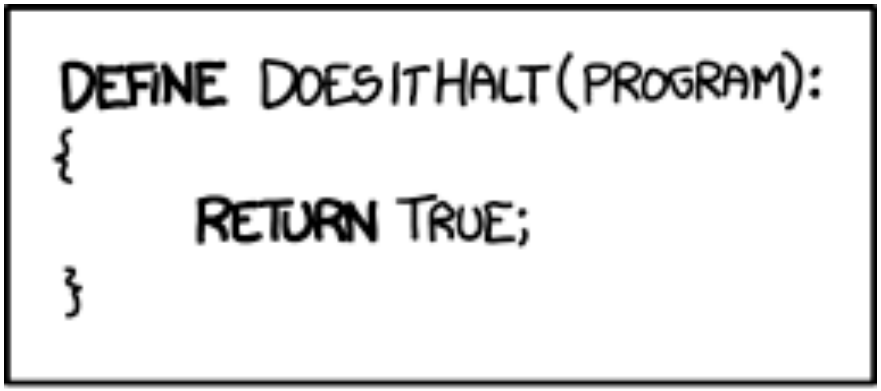
CSE 311: Foundations of Computing

Lecture 25: Limits of Computation



THE BIG PICTURE SOLUTION
TO THE HALTING PROBLEM

Showing a Language L is not regular $\{0^n 1^n : n \geq 0\}$

1. "Suppose for contradiction that some DFA M accepts L."
2. Consider an **INFINITE** set of "half strings" (which we intend to complete later). It is imperative that every string in our set have a **DIFFERENT, SINGLE** "accept" completion.
3. "Since **S** is infinite and **M** has finitely many states, there must be two strings s_i and s_j in **S** for some $i \neq j$ that end up at the same state of **M**."
4. Consider appending the (correct) completion to one of the two strings.
5. "Since s_i and s_j both end up at the same state of **M**, and we appended the same string **t**, both $s_i t$ and $s_j t$ end at the same state of **M**. Since $s_i t \in L$ and $s_j t \notin L$, **M** does not recognize **L**."
6. "Since **M** was arbitrary, no DFA recognizes **L**."

Prove A = {0^n1^n : n ≥ 0} is not regular

Suppose for contradiction that some DFA, M, accepts A.

Let S = $\{0^n : n \geq 0\}$

Since S is infinite, there are two strings 0^a and 0^b where $a \neq b$, $a, b \in \mathbb{N}$ and 0^a, 0^b both end in the same state

Append 1^a to both to get:

0^a1^a

0^b1^a

- Tokens
- Workshop today

Prove $A = \{0^n1^n : n \geq 0\}$ is not regular

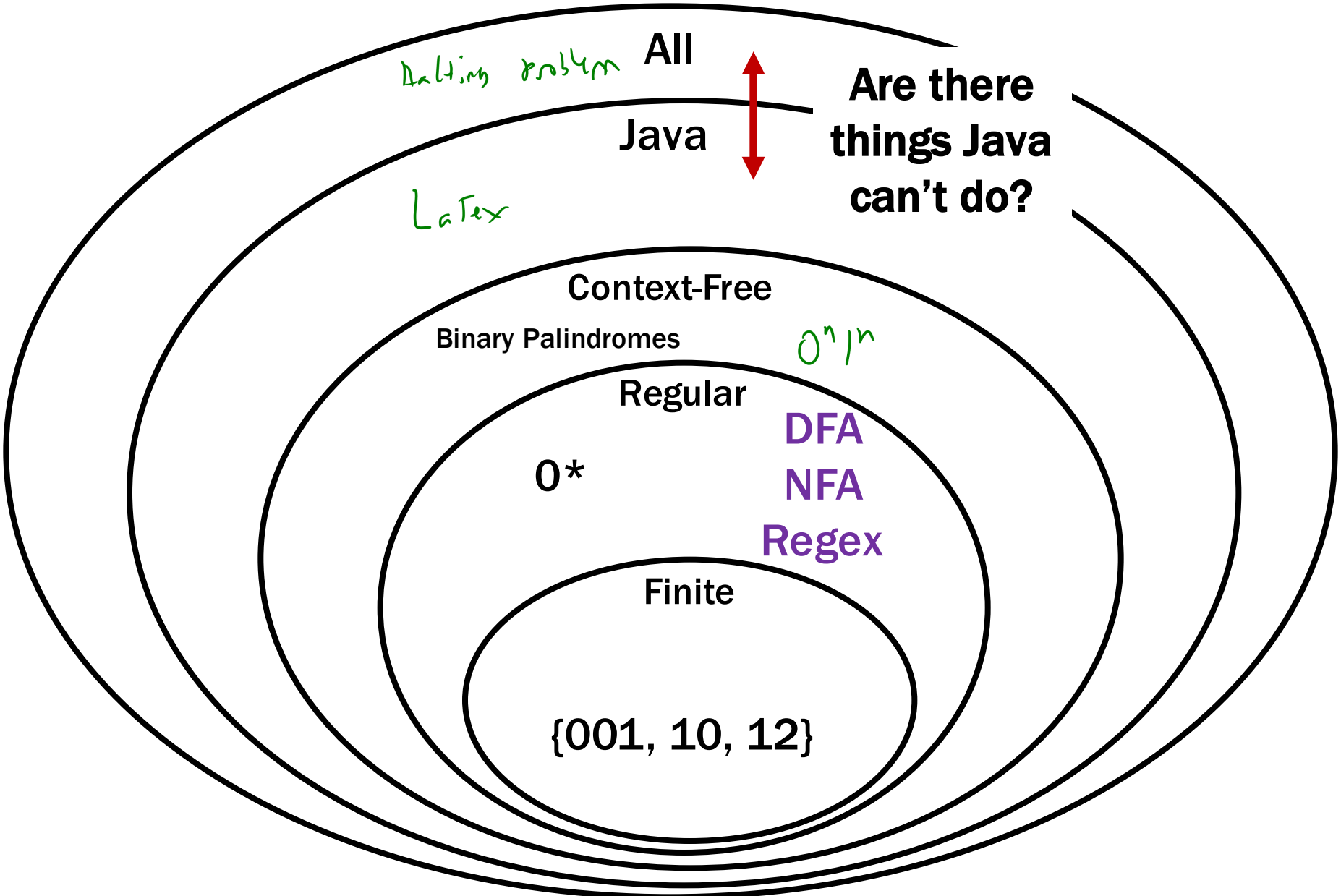
Suppose for contradiction that some DFA, M, accepts A.

Let $S = \{0^n : n \geq 0\}$. Since S is infinite and M has finitely many states, there must be two strings, 0^i and 0^j (for some $i \neq j$) that end in the same state in M.

Consider appending 1^i to both strings. Note that $0^i1^i \in A$, but $0^j1^i \notin A$ since $i \neq j$. But they both end up in the same state of M. Since that state can't be both an accept and reject state, M does not recognize A.

Since M was arbitrary, no DFA recognizes A.

Languages and Machines!



What We're About To Do....

Today, we will dispel the notion that Java is a magical language that allows us to solve any problem we want if we're smart enough.

An Assignment Too Simple for 142!

Students should write a Java program that…

- Prints "Hello" to the console
- Eventually exits

GradeIt, PracticeIt, etc. need to grade the students.

How do we write that grading program?

Follow Up Question

What does this program do?

```
_(__,___,____){___/__<=1?_(__,___+1,___
_):!(___%__)?_(__,___+1,0):___%__==___
/__&&!____?(printf("%d\t",___/__),_(__,_
__+1,0)):___%__>1&&___%__<___/__?_( __,1+
___,____+!(___/__%(___%__))):___<__*__
?_(__,___+1,____):0;}main(){_(100,0,0);}
```

Follow Up Question

```
public static int collatz(n) {
    if (n == 1) {
        return 1;
    }
    if (n % 2 == 0) {
        return collatz(n/2)
    }
    else {
        return collatz(3n + 1)
    }
}
```

What is in the set {x : collatz(n) = 1}?

Some Notation and Starting Ideas

We're going to be talking about *Java code* a lot.

CODE(**P**) will mean "the code of the program P"

So, consider the following function:

```
public String P(String x) {
    return new String(Arrays.sort(x.toCharArray());
}
```

What is **P**(CODE(**P**))?

"((()))..;AACPSSaaabceeggghiiiilnnnnnooprrrrrrrrrrrssstttttttuuwxxyy{}"

The Halting Problem

Given:

- CODE(**P**) for a program **P**

Output:

- **true** if **P** halts
- **false** if **P** does not halt

HALT("while(true);")

HALT("return 0;")

The "standard" version of the halting problem takes some number as input. We consider this one, because it's easier to think about.

Proof Strategy

Remember, this means X is a program and HALT(X) is true when X halts and false otherwise.

Imagine we had a HALT(X) function which solved the halting problem...

Our goal is to write a program that CONFUSES the function HALT so that it does the wrong thing.

```
public static void PROGRAM() {




}
```

Proof Strategy

Remember, this means X is a program and HALT(X) is true when X halts and false otherwise.

Imagine we had a HALT(X) function which solved the halting problem...

Our goal is to write a program that CONFUSES the function HALT so that it does the wrong thing.

```
public static void PROGRAM() {
   if (/* I should halt */) {
      /* don't halt */
   }
   else {
      /*    halt    */
   }
}
```

Proof Strategy

Remember, this means X is a program and HALT(X) is true when X halts and false otherwise.

Imagine we had a HALT(X) function which solved the halting problem...

Our goal is to write a program that CONFUSES the function HALT so that it does the wrong thing.

```
public static void PROGRAM() {
   if (/* I should halt */) {
      while (true);
   }
   else {
      return;
   }
}
```

Proof Strategy

Remember, this means X is a program and HALT(X) is true when X halts and false otherwise.

Imagine we had a HALT(X) function which solved the halting problem...

Our goal is to write a program that CONFUSES the function HALT so that it does the wrong thing.

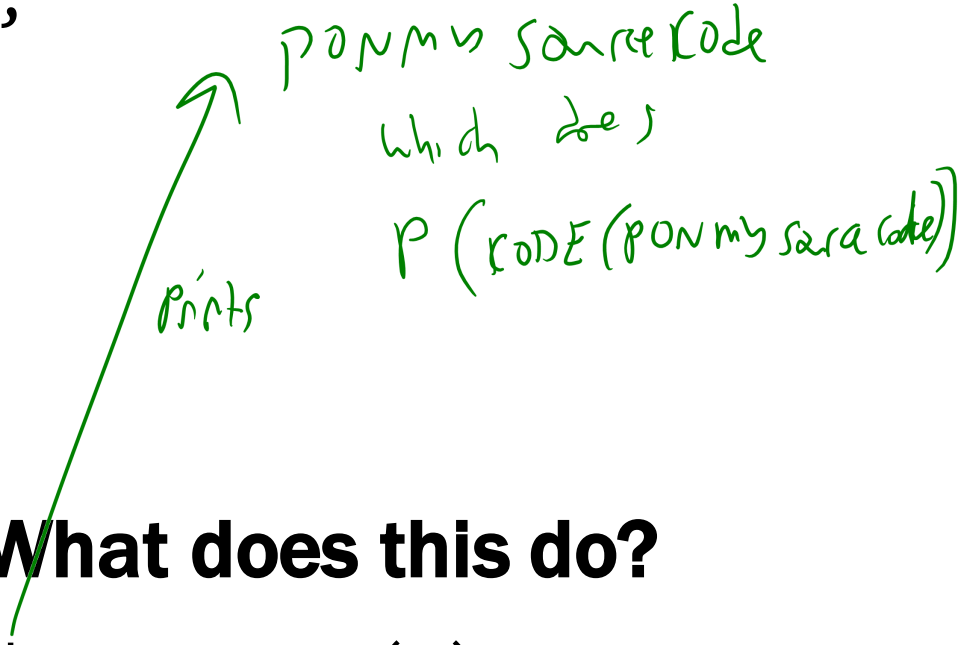
```
public static void PROGRAM() {
   if (HALT(MY_SOURCE_CODE)) {
      while (true);
   }
   else {
      return;
   }
}
```

Proof Strategy

Remember, this means X is a program and HALT(X) is true when X halts and false otherwise.

Suppose for contradiction we had a HALT(X) function which solved the halting problem...

```
public static void P(String input) {
   if (HALT(input)) {
      while (true);
   }
   else {
      return;
   }
}
```



Quick Question. What does this do?

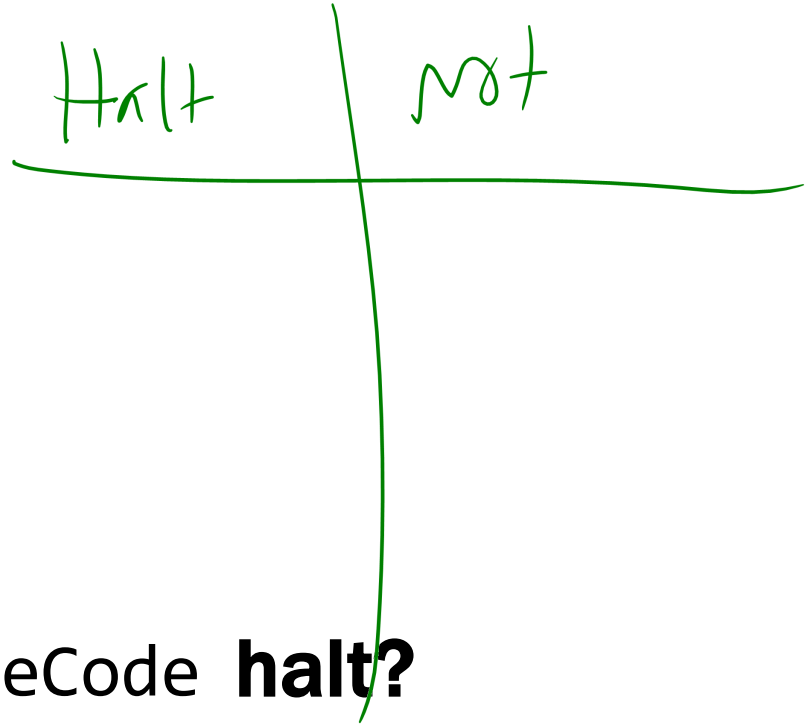
```
OnMySourceCodeGENERATOR(P)
```

Proof Strategy

Remember, this means X is a program and HALT(X) is true when X halts and false otherwise.

Suppose for contradiction we had a HALT(X) function which solved the halting problem...

```
public static void P(String input) {
   if (HALT(input)) {
      while (true);
   }
   else {
      return;
   }
}
```



Does `POnMySourceCode` **halt?**

```
public static void HALT(String input) {
   // We don't know how this works,
   // but we assume that it does.

   // So, if input is a program that
   // halts, then this returns true.
   // Otherwise, it returns false.
}
```

```
void P(String input) {
   if (HALT(input)) {
     while (true);
   }
   else {
     return;
   }
}
```

Does POnMySourceCode halt?

Recall that POnMySourceCode does the same thing as P(CODE(POnMySourceCode)).

```
void POnMySourceCode() {
   if (HALT(CODE(POnMySourceCode))) {
        while (true);
   }
   else {
     return;
   }
}
```

```
public static void HALT(String input) {
   // We don't know how this works,
   // but we assume that it does.

   // So, if input is a program that
   // halts, then this returns true
   // Otherwise, it returns false.
}
```

Suppose **POnMySourceCode** **halts**.

HALT(**CODE**(**POnMySourceCode**)) is true.

```
void POnMySourceCode() {
   if (HALT(CODE(POnMySourceCode))) {
        while (true);
   }
   else {
     return;
   }
}
```

So, this if statement is true!

So, the code loops forever!

This is a contradiction, so **POnMySourceCode** **does not halt**.

```
public static void HALT(String input) {
   // We don't know how this works,
   // but we assume that it does.

   // So, if input is a program that
   // halts, then this returns true.
   // Otherwise, it returns false.
}
```

Suppose **POnMySourceCode** **does not halt.**

HALT(CODE(POnMySourceCode)) is false.

```
void POnMySourceCode() {
   if (HALT(CODE(POnMySourceCode))) {
        while (true);
   }
   else {
     return;
   }
}
```

So, this if statement is false!

So, the code halts!

This is a contradiction, so **POnMySourceCode** **can't not halt.**

Suppose for contradiction we had a HALT(X) function which solved the halting problem…

Suppose POnMySourceCode **halts**.

Then, **HALT**(**CODE**(POnMySourceCode)) is true.

So, the if statement in POnMySourceCode **is true!**

So, the code loops forever!

This is a contradiction, so POnMySourceCode **does not halt**.

Suppose POnMySourceCode **does not halt**.

Then, **HALT**(**CODE**(POnMySourceCode)) is false.

So, the if statement in POnMySourceCode **is true!**

So, the code halts!

This is a contradiction, so POnMySourceCode **can't not halt**.

So, POnMySourceCode. So, P does not exist. So, **HALT** does not exist.

That's it!

- We proved that there is no Java program that can solve the Halting Problem.

- This tells us that there is no compiler that can check our programs and guarantee to find any infinite loops they might have.

~~That's it!~~

BUT WAIT...

THERE'S MORE!

```
public static void D(String input) {
   // Returns true if, when run, input
   // does X.
   // Otherwise, it returns false.
}
```

```
void P(String input) {
   if (D(input)) {
     D_IS_FALSE();
   }
   else {
     D_IS_TRUE();
   }
}
```

Is D(POnMySourceCode)true?

Recall that POnMySourceCode does the same thing as P(CODE(POnMySourceCode)).

```
void POnMySourceCode() {
   if (D(CODE(POnMySourceCode))) {
      D_IS_FALSE();
   }
   else {
      D_IS_TRUE();
   }
}
```

Rice's Theorem

- We've now proven that for any property about the "behavior" of programs, D, if…
  - There is some program `D_IS_FALSE()` for which D is false.
  - There is some program `D_IS_TRUE()` for which D is true.
- Then, D does not exist.

Rice's Theorem

- Does P have a NullPointerException?
- Do P and Q do the same thing?
- Does P output 0 on any input?
- Does P have a buffer overflow?
- Does P have a virus?
- Does P have "dead code"?
- ...

~~That's it!~~

BUT WAIT...

THERE'S MORE!

Church-Turing Thesis

> < .)

“All physically computable functions are Java-decidable”

That is, there is no programming language more powerful than Java.