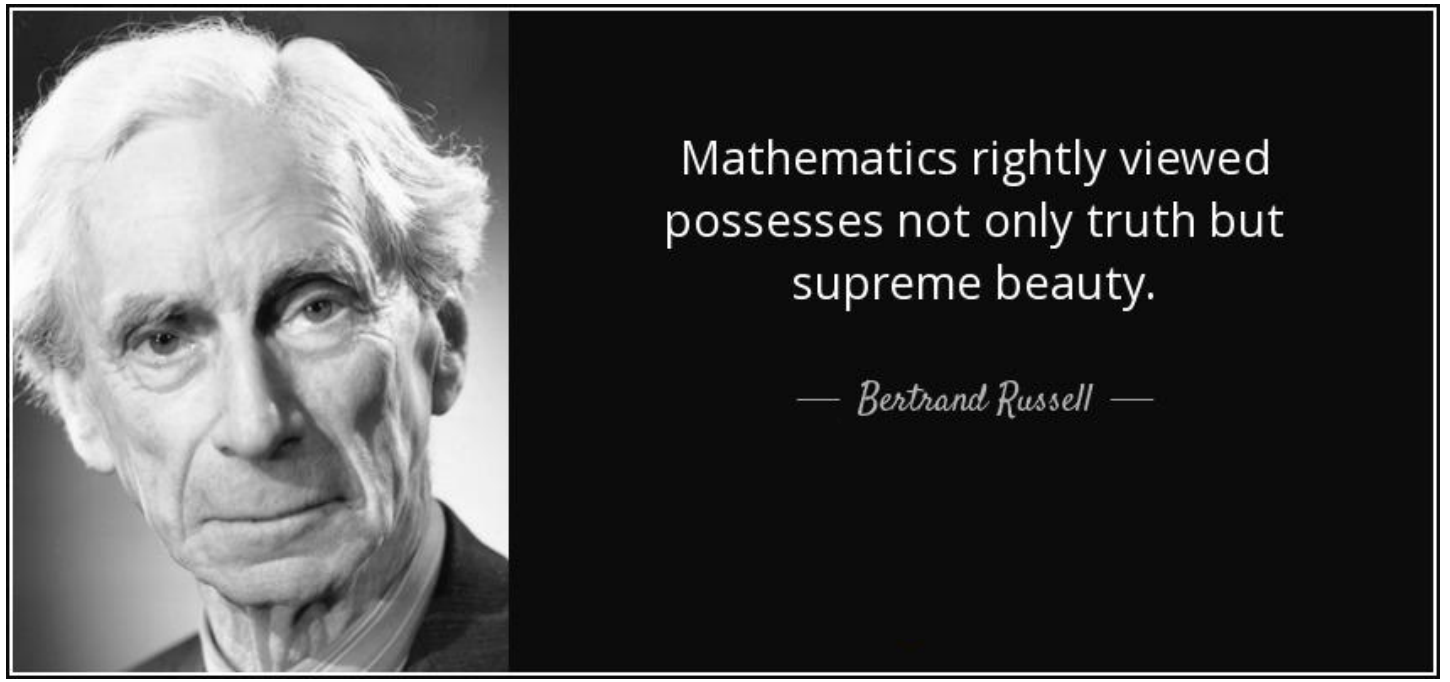


Limits of Computation

CSE 311: Foundations of Computing I
Lecture 24

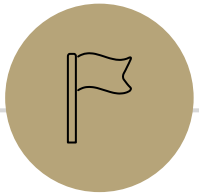
Announcements

- HW7 due tonight on Grin at 11:59 pm. No Late Days are permitted.
- All Section Makeups are due tonight on Gradescope at 11:59 pm.
- The Final Exam is in section tomorrow, and in lecture on Friday.
  - See Exams page on website for more details
  - A recording of yesterday's review session has been posted

Goals for Today

A final taste of more advanced CS Theory

- Goal 1: Prove that there exists a language that is irregular.
- Goal 2: Prove that there is a problem that no computer can solve.



Proof of Irregularity

Previous Results

Friday

Theorem: For every NFA, there is a DFA that recognizes the same language.

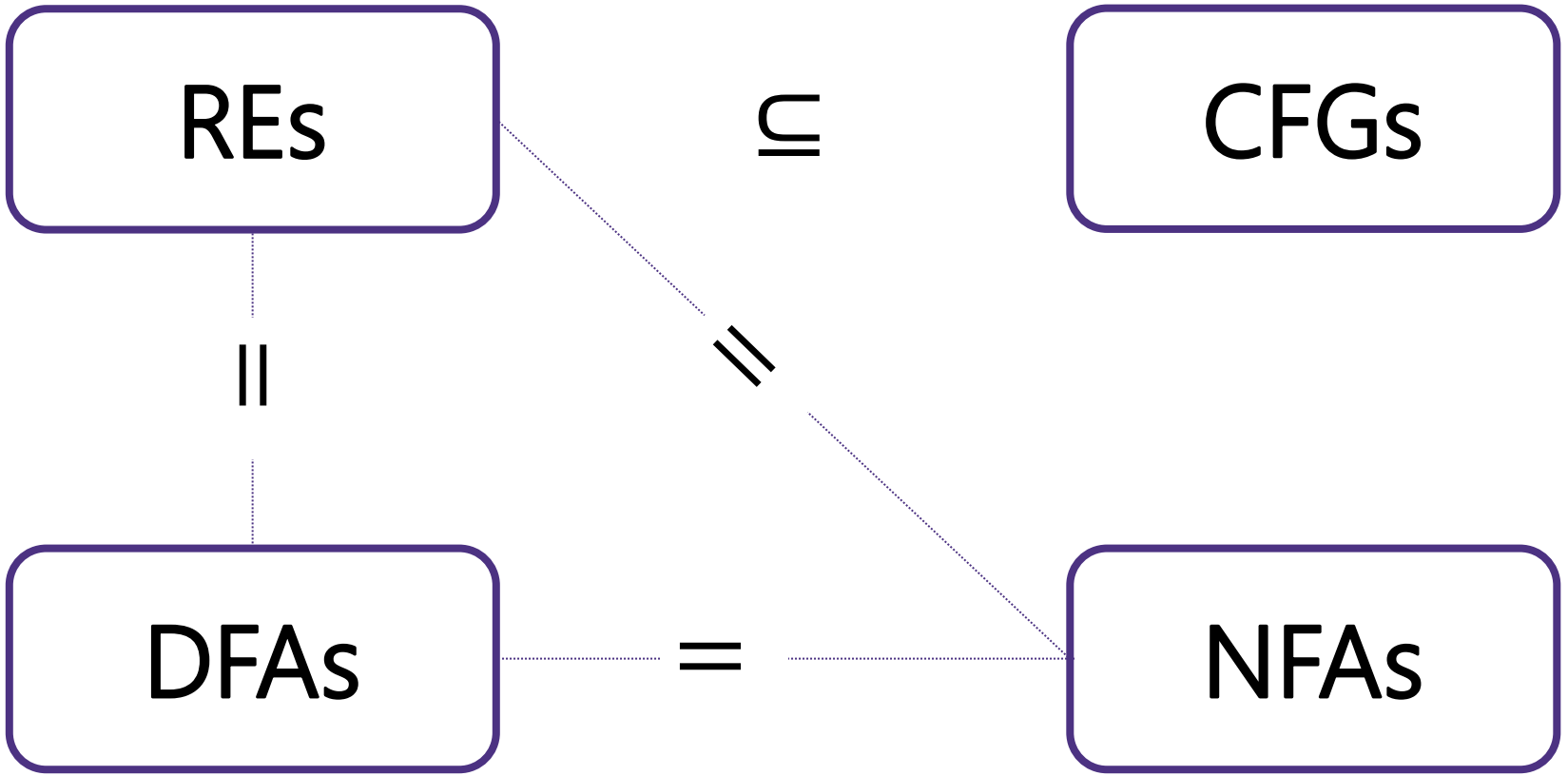
Monday

Theorem 1: For every Regular Expression, there is a CFG that generates the same language.

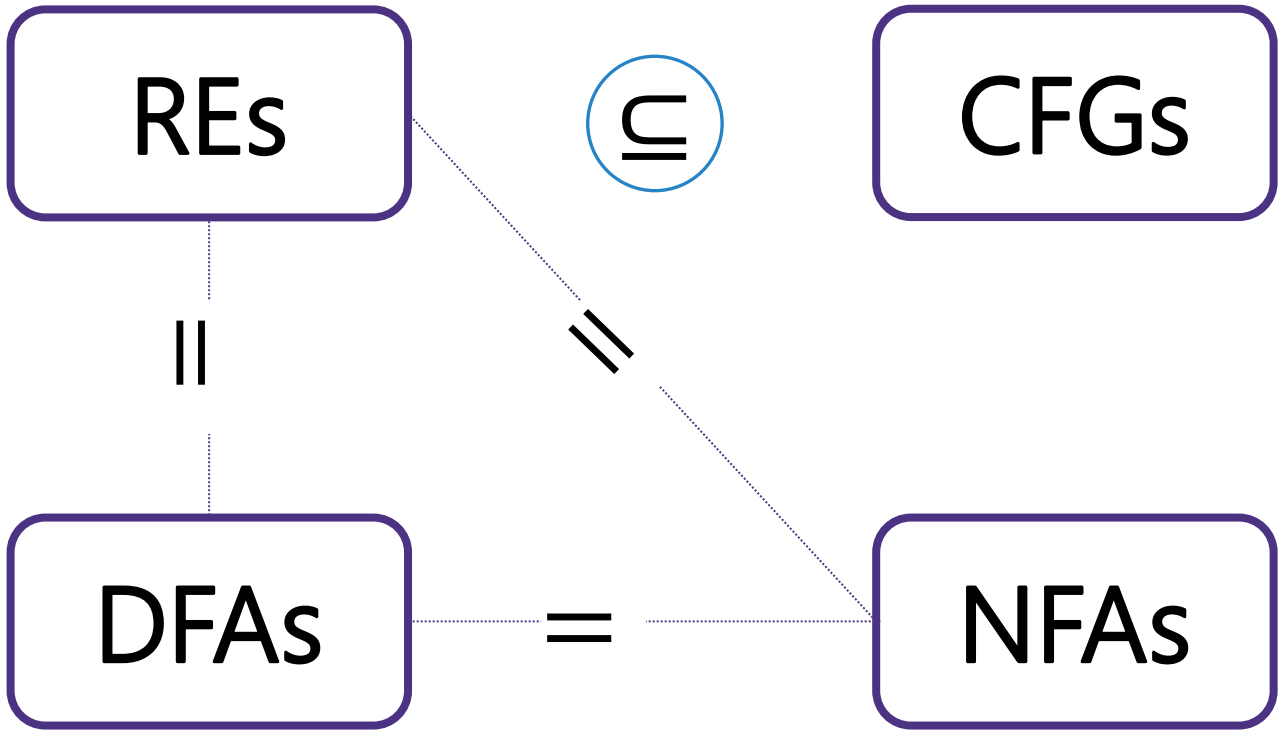
Theorem 2: For every Regular Expression, there is an NFA that recognizes the same language.

Theorem 3: For every NFA, there is a Regular Expression that matches the same language.

The story so far…



Our next goal…



We want to show that this ⊆ is actually ⊊

In other words, we want to show that there is at least 1 language that is context-free but not regular.

Our next goal…

Theorem 4:
There exists a language that is context-free but not regular.

The language $L = \{0^n 1^n : n \geq 0\}$ is Context-Free

Observe that $L = \{\varepsilon, 01, 0011, 000111, \ldots\}$. There is a CFG that generates L:

Is the language $L = \{0^n 1^n : n \geq 0\}$ Regular?

If L was Regular, then there would be a DFA, an NFA, and a RE for it.

Let's focus on the DFA. I claim it's not possible to construct a DFA for L.

Intuition (Not a Proof):

Q: What would a DFA need to keep track of to decide?

A: It would need to keep track of the number of 0s at the start, in order to check that there is an equal number of 1s after.
But there are an infinite # of possible 0s at the start, and we only have finitely many states.

Proving $L = \{0^n 1^n : n \geq 0\}$ is Irregular

- Proof Technique: _____
- We'll assume for the sake of contradiction that _____

  _____

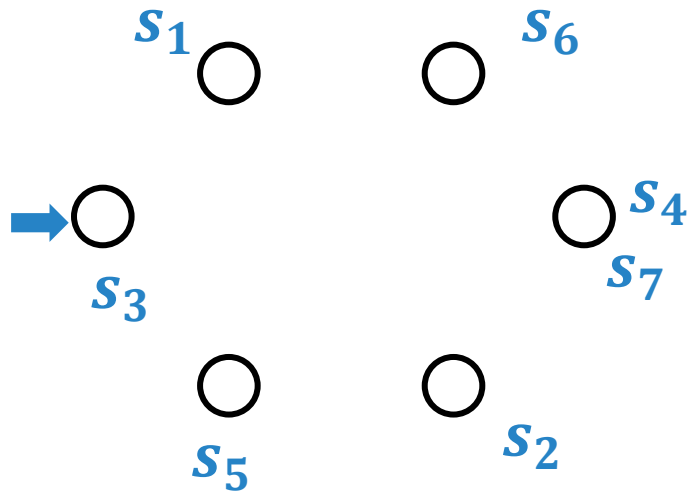
  - We could instead assume that there exists a RE or an NFA, but DFA will be easier.
- But what contradiction could we find?
  - We know that every state in a DFA is either accepting or rejecting.
  - The contradiction we'll find is that _____

    _____

Facts about DFAs

Fact 1: For any DFA and any infinite set S of strings over Σ, the DFA takes at least two strings in S to the same state.

The DFA can only have finitely many states.

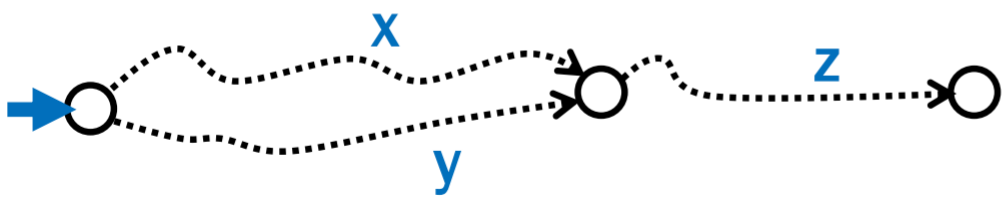


$$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7 \dots\}$$

Facts about DFAs

Fact 2: If a DFA takes the strings $x, y \in \Sigma^*$ to the same state, then for any string $z \in \Sigma^*$, the DFA takes xz and yz to the same state.

The DFA can't remember that the start of the string was x vs. y.



$x \bullet z = x_1 x_2 \ldots x_n z_1 z_2 \ldots z_k$

$y \bullet z = y_1 y_2 \ldots y_m z_1 z_2 \ldots z_k$

Proof Structure

Suppose for the sake of contradiction that there exists a DFA that accepts the language $L = \{0^n 1^n : n \geq 0\}$.

Somehow use these two facts:

- Fact 1: For any DFA and any infinite set S of strings over Σ^*, the DFA takes at least two strings $x, y \in S$ to the same state.
- Fact 2: If a DFA takes $x, y \in \Sigma^*$ to the same state, then for any $z \in \Sigma^*$, the DFA takes xz and yz to the same state.

Arrive at the contradiction that there exists a state that is accepting and rejecting. Thus there is no DFA that recognizes L, so L is irregular.

Proof

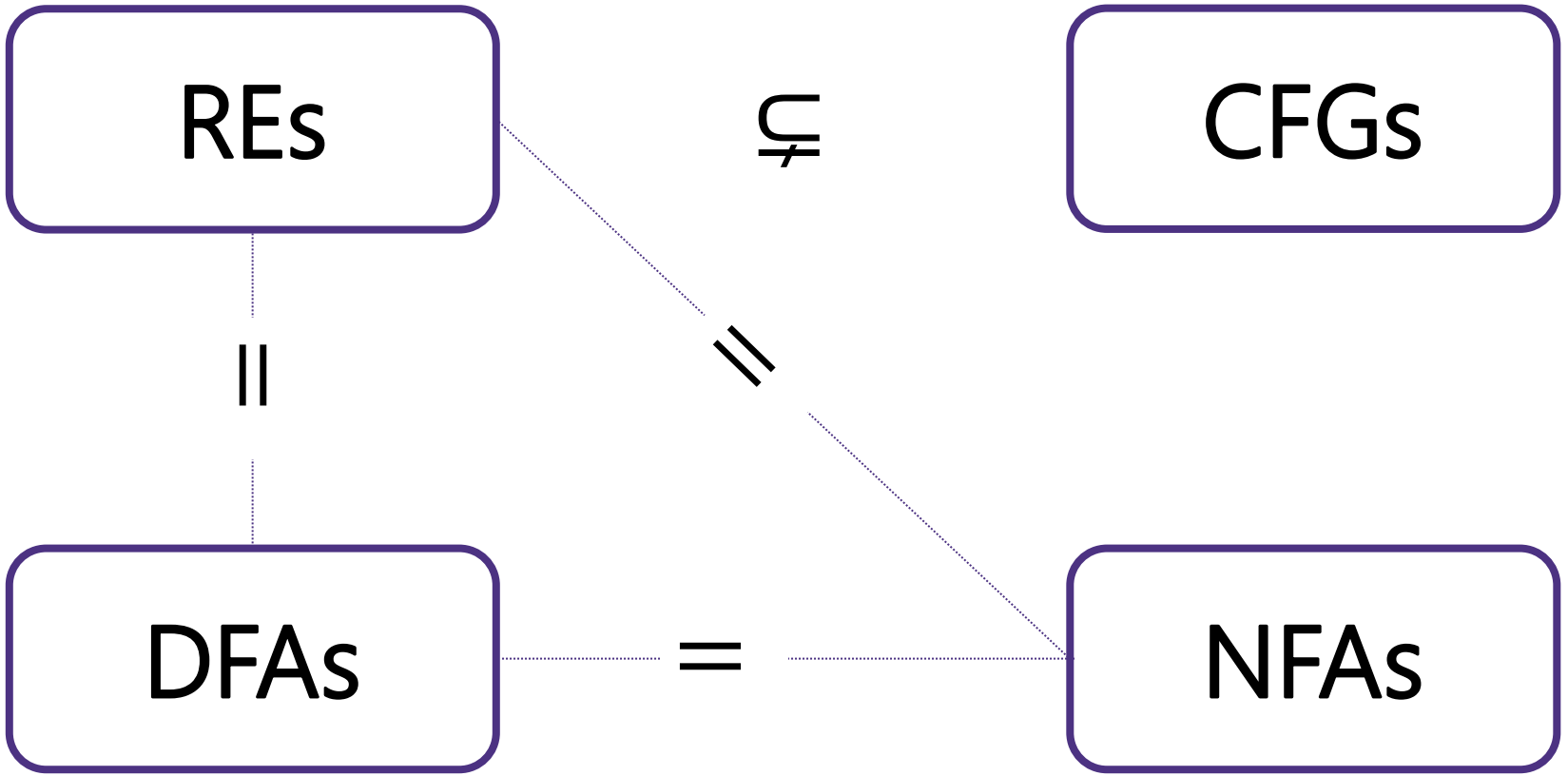
Fact 1: For any DFA and any infinite set S of strings over Σ^*, the DFA takes at least two strings $x, y \in S$ to the same state.

Fact 2: If a DFA takes $x, y \in \Sigma^*$ to the same state, then for any $z \in \Sigma^*$, the DFA takes xz and yz to the same state.

In conclusion…

- There are languages that are context-free but not regular
- This proof strategy can be used in general to show a language is irregular
  - Exercise: Show that the language of binary palindromes is irregular.

In conclusion…

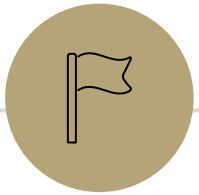


What do I need to know about this proof for the Final?

Theorem 4: There exists a language that is context-free but not regular.

That's it! We won't test you on *writing* proofs of irregularity.

The rest of the material from today will not be on the Final at all. It is a **very** important result in CS Theory.



The Halting Problem

Detecting Infinite Loops

- We've all written infinite loops in our code.
- Why isn't there a compiler error when your code has an infinite loop?
- Why isn't there a runtime error when your code has an infinite loop? It will just keep looping.
- Shouldn't your computer be checking if the code has a loop and reporting it as an error?

The Halting Problem

The Halting Problem

Given:

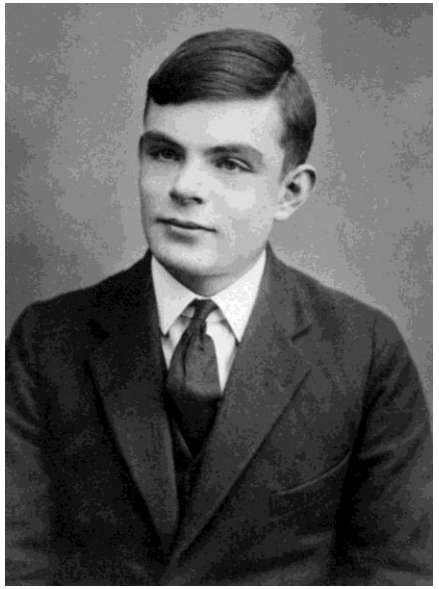
- CODE(P) - the code for a program P.
- An input string x.

Return:

- True if P halts on x.
- False if P encounters an infinite loop.

The Halting Problem

Theorem:
There does not exist a program that solves the Halting Problem.



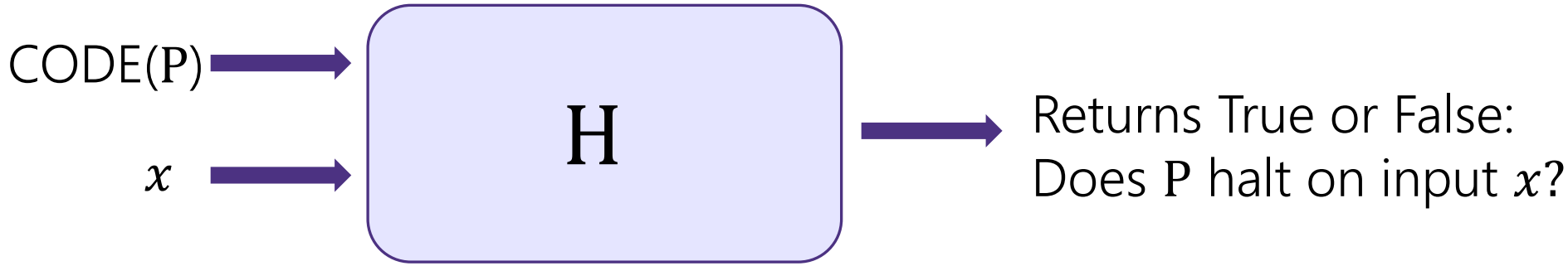
Alan Turing
(1912 – 1954)

First proved by Alan Turing in 1936

Proof by Contradiction

Suppose for the sake of contradiction that there is a Java function H that solves the Halting Problem.

You can think of H as a black box:



In Java, H would look like:

```
public boolean H(String P, String x) { … }
```

Proof by Contradiction

Suppose for the sake of contradiction that there is a Java function H that solves the Halting Problem.

Now consider this Java function D:

```
public static void D(String s) {
    if (H(s,s)) {
        while (true);  // don't halt
    } else {
        return;        // halt
    }
}

public static bool H(String s, String x) { ... }
```

Does $\mathrm{D}(\mathrm{CODE}(\mathrm{D}))$ halt? That is, what happens when we run D on input $\mathrm{CODE}(\mathrm{D})$?

Proof by Contradiction

```
public static void D(String s) {
    if (H(s,s)) {
        while (true);  // don't halt
    } else {
        return;        // halt
    }
}

public static bool H(String s, String x) { ... }
```

Does $\mathrm{D}(\mathrm{CODE}(\mathrm{D}))$ halt?
That is, what happens when we run D with input $\mathrm{CODE}(\mathrm{D})$?

Suppose D halts when run on input $\mathrm{CODE}(\mathrm{D})$. Then $\mathrm{H}(\mathrm{CODE}(\mathrm{D}), \mathrm{CODE}(\mathrm{D}))$ would return true. But then D would enter an infinite loop. So D does not halt. This is a contradiction.

Suppose D does not halt when run on input $\mathrm{CODE}(\mathrm{D})$. Then $\mathrm{H}(\mathrm{CODE}(\mathrm{D}), \mathrm{CODE}(\mathrm{D}))$ would return false. But then D would return. So D does halt. This is a contradiction.

So, such a function H cannot exist.

Final Proof

Suppose for the sake of contradiction that there is a Java function H that solves the Halting Problem. Then consider the Java function D:

```
public static void D(String s) {
    if (H(s,s)) {
        while (true);  // don't halt
    } else {
        return;        // halt
    }
}

public static bool H(String s, String x) { ... }
```

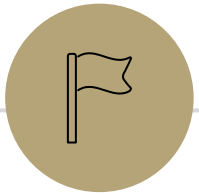
Suppose D halts when run on input $\mathrm{CODE(D)}$. Then $\mathrm{H}\big(\mathrm{CODE(D)}, \mathrm{CODE(D)}\big)$ would return true. But then D would enter an infinite loop. So D does not halt. This is a contradiction.

Suppose D does not halt when run on input $\mathrm{CODE(D)}$. Then $\mathrm{H}\big(\mathrm{CODE(D)}, \mathrm{CODE(D)}\big)$ would return false. But then D would return. So D does halt. This is a contradiction.

Thus, such a function H cannot exist.

What did we just prove?

- We showed that there is no computer program that can solve the Halting Problem for all inputs
  - There is nothing special about Java here (Church-Turing Thesis)
  - This means no compiler could check any program and determine if it loops infinitely
- That doesn't mean that there aren't algorithms that *often* get the answer right
  - For example, if there's no loops, no recursion, and no method calls, it definitely halts.



Concluding Remarks

Victory Lap!

- Propositional Logic
- Predicate Logic
- Proof Strategies
- Number Theory
- Set Theory
- Induction
- Models of Computation

Quantifiers are used in Data Structures to define big-O

Number Theory is essential in Cryptography

Induction is critical in many Algorithms proofs

REs, DFAs, CFGs used in Compilers and Programming Languages

What's next?

- CSE 312 (Foundations II) – Intro to Probability and Statistics
- CSE 332 – Data Structures and Parallelism
- CSE 421 – Algorithms
- CSE 431 – Complexity Theory

- CSE 373 – Non-Majors Data Structures
- CSE 417 – Non-Majors Algorithms

Feedback: Course Evaluations

Thank you for a great quarter! Before you go, we'd really appreciate your feedback in the course evaluations.