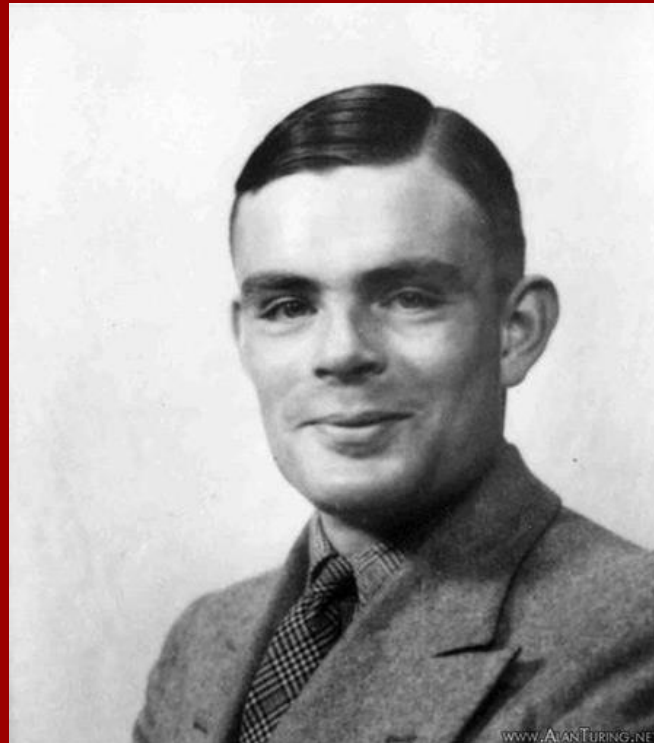


15-251: Great Theoretical Ideas in Computer Science
Lecture 5

Turing's Legacy



What is **computation**?

What is an **algorithm**?

How can we mathematically define them?

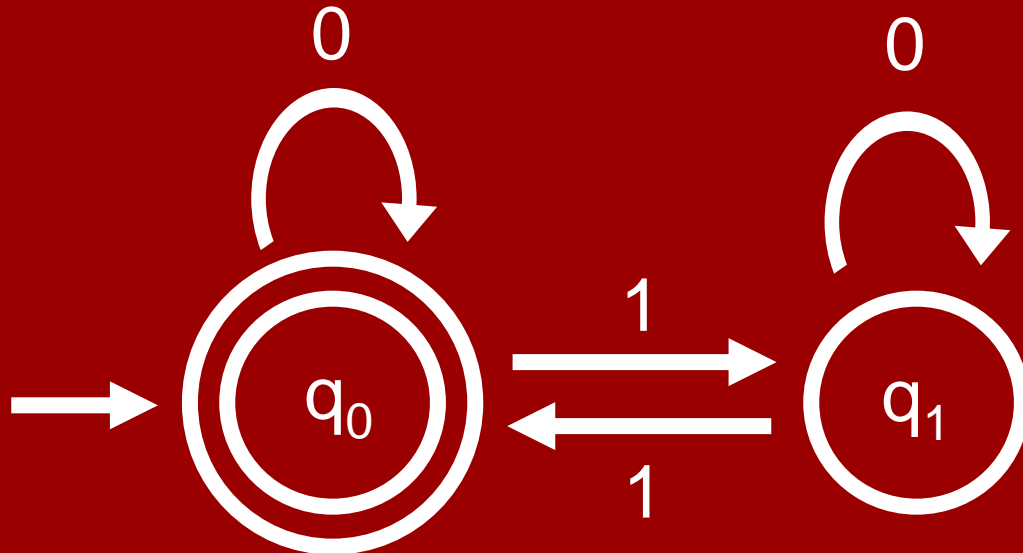
Quick Recap

Mathematical definition of a (computational) problem:

Input / output function: $f : \Sigma^* \rightarrow \Sigma^*$

Language: $L = \{x \in \Sigma^* \mid f(x) = 1\} \subseteq \Sigma^*$

A simple mathematical model for algorithms: DFAs



All Languages

Decidable Languages
(Solvable with “algorithms”) ???

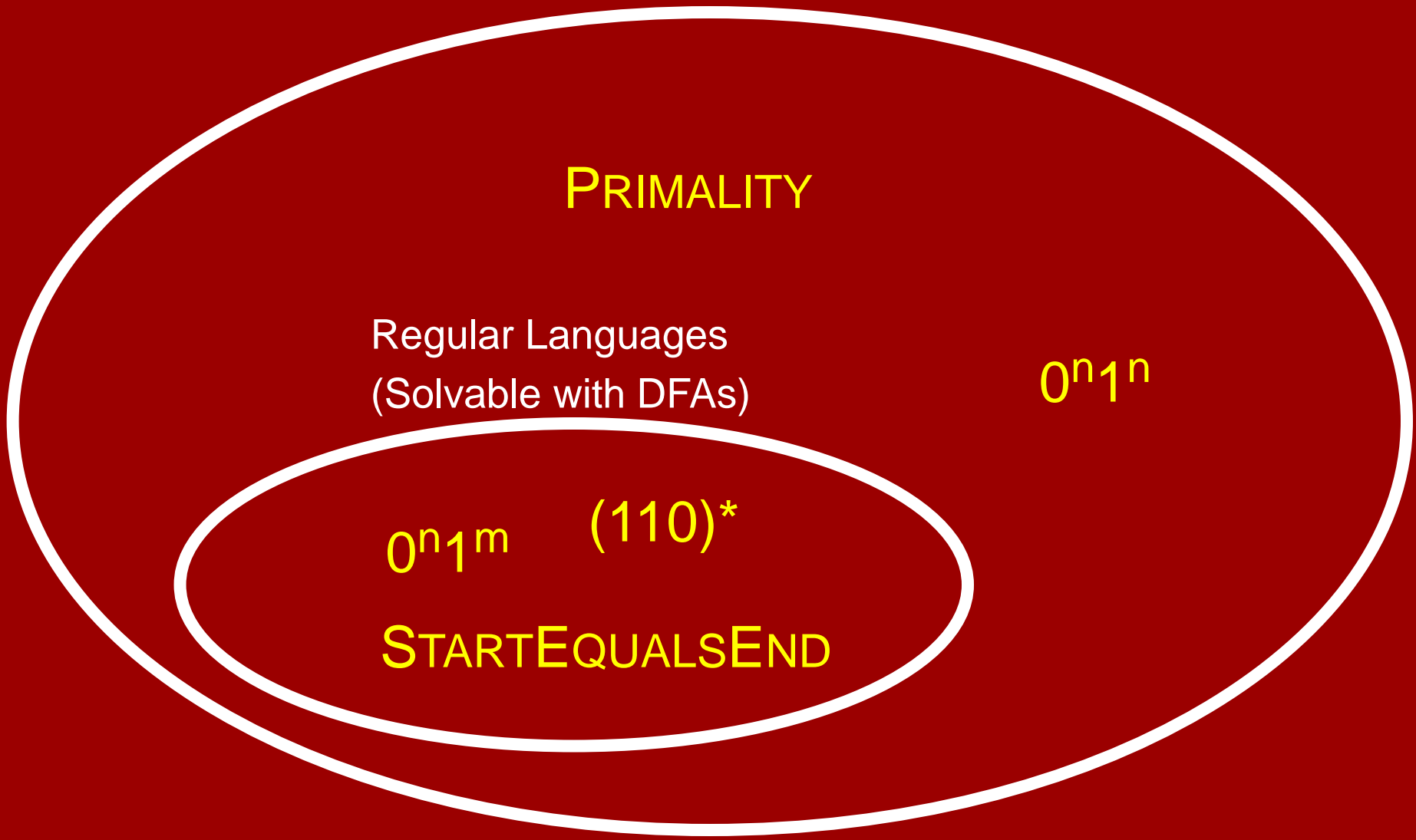
PRIMALITY

Regular Languages
(Solvable with DFAs)

$0^n 1^n$

$0^n 1^m$ $(110)^*$

STARTEQUALSEND



Solving 0^n1^n with Python

```
# Determines if string S is of form  $0^n 1^n$ 
def Solution( S ):
    i = 0
    j = len(S)-1
    while j >= i:
        if S[i] != '0' or S[j] != '1':
            return False
        i = i + 1
        j = j - 1
    return True
```

Solving $0^n 1^n$ with C

```
/* Determines if string S is of form 0^n 1^n */
int Solution(char S[])
{
    int i = 0, j;
    while (S[j] != NULL) /* NULL is end-of-string char */
        j++;
    j--;
    while (j >= i)
    {
        if (S[i] != '0' || S[j] != '1')
            return 0; /* Reject */
        i++;
        j--;
    }
    return 1; /* Accept */
}
```

All Languages

Decidable Languages
(Solvable with “algorithms”)

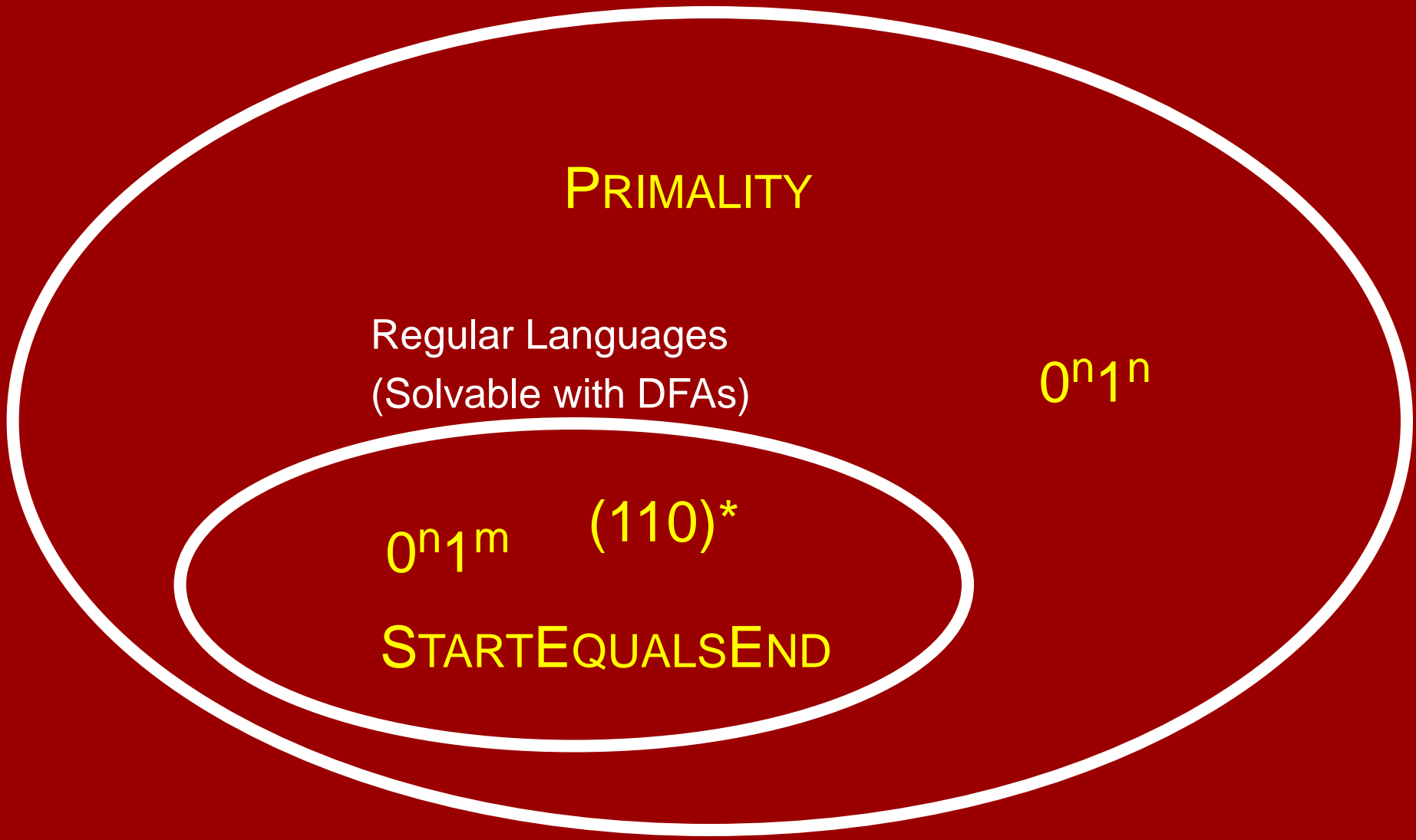
PRIMALITY

Regular Languages
(Solvable with DFAs)

$0^n 1^n$

$0^n 1^m$ $(110)^*$

STARTEQUALSEND



All Languages

Decidable Languages
(Solvable with Python)

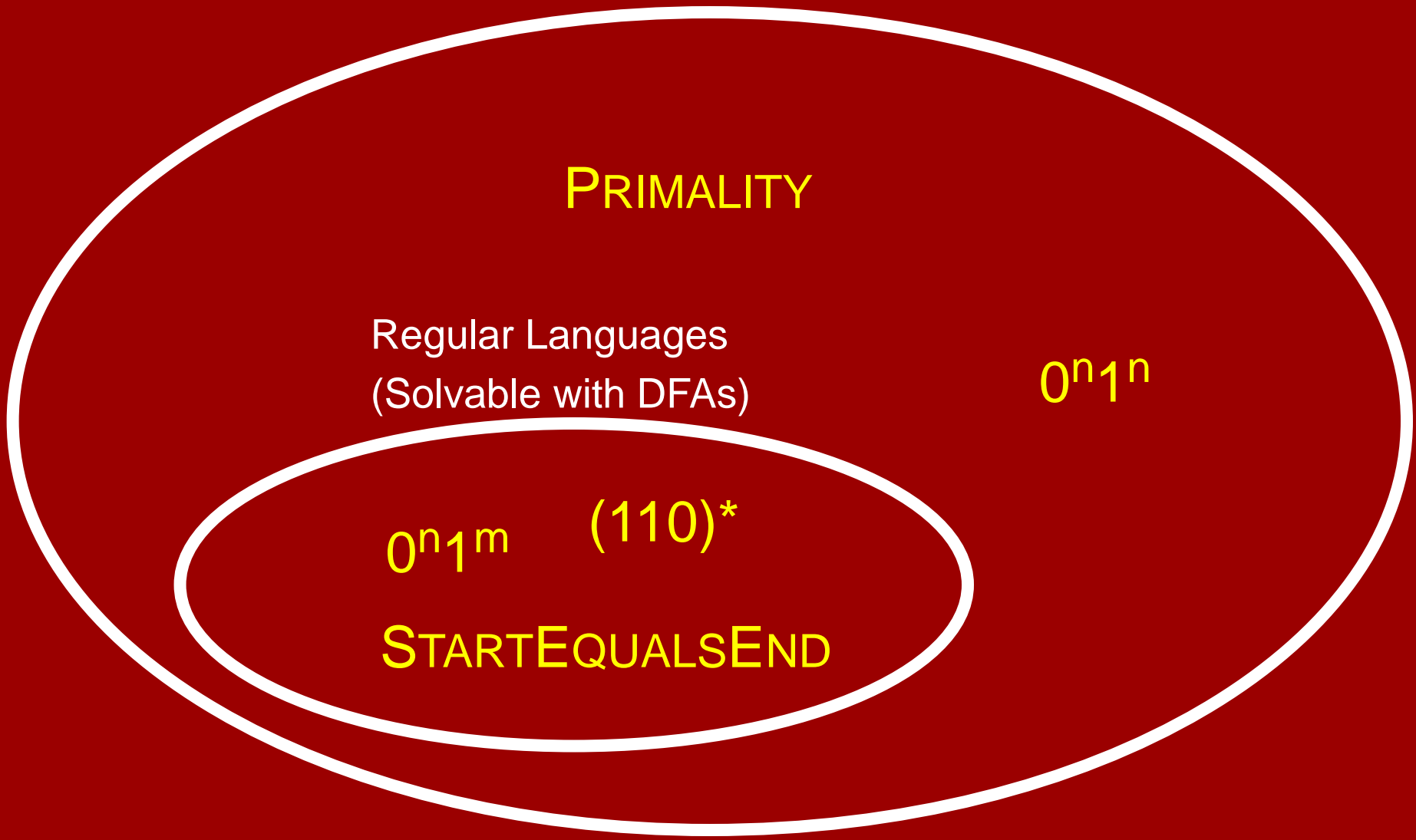
PRIMALITY

Regular Languages
(Solvable with DFAs)

$0^n 1^n$

$0^n 1^m$ $(110)^*$

STARTEQUALSEND



Question:

Should we just define “algorithm” to mean
a function written in Python?

(allowed access to unlimited memory)

Answer:

Actually, we'll see that this would be OK!

Downsides as a formal definition:

- Why choose Python?
Why not C, or Java, or SML, or...?
- Extremely complicated to rigorously define.
E.g., official 2011 ISO definition of C
requires a 701-page PDF file!
- A “philosophical” justification would be nice...

Downsides as a formal definition:

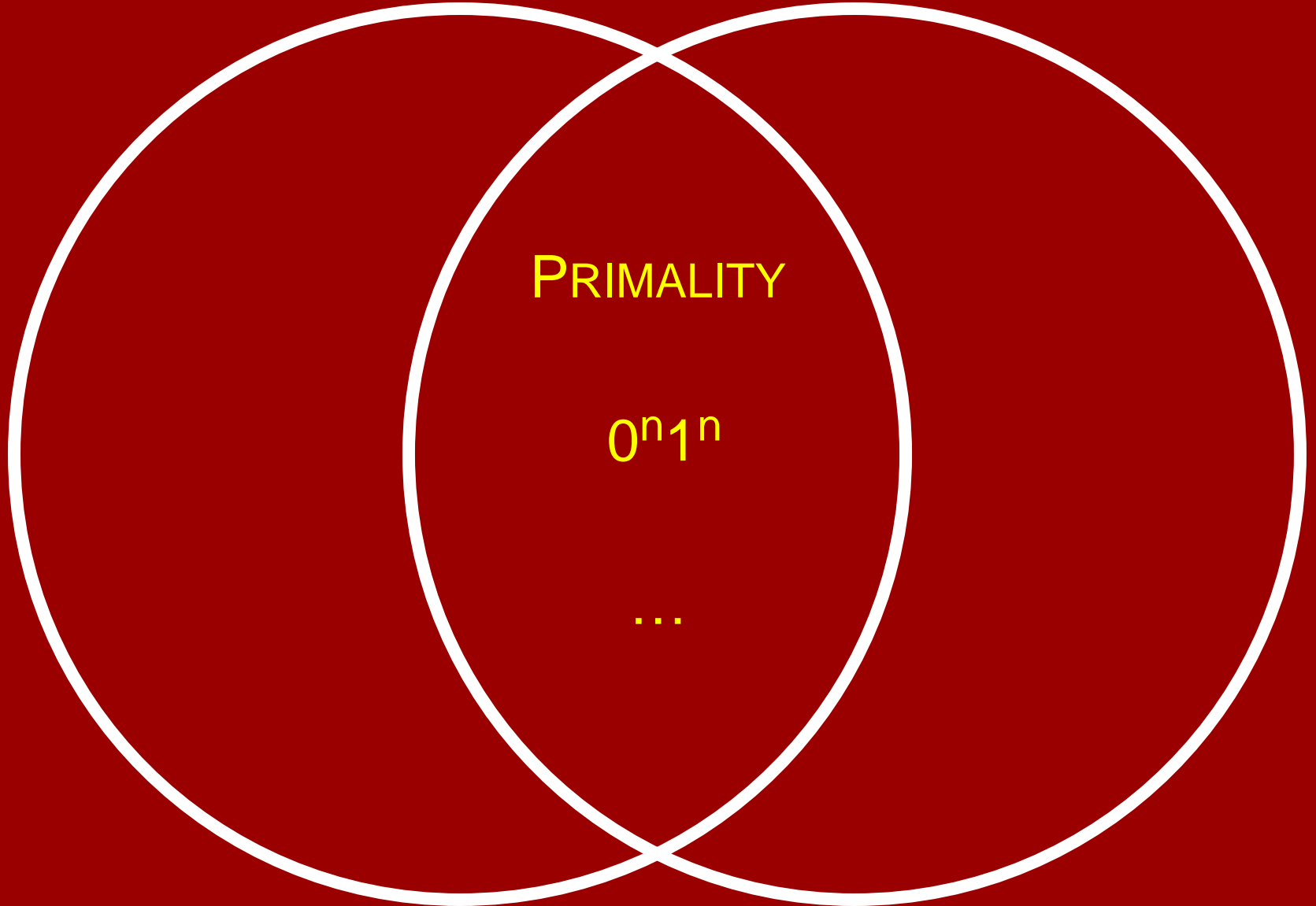
- Why choose Python?
Why not C, or Java, or SML, or...?
- Extremely complicated to rigorously define.
E.g., official 2011 ISO definition of C
requires a 701-page PDF file!
- A “philosophical” justification would be nice...

Claim:

Solvable with Python

=

Solvable with C



Claim:

Solvable with Python

=

Solvable with C

Proof intuition:

Our shared experience with programming.

“Proof:”

1. Solvable with Python \subseteq Solvable with C.

The standard **Python interpreter** is written in C.

2. Solvable with C \subseteq Solvable with Python.

It's pretty clear one can write a **C interpreter** in Python.

Interpreters

A Python function is (representable by) a string.

A **Python interpreter** is an algorithm M that
takes two inputs: P , a Python function;
 x , a string;
and step-by-step simulates $P(x)$.

In particular, $M(P,x)$ accepts if and only if $P(x)$ accepts.

Interpreters

You can write a **Python** interpreter in **C**.

You can write a **C** interpreter in **Python**.

You can write a **Python** interpreter in **Java**.

You can write a **Java** interpreter in **Python**.

You can write a **Python** interpreter in **SML**.

You can write an **SML** interpreter in **Python**.

You can write a **Python** interpreter in **Python!!**

The last one is called a
“**Universal Python Program**”

Solvable with Python
= Solvable with C
= Solvable with Java
= Solvable with SML

What we want to define
to be “computable”.



PRIMALITY

Regular Languages
(Solvable with DFAs)

$0^n 1^n$

$0^n 1^m$ $(110)^*$

STARTEQUALSEND

Downsides as a formal definition:

- Why choose Python?
Why not C, or Java, or SML, or...?
- Extremely complicated to rigorously define.
E.g., official 2011 ISO definition of C
requires a 701-page PDF file!
- A “philosophical” justification would be nice...

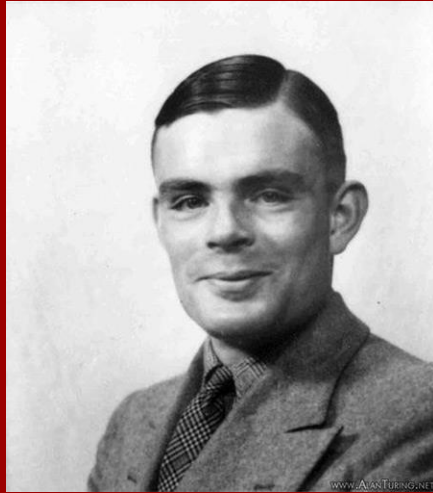
Downsides as a formal definition:

- Why choose Python?
Why not C, or Java, or SML, or...?
- **Extremely complicated to rigorously define.**
E.g., official 2011 ISO definition of C
requires a 701-page PDF file!
- A “philosophical” justification would be nice...

It would be nice to have a **totally minimal** (“**TM**”) programming language such that:

- a) can simulate Python, C, Java, SML, etc.;
- b) is simple enough to reason about rigorously completely mathematically.

Turing MachineTM



Inspired by





“★★★★”

ANN HORNADAY | THE WASHINGTON POST

HIDDEN FIGURES

NOW PLAYING

#HIDDENFIGURES 

Turing's mathematical abstraction of a computer

- A (human) computer writes symbols on paper
- WLOG, the paper is a sequence of squares
- No upper bound on the number of squares
- At most finitely many kinds of symbols
- Human observes one square at a time
- Human has only finitely many mental states
- Human can change symbols and change focus to a neighboring square, but only based on its state and the symbol it observes
- Human acts deterministically

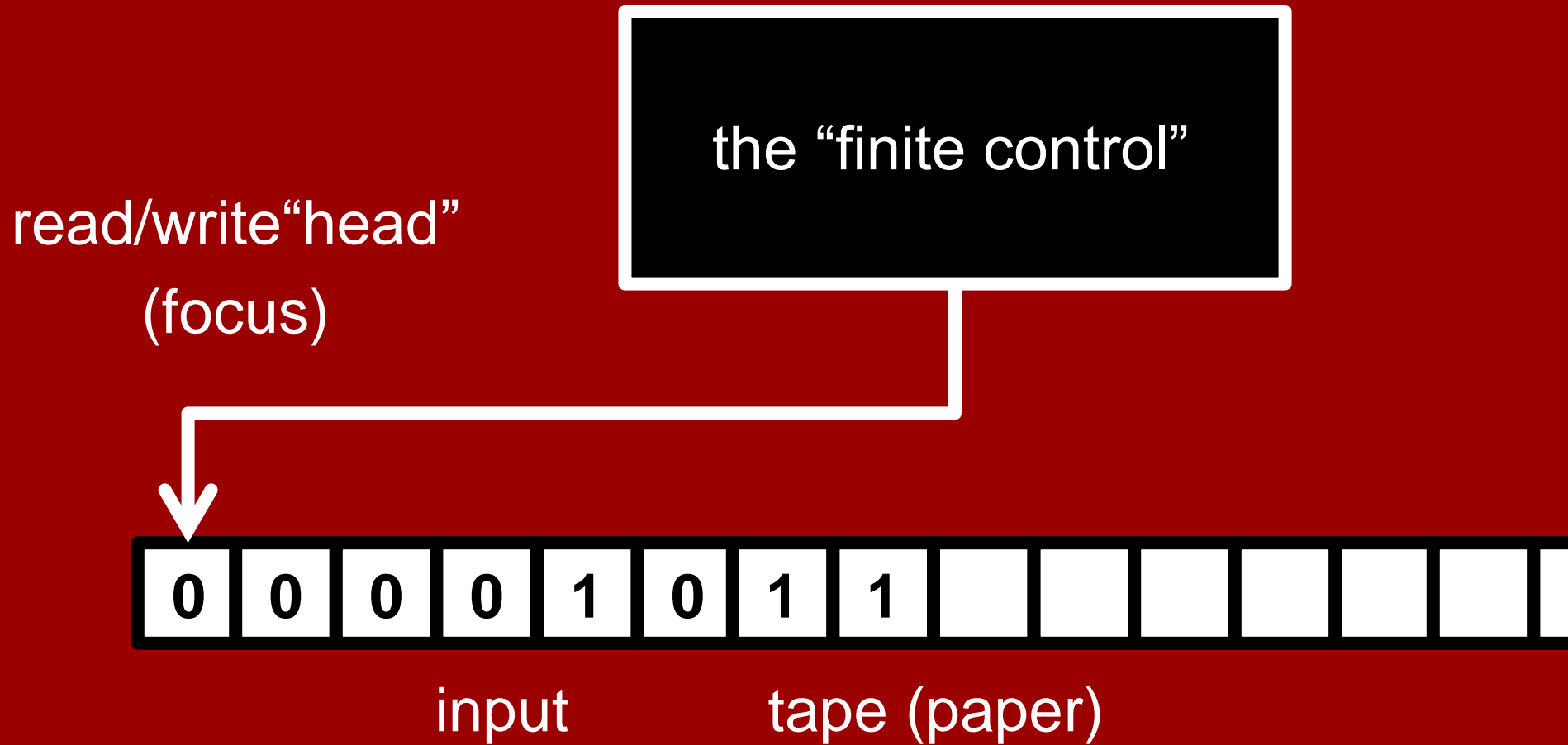


Illustration of a TM trying to decide $\{0^n 1^n : n \in \mathbb{N}\}$
We'll explain the finite control later

the “finite control”



the “tape alphabet”

in this example is $\{0,1,\#, \sqcup\}$

(blank)

the "finite control"



the "finite control"



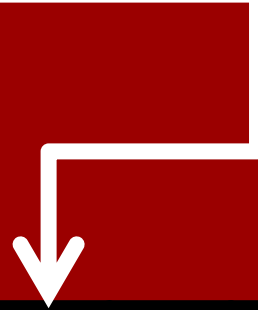
the "finite control"



the "finite control"



the "finite control"



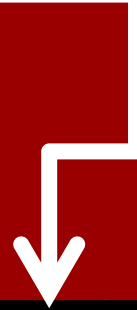
the “finite control”



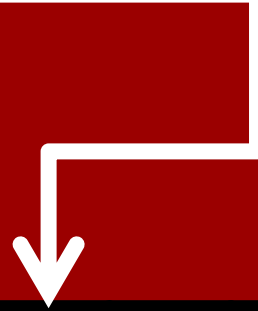
the “finite control”



the “finite control”



the "finite control"



the "finite control"



the "finite control"



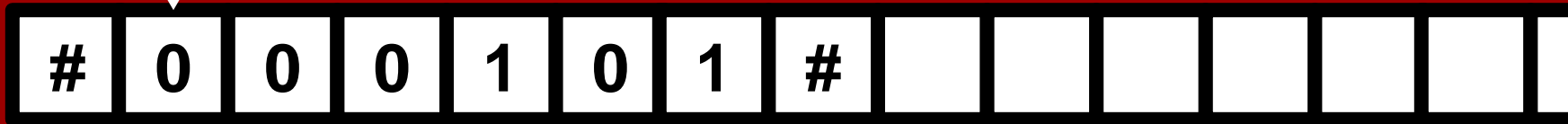
the "finite control"



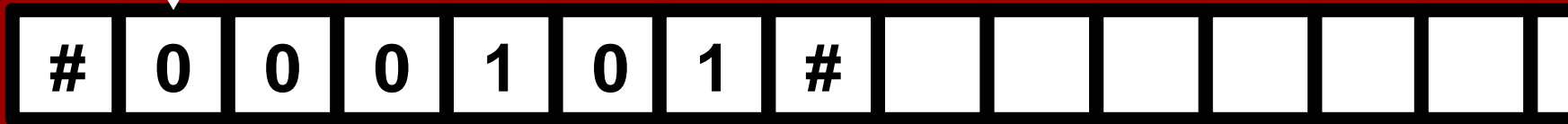
the "finite control"



the "finite control"



the "finite control"



the "finite control"



the "finite control"



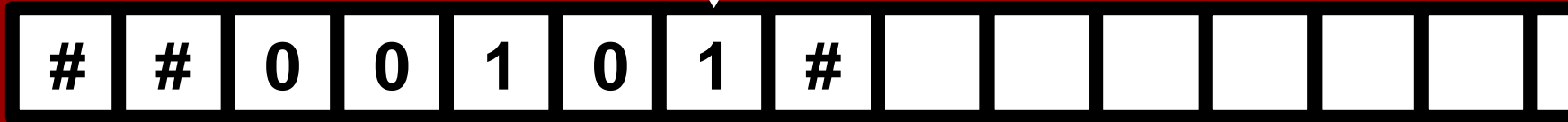
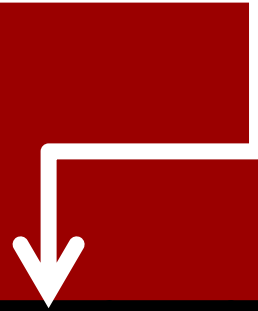
the "finite control"



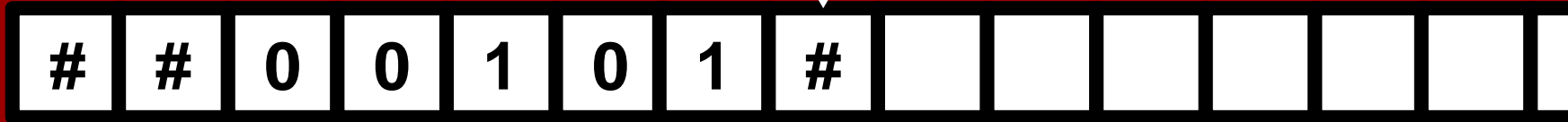
the "finite control"



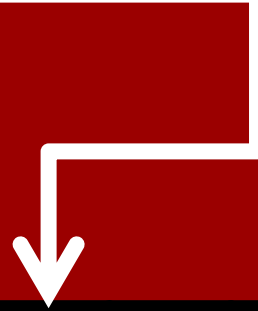
the "finite control"



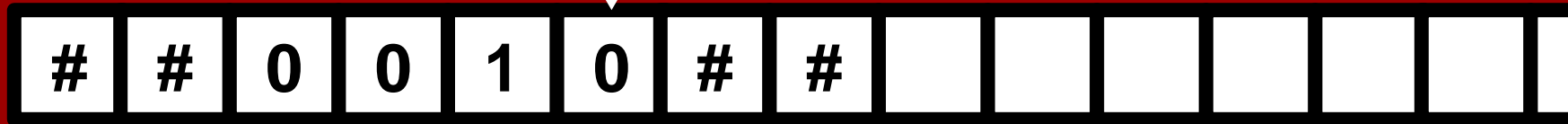
the “finite control”



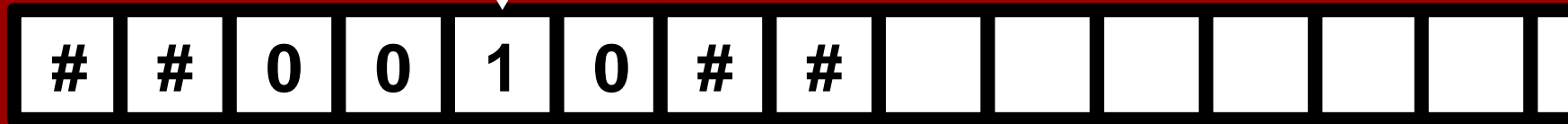
the "finite control"



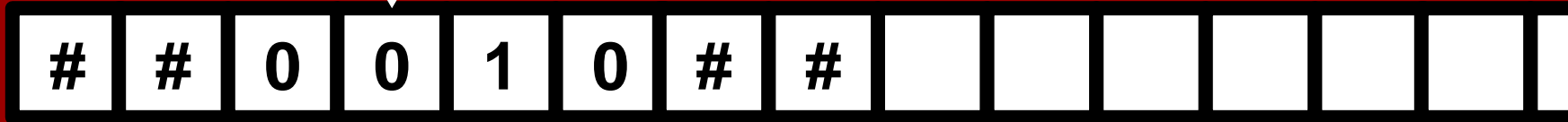
the "finite control"



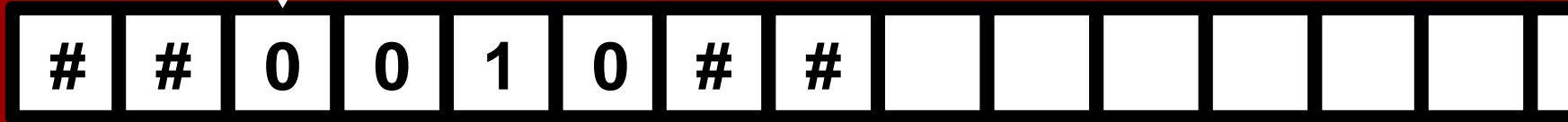
the "finite control"



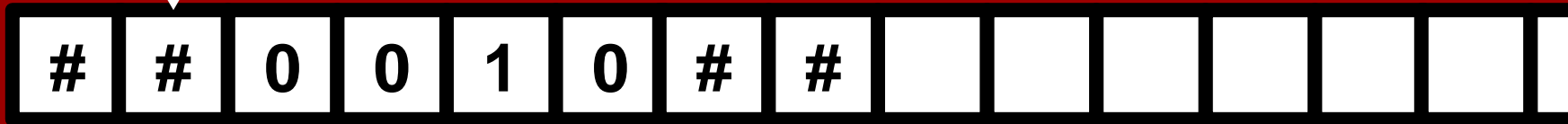
the "finite control"



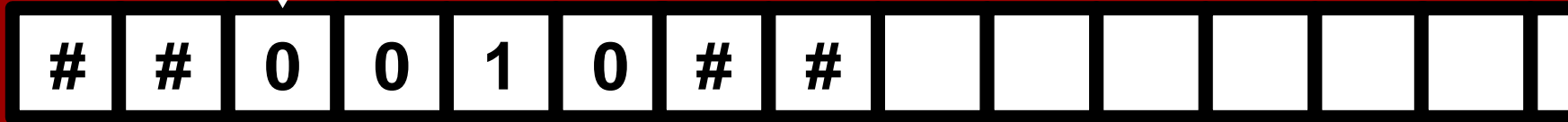
the "finite control"



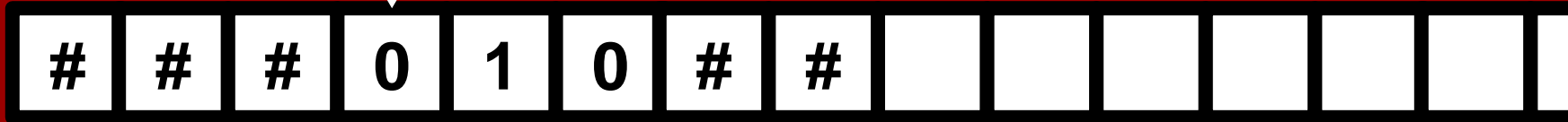
the "finite control"



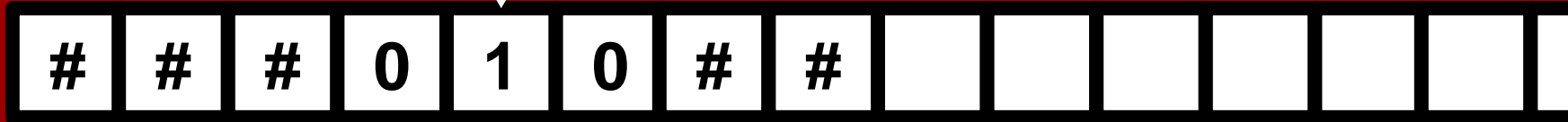
the "finite control"



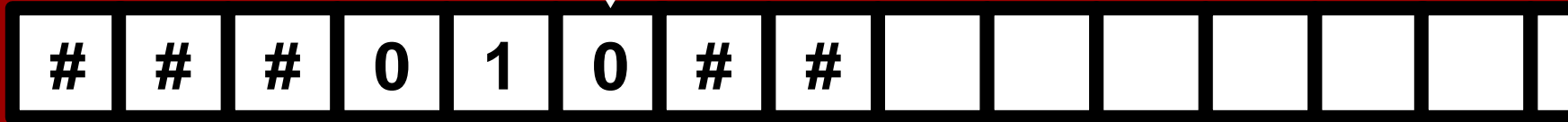
the "finite control"



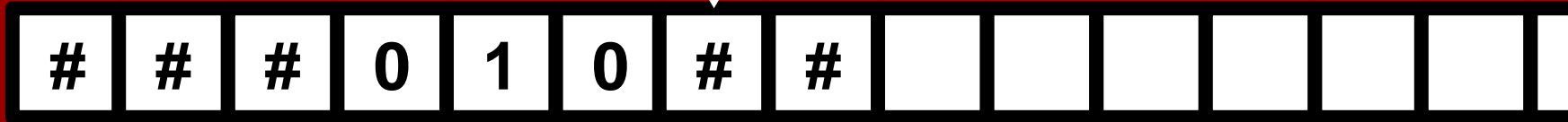
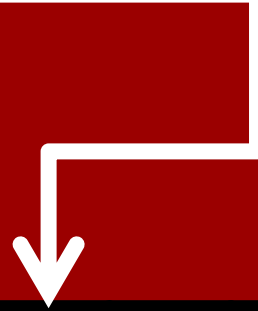
the "finite control"



the "finite control"

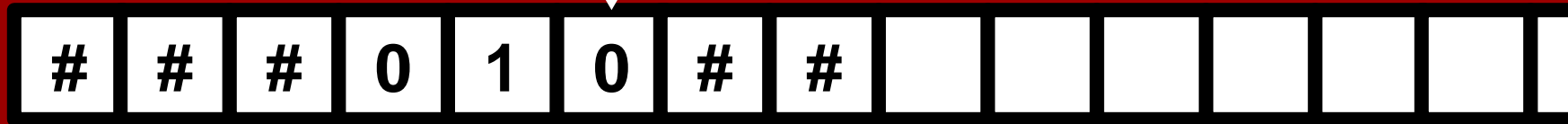


the "finite control"



REJECT

the "finite control"



the "finite control"

read/write head



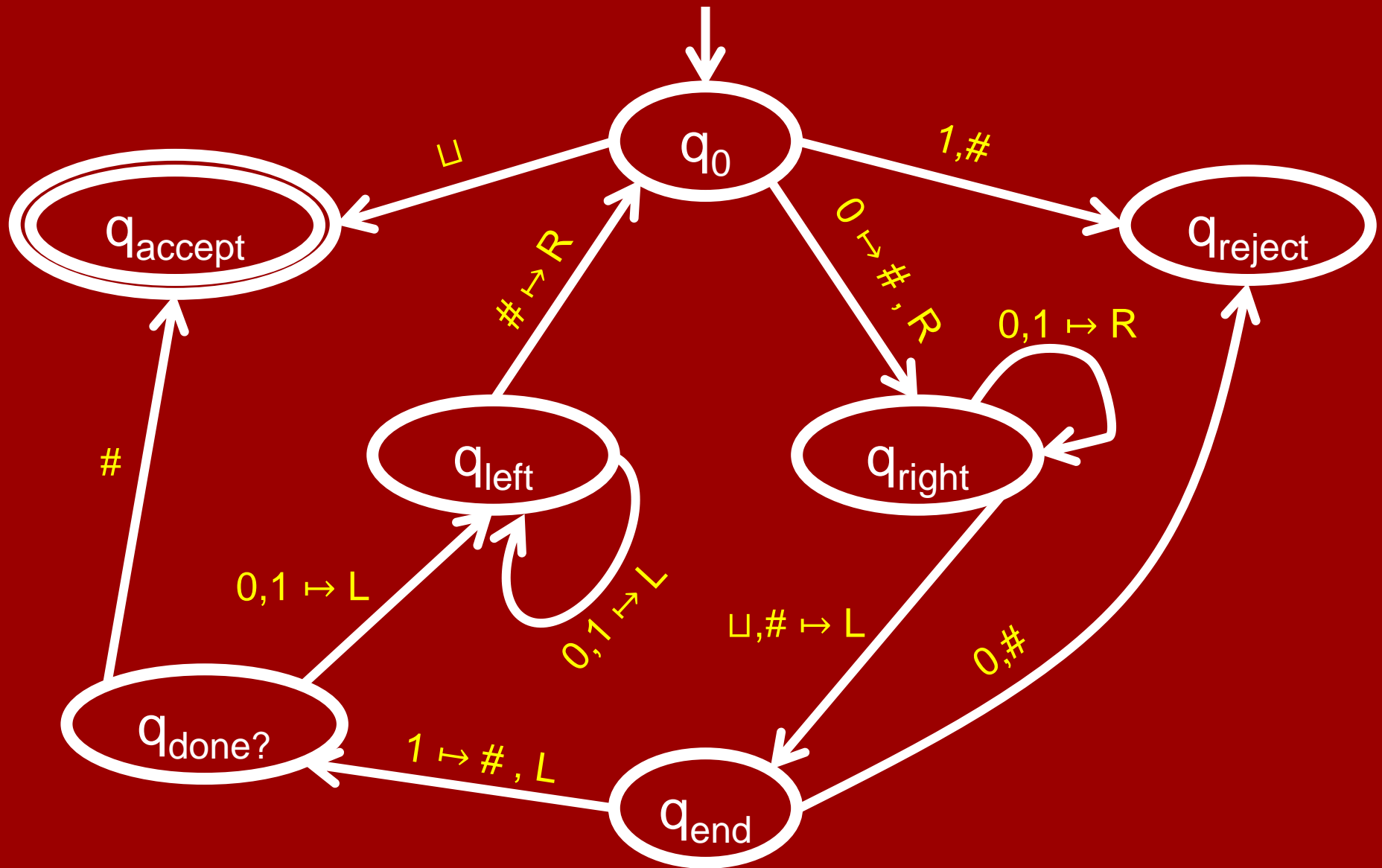
input

tape (paper)

Turing's mathematical abstraction of a computer

- A (human) computer writes symbols on paper
- WLOG, the paper is a sequence of squares
- No upper bound on the number of squares
- At most finitely many kinds of symbols
- Human observes one square at a time
- Human has only **finitely many** mental **states**
- Human **can change its state**, change symbols, and change focus to a neighboring square, but only **based on its state and the symbol it observes**
- Human acts **deterministically**

The **finite control** (aka transition rules)



Formal definition of Turing Machines

A Turing Machine is a 7-tuple

$$M = (Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \Sigma, \Gamma, \delta):$$

Q is a finite set of **states**,

$q_0 \in Q$ is the **start state**,

$q_{\text{accept}} \in Q$ is the **accept state**,

$q_{\text{reject}} \in Q$ is the **reject state**, $q_{\text{reject}} \neq q_{\text{accept}}$.

Σ is a finite **input alphabet** (with $\sqcup \notin \Sigma$),

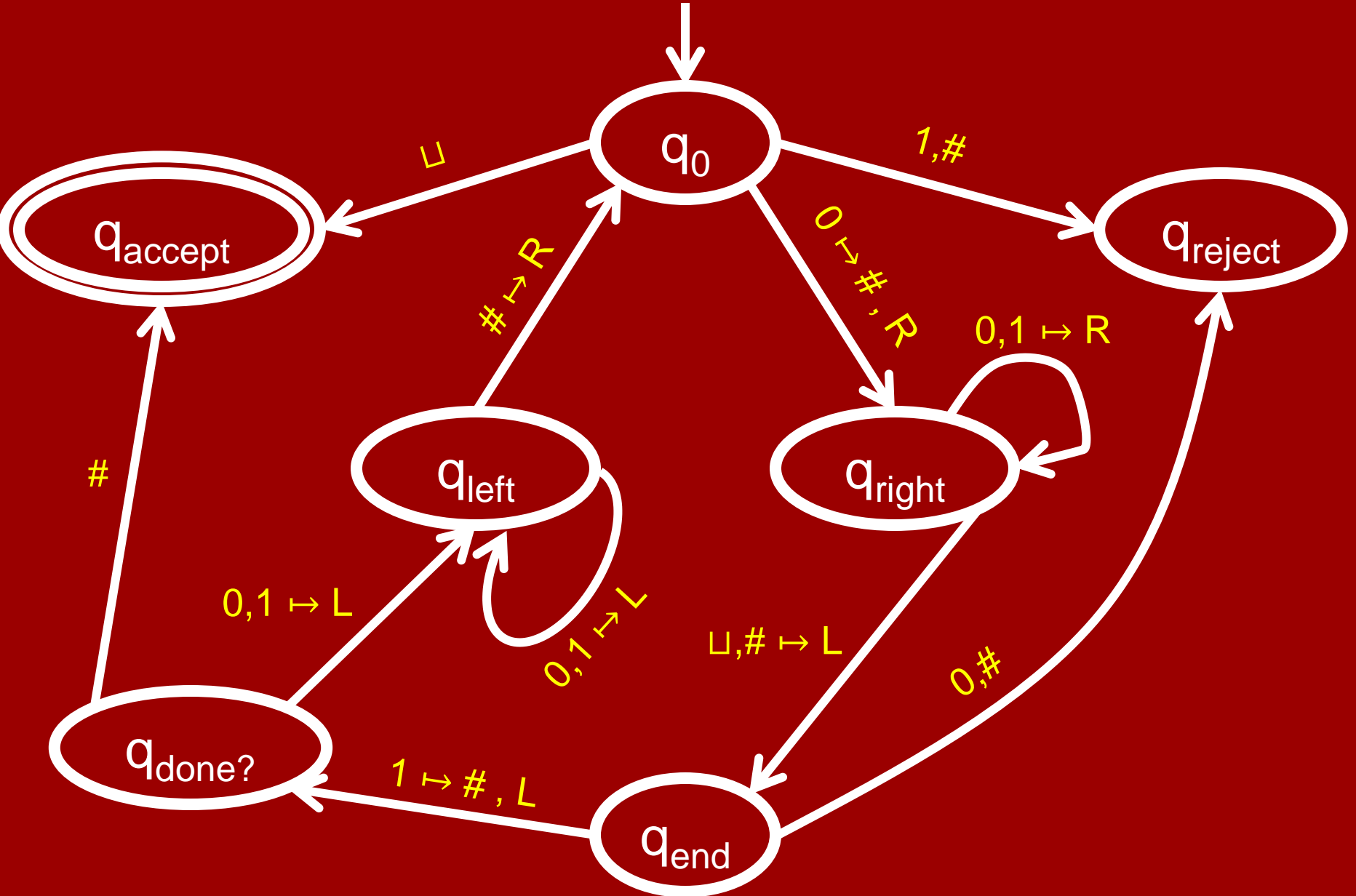
Γ is a finite **tape alphabet** (with $\sqcup \in \Gamma$, $\Sigma \subseteq \Gamma$),

$\delta : Q' \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the **transition function**

(here $Q' = Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$)

Input alphabet: $\Sigma = \{0,1\}$

Tape alphabet: $\Sigma = \{0,1,\#, \sqcup\}$



Formal definition of Turing Machines

Rules of computation:

Tape starts with input $x \in \Sigma^*$, followed by infinite \sqcup 's.

Control starts in state q_0 , head starts in leftmost square.

If the current state is q and head is reading symbol $s \in \Gamma$,
the machine transitions according to $\delta(q,s)$, which gives:
the next state,
what tape symbol to overwrite the current square with,
and whether the head moves Left or Right.

Technicality: moving left from the leftmost square \equiv staying put.

Continues until either the accept state or reject state reached.

When accept/reject state is reached, **M halts**.

M might also never halt, in which case we say it **loops**.

Decidable languages

Definition:

A language $L \subseteq \Sigma^*$ is **decidable** if there is a Turing Machine M which:

1. **Halts on every input** $x \in \Sigma^*$.
2. Accepts inputs $x \in L$ and rejects inputs $x \notin L$.

Such a Turing Machine is called a **decider**.

It **'decides'** the language L .

We like deciders. We don't like TM's that sometimes loop.

Computable functions

Definition:

A function $f : \Sigma^* \rightarrow \{0,1\}$ is **computable** if $L = \{x \in \Sigma^* : f(x) = 1\}$ is **decidable**

A function $f: \Sigma^* \rightarrow (\Gamma \setminus \{\sqcup\})^*$ is **computable** if there is a Turing Machine M which:

Halts on every input $x \in \Sigma^*$ with the tape containing $f(x)$ followed by \sqcup 's.

Decidable languages

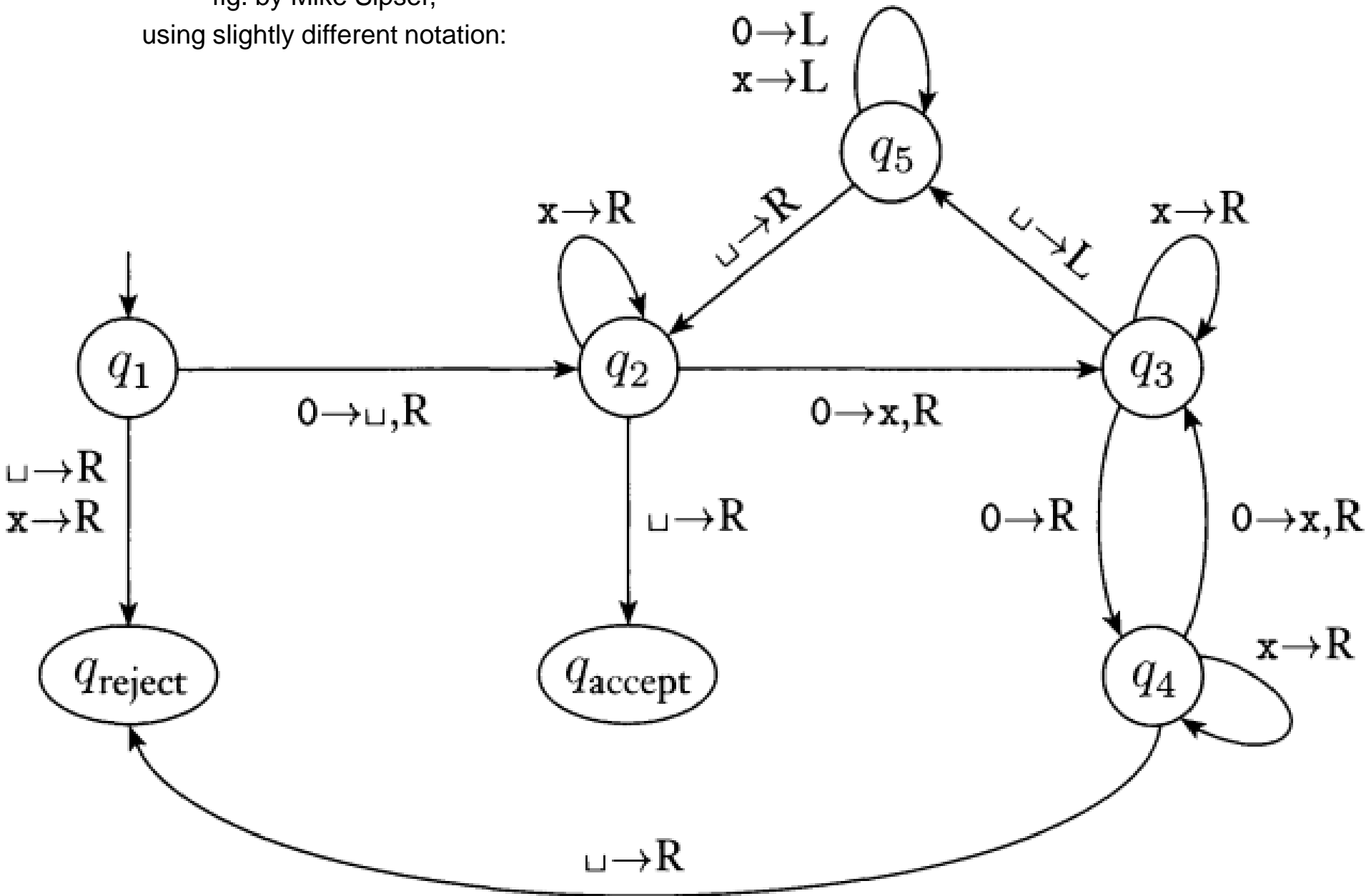
Examples:

Hopefully you're convinced that $\{0^n 1^n : n \in \mathbb{N}\}$ is decidable. (Recall it's not "regular".)

The language $\{0^{2^n} : n \in \mathbb{N}\} \subseteq \{0\}^*$,
i.e. $\{0, 00, 0000, 00000000, \dots\}$,
is decidable.

Proof: I'll show you a decider TM for it...

fig. by Mike Sipser,
using slightly different notation:



Describing Turing Machines

Low Level:

Explicitly describing all states and transitions.

Medium Level:

Carefully describing in English how the TM operates. Should be 'obvious' how to translate into a Low Level description.

High Level:

Skips 'standard' details, just highlights 'tricky' details. **For experts only!**

$\{0^{2^n} : n \in \mathbb{N}\}$ is decidable

Medium Level description:

1. Sweep from left to right across the tape, overwriting a # over top of every *other* 0.
2. If you saw one 0 on the sweep, **accept**.
3. If you saw an odd number of 0's, **reject**.
4. Move back to the leftmost square.
(Say you write \sqcup on the leftmost square at the very beginning so that you can recognize it later.)
5. Go back to step 1.

TM programming exercises & tricks

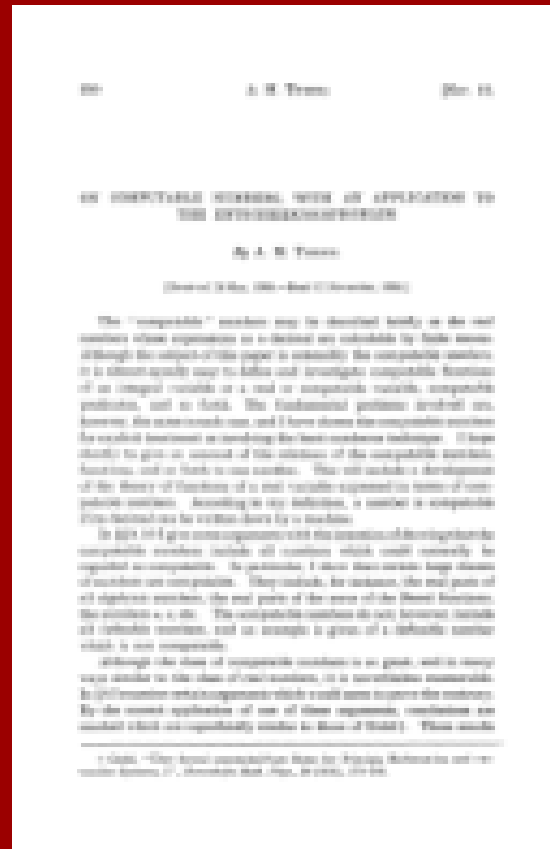
1. Move right (or left) until first \sqcup encountered.
2. Shift entire input string one cell to the right
3. Convert input $x_1x_2x_3\cdots x_n$ to $x_1\sqcup x_2\sqcup x_3\sqcup\cdots\sqcup x_n$
4. Simulate large tape alphabet Γ with just $\{0,1,\sqcup\}$
5. Ability to “mark” cells (e.g., replace symbol a by $\overset{\circ}{a}$)
6. Copy a stretch of tape between two marked cells into another marked section
7. Increment or Decrement an input in binary.
8. Implement basic string and arithmetic operations

TM programming exercises & tricks

9. Simulate a TM with 2 tapes and read/write heads
10. Implement a dictionary data structure
11. Simulate “random access memory”
12.
13. Simulate an assembly language interpreter
14. Simulate a C interpreter
15. Create a Turing Machine interpreter or Universal TM, i.e.,
a Turing Machine **U** whose input is
 $\langle M \rangle$, the **encoding** of a TM M ,
 x , a string
and which **simulates** the execution of M on x .

Universal Turing Machine

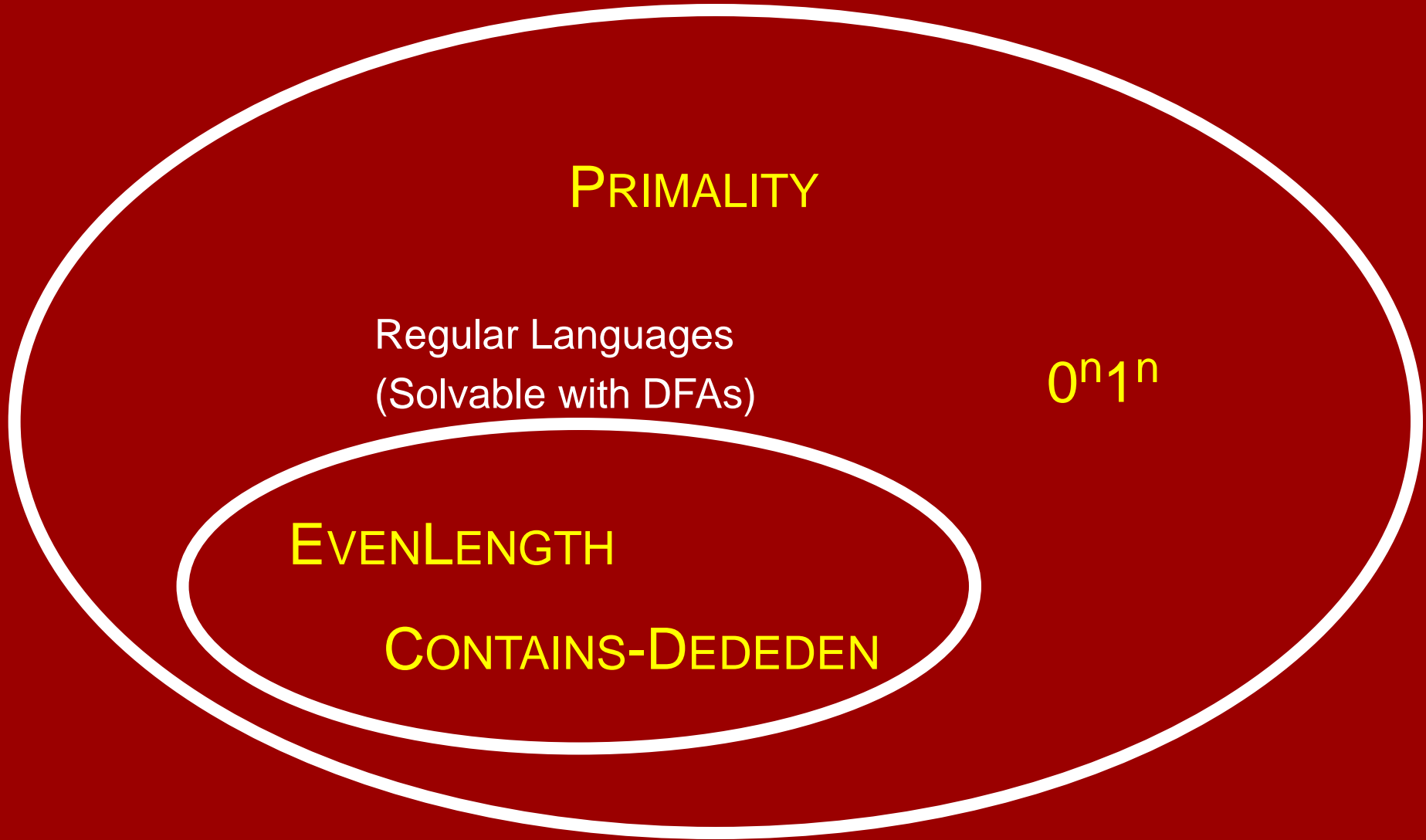
If you get stuck on the last exercise, you can look up the answer in Turing's 1936 paper!



Solvable with Python
= Solvable with C
= Solvable with Java
= Solvable with SML

=

Decidable Languages
(decidable by Turing Machines)



Church–Turing Thesis:

“Any natural / reasonable notion of computation can be simulated by a TM.”

Describing Turing Machines

Low Level:

Medium Level:

High Level:

Super-high Level:

Just describe an algorithm / pseudocode.

Assuming the Church–Turing Thesis
there exists a TM which executes that algorithm.

Definitions:

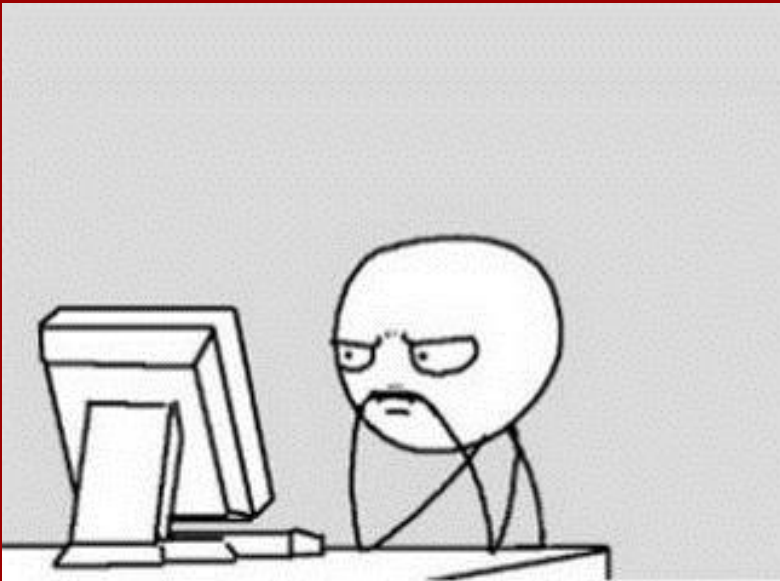
Turing Machines
Decidable languages/
computable functions
Universal TM
Church–Turing Thesis

Theorems/proofs:

$\{0^{2^n} : n \in \mathbb{N}\}$ is decidable
 $\{0^n 1^n : n \in \mathbb{N}\}$ is decidable
Equivalence of Solvability
(between Python, C, TM)

Practice:

Programming with TM's



Study Guide