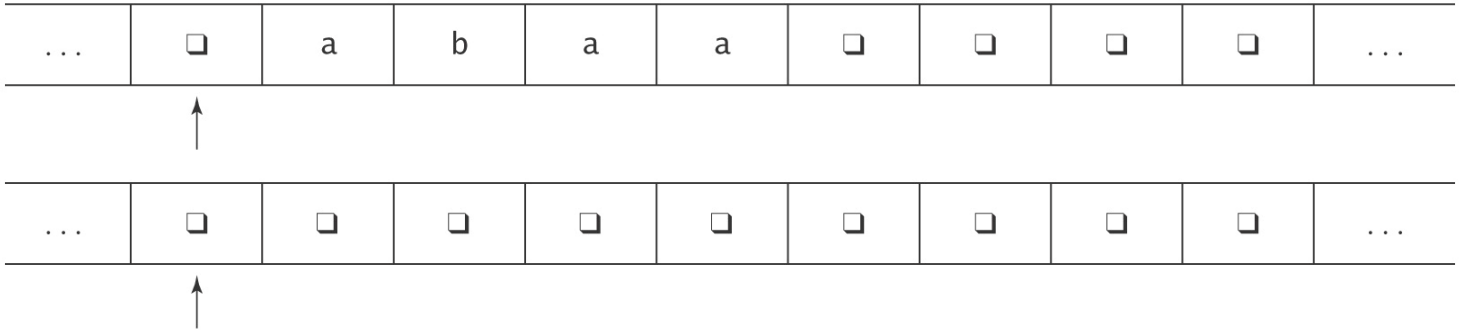
Adding Tapes Does Not Add Power

Theorem: Let $M = (K, \Sigma, \Gamma, \delta, s, H)$ be a k-tape Turing machine for some $k > 1$. Then there is a standard TM $M' = (K', \Sigma', \Gamma', \delta', s', H')$ 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)

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

(b)

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

Æ , a, b, (Æ , 1, Æ , 1), (a, 0, Æ ,0), (b, 0, Æ , 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 · ($|w| + n$).
2.2 = 2 · ($|w| + n$).
Total: $\mathcal{O}(n \cdot (|w| + n))$.

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

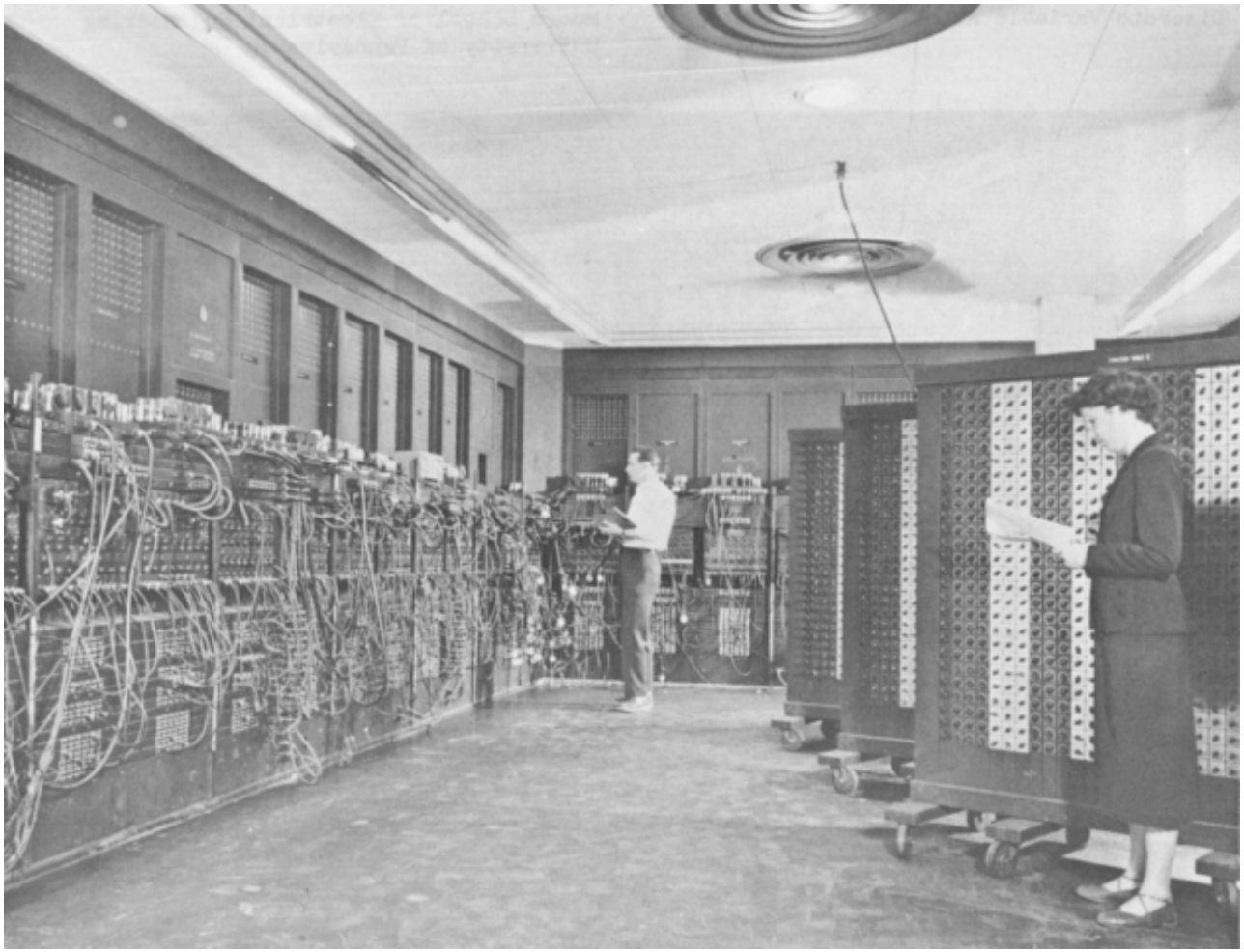
Total: $\mathcal{O}(n \cdot (|w| + n))$.
$= \mathcal{O}(n^2)$. *

* 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 on that input, 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 scheme for TMs.

2. Describe the operation of *U* when it is given input <*M*, *w*>, the encoding of:

    - a TM *M*, and

    - an input string *w*.

Encoding the States

- Let i be $\lceil \log_2(|K|) \rceil$.
  Each state is encoded by a letter and a string of i binary digits.

- Number the states from 0 to $|K|$-1 in binary:
  - The start state, s, is numbered 0.
  - Number the other states 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 yt'.
  - If t is the halting state n, assign it the string nt'.
  - If t is the halting state h, assign it the string ht'.
  - If t is any other state, assign it the string qt'.

Example of Encoding the States

Suppose *M* has 9 states.

i = 4

s = `q0000`,

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

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

Encoding a Turing Machine *M*, Continued

The tape alphabet:

Let j be $\lceil \log_2(|\Gamma|) \rceil$.
Each tape alphabet symbol is encoded as
$a y$ for some $y \in \{0, 1\}^+$, $|y| = j$

The blank symbol gets the j-character representation of 0

Example: $\Gamma = \{$ Æ , a, b, c $\}$. $j = 2$.

Æ = a00
a = a01
b = a10
c = a11