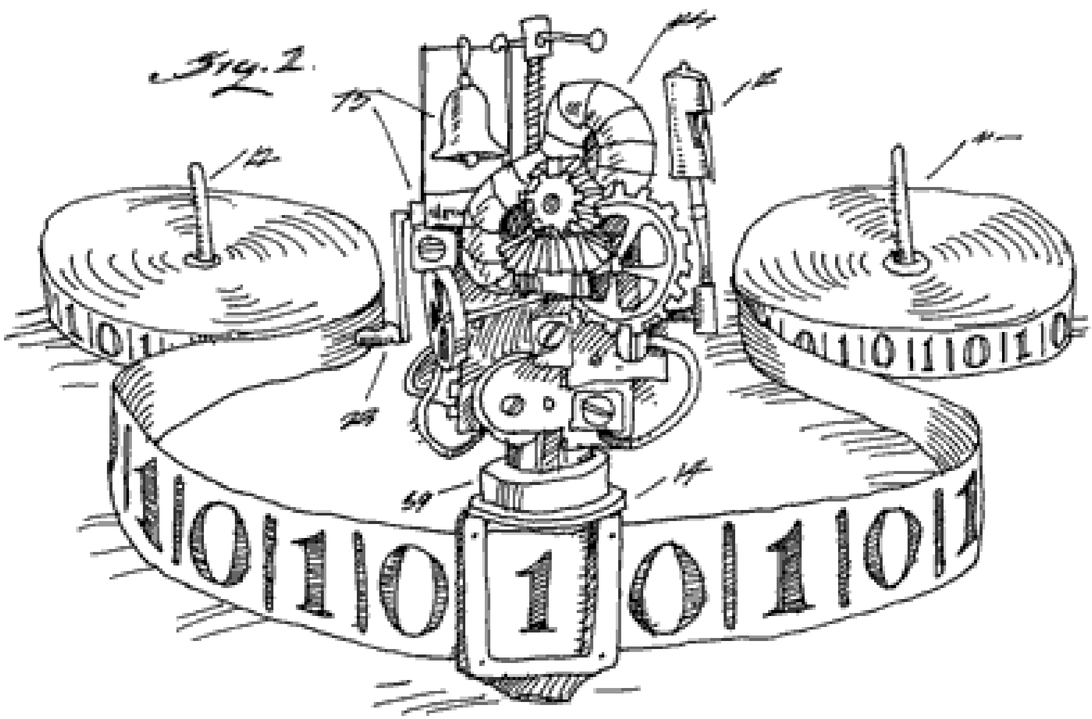
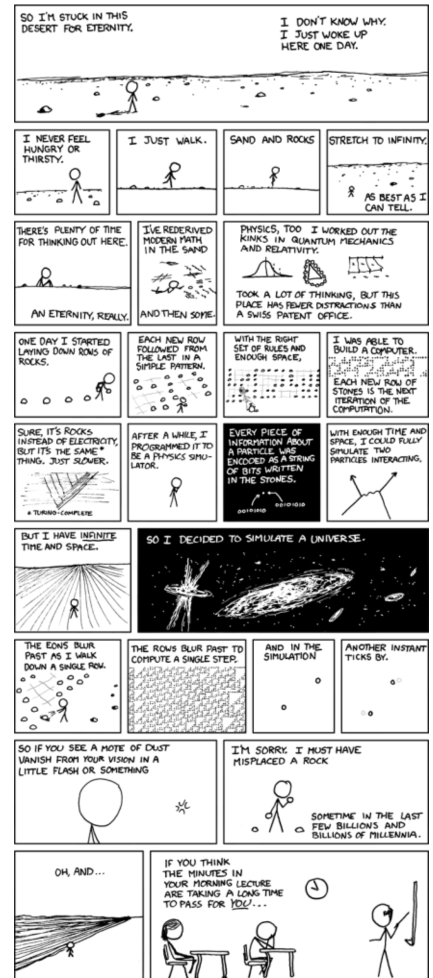
Fall 2015

Lecture 28: The halting problem and undecidability





We saw that the real numbers between 0 and 1 are **uncountable**.

Suppose, for the sake of contradiction, that there is a list of them:

		1	2	3	4						
r_1	0.	5^{1}	0	0	0						
r_2	0.	3	3^{5}	3	3						
r_3	0.	1	4	2^{5}	8	5	7	1	4	...	...
r_4	0.	1	4	1	5^{1}	9	2	6	5	...	...
						2^{5}	1	2	2	...	...
						0	0^{5}	0	0	...	...
						8	1	8^{5}	2	...	...

Flipping rule:
If digit is 5, make it 1.
If digit is not 5, make it 5.

For every $n \geq 1$:
$r_n \neq 0.\tilde{x}_{11}\tilde{x}_{22}\tilde{x}_{33}\tilde{x}_{44}\tilde{x}_{55}\cdots$
because the numbers differ on the nth digit!

So the list is incomplete, which is a contradiction.
Thus the real numbers between 0 and 1 are **uncountable**.

$(\mathbb{N} \setminus 0)$

the set of all functions $f : \cancel{\mathbb{N}} \to \{0, \dots, 9\}$ is uncountable

Supposed listing of all the functions:

	1	2	3	4	5	6	7	8	9	...
f_1	5	0	0	0	0	0	0	0	...	...
f_2	3	3	3	3	3	3	3	3	...	...
f_3	1	4	2	8	5	7	1	4	...	...
f_4	1	4	1	5	9	2	6	5	...	...
f_5	1	2	1	2	2	1	2	2	...	...
f_6	2	5	0	0	0	0	0	0	...	...
f_7	7	1	8	2	8	1	8	2	...	...
f_8	6	1	8	0	3	3	9	4	...	...
...	...	....	....	...	...	...	...	...	...	

the set of all functions $f : \mathbb{N} \to \{0, \dots, 9\}$ is uncountable

Supposed listing of all the functions:

Flipping rule:

If $f_n(n) = 5$, set $D(n) = 1$

If $f_n(n) \neq 5$, set $D(n) = 5$

	1	2	3	4						
f_1	5^1	0	0	0						
f_2	3	3^5	3	3						
f_3	1	4	2^5	8	5	7	1	4	...	...
f_4	1	4	1	5^1	9	2	6	5	...	...
f_5	1	2	1	2	2^5	1	2	2	...	...
f_6	2	5	0	0	0	0^5	0	0	...	...
f_7	7	1	8	2	8	1	8^5	2	...	...
f_8	6	1	8	0	3	3	9	4^5	...	...
...	...	....	....	...	...	...	...	...	...	

Σ^* countable

the set of all functions $f : \mathbb{N} \to \{0, \dots, 9\}$ is uncountable

Supposed listing of all the functions: $\mathbb{N}$, $P(\mathbb{N})$, $P(P(\mathbb{N}))$, ...

$\mathbb{R}$

	1	2	3	4						
f_1	5^{1}	0	0	0						
f_2	3	3^{5}	3	3						
f_3	1	4	2^{5}	8	5	7	1	4	...	...
f_4	1	4	1	5^{1}	9	2	6	5	...	...
f_5	1	2	1	2	2^{5}	1	2	2	...	...
f_6	2	5	0	0	0	0^{5}	0	0	...	...
f_7	7	1	8	2	8	1	8^{5}	2	...	...

Flipping rule:

If $f_n(n) = 5$, set $D(n) = 1$

If $f_n(n) \neq 5$, set $D(n) = 5$

For all n, we have $D(n) \neq f_n(n)$. Therefore $D \neq f_n$ for any n and the list is incomplete! $\Rightarrow$ $\{f \mid f : \mathbb{N} \to \{0,1, \dots, 9\}\}$ is **not** countable

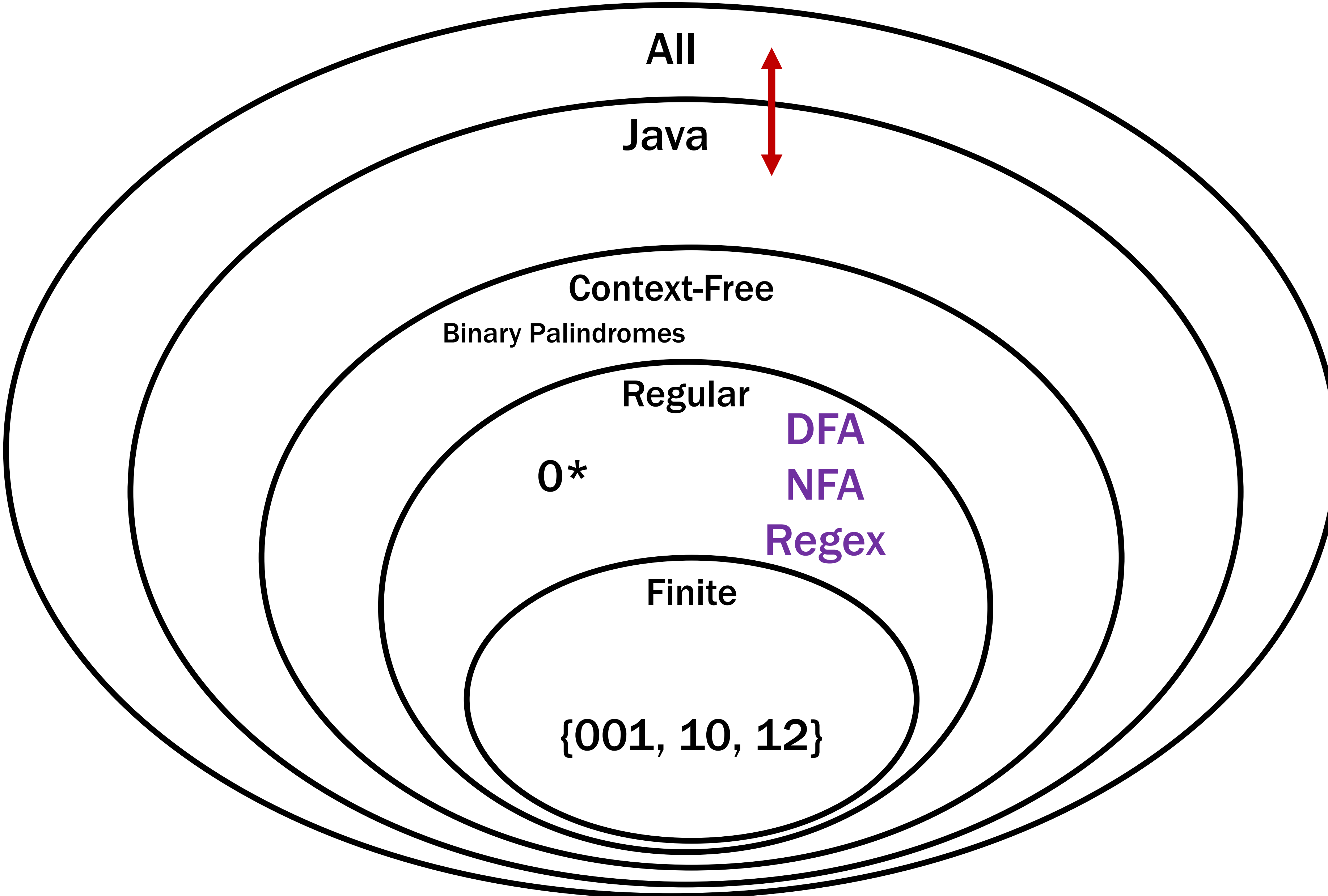
uncomputable functions

We have seen that:

- [last time] The set of all (Java) programs is countable
- The set of all functions $f : \mathbb{N} \to \{0, \dots, 9\}$ is not countable

So: There must be some function $f : \mathbb{N} \to \{0, \dots, 9\}$ that is not computable by any Java program!

recall our language picture



a cse ~~141~~ 142 assignment

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?

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 #2

int

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

n = 6

6, 3, 10, 5, 16,

8, 4, 2, 1

What does this program do?

… on n=5?

… on n=1000000000000000001?

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?

IMPOSSIBLE

some notation

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

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))?

"(((())).. ;AACPSSaaabceeggghiiiilnnnnnooprrrrrrrrrsssttttttuuwxxyy{}"

Given: - CODE(**P**) for any program **P**

- input **x**

Output: **true** if **P** halts on input **x**

false if **P** does not halt on input **x**

or CODE(P) is invalid

It turns out that it isn't possible to write a program that solves the Halting Problem.

proof by contradiction

- Suppose that **H** is a Java program that solves the Halting problem. Then we can write this program:

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

- Does **D**(CODE(**D**)) halt?

Does **D**(CODE(**D**)) halt?

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

H solves the halting problem implies that

H(CODE(**D**),x) is **true** iff **D**(x) halts, H(CODE(**D**),x) is **false** iff not

Does D(CODE(D)) halt?

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

H solves the halting problem implies that

H(CODE(D),x) is **true** iff D(x) halts, H(CODE(D),x) is **false** iff not

Suppose D(CODE(D)) **halts**.

Then, we must be in the **second** case of the if.

So, H(CODE(D), CODE(D)) is **false**

Which means D(CODE(D)) **doesn't halt**

Does D(CODE(D)) halt?

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

H solves the halting problem implies that
H(CODE(D),x) is **true** iff D(x) halts, H(CODE(D),x) is **false** iff not

Suppose D(CODE(D)) **halts**.
Then, we must be in the **second** case of the if.
So, H(CODE(D), CODE(D)) is **false**
Which means D(CODE(D)) **doesn't halt**

Suppose D(CODE(D)) **doesn't halt**.
Then, we must be in the **first** case of the if.
So, H(CODE(D), CODE(D)) is **true**.
Which means D(CODE(D)) **halts**.

Does D(CODE(D)) halt?

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

H solves the halting problem implies that

H(CODE(D),x) is **true** iff D(x) halts, H(CODE(D),x) is **false** iff not

Suppose D(CODE(D)) **halts**.

Then, we must be in the **second** case of the if.

So, H(CODE(D), CODE(D)) is **false**

Which means D(CODE(D)) **doesn't halt**

Suppose D(CODE(D)) **doesn't halt**.

Then, we must be in the **first** case of the if.

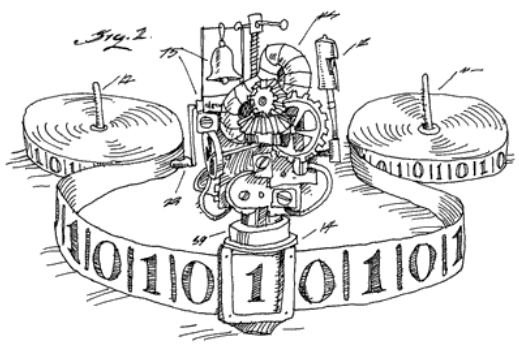
So, H(CODE(D), CODE(D)) is **true**.

Which means D(CODE(D)) **halts**.

Contradiction!

done

- We proved that there is no computer program that can solve the Halting Problem.
  - There was nothing special about Java*   [Church-Turing thesis]



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

connection to diagonalization

Some possible inputs **x**

programs **P**

	$<P_1>$	$<P_2>$	$<P_3>$	$<P_4>$	$<P_5>$	$<P_6>$	....					
P_1	0	1	1	0	1	1	1	0	0	0	1	...
P_2	1	1	0	1	0	1	1	0	1	1	1	...
P_3	1	0	1	0	0	0	0	0	0	0	1	...
P_4	0	1	1	0	1	0	1	1	0	1	0	...
P_5	0	1	1	1	1	1	1	0	0	0	1	...
P_6	1	1	0	0	0	1	1	0	1	1	1	...
P_7	1	0	1	1	0	0	0	0	0	0	1	...
P_8	0	1	1	1	1	0	1	1	0	1	0	...
P_9	. . . .											
.	. . . .											
.												

(P,x) entry is **1** if program **P** halts on input **x**
and **0** if it runs forever

connection to diagonalization

Some possible inputs **x**

programs **P**

	$\langle P_1 \rangle$	$\langle P_2 \rangle$	$\langle P_3 \rangle$	$\langle P_4 \rangle$	$\langle P_5 \rangle$	$\langle P_6 \rangle$	....					
P_1	0^{1}	1	1	0	1	1	1	0	0	0	1	...
P_2	1	1^{0}	0	1	0	1	1	0	1	1	1	...
P_3	1	0	1^{0}	0	0	0	0	0	0	0	1	...
P_4	0	1	1	0^{1}	1	0	1	1	0	1	0	...
P_5	0	1	1	1	1^{0}	1	1	0	0	0	1	...
P_6	1	1	0	0	0	1^{0}	1	0	1	1	1	...
P_7	1	0	1	1	0	0	0^{1}	0	0	0	1	...
P_8	0	1	1	1	1	0	1	1^{0}	0	1	0	...
P_9	. . . .											
.	. . . .											
.												

(P,x) entry is **1** if program **P** halts on input **x**
and **0** if it runs forever

- Can use undecidability of the halting problem to show that other problems are undecidable.

- For instance:

  $\text{EQUIV}(P, Q)$: **True** if $P(x) = Q(x)$ for every input x
  False otherwise

$H(P, x)$:

?

Rice's theorem

Not *every* problem on programs is undecidable!

Which of these is decidable?

- Input CODE(P) and x
  Output: true if P prints "ERROR" on input x
  after less than 100 steps
  false otherwise
- Input CODE(P) and x
  Output: true if P prints "ERROR" on input x
  after more than 100 steps
  false otherwise

Compilers Suck Theorem (informal):

Any "non-trivial" property the **input-output behavior** of Java programs is undecidable.