CSE 311 Foundations of Computing I

Lecture 30

Computability: Other Undecidable Problems

Autumn 2012

Announcements

- Reading
  - 7th edition: p. 201 and 13.5
  - 6th edition: p. 177 and 12.5
  - 5th edition: p. 222 and 11.5
- Topic list and sample final exam problems have been posted
- Comprehensive final, roughly 67% of material post midterm
- Review session, Saturday, December 8, 10 am – noon, EEB 125
- Final exam, Monday, December 10
  - 2:30-4:20 pm or 4:30-6:20 pm, Kane 220.

Last lecture highlights

Turing machine = Finite control + Recording Medium + Focus of attention

Finite Control:
program **P**

	_	0	1
s_1	$(1,s_3)$	$(1,s_2)$	$(0,s_2)$
s_2	(H,s_3)	(R,s_1)	(R,s_1)
s_3	(H,s_3)	(R,s_3)	(R,s_3)

Recording Medium

input **x**

_	_	1	1	0	1	1	_	_

_	_	0	1	0	1	1	_	_

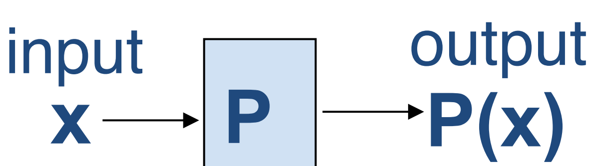
……

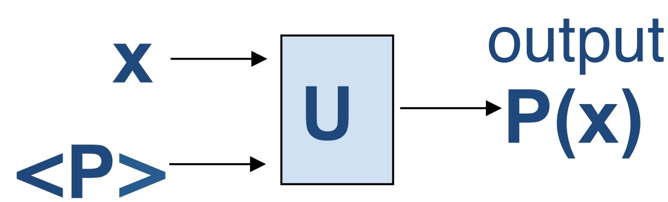
output

_	_	0	0	1	0	0	1	_

Last lecture highlights

- The Universal Turing Machine **U**
  - Takes as input: **(<P>,x)** where **<P>** is the code of a program and **x** is an input string
  - Simulates **P** on input **x**
- Same as a Program Interpreter





Last lecture highlights

Program **P**

	_	0	1
s_1	$(1,s_3)$	$(1,s_2)$	$(0,s_2)$
s_2	(H,s_3)	(R,s_1)	(R,s_1)
s_3	(H,s_3)	(R,s_3)	(R,s_3)

input **x**

_	_	1	1	0	1	1	_	_

Universal TM **U**

	_	0	1	(	)	s	2	3	...
s_1									
s_2									
	...								

Program code **<P>** input **x**

(	1	,	s	3	)	(	1	. . . .	1	1	0	1	1	_	_

output

(	1	,	s	3	)	(	1	. . . .	0	0	1	0	0	1	_

Programs about Program Properties

- The Universal TM takes a program code **<P>** as input, and an input **x**, and interprets **P** on **x**
  - Step by step by step by step…
- Can we write a TM that takes a program code **<P>** as input and checks some property of the program?
  - Does **P** ever return the output “ERROR”?
  - Does **P** always return the output “ERROR”?
  - Does **P** halt on input **x**?

Halting Problem

- Given: the code of a program **P** and an input **x** for **P**, i.e. given **(<P>,x)**
- Output: **1** if **P** halts on input **x**
  0 if **P** does not halt on input **x**

Theorem (Turing): There is no program that solves the halting problem
"The halting problem is undecidable"

Proof by contradiction

- Suppose that **H** is a Turing machine that solves the Halting problem

Function **D(x)**:
- if **H(x,x)=1** then
  - **while** (true); /* loop forever */
- else
  - **no-op**; /* do nothing and halt */
- endif

- What does **D** do on input **<D>**?
  - Does it halt?

Does D halt on input <D>?

Function D(x):

- if H(x,x)=1 then
  - **while** (true); /* loop forever */
- else
  - **no-op**; /* do nothing and halt */
- endif

D halts on input <D>

⇔ H outputs 1 on input (<D>,<D>)

[since H solves the halting problem and so H(<D>,x) outputs 1 iff D halts on input x]

⇔ D runs forever on input <D>

[since D goes into an infinite loop on x iff H(x,x)=1]

That's it!

- We proved that there is no computer 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

SCOOPING THE LOOP SNOOPER

A proof that the Halting Problem is undecidable

by Geoffrey K. Pullum (U. Edinburgh)

No general procedure for bug checks succeeds.
Now, I won't just assert that, I'll show where it leads:
I will prove that although you might work till you drop,
you cannot tell if computation will stop.

For imagine we have a procedure called *P*
that for specified input permits you to see
whether specified source code, with all of its faults,
defines a routine that eventually halts.

You feed in your program, with suitable data,
and *P* gets to work, and a little while later
(in finite compute time) correctly infers
whether infinite looping behavior occurs...

SCOOPING THE LOOP SNOOPER

...

Here's the trick that I'll use -- and it's simple to do.
I'll define a procedure, which I will call *Q*,
that will use *P*'s predictions of halting success
to stir up a terrible logical mess.

...

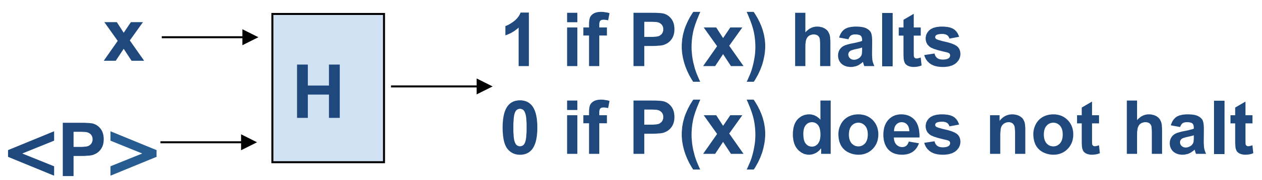
And this program called *Q* wouldn't stay on the shelf;
I would ask it to forecast its run on *itself*.
When it reads its own source code, just what will it do?
What's the looping behavior of *Q* run on *Q*?

...

Full poem at:

http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html

Halting Problem



The "Always Halting" problem

- **Given:** **<Q>**, the code of a program **Q**
- **Output:** **1** if **Q** halts on every input
  0 if not.

Claim: the "always halts" problem is undecidable

Proof idea:

- Show we could solve the Halting Problem **if** we had a solution for the "always halts" problem.
- No program solving for Halting Problem exists $\Rightarrow$ no program solving the "always halts" problem exists

The "Always Halting" problem

x → **H** → 1 if P(x) halts
<P> → 0 if P(x) does not halt

Suppose we had a TM **A** for the Always Halting problem

x →
<P> →

Program Q(x')
… ignore **x'**
… run **P** on **x**

<Q> → **A** → 1 if P(x) halts
0 if P(x) does not halt

The “Always ERROR” problem

- **Given:** **<R>**, the code of a program **R**
- **Output:** **1** if **R** always prints ERROR
  0 if **R** does not always print ERROR

The “Always ERROR” problem

<Q> → **A** → **1 if Q always halts**
0 if Q does not always halt

Suppose we had a TM **E** for the ERROR problem

<Q> →

Program R(x)
Run **Q** on **x**
Print “ERROR”
Halt

<R> → **E** → **1 if Q always halts**
0 if Q does not always halt

Pitfalls

- Not *every* problem on programs is undecidable! Which of these is decidable?
- Input <P> and x
  Output: 1 if P prints “ERROR” on x
  after less than 100 steps
  0 otherwise
- Input <P> and x
  Output: 1 if P prints “ERROR” on x
  after more than 100 steps
  0 otherwise