A Template (for Proving Things Undecidable) and a Worked Example of Using the Template

Lane A. Hemaspaandra, Univ. of Rochester

2021/3/18 (last updated 2021/3/18/331pm)

CSC 280: Computer Models & Limitations Spring 2021

1 Framework

In case you would like a framework in which to (usually) do problems, I'll include in a moment a bit of "framework" text (plus a pointer to a worked example of using that framework). What I mean by that is you may well want to fit all your answers to many problems regarding proving undecidability into the pattern that that framework text sets out.

I'm not guaranteeing that this text will solve every problem in the universe. However, it is sort of the canonical framing of the "contradiction" approach, and it usually does work.

So... here is the framework text. The ??? and ???[comment]??? marks are places where you'll need to fill in text! (You can probably copy-paste this to form the start of your answers when jotting down solutions to proving-something-undecidable challenges, though do be careful that your copy-paste does not garble letters or math symbols. But it is basically plain-text, on purpose, to try to make it easier to copy.)

```
THE TEMPLATE:
[We're trying to prove some particular set, G, is undecidable.]
Our decider for A_TM assuming that the above set G is decidable
by machine R is this machine/program S:
S:
On input <M,w> build the machine M' that itself, when its input is y,
 ???[here you have to describe the program of M', that is, what it does
 on input y]???
Simulate R(???[the argument to R had better match the syntax/type
 of the problem we're attacking!]???), and ???[here you say what
 we do, e.g., "if R accepts we accept and if R rejects we reject"
 or "if R accepts we reject and if R rejects we accept" are the
 most typical things one would have here]???
End of the program S
???[here goes your explanation of why S is a decider for A_TM.  do
NOT forget this part of our template.  it should have 3 subparts:
(1) argue that S is a decider.
(2) argue that if <M,w>\in A_TM then S accepts on input <M,w>.
(3) argue that if <M,w>\not\in A_TM then S rejects on input <M,w>.
NOTE: (2) and (3) actually basically give (1), give or take the issue
of syntactically illegal inputs to M'.  But still, just as an
automatic thing, do include (1), (2), and (3).]???
```

WARNING: The above template assumes one just calls R on one input. There exist examples where one would have to call R lots, and then do more complex things as to how to turn the outcomes from those runs into an accept/reject action. But the above simplified template covers the vast majority of what one would ever face (and of course, even if I tell you to use the template, if the use of it requires an adjustment of the sort I just mentioned, you will need to make such an adjustment).

We will soon do the promised example.

But first a comment. At times (perhaps at WSs or ROHs or *on Midterm II*), you may be asked some pretty hard questions. For example, when building the machine M' of the template above, for some of the problems you might NOT be able to assume that M' ignores its input! Other problems might just be quite different in look and flavor from the problems we've typically been doing. You'll need to really understand how to do these problems—not just to have memorized some examples. Please study and practice (and use office hours) well, until you're pretty comfortably on top of this material.

2 Worked Example

And on we go to the promised example. That is, I will now give a quick example of a problem where M' cannot ignore its input, and will using the template prove that that problem is not decidable.

So, recall that our template is as stated above.

```
Question: Prove that G = { <M> | L(M) is NOT a context-free language}
is undecidable.
Answer: Our decider for A_TM assuming that the above set G is decidable
by machine R is this machine/program S:
S:
On input <M,w> build the machine M' that itself, if its input is y,
  simulates M on input w and if that rejects we reject,
  and if that runs forever we obviously run forever as we simulate it,
  but, crucially, if that accepts, instead of ourselves accepting,
  we instead accept if y is of the form zz for some string z and otherwise
  we reject.
Simulate R(<M'>), and if R accepts we accept and if R rejects we reject.
End of the program S

S is clearly a decider, basically because R is a decider and S just
builds a machine in a doable way and then simulates the decider R on
that and then accepts/rejects in a simple way based on the action of
R.

If <M,w> is in A_TM, then L(M') is {zz | z \in Sigma^Star}, which
is NOT a CFL, and so R will accept [sic] on input <M'>, and so S will
accept on input <M,w>.

If <M,w> \not\in A_TM, then L(M') = \emptyset, which is
not NOT [sic!!!] a CFL (i.e., it is a CFL), and so R will reject [sic]
on input <M'>, and so S will reject.
```