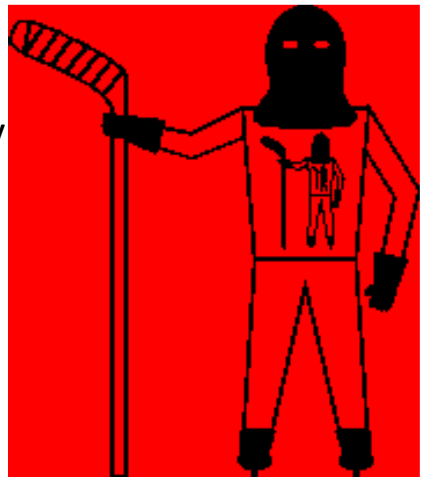
Class 37: Uncomputability



Halting Problems Hockey Team

David Evans
University of Virginia cs1120

Project Updates

We will provide a server to host your web project externally

<your site>.cs.virginia.edu

e.g.,

overheardit.cs.virginia.edu

If you want a site like this, send me email (one per team!) with your preferred name as soon as possible (but definitely not later than Monday, Nov 30).

Impossibility Results

Last Friday

Mathematics (Declarative Knowledge)

Gödel: Any powerful axiomatic system cannot be both *complete* and *consistent*

If it is possible to express "This statement has no proof." in the system, it must be incomplete or inconsistent.

Today

Computer Science (Imperative Knowledge)

Are there (well-defined) problems that cannot be solved by *any* algorithm?

Alan Turing (and Alonzo Church): Yes!

Computability

A problem is *computable* if there is an *algorithm* that solves it.

What is an *algorithm*? A *procedure* that **always finishes.**

What is a *procedure*? A precise description of a series of **steps that can be followed mechanically*** (without any thought).

*A formal definition of computable requires a more formal definition of a *procedure*.

What does it mean to have an algorithm that *solves* a problem?

We have a procedure that always finished, and always provides a correct output for any problem instance.

Computability

Is there an *algorithm* that solves a problem?

Computable (decidable) problems can be solved by some algorithm.

Make a photomosaic, sorting, drug discovery, winning chess (it doesn't mean we know the algorithm, but there is one)

Noncomputable (undecidable) problems cannot be solved by *any* algorithm.

There might be a procedure (but it doesn't finish for some inputs).

Are there any **noncomputable** problems?

Alan Turing (1912-1954)

Published *On Computable Numbers …* (1936)

Introduced the *Halting Problem*

5 years after Gödel's proof!

Formal model of computation

(now known as "Turing Machine")

Codebreaker at Bletchley Park

Led efforts to break Enigma Cipher

After the war: convicted of "gross indecency" (homosexuality, then a crime in Britain), forced to undergo hormone treatments, committed suicide eating cyanide apple

Prime Minister's Apology

It is no exaggeration to say that, without his outstanding contribution, the history of World War Two could well have been very different. He truly was one of those individuals we can point to whose unique contribution helped to turn the tide of war. The debt of gratitude he is owed makes it all the more horrifying, therefore, that he was treated so inhumanely....

So on behalf of the British government, and all those who live freely thanks to Alan's work I am very proud to say: we're sorry, you deserved so much better.

Gordon Brown, 10 September 2009

The (Pythonized) Halting Problem

Input: a string representing a Python program.

Output: If evaluating the input program would ever finish, output **true**. Otherwise, output **false**.

Suppose **halts** solves Halting Problem

```
def halts(code):
    ... ? ...
```

Input: a string representing a Python program.

Output: If evaluating the input program would ever finish, output **true**. Otherwise, output **false**.

```
>>> halts('3 + 3')
True
>>> halts("""
    i = 0
    while i < 100:
        i = i * 2""")
False
```

Halting Examples

```
>>> halts("""
def fact(n):
  if n = 1: return 1
  else: return n * fact(n - 1)
fact(7)
""")
True
```

```
>>> halts("""
def fact(n):
  if n = 1: return 1
  else: return n * fact(n - 1)
fact(0)
""")
False
```

```
halts("""
  def fibo(n):
    if n == 1 or n == 2: return 1
    else: return fibo(n 1) + fibo(n 2)
  fibo(60)
""")
```

Can we define **halts**?

Attempt #1:

```
def halts(code):
  eval(code)
  return True
```

Attempt #2:

```
def halts(code):
  try:
    with Timer(100):
      eval(code)
      return True
  except Timer:
    return False
```

These two approaches fail, but not a proof it cannot be done!

Impossibility of Halts

Recall how Gödel showed incompleteness of PM:

- Find a statement that leads to a **contradiction**
- Gödel's statement: "This statement has no proof."

Is there an input to halts that leads to a contradiction?

Informal Proof

```
def paradox():
  if halts('paradox()'):
    while True:
      pass
```

Does paradox() halt?

Yes?: If **paradox** halts, the if test is true and it evaluates to an infinite loop: it doesn't halt!

No?: If **paradox** doesn't halt, the if test is false and it finishes. It halts!

Proof by Contradiction

1. Show ***X*** is nonsensical.
2. Show that if you have ***A*** you can make ***X***.
3. Therefore, ***A*** must not exist.

Turing: Noncomputability

X = paradox procedure

A = algorithm that solves Halting Problem

Gödel: Incompleteness

X = "This statement has no proof."

A = a complete and consistent axiomatic system

Halting Problem is Noncomputable

```
def paradox():
  if halts('paradox()'):
    while True:
      pass
```

1. **paradox** leads to a contradiction.
2. If we have **halts**, an algorithm that solves the Halting Problem, we can define **paradox**.
3. Therefore, **halts** does not exist.

Are there any other noncomputable problems?

Evaluates-to-3 Problem

Input: A string, *s*, representing a Python program.

Output: True if *s* evaluates to 3; otherwise, **False**.

Is "Evaluates-to-3" computable?

Proof by Contradiction

1. Show ***X*** is nonsensical.
2. Show that if you have ***A*** you can make ***X***.
3. Therefore, ***A*** must not exist.

X = halts algorithm

A = Evaluates-to-3 algorithm

Undecidability Proof

Suppose we could define evaluates_to_3(). that decides it. Could we define halts()?

```
def halts(s):
  return evaluates_to_3(s + """
      return 3
      """)
```

The only way the program passed to evaluates_to_3 could not evaluate to 3, is if *s* doesn't halt. (Note: assumes evaluating *s* cannot produce an error.)

How convincing is our Halting Problem proof?

```
def paradox():
  if halts('paradox()'):
    while True:
      pass
```

1. **paradox** leads to a contradiction.
2. If we have **halts**, an algorithm that solves the Halting Problem, we can define **paradox**.
3. Therefore, **halts** does not exist.

This "proof" assumes Python exists and is means exactly what it should! Python is too complex to believe this: we need a simpler and more precise model of computation. **Monday's class**

Charge

Enjoy your Thanksgiving!

Team meetings today in **Olsson 226D:**
11:00 Colin, Taylor, Will
11:20 Kiran, Muzzammil, Omer, Qihan
11:40 Kevin, Rachel, Rose

Conference room at back corner of Olsson Hall – go all the way to the end of the hallway and turn right, room is in back corner.