The Halting Problem

P vs. NP
Halting Problem Motivation...

• Infinite loops are a problem:

```plaintext
x = 10;
while x < 20,
    x = x - 1;
end
```

• Why don't compilers check for this?

• Can't be done (perfectly). We'll see a proof by contradiction.
HALT program

• Assume we have a program Halt:
  − Take a program description, as well as a program input.
  − Returns “halt” if the program halts.
  − Returns “loops” if the program loops forever.

\[ \langle P, I \rangle \xrightarrow{} \text{Halt} \xrightarrow{} \text{“halts” or “loops”} \]

(this follows a proof from Sipser's Introduction to the Theory of Computation, 2006)
A New Machine...

- Let's create a new program D, that uses HALT as a subroutine.

- D takes a program description as input.
  - Sends the program as both program and input to HALT.
  - If HALT says “halt” D enters an infinite loop.
  - If HALT says “loops” D halts.
The Contradiction...

• Now we pass \(<D>\) to the program D.

• What happens?
  - If HALT tells D that D halts on input \(<D>\), then D loops forever on input \(<D>\).
  - If HALT tells D that D loops forever on input \(<D>\), then D halts on input \(<D>\).
What?

- We assumed the existence of a program HALT that can always determine whether a program will halt or run forever.
- We constructed a scenario in which, no matter what answer HALT returns it is wrong.
- Therefore HALT cannot exist.
Rice's Theorem

• Any non-trivial property of programs is undecidable.

• The halting problem is one example among many.
Intractable Problems

• I’ve claimed that the traveling salesperson problem is intractable.

• The best algorithm we’ve seen so far is $O(N!)$. 

• How can we know that there isn’t a better algorithm? Maybe $O(N)$?

• The short answer is that we can’t.

• There is a longer answer that leads us to conclude that TSP is very likely intractable…
P and NP

• P the set of problems that can be solved in polynomial time by some algorithm.

• NP the set of problems that can be solved in polynomial time by a non-deterministic algorithm.

• Non-deterministic??
Non-Determinism

- Deterministic algorithm must *find* a solution.
- Non-deterministic algorithms get to guess a solution, and only need to *check it*.

- Examples:
  - Sorting has a deterministic polynomial time algorithm.
  - TSP-decision problem has a non-deterministic polynomial time algorithm.
What's the Difference?

- Everyone who is anyone believes that P and NP are different sets.
- In other words, there are some problems that can be solved in polynomial time using nondeterminism, but can’t be solved in polynomial time deterministically.
An Aside: Reductions

- Problem A is said to be reducible to problem B if
  - An efficient solution to B would yield an efficient solution to A.

- Example:
  - IS-THERE-A-PATH is reducible to FIND-SHORTEST-PATH.

- Informally, if A reduces to B, B is at least as hard as A.
NP-Completeness

• It turns out there is (at least) one problem that all problems in NP reduce to: SAT.
  - SAT is the problem of checking to see if a boolean expression is satisfiable.

• Therefore an efficient solution to SAT would yield an efficient solution to every problem in NP.
  - SAT is NP-Complete.
  - SAT is (probably) intractable.

• (We won't prove this...)
Are There Other NP-Complete Problems?

- If so, how would we show it?
- Reductions – Any problem that SAT reduces to is also NP-Complete.
- There are many such problems.