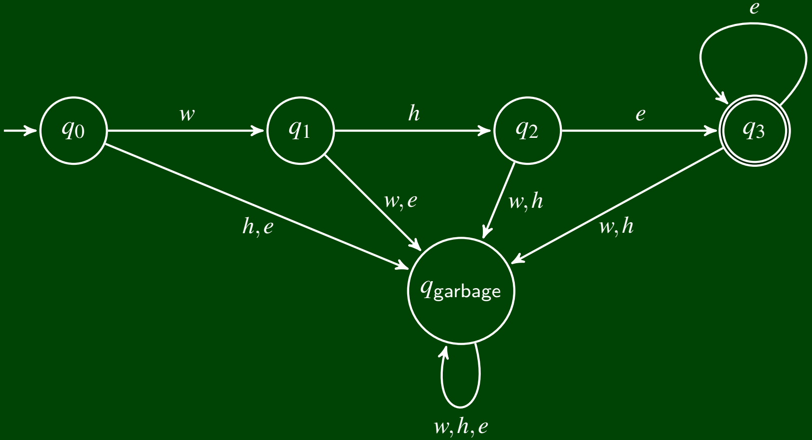
CSE 120

Computer Science Principles

Proofs & Computation



At the very "lowest" level is hardware which Justin has talked about.

At the very "highest" level is Theory which is what today is about!

In this lecture, we will explore the **abstract**! And we will apply it to **computation**!

But we start simple...

How many numbers are there?

In this lecture, we will explore the **abstract**! And we will apply it to **computation**!

But we start simple...

How many numbers are there?

...Infinity, of course!

What's the biggest number you can name?

What's the biggest number you can name?

0, 1, 2, . . . , 4000000000000000, . . .

If you give me a number, I can get a bigger one by adding 1:

$$x \mapsto x + 1$$

If we collect all of these numbers together, we call the resulting set "the natural numbers".

Imagine an incredibly large (infinite, actually) index of numbers:

0:

1:

2:

3:

4:

5:

6:

7:

...

We say a set of numbers is **countable** (or the same size as the natural numbers) whenever we can **list them out**.

Even Numbers

"Obvious" Theorem

There are as many even numbers as odd numbers.

"Obvious" Theorem

There are as many even numbers as odd numbers.

Are there more even numbers than natural numbers?

0:

1:

2:

3:

4:

5:

6:

7:

Are there more integers than natural numbers?

0:

1:

2:

3:

4:

5:

6:

7:

8:

...

Are there more fractions than natural numbers?

0:

1:

2:

3:

4:

5:

6:

7:

8:

...

Are there more Strings than natural numbers?

Program

1. List out Strings of length 1:
   0. a
   1. b
   2. c
   3. ...
2. List out Strings of length 2:
   4. aa
   5. ab
   6. ac
   7. ...
3. List out Strings of length 3:
   7. aaa
   8. aab
   9. aac
   10. ...
4. ...

Are there more real numbers than natural numbers?

Incredibly, this is enough machinery to prove interesting results.

Incredibly, this is enough machinery to prove interesting results.

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Example

- "one"

Incredibly, this is enough machinery to prove interesting results.

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Example

- "one"
- "two"

Incredibly, this is enough machinery to prove interesting results.

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Example

- "one"
- "two"
- "π"

Incredibly, this is enough machinery to prove interesting results.

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Example

- "one"
- "two"
- "π"
- "the smallest number with four million digits"

Describable Numbers

Incredibly, this is enough machinery to prove interesting results.

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Example

- "one"
- "two"
- "π"
- "the smallest number with four million digits"

Describable Numbers

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Definition (Interesting)

A number is **interesting** when it's the smallest number with some interesting property.

Describable Numbers

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Definition (Interesting)

A number is **interesting** when it's the smallest number with some interesting property.

Example

- 0 is interesting because it's "the smallest non-negative number"

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Definition (Interesting)

A number is **interesting** when it's the smallest number with some interesting property.

Example

- 0 is interesting because it's "the smallest non-negative number"
- 1 is interesting because it's "$1 \times x = x$ for all x"

Describable Numbers

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Definition (Interesting)

A number is **interesting** when it's the smallest number with some interesting property.

Example

- 0 is interesting because it's "the smallest non-negative number"
- 1 is interesting because it's "$1 \times x = x$ for all x"
- 2 is interesting because it's "the smallest prime number"

Describable Numbers

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Definition (Interesting)

A number is **interesting** when it's the smallest number with some interesting property.

Questions

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Definition (Interesting)

A number is **interesting** when it's the smallest number with some interesting property.

Questions

- What is the smallest uninteresting number?

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Definition (Interesting)

A number is **interesting** when it's the smallest number with some interesting property.

Questions

- What is the smallest uninteresting number?
- Is every interesting number describable?

Definition (Describable)

A number is **describable** when it can **unambiguously** be described by some String.

Definition (Interesting)

A number is **interesting** when it's the smallest number with some interesting property.

Questions

- What is the smallest uninteresting number?
- Is every interesting number describable?
- Is every real number describable?

Definition (Computable)

A number is **computable** when it can **unambiguously** printed out by some program.

Example

- 0 is interesting because `text("0", 0, 0)`
- 1 is interesting because `text("1", 0, 0)`
- π is interesting because...

Question

- Is every number computable?

We now know there **is** something that isn't computable. But can we find something specific?

We now know there **is** something that isn't computable. But can we find something specific?

Halting Problem

Given a program P as input, can we determine if it ever finishes running?

We now know there **is** something that isn't computable. But can we find something specific?

Halting Problem

Given a program P as input, can we determine if it ever finishes running?

It turns out the answer is no!

We now know there **is** something that isn't computable. But can we find something specific?

Halting Problem

Given a program P as input, can we determine if it ever finishes running?

It turns out the answer is no!

The Idea

Hypothetically, consider what would happen if someone really smart has written a program:

We now know there **is** something that isn't computable. But can we find something specific?

Halting Problem

Given a program P as input, can we determine if it ever finishes running?

It turns out the answer is no!

The Idea

Hypothetically, consider what would happen if someone really smart has written a program:

HALT(P) which returns true when P finishes and false if it doesn't.

We now know there **is** something that isn't computable. But can we find something specific?

Halting Problem

Given a program P as input, can we determine if it ever finishes running?

It turns out the answer is no!

The Idea

Hypothetically, consider what would happen if someone really smart has written a program:

HALT(P) which returns true when P finishes and false if it doesn't.

Then, we will find a program CONFUSE which will confuse the HALT program. . .

We now know there **is** something that isn't computable. But can we find something specific?

Halting Problem

Given a program P as input, can we determine if it ever finishes running?

It turns out the answer is no!

The Idea

Hypothetically, consider what would happen if someone really smart has written a program:

HALT(P) which returns true when P finishes and false if it doesn't.

Then, we will find a program CONFUSE which will confuse the HALT program. . . which means it doesn't work.

We now know there **is** something that isn't computable. But can we find something specific?

Halting Problem

Given a program P as input, can we determine if it ever finishes running?

It turns out the answer is no!

The Idea

Hypothetically, consider what would happen if someone really smart has written a program:

HALT(P) which returns true when P finishes and false if it doesn't.

Then, we will find a program CONFUSE which will confuse the HALT program...which means it doesn't work. So, it can't be written!

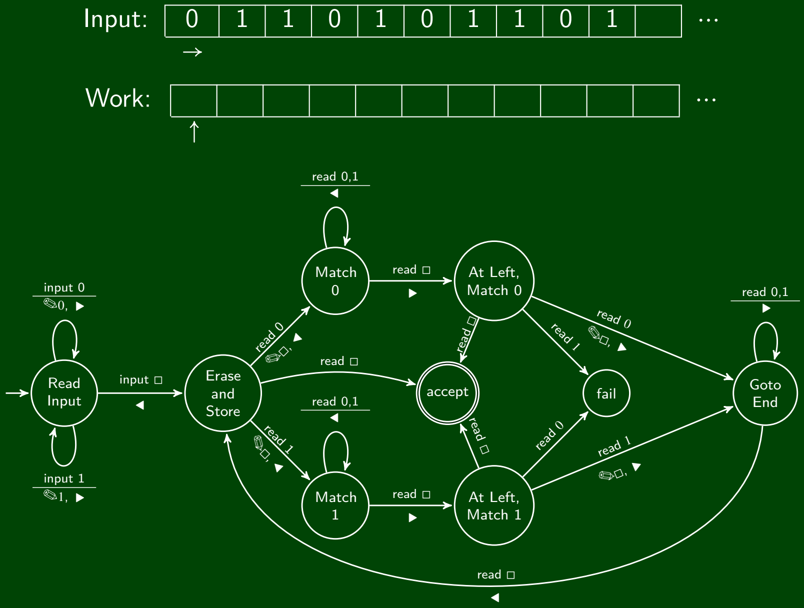
Suppose we have a program HALT such that:

HALT(P) returns true when P finishes and false if it doesn't.

Our Program

```
void CONFUSE() {
   if (HALT(SOURCE_CODE(CONFUSE))) {
      while (true) {
         text("ha ha", 0, 0);
      }
   }
   else {
      return;
   }
}
```

A Flow Chart



Input:
0 1 1 0 1 0 1 1 0 1 …
Work:
…
read 0,1
Match 0
read □
At Left, Match 0
read 0,1
input 0
✎0, ▶
Read Input
input □
Erase and Store
read 0
✎□, ▶
read □
accept
read □
read 1
read 0
✎□, ▶
fail
Goto End
input 1
✎1, ▶
read 0,1
read 1
✎□, ▶
Match 1
read □
At Left, Match 1
read □
read 0
read 1
✎□, ▶
read □

Some **infinite** tapes:

Some **infinite** tapes: (how many doesn't matter; one tape for input and work, etc.)

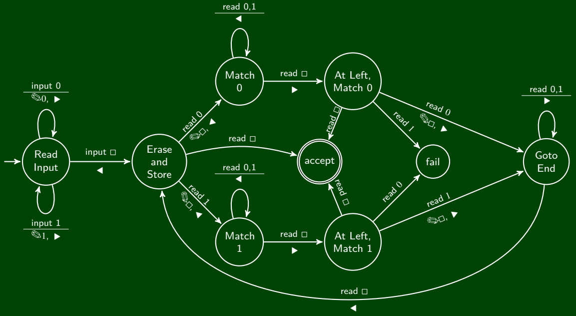
Input:	0	1	1	0	1	0	1	1	0	1		⋯
	→											
Work:												⋯
	↑											

Some **infinite** tapes: (how many doesn't matter; one tape for input and work, etc.)

Input: | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | | …
→

Work: | | | | | | | | | | | | …
↑

A **finite-state controller**:

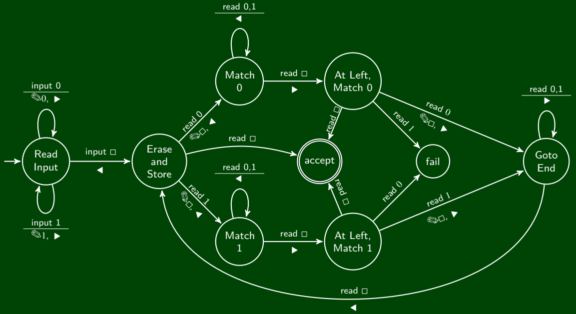


Some **infinite** tapes: (how many doesn't matter; one tape for input and work, etc.)

Input:	0	1	1	0	1	0	1	1	0	1		⋯
	→											

Work:												⋯
	↑											

A **finite-state controller**:



That's it. These things can decide **exactly the same languages** as register machines, and lambda calculus.