What is Computation?

Boolean Circuits and Finite State Machines

Carlotta Pavese

10.17.13

Outline

Review

Computation and Functionalism

- Functionalist understanding of minds: what they do

Review

Computation and Functionalism

- Functionalist understanding of minds: what they do
  - Abstracts from materials used to do those things

Review

Computation and Functionalism

- Functionalist understanding of minds: what they do
  - Abstracts from materials used to do those things
- Computer science offers an account of computation that has exactly that abstract character

Review

Computation and Functionalism

- Functionalist understanding of minds: what they do
  - Abstracts from materials used to do those things
- Computer science offers an account of computation that has exactly that abstract character
- This parallel is not an accident!

Review

Computation and Functionalism

- Functionalist understanding of minds: what they do
  - Abstracts from materials used to do those things
- Computer science offers an account of computation that has exactly that abstract character
- This parallel is not an accident!
  - Putnam had computation in mind when he was advocating for functionalism!

Review

Computation and Functionalism

- Functionalist understanding of minds: what they do
  - Abstracts from materials used to do those things
- Computer science offers an account of computation that has exactly that abstract character
- This parallel is not an accident!
  - Putnam had computation in mind when he was advocating for functionalism!

Review

Computation and Functionalism

- Functionalist understanding of minds: what they do
  - Abstracts from materials used to do those things
- Computer science offers an account of computation that has exactly that abstract character
- This parallel is not an accident!
  - Putnam had computation in mind when he was advocating for functionalism!

Our Big Question

So what exactly is computation and are all mental activities just computation?

Today's Lecture

Part 1: Boolean Circuits

Our Big Question

So what exactly is computation and are all mental activities just computation?

Today's Lecture

Part 1: Boolean Circuits

Our Big Question

So what exactly is computation and are all mental activities just computation?

- First essential building block of computers:

Today's Lecture

Part 1: Boolean Circuits

Our Big Question

So what exactly is computation and are all mental activities just computation?

- First essential building block of computers:
  - Boolean Circuits

Today's Lecture

Part 1: Boolean Circuits

Our Big Question

So what exactly is computation and are all mental activities just computation?

- First essential building block of computers:
  - Boolean Circuits

Part of the Big Question

Today's Lecture

Part 1: Boolean Circuits

Our Big Question

So what exactly is computation and are all mental activities just computation?

- First essential building block of computers:
  - Boolean Circuits

Part of the Big Question

- What exactly are Boolean circuits capable of?

Today's Lecture

Part 1: Boolean Circuits

Our Big Question

So what exactly is computation and are all mental activities just computation?

- First essential building block of computers:
  - Boolean Circuits

Part of the Big Question

- What exactly are Boolean circuits capable of?
- Which mental activities can they perform?

Today's Lecture

Part 1: Boolean Circuits

Our Big Question

So what exactly is computation and are all mental activities just computation?

- First essential building block of computers:
  - Boolean Circuits

Part of the Big Question

- What exactly are Boolean circuits capable of?
- Which mental activities can they perform?
- Which mental activities can't they perform?

Today's Lecture

Part 2: Finite State Machines

Our Big Question

So what exactly is computation and are all mental activities just computation?

Today's Lecture

Part 2: Finite State Machines

Our Big Question

So what exactly is computation and are all mental activities just computation?

- Second essential building block of computers:

Today's Lecture

Part 2: Finite State Machines

Our Big Question

So what exactly is computation and are all mental activities just computation?

- Second essential building block of computers:
  - Finite State Machine (FSM)

Today's Lecture

Part 2: Finite State Machines

Our Big Question

So what exactly is computation and are all mental activities just computation?

- Second essential building block of computers:
  - Finite State Machine (FSM)

Part of the Big Question

Today's Lecture

Part 2: Finite State Machines

Our Big Question

So what exactly is computation and are all mental activities just computation?

- Second essential building block of computers:
  - Finite State Machine (FSM)

Part of the Big Question

- What exactly are FSMs capable of?

Today's Lecture

Part 2: Finite State Machines

Our Big Question

So what exactly is computation and are all mental activities just computation?

- Second essential building block of computers:
  - Finite State Machine (FSM)

Part of the Big Question

- What exactly are FSMs capable of?
- Which mental activities can they perform?

Today's Lecture

Part 2: Finite State Machines

Our Big Question

So what exactly is computation and are all mental activities just computation?

- Second essential building block of computers:
  - Finite State Machine (FSM)

Part of the Big Question

- What exactly are FSMs capable of?
- Which mental activities can they perform?
- Which mental activities can't they perform?

Preview

Today's Class

1. When it comes to 'crunching bits', Boolean Circuits can do anything!

Preview

Today's Class

1. When it comes to 'crunching bits', Boolean Circuits can do anything!
2. Some basic tasks require more than crunching bits and Boolean Circuits can't do them

Preview

Today's Class

1. When it comes to 'crunching bits', Boolean Circuits can do anything!
2. Some basic tasks require more than crunching bits and Boolean Circuits can't do them
3. Minds can perform these tasks, so we must search on

Preview

Today's Class

1. When it comes to 'crunching bits', Boolean Circuits can do anything!
2. Some basic tasks require more than crunching bits and Boolean Circuits can't do them
3. Minds can perform these tasks, so we must search on
4. Finite State Machines can perform these tasks

Preview

Today's Class

1. When it comes to 'crunching bits', Boolean Circuits can do anything!
2. Some basic tasks require more than crunching bits and Boolean Circuits can't do them
3. Minds can perform these tasks, so we must search on
4. Finite State Machines can perform these tasks
   - They have a new component: a register

Preview

Today's Class

1. When it comes to 'crunching bits', Boolean Circuits can do anything!
2. Some basic tasks require more than crunching bits and Boolean Circuits can't do them
3. Minds can perform these tasks, so we must search on
4. Finite State Machines can perform these tasks
   - They have a new component: a register
5. We'll find things minds can do that FSMs can't

Preview

Today's Class

1. When it comes to 'crunching bits', Boolean Circuits can do anything!
2. Some basic tasks require more than crunching bits and Boolean Circuits can't do them
3. Minds can perform these tasks, so we must search on
4. Finite State Machines can perform these tasks
   - They have a new component: a register
5. We'll find things minds can do that FSMs can't
6. Computers consist of Boolean Circuits, an FSM and another component we'll discuss next week

Outline

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers
  - There are rules for correct calculation

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers
  - There are rules for correct calculation
- Boolean Logic: $(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers
  - There are rules for correct calculation
- Boolean Logic: $(A \wedge B) \wedge C = A \wedge (B \wedge C)$
  - Variables A, B, C can be any true/false claim

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers
  - There are rules for correct calculation
- Boolean Logic: $(\mathrm{A} \wedge \mathrm{B}) \wedge \mathrm{C} = \mathrm{A} \wedge (\mathrm{B} \wedge \mathrm{C})$
  - Variables $\mathrm{A}, \mathrm{B}, \mathrm{C}$ can be any true/false claim
    - E.g. A can be: *Obama ate yesterday*

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers
  - There are rules for correct calculation
- Boolean Logic: $(\mathrm{A} \wedge \mathrm{B}) \wedge \mathrm{C} = \mathrm{A} \wedge (\mathrm{B} \wedge \mathrm{C})$
  - Variables $\mathrm{A}, \mathrm{B}, \mathrm{C}$ can be any true/false claim
    - E.g. A can be: *Obama ate yesterday*
  - $\wedge, \vee, \neg$ operate on truth/falsity of claims

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers
  - There are rules for correct calculation
- Boolean Logic: $(A \wedge B) \wedge C = A \wedge (B \wedge C)$
  - Variables A, B, C can be any true/false claim
    - E.g. A can be: *Obama ate yesterday*
  - $\wedge, \vee, \neg$ operate on truth/falsity of claims
    - Just as *and*, *or* and *not* combine statements

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers
  - There are rules for correct calculation
- Boolean Logic: $(A \wedge B) \wedge C = A \wedge (B \wedge C)$
  - Variables A, B, C can be any true/false claim
    - E.g. A can be: *Obama ate yesterday*
  - $\wedge, \vee, \neg$ operate on truth/falsity of claims
    - Just as *and*, *or* and *not* combine statements
  - The way they operate give rise to certain laws

Boolean Logic

The Basics

- Algebra: $(x + y) + z = x + (y + z)$
  - Variables x, y, z can be any number
  - $+, \times, -$, etc. operate on numbers
  - There are rules for correct calculation
- Boolean Logic: $(A \wedge B) \wedge C = A \wedge (B \wedge C)$
  - Variables A, B, C can be any true/false claim
    - E.g. A can be: *Obama ate yesterday*
  - $\wedge, \vee, \neg$ operate on truth/falsity of claims
    - Just as *and*, *or* and *not* combine statements
  - The way they operate give rise to certain laws
  - These laws provide rules for correct reasoning

Boolean Logic

The Boolean Connectives and Laws of Thought

Negation ($\neg$): *Not*

A	$\neg$A
TRUE	FALSE
FALSE	TRUE

Conjunction ($\wedge$): *And*

A	B	A $\wedge$ B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

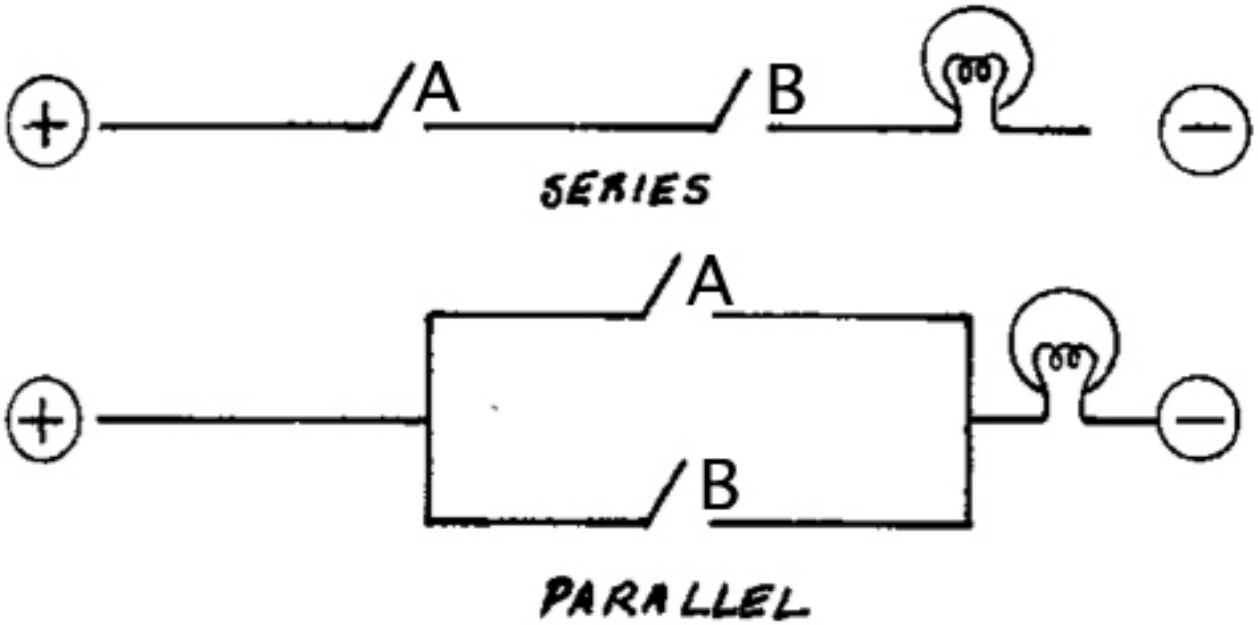
Disjunction ($\vee$): *Or*

A	B	A $\vee$ B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- $\neg$ flips truth/falsity
- $\wedge$ takes worstvalue
- $\vee$ takes bestvalue

Shannon's Insight

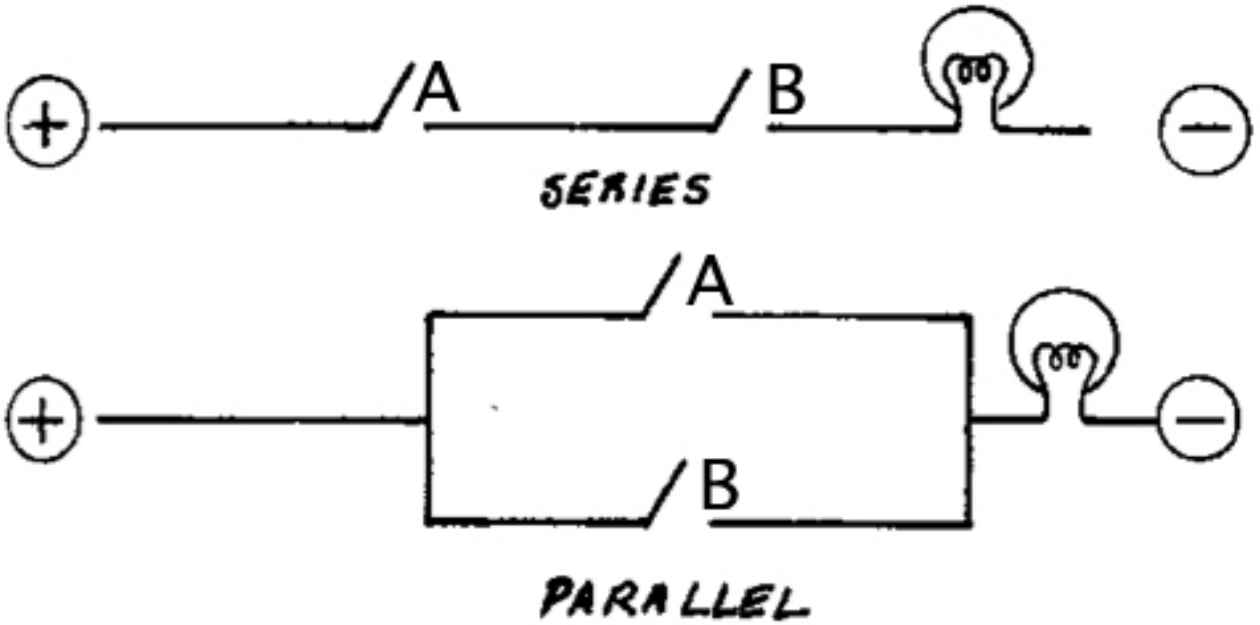
Electrical Boolean Circuits



1. Boole's variables become switches
2. Truth becomes current flowing (closed)
3. Falsity becomes current not flowing (open)

Shannon's Insight

Electrical Boolean Circuits



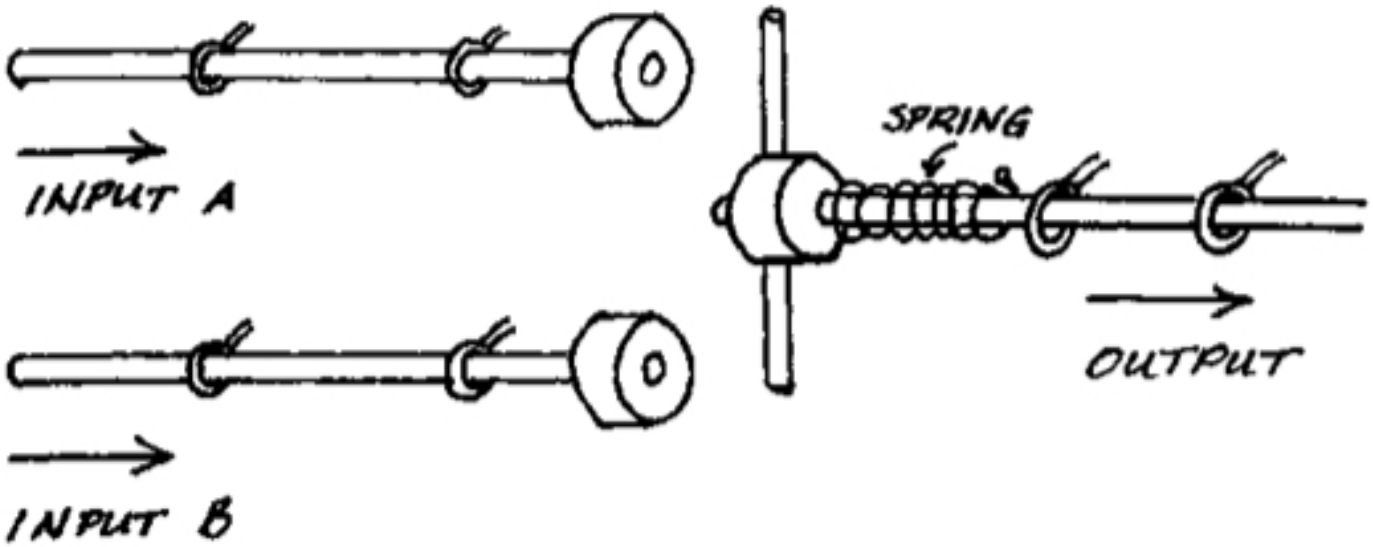
$A \wedge B$

$A \vee B$

1. Boole's variables become switches
2. Truth becomes current flowing (closed)
3. Falsity becomes current not flowing (open)

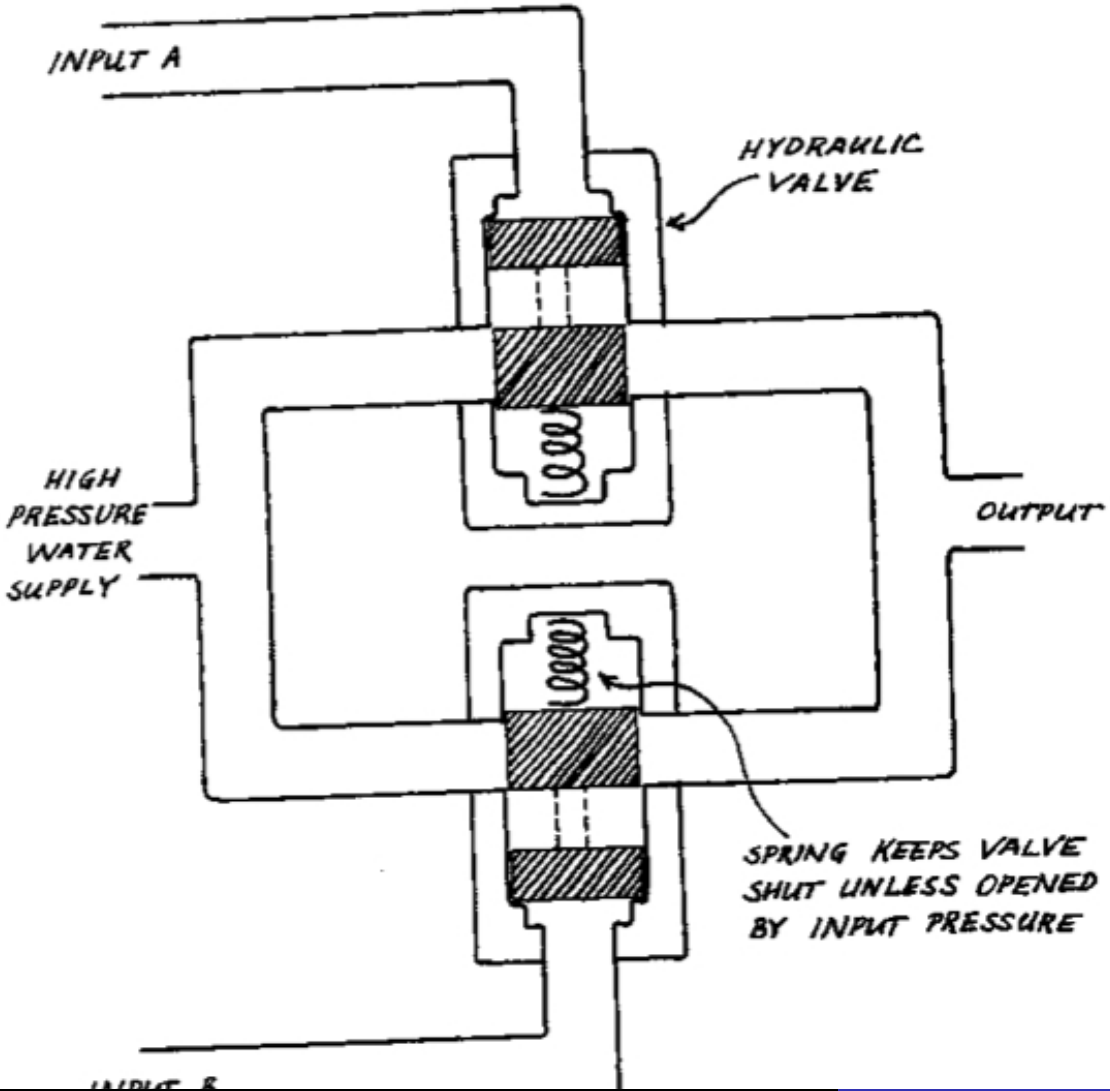
Shannon's Insight

Mechanical OR Circuit



Shannon's Insight

Isn't Specific to Electricity: Hydraulic OR



What is a Bit?

A Binary Sigal

What's Common?

- $\vee$ combines truth-values (True/False)

What is a Bit?

A Binary Sigal

What's Common?

- $\vee$ combines truth-values (True/False)
- Electrical OR combines binary currents (High/Low)

What is a Bit?

A Binary Sigal

What's Common?

- $\vee$ combines truth-values (True/False)
- Electrical OR combines binary currents (High/Low)
- Mechanical OR combines binary stick positions (Moved Right/Not Moved Right)

What is a Bit?

A Binary Sigal

What's Common?

- $\vee$ combines truth-values (True/False)
- Electrical OR combines binary currents (High/Low)
- Mechanical OR combines binary stick positions (Moved Right/Not Moved Right)
- Hydraulic OR combines binary water pressures (Present/Absent)

What is a Bit?

A Binary Sigal

What's Common?

- $\vee$ combines truth-values (True/False)
- Electrical OR combines binary currents (High/Low)
- Mechanical OR combines binary stick positions (Moved Right/Not Moved Right)
- Hydraulic OR combines binary water pressures (Present/Absent)

- Abstraction: these are all binary signals!

What is a Bit?

A Binary Sigal

What's Common?

- $\vee$ combines truth-values (True/False)
- Electrical OR combines binary currents (High/Low)
- Mechanical OR combines binary stick positions (Moved Right/Not Moved Right)
- Hydraulic OR combines binary water pressures (Present/Absent)

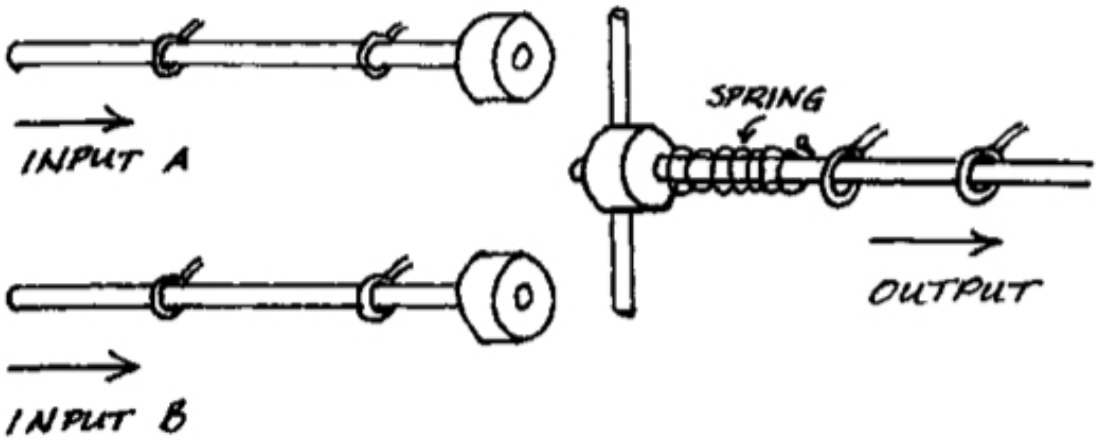
- Abstraction: these are all binary signals!
- **Definition**: A bit is any binary signal

Thinking of Bits as 1's and 0's

Forget About The Details, Example of OR

Disjunction and Mechanical OR

A	B	A ∨ B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

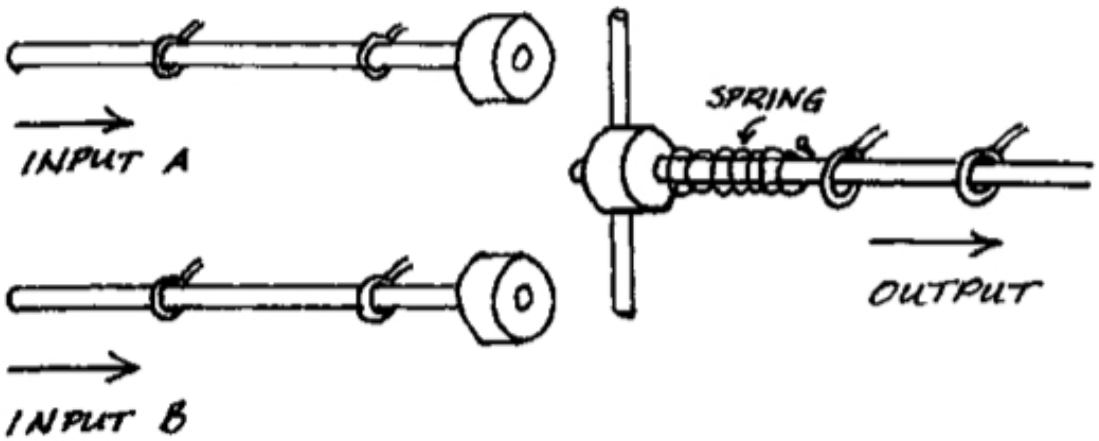


Thinking of Bits as 1's and 0's

Forget About The Details, Example of OR

Disjunction and Mechanical OR

A	B	A ∨ B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE



OR in Bits

A	B	A OR B
1	1	1
1	0	1
0	0	1
0	0	0

Forget what 1 & 0 really are and how those things are combined!

Logic Functions

Any Way of Crunching Bits

Logic Functions

1. A logic function is any way of taking in bits and responding with bits
2. Any way of taking in 1's and 0's and responding with 1's and 0's

Logic Functions

Any Way of Crunching Bits

Logic Functions

1. A logic function is any way of taking in bits and responding with bits
2. Any way of taking in 1's and 0's and responding with 1's and 0's

- Carrying out a logic function: *crunching bits*

Logic Functions

Any Way of Crunching Bits

Logic Functions

1. A logic function is any way of taking in bits and responding with bits
2. Any way of taking in 1's and 0's and responding with 1's and 0's

- Carrying out a logic function: *crunching bits*

Bit Crunchers (Logic Functions) So Far

1. AND
2. OR
3. INVERT

Logic Functions

Any Way of Crunching Bits

Bit Crunchers (Logic Functions) So Far

1. AND
2. OR
3. INVERT

Questions about Bit Crunching

1. Are there any bit crunching tasks these three can't do?

Logic Functions

Any Way of Crunching Bits

Bit Crunchers (Logic Functions) So Far

1. AND
2. OR
3. INVERT

Questions about Bit Crunching

1. Are there any bit crunching tasks these three can't do?
2. These just take in 2 bits and spit out 1, how do we do tasks involving more inputs and outputs?

Logic Functions

Any Way of Crunching Bits

Bit Crunchers (Logic Functions) So Far

1. AND
2. OR
3. INVERT

Questions about Bit Crunching

1. Are there any bit crunching tasks these three can't do?
2. These just take in 2 bits and spit out 1, how do we do tasks involving more inputs and outputs?
3. Why do we care about either of the questions above?

Answering the First Question

The Nature of Bit Crunching

Question 1

Are there any bit crunching tasks which (together) AND, OR and INVERT can't perform?

Answer

- No: together AND, OR and INVERT can perform *any* logic function

Answering the First Question

The Nature of Bit Crunching

Question 1

Are there any bit crunching tasks which (together) AND, OR and INVERT can't perform?

Answer

- No: together AND, OR and INVERT can perform *any* logic function
- In fact: just OR + INVERT or AND + INVERT can perform any logic function

Answering the First Question

The Nature of Bit Crunching

Question 1

Are there any bit crunching tasks which (together) AND, OR and INVERT can't perform?

Answer

- No: together AND, OR and INVERT can perform *any* logic function
- In fact: just OR + INVERT or AND + INVERT can perform any logic function
- Recall from last class that INVERT and OR can be combined to do what AND does

Answering the Second Question

More on the Nature of Bit Crunching

Question 2

AND, OR and INVERT just take in 2 bits and spit out 1, how do we do tasks involving more inputs and outputs?

Answering the Second Question

More on the Nature of Bit Crunching

Question 2

AND, OR and INVERT just take in 2 bits and spit out 1, how do we do tasks involving more inputs and outputs?

Answer

- By chaining together AND, OR and INVERT one can construct bit crunchers that take in more than 2 bits and spit out more than 1

Three Inputs

By Combining ORs

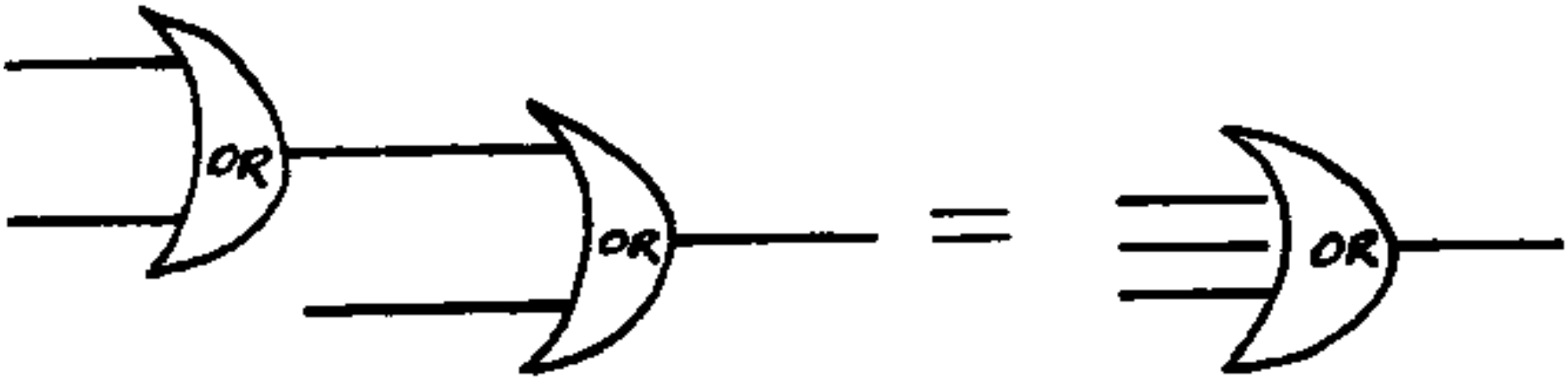


FIGURE 10

A three-input Or block made from a pair of two-input Or blocks

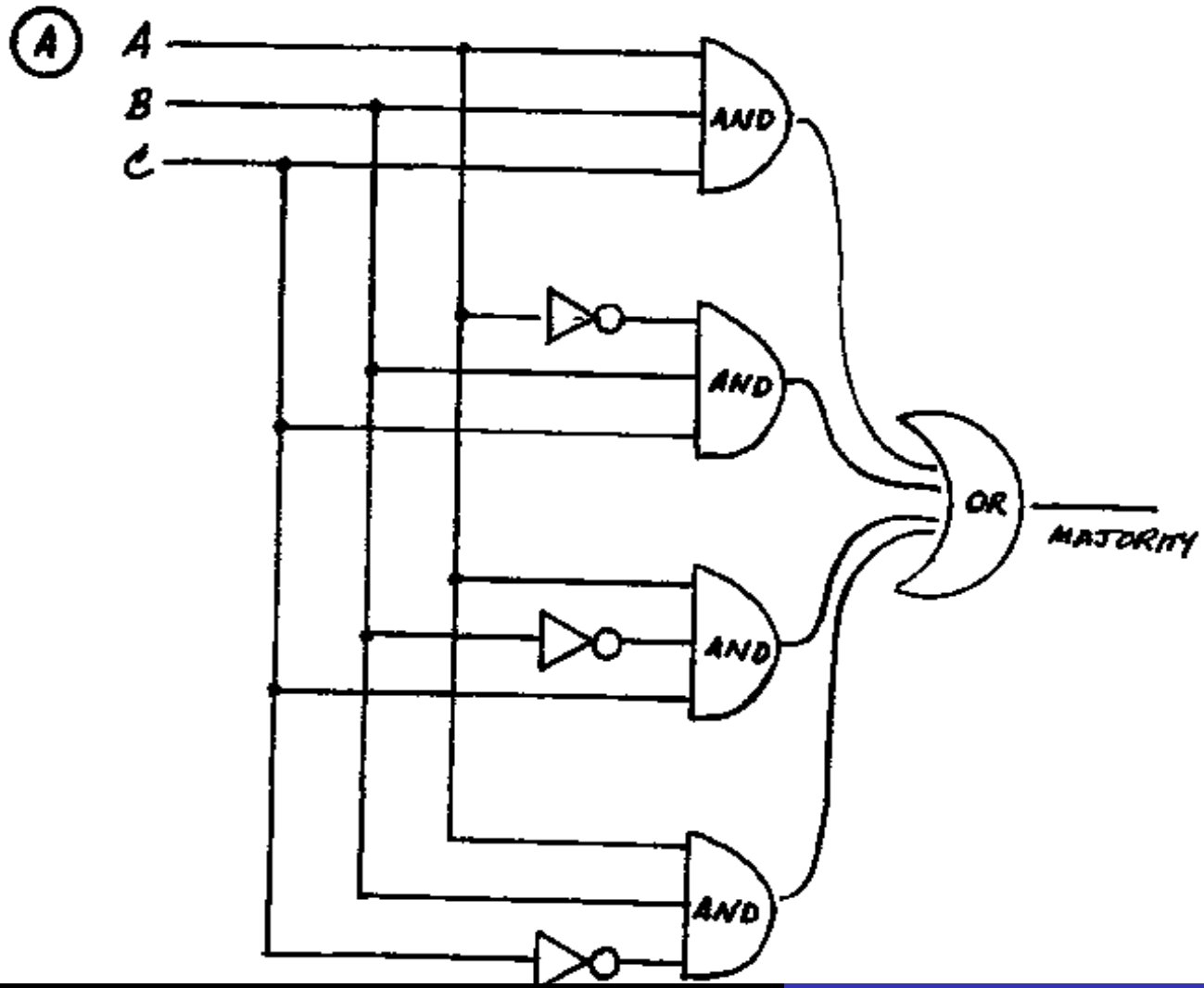
Majority Rules

Take-in 3, Put-out 1

Inputs			Majority Output
A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

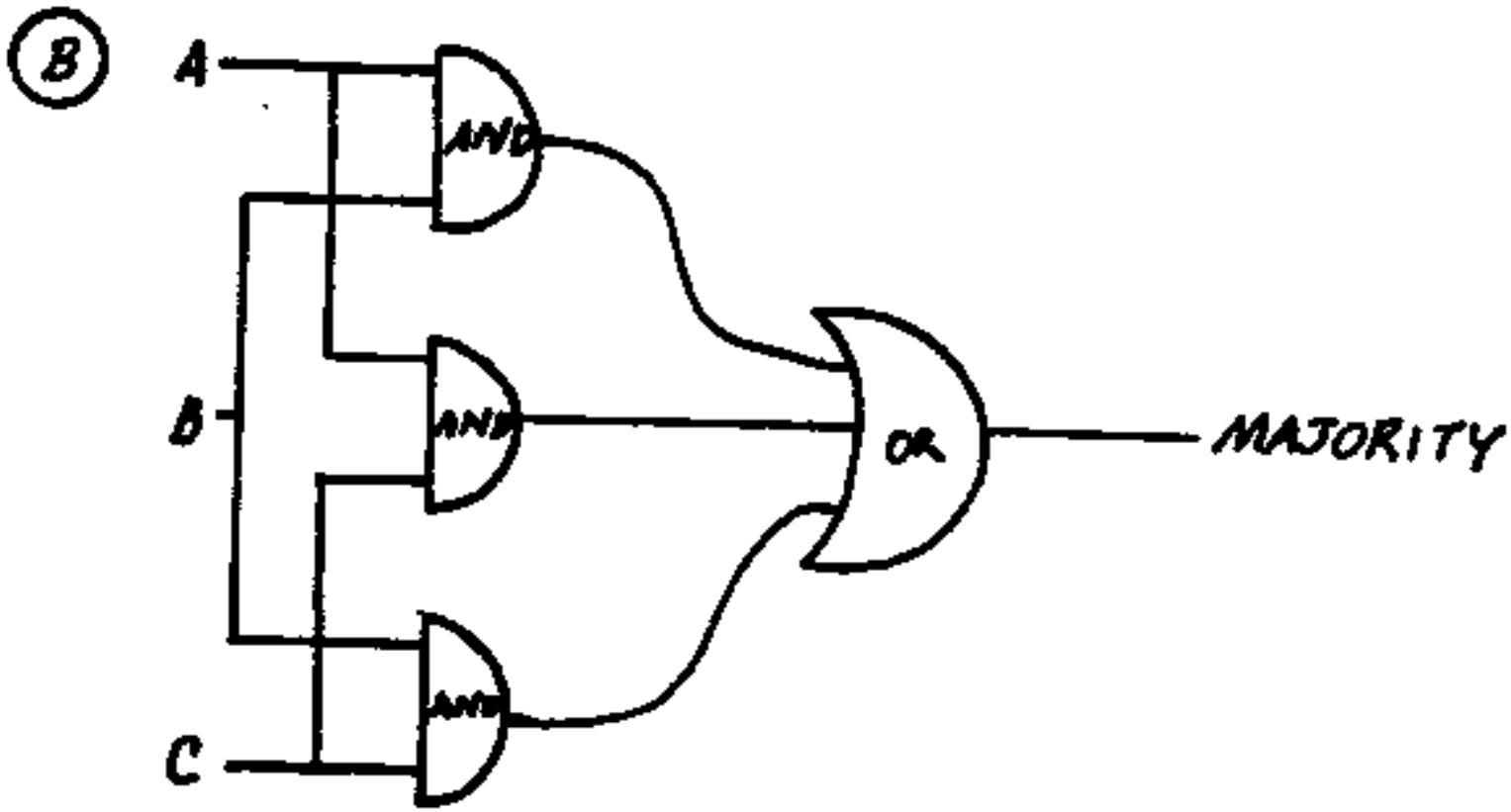
Majority Rules Diagram, v1

Take-in 3, Put-out 1



Majority Rules Diagram, v2

Take-in 3, Put-out 1



Answering Question 3

Why You Should Care about the Nature of Bit Crunchers

Question 3

Why care whether Boolean circuits can do any bit crunching task? Why care about more than 2 inputs or more than 1 output?

Answer

- Now we know that any bit crunching task can be reduced to two incredibly simple tasks

Answering Question 3

Why You Should Care about the Nature of Bit Crunchers

Question 3

Why care whether Boolean circuits can do any bit crunching task? Why care about more than 2 inputs or more than 1 output?

Answer

- Now we know that any bit crunching task can be reduced to two incredibly simple tasks
  - So by studying these bit crunchers, we can figure out what *any* bit crunchers could do

Answering Question 3

Why You Should Care about the Nature of Bit Crunchers

Question 3

Why care whether Boolean circuits can do any bit crunching task? Why care about more than 2 inputs or more than 1 output?

Answer

- Now we know that any bit crunching task can be reduced to two incredibly simple tasks
  - So by studying these bit crunchers, we can figure out what *any* bit crunchers could do
- Minds do things that require >2 inputs and outputs

Answering Question 3

Why You Should Care about the Nature of Bit Crunchers

Question 3

Why care whether Boolean circuits can do any bit crunching task? Why care about more than 2 inputs or more than 1 output?

Answer

- Now we know that any bit crunching task can be reduced to two incredibly simple tasks
  - So by studying these bit crunchers, we can figure out what *any* bit crunchers could do
- Minds do things that require >2 inputs and outputs
  - E.g. judging the winner of Rock-Paper-Scissors

Rock-Paper-Scissors Judge

Ternary Input, Ternary Output!

Input A	Input B	Output
Scissors	Scissors	Tie
Scissors	Paper	A wins
Scissors	Rock	B wins
Paper	Scissors	B wins
Paper	Paper	Tie
Paper	Rock	A wins
Rock	Scissors	A wins
Rock	Paper	B wins
Rock	Rock	Tie

Rock-Paper-Scissors

Using Two Binary Inputs to Mimic One Ternary Input

Scissors = 01
Paper = 10
Rock = 11
A wins = 10
B wins = 01
Tie = 00

A Inputs	B Inputs	Outputs
01	01	00
01	10	10
01	11	01
10	01	01
10	10	00
10	11	10
11	01	10
11	10	01
11	11	00

Bit Crunching

The Core Idea

What is a Bit Cruncher?

Takes x input bits and returns y output bits

Bit Crunching

The Core Idea

What is a Bit Cruncher?

Takes x input bits and returns y output bits

- For each set of x inputs, there is a set of y outputs that is always the result of that input

Bit Crunching

The Core Idea

What is a Bit Cruncher?

Takes x input bits and returns y output bits

- For each set of x inputs, there is a set of y outputs that is always the result of that input
- Constant behavior with respect to inputs

Bit Crunching

The Core Idea

What is a Bit Cruncher?

Takes x input bits and returns y output bits

- For each set of x inputs, there is a set of y outputs that is always the result of that input
- Constant behavior with respect to inputs

- Input 1 to a NOT circuit; you always get a 0 back!

Bit Crunching

The Core Idea

What is a Bit Cruncher?

Takes x input bits and returns y output bits

- For each set of x inputs, there is a set of y outputs that is always the result of that input
- Constant behavior with respect to inputs

- Input 1 to a NOT circuit; you always get a 0 back!
- Thinking of inputs as perceptions, does this seem like an adequate account of minds?

Bit Crunching

The Core Idea

What is a Bit Cruncher?

Takes x input bits and returns y output bits

- For each set of x inputs, there is a set of y outputs that is always the result of that input
- Constant behavior with respect to inputs

- Input 1 to a NOT circuit; you always get a 0 back!
- Thinking of inputs as perceptions, does this seem like an adequate account of minds?
- Are minds just giant Boolean circuits? ('Logic blocks')

Outline

Minds Aren't Just Logic Blocks

You Don't Always Do the Same Thing in Response to a Signal

- Logic blocks always react to the same signal/input in the same way, but things with minds don't!

Minds Aren't Just Logic Blocks

You Don't Always Do the Same Thing in Response to a Signal

- Logic blocks always react to the same signal/input in the same way, but things with minds don't!
- Minds react on the basis of their previous experiences!

Minds Aren't Just Logic Blocks

You Don't Always Do the Same Thing in Response to a Signal

- Logic blocks always react to the same signal/input in the same way, but things with minds don't!
- Minds react on the basis of their previous experiences!
- Even very simple machines do this:

Minds Aren't Just Logic Blocks

You Don't Always Do the Same Thing in Response to a Signal

- Logic blocks always react to the same signal/input in the same way, but things with minds don't!
- Minds react on the basis of their previous experiences!
- Even very simple machines do this:
  - Clicking pen, combo lock

Minds Aren't Just Logic Blocks

You Don't Always Do the Same Thing in Response to a Signal

- Logic blocks always react to the same signal/input in the same way, but things with minds don't!
- Minds react on the basis of their previous experiences!
- Even very simple machines do this:
  - Clicking pen, combo lock
- Machines which monitor and change an internal **register**: Finite State Machines

Minds Aren't Just Logic Blocks

You Don't Always Do the Same Thing in Response to a Signal

- Logic blocks always react to the same signal/input in the same way, but things with minds don't!
- Minds react on the basis of their previous experiences!
- Even very simple machines do this:
  - Clicking pen, combo lock
- Machines which monitor and change an internal **register**: Finite State Machines
  - 'Register': a limited, finite memory

Minds Aren't Just Logic Blocks

You Don't Always Do the Same Thing in Response to a Signal

- Logic blocks always react to the same signal/input in the same way, but things with minds don't!
- Minds react on the basis of their previous experiences!
- Even very simple machines do this:
  - Clicking pen, combo lock
- Machines which monitor and change an internal **register**: Finite State Machines
  - 'Register': a limited, finite memory
- Register keeps track of the previous input

Minds Aren't Just Logic Blocks

You Don't Always Do the Same Thing in Response to a Signal

- Logic blocks always react to the same signal/input in the same way, but things with minds don't!
- Minds react on the basis of their previous experiences!
- Even very simple machines do this:
  - Clicking pen, combo lock
- Machines which monitor and change an internal **register**: Finite State Machines
  - 'Register': a limited, finite memory
- Register keeps track of the previous input
- Register is itself is a Boolean circuit

Finite State Machines

Combo Lock For 0-5-2 (p.36)

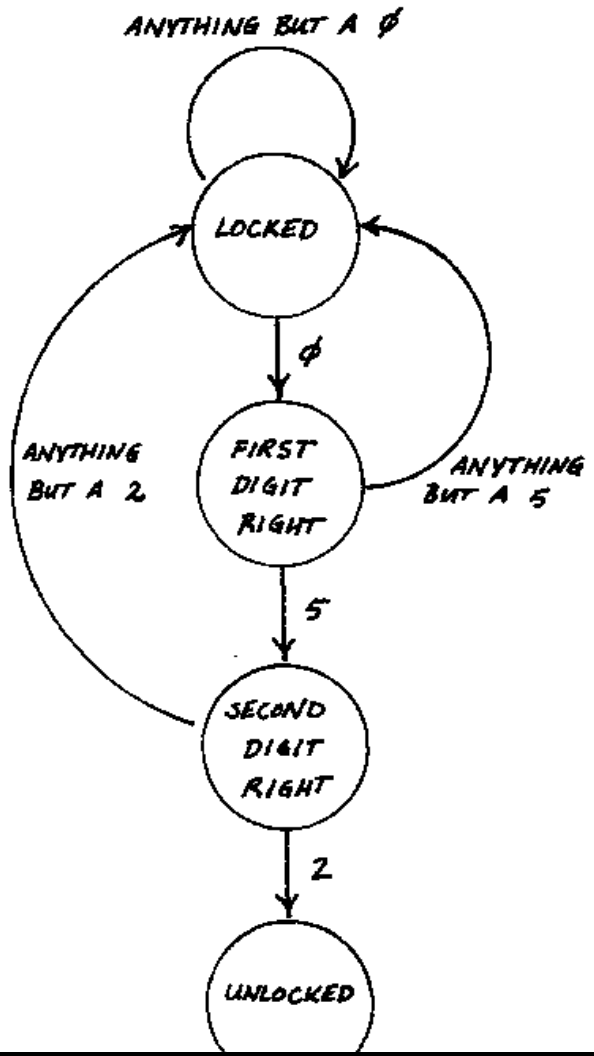


FIGURE 15

State diagram for a lock

Finite State Machines

Combo Lock For 0-5-2 (p.36)

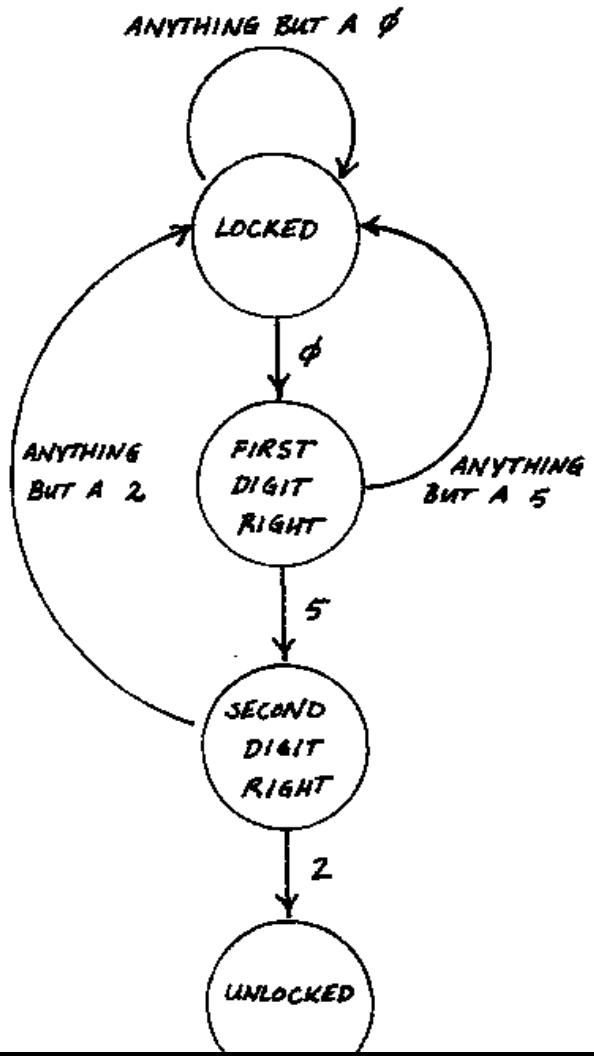


FIGURE 15

State diagram for a lock

Finite State Machines

Combo Lock For 0-5-2 (p.36)

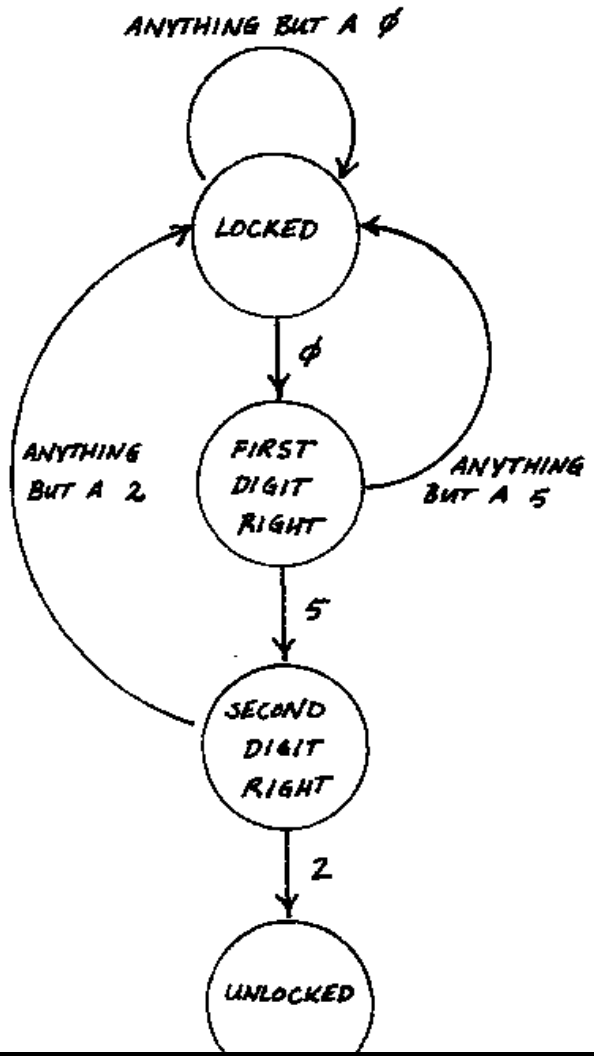


FIGURE 15

State diagram for a lock

Finite State Machines

Combo Lock For 0-5-2 (p.36)

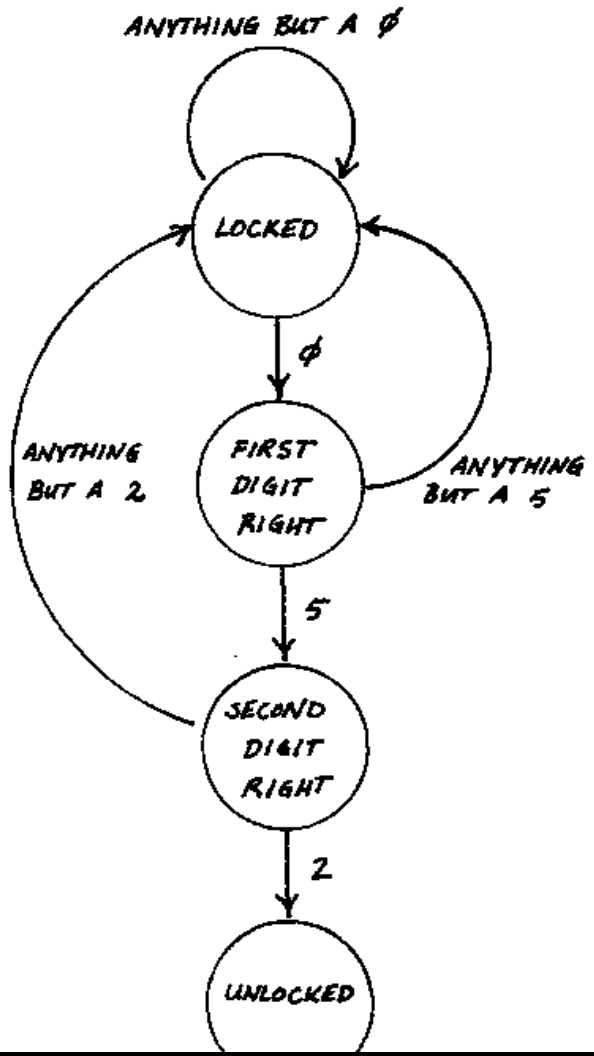


FIGURE 15

State diagram for a lock

FSMs Listen to Themselves

- Response = input + current state
- Current state = previous input

Stoplight Controller

As a Finite State Machine

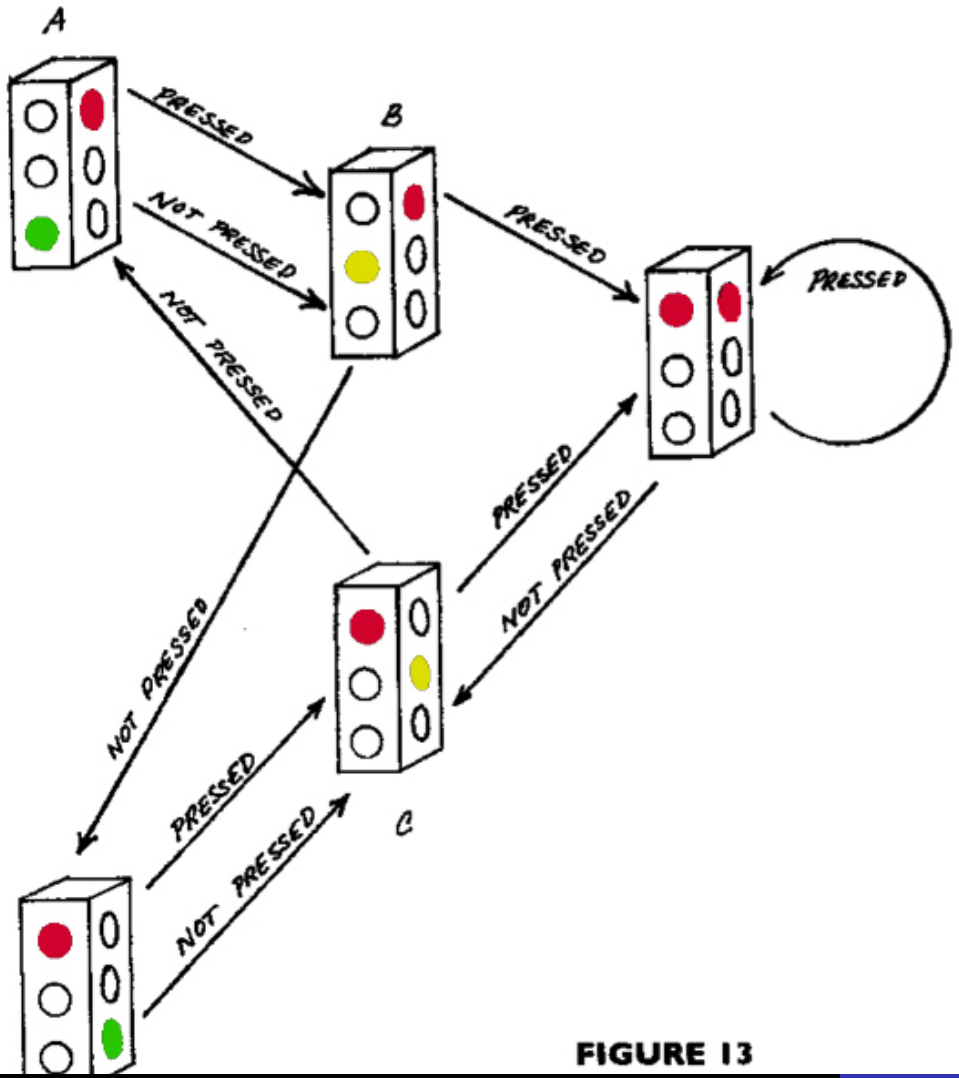


FIGURE 13

How A Stoplight Works

In Terms of Input/Output

Inputs:		*Outputs:*		
Walk Button	**Current State**	**Main Road**	**Cross Road**	**Next State**
Not Pressed	A	Red	Green	B
Not Pressed	B	Red	Yellow	D
Not Pressed	C	Yellow	Red	A
Not Pressed	D	Green	Red	C
Not Pressed	Walk	Walk	Walk	D
Pressed	A	Red	Green	B
Pressed	B	Red	Yellow	Walk
Pressed	C	Yellow	Red	Walk
Pressed	D	Green	Red	C
Pressed	Walk	Walk	Walk	Walk

Finite State Machines

The Mealy Machine

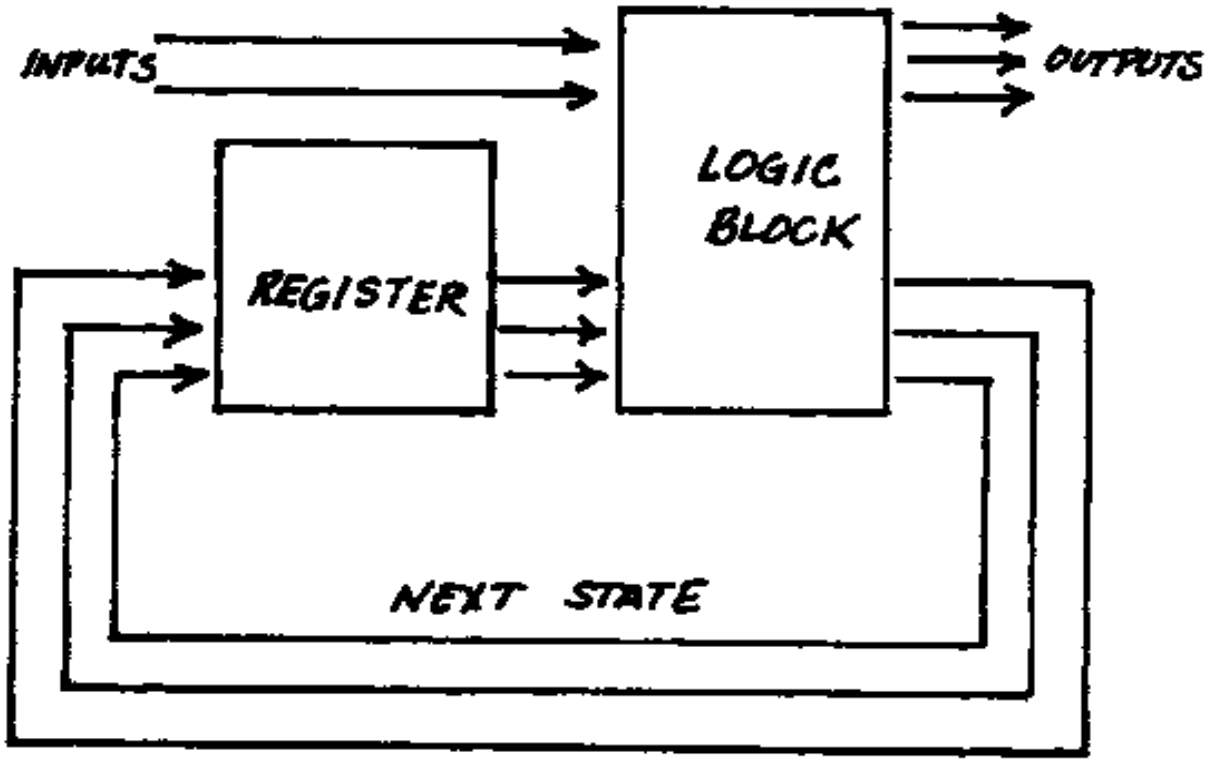


FIGURE 14

Finite-state machine, with logic block feeding register

(G.H. Mealy, "A Method for Synthesizing Sequential Circuits" 1955)

Finite State Machines

In General

In General

1. Register built from Boolean circuits

Finite State Machines

In General

In General

1. Register built from Boolean circuits
2. Register and inputs feed into second logic block

Finite State Machines

In General

In General

1. Register built from Boolean circuits
2. Register and inputs feed into second logic block
3. This logic block produces an output and sends bits back to register

Finite State Machines

In General

In General

1. Register built from Boolean circuits
2. Register and inputs feed into second logic block
3. This logic block produces an output and sends bits back to register
4. Register crunches these bits

Finite State Machines

In General

In General

1. Register built from Boolean circuits
2. Register and inputs feed into second logic block
3. This logic block produces an output and sends bits back to register
4. Register crunches these bits
5. And then the cycle repeats

Finite State Machines

In General

In General

1. Register built from Boolean circuits
2. Register and inputs feed into second logic block
3. This logic block produces an output and sends bits back to register
4. Register crunches these bits
5. And then the cycle repeats

- This allows finite state machines to recognize patterns

Finite State Machines

In General

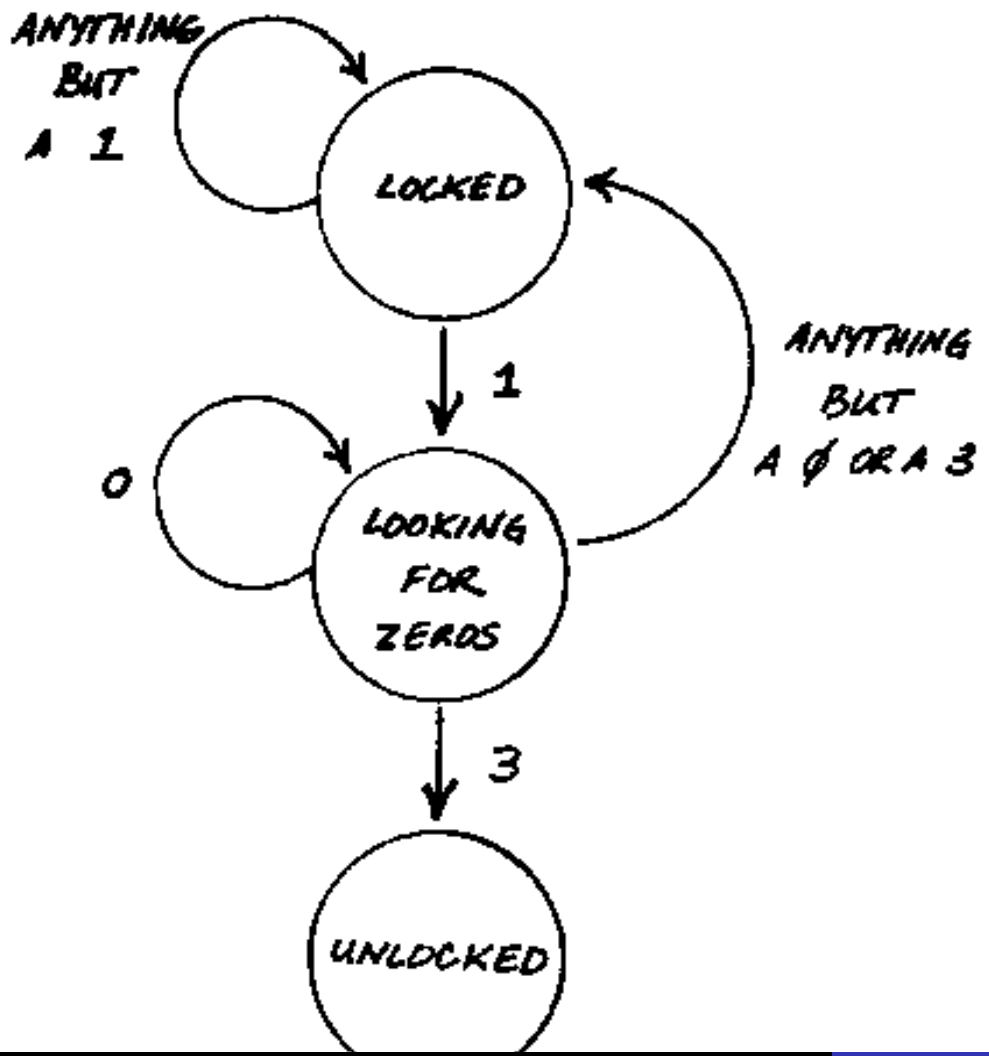
In General

1. Register built from Boolean circuits
2. Register and inputs feed into second logic block
3. This logic block produces an output and sends bits back to register
4. Register crunches these bits
5. And then the cycle repeats

- This allows finite state machines to recognize patterns
- All patterns?

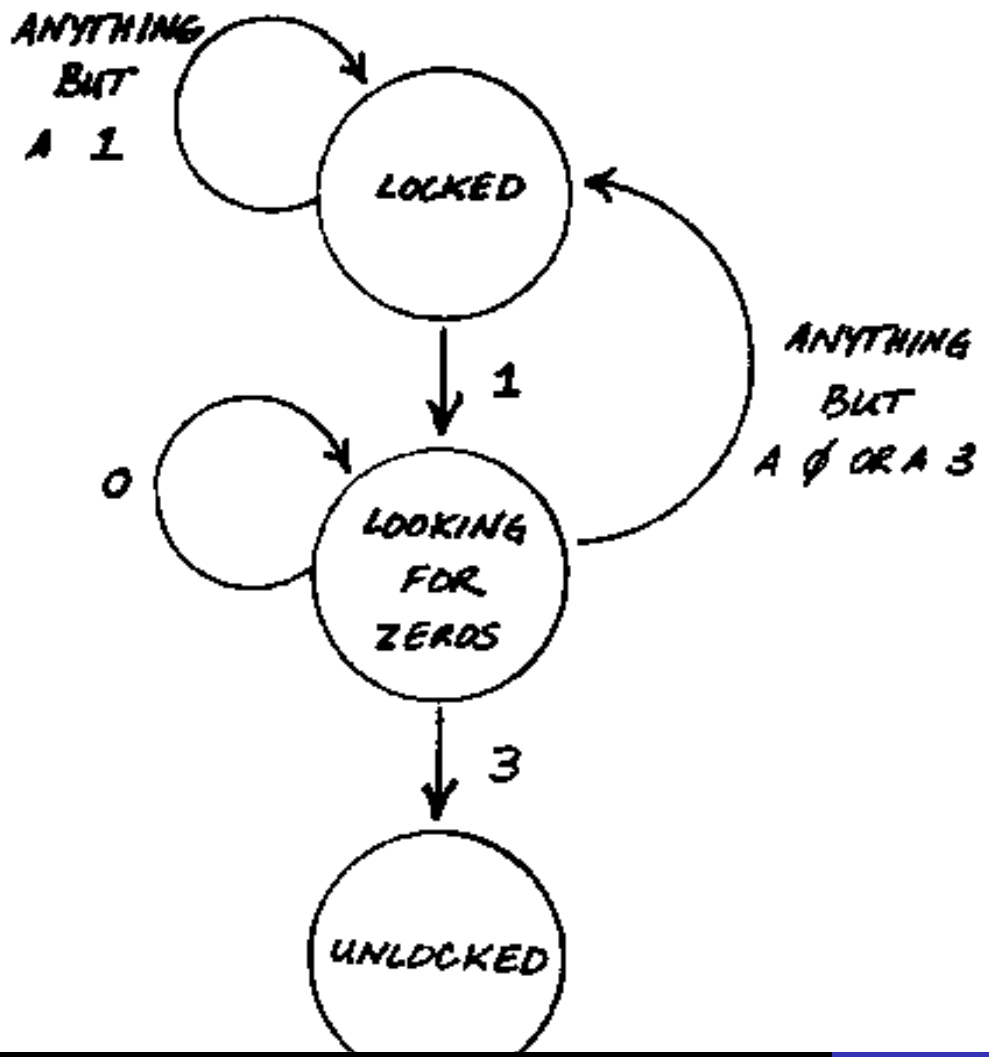
Finite State Machines

Recognizing Patterns: begins w/1, any number of 0s, ends w/3 (p.35)



Finite State Machines

Recognizing Patterns: begins w/1, any number of 0s, ends w/3 (p.35)



Important

There is one state for each digit that has to be remembered.

Finite State Machines

Recognizing Patterns

Palindrome

A string of symbols which is the same whether read backwards or forwards.

- E.g. 99, 757, 1001, abba, abccba, dad, mom

Finite State Machines

Recognizing Patterns

Palindrome

A string of symbols which is the same whether read backwards or forwards.

- E.g. 99, 757, 1001, abba, abccba, dad, mom
- A finite state combo lock that detects only palindromes is not possible (even if alphabet is finite)

Finite State Machines

Recognizing Patterns

Palindrome

A string of symbols which is the same whether read backwards or forwards.

- E.g. 99, 757, 1001, abba, abccba, dad, mom
- A finite state combo lock that detects only palindromes is not possible (even if alphabet is finite)

Why

To recognize a palindrome, you need to remember every digit of the first half. You need one state for each such memory. But there are infinitely many first halves, so you would need infinitely many states.

Finite State Machines

And Human Language

- Chomsky (*Syntactic Structures* 1957) argued an FSM could not recognize all sentences of a human language

Finite State Machines

And Human Language

- Chomsky (*Syntactic Structures* 1957) argued an FSM could not recognize all sentences of a human language
- Consider a sentence of this form:

Finite State Machines

And Human Language

- Chomsky (*Syntactic Structures* 1957) argued an FSM could not recognize all sentences of a human language
- Consider a sentence of this form:
  - *Either* S_1 *or* S_2
    (S_1 and S_2 are sentences)

Finite State Machines

And Human Language

- Chomsky (*Syntactic Structures* 1957) argued an FSM could not recognize all sentences of a human language
- Consider a sentence of this form:
  - *Either* S_1 *or* S_2
    (S_1 and S_2 are sentences)
  - E.g. *Either Jill is happy or Sarah is happy*

Finite State Machines

And Human Language

- Chomsky (*Syntactic Structures* 1957) argued an FSM could not recognize all sentences of a human language
- Consider a sentence of this form:
  - *Either* S_1 *or* S_2
    (S_1 and S_2 are sentences)
  - E.g. *Either Jill is happy or Sarah is happy*
- To recognize all sentences of this form, a FSM would have to have a state for each different S_1

Finite State Machines

And Human Language

- Chomsky (*Syntactic Structures* 1957) argued an FSM could not recognize all sentences of a human language
- Consider a sentence of this form:
  - *Either* S_1 *or* S_2
    (S_1 and S_2 are sentences)
  - E.g. *Either Jill is happy or Sarah is happy*
- To recognize all sentences of this form, a FSM would have to have a state for each different S_1
- But there are infinitely many!

Finite State Machines

And Human Language

- Chomsky (*Syntactic Structures* 1957) argued an FSM could not recognize all sentences of a human language
- Consider a sentence of this form:
  - *Either* S_1 *or* S_2
    (S_1 and S_2 are sentences)
  - E.g. *Either Jill is happy or Sarah is happy*
- To recognize all sentences of this form, a FSM would have to have a state for each different S_1
- But there are infinitely many!
- Think you've found the last sentence? Call it S.

Finite State Machines

And Human Language

- Chomsky (*Syntactic Structures* 1957) argued an FSM could not recognize all sentences of a human language
- Consider a sentence of this form:
  - *Either* S_1 *or* S_2
    (S_1 and S_2 are sentences)
  - E.g. *Either Jill is happy or Sarah is happy*
- To recognize all sentences of this form, a FSM would have to have a state for each different S_1
- But there are infinitely many!
- Think you've found the last sentence? Call it S.
- Isn't this a sentence: *I don't believe that S*?

Reading

For 10.22

Required

Ch. 3 of *The Pattern on the Stone*

Recommended