CS 4510 Automata and Complexity 4/4/2022

Lecture 16

Lecturer: Frederic Faulkner *Scribe(s):*

1 An Explicit Undecidable Language

Don't you hate it when an application freezes and you get a "Program is not responsive" popup? The app has probably crashed, but not necessarily; it might just performing a long task, and if you wait it will get back up and running. Should you restart the program or wait? It would be nice if the computer itself told you which one to do, but this is actually not possible (at least not with 100% accuracy) due to the following theorem:

Theorem 1 *The language $HALT_{TM} = \{(\langle M \rangle, x) \mid M$ halts on input $x\}$ is undecidable.*

So there is no Turing machine that can examine another Turing machine and its input and tell you whether the machine will ever stop running.

Proof. As a contradiction, assume that H is a decider for $HALT_{TM}$. We will construct a new Turing machine D that runs H as a subroutine.

On input $\langle M \rangle$, D does the following:

1. 1. Run H on the input $(< M >, < M >)$
2. 2. If H returns "halt", run forever.
3. 3. If H returns "run forever", accept.

This proof can be a little difficult to follow because of all the different Turing machines floating around, so let's keep track of them. There are three machines to worry about:

- H is a decider for $HALT_{TM}$. If you give it a Turing machine and an input, it will tell you whether the machine will halt. Note: we do not care how H works. It is black box for us that we use inside D.
- D is a machine that we create, that follows the algorithm given above. D uses H as a subroutine. The point of the proof is that we will show that D behaves in a contradictory manner; i.e. D cannot exist. But if H exists then D exists; so H cannot exist either.
- M is the input to D. It is not an explicit machine, it is a parameter, like the x in $def\ factorial(x):$. Note that M is never run; M is not guaranteed to be a decider, so we could loop forever if we ran M directly. Instead, we use M as an input ourselves to the black box H.

Another note: in step 1 of the algorithm up above, we ask H whether M will halt on the input... M itself! Or more precisely, on the description of M itself. There is nothing wrong with this. After all, a python program is a just a string; any function that takes a string could be fed its own description as input. (The "print(x)" function could be given a copy of the code for the print function as x, for example.) In a similar way, we can encode M as a binary string, and then use it as an input just like any other binary string.

Final note: the step "run forever" is a valid step in a Turing machine, in the same way that $while(True) : continue$ is a valid program.

Back to the proof. We have just defined the machine D. What does D do on the input $\langle D \rangle$?

Well, first it checks to see what H expects would happen if D were run on $\langle D \rangle$ and then it does the opposite. So if H says "halt", we run forever; and if H says "run forever" than we halt. But H is a decider for $HALT_{TM}$! So D runs forever on $\langle D \rangle$, then H would know that, and similarly if D halts on $\langle D \rangle$. Thus we have the following:

D halts on $\langle D \rangle \longleftrightarrow H(\langle D \rangle, \langle D \rangle)$ returns "halt" $\longleftrightarrow$ D does the opposite of $H \longleftrightarrow D$ runs forever on $\langle D \rangle$.

So D halts on $\langle D \rangle$ if and only if D runs forever on $\langle D \rangle$. This is a contradiction! But the only assumption we made was that there was some decider for $HALT_{TM}$. So that assumption is false and $HALT_{TM}$ is undecidable.

1.1 Halting and Diagonalization

Another way to interpret the proof above is to view it as a diagonalization problem. Consider a table with the left hand side being a list of all the Turing machines M_1, M_2, etc., and the top being the descriptions $\langle M_1 \rangle$, $\langle M_2 \rangle$, etc. Then the cell in the i'th row and j'th column is a "Halt" if the i'th machine accepts the description of the j'th machine.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\langle M_6 \rangle$	...
M_1	No	No	No	Halt	Halt	Halt	
M_2	Halt	Halt	Halt	Halt	Halt	Halt	
M_3	Halt	No	No	No	Halt	No	
M_4	No	No	No	Halt	No	No	
...	...						

D accepts the language formed by doing the opposite of the diagonal.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\langle M_6 \rangle$	...
M_1	**No**	No	No	Halt	Halt	Halt	
M_2	Halt	**Halt**	Halt	Halt	Halt	Halt	
M_3	Halt	No	**No**	No	Halt	No	
M_4	No	No	No	**Halt**	No	No	
...	...						
D?	Halt	No	Halt	No	...		

However, where is D in the table? It is not the case that $D = M_i$ for any i, since D and M_i do different things on the input $\langle M_i \rangle$. So D cannot exist. But we showed how to construct D from H; so if D cannot exist, neither can H.

2 More Theorems on Decidability

Theorem 2 *If L is decidable, then $\overline{L}$ is decidable.*

Proof. Let S be a decider for L. Then the following is a decider for $\overline{L}$:

```
On input $x$:
1. Let $r = S(x)$.
2. Return $!r$.
```

Self-check. Why doesn't this proof also show that the compliment of a recognizable language is recognizable?

Corollary 3 *$\overline{HALT_{TM}}$ is undecidable.*

We can do better than that. It turns out that $\overline{HALT_{TM}}$ is, so to speak, even harder than $HALT_{TM}$. Although we just proved that $HALT_{TM}$ is not decidable, it *is* recognizable as proven by the following recognizer:

```
On input $\langle M \rangle, x$:
1. Simulate $M$ on $x$
2. If $M$ accepts or rejects, accept.
```

However, $\overline{HALT_{TM}}$ cannot even be recognized!

Theorem 4 *If L is recognizable and $\overline{L}$ is recognizable, then L is decidable.*

Proof. Let S be a recognizer for L and S' be a recognizer for $\overline{L}$. Then we can decide L by using a 2-tape Turing machine. First copy the input to both tapes. Then, on tape 1 we simulate S and on tape 2 we simulate S'. If S would accept on tape 1, we accept; if S' would accept on tape 2, we reject. One of the two machines must halt (in fact, one of the two machines must accept), since either $x \in L$ and S accepts or $x \notin L$ and S' accepts. So our machine will never loop forever, and we have created a decider for L.

Corollary 5 *$\overline{HALT_{TM}}$ is not recognizable.*

Proof. Since $HALT_{TM}$ is recognizable, if $\overline{HALT_{TM}}$ were also recognizable then $HALT_{TM}$ would be decidable by the theorem just proved. But it is not decidable. So $\overline{HALT_{TM}}$ is not recognizable.