The Limits of Computation 7B

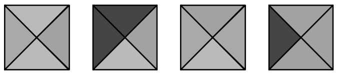
Noncomputability

It gets worse...

- Tractable Problems
  - Problems that have reasonable, polynomial-time solutions
- Intractable Problems
  - Problems that have no reasonable, polynomial-time solutions
- Noncomputable Problems
  - Problems that have no algorithms at all to solve them

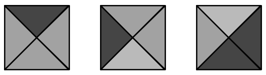
Noncomputability and Undecidability

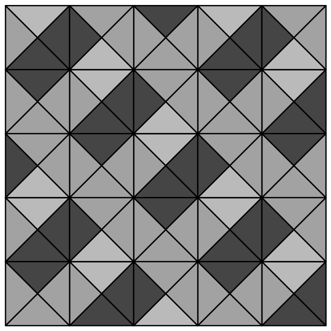
- An algorithmic problem that has no algorithm is called **noncomputable**.
- If the noncomputable algorithm requires a yes/no answer, the problem is called **undecidable.**



- Example:
  - Given <u>any</u> set of <u>any</u> number of different tile designs (examples shown above), with an infinite number of each type of tile, can we tile <u>any</u> area with these tiles so that like colored edges touch?
  - This problem is undecidable!

Tiling Problem

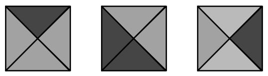




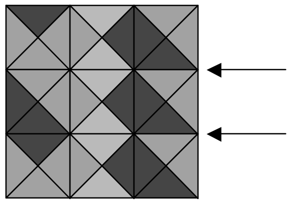
YES

Note the periodicity in the tiling.

Tiling Problem



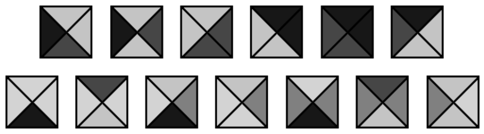
NO



For this 3 X 3 room, if we try all 3^9 tiling configurations, no tiling works.

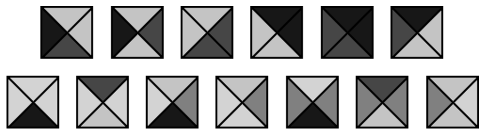
Tiling Problem

- Possible algorithm:
  - If we find a repeating pattern, report YES.
  - If we find a floor we cannot tile, report NO.
- BUT: there are some tilings which have no repeating pattern!



Tiling Problem

- The only way to know if this set of tiles can tile every finite-sized floor is to evaluate every possible floor.
- BUT: there are an infinite number of finite-sized floors!
  - So we could never answer YES in this case.
- This problem is undecidable.



Word Correspondence

Given a finite alphabet (e.g. {*a*, *b*})

Consider two groups (call them X and Y) of N words made from the alphabet.

	1	2	3	4	5
X	*abb*	*a*	*bab*	*baba*	*aba*
Y	*bbab*	*aa*	*ab*	*aa*	*a*

Is there a way to concatenate words in the same sequence from X and Y to form the same word?

Word Correspondence

Sequence: 2, 1, 1, 4, 1, 5

X: *a* + *abb* + *abb* + *baba* + *abb* + *aba*

Y: *aa* + *bbab* + *bbab* + *aa* + *bbab* + *a*

Both sequences yield *aabbabbbabaabbaba*

Answer: yes

	1	2	3	4	5
X	*abb*	*a*	*bab*	*baba*	*aba*
Y	*bbab*	*aa*	*ab*	*aa*	*a*

Word Correspondence

Consider the following word correspondence problem.
Is there a solution this time?

	1	2	3	4	5
X	*bb*	*a*	*bab*	*baba*	*aba*
Y	*bab*	*aa*	*ab*	*aa*	*a*

Word Correspondence

Word Correspondence

- The word correspondence problem is undecidable.
- For <u>any</u> set of <u>n</u> pairs of words for a finite alphabet of <u>any</u> size, there is no general algorithm that can determine if there is a valid correspondence or not.
  - If there were an algorithm, how would it know when to output an answer of NO?

Another Undecidable Problem: The Barber Paradox

Suppose there is a town with one male barber; and that every man in the town keeps himself clean-shaven: some shave themselves and some are shaved by the barber. Only the barber can shave another man. The barber shaves all and only those men who do not shave themselves.

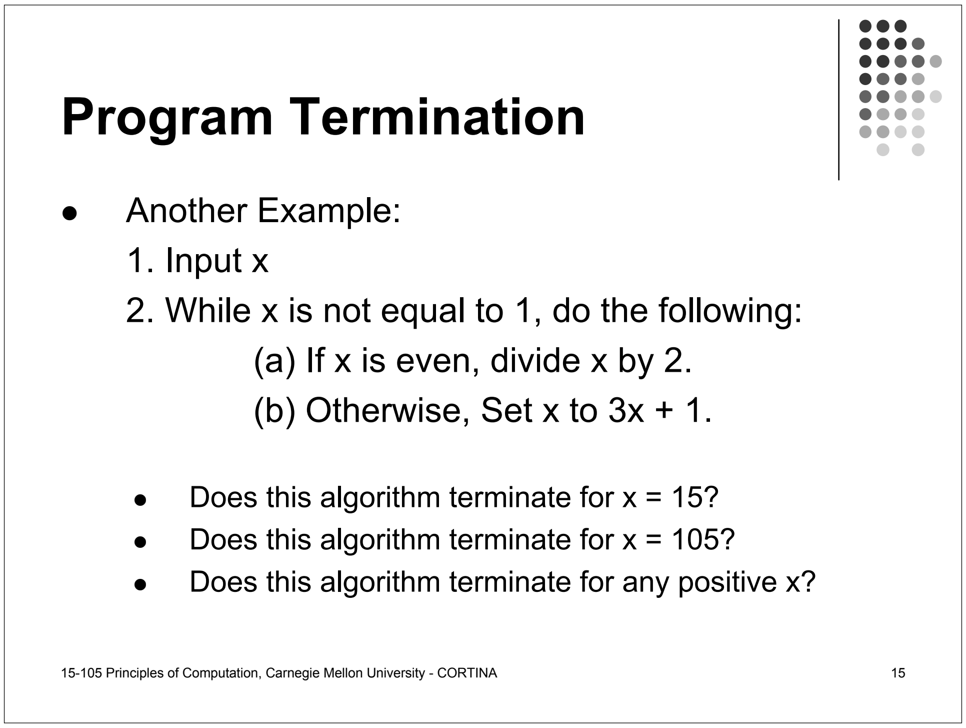
Does the barber shave himself?

Program Termination

- Can we determine if a program will terminate given a valid input?
- Example:

```
1. Input x
2. While x is not equal to 1, do the following:
      (a) Subtract 2 from x.
```

  - Does this algorithm terminate when x = 15105?
  - Does this algorithm terminate when x = 2008?

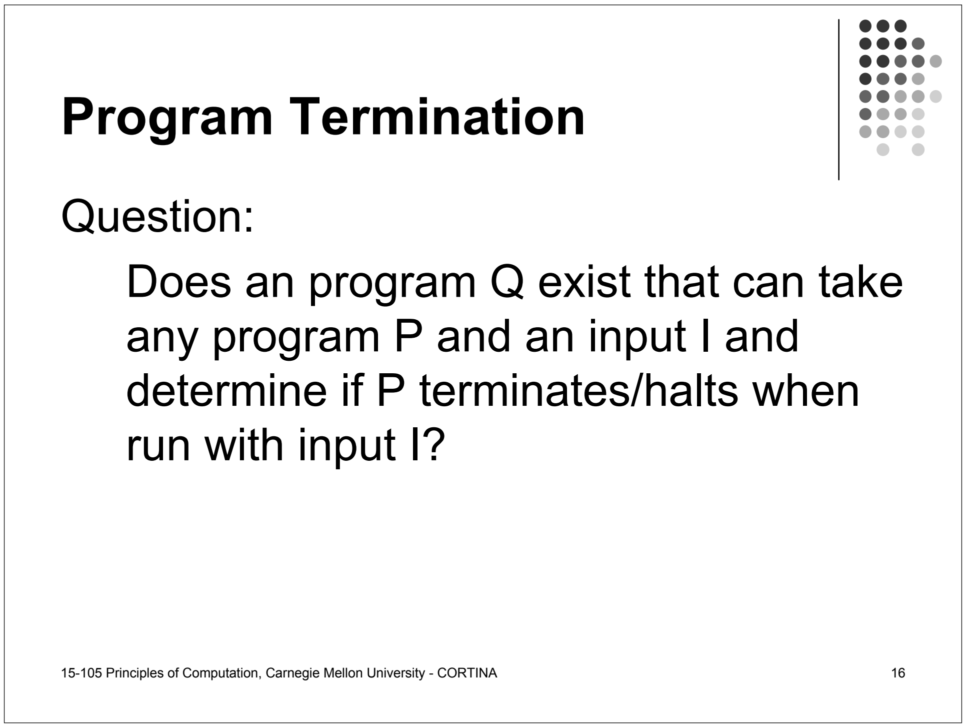


Program Termination

- Another Example:

```
1. Input x
2. While x is not equal to 1, do the following:
        (a) If x is even, divide x by 2.
        (b) Otherwise, Set x to 3x + 1.
```

  - Does this algorithm terminate for x = 15?
  - Does this algorithm terminate for x = 105?
  - Does this algorithm terminate for any positive x?



Program Termination

Question:

Does an program Q exist that can take any program P and an input I and determine if P terminates/halts when run with input I?

What is a program anyway?

- A computer program is simply a long string of characters.
- Example:

```
Program P(I);
output I*2;
end.
```

Program P requires one input I.

Program P is stored in the computer as one long string.

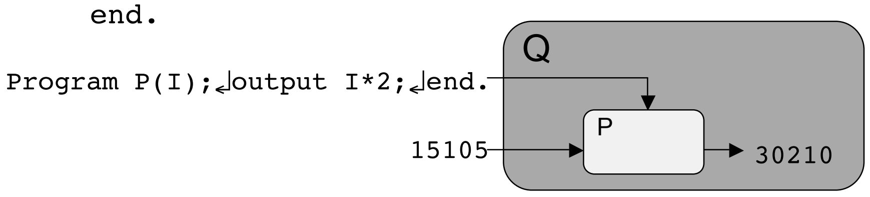
```
Program P(I);↲output I*2;↲end.
```

A program that runs a program

- Let Q be a program that takes another program P (as a string) and an input I, and Q runs P using input I.

```
Program Q(P,I);
run P(I);
end.
```

Q runs program P, feeding it input I



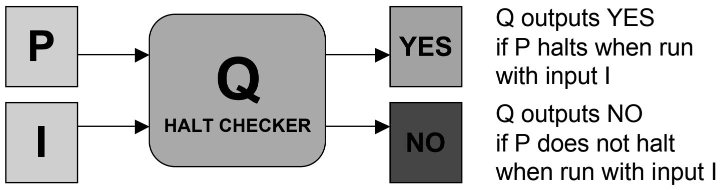
The Halting Problem

- Can we write a general program Q that takes as its input any program P and an input I and determines if program P will terminate (halt) when run with input I?
  - It will answer YES if P terminates successfully on input I.
  - It will answer NO if P never terminates on input I.
- This computational problem is undecidable!
  - No such general program Q can exist!

The Halting Problem (1)

Proof by Contradiction

- Assume a program Q exists that requires a program P and an input I.
  - Q determines if program P will halt when P is executed using input I.

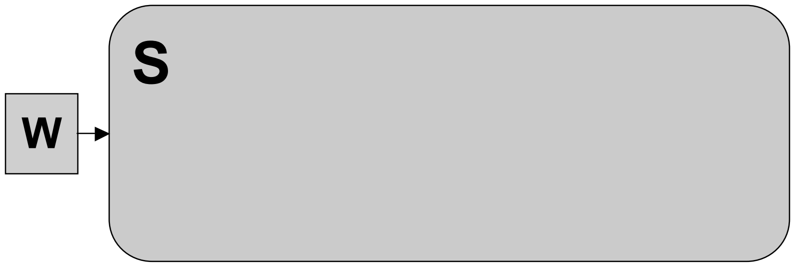

P
I
Q
HALT CHECKER
YES
NO
Q outputs YES if P halts when run with input I
Q outputs NO if P does not halt when run with input I

  - We will show that Q can never exist through contradiction.

The Halting Problem (2)

Proof by Contradiction

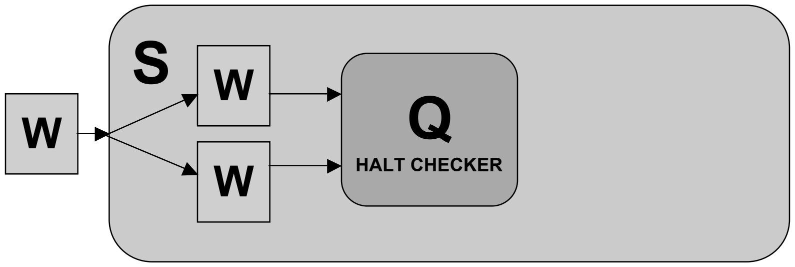
- Define a new program S that takes a program W as its input.



The Halting Problem (3)

Proof by Contradiction

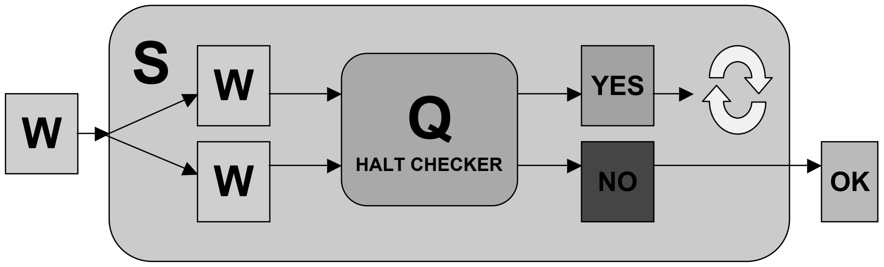
- S feeds W as the inputs for Q as the program and the program's input.



The Halting Problem (4)

Proof by Contradiction

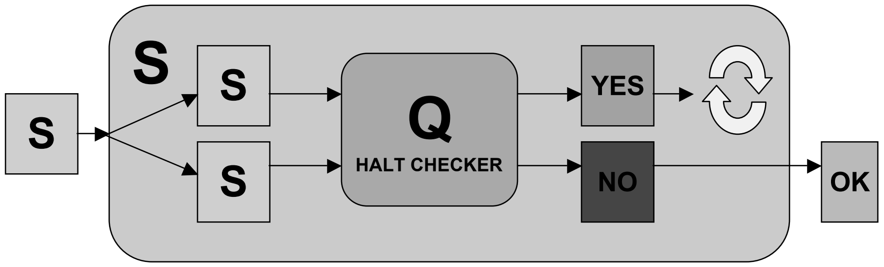
- Then S looks at the answer Q gives.
  - If Q answers YES, S purposely forces itself into an infinite loop.
  - If Q answers NO, S halts with an output of OK.



The Halting Problem (5)

Proof by Contradiction

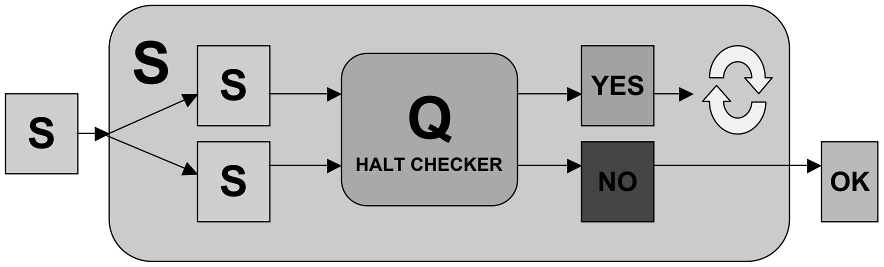
- Since S requires as its input a program, and S is a program, what happens if the input to S is itself?



The Halting Problem (6)

Proof by Contradiction

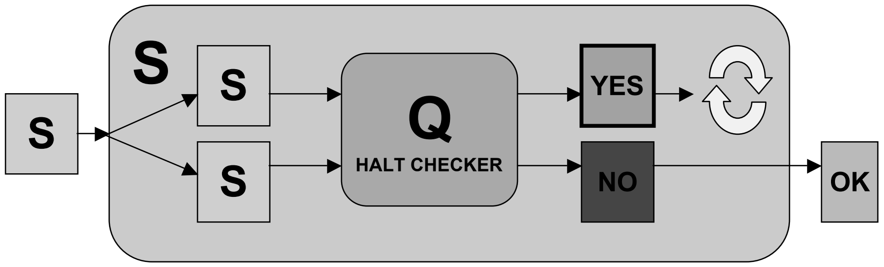
- Put another way, S asks Q:
  "What do I do if I execute using myself as input?"



The Halting Problem (7)

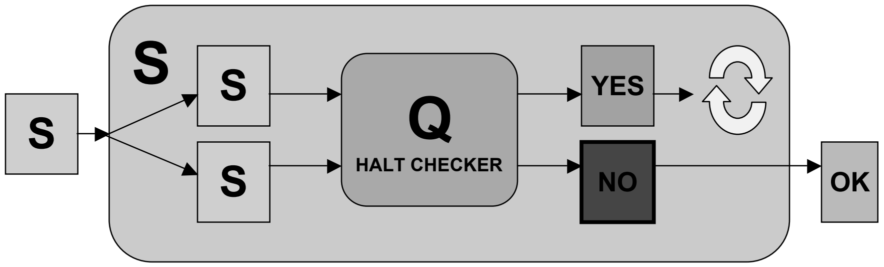
Proof by Contradiction

- If Q outputs YES, it computes that S will halt if it uses itself as input.
- But if Q outputs YES, S purposely goes into an infinite loop when it uses itself as input.



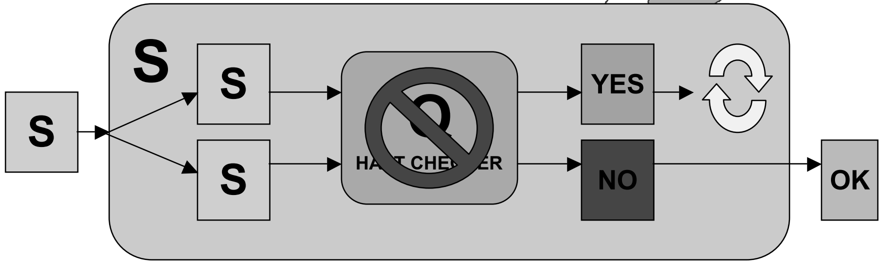
The Halting Problem (8)
Proof by Contradiction

- If Q outputs NO, it computes that S will not halt if it uses itself as input (i.e. it will run forever).
- But if Q outputs NO, S purposely halts with the output "OK" when it uses itself as input.



The Halting Problem (9)
Proof by Contradiction

- We get contradictions no matter what Q outputs.
- Our initial assumption must have been false! **Q can't exist!**



The Halting Problem is Undecidable

- We can <u>never</u> write a computer program that determines if ANY program halts with ANY input.
  - It doesn’t matter how powerful the computer is.
  - It doesn’t matter how much time we devote to the computation.
- It’s undecidable!

Contradiction isn't just for computer scientists...