Semi-Secure Websites & Modeling Computation

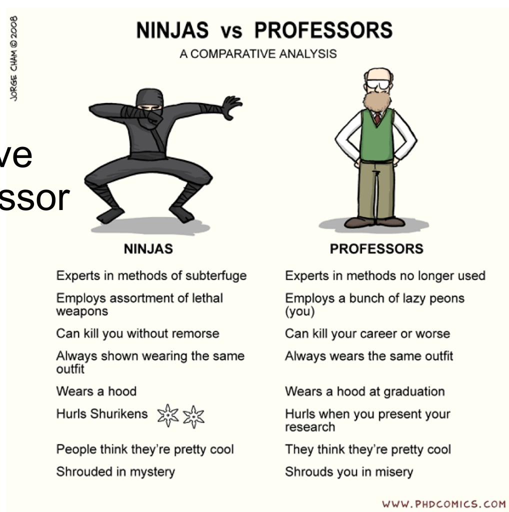
One-Slide Summary

- Users often **authenticate** themselves by providing passwords. We store and compare **hashes** of passwords, where a hash is a secure one-way function. A **cookie** is a bit of state associated with a webpage that is stored on the client.
- The **Turing machine** is a fundamental model of computation. It models input, output, processing and memory. A Turing machine has a **finite state machine controller** as well as an **infinite tape**. At each step it reads the current tape symbol, writes a new tape symbol, moves the tape head left or right one square, and moves to a new state in the finite state machine controller. Turing machines are **universal**: they are just as powerful as Scheme, Python, C, or Java.

#2

Authentication

How would I prove that I am a professor and not a ninja?



How do you authenticate?

- Something you know
  - Password
- Something you have
  - Physical key (email account?, transparency?)
- Something you are
  - Biometrics (voiceprint, fingerprint, etc.)

Serious authentication requires at least 2 kinds

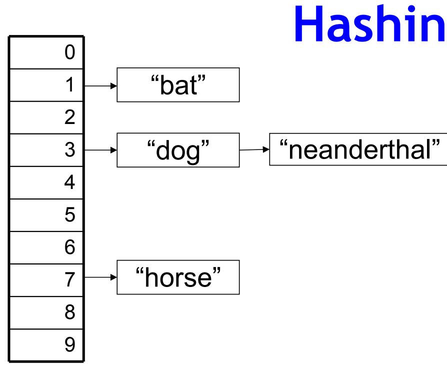
First Try: Encrypt Passwords

- Instead of storing password, store password encrypted with secret *K*.
- When user logs in, encrypt entered password and compare to stored encrypted password.

UserID	Password
alyssa	encrypt_K ("fido")
ben	encrypt_K ("schemer")
weimer	encrypt_K ("Lx.Ly.x")

Problem if *K* isn't so secret: $\text{decrypt}_K(\text{encrypt}_K(P)) = P$

Hashing



$H(\text{string } s) = (s[0] - \text{'a'}) \bmod 10$

Many-to-one: maps a large number of values to a small number of hash values

Even distribution: for typical data sets, probability of $(H(x) = n) = 1/N$ where N is the number of hash values and $n = 0..N-1$.

Efficient: $H(x)$ is easy to compute.

Cryptographic Hash Functions

One-way

Given h, it is hard to find x such that $H(x) = h$.

Collision resistance

Given x, it is hard to find $y \neq x$ such that $H(y) = H(x)$.

Example One-Way Function

Input: two 100 digit numbers, x and y

Output: the middle 100 digits of $x * y$

Given x and y, it is easy to calculate

$f(x, y)$ = select middle 100 digits $(x * y)$

Given $f(x, y)$ hard to find x and y.

A Better Hash Function?

- $H(x) = \text{encrypt}_x (0)$
- Weak collision resistance?
  - Given x, it should be hard to find $y \neq x$ such that $H(y) = H(x)$.
  - Yes - encryption is one-to-one. (There is no such y.)
- A good hash function?
  - No, its output is as big as the message!

Actual Hashing Algorithms

- Based on cipher block chaining
  - Start by encrypting 0 with the first block
  - Use the next block to encrypt the previous block
- SHA [NIST95] - 512 bit blocks, 160-bit hash
- MD5 [Rivest92] - 512 bit blocks, produces 128-bit hash
  - This is what we use in HoosHungry
  - It has been broken!

Hashed Passwords

UserID	Password
alyssa	*md5* (“fido”)
ben	*md5* (“schemer”)
weimer	*md5* (“Lx.Ly.x”)

Dictionary Attacks

- Try a list of common passwords
  - All 1-4 letter words
  - List of common (dog) names
  - Words from dictionary
  - Phone numbers, license plates
  - All of the above in reverse
- Simple dictionary attacks retrieve most user-selected passwords
- Precompute $H(x)$ for all dictionary entries

(at least) 86% of users are dumb and dumber

Single ASCII character	0.5%
Two characters	2%
Three characters	14%
Four alphabetic letters	14%
Five same-case letters	21%
Six lowercase letters	18%
Words in dictionaries or names	15%
Other (possibly good passwords)	**14%**

(Morris/Thompson 79)

Authenticating Users

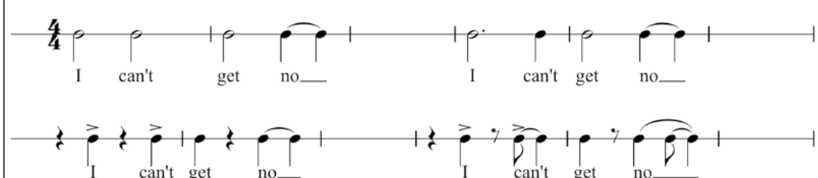
- User proves they are a worthwhile person by having a legitimate email address
  - Not everyone who has an email address is worthwhile
  - Its not too hard to snoop (or intercept) someone's email
- But, provides much better authenticating than just the honor system

Liberal Arts Trivia: Literature

- This slave and storyteller in Ancient Greece is credited with *The Fox and the Grapes*, *The Tortoise and the Hare*, and *The Boy Who Cried Wolf*. His work inspired the French fabulist Jean de la Fontaine, a widely-read 17th century poet.

Liberal Arts Trivia: Music Theory

- This term includes a variety of rhythms which are in some way unexpected: they deviate from the strict succession of regularly spaced strong and weak beats in a meter. This can arise from a stress on a normally unstressed beat or a rest where one would normally be stressed. This technique is used in many musical styles, including funk, reggae, ragtime, jazz, classical music, and is a popular back beat for contemporary popular music.



Using Cookies

- A **cookie** is website state stored on the client, not on a backend database.
- Cookie must be sent before any HTML is sent (util.printHeader does this)
- Be careful how you use cookies – anyone can generate any data they want in a cookie
  - Make sure they can't be tampered with: use md5 hash with secret to authenticate
  - Don't reuse cookies - easy to intercept them (or steal them from disks): use a counter than changes every time a cookie is used

Hungry vs. Cookies

```
def checkCookie ():
  try:
    if 'HTTP_COOKIE' in os.environ:
      cookies = os.environ['HTTP_COOKIE']
      c = Cookie.SimpleCookie(cookies)
      user = c['user'].value
      auth = c['authenticator'].value
      count = users.userTable.getCookieCount (user)
      ctest = md5crypt.encrypt (constants.ServerSecret + str(count) + user, \
                                str(count))
      if ctest == auth:
        users.userTable.setCurrentUser (user)
        return True
      else:
        users.userTable.setCurrentUser (False)
        return False
    else:
      return False
  except:
    return False
```

Paging UVA NetBadge!

Problems Left

- The database password is visible in plaintext in the Python code
  - No way around this (with UVa mysql server)
  - Anyone who can read UVa filesystem can access your database
- The password is transmitted unencrypted over the Internet (later)
- Proving you can read an email account is not good enough to authenticate for important applications

How convincing was our Halting Problem proof?

```
def contradict_halts(x):
    if halts?(contradict_halts):
        loop_forever()
    else:
        return True
```

contradicts_halts cannot exist. Everything we used to make it except halts? does exist, therefore halts? cannot exist.

This “proof” assumes Python exists and is consistent!

PyCharming

Is PyCharm a proof that Python exists?

PyCharm

Is PyCharm a proof that PyCharm exists?

```
def make_huge(n):
    If (n == 0): return []
    else: return make_huge(n-1) + \
        make_huge(n-1)
make_huge(10000)
```

No!
Python/Java/etc. all fail to evaluate some program!

Solutions

- Option 1: Prove “Python” does exist
  - Show that we could implement all the evaluation rules (if we had “Java”, our Mini Python interpreter would be a good start, but we don’t have “Java”)
- Option 2: Find a simpler computing model
  - Define it precisely
  - Show that “contradict_halts” can be defined in this model

Modeling Computation

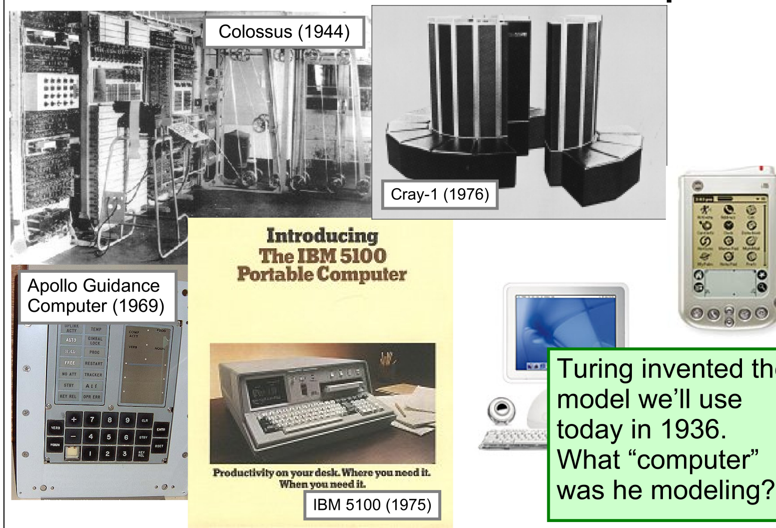
- For a more convincing proof, we need a more precise (but simple) model of what a computer can do
- Another reason we need a model:

  Does complexity really make sense without this? (how do we know what a “step” is? are they the same for all computers?)

What is a model?

$$\nabla \cdot D = \rho$$
$$\nabla \cdot B = 0$$
$$\nabla \times E = -\frac{\partial B}{\partial t}$$
$$\nabla \times H = J + \frac{\partial D}{\partial t}$$

How should we model a Computer?



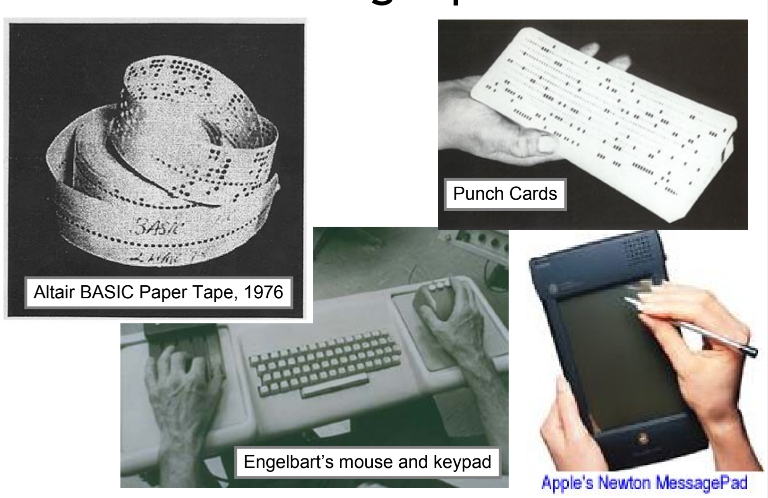
Turing invented the model we'll use today in 1936. What "computer" was he modeling?

"Computers" before WWII

Modeling Computers

- **Input**
  - Without it, we can't describe a problem
- **Output**
  - Without it, we can't get an answer
- **Processing**
  - Need some way of getting from the input to the output
- **Memory**
  - Need to keep track of what we are doing

Modeling Input



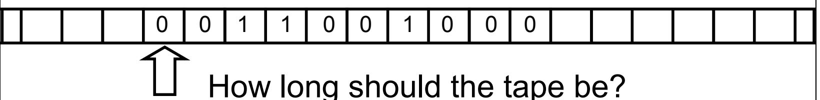
Turing's "Computer"

"Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book."

Alan Turing,
On computable numbers, with an application to the Entscheidungsproblem, 1936

Simplest Input

- Non-interactive: like punch cards and paper tape
- One-dimensional: just a single **tape** of values, pointer to one square on tape

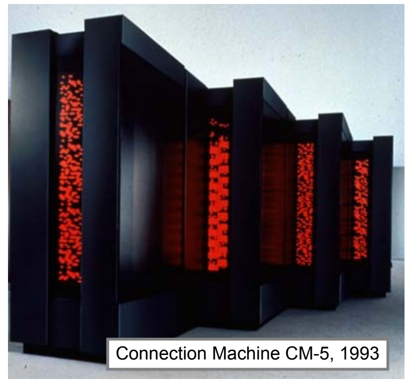


How long should the tape be?

Infinitely long! We are *modeling* a computer, not building one. Our model should not have silly practical limitations (like a real computer does).

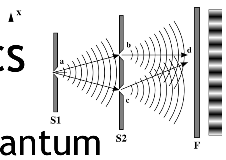
Modeling Output

- Blinking lights are cool, but hard to model
- Output is what is written on the tape at the end of a computation



Connection Machine CM-5, 1993

Liberal Arts Trivia: Physics



- The double-slit experiment in quantum mechanics demonstrates that light is inseparably both a *blank* and a *blank*. A coherent light source illuminates a thin plate with two parallel slits cut in it, and the light passing through the slits strikes a screen behind them. The light passing through both slits interferes, creating a pattern of light and dark bands.

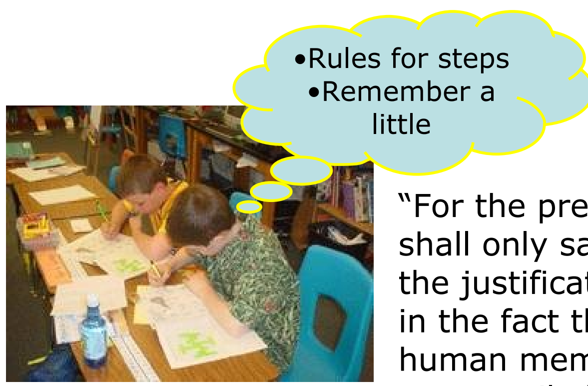
Liberal Arts Trivia: Sports

- This United Kingdom football sport dates back to a game played by ancient Greeks (called episkyros or επίσκυρος). The Cornish called it “hurling to goals” dating back to the bronze age. It was popularized by the eponymous board school after 1750. The object of the game is to carry the ball beyond the opponent's touch line (called a Try) or kick it between the goal posts. The 15-player team is allowed only “modest” padding.

Liberal Arts Trivia: Geography

- This Nordic country can claim Celsius (temperature), Nobel (dynamite), and Ericsson (telecom), as well as ABBA, Uppsala University, and Stockholm. It is the third-largest music exporter in the world, with over 800 million dollars revenue in 2007 alone, surpassed only by the US and the UK.

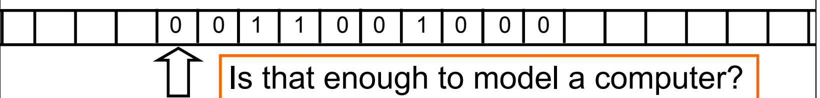
Modeling Processing (Brains)



“For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited.”
Alan Turing

Modeling Processing

- Evaluation Rules
  - Given an input on our tape, how do we evaluate to produce the output
- What do we need:
  - Read what is on the tape at the current square
  - Move the tape one square in either direction
  - Write into the current tape square

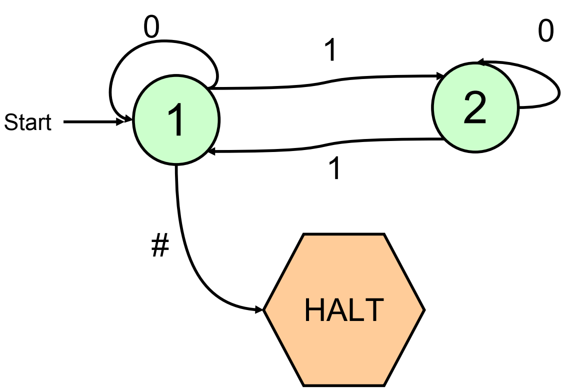


Is that enough to model a computer?

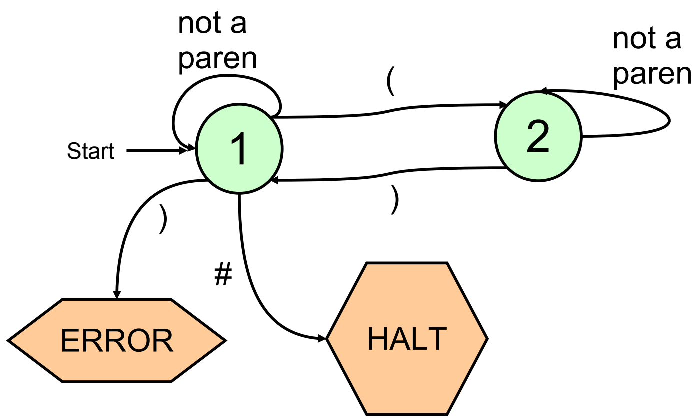
Modeling Processing

- Read, write and move is not enough
- We also need to keep track of what we are doing:
  - How do we know whether to read, write or move at each step?
  - How do we know when we're done?
- What do we need for this?

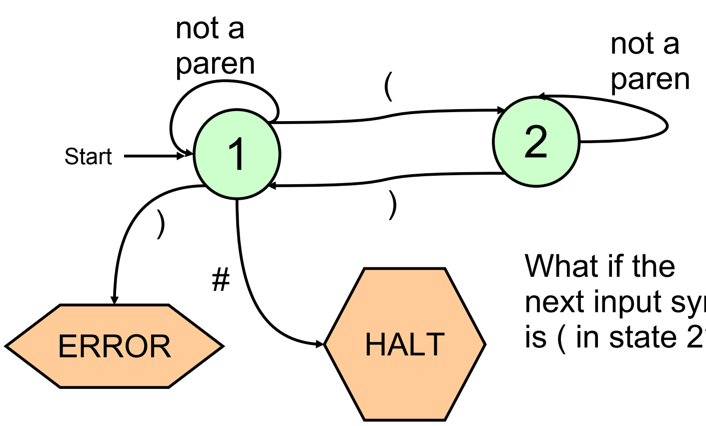
Finite State Machines



Hmmm…maybe we don't need those infinite tapes after all?

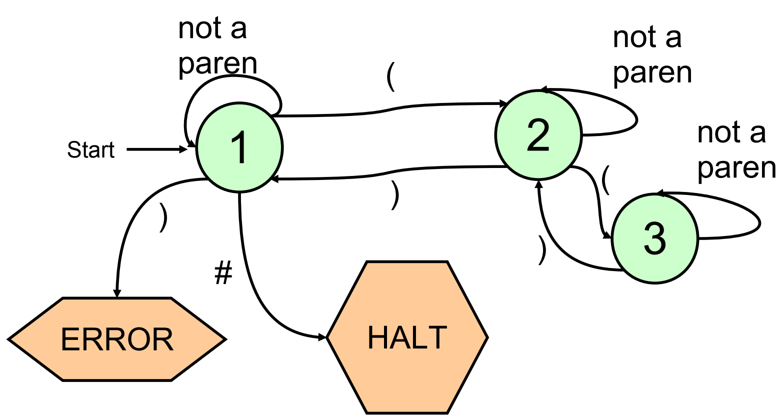


Hmmm…maybe we don't need those infinite tapes after all?

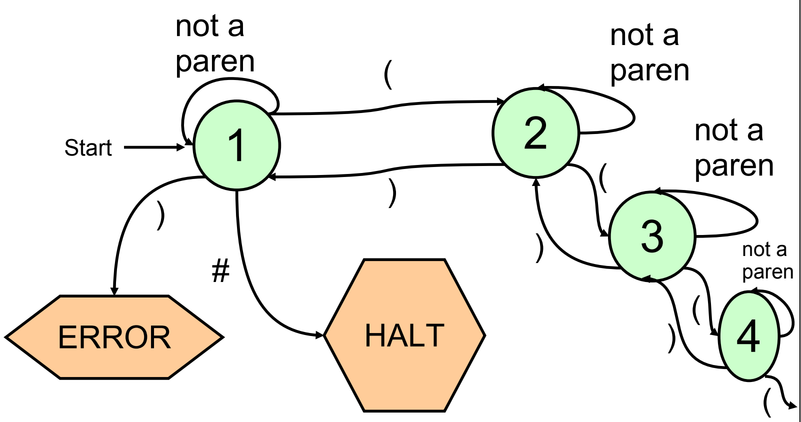


What if the next input symbol is (in state 2?

How many states do we need?



How many states do we need?



Finite State Machine

- There are lots of things we can't compute with only a finite number of states
- Solutions:
  - Infinite State Machine
    - Hard to describe and draw
  - **Add an infinite tape to the Finite State Machine**
    - We'll do this instead.

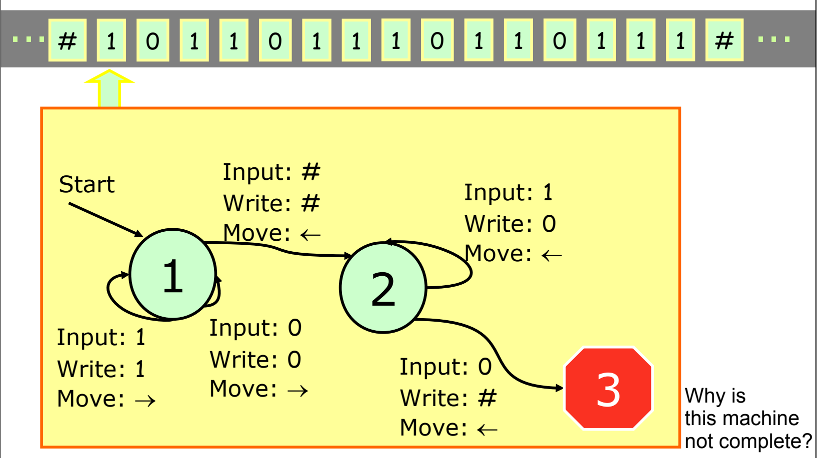
Turing's Explanation

"We have said that the computable numbers are those whose decimals are calculable by **finite** means. ... For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited."

FSM + Infinite Tape

- Start:
  - FSM in Start State
  - Input on Infinite Tape
  - Pointer to start of input
- Step:
  - Read one input symbol from tape
  - Write symbol on tape, and move L or R one square
  - Follow transition rule from current state
- Finish:
  - Transition to halt state

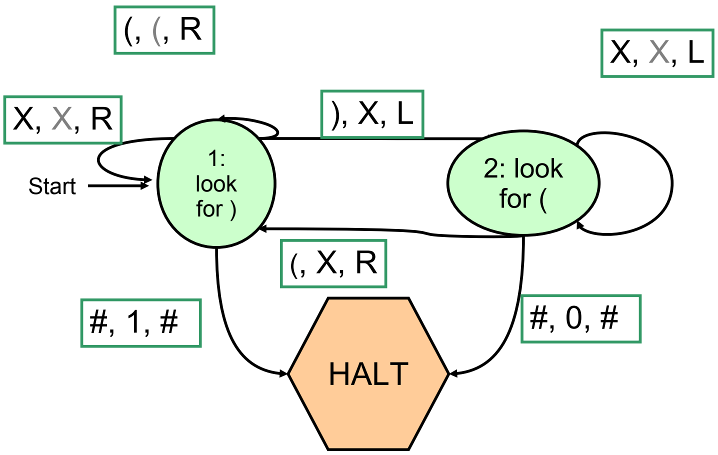
Turing Machine



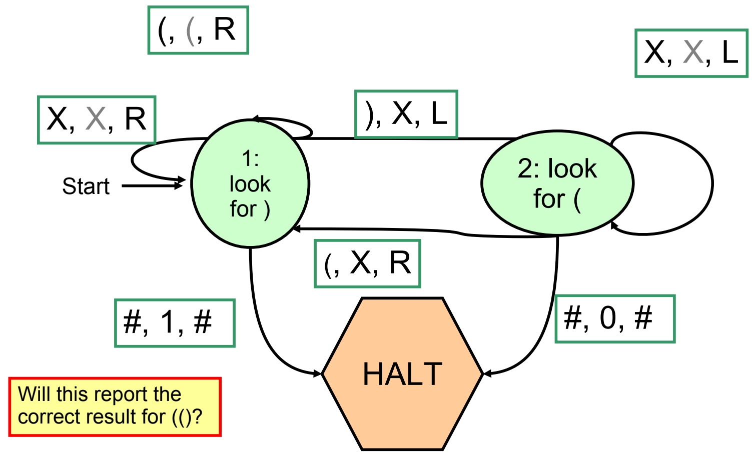
Matching Parentheses

- Find the leftmost)
  - If you don't find one, the parentheses match, write a 1 at the tape head and halt.
- Replace it with an X
- Look left for the first (
  - If you find it, replace it with an X (they matched)
  - If you don't find it, the parentheses didn't match – end write a 0 at the tape head and halt

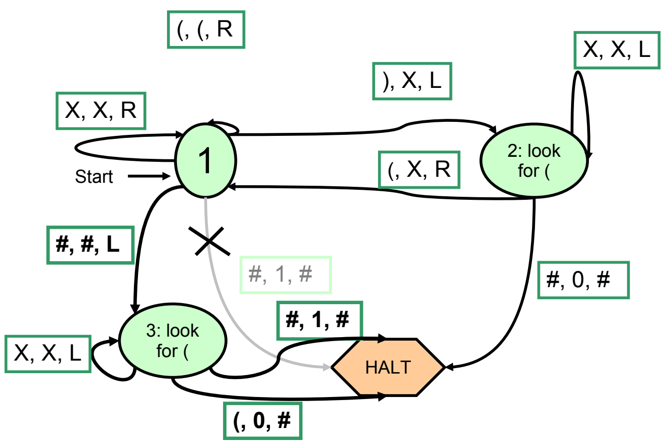
Matching Parentheses



Matching Parentheses

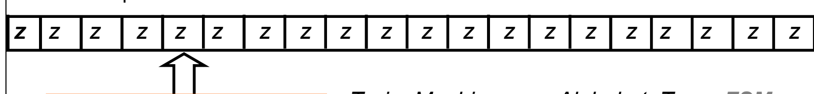


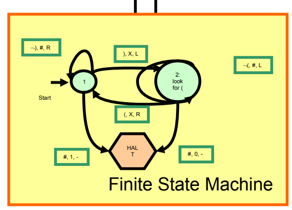
Matching Parentheses



Turing Machine

Infinite Tape





TuringMachine ::= < *Alphabet*, *Tape*, ***FSM*** >
Alphabet ::= { *Symbol** }
Tape ::= < *LeftSide*, *Current*, *RightSide* >
OneSquare ::= *Symbol* | #
Current ::= *OneSquare*
LeftSide ::= [*Square**]
RightSide ::= [*Square**]

Everything to left of *LeftSide* is **#**.
Everything to right of *RightSide* is **#**.

How well does this model your computer?

Homework

- Exam 2 Out, Due Thursday