CS245 — Logic and Computation

Undecidability and the Halting Problem

Jonathan Buss

with thanks to

Borzoo Bonakdarpour, Daniela Maftuleac and Tyrel Russell

What is Computability?

From an informal or intuitive perspective what might we mean by computability?

One natural interpretation is that something is **computable** if it can be calculated by a systematic procedure.

We may think that given enough resources and a clever enough program that a computer could solve any problem.

However, some problems cannot be automated.

What is Computability?

We remark that an instruction such as *"guess the correct answer"* does not seem to be systematic.

An instruction such as *"try all possible answers"* is less clear cut: it depends on whether the possible answers are finite or infinite in number.

Decision Problems

A ***decision problem*** is a problem which calls for an answer of either **yes** or **no**.

Examples

1. Given a formula of propositional logic, is it satisfiable?
2. Given a positive integer, is it prime?
3. Given a graph and two of its vertices, is there a path between the two vertices?
4. Given a multivariate polynomial equation, does it have any integer solutions?
5. Given a program and input, will the program terminate on the input?

Decision problem

Sometime an apparently reasonable decision problem has subtleties.

Example: **The Barber Paradox**

> There is a barber who is said to shave all men, and only those men, who do not shave themselves. Who shaves the barber?

As phrased, the barber could be a woman. But what if we insist the barber is a man?

Then if the barber shaves himself it is because he does not shave himself, and in turn this is because he does shave himself.

"Does the barber shave himself?" is unanswerable.

A Curious Problem

Do the following terminate, given a positive integer n?

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

```
( define (C n)
    ( cond
        ( (<= n 1 ) 1 )
        ( (even? n) (C (/ n 2) ) )
        ( else ( C (+ (* 3 n) 1) ) )
        ) )
( C n )
```

Nobody knows!

Nevertheless, it is a decision problem.

Decidable and Undecidable Problems

A decision problem is *decidable* if there is an algorithm that, given an input to the problem,

- outputs "yes" (or "true") if the input has answer "yes" and
- outputs "no" (or "false") if the input has answer "no".

A problem is *undecidable* if it is not decidable.

The Halting Problem

One of the best-known undecidable problem is the ***Halting Problem***.

> Given a program P and an input I, will P terminate if run on input I?

E.g.: A Scheme program that does not terminate:

```
(define (loop x) (loop loop))
```

Can we write a program which takes as an input any program P and input I for P and returns true, if the program terminates (halts) on I and returns false, otherwise?

Testing Whether a Program Halts

```
;; Contract: halts?:  SchemeProgram Input → boolean
;;
;; If the evaluation of (P I) halts, then (halts? P I)
;; halts with value true, and
;; If the evaluation of (P I) does not halt, then
;; (halts? P I) halts with the value false.
;;
;; Example: (halts? loop loop) returns false,
;; given definition (define (loop x) (loop loop))


( define ( halts? P I )
        :
   )
```

Testing Whether a Program Halts

Theorem.

> No Scheme function can perform the task required of `halts?`, correctly for all programs.

That is, the Halting Problem is undecidable.

Undecidability of Halting: Proof by contradiction

Proof: By contradiction.

Assume that there exists some function `(halts? P I)` which returns `true` if the program P halts on input I and returns `false` if P does not halt on input I.

A key idea: one possible input to the program P would be the program P itself. Therefore, `(halts? P P)` would return `true` if the program halts when given itself as an input.

A function `halts?` could be called by other functions.

A Program Calling Itself

Let's consider the following function, which uses `halts?`.

```
(define ( self-halt? P )
   ( halts? P P ) )
```

This should determine whether P terminates when given itself as input.

What happens if we call `(self-halt? self-halt?)` ?

"What Do I Do?"

`(self-halt? self-halt?)`

$\Rightarrow$ `(halts? (self-halt? self-halt?))`

$\Rightarrow \ldots$

$$\Rightarrow \begin{cases} \texttt{true}, & \texttt{if (self-halt? self-halt?) halts} \\ \texttt{false}, & \texttt{if (self-halt? self-halt?) does not halt.} \end{cases}$$

Since evaluation of `halts?` always terminates, the evaluation of `(self-halt? self-halt?)` also terminates. And, since `halts?` gives the correct answer, the final result must be `true`.

"What Do I Do, If I Don't?"

OK, now let's consider the following function.

```
( define ( halt-if-loop P )
      ( cond
            [ (halts? P P) (loop loop) ]
            [ else  true ]
      )
)
```

What happens if we invoke `halt-if-loops` with itself as its argument?

"Aacckk!"

```
( halt-if-loop  halt-if-loop)
  ⇒ (cond [ ( halts? halt-if-loop halt-if-loop )
            (loop loop)]
          [ else true ]
          )
  ⇒ ...
```

$$\Rightarrow \begin{cases} \texttt{(loop loop)}, & \\ \qquad\qquad \texttt{if (halt-if-loop halt-if-loop) halts}, & \\ \texttt{true}, \quad \texttt{if (halt-if-loop halt-if-loop) does not halt}. & \end{cases}$$

The evaluation of `(halt-if-loop halt-if-loop)` terminates iff the evaluation of `(halt-if-loop halt-if-loop)` does not terminate.

Program `halt-if-loop` cannot exist!
Thus also program `halts?` cannot exist.

Undecidability of the Halting Problem

The proof of the undecidability of the halting problem uses a technique called **diagonalization**, devised by Georg Cantor in 1873.

Cantor was concerned with the problem of measuring the sizes of infinite sets. For two infinite sets, how can we tell whether one is larger than the other or whether they are of the same size?

Example. Which set is larger, the set of even integers or the set of all infinite sequences over $\{0, 1\}$?

Another Undecidable Problem

Provability of a formula in FOL:

> 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:

1. Devise an algorithm to solve the following problem:

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

   (Base the formula on *Step*.)

2. If some algorithm solves Provability, then we can combine it with the above algorithm to get an algorithm that solves the Halting Problem.

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 imput?

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 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.

Yet Another Undecidable Problem

The problem

> Given a multivariate polynomial equation, does it have any integer solutions?

is undecidable.

The proof is similar in principle: show that deciding whether an equation has integral solutions allows deciding whether a program halts.

The details, however, get quite complicated.

The issue was first raised in 1900 (by Hilbert) and not solved until 1971 (by Matjasevič and Robinson).