

Last lecture highlights
Turing machine definition

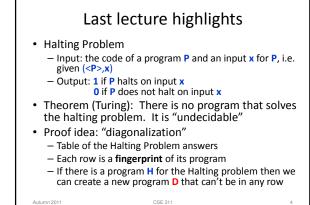
Intuitive justification, Church-Turing Thesis

Programs ≡ Turing machines

Distinction between the executing program P and its code <P>

Program Interpreter U (Universal TM)

Takes as input: (<P>,x) where <P> is the code of a program and x is an input string
Simulates P on input x

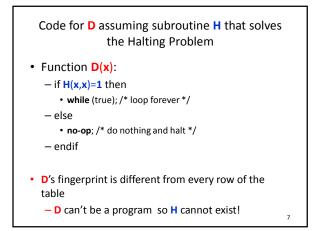


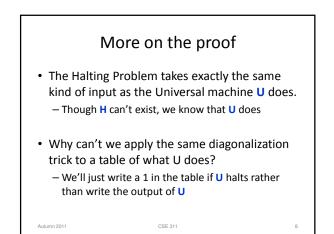
				ir	put	x					
	λ	0	1	00	01	10	11	000	001	010	011
λ	0	1	1	0	1	1	1	0	0	0	1
0	1	1	0	1	0	1	1	0	1	1	1
∧ 1	1	0	1	0	0	0	0	0	0	0	1
ê 00	0	1	1	0	1	0	1	1	0	1	0
	0	1	1	1	1	1	1	0	0	0	1
01 og	1	1	0	0	0	1	1	0	1	1	1
	1	0	1	1	0	0	0	0	0	0	1
<u>6</u> 000	0	1	1	1	1	0	1	1	0	1	0
000 001	•					•					
a.	•					•					
<ul> <li>(<p>,x) entry is 1 if program P halts on input x and 0 if it runs forever</p></li> </ul>											

CSE 311

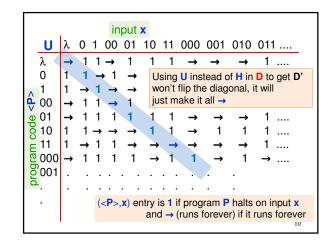
Autumn 2011

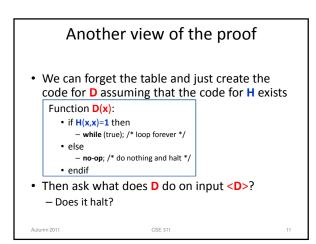
		input <b>x</b>						Flipped Diagonal					
	λ	0	1	00	01	10	11	000	001	010	011		
λ	1	1	1	0	1	1	1	0	0	0	1		
0	1	0	0	1	0	1	1	0	1	1	1		
1	1	0	0	0	0	0	0	0	0	0	1		
00	0	1	1	1	1	0	1	1	0	1	0		
	0	1	1	1	0	1	1	0	0	0	1		
B 01 0 10	1	1	0	0	0	0	1	0	1	1	1		
	1	0	1	1	0	0	1	0	0	0	1		
000	0	1	1	1	1	0	1	0	0	1	0		
11 000 001													
ā.													
•			V F	Vant	t to c erties	reate s are	e a r give	ew pr	ogram the <b>fli</b> j	n whos pped o	e halting diagonal		

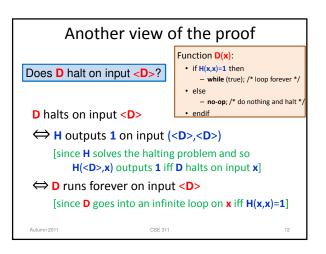


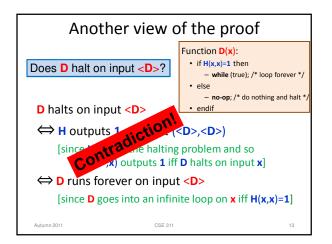


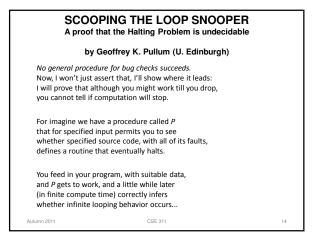
input <b>x</b>												
U	λ	0	1	00	01	10	11	000	001	010	011	
λ	<b>→</b>	1	1	→	1	1	1	<b>→</b>	→	<b>→</b>	1	
0	1	1	-	<b>•</b> 1	→	1	1	→	1	1	1	
∧ 1	1	→	1	→	→	→	→	<ul> <li>→</li> </ul>	$\rightarrow$	$\rightarrow$	1	
<b>0</b> 0	→	1	1	→	1	→	1	1	$\rightarrow$	1	$\rightarrow$	
<u>e</u> 01	→	1	1	1	1	1	1	→	→	$\rightarrow$	1	
ğ 10	1	1	→	→	→	1	1	→	1	1	1	
E 11	1	<b>→</b>	1	1	<b>→</b>	→	→	<b>→</b>	→	<b>→</b>	1	
<u>e</u> 000	<b>→</b>	1	1	1	1	$\rightarrow$	1	1	$\rightarrow$	1	→	
01 01 01 01 000 000 001	•	•	•	·	• •	•	• •	•	•	•		
<u> </u>	•	•	•	·				•	•	·		
			(•	< <b>P</b> >	, <b>x</b> ) ei					halts er) if it		nput <b>x</b> forever

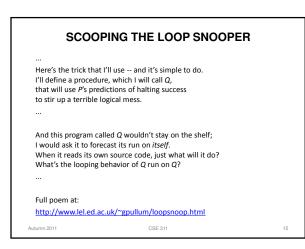










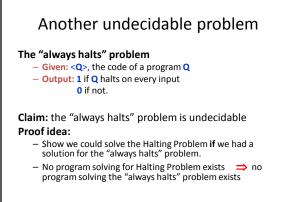


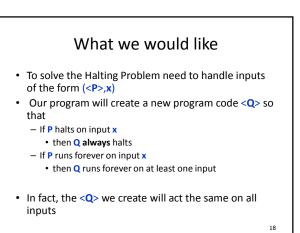
# Using undecidability of the halting problem

- We have one problem that we know is impossible to solve

   Halting problem
- Showing this took serious effort
- We'd like to use this fact to derive that other problems are impossible to solve
  - don't want to go back to square one to do it

16

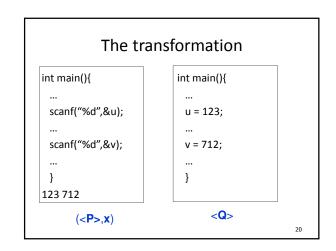




17

# Creating <Q> from (<P>,x)

- Given (<P>,x) modify code of P to:
   Replace all input statements of P that read input x, by assignment statements that 'hard-code' x in
- This creates a new program text <Q>
- It would be easy to write a program T that changes (<P>,x) to <Q>



Program to solve Halting Problem if "always halts" were decidable

- Suppose "always halts" were solvable by program A
- On input (<P>,x)

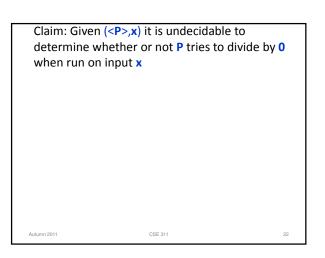
Autumn 2011

- execute the program T to transform (<P>,x) into <Q> as on last slide
- call A with <Q> (the output of T) as its input and use A's output as the answer.
- This would do the job of **H** which we know can't exist so **A** can't exist

21

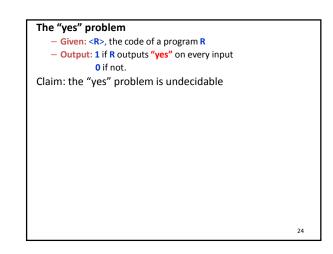
23

19



Claim: Given (<**P**>,**x**) it is undecidable to determine whether or not **P** accesses an array out of bounds when run on input **x** 

CSE 311

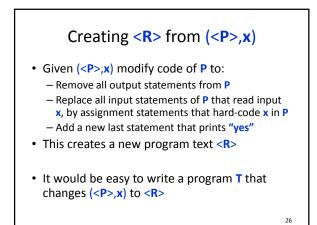


# Same kind of idea as "always halts" To solve the Halting Problem need to be able to handle inputs of the form (<P>,x) We'll create a new program code <R> so that If P halts on input x then R always outputs "yes" If P runs forever on input x

then R does something else on at least one input.

25

27



Program to solve Halting Problem if the "yes" problem were decidable

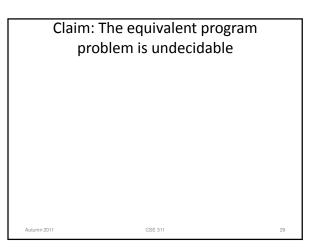
- Suppose the "yes" problem were solvable by program Y
- On input (<P>,x)
  - execute the code to transform (<P>,x) into <R> as on last slide
  - call Y with <R> (the output of T) as its input and use Y's output as the answer.

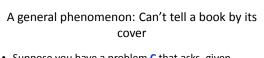
# Input: the codes of two programs, <P> and <Q> Output: 1 if P produces the same output as Q does on every input 0 otherwise

Equivalent program problem

Claim: The equivalent program problem is undecidable

28





 Suppose you have a problem C that asks, given program code <P>, to determine some property of the input-output behavior of P, answering 1 if P has the property and 0 if P doesn't have the property.

**Rice's Theorem:** If **C**'s answer isn't always the same then there is no program deciding **C** 

30

## Even harder problems

- Recall that with the halting problem, we could always get at least one of the two answers correct
  - if it halted we could always answer 1 (and this would cover precisely all 1's we need to do) but we can't be sure about answering 0
- There are natural problems where you can't even do that!
  - The equivalent program problem is an example of this kind of even harder problem.

31

### **Quick lessons**

- Don't rely on the idea of improved compilers and programming languages to eliminate major programming errors
  - truly safe languages can't possibly do general computation
- Document your code!!!!
  - there is no way you can expect someone else to figure out what your program does with just your code ....since....in general it is provably impossible to do this!

32