CS 4510 Automata and Complexity	4/6/2022
Lecture 17	
Lecturer: Frederic Faulkner	*Scribe(s):*

1 More Undecidable Languages

Last class we saw that $HALT$ was undecidable. In other words, there is no algorithmic way of deciding in general whether a TM M will halt on an input x. Of course, you could just run M on x; but if M keeps running, you will never be quite sure if the machine will run forever or is in fact about to finish its computation.

You might suspect that $HALT$ is not the only language which is undecidable, and you would be right! In this lecture we will explore several more undecidable languages.

Theorem 1 *$A_{TM} = \{\langle M\rangle, x \mid M$ accepts $x\}$ is undecidable.*

In other words, there is no foolproof method of simply looking at a TM M and deciding whether it will accept x. Again, you could just run M on x and see what it does; but if M runs forever you will never know whether it accepts x or not.

How can we prove that A_{TM} is undecidable? Well, A_{TM} is very similar to $HALT$, and we could use a proof similar to the proof of $HALT$'s undecidability from last lecture. However, we will not do that for two reasons:

- That proof was complex and hard to generalize.
- It doesn't make use of the fact that we have already proven: that $HALT$ is undecidable.

How can use the undecidability of $HALT$ to say anything about A_{TM}? With a reduction!

1.1 Reducing $HALT$ to A_{TM}

Suppose you had some machine M, and a string x and you wanted to know whether M halted on input x. We already know there is no general procedure to figure this out. However, you have a plan: you create a second machine M'. M' does the following: it erases whatever you put on the tape, and writes x instead. Then, it simulates M. Finally, after it finishes simulating M, it enters the accept state. In other words, M' executes the following algorithm:

M' on input y:

1. run $M(x)$
2. accept y

Notice something important about M': if M (your original machine) halts on x, then M' always accepts whatever you write on its tape. For example, if you write *hello* on the tape of M', in step 1 M' simulates $M(x)$ to completion (remember, we are considering the scenario where M halts on x), and in step two it accepts *hello*. On the other hand, if M runs forever on x, then M' will always get stuck in an infinite loop in step 1, and so it will never enter the accept state. In other words, if you knew definitively whether M' accepted "hello", then, by using the arguments above, *you could then deduce whether M halts on x!*

So now, if we had a procedure to decide whether any machine accepted any string - i.e. a decider for A_{TM}, we could learn whether M' accepts "*hello*", and then we would know whether M halts on x. Notice what we have done here: we have transformed an problem about membership in $HALT$ (i.e. does some M halt on some input x?) into a problem about membership in A_{TM} (i.e. does M' accept "hello"?). But this proves that there is no decider for A_{TM}. Do you see why?

It is because, based on the process we just described, if we had a decider for A_{TM}, we could write a deciding algorithm for $HALT$ like so: Let $R_{A_{TM}}$ be some decider for A_{TM}. In other words, if you give $R_{A_{TM}}$ a machine and a string, it will tell you whether the machine accepts the string. Then we build a decider R_{HALT} as follows:

R_{HALT} on input $\langle M \rangle, x$:

1. Create M' as described above from M and x.
2. Run $R_{A_{TM}}$ to see whether M' accepts "*hello*"
3. If $R_{A_{TM}}$ says that M' accepts "*hello*": return "M halts on x" else: return "M doesn't halt on x"

Now R_{HALT} is a decider for $HALT$, but we know that no such decider can exist, because we have already proven that $HALT$ is undecidable. We have thus constructed an impossible machine, by using $R_{A_{TM}}$, a decider for A_{TM}. So $R_{A_{TM}}$ cannot exist either.

1.2 Reductions

What we have just seen is an example of a *reduction*. If we can convert a problem of type A into a problem of type B, such that the solution to the new type-B problem can be converted back into a solution for the original type-A problem, we say that A can be reduced to B.

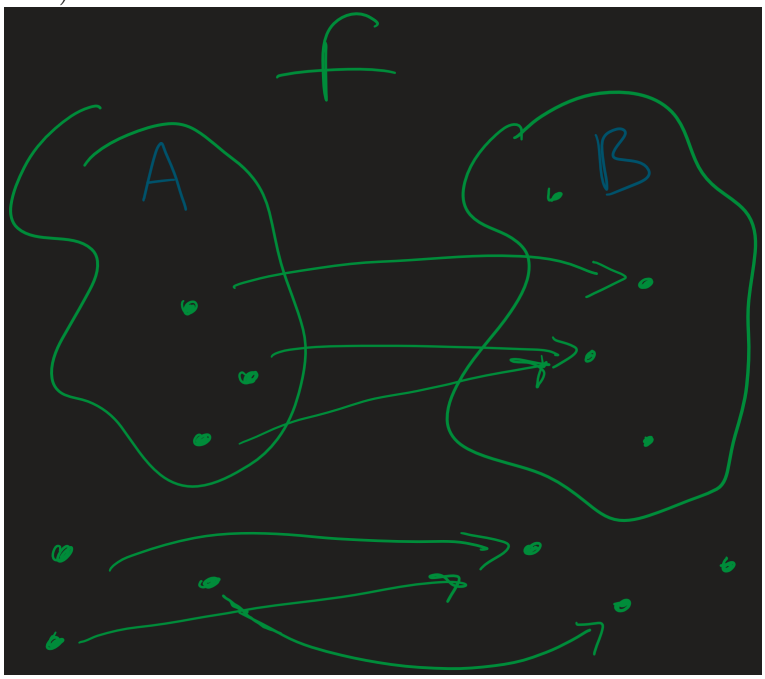
You may already be familiar with reductions from other classes, as they are useful tools for showing that two classes of problem are equally difficult. For example, NP-completeness reductions are used to show that two problems are equally difficult to find a solution for in polynomial time. Above, we used a reduction to show that two different languages, $HALT$ and A_{TM}, are each equally difficult to build a decider for. But we also know that $HALT$ has no deciders at all, from last class; so since $HALT$ and A_{TM} are equivalently difficult to decide, A_{TM} also has no deciders. We now formalize the notion of a reduction:

Definition 1 We say that language A "mapping-reduces" to language B, written $A \leq_m B$, if there is some function $f : \Sigma^* \to \Sigma^*$ with two properties:

- if $x \in A$, $f(x) \in B$
- if $x \notin A$, $f(x) \notin B$.

$\diamondsuit$

In other words, f maps things in A to things in B, and things not in A to things not in B. Below is a conceptual picture of one such f. (Note that f is *not* required to be one-one or onto.)



Theorem 2 *If $A \leq_m B$, and B is decidable, then A is decidable.*

Proof. Suppose $A \leq_m B$ and B is decidable. We would like to show that A is decidable, so we need to construct some decider for it. What resources do we have to build this decider from? Well, we know there is some decider R_B for B, and we know there is some function f that satisfies $x \in A \iff f(x) \in B$. We will use R_B and f to construct a decider for A as follows, completing the proof:

On input x:

1. y = f(x)
2. run R_B on input y
3. if R_B accepts y, accept x; else reject x

This Turing machine is a decider, because R_B is a decider so each step of the algorithm is guaranteed to halt. It is correct, because f maps strings that should be accepted by A to

strings that should be accepted by B, and strings that should be rejected by A to strings that should be rejected by B.

This theorem is okay, but what we really want is the contrapositive:

Corollary 3 *If $A \leq_m B$, and A is undecidable, then B is undecidable.*

Now we have a way to prove that languages are undecidable! If we want to show that some language B is undecidable, we just have to pick some A that we already know is undecidable, and find a mapping reduction f from A to B.

We already basically proved that A_{TM} is undecidable, but now we formalize that proof.

Theorem 4 *A_{TM} is undecidable.*

Proof. We show that $HALT \leq_m A_{TM}$ by constructing the mapping reduction f: $f(\langle M \rangle, x) = \langle M' \rangle$, "*hello*", where M' is defined from M and x as

> M' on input y:
>
> 1. run $M(x)$
> 2. accept y

We now show that f is a mapping reduction, which means we must show two facts:

- $\langle M \rangle, x \in HALT \rightarrow \langle M' \rangle, hello \in A_{TM}$

  If $\langle M \rangle, x \in HALT$, then M halts on input x. But that means that for any input, including the input "*hello*", M' reaches step 2 and accepts.

- $\langle M' \rangle, hello \in A_{TM} \rightarrow \langle M \rangle, x \in HALT$

  If $M', hello \in A_{TM}$, that means that M' accepts "*hello*". How can that happen? Well, that can only happen if M' makes it to step 2. But this can only happen if step 1 terminates, meaning that M eventually terminates on input x.

1.2.1 Technical details

Two small details were left out of the above section. Firstly, the function f used in a mapping reduction must be a *computable* function; in other words, some Turing machine must exist which, when the input x is written on the tape as input, ends its computation with $f(x)$ written on the tape. Note that this notion of a Turing machine as computing a function is distinct from how we discuss them in class, where we consider TM's as deciders/recognizers (yes-no output). However, the two models are closely linked, in the sense that f is a computable function if and only if $\{(x, f(x)) \mid x \in \Sigma^*\}$ is a decidable language.

The other detail we omitted is what f should do in cases where the input is "ill-formed", so to speak. Consider the reduction from $HALT$ to A_{TM}. The elements of $HALT$ are strings which can be interpreted as an ordered pair of a TM description and an input which causes the previously described machine to halt. However, elements not in $HALT$ have

two forms: machines which do not halt on the given input, and strings which do not even contain descriptions of valid Turing machines! We did not say what f does in those cases. It is easy enough to handle however: for any input s which doesn't contain a valid Turing machine description as a substring, $f(s) = \varepsilon$ (or any other string which is not in A_{TM}.) Note that deciding whether s contains a well-formed TM description is a decidable property.

2 One more reduction

Theorem 5 *$ANY_{TM} = \{\langle M\rangle \mid M$ accepts at least one input$\}$ is undecidable.*

Proof. We give a mapping reduction f from a known undecidable language, A_{TM} to ANY_{TM}. Note that f is a function that takes in both a machine description and a string, but it returns *only* a machine description. Why? Because elements of A_{TM} consist of a machine description and an input string, but elements of ANY_{TM} consist only of a machine description.

$f(\langle M\rangle, x) = \langle M'\rangle$ where M' is constructed from M and x as follows:

> M' on input y:
>
> 1. run $M(x)$
> 2. if M accepts x, accept y, else reject y.

Again, we must prove that f is a mapping-reduction, i.e. that $\langle M\rangle, x \in A_{TM} \iff \langle M'\rangle \in ANY_{TM}$.

- $\langle M\rangle, x \in A_{TM} \rightarrow \langle M'\rangle \in ANY_{TM}$

  Suppose M accepts x. Then, in step 1, we learn this, so in step two we accept whatever input we had. In other words, M' will accept everything. But if M' accepts everything, then M' definitely accepts at least one string.

- $\langle M'\rangle \in ANY_{TM} \rightarrow \langle M\rangle, x \in A_{TM}$

  Suppose M' accepts at least one string. Then that must mean that it both reaches step 2, and that in step 1, M accepted x.