Warm-Up Problem

Please fill out your Teaching Evaluation Survey! Please comment on the warm-up problems if you haven't filled in your survey yet.

Warm up: Given

> There is a barber who is said to shave only men who do not shave themselves.

Question: Does the barber shave themself?

Warm-Up Problem

Please fill out your Teaching Evaluation Survey! Please comment on the warm-up problems if you haven't filled in your survey yet.

Warm up: Given

> There is a barber who is said to shave only men who do not shave themselves.

Question: Does the barber shave themself?

What if the barber must be a man?

Decidability
Introduction and Reductions to the Halting Problem

Carmen Bruni

Lecture 22

Based on slides by Jonathan Buss, Lila Kari, Anna Lubiw and Steve Wolfman with thanks to B. Bonakdarpour, A. Gao, D. Maftuleac, C. Roberts, R. Trefler, and P. Van Beek

Last Time

- Complete the example of reversing an array and prove partial correctness.
- Show that reversing an array is totally correct.

Learning Goals

- Define a decidable problem.
- Describe the Halting Problem.
- Show that problems are decidable.
- Give reductions to prove undecidability

A Question

Given as many resources as you wanted, is every problem solvable with a computer?

A Question

Given as many resources as you wanted, is every problem solvable with a computer?

Perhaps an even easier question: What is a computer?

History of "Computers"

- Human Computers: Earliest recorded use was in 1613 (Oxford Dictionary).
- WWII: Women participated in the war as human computers since men were often drafted to the war.
- Alan Turing (1912-1954): Created the **Turing machine**, a model for a machine that used a ticker tape that moved left and right (and that is theoretically infinite in both directions), could write zeroes and ones, and a table of rules as to how to move from state to state.
- For more on this check out CS 360 (or for a Hollywood version, check out "The Imitation Game" and "Hidden Figures").

For us, we can think of a computer as a Turing Machine; A machine that outputs zeroes and ones according to an list of instructions (an algorithm) that the machine can follow.

Decision Problems

We will be discussing decision problems

Definition

A **decision problem** is a problem which has a yes or no answer on a given input.

Example

The problem "Is an integer x positive'?' is a decision problem.

Decidablility

Definition

A decision problem is **decidable** if and only if there exists an algorithm that decides it (that is, an algorithm that determines yes or no correctly on any valid input). Otherwise, we say that the decision problem is **undecidable**

Examples; Which of the following are decidable?

- "Given a twice continuously differentiable real function on a closed interval, does it have a maximum value at a given x in the interval?"
- "Given a graph (set of vertices and edges between the vertices), how many vertices does it have?"
- "Hilbert's Tenth Problem: Given an integer multivariate polynomial equation, does it have a solution"
- "Given a program P, does P terminate on input I?"

Decidablility

Definition

A decision problem is **decidable** if and only if there exists an algorithm that decides it (that is, an algorithm that determines yes or no correctly on any valid input). Otherwise, we say that the decision problem is **undecidable**

Examples; Which of the following are decidable?

- "Given a twice continuously differentiable real function on a closed interval, does it have a maximum value at a given x in the interval?"
- "Given a graph (set of vertices and edges between the vertices), how many vertices does it have?"
- "Hilbert's Tenth Problem: Given an integer multivariate polynomial equation, does it have a solution"
- "Given a program P, does P terminate on input I?"

For more information on Hilbert's Tenth Problem, see Matiyasevich's Theorem or the MRDP (Matiyasevich, Robinson, Davis, Putnam)

An Open Example

Does the following example on an input n terminate?

```
while (n > 1){
    if ( (n % 2) == 0){
        n = n/2;
    } else {
        n = 3 * n + 1;
    }
}
```

The problem of "Does this code always terminate on positive integers n?" is known as The Collatz Conjecture.

Theorem

Theorem (Turing, 1936)

The Halting Problem is undecidable. That is, the problem of "Given a program P and an input I, determine if P halts on input I" is undecidable.

Normally, I would do the proof now, but we delay this until we see some uses of this theorem.

Reduction of Problems:

To show problems are undecidable, there are two common approaches:

- Prove from first principles that the problem is undecidable.
- Prove by reducing the problem to a known undecidable problem.

Typically we use the second approach; Abstractly, we show problem Q_1 is reducible to problem Q_2:

- Given an algorithm for deciding Q_2, we could use it to decide Q_1.
- We prove:
  If Q_2 is decidable, then Q_1 is decidable.
- Taking the contrapositive:
  If Q_1 is undecidable, then Q_2 is undecidable.

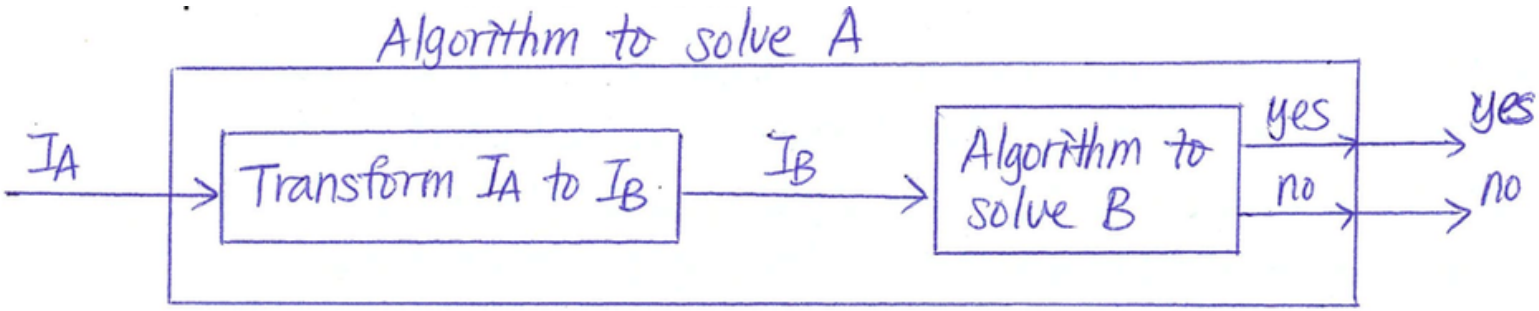
Reduction of problems.

More concretely:

- Assume towards contradiction that there is an algorithm B that decides a new problem P_0.
- Show that we can create a new algorithm A such that it decides the Halting Problem (or another problem we know is undecidable).
- Conclude by Turing's theorem that such a B should not have existed since the Halting Problem is undecidable and hence P_0 is undecidable.

Pictorially

As a diagram, we can view this as the following black box picture:



Note above, I_A is P and I if reducing to the Halting Problem as we will typically do.

Halting-No-Input Problem

Problem: Given a program P that requires no input, does P halt?

Theorem: The Halting-No-Input Problem is undecidable.

Halting-No-Input Problem

Problem: Given a program P that requires no input, does P halt?

Theorem: The Halting-No-Input Problem is undecidable.

Proof: Assume towards a contradiction that there is an algorithm B such that when the program P halts it returns "yes". We claim that we can construct an algorithm A to decide the Halting Problem. Let P and I be a program and input pair. Construct program Q that

- Takes no input
- Runs P on I. If it halts, it returns the output.

Algorithm A is given by

- Run algorithm B on program Q.
- Return the result of B run on Q.

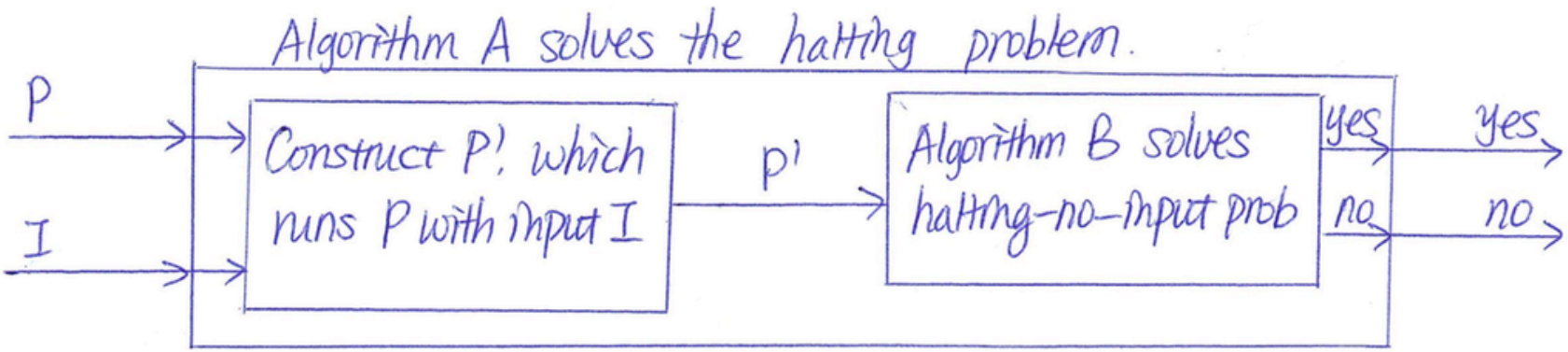
(continued on next slide)

Halting-No-Input Problem

Problem: Given a program P that requires no input, does P halt?

Theorem: The Halting-No-Input Problem is undecidable.

Proof: (Continued...) Now, by construction, A halts and returns "yes" if and only if program P halts on input I. Thus if algorithm B exists, then the Halting Problem is decidable. This is a contradiction. Thus the Halting-No-Input problem is undecidable.



Both Halt Problem

Problem: Given two programs P_1 and P_2, do both halt on a given input I?

Theorem: The Both Halt Problem is undecidable.

Both Halt Problem

Problem: Given two programs P_1 and P_2, do both halt on a given input I?

Theorem: The Both Halt Problem is undecidable.

Proof: Assume towards a contradiction that there is an algorithm B such that it return "yes" if and only if P_1 and P_2 both halt on a given input I. We claim that we can construct an algorithm A to decide the Halting Problem. Let P and I be a program and input pair. Construct program Q that does nothing and halts.

Algorithm A is given by

- Run algorithm B on programs P, Q and input I.
- Return the result of B run on this triple P, Q and I.

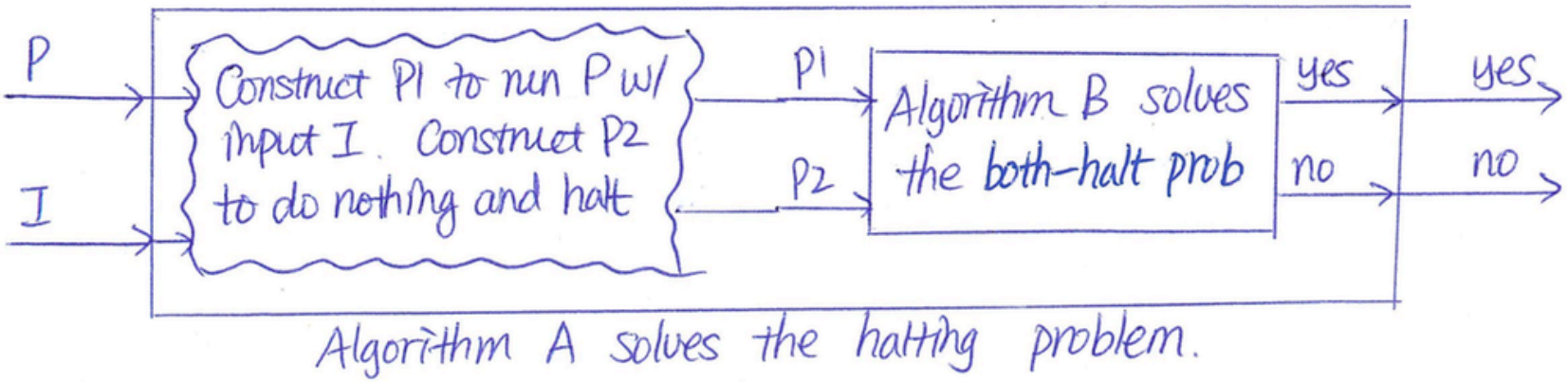
(continued on next slide)

Both Halt Problem

Problem: Given two programs P_1 and P_2, do both halt on a given input I?

Theorem: The Both Halt Problem is undecidable.

Proof: (Continued...) Now, by construction, A halts and returns "yes" if and only if program P halts on input I. Thus if algorithm B exists, then the Halting Problem is decidable. This is a contradiction. Thus the Both Halt problem is undecidable.



Partial Correctness Problem

Problem: Given a Hoare triple $(\!| P_1 |\!)\ C\ (\!| Q_1 |\!)$, does C satisfy the triple under partial correctness?

Theorem: The Partial Correctness Problem is undecidable.

Partial Correctness Problem

Problem: Given a Hoare triple $(\!| P_1 |\!)\ C\ (\!| Q_1 |\!)$, does C satisfy the triple under partial correctness?

Theorem: The Partial Correctness Problem is undecidable.

Proof: Assume towards a contradiction that there is an algorithm B such that when $(\!| P_1 |\!)\ C\ (\!| Q_1 |\!)$, is satisfied under partial correctness it returns "yes". We claim that we can construct an algorithm A to decide the Halting-No-Input Problem. Let P be a program. Algorithm A is given by

- Run algorithm B on program $(\!| \text{true} |\!)\ P\ (\!| \text{false} |\!)$ and return the negated answer.

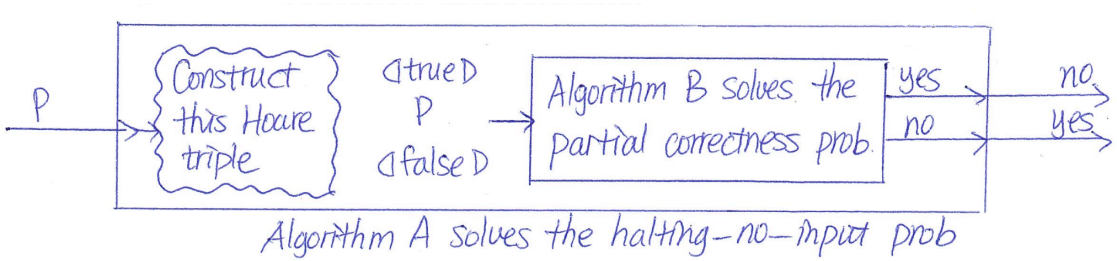
(continued on next slide)

Partial Correctness Problem

Problem: Given a Hoare triple $(\!| P_1 |\!)\ C\ (\!| Q_1 |\!)$, does C satisfy the triple under partial correctness?

Theorem: The Partial Correctness Problem is undecidable.

Proof: (Continued...) Now, by construction, A halts and returns "yes" if and only if program P halts. This is true since the Hoare triple $(\!| \text{true} |\!)\ P\ (\!| \text{false} |\!)$ is never satisfied and so by the definition of partial correctness, this triple satisfies partial correctness if and only if P does not halt. Thus algorithm B will return "no" if and only if program P halts. This is the negation of what we want our program do so flip the answer. Thus if algorithm B exists, then the Halting-No-Input Problem is decidable. This is a contradiction. Thus the Partial Correctness problem is undecidable.



Specific Input Run Forever Problem

Problem: Given a program P, does it run forever on input $n = 0$?

Theorem: The Specific Input Run Forever Problem is undecidable.

Specific Input Run Forever Problem

Problem: Given a program P, does it run forever on input $n = 0$?

Theorem: The Specific Input Run Forever Problem is undecidable.

Proof: Assume towards a contradiction that the specific input problem is decidable. That is, there is an algorithm B such that when I give it a program P', it returns "yes" if and only if it runs forever on the input $n = 0$.
Given a program P and an input I, construct a program Q that:

- Ignores the input and runs P on I.
- If P when run on I halts, then halt and return 0.

Note that this Q halts at 0 [in fact it halts on *all* inputs!] if and only if P halts on I. We now show we can decide the Halting Problem. (Continued on next slide)

Specific Input Problem

Problem: Given a program P, does it run forever on input $n = 0$?

Theorem: The Specific Input Problem is undecidable.

Proof: (Continued...) We now find an algorithm that decides the Halting Problem. Consider an algorithm A:

- It consumes P and I
- It creates Q
- It runs algorithm B on Q and then negates the output.

Now, algorithm B on Q returns "yes" if and only if Q runs forever on input $n = 0$. But, this can happen by construction if and only if P does not halt on I. So negating the final answer means that if we get "no" from the above algorithm, then we have that P does not halt on input I, a contradiction. Hence algorithm B cannot exist, that is, the Specific Input Run Forever Problem is undecidable.