Lecture 17: Undecidability II

built on 2021/03/02 at 09:41:18

1 Recap: A High-Level View

What models have we seen so far?

- The finite-state machines (FSMs): they have finite states; no memory otherwise. one pass over input.
- The Turing machines (TMs): finite states; infinite mem (no access restriction).

In terms of computability power (i.e., what can be computed/solved using them), it has been shown that: FSM $<$ TM (which includes many other models, including λ-calculus). This means, there are things a TM can do but an FSM cannot.

1.1 Decidability

Our notion of computability is centered upon Turing computability. More precisely, we say that

a language L on Σ^* is *decidable* $\iff$ there is a Turing machine M such that

- M halts on every input from Σ^*; and
- M accepts every $x \in L$ and rejects everything else.

Based on this idea, we say that a language L is *<u>un</u>decidable* (i.e., not decidable) $\iff$ there exists no Turing machine M such that

- M halts on every input from Σ^*; and
- M accepts every $x \in L$ and rejects everything else.

1.2 ACCEPT is undeciable

Consider the following language

$$\mathsf{ACCEPT} = \{\langle M, x\rangle \mid M \text{ is a TM which accepts } x\}.$$

Remember that $\langle M, x\rangle$ is an encoding of a pair consisting of the machine M and a string x. We'll prove that ACCEPT is undeciable. Let's prove this directly using the method we used to prove that HALT is undeciable.

Proof: Suppose for a contradiction that ACCEPT is decidable, so there is a TM M_A that correctly decides ACCEPT. Consider the following Turing machine D:

Input: $\langle M\rangle$

The machine D proceeds as follows:

1. Run the Turing machine M_A with input $\langle M, \langle M\rangle\rangle$
2. If M_A says ACCEPT, then D rejects.
   If M_A says REJECT, then D accepts. (Why can't M_A loop forever?)

How do we get a contradiction? Before we proceed, let's try to remind ourselves of a few things:

- As a decider for ACCEPT, $M_A(\langle M, x\rangle)$ offers a prediction whether the Turing machine M will accept on input x.
- We say that M_A is correct if the prediction is correct on all possible inputs. That is, M_A is wrong if it predicts incorrectly *even* on one of the input.

To derive a contradiction, we'll consider what happens to M_A when fed the input $\langle D, \langle D \rangle\rangle$. Specifically, we'll look at what M_A predicts and what the machine actually does. Consider the outcome of $M_A(\langle D, \langle D \rangle\rangle)$.

- $M_A(\langle D, \langle D \rangle\rangle)$ says ACCEPT: that is M_A predicts that D will accept $\langle D \rangle$. If we examine D, we'll see that D will do the exact opposite—it will reject.
- $M_A(\langle D, \langle D \rangle\rangle)$ says REJECT: that is M_A predicts that D will reject $\langle D \rangle$. If we examine D, we'll see that D will, again, do the exact opposite—it will accept.

Hence, either way, we have that M_A mispredicts at least *one* TM, precisely the TM D, leading to a contradiction, so M_A cannot exist. ■

1.3 Reductions: Solve A Using B

Say you have two problems (or languages) A and B. In order to solve A, you will come up with a Turing machine (or an algorithm) for solving A. Let's call that M_A. If M_A in the process of solving A uses M_B—the solution for B—as a subroutine, then we say that A reduces to B. Notationally, we write $A \leqslant_T B$.

This literally means A is **no harder than** B. This is true because you can use B to solve A.

This also means:

> Suppose $A \leqslant_T B$. If B is decidable, then so is A.

The contrapositive of the following statement is:

> Suppose $A \leqslant_T B$. If A is undecidable, then so is B.

2 Many Things Are Equally Hard

As a reminder, we've proved (and you still remember) that HALT is undecidable.

Theorem 2.1 *HALT is undecidable.*

There is apparently an easier(?) way to prove that ACCEPT is undecidable: You probably believe that ACCEPT is harder than HALT. Here is a new proof strategy: Try to show that *ACCEPT is at least as hard as HALT*. That is, it suffices to show that $\mathsf{HALT} \leqslant_T \mathsf{ACCEPT}$.

Theorem 2.2 *ACCEPT is undecidable.*

Proof: Suppose for a contradiction that ACCEPT is decidable, so there is a TM M_{ACCEPT} that decides it. We'll show a Turing machine that decides HALT using M_{ACCEPT}. Given $\langle M, x \rangle$ as input:

1. Run $M_{\mathsf{ACCEPT}}(\langle M, x \rangle)$. If it accepts, we'll also accept.
2. Change M by reversing the role of accept/reject, call this M'.
3. Run $M_{\mathsf{ACCEPT}}(\langle M', x \rangle)$. If it accepts, we'll accept.
4. Otherwise, we reject

This means, for a given $\langle M, x \rangle$, if M accepts x, we'll accept; if M rejects x, we'll accept. The only case that we'll reject is when M neither accept nor reject x—that is, M loops on x.

This means our TM can correctly decide HALT provided that there is a TM M_{ACCEPT}. But this is absurd—no one can decide HALT! So we know that M_{ACCEPT} cannot possibly exist, a contradiction! ■

Aside: By showing a TM that decides HALT using a TM for ACCEPT, we show that $\mathsf{HALT} \leqslant_T \mathsf{ACCEPT}$. This means, if HALT is "hard", ACCEPT is at least as hard—because one can use it as a subroutine to solve HALT. This lets us conclude that because HALT is undecidable, ACCEPT is also undecidable.

Python? If we were to describe this reduction using real programs, here's roughly what it means to reduce HALT to ACCEPT:

First, we suppose that there is a solution for ACCEPT. This means, there is a function (i.e., program) `decide_accept(M, x)` where

> `decide_accept(M, x)` will say "yes" if running `M(x)` will result in accept; and it will say "no" otherwise.

Our goal is to implement `decide_halt`—the specification is the same as above. The thing to keep in mind is we have `decide_accept` and we could call it. Here's how we can implement `decide_halt`

```python
from magicbox import decide_accept

def decide_halt(M, x):
    if decide_accept(M, x):
        # M is known to accept x so it surely halts on x
        return True
    # this function returns the negation of M(x)
    def M_prime(M, x):
        return not M_prime(M, x)
    return decide_accept(M_prime, x)
```

This means that if `decide_accept` existed and followed the claimed specifications, then we would readily have `decide_halt`, which has been ruled out as impossible earlier. This is why `decide_accept` can't possibly exist.

2.1 What Else Is Hard?

Theorem 2.3 *ALL* $= \{\langle M\rangle \mid M$ *accepts all strings*$\}$ *is undecidable.*

Proof: (by reduction from ACCEPT, i.e., ACCEPT $\leqslant_T$ ALL) Suppose for a contradiction that ALL is decidable, so there is a TM M_{ALL} that decides it. Now we're going to show how to decide ACCEPT using M_{ALL}. Given $\langle M, x\rangle$:

1. Write down the description $\langle M'\rangle$, where M' is a TM that does the following: Overwrite the input with x and then run M (Remember that M' has the description of x baked into its description.)
2. Run M_{ALL} with $\langle M'\rangle$ as input.
3. If it accepts, we accept; if it rejects, we reject.

Notice that this mechanism accepts if and only if $M(x)$ accepts (after all, it discards the input provided to M'). ■

> **Exercise:** Show that EMPTY $= \{\langle M\rangle \mid M$ accepts nothing$\}$ is undecidable. (*Hint:* show that ACCEPT $\leqslant_T$ EMPTY.)