CS103 Handout 24
Fall 2013 November 15, 2013

Alternate Proofs on Turing Machines

We've gone over many proofs exploring why certain problems are impossible to solve with computers. These results are fundamental in theoretical computer science and, appropriately, there are many different ways to prove them. This handout includes some alternate proofs of some of the results that we've covered. I hope this will shed more light on why the results are true. I also hope that this handout will give you some examples of proofs involving Turing machines that you can use as a reference in the problem sets.

Proving $L_D \notin$ RE

The first "impossible problem" we encountered was L_D, which was defined as

$$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$$

This language is the diagonalization language you get if you diagonalize against the languages of all Turing machines. The proof we did in lecture was a standard diagonalization proof along the lines of the ones we've done in the past. It's also possible to prove this result in a different way by thinking more directly about what a TM for L_D would have to do on itself.

If R is a recognizer for L_D, then $\mathscr{L}(R) = L_D$. This means that for any TM M, we have $\langle M \rangle \in \mathscr{L}(R)$ iff $\langle M \rangle \in L_D$. So what happens, in particular, if we focus on the machine R and its encoding $\langle R \rangle$? Substituting in $\langle R \rangle$ for $\langle M \rangle$ in the preceding step, this means that $\langle R \rangle \in \mathscr{L}(R)$ iff $\langle R \rangle \in L_D$. Now, remember that L_D is the set of all descriptions of TMs that don't accept their own encodings, which means that $\langle R \rangle \in L_D$ iff $\langle R \rangle \notin \mathscr{L}(R)$. But this means $\langle R \rangle \in \mathscr{L}(R)$ iff $\langle R \rangle \notin \mathscr{L}(R)$, and that's definitely troubling!

We can formalize this as follows:

Proof: By contradiction; suppose that $L_D \in$ **RE**. This means that there is some recognizer R where $\mathscr{L}(R) = L_D$. Since R is a recognizer for L_D, we know for any TM M that $\langle M \rangle \in L_D$ iff $\langle M \rangle \in \mathscr{L}(R)$. By definition, we know for any TM M that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Therefore, combined with the above statement, we see for any TM M that $\langle M \rangle \notin \mathscr{L}(M)$ iff $\langle M \rangle \in \mathscr{L}(R)$. In particular, this means that $\langle R \rangle \notin \mathscr{L}(R)$ iff $\langle R \rangle \in \mathscr{L}(R)$, which is impossible. We have reached a contradiction, so our assumption must have been wrong. Thus $L_D \notin$ **RE**. ■

In many ways this proof is more direct than the one from lecture. I opted to use a different proof in lecture because I thought it would more directly call back to our diagonalization proofs from earlier in the quarter, though I think this particular proof is more elegant.

There's yet another way you can prove this result. The definition of L_D given above is phrased in terms of the languages of Turing machines, but it can also be expressed equivalently in terms of the behavior of various TMs on their own encodings. In particular:

$$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ does not accept } \langle M \rangle \}$$

This gives a more "mechanical" description of what L_D is – it's the set of all descriptions of TMs that don't accept their own descriptions. We can prove $L_D \notin$ **RE** by using a more "mechanical" proof by talking about what a recognizer for L_D would have to do when run on itself. Let's suppose that R is a recognizer for L_D. Then:

- If R accepts $\langle R \rangle$, then $\langle R \rangle \notin L_D$, and so R doesn't accept $\langle R \rangle$.
- If R doesn't accept $\langle R \rangle$, then $\langle R \rangle \in L_D$, and so R accepts $\langle R \rangle$.

That's a problem – R accepts itself iff it doesn't accept itself! That gives us a contradiction, so we know that there can't be a TM like R whose language is L_D. Here's a formal version of that proof:

Proof: By contradiction; assume that $L_D \in$ **RE**. That means that there is some recognizer R where $\mathscr{L}(R) = L_D$. Consider what happens when we run R on its own encoding $\langle R \rangle$. We consider two options:

Case 1: R accepts $\langle R \rangle$. This means that $\langle R \rangle \in \mathscr{L}(R) = L_D$. Since $\langle R \rangle \in L_D$, we see that R does not accept $\langle R \rangle$. This is impossible, since in this case we're assuming that R accepts $\langle R \rangle$.

Case 2: R does not accept $\langle R \rangle$. By definition of L_D, this means that $\langle R \rangle \in L_D$. Since R is a recognizer for L_D, we see that R must accept $\langle R \rangle$. This is impossible, since in this case we're assuming that R does not accept $\langle R \rangle$.

In both cases, we reach a contradiction, so our assumption must have been wrong. Therefore, we know that $L_D \notin$ **RE**. ■

Proving $A_{TM} \notin$ R

Our first example of an undecidable problem was A_{TM}, the language of the universal Turing machine. This language is defined as

$$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$$

The proof that A_{TM} is undecidable that we covered in lecture was built on top of several other results, namely that $\overline{A_{TM}} \notin$ **RE**, that **R** is closed under complementation, and that **R** ⊆ **RE**. As I mentioned in lecture, this is not the standard proof of this result. Typically, this result is proven "all at once" by combining together the necessary pieces of these earlier proofs together. The resulting proof usually works by showing that a decider for A_{TM} will lead to a the construction of a paradoxical machine that accepts itself iff it doesn't accept itself.

Here is an alternate proof that $A_{TM} \notin$ **RE** that works by using this approach:

Proof: By contradiction; suppose that $A_{TM} \in \mathbf{R}$. This means that there must be a decider D where $\mathscr{L}(D) = A_{TM}$.

Now, consider this new machine TM H:

```
H = "On input ⟨M⟩, where M is a TM:
        Run D on ⟨M, ⟨M⟩⟩.
        If D accepts ⟨M, ⟨M⟩⟩, then H rejects ⟨M⟩.
        If D rejects ⟨M, ⟨M⟩⟩, then H accepts ⟨M⟩."
```

First, notice that H is a decider. To see this, note that on any input $\langle M \rangle$, the TM H first runs D on the string $\langle M, \langle M \rangle \rangle$. Since D is a decider, D halts when run on this input. As soon as D halts, the machine H halts as well. Therefore, H halts on all inputs.

Now, consider what happens when we run H on $\langle H \rangle$. We consider two options:

Case 1: H accepts $\langle H \rangle$. This means that D rejects $\langle H, \langle H \rangle \rangle$. Since D is a decider for A_{TM}, this means that H does not accept $\langle H \rangle$. But this is impossible, since we're assuming that H does indeed accept $\langle H \rangle$.

Case 2: H rejects $\langle H \rangle$. This means that D accepts $\langle H, \langle H \rangle \rangle$. Since D is a decider for A_{TM}, this means that H accepts $\langle H \rangle$. But this is impossible, since we're assuming that H does not accept $\langle H \rangle$.

In both cases we get a contradiction, so our assumption must have been wrong. Thus $A_{TM} \notin \mathbf{R}$. ■

I personally think this proof is trickier to follow than the one from lecture, though it is more self-contained. It does give a good example of the sort of "iff chain" reasoning that often happens when working with Turing machines.

Proving *HALT* ∉ R

The undecidability of the halting problem is a hallmark result in theoretical computer science and, appropriately, it's been proven in many, many different ways.

Recall that *HALT* is the language of all TM/string pairs where the TM halts on the input string:

$$HALT = \{ \langle M, w \rangle \mid M \text{ is a TM that halts on } w \}$$

The proof that we went over in lecture essentially used a mapping reduction (which we'll explore on Friday) to show that a decider for *HALT* could be used to build a decider for A_{TM}, which is impossible. There's another way to show that *HALT* is undecidable that also works by building a decider for A_{TM} out of a decider for *HALT*, but by going along a different route that in some ways might be more instructive.

The intuition behind the proof is the following: the "hard part" of A_{TM} is handling the case where the input TM M goes into an infinite loop when run on the input string w. If M never looped on w, then we could just run *M* on *w* and see whether it accepts or rejects without having to worry about *M* looping. If *HALT* were decidable, then we could patch this hole by using a decider for *HALT* to check whether the input TM would halt on *w*. If so, we can just run the TM on *w* and see what happens. If not, then we know for a fact that *M* isn't going to accept *w*, so we can just reject the input without any more processing.

This gives rise to the following proof:

Proof: By contradiction; assume that $HALT \in \mathbf{R}$. Then there is a decider *H* where $\mathscr{L}(H) = HALT$. Now, consider the following TM *D*:

```
D = "On input ⟨M, w⟩, where M is a TM and w is a string:
        Run H on ⟨M, w⟩.
        If H rejects ⟨M, w⟩, then D rejects ⟨M, w⟩.
        Otherwise (that is, H accepts ⟨M, w⟩), run M on w.
        If M accepts w, then D accepts ⟨M, w⟩.
        If M rejects w, then D rejects ⟨M, w⟩."
```

We claim that *D* is a decider for A_{TM}. Since we know $A_{TM} \notin \mathbf{R}$, this is impossible and we have reached a contradiction. Therefore, our assumption must have been wrong, so it must be the case that $HALT \notin \mathbf{R}$.

To prove that *D* is a decider for A_{TM}, we first prove that *D* is a decider, then that $\mathscr{L}(D) = A_{TM}$. To see that *D* is a decider, consider the operation of *D* on an arbitrary TM/string pair ⟨*M*, *w*⟩. First, *D* runs the decider *H* on ⟨*M*, *w*⟩, which must halt. If *H* rejected ⟨*M*, *w*⟩, then *D* rejects ⟨*M*, *w*⟩ and halts. Otherwise, *H* accepts ⟨*M*, *w*⟩, and since *H* is a decider for *HALT*, we know that *M* halts on *w*. Therefore, when *H* runs *M* on *w*, we are guaranteed that *M* will halt on *w*. Since *D* halts as soon as *M* halts on *w*, we know that in this case *D* must halt on ⟨*M*, *w*⟩. Since our choice of ⟨*M*, *w*⟩ was arbitrary, this means that *D* halts on all inputs, so *D* is a decider.

To prove that $\mathscr{L}(D) = A_{TM}$, note that *D* accepts ⟨*M*, *w*⟩ iff *H* accepts ⟨*M*, *w*⟩ and *M* accepts *w*. As mentioned above, we know that *H* accepts *w* iff *M* halts on *w*. Therefore, we see that *D* accepts ⟨*M*, *w*⟩ iff *M* halts on *w* and *M* accepts *w*. Note that the statement "*M* halts on *w*" is subsumed by the statement "*M* accepts *w*," so this means that *D* accepts ⟨*M*, *w*⟩ iff *M* accepts *w*. Finally, note that *M* accepts *w* iff $\langle M, w\rangle \in A_{TM}$, so we see that *D* accepts ⟨*M*, *w*⟩ iff $\langle M, w\rangle \in A_{TM}$. Therefore, $\mathscr{L}(D) = A_{TM}$, as required. ■