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 a program P that accepts input, is there an input where the program runs forever?

Theorem: The Exists-Input-Runs-Forever Problem is undecidable.

Decidability
Introduction and Reductions to the Halting Problem

Carmen Bruni

Lecture 24

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

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

Learning Goals

- Prove that the Halting Problem is Undecidable
- Prove other problems are undecidable based on reductions to the Halting Problem.

The Halting problem

The decision problem: Given a program P and an input I, will P halt when run with input I?

- "Halts" means "terminates" or " does not get stuck."
- One of the first known undecidable problems

The Halting problem is undecidable.

There does not exist an algorithm H, which gives the correct answer for the Halting problem for every program P and every input I.

Exercise: Translate the above statement into a Predicate formula.

The Halting Problem is Undecidable

A proof by video

`https://www.youtube.com/watch?v=92WHN-pAFCs`

Common questions about the video

- Why can we feed a program as an input to itself? That is, why does $H(P, P)$ make sense?
  We can convert any program to a string, then we can feed the string of the program to itself as input.

- What does the negator do?
  It negates the behaviour of the machine. If H predicts that the program halts, then the negator goes into an infinite loop and does not halt. If H predicts that the program does not halt, then the negator halts.
  The negator is designed to make H fail at its prediction task.

- Why do we need the photocopier?
  In the video, H takes two inputs. We need to make two copies of the input. In code, we do not need the photocopier. We simply need to call $H(P, P)$.

The Halting Problem is Undecidable

Theorem: The Halting problem is undecidable.

Proof by contradiction.

Assume that there exists an algorithm H, which decides the Halting problem for every program and every input.

We will construct an algorithm X, which takes program P as input.

We will show that H gives the wrong answer when predicting whether the program X halts when run with input X. This contradicts the fact that H decides the Halting problem for every program and every input. Therefore, H does not exist.

□

The Halting Problem is Undecidable.

The algorithm X:

- Takes a program Q as input
- Makes two copies of Q
- Runs $H(Q, Q)$:
  - If $H(Q, Q)$ outputs "yes", then X goes into an infinite loop and runs forever (ie. it doesn't halt)
  - If $H(Q, Q)$ outputs "no", then X halts and returns 0

Now, we ask "What happens if we try $X(X)$?"

Proof that $X(X)$ Does Halt Is Impossible

Suppose that $X(X)$ halted. Then $H(X, X)$ by construction must have outputted "no", that is, X does not halt when given itself as input. This contradicts our assumption that X did halt when given itself as input.

Proof that $X(X)$ Does Not Halt Is Impossible

Suppose that $X(X)$ did not halt. Then $H(X, X)$ by construction must have outputted "yes", that is, X does halt when given itself as input. This contradicts our assumption that X did not halt when given itself as input.

The combination of this slide and the next shows that this machine X cannot exist. However, every part of the machine X does exist except for possibly the machine H. Hence H cannot exist and we have a contradiction.

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 solve 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 solve 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.

Another Undecidable Problem

Provability of a formula in Predicate Logic:

> Given a formula φ, does φ have a proof?
> (Or, equivalently, is φ valid?)

No algorithm exists to solve provability:

Theorem Provability is undecidable.

Provability In Predicate Logic Is Undecidable

Outline of proof: Suppose that some algorithm Q decides provability.

1. Devise an algorithm to solve the following problem:

   Given a program P and input I, produce a formula $\varphi_{P,I}$ such that $\varphi_{P,I}$ has a proof iff P halts on input I.

2. Combine algorithm Q with the above algorithm to get an algorithm that decides the Halting Problem: On input (P, I), give the formula $\varphi_{P,I}$ as input to Q.

But no algorithm decides the Halting problem.

Thus no algorithm decides provability.

Another Undecidable Problem

The Post Correspondence Problem (devised by Emil Post)

Given a finite sequence of pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ such that all s_i and t_i are binary strings of positive length, is there a sequence of indices $i_1, i_2, \ldots, i_n$ with $n \geq 1$ such that the concatenation of the strings $s_{i_1} s_{i_2} \ldots s_{i_n}$ equals $t_{i_1} t_{i_2} \ldots t_{i_n}$?

An Instance of the Post Correspondence Problem

Suppose we have the following pairs: `(1,101),(10,00),(011,11)`. Can we find a solution for this input?

Yes. Indices (1,3,2,3) work: $s_{i_1}s_{i_3}s_{i_2}s_{i_3}$ equals $t_{i_1}t_{i_3}t_{i_2}t_{i_3}$, as both yield `101110011`.

What about the pairs `(001,0)`, `(01,011)`, `(01,101)`, `(10,001)`?

In this case, there is no sequence of indices.

Remember that an index can be used arbitrarily many times. This gives us some indication that the problem might be unsolvable in general, as the search space is infinite.

Check out CS360/365 for a complete proof!

Poll Everywhere!

Some final Questions about the course