What is a Computer?

Programming, Memory and Limits of Computation

Carlotta Pavese

10.22.13

Outline

Outline

Review

From Minds to Machines

- Dualism 👎, Identity Theory 👎, Functionalism 👍

Review

From Minds to Machines

- Dualism ☟, Identity Theory ☟, Functionalism ☝
- Functionalism analyzes minds in terms of what they do

Review

From Minds to Machines

- Dualism 👎, Identity Theory 👎, Functionalism 👍
- Functionalism analyzes minds in terms of what they do
- Functionalism abstracts from how they physically do it.

Review

From Minds to Machines

- Dualism 👎, Identity Theory 👎, Functionalism 👍
- Functionalism analyzes minds in terms of what they do
- Functionalism abstracts from how they physically do it.
- Computationalism provides one precise account of what mental processes might be: computation

Review

From Minds to Machines

- Dualism 👎, Identity Theory 👎, Functionalism 👍
- Functionalism analyzes minds in terms of what they do
- Functionalism abstracts from how they physically do it.
- Computationalism provides one precise account of what mental processes might be: computation
- So what exactly is computation?

Review

From Minds to Machines

- Dualism , Identity Theory , Functionalism
- Functionalism analyzes minds in terms of what they do
- Functionalism abstracts from how they physically do it.
- Computationalism provides one precise account of what mental processes might be: computation
- So what exactly is computation?
- Could the things minds do amount to computation?

Review

From Boolean Circuits to Finite State Machines

- Boolean Circuits (Bit Crunchers) are an important building block of computers

Review

From Boolean Circuits to Finite State Machines

- Boolean Circuits (Bit Crunchers) are an important building block of computers
- Boolean Logic captures at least (part of) one component of thought: inference

Review

From Boolean Circuits to Finite State Machines

- Boolean Circuits (Bit Crunchers) are an important building block of computers
- Boolean Logic captures at least (part of) one component of thought: inference
- Boolean Circuits can do Boolean Logic

Review

From Boolean Circuits to Finite State Machines

- Boolean Circuits (Bit Crunchers) are an important building block of computers
- Boolean Logic captures at least (part of) one component of thought: inference
- Boolean Circuits can do Boolean Logic
- But Boolean Circuits always responds the same way to any input and minds don't

Review

From Boolean Circuits to Finite State Machines

- Boolean Circuits (Bit Crunchers) are an important building block of computers
- Boolean Logic captures at least (part of) one component of thought: inference
- Boolean Circuits can do Boolean Logic
- But Boolean Circuits always responds the same way to any input and minds don't
- But neither do Finite State Machines

Review

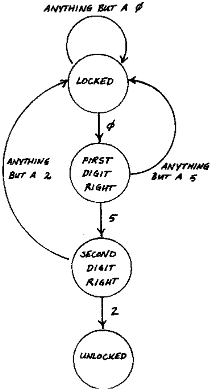
From Boolean Circuits to Finite State Machines

- Boolean Circuits (Bit Crunchers) are an important building block of computers
- Boolean Logic captures at least (part of) one component of thought: inference
- Boolean Circuits can do Boolean Logic
- But Boolean Circuits always responds the same way to any input and minds don't
- But neither do Finite State Machines
- And FSMs are another important building block of computers

Outline

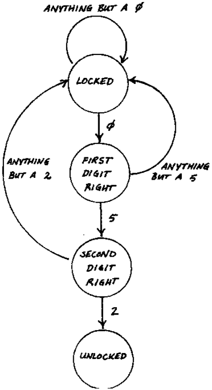
Finite State Machines

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



Finite State Machines

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

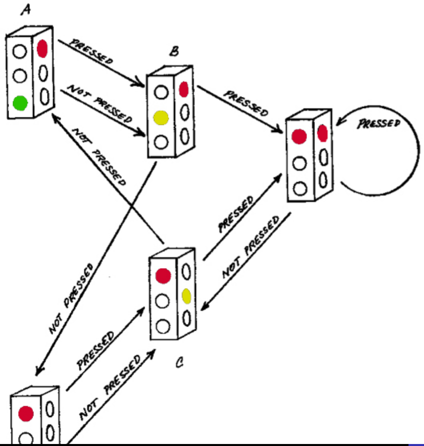


FSMs Listen to Themselves

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

Stoplight Controller

As a Finite State Machine



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

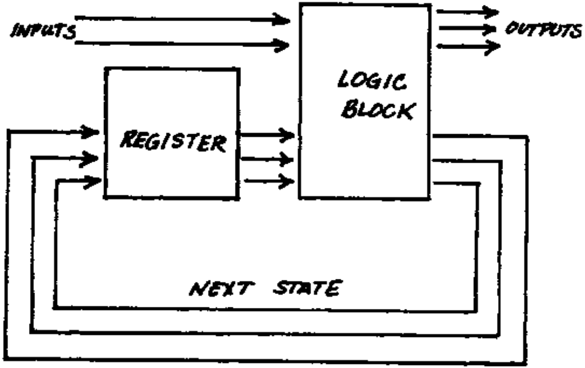


FIGURE 14

Finite State 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

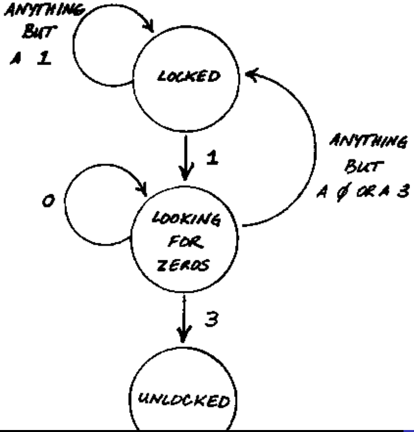
In General

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?

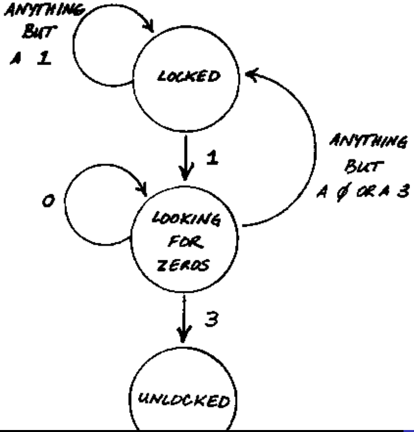
Finite State Machines

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.

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!
- Machines which are able to 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!
- Machines which are able to 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!
- Machines which are able to monitor and change an internal **register**: Finite State Machines
- 'Register': a limited, finite memory
- The register keeps track of the previous input by switching into a state unique to that 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!
- Machines which are able to monitor and change an internal **register**: Finite State Machines
- 'Register': a limited, finite memory
- The register keeps track of the previous input by switching into a state unique to that input
- The register itself is a logic block: a configuration of bits that changes upon input

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

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?

The Bridge

From Finite State Machines to Computers

- FSMs produce output based on input **and** current internal state (their 'register')

The Bridge

From Finite State Machines to Computers

- FSMs produce output based on input **and** current internal state (their ‘register’)
  - And then they update that internal state

The Bridge

From Finite State Machines to Computers

- FSMs produce output based on input **and** current internal state (their 'register')
  - And then they update that internal state
- They are one step closer to a mind, but fall short

The Bridge

From Finite State Machines to Computers

- FSMs produce output based on input **and** current internal state (their 'register')
  - And then they update that internal state
- They are one step closer to a mind, but fall short
- FSMs cannot recognize various patterns that people can (palindromes, arguably certain sentences)

The Bridge

From Finite State Machines to Computers

- FSMs produce output based on input **and** current internal state (their 'register')
  - And then they update that internal state
- They are one step closer to a mind, but fall short
- FSMs cannot recognize various patterns that people can (palindromes, arguably certain sentences)
- This is because FSMs don't have a sophisticated enough memory

The Bridge

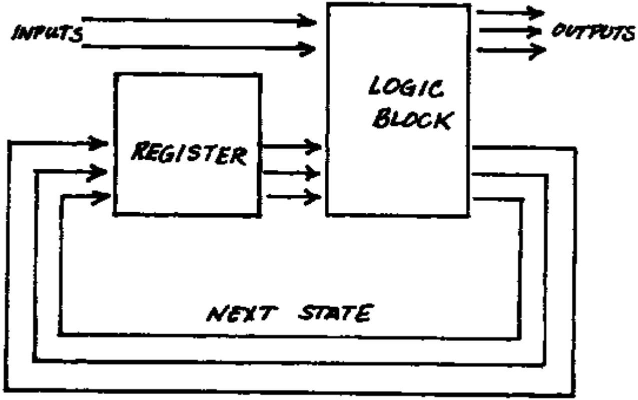
From Finite State Machines to Computers

- FSMs produce output based on input **and** current internal state (their ‘register’)
  - And then they update that internal state
- They are one step closer to a mind, but fall short
- FSMs cannot recognize various patterns that people can (palindromes, arguably certain sentences)
- This is because FSMs don’t have a sophisticated enough memory
- Final ingredient of a computer: a more sophisticated kind of memory

Outline

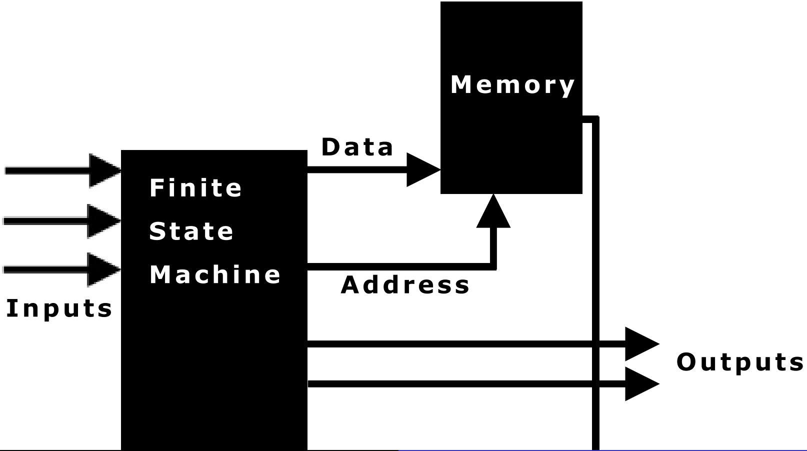
Finite State Machines

In General



Finite State Machine

With a Real Memory



What is A Real Memory?

Memory, Registers, Words, Addresses, Reading, Writing

- A memory is a massive array of registers

What is A Real Memory?

Memory, Registers, Words, Addresses, Reading, Writing

- A memory is a massive array of registers
- Each register contains a bit pattern called a word (64 these days)

What is A Real Memory?

Memory, Registers, Words, Addresses, Reading, Writing

- A memory is a massive array of registers
- Each register contains a bit pattern called a word (64 these days)
- Each register has an address

What is A Real Memory?

Memory, Registers, Words, Addresses, Reading, Writing

- A memory is a massive array of registers
- Each register contains a bit pattern called a word (64 these days)
- Each register has an address
- The memory also contains Boolean logic blocks that take an address from the FSM, find it's register

What is A Real Memory?

Memory, Registers, Words, Addresses, Reading, Writing

- A memory is a massive array of registers
- Each register contains a bit pattern called a word (64 these days)
- Each register has an address
- The memory also contains Boolean logic blocks that take an address from the FSM, find it's register
- Then they either change the bit pattern in that register (write)

What is A Real Memory?

Memory, Registers, Words, Addresses, Reading, Writing

- A memory is a massive array of registers
- Each register contains a bit pattern called a word (64 these days)
- Each register has an address
- The memory also contains Boolean logic blocks that take an address from the FSM, find it's register
- Then they either change the bit pattern in that register (write)
- Or they read it and send it back to the FSM

What is A Real Memory?

Memory, Registers, Words, Addresses, Reading, Writing

- A memory is a massive array of registers
- Each register contains a bit pattern called a word (64 these days)
- Each register has an address
- The memory also contains Boolean logic blocks that take an address from the FSM, find it's register
- Then they either change the bit pattern in that register (write)
- Or they read it and send it back to the FSM
- This bit pattern is either an instruction or some data

What is A Real Memory?

Memory, Registers, Words, Addresses, Reading, Writing

- A memory is a massive array of registers
- Each register contains a bit pattern called a word (64 these days)
- Each register has an address
- The memory also contains Boolean logic blocks that take an address from the FSM, find it's register
- Then they either change the bit pattern in that register (write)
- Or they read it and send it back to the FSM
- This bit pattern is either an instruction or some data
- Data are treated as input, instructions are executed

A Real Memory

Remembering Instructions

- When an FSM is hooked up to a memory, it can not only retrieve data, but whole instructions

A Real Memory

Remembering Instructions

- When an FSM is hooked up to a memory, it can not only retrieve data, but whole instructions
- For instance, the FSM may react to a particular bit pattern from memory by adding

A Real Memory

Remembering Instructions

- When an FSM is hooked up to a memory, it can not only retrieve data, but whole instructions
- For instance, the FSM may react to a particular bit pattern from memory by adding
- This involves first getting bit patterns from inputs or registers

A Real Memory

Remembering Instructions

- When an FSM is hooked up to a memory, it can not only retrieve data, but whole instructions
- For instance, the FSM may react to a particular bit pattern from memory by adding
- This involves first getting bit patterns from inputs or registers
- A logic block then adds these bit patterns, and writes the result to the memory

Outline

A General Purpose Computer

How Do You Get a Computer To Do Your Bidding?

Zuse Z3 (1941): 1st Programmable Computer

- How does a computer interpret bit patterns that are to be treated as instructions?
- How do you program one to do new things?

Machine Language

The Computer's Mother Tongue

- The FSM is hardwired to respond in a particular way to certain bit patterns from memory

Machine Language

The Computer's Mother Tongue

- The FSM is hardwired to respond in a particular way to certain bit patterns from memory
  - The primitive words of its 'machine language'

Machine Language

The Computer's Mother Tongue

- The FSM is hardwired to respond in a particular way to certain bit patterns from memory
  - The primitive words of its 'machine language'
- Responses to processing instructions are: move data to and from memory, perform arithmetic, etc.

Machine Language

The Computer's Mother Tongue

- The FSM is hardwired to respond in a particular way to certain bit patterns from memory
  - The primitive words of its 'machine language'
- Responses to processing instructions are: move data to and from memory, perform arithmetic, etc.
- Responses to control instructions determine address of next instruction to be retrieved

Machine Language

The Computer's Mother Tongue

- The FSM is hardwired to respond in a particular way to certain bit patterns from memory
  - The primitive words of its 'machine language'
- Responses to processing instructions are: move data to and from memory, perform arithmetic, etc.
- Responses to control instructions determine address of next instruction to be retrieved
  - Stored in special register: program counter

Machine Language

The Computer's Mother Tongue

- The FSM is hardwired to respond in a particular way to certain bit patterns from memory
  - The primitive words of its 'machine language'
- Responses to processing instructions are: move data to and from memory, perform arithmetic, etc.
- Responses to control instructions determine address of next instruction to be retrieved
  - Stored in special register: program counter
- Normally computer will just move from one register to next, but control instructions can tell it to skip some or repeat some until a condition is met (loop)

Programming Languages

Are Everywhere

- So how do you teach your computer to do new things like record music or play video's?

Programming Languages

Are Everywhere

- So how do you teach your computer to do new things like record music or play video's?
- You don't really

Programming Languages

Are Everywhere

- So how do you teach your computer to do new things like record music or play video's?
- You don't really
- There are various programming languages used to make computers do these things

Programming Languages

Are Everywhere

- So how do you teach your computer to do new things like record music or play video's?
- You don't really
- There are various programming languages used to make computers do these things
- Your web browsers turn instructions in HTML, CSS and Javascript into what you see

Programming Languages

Are Everywhere

- So how do you teach your computer to do new things like record music or play video's?
- You don't really
- There are various programming languages used to make computers do these things
- Your web browsers turn instructions in HTML, CSS and Javascript into what you see
- My computer turned LaTeX into these slides

Programming Languages

Are Everywhere

- So how do you teach your computer to do new things like record music or play video's?
- You don't really
- There are various programming languages used to make computers do these things
- Your web browsers turn instructions in HTML, CSS and Javascript into what you see
- My computer turned LaTeX into these slides
- The computer does this by translating the programming language into its machine language

For Example

Logo and the Tortoise

- Sequences of characters are stored in adjacent registers

For Example

Logo and the Tortoise

- Sequences of characters are stored in adjacent registers
- The memory also stores a directory of the address patterns that correspond to commands of Logo

For Example

Logo and the Tortoise

- Sequences of characters are stored in adjacent registers
- The memory also stores a directory of the address patterns that correspond to commands of Logo
  - And lists next to them the addresses of the machine code commands the Logo commands correspond to

For Example

Logo and the Tortoise

- Sequences of characters are stored in adjacent registers
- The memory also stores a directory of the address patterns that correspond to commands of Logo
  - And lists next to them the addresses of the machine code commands the Logo commands correspond to
- When a sequence corresponding to a Logo command is entered, the FSM looks up and performs the corresponding commands of machine code

For Example

Logo and the Tortoise

- Sequences of characters are stored in adjacent registers
- The memory also stores a directory of the address patterns that correspond to commands of Logo
  - And lists next to them the addresses of the machine code commands the Logo commands correspond to
- When a sequence corresponding to a Logo command is entered, the FSM looks up and performs the corresponding commands of machine code
- So Logo is just a convenient Macro!

The Hierarchy

In Brief

1. Computers do stuff by executing programs

The Hierarchy

In Brief

1. Computers do stuff by executing programs
2. These are converted to machine language by compilers/interpreters by operating system

The Hierarchy

In Brief

1. Computers do stuff by executing programs
2. These are converted to machine language by compilers/interpreters by operating system
3. These programs, together w/data, are stored in memory

The Hierarchy

In Brief

1. Computers do stuff by executing programs
2. These are converted to machine language by compilers/interpreters by operating system
3. These programs, together w/data, are stored in memory
4. Both are fetched and the latter executed by a FSM

The Hierarchy

In Brief

1. Computers do stuff by executing programs
2. These are converted to machine language by compilers/interpreters by operating system
3. These programs, together w/data, are stored in memory
4. Both are fetched and the latter executed by a FSM
5. Both are stored as bit patterns

The Hierarchy

In Brief

1. Computers do stuff by executing programs
2. These are converted to machine language by compilers/interpreters by operating system
3. These programs, together w/data, are stored in memory
4. Both are fetched and the latter executed by a FSM
5. Both are stored as bit patterns
6. The FSM and memory are built from logic blocks

The Hierarchy

In Brief

1. Computers do stuff by executing programs
2. These are converted to machine language by compilers/interpreters by operating system
3. These programs, together w/data, are stored in memory
4. Both are fetched and the latter executed by a FSM
5. Both are stored as bit patterns
6. The FSM and memory are built from logic blocks
7. Logic blocks crunch bits

The Hierarchy

In Brief

1. Computers do stuff by executing programs
2. These are converted to machine language by compilers/interpreters by operating system
3. These programs, together w/data, are stored in memory
4. Both are fetched and the latter executed by a FSM
5. Both are stored as bit patterns
6. The FSM and memory are built from logic blocks
7. Logic blocks crunch bits
8. Crunching bits is manipulating a physical substance to send one of two possible signals 1 or 0

Outline

Introduction

Finite State Machines

Computers

Programming

Computation and Turing Machines

Extents and Limits of Computation

Today

Overview

1. Learn Turing's way of thinking about the memory

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine
2. Use this to connect computation to a familiar idea

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine
2. Use this to connect computation to a familiar idea
   - Calculation: following step-by-step rules

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine
2. Use this to connect computation to a familiar idea
   - Calculation: following step-by-step rules
3. The connection is the Church-Turing Thesis

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine
2. Use this to connect computation to a familiar idea
   - Calculation: following step-by-step rules
3. The connection is the Church-Turing Thesis
   - Whatever can be calculated, can be computed

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine
2. Use this to connect computation to a familiar idea
   - Calculation: following step-by-step rules
3. The connection is the Church-Turing Thesis
   - Whatever can be calculated, can be computed
4. Learn about Universality

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine
2. Use this to connect computation to a familiar idea
   - Calculation: following step-by-step rules
3. The connection is the Church-Turing Thesis
   - Whatever can be calculated, can be computed
4. Learn about Universality
   - The Universal Turing Machine can do what any other Turing Machine can

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine
2. Use this to connect computation to a familiar idea
   - Calculation: following step-by-step rules
3. The connection is the Church-Turing Thesis
   - Whatever can be calculated, can be computed
4. Learn about Universality
   - The Universal Turing Machine can do what any other Turing Machine can
5. Learn about the limits of computability:

Today

Overview

1. Learn Turing's way of thinking about the memory
   - His model of a computer: Turing Machine
2. Use this to connect computation to a familiar idea
   - Calculation: following step-by-step rules
3. The connection is the Church-Turing Thesis
   - Whatever can be calculated, can be computed
4. Learn about Universality
   - The Universal Turing Machine can do what any other Turing Machine can
5. Learn about the limits of computability:
   - Halting Problem: no computer will ever solve it

Calculation

A Person, A Pencil and Some Paper

- You start by writing some things down

$$X - Z = Y$$

Calculation

A Person, A Pencil and Some Paper

- You start by writing some things down
- You follow some rules for using them to get some thing else

$$X - Z = Y$$

Calculation

A Person, A Pencil and Some Paper

- You start by writing some things down
- You follow some rules for using them to get some thing else
- You do this by writing out each application of the rule

$$X - Z = Y$$

Calculation

A Person, A Pencil and Some Paper

- You start by writing some things down
- You follow some rules for using them to get some thing else
- You do this by writing out each application of the rule
- These rules have the form: if you've written X, you can write Y

$$X - Z = Y$$

Calculation and Computation

Turing's Staring Point

Effective Calculation

An effective calculation is a calculation that:

1. Is set out in a finite number of exact instructions

Calculation and Computation

Turing's Staring Point

Effective Calculation

An effective calculation is a calculation that:

1. Is set out in a finite number of exact instructions
2. If carried out w/o error, produces the desired result in a finite number of steps

Calculation and Computation

Turing's Staring Point

Effective Calculation

An effective calculation is a calculation that:

1. Is set out in a finite number of exact instructions
2. If carried out w/o error, produces the desired result in a finite number of steps
3. Can be carried out (in principle) by a human with pencil and paper

Calculation and Computation

Turing's Staring Point

Effective Calculation

An effective calculation is a calculation that:

1. Is set out in a finite number of exact instructions
2. If carried out w/o error, produces the desired result in a finite number of steps
3. Can be carried out (in principle) by a human with pencil and paper
4. Demands no insight or ingenuity in following the instructions

Calculation and Computation

Turing's Staring Point

Effective Calculation

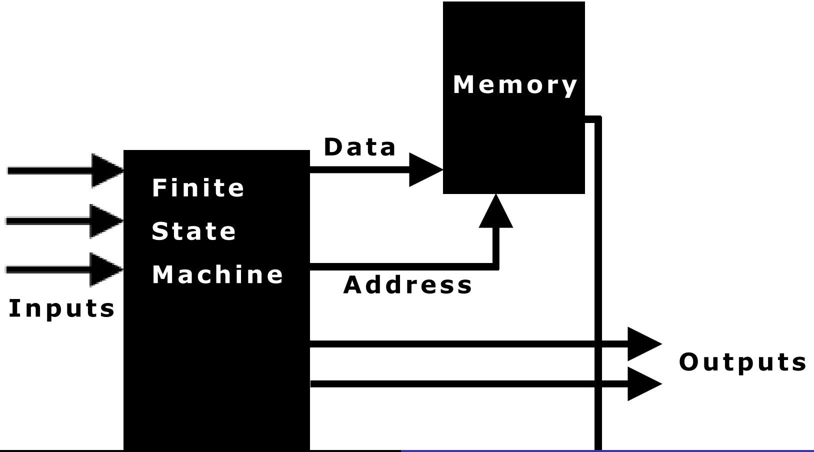
An effective calculation is a calculation that:

1. Is set out in a finite number of exact instructions
2. If carried out w/o error, produces the desired result in a finite number of steps
3. Can be carried out (in principle) by a human with pencil and paper
4. Demands no insight or ingenuity in following the instructions

Turing: to compute is just to effectively calculate

A Computer

FSM with a Memory



Computation and Calculation

Turing Machines

- Is a computer really doing (effective) calculations?

Computation and Calculation

Turing Machines

- Is a computer really doing (effective) calculations?
- To help see why this is plausible, we'll look at a slightly different model of a computer

Computation and Calculation

Turing Machines

- Is a computer really doing (effective) calculations?
- To help see why this is plausible, we'll look at a slightly different model of a computer
- The Turing Machine:
  - The memory is a long tape divided into cells
  - Each cell has a symbol in it
  - There's a 'head' always 'looking' at one cell

Computation and Calculation

Turing Machines

- Is a computer really doing (effective) calculations?
- To help see why this is plausible, we'll look at a slightly different model of a computer
- The Turing Machine:
  - The memory is a long tape divided into cells
  - Each cell has a symbol in it
  - There's a 'head' always 'looking' at one cell

...	a	b	a	a	...

Computation and Calculation

Turing Machines

- Is a computer really doing (effective) calculations?
- To help see why this is plausible, we'll look at a slightly different model of a computer
- The Turing Machine:
  - The memory is a long tape divided into cells
  - Each cell has a symbol in it
  - There's a 'head' always 'looking' at one cell

...	a	b	a	a	...

- 3 machine commands: write, move head left ($\ll$), move head right ($\gg$)

Turing Machines

And Programs

- The role of ‘instructions’ in a Turing Machine are played by a program

Turing Machines

And Programs

- The role of 'instructions' in a Turing Machine are played by a program
- They provide a machine command for each combination of two factors:
  1. The symbol currently detected by the head
  2. The current state of the FSM

Turing Machines

And Programs

- The role of ‘instructions’ in a Turing Machine are played by a program
- They provide a machine command for each combination of two factors:
    1. The symbol currently detected by the head
    2. The current state of the FSM
- And, it tells it which state to transition to

Turing Machines

And Programs

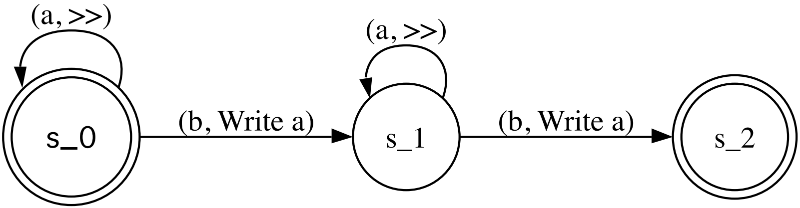
- The role of ‘instructions’ in a Turing Machine are played by a program
- They provide a machine command for each combination of two factors:
  1. The symbol currently detected by the head
  2. The current state of the FSM
- And, it tells it which state to transition to

	a	b
s_0	$\gg, s_0$	Write a, s_1
s_1	$\gg, s_1$	Write a, s_2
s_2	End State	End State

Visualizing A Turing Machine

Executing This Program

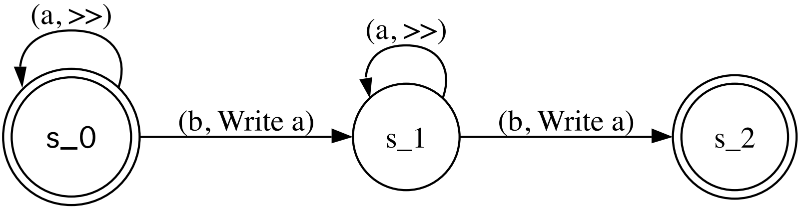
	a	b
s_0	$\gg, s_0$	Write a, s_1
s_1	$\gg, s_1$	Write a, s_2
s_2	End State	End State



Visualizing A Turing Machine

Executing This Program

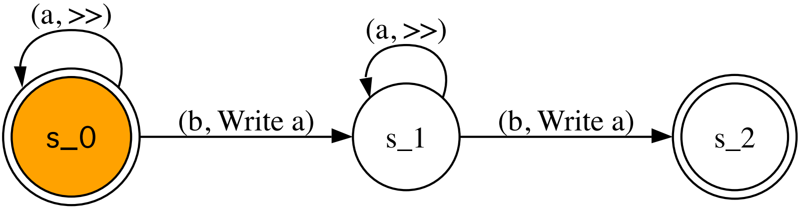
	a	b
s_0	$\gg, s_0$	Write a, s_1
s_1	$\gg, s_1$	Write a, s_2
s_2	End State	End State



Visualizing A Turing Machine

Executing This Program

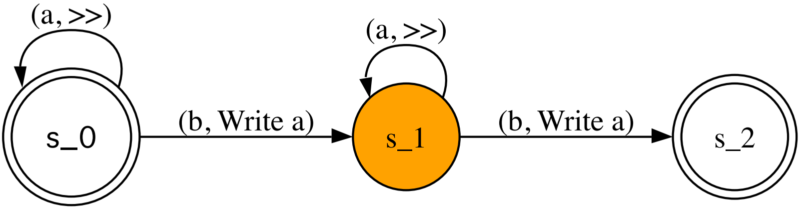
	a	b
s_0	$\gg, s_0$	Write a, s_1
s_1	$\gg, s_1$	Write a, s_2
s_2	End State	End State



Visualizing A Turing Machine

Executing This Program

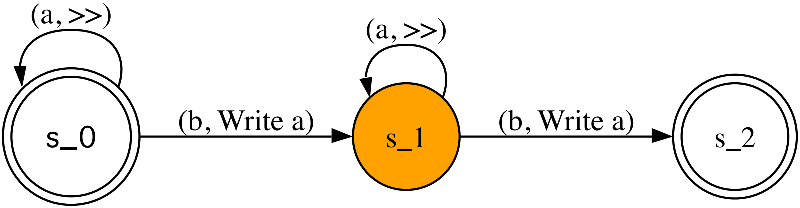
	a	b
s_0	$\gg$, s_0	Write a, s_1
s_1	$\gg$, s_1	Write a, s_2
s_2	End State	End State



Visualizing A Turing Machine

Executing This Program

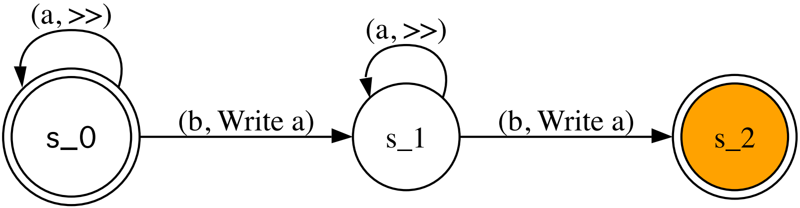
	a	b
s_0	$\gg$, s_0	Write a, s_1
s_1	$\gg$, s_1	Write a, s_2
s_2	End State	End State



Visualizing A Turing Machine

Executing This Program

	a	b
s_0	$\gg$, s_0	Write a, s_1
s_1	$\gg$, s_1	Write a, s_2
s_2	End State	End State



A Question

About Programs and Turing Machines

Question

When a Turing Machine executes a program, what part of the machine is causing it to follow the routine? Where is the program in the machine and how does it cause the memory and states to change?

A Question

About Programs and Turing Machines

Question

When a Turing Machine executes a program, what part of the machine is causing it to follow the routine? Where is the program in the machine and how does it cause the memory and states to change?

- Recall that the Turing Machine consists of the tape assembly wired to a FSM

A Question

About Programs and Turing Machines

Question

When a Turing Machine executes a program, what part of the machine is causing it to follow the routine? Where is the program in the machine and how does it cause the memory and states to change?

- Recall that the Turing Machine consists of the tape assembly wired to a FSM
- A Turing Machine can be programmed by constructing the internal configuration and wiring of the FSM in a particular way

A Question

About Programs and Turing Machines

Question

When a Turing Machine executes a program, what part of the machine is causing it to follow the routine? Where is the program in the machine and how does it cause the memory and states to change?

- Recall that the Turing Machine consists of the tape assembly wired to a FSM
- A Turing Machine can be programmed by constructing the internal configuration and wiring of the FSM in a particular way
- But Turing showed there's another possibility

Programs and The Universal Turing Machine

The Programs Can Also be Stored on the Tape!

- Turing's showed that it is possible to transcribe any program table into binary code

Programs and The Universal Turing Machine

The Programs Can Also be Stored on the Tape!

- Turing's showed that it is possible to transcribe any program table into binary code
- Given a certain minimal kind of wiring and configuration of the FSM, loading a tape with this code on it will cause the machine to operate exactly like 'hard-programmed' machine!

Programs and The Universal Turing Machine

The Programs Can Also be Stored on the Tape!

- Turing's showed that it is possible to transcribe any program table into binary code
- Given a certain minimal kind of wiring and configuration of the FSM, loading a tape with this code on it will cause the machine to operate exactly like 'hard-programmed' machine!
- This is what's called the Universal Turing Machine

Programs and The Universal Turing Machine

The Programs Can Also be Stored on the Tape!

- Turing's showed that it is possible to transcribe any program table into binary code
- Given a certain minimal kind of wiring and configuration of the FSM, loading a tape with this code on it will cause the machine to operate exactly like 'hard-programmed' machine!
- This is what's called the Universal Turing Machine

Universal Turing Machine (UTM)

Given the appropriate tape, the UTM can do anything that any other Turing Machine can do. With the right software, it can do anything! (Though maybe much slower!)

The Church-Turing Thesis

Effective Calculation

The Church-Turing Thesis (Widely Accepted Conjecture)

Any effective calculation can be performed by a Turing Machine.

- Made plausible by similarity between familiar idea of calculation and the way a Turing Machine computes

The Church-Turing Thesis

Effective Calculation

The Church-Turing Thesis (Widely Accepted Conjecture)

Any effective calculation can be performed by a Turing Machine.

- Made plausible by similarity between familiar idea of calculation and the way a Turing Machine computes
- This thesis implies that any effective calculation can be performed by a particular Turing Machine

The Church-Turing Thesis

Effective Calculation

The Church-Turing Thesis (Widely Accepted Conjecture)

Any effective calculation can be performed by a Turing Machine.

- Made plausible by similarity between familiar idea of calculation and the way a Turing Machine computes
- This thesis implies that any effective calculation can be performed by a particular Turing Machine
  - The Universal Turing Machine

The Church-Turing Thesis

Effective Calculation

The Church-Turing Thesis (Widely Accepted Conjecture)

Any effective calculation can be performed by a Turing Machine.

- Made plausible by similarity between familiar idea of calculation and the way a Turing Machine computes
- This thesis implies that any effective calculation can be performed by a particular Turing Machine
  - The Universal Turing Machine
- If thought just is effective calculation, then the UTM can do it

The Church-Turing Thesis

Effective Calculation

The Church-Turing Thesis (Widely Accepted Conjecture)

Any effective calculation can be performed by a Turing Machine.

- Made plausible by similarity between familiar idea of calculation and the way a Turing Machine computes
- This thesis implies that any effective calculation can be performed by a particular Turing Machine
  - The Universal Turing Machine
- If thought just is effective calculation, then the UTM can do it

Outline

Alternative Computation?

Is Turing All There Is?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Turing Machines work with bits: 0s and 1s

Alternative Computation?

Is Turing All There Is?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Turing Machines work with bits: 0s and 1s
- Surely, 3-way signals would make it more powerful

Alternative Computation?

Is Turing All There Is?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Turing Machines work with bits: 0s and 1s
- Surely, 3-way signals would make it more powerful
- Surely this would allow you to do more!

Alternative Computation?

Is Turing All There Is?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Turing Machines work with bits: 0s and 1s
- Surely, 3-way signals would make it more powerful
- Surely this would allow you to do more!
- Well, think back to our discussion of Rock-Paper-Scissors

Rock-Paper-Scissors

Not A Binary Input, Not A Binary 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

Alternative Computation?

Bits Can Do Anything Other Finite Signals Can

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Any process performed with 3-way signals can also be performed with bits

Alternative Computation?

Bits Can Do Anything Other Finite Signals Can

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Any process performed with 3-way signals can also be performed with bits
- In general this is true for any signal that distinguishes finitely many possibilities

Alternative Computation?

Bits Can Do Anything Other Finite Signals Can

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Any process performed with 3-way signals can also be performed with bits
- In general this is true for any signal that distinguishes finitely many possibilities
- So having the system built on more sophisticated signals doesn't really change anything

Alternative Computation?

Bits Can Do Anything Other Finite Signals Can

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- So a three-state computer would be able to do no more than a two-state computer.

Alternative Computation?

Bits Can Do Anything Other Finite Signals Can

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- So a three-state computer would be able to do no more than a two-state computer.
- Because you can simulate the one using the latter

Alternative Computation?

Bits Can Do Anything Other Finite Signals Can

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- So a three-state computer would be able to do no more than a two-state computer.
- Because you can simulate the one using the latter
- So having the system built on more sophisticated signals doesn't really change anything

Alternative Computation?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Okay, what about analog signals?

Alternative Computation?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Okay, what about analog signals?
  - Signals that can come in infinitely different varieties

Alternative Computation?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Okay, what about analog signals?
  - Signals that can come in infinitely different varieties
  - That is, instead of 0 and 1, any fraction of 1

Alternative Computation?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Okay, what about analog signals?
  - Signals that can come in infinitely different varieties
  - That is, instead of 0 and 1, any fraction of 1
  - Represented to any level of precision (decimal points!)

Alternative Computation?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Okay, what about analog signals?
  - Signals that can come in infinitely different varieties
  - That is, instead of 0 and 1, any fraction of 1
  - Represented to any level of precision (decimal points!)
- These exist and are called analog computers

Alternative Computation?

Question

Are there any kinds of computers more powerful than Turing Machines/modern computers? (Setting aside efficiency.)

- Okay, what about analog signals?
  - Signals that can come in infinitely different varieties
  - That is, instead of 0 and 1, any fraction of 1
  - Represented to any level of precision (decimal points!)
- These exist and are called analog computers
- So are they more powerful?

Analog Computation

Noise

- Instead of responding differently to some/no voltage, it will respond differently to any fraction of voltage differently

Analog Computation

Noise

- Instead of responding differently to some/no voltage, it will respond differently to any fraction of voltage differently
- The problem with this method is noise

Analog Computation

Noise

- Instead of responding differently to some/no voltage, it will respond differently to any fraction of voltage differently
- The problem with this method is noise
- Modern physics tells us that at the level of electrons, there's truly random things happening

Analog Computation

Noise

- Instead of responding differently to some/no voltage, it will respond differently to any fraction of voltage differently
- The problem with this method is noise
- Modern physics tells us that at the level of electrons, there's truly random things happening
- They will interact with the analog signal

Analog Computation

Noise

- Instead of responding differently to some/no voltage, it will respond differently to any fraction of voltage differently
- The problem with this method is noise
- Modern physics tells us that at the level of electrons, there's truly random things happening
- They will interact with the analog signal
- As a result, when the signal reaches the switches it will not reflect each nuance of the actual input

Analog Computation

Noise

- Instead of responding differently to some/no voltage, it will respond differently to any fraction of voltage differently
- The problem with this method is noise
- Modern physics tells us that at the level of electrons, there's truly random things happening
- They will interact with the analog signal
- As a result, when the signal reaches the switches it will not reflect each nuance of the actual input
- To control for this noise you have to ignore some of the nuances of the signal

Analog Computation

Noise

- So to control for noise, you have to ignore some of the analog nuance some decimal places

Analog Computation

Noise

- So to control for noise, you have to ignore some of the analog nuance some decimal places
- Even in very good circuits you have to ignore enough to be back to recognizing only finitely many different messages

Analog Computation

Noise

- So to control for noise, you have to ignore some of the analog nuance some decimal places
- Even in very good circuits you have to ignore enough to be back to recognizing only finitely many different messages
- The level of randomness appears to be high enough such that the infinite nuance of analog computers will never be useable in our universe

Analog Computation

Noise

- So to control for noise, you have to ignore some of the analog nuance some decimal places
- Even in very good circuits you have to ignore enough to be back to recognizing only finitely many different messages
- The level of randomness appears to be high enough such that the infinite nuance of analog computers will never be useable in our universe
- Could computers somehow be built to harness the randomness of the quantum, sub-atomic world?

Analog Computation

Noise

- So to control for noise, you have to ignore some of the analog nuance some decimal places
- Even in very good circuits you have to ignore enough to be back to recognizing only finitely many different messages
- The level of randomness appears to be high enough such that the infinite nuance of analog computers will never be useable in our universe
- Could computers somehow be built to harness the randomness of the quantum, sub-atomic world?
  - The jury's still out on that one

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Suppose it is, call it Test-for-Halt

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Suppose it is, call it Test-for-Halt
- I can then write another program called Paradox which itself contains Test-for-Halt

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Suppose it is, call it Test-for-Halt
- I can then write another program called Paradox which itself contains Test-for-Halt
- Further, Paradox can be written so that here Test-for-Halt gets applied to Paradox itself

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Suppose it is, call it Test-for-Halt
- I can then write another program called Paradox which itself contains Test-for-Halt
- Further, Paradox can be written so that here Test-for-Halt gets applied to Paradox itself
- If Test-for-Halt says yes, Paradox goes into an infinite loop

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Suppose it is, call it Test-for-Halt
- I can then write another program called Paradox which itself contains Test-for-Halt
- Further, Paradox can be written so that here Test-for-Halt gets applied to Paradox itself
- If Test-for-Halt says yes, Paradox goes into an infinite loop

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Now feed Paradox into Test-for-Halt

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Now feed Paradox into Test-for-Halt
- Suppose Test-for-Halt says 'yes'

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Now feed Paradox into Test-for-Halt
- Suppose Test-for-Halt says 'yes'
  - Then it's wrong, Paradox would've gone into an infinite loop

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Now feed Paradox into Test-for-Halt
- Suppose Test-for-Halt says 'yes'
  - Then it's wrong, Paradox would've gone into an infinite loop
- Suppose Test-for-Halt says 'no'

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Now feed Paradox into Test-for-Halt
- Suppose Test-for-Halt says 'yes'
    - Then it's wrong, Paradox would've gone into an infinite loop
- Suppose Test-for-Halt says 'no'
    - Then it's also wrong, Paradox would've halted

The Halting Problem

Something Computers Can't Do

The Halting Problem

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- Now feed Paradox into Test-for-Halt
- Suppose Test-for-Halt says 'yes'
  - Then it's wrong, Paradox would've gone into an infinite loop
- Suppose Test-for-Halt says 'no'
  - Then it's also wrong, Paradox would've halted

Answer: No, it's not possible. You can always build a

Algorithms

Algorithms

Algorithms

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- An algorithm is a fail-safe procedure that, if executed correctly, guarantees the result.

Algorithms

Algorithms

Algorithms

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- An algorithm is a fail-safe procedure that, if executed correctly, guarantees the result.
- Programs are representations of algorithms in some programming language.

Algorithms

Algorithms

Algorithms

Is it possible to write a program that inspects any other programs and determines whether or not they will eventually stop?

- An algorithm is a fail-safe procedure that, if executed correctly, guarantees the result.
- Programs are representations of algorithms in some programming language.
- There are usually several ways of expressing in algorithms, depending on the programming language.

Algorithms

Comparing Algorithms

- Algorithms are compared on the basis of how fast they are.

Algorithms

Comparing Algorithms

- Algorithms are compared on the basis of how fast they are.
- But how algorithms' speed can be compared?

Algorithms

Comparing Algorithms

- Algorithms are compared on the basis of how fast they are.
- But how algorithms' speed can be compared?
  - Algorithms can be implemented in different ways!

Algorithms

Comparing Algorithms

- Algorithms are compared on the basis of how fast they are.
- But how algorithms' speed can be compared?
  - Algorithms can be implemented in different ways!
  - Algorithms can be applied to programs of different size!

Algorithms

Comparing Algorithms

- Algorithms are compared on the basis of how fast they are.
- But how algorithms' speed can be compared?
  - Algorithms can be implemented in different ways!
  - Algorithms can be applied to programs of different size!

Algorithms

Comparing Algorithms

- Algorithms are compared on the basis of how fast they are.
- But how algorithms' speed can be compared?
  - Algorithms can be implemented in different ways!
  - Algorithms can be applied to programs of different size!

Answer: We can describe the speed of an algorithm on the basis of how much the time required for performing the task grows along with the size of the problem.

Algorithms

Comparing Algorithms

First sock-matching algorithm

- Pull out a random sock out of the basket.
- Pull out a second sock out of the basket.
- Compare it with the first pulled out.
- If it matches, bundle them together.
- if it does not match, throw it back in the basket and restart the process.

Algorithms

Comparing Algorithms

Second sock-matching algorithm

- Pull out a random sock out of the basket on the table.
- Pull out a second sock out of the basket.
- Compare it with the first pulled out.
- If it matches with any sock on the table, bundle them together.
- if it does not match, set it next to the first.
- Repeat the process until all the socks are matched.

Algorithms

Comparing Algorithms

Compare the first and the second algorithm

- Assume that there are n socks in the basket.
- Using the first algorithm, finding two that match requires pulling out and pulling back an average of half of the remaining socks.
- So the number of sock removals is proportional to the square of number of socks.
- The algorithm is of order n^2, meaning that for large problem, the time of execution grows as the square of the problem size.

Algorithms

Comparing Algorithms

Compare the first and the second algorithm

- In other words, if there are m times as many socks in the basket, the time of execution of the second algorithm will only by m by n.
- The time of execution of the first algorithm will be instead m^2 by n.
- So if n=10, then using the second algorithm to match twice as many socks will take twice as much time.
- Instead, if n=10, then using the first algorithm to match twice as many socks will take four times as much time.

Algorithms

Comparing Algorithms

Compare the first and the second algorithm

- In other words, if there are m times as many socks in the basket, the time of execution of the second algorithm will only by m by n.
- The time of execution of the first algorithm will be instead m^2 by n.
- So if n=10, then using the second algorithm to match twice as many socks will take twice as much time.
- Instead, if n=10, then using the first algorithm to match twice as many socks will take four times as much time.