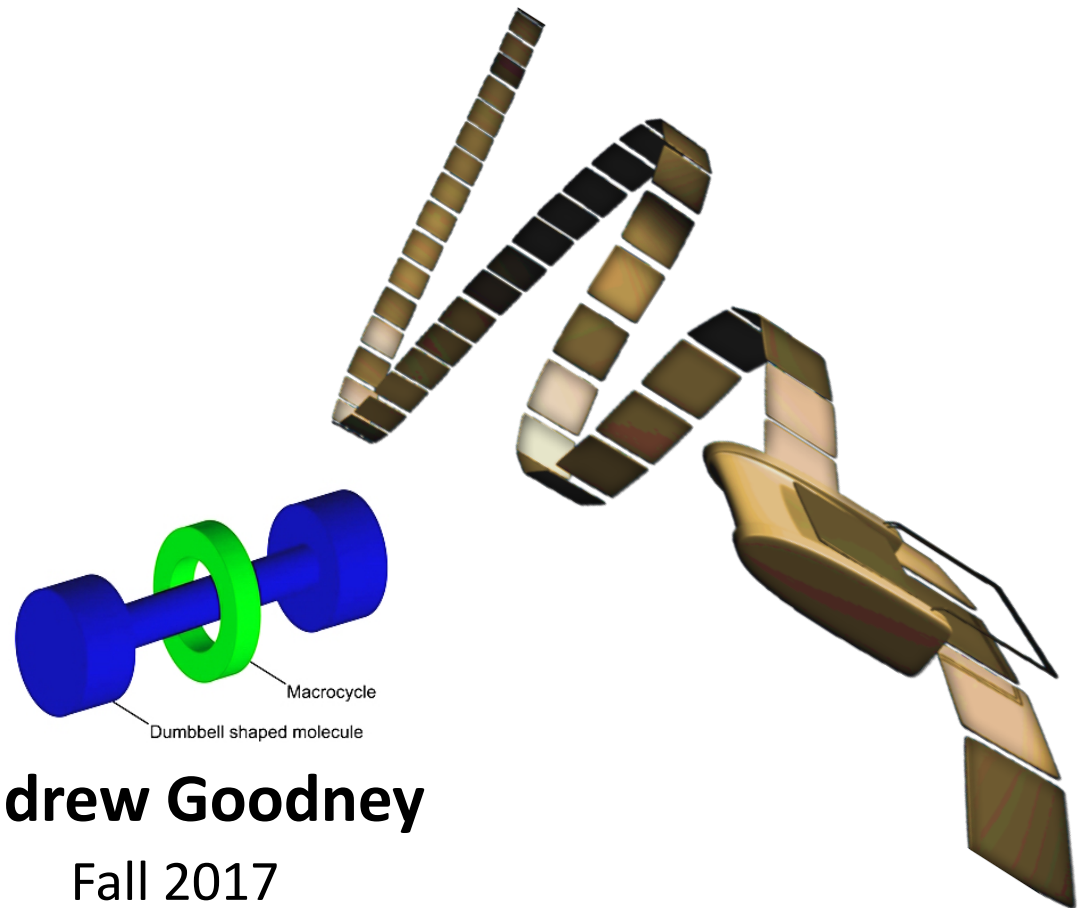
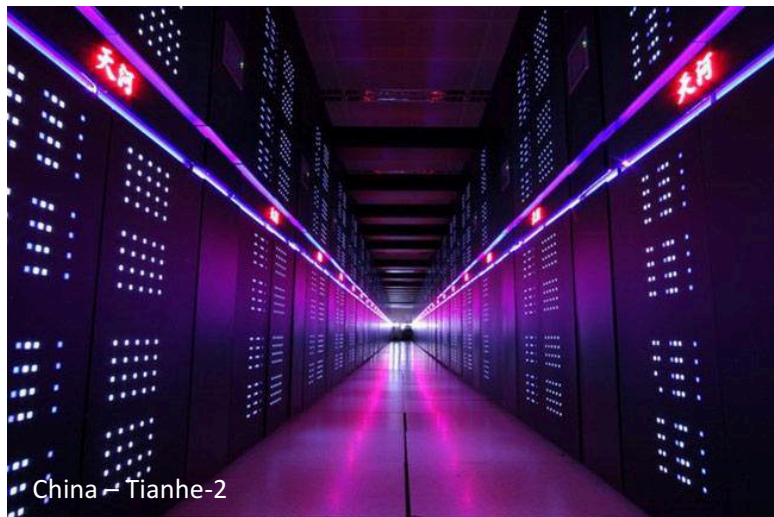


Introduction to Computer Science

CSCI 109



Andrew Goodney

Fall 2017

Schedule

Date	Topic		Assigned	Due	Quizzes/Midterm/Final
21-Aug	Introduction	What is computing, how did computers come to be?			
28-Aug	Computer architecture	How is a modern computer built? Basic architecture and assembly	HW1		
4-Sep	Labor day				
11-Sep	Data structures	Why organize data? Basic structures for organizing data		HW1	
12-Sep	Last day to drop a Monday-only class without a mark of "W" and receive a refund or change to Pass/No Pass or Audit for Session 001				
18-Sep	Data structures	Trees, Graphs and Traversals	HW2		Quiz 1 on material taught in class 8/21-8/28
25-Sep	More Algorithms/Data Structures	Recursion and run-time			
2-Oct	Complexity and combinatorics	How "long" does it take to run an algorithm.		HW2	Quiz 2 on material taught in class 9/11-9/25
6-Oct	Last day to drop a course without a mark of "W" on the transcript				
9-Oct	Algorithms and programming	(Somewhat) More complicated algorithms and simple programming constructs			Quiz 3 on material taught in class 10/2
16-Oct	Operating systems	What is an OS? Why do you need one?	HW3		Quiz 4 on material taught in class 10/9
23-Oct	Midterm	Midterm			Midterm on all material taught so far.
30-Oct	Computer networks	How are networks organized? How is the Internet organized?		HW3	
6-Nov	Artificial intelligence	What is AI? Search, planning and a quick introduction to machine learning			Quiz 5 on material taught in class 10/30
10-Nov	Last day to drop a class with a mark of "W" for Session 001				
13-Nov	The limits of computation	What can (and can't) be computed?	HW4		Quiz 6 on material taught in class 11/6
20-Nov	Robotics	Robotics: background and modern systems (e.g., self-driving cars)			Quiz 7 on material taught in class 11/13
27-Nov	Summary, recap, review	Summary, recap, review for final		HW4	Quiz 8 on material taught in class 11/20
8-Dec	Final exam 11 am - 1 pm in SAL 101				Final on all material covered in the semester



Abstract machines and theory

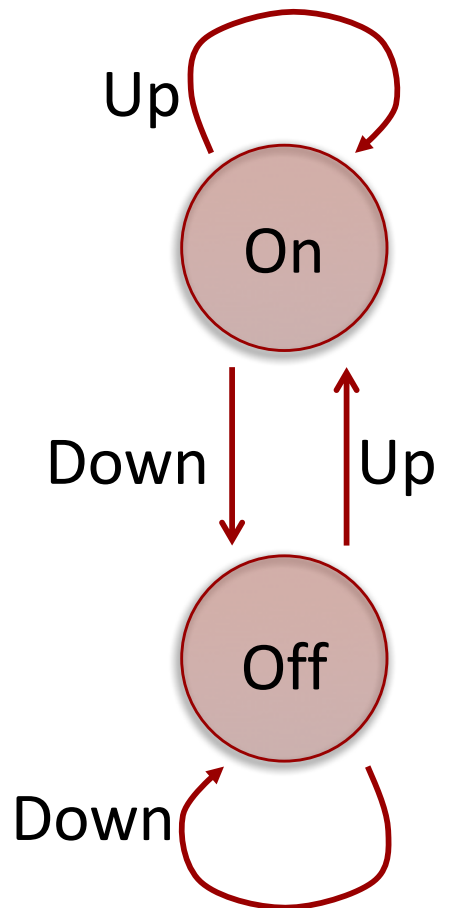
- ◆ What is theoretical computer science?
- ◆ Finite state machines
- ◆ The Turing machine
 - ◆ The Church-Turing thesis
 - ◆ The halting problem
- ◆ The analysis of algorithms
- ◆ The analysis of problems

Reading:
St. Amant Ch. 8

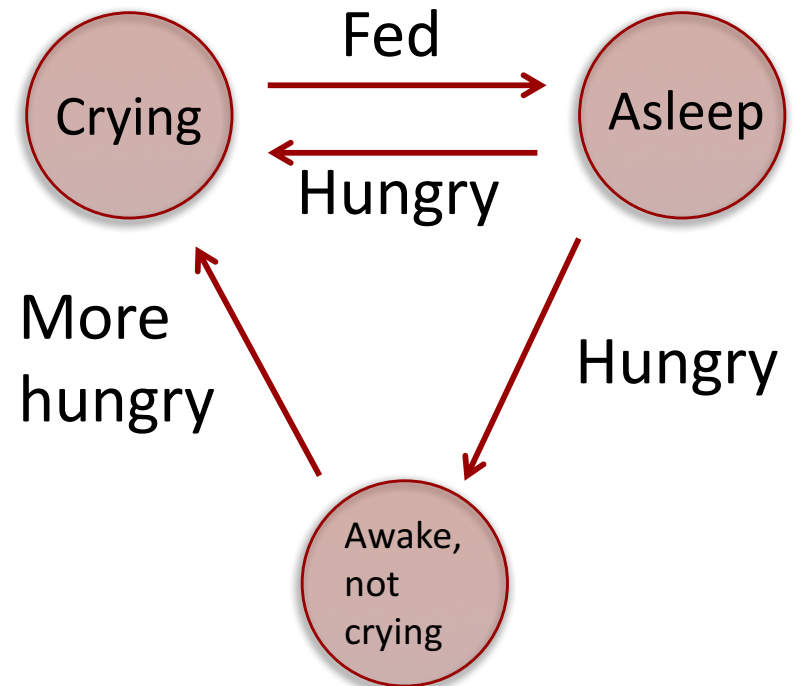
What is theoretical computer science?

- ◆ The study of what can be computed and how efficiently it can be computed
- ◆ Based on abstract mathematical models
- ◆ Insight into the behavior of a real computer...
...**any** real computer

Finite state machines

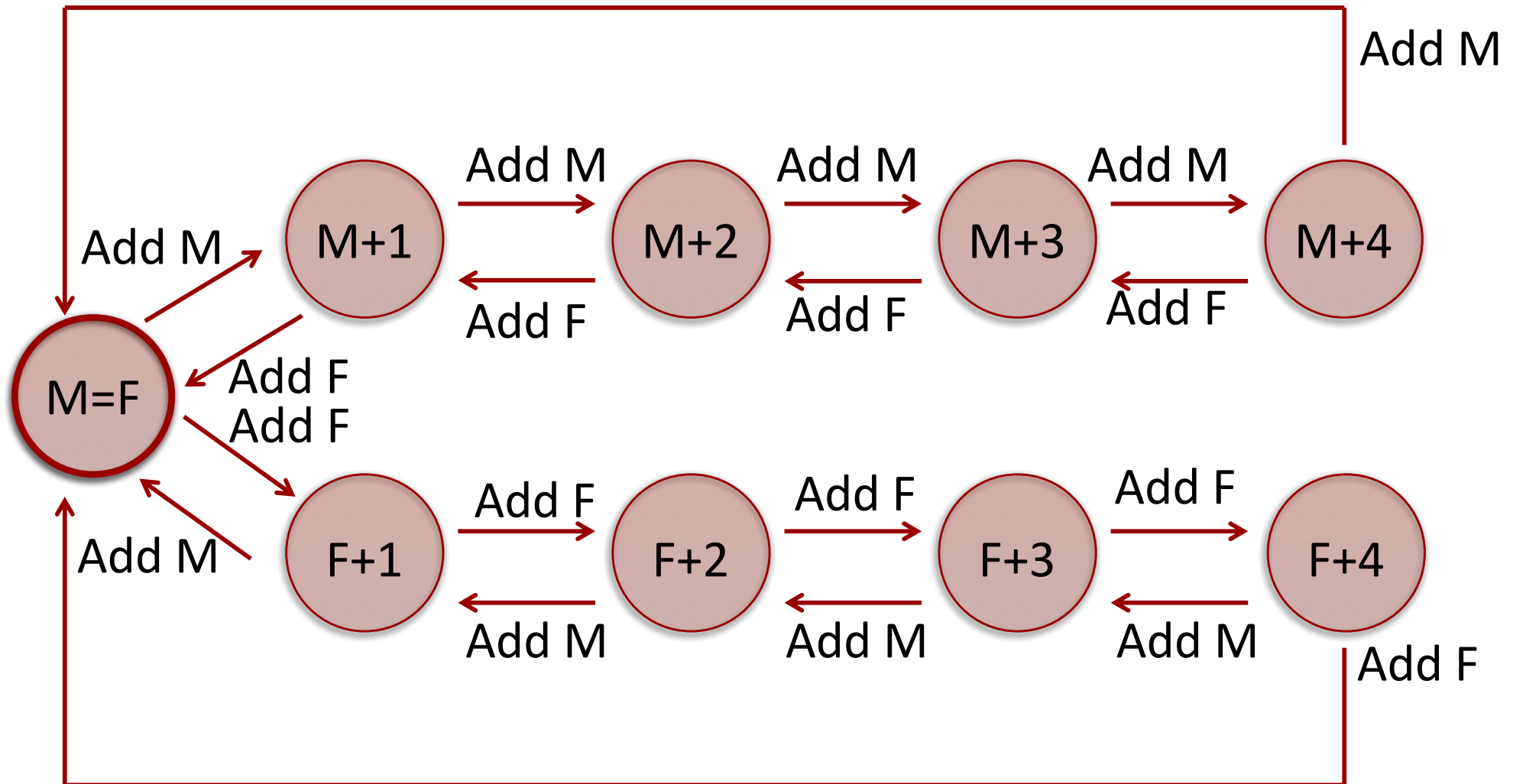


Light switch



10 day old infant

Finite state machines

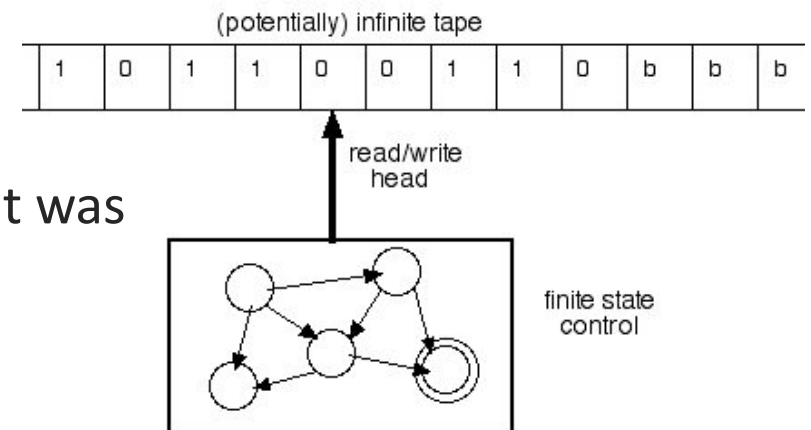
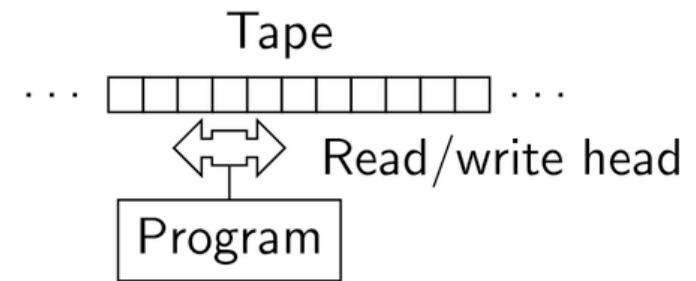


Finite state machine limitations

- ◆ If a finite state machine with n states is given $n+1$ symbols, it will revisit one state at least once
- ◆ No way to tell if revisited state is being visited for the 1st time or 5th time
- ◆ The *pigeonhole principle*
- ◆ Problems an FSM cannot solve:
 - ❖ Given a string of As and Bs, tell if the number of As and Bs are the same
 - ❖ Tell if a string is a palindrome

The Turing machine

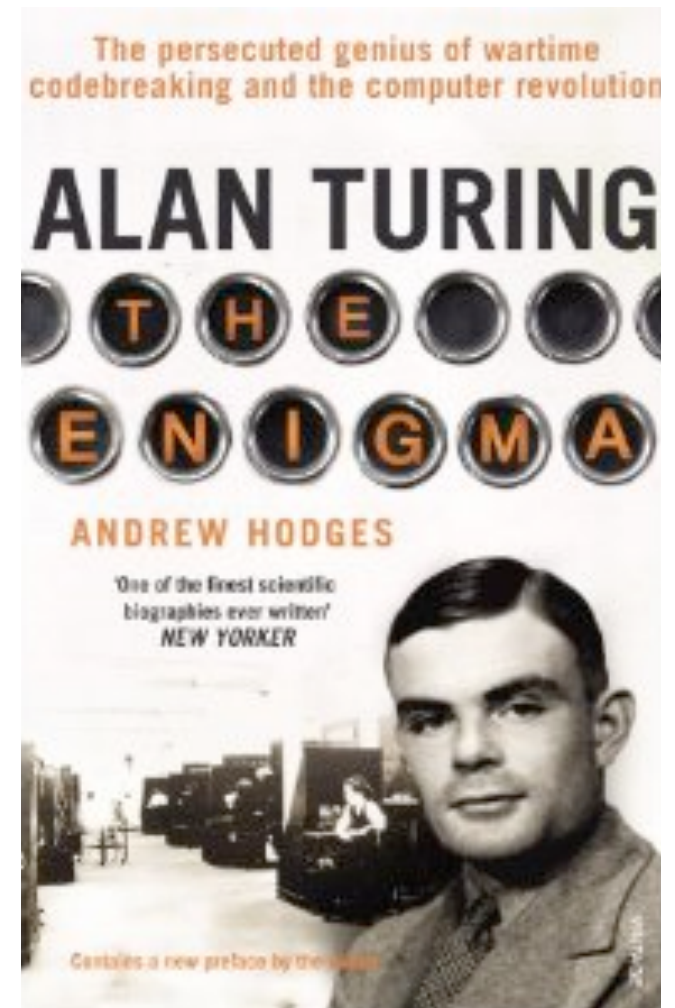
- ◆ An infinite tape, with squares marked on it
- ◆ Each square can be blank or hold a symbol
- ◆ A machine moves over the tape
- ◆ When positioned over a square, the machine can
 - ❖ Read a symbol from the tape
 - ❖ Erase a symbol from the tape
 - ❖ Write a symbol onto the tape (erasing what was on the square)
 - ❖ Do nothing
- ◆ The machine can move either one square to right or left



Simple Turing Machine with 1 symbol

<https://vimeo.com/46913004>

Alan Turing



Turing Machine Representation: Table

This machine has 4 symbols: 0, 1, _ and *

L means the head moves to the left

R means the head moves to the right

State / Input	0	1	*	_
0	*,L,1	*,L,2	*,L,0	_,L,4
1	0,L,1	*,R,3	*,L,1	_,R,5
2	*,R,3	1,L,2	*,L,2	_,R,5
3	0,R,3	1,R,3	*,*,R	_,L,0
4	accept	accept	accept	accept
5	reject	reject	reject	reject

Turing Machine Representation: Table

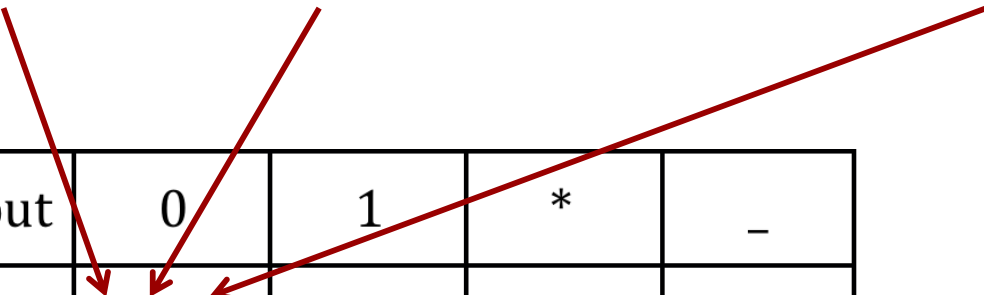
State \ Input	0	1	*	-
0	*,L,1	*,L,2	*,L,0	_,L,4
1	0,L,1	*,R,3	*,L,1	_,R,5
2	*,R,3	1,L,2	*,L,2	_,R,5
3	0,R,3	1,R,3	*,*,R	_,L,0
4	accept	accept	accept	accept
5	reject	reject	reject	reject

Turing Machine Representation: Table

State \ Input	0	1	*	_
0	*,L,1	*,L,2	*,L,0	_,L,4
1	0,L,1	*,R,3	*,L,1	_,R,5
2	*,R,3	1,L,2	*,L,2	_,R,5
3	0,R,3	1,R,3	*,*,R	_,L,0
4	accept	accept	accept	accept
5	reject	reject	reject	reject

Turing Machine Representation: Table

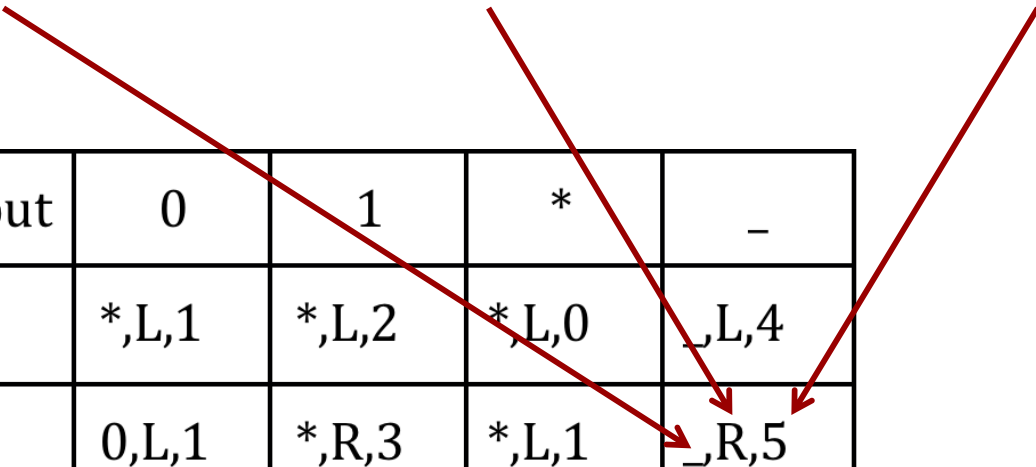
write_symbol, head_move_direction, state_update



State / Input	0	1	*	-
0	*,L,1	*,L,2	*,L,0	_,L,4
1	0,L,1	*,R,3	*,L,1	_,R,5
2	*,R,3	1,L,2	*,L,2	_,R,5
3	0,R,3	1,R,3	*,*,R	_,L,0
4	accept	accept	accept	accept
5	reject	reject	reject	reject

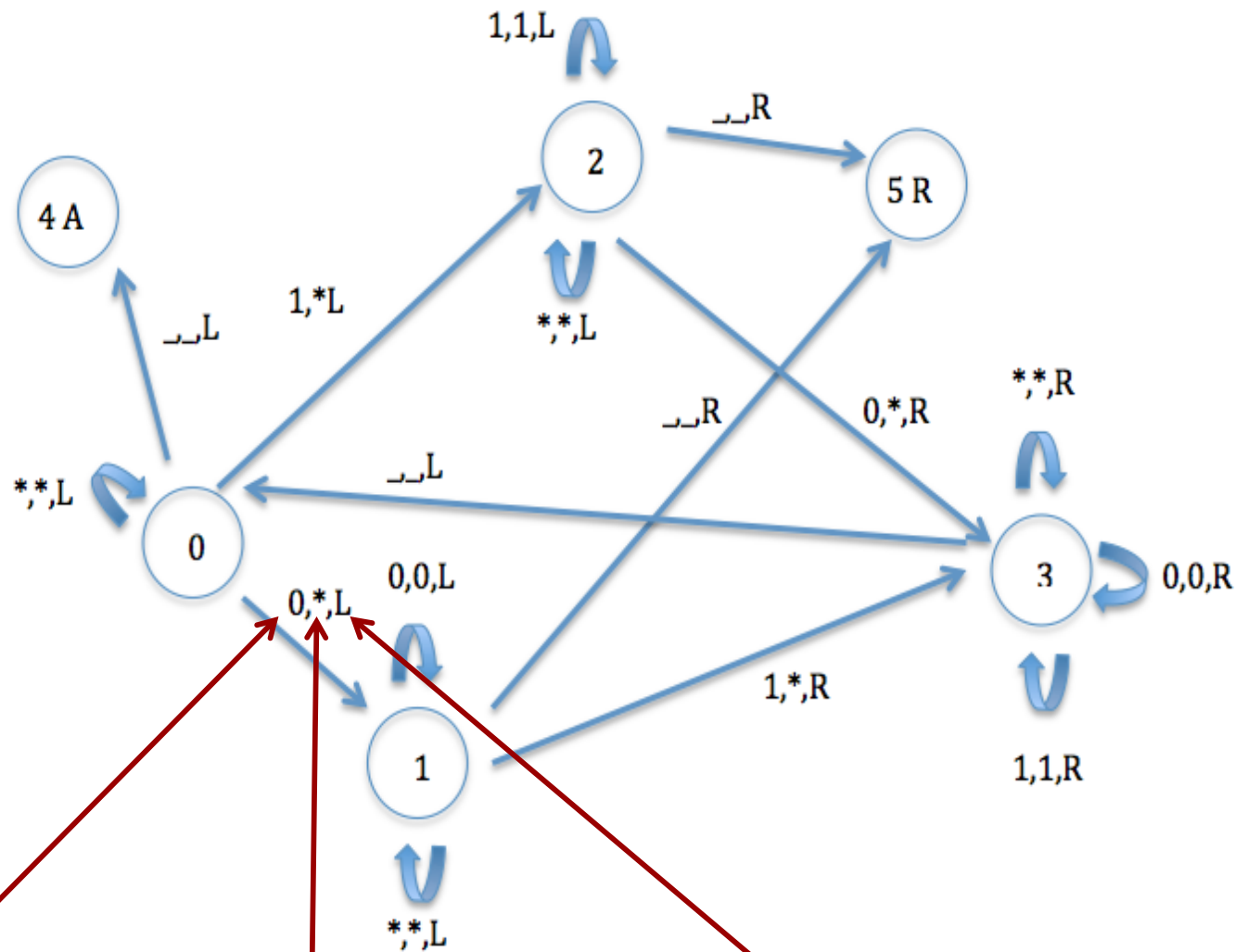
Turing Machine Representation: Table

write_symbol, head_move_direction, state_update



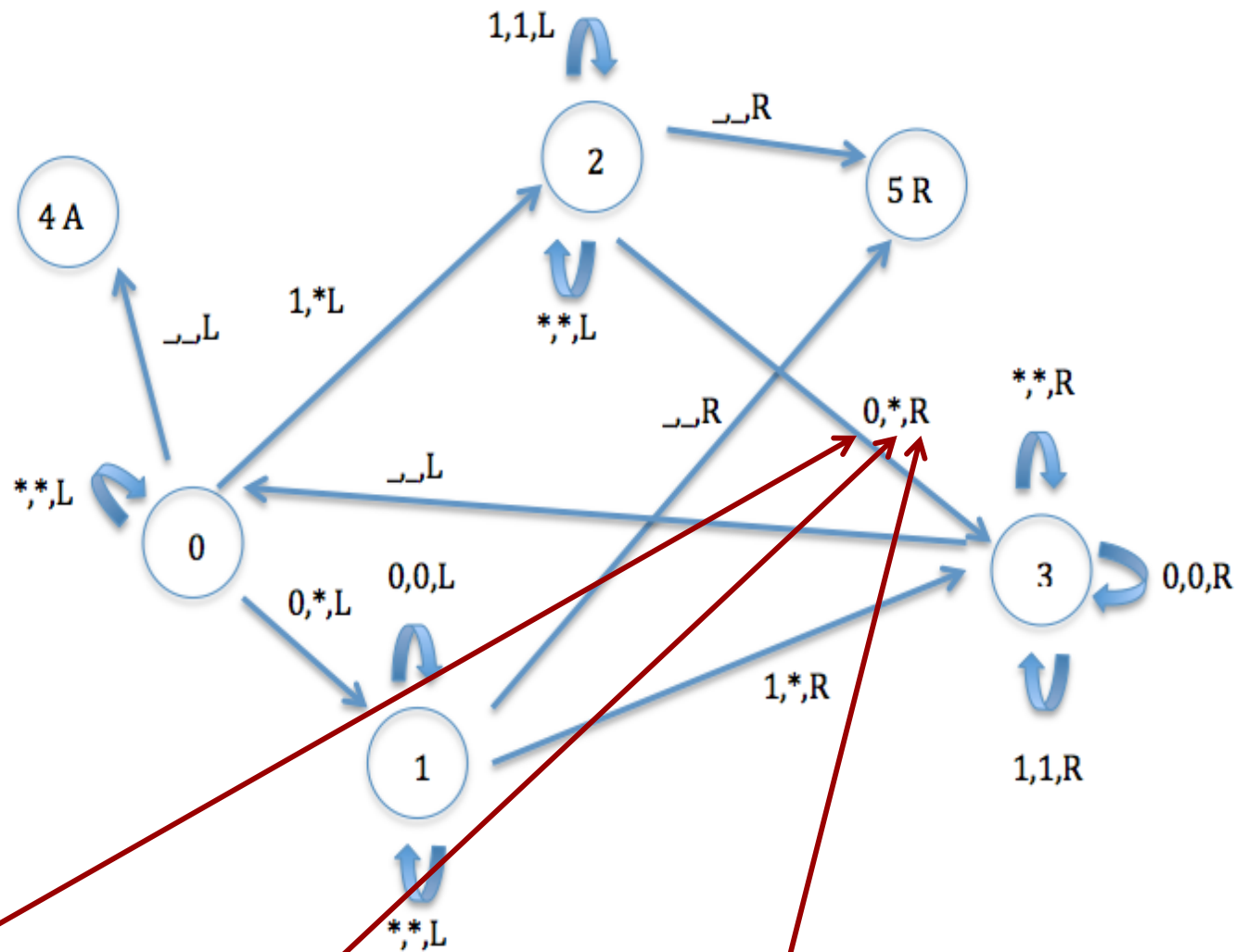
State / Input	0	1	*	-
0	*,L,1	*,L,2	*,L,0	_,L,4
1	0,L,1	*,R,3	*,L,1	_,R,5
2	*,R,3	1,L,2	*,L,2	_,R,5
3	0,R,3	1,R,3	*,*,R	_,L,0
4	accept	accept	accept	accept
5	reject	reject	reject	reject

Same Machine as a State Diagram



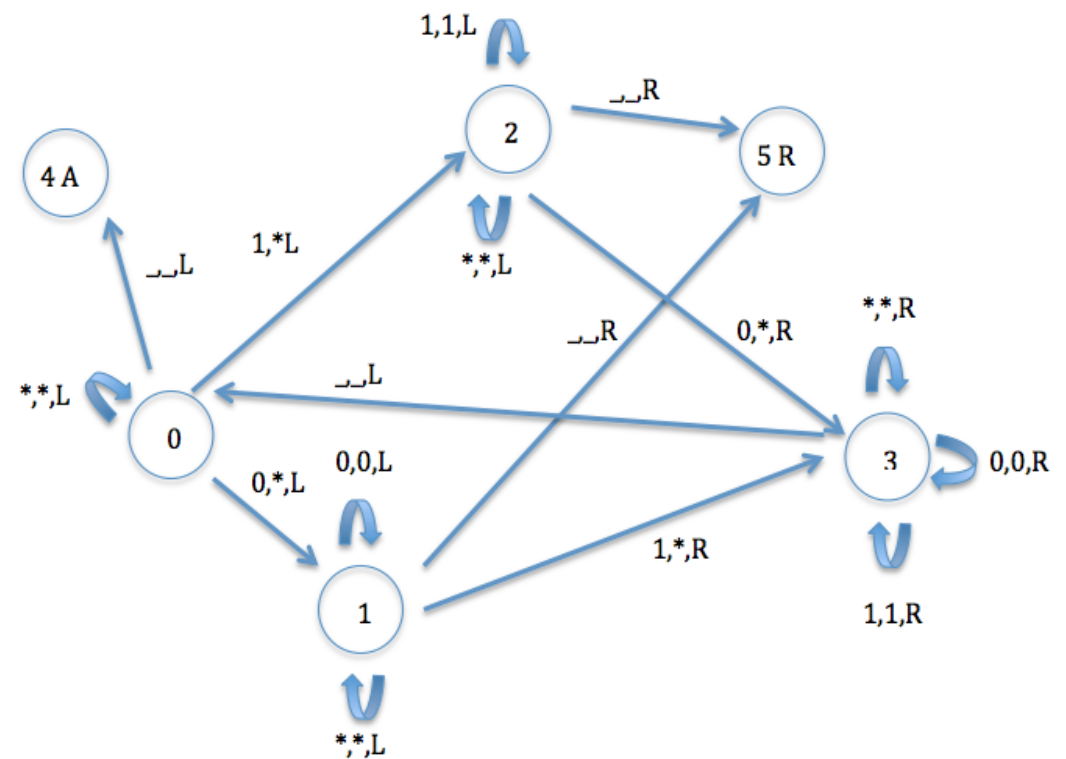
read_symbol, write_symbol, head_position_move

Same Machine as a State Diagram



read_symbol,write_symbol,head_position_move

State / Input	0	1	*	_
0	*,L,1	*,L,2	*,L,0	_,L,4
1	0,L,1	*,R,3	*,L,1	_,R,5
2	*,R,3	1,L,2	*,L,2	_,R,5
3	0,R,3	1,R,3	*,*,R	_,L,0
4	accept	accept	accept	accept
5	reject	reject	reject	reject

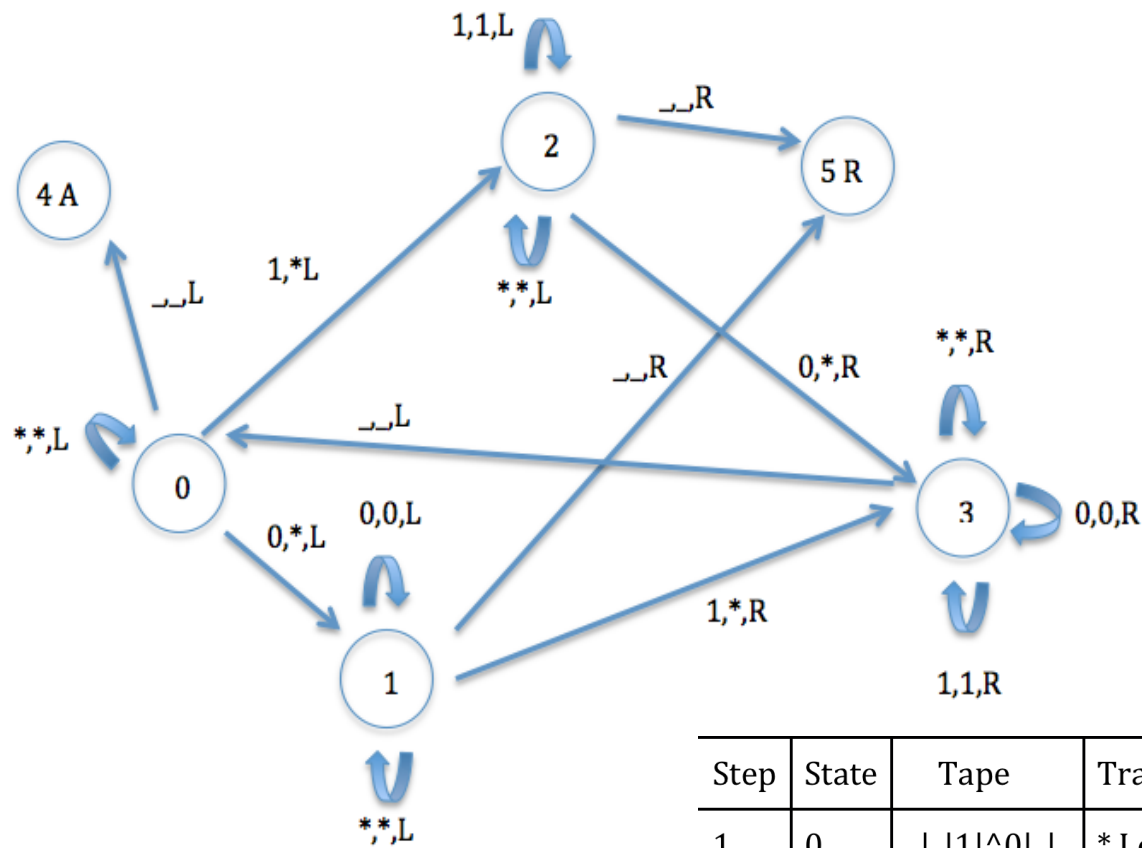


Example execution

- ◆ Assume the machine always begins in state 0 with the head positioned on the rightmost non-blank square
- ◆ The machine stops when it is in state 4 (called an accept state)
- ◆ The machine stops when it is in state 5 (called a reject state)
- ◆ How does the execution look if the tape has an input of 10 written on it at the beginning ?

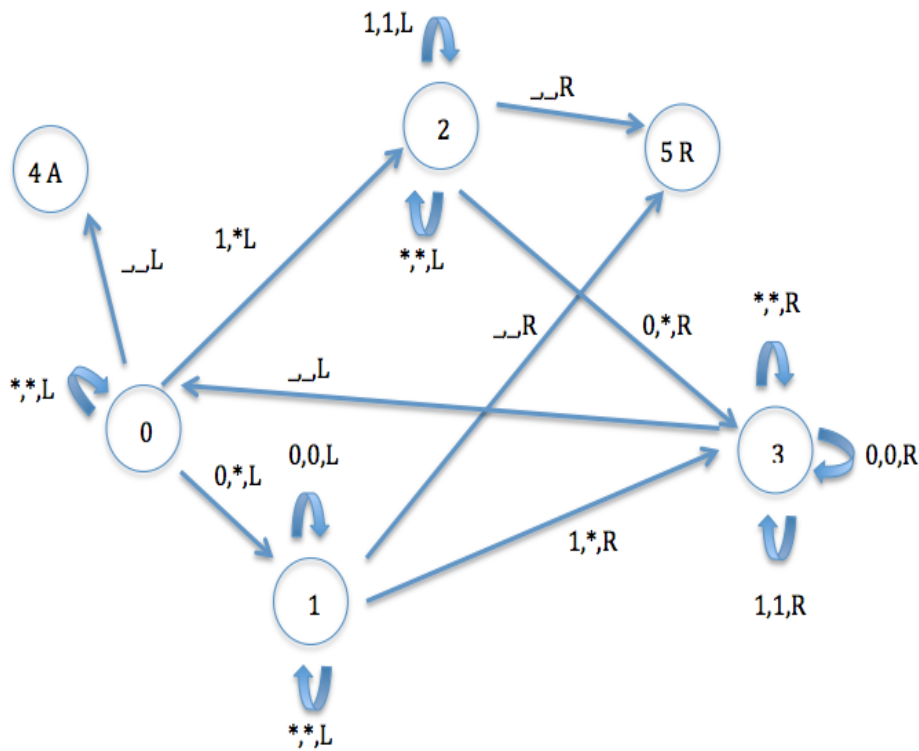
Behavior of the machine on input 10

Step	State	Tape	Transition	Note
1	0	_ _ 1 ^ 0 _ _	*,Left,1	Found 0, Write *, Move Left, Switch to State 1
2	1	_ _ ^ 1 * _ _	*,Right,3	Found 1, Write *, Move Right Switch to State 3
3	3	_ _ * ^ * _ _	*,Right,3	Found *, Write *, Move Right, Switch to State 3
4	3	_ _ * * ^ _ _	_ ,Left,0	Found _ , Write _ , Move L, switch to State 0
5	0	_ _ * ^ * _ _	*,Left,0	Found *, Write *, Move Left, Switch to State 0
6	0	_ _ ^ * * _ _	*,Left,0	Found *, Write *, Move Left, Switch to State 0
7	0	_ ^ _ * * _ _	_ ,Left,4	Found _ , Write _ , Move Left Switch to State 4
8	4	^ _ _ * * _ _	ACCEPT	ACCEPT



Behavior of the machine on input 10

Step	State	Tape	Transition	Note
1	0	<u>_</u> 1 ^0 _	*,Left,1	Found 0, Write *, Move Left, Switch to State 1
2	1	<u>_</u> ^1 ^* _	*,Right,3	Found 1, Write *, Move Right Switch to State 3
3	3	<u>_</u> * ^* _	*,Right,3	Found *, Write *, Move Right, Switch to State 3
4	3	<u>_</u> * ^* ^	_,Left,0	Found _, Write _, Move L, switch to State 0
5	0	<u>_</u> * ^* _	*,Left,0	Found *, Write *, Move Left, Switch to State 0
6	0	<u>_</u> ^* ^* _	*,Left,0	Found *, Write *, Move Left, Switch to State 0
7	0	<u>_</u> ^_ ^* ^	_,Left,4	Found _, Write _, Move Left Switch to State 4
8	4	^_ ^* ^	ACCEPT	ACCEPT



Behavior of
the machine
on input 1010

Step	State	Tape	Transition	Note
1	0	_ 1 0 1 ^0 _	*,Left,1	Found 0, Write *, Move Left, Switch to State 1
2	1	_ 1 0 ^1 ^* _	*,Right,3	Found 1, Write *, Move Right Switch to State 3
3	3	_ 1 0 ^* ^* _	*,Right,3	Found *, Write *, Move Right, Switch to State 3
4	3	_ 1 0 ^* ^* ^_	_,Left,0	Found _, Write _, Move L, switch to State 0
5	0	_ 1 0 ^* ^* _	*,Left,0	Found *, Write *, Move Left, Switch to State 0
6	0	_ 1 0 ^* ^* _	*,Left,0	Found *, Write *, Move Left, Switch to State 0
7	0	_ 1 ^0 ^* ^* _	*,Left,1	Found 0, Write *, Move Left Switch to State 1
8	1	_ ^1 ^* ^* ^* _	*,Right,3	Found 1, Write *, Move Right Switch to State 3
9	3	_ * ^* ^* ^* _	*,Right,3	Found *, Write *, Move Right Switch to State 3
10	3	_ * ^* ^* ^* ^_	*,Right,3	Found *, Write *, Move Right Switch to State 3
11	3	_ * ^* ^* ^* ^_	*,Right,3	Found *, Write *, Move Right Switch to State 3
12	3	_ * ^* ^* ^* ^_	_,Left,0	Found _, Write _, Move Left Switch to State 0
13	0	_ * ^* ^* ^* ^_	*,Left,0	Found *, Write *, Move Left Switch to State 0
14	0	_ * ^* ^* ^* ^_	*,Left,0	Found *, Write *, Move Left Switch to State 0
15	0	_ * ^* ^* ^* ^_	*,Left,0	Found *, Write *, Move Left Switch to State 0
16	0	_ ^* ^* ^* ^* ^_	*,Left,0	Found *, Write *, Move Left Switch to State 0
17	0	_ ^* ^* ^* ^* ^_	_,Left,4	Found _, Write _, Move Left Switch to State 4
18	4	_ ^_ ^* ^* ^* ^_	Accept	Accept

What does this Turing machine do?

- ◆ Checks if a binary string has the same number of 1's and 0's and enters the accept state if it does, otherwise it enters the reject state.

Turing Machine Programming

- ◆ Writing programs for a Turing machine is very tedious
- ◆ However...

The Church-Turing thesis

Turing machines are capable of solving any effectively solvable algorithmic problem

Turing machines are thus excellent models for what (all) real computers are capable of doing

Problems not 'effectively computable'

- ◆ Are there problems that are not effectively computable ?
- ◆ Yes, infinite such problems
- ◆ Could these be computable someday ?
- ◆ Perhaps, but not on a Turing-equivalent computer

A halting question

- ◆ Deciding whether a program will halt (terminate)
- ◆ Does the following program halt ?

Input **number**

While **number** is not 0

Print **number**

number = **number** - 1

The halting problem

- ◆ Deciding whether a given program will halt (terminate) on an arbitrary input
- ◆ In 1936, Turing proved that a general algorithm to the halting problem for all possible programs over all possible inputs cannot exist

The halting problem is undecidable

Proof sketch for halting problem

- ◆ Want to decide if program A halts on input a
- ◆ Need to write a program, lets call it `halter`
 - ❖ Inputs: A, a
 - ❖ Output: Yes (if program A halts on input a) or No (if program A does not halt on input a)

```
halter(A, a)
```

```
    if(some complex computation) is true
```

```
        Return Yes
```

```
    else
```

```
        Return No
```

The halting problem is undecidable

- ◆ In 1936, Turing proved that a general algorithm to the halting problem for all possible programs over all possible inputs cannot exist
- ◆ In other words, a general version of `halter` (one that works for all programs and their inputs) does not exist

Proof by contradiction

- ◆ Assume that a general version of `halter` exists
- ◆ Show that this assumption leads to a contradiction
- ◆ Invent a new program called `clever`

```
clever(program, input)
```

```
    result = halter(program, input)
```

```
    if result is No
```

```
        return Yes
```

```
    else
```

```
        loop forever
```

Proof by contradiction

- ◆ What does `clever` do when given itself as input?

```
clever(program, input)
```

```
    result=halter(program, input)
```

```
    if result is No
```

```
        return Yes
```

```
    else
```

```
        loop forever
```


Proof by contradiction

- ◆ If halter says that clever halts, then clever loops forever (which means it doesn't halt)
- ◆ If halter says that clever does not halt then clever halts and returns Yes

```
clever(program, input)
result=halter(program, input)
    if result is No
        return Yes
    else
        loop forever
```

Conclusion: The program
halter cannot exist

Russell's Paradox

- ◆ In a hypothetical small town:
 - ❖ There is one male barber
 - ❖ The barber shaves all those and only those men in town who do not shave themselves
 - ❖ All men stay clean shaven either by shaving themselves or going to the barber
- ◆ Who shaves the barber?
- ◆ More generally:
 - ❖ Let R be the set of all sets that are not members of themselves
 - ❖ If R is not a member of itself, then its definition dictates that it must contain itself, and if it contains itself, then it contradicts its own definition as the set of all sets that are not members of themselves

Analysis of problems

- ◆ Study of algorithms illuminates the study of classes of problems
- ◆ If a polynomial time algorithm exists to solve a problem then the problem is called *tractable*
- ◆ If a problem cannot be solved by a polynomial time algorithm then it is called *intractable*
- ◆ This divides problems into three groups:
 - ❖ Problems with known polynomial time algorithms
 - ❖ Problems that are proven to have no polynomial-time algorithm
 - ❖ Problems with no known polynomial time algorithm but not yet proven to be intractable

Tractable and Intractable

◆ Tractable problems (**P**)

- ❖ Sorting a list
- ❖ Searching an unordered list
- ❖ Finding a minimum spanning tree in a graph

◆ Intractable

- ❖ Listing all permutations (all possible orderings) of n numbers

◆ Might be (in)tractable

- ❖ Subset sum: given a set of numbers, is there a subset that adds up to a given number?
- ❖ Travelling salesperson: n cities, $n!$ routes, find the shortest route

These problems have no known polynomial time solution

However no one has been able to prove that such a solution does not exist

Tractability, Intractability, Undecidability

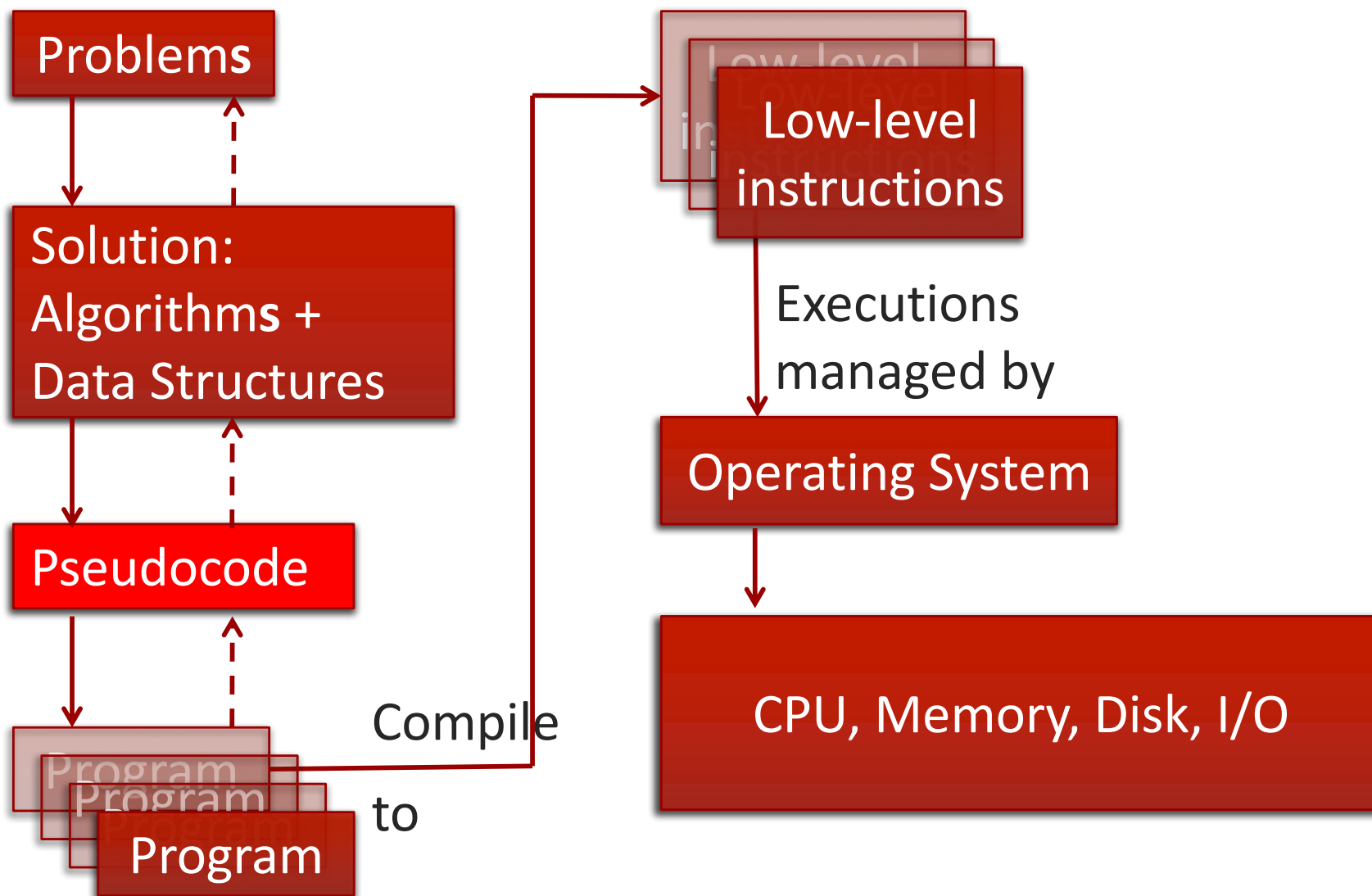
- ◆ ‘Properties of problems’ (*NOT* ‘properties of algorithms’)
- ◆ Tractable: problem can be solved by a polynomial time algorithm (or something more efficient)
- ◆ Intractable: problem cannot be solved by a polynomial time algorithm (all solutions are proven to be more inefficient than polynomial time)
- ◆ Unknown: not known if the problem is tractable or intractable (no known polynomial time solution, no proof that a polynomial time solution does not exist)
- ◆ Undecidable: *decision problem* for which there is (provably) no algorithmic solution on a Turing machine
 - ❖ **Non-existence**
 - ❖ **Not a matter of efficiency**

Abstract machines and theory

- ◆ What is theoretical computer science?
- ◆ Finite state machines
- ◆ The Turing machine
 - ◆ The Church-Turing thesis
 - ◆ The halting problem
- ◆ The analysis of algorithms
- ◆ The analysis of problems

Reading:
St. Amant Ch. 8

Overview



Quiz #6

<http://bit.ly/2yXjboq>

