Class 25: Undecidable Problems

Menu

- Review:
  - Undecidability
  - Halting Problem
- How do we prove a problem is undecidable?
- What do we do when faced with an undecidable problem?

Problem Classes if \( P \neq NP \):

- Decidable
- \( P \)
- \( NP \)
- \( \Theta(n^3) \)
- \( \Theta(n) \)
- Smalleys
- NP-Complete
- 3SAT
- Undecidable

Halting Problem

Define a procedure \( \text{halts?} \) that takes a procedure and an input evaluates to \#t if the procedure would terminate on that input, and to \#f if it would not terminate.

\[
\text{(define (halts? procedure input) … )}
\]

Informal Proof

\[
\begin{align*}
\text{(define (contradict-halts x)} & \text{)} \\
\text{(if (halts? contradict-halts null)} & \text{)} \\
\text{(loop-forever)} & \text{#t)}
\end{align*}
\]

If contradict-halts halts, the if test is true and it evaluates to (loop-forever) - it doesn't halt!

If contradict-halts doesn't halt, the if test if false, and it evaluates to \#t. It halts!

Proof by Contradiction

1. Show \( X \) is nonsensical.
2. Show that if you have \( A \) and \( B \) you can make \( X \).
3. Show that you can make \( A \).
4. Therefore, \( B \) must not exist.

\[
\begin{align*}
X = \text{contradict-halts} \\
A = \text{a Scheme interpreter that follows the evaluation rules} \\
B = \text{halts?}
\end{align*}
\]
“Evaluates to 3” Problem
Input: A procedure $P$ and input $I$
Output: true if evaluating $(P \ I)$ would result in 3; false otherwise.

Is “Evaluates to 3” decidable?

Undecidability Proof
Suppose we could define evaluates-to-3? that decides it. Then we could define halts?:

\[
\begin{align*}
\text{(define (halts? P I) } & \\
\text{ (if (evaluates-to-3? (begin (P I) 3)) } & \\
\t & \\
\text{ (begin (P I) 3)) } & \\
\text{ #t } & \\
\text{ #f) } & \\
\end{align*}
\]

Since it evaluates to 3, we know $(P I)$ must halt. (The only way it could not evaluate to 3, is if $(P I)$ doesn’t halt. (Note: assumes $(P I)$ cannot produce an error.)

Hello-World? Problem
Input: A procedure $P$ and input $I$
Output: true if evaluating $(P \ I)$ would print out “Hello World!”; false otherwise.

Is Hello-World? decidable?

Undecidability Proof
Suppose we could define hello-world? that decides it. Then we could define halts?:

\[
\begin{align*}
\text{(define (halts? P I) } & \\
\text{ (if (hello-world? (begin ((remove-prints P) I) (print “Hello World!”)) } & \\
\text{ #t } & \\
\text{ #f)) } & \\
\end{align*}
\]

Proof by Contradiction
1. Show $X$ is nonsensical.
2. Show that if you have $A$ and $B$ you can make $X$.
3. Show that you can make $A$.
4. Therefore, $B$ must not exist.

$X = \text{halts}?$
$A = \text{a Scheme interpreter that follows the evaluation rules}$
$B = \text{hello-world}?$

From Paul Graham’s “Undergraduation”:
My friend Robert learned a lot by writing network software when he was an undergrad. One of his projects was to connect Harvard to the Arpanet; it had been one of the original nodes, but by 1984 the connection had died. Not only was this work not for a class, but because he spent all his time on it and neglected his studies, he was kicked out of school for a year. ...

... When Robert got kicked out of grad school for writing the Internet worm of 1988, I envied him enormously for finding a way out without the stigma of failure. ...

... It all evened out in the end, and now he’s a professor at MIT. But you’ll probably be happier if you don’t go to that extreme; it caused him a lot of worry at the time.

3 years of probation, 400 hours of community service, $10,000+$ fine
Morris Internet Worm (1988)
- $P = \text{fingerd}$
  - Program used to query user status
  - Worm also attacked other programs
- $I = \text{"nop" 400 pushl$68732f pushl$6e69622f movl sp,r10 pushl$0 pushl$0 pushl r10 pushl$3 movl sp,ap chmk $3b"}$
  - (is-worm? $P I$) should evaluate to $\#t$
- Worm infected several thousand computers (~10% of Internet in 1988)

Worm Detection Problem
Input: A program $P$ and input $I$
Output: true if evaluating $(P I)$ would cause a remote computer to be "infected".

Virus Detection Problem
Input: A program $P$ and input $I$
Output: true if evaluating $(P I)$ would cause a file on the host computer to be "infected".

Undecidability Proof
Suppose we could define is-worm? Then:
$(\text{define (halts? } P I) \quad (\text{if (is-worm? } (\text{begin ((deworm } P I) \text{) worm-code)})) \quad \#t$
Since it is a worm, we know worm-code was evaluated, and $P$ must halt.
$\#f))$
The worm-code would not evaluate, so $P$ must not halt.
Can we make deworm?

Conclusion?
- Anti-Virus programs cannot exist!
  - "The Art of Computer Virus Research and Defense" Peter Szor, Symantec

"Solving" Undecidable Problems
- No perfect solution exists:
  - Undecidable means there is no procedure that:
    1. Always gives the correct answer
    2. Always terminates
  - Must give up one of these to "solve" undecidable problems
    - Giving up $\#2$ is not acceptable in most cases
    - Must give up $\#1$

Actual is-virus? Programs
- Give the wrong answer sometimes
  - "False positive": say $P$ is a virus when it isn't
  - "False negative": say $P$ is safe when it is
- Database of known viruses: if $P$ matches one of these, it is a virus
- Clever virus authors can make viruses that change each time they propagate
  - A/V software ~ finite-proof-finding
  - Emulate program for a limited number of steps; if it doesn't do anything bad, assume it is safe
Proof Recap

- If we had is-virus? we could define halts?
- We know halts? is undecidable
- Hence, we can't have is-virus?
- Thus, we know is-virus? is undecidable

How convincing is our Halting Problem proof?

(define (contradict-halts x)
  (if (halts? contradict-halts null)
      (loop-forever)
      #t))

If contradict-halts halts, the if test is true and it evaluates to (loop-forever) - it doesn't halt!
If contradict-halts doesn't halt, the if test if false, and it evaluates to #t. It halts!

This "proof" assumes Scheme exists and is consistent!

Charge

- Scheme is very complicated (requires more than 1 page to define):
  - Unlikely we could prove it is consistent
- To have a convincing proof, we need a simpler programming model in which we can write contradict-halts:
  - Next week: Turing's model