COMPSCI 511: Formal Language Theory

Marius Minea

Lecture 16 Notes

Rik Sengupta

March 7, 2019

The lecture today followed Section 5.1 in the textbook closely.

Recall the decidable problems we saw so far.

$$\begin{aligned}
A_{DFA} &= \{\langle B, w\rangle : B \text{ is a DFA that accepts input } w\} \\
E_{DFA} &= \{\langle B\rangle : B \text{ is a DFA with } \mathcal{L}(B) = \varnothing\} \\
A_{REX} &= \{\langle R, w\rangle : R \text{ is a regex that generates the word } w\} \\
EQ_{DFA} &= \{\langle A, B\rangle : A, B \text{ are DFAs satisfying } \mathcal{L}(A) = \mathcal{L}(B)\} \\
A_{CFG} &= \{\langle G, w\rangle : G \text{ is a CFG that generates the word } w\} \\
E_{CFG} &= \{\langle G\rangle : G \text{ is a CFG satisfying } \mathcal{L}(G) = \varnothing\}.
\end{aligned}$$

Of course, we also have the nondeterministic versions A_{NFA} and E_{NFA} decidable as well, using the same arguments. Recall further that we couldn't quite get a decidability proof for EQ_{CFG} similarly to the proof for DFAs, because unlike regular languages, CFLs are *not* closed under intersection, and therefore not under symmetric difference as well. (In fact, somewhat surprisingly, EQ_{CFG} will turn out to be an *undecidable* language later!)

In the last class, we also saw our first two undecidable problems. In some sense, these are the two canonical problems: most reductions you see will be from one of these.

$$\begin{aligned}
A_{TM} &= \{\langle M, w\rangle : M \text{ is a TM that accepts input } w\} \\
HALT_{TM} &= \{\langle M, w\rangle : M \text{ is a TM that halts on input } w\}.
\end{aligned}$$

Now let's look at some other undecidable problems, and show that they are undecidable by means of *reductions*. Roughly speaking, reductions are just what they sound like: you *reduce* a problem A to a problem B (often written as $A \leq B$) if – given an instance of A – you can build an instance of B that agrees with the answer to your instance of A. That is to say, your constructed instance of B outputs **YES** if and only if your original instance of A should. The nature of this reduction depends on the model being used in question. This will come up as a theme throughout the rest of the semester!

Claim 1. *The problem $E_{TM} = \{\langle M\rangle : M$ is a TM with $\mathcal{L}(B) = \varnothing\}$ is undecidable.*

Proof. We reduce from A_{TM}. Suppose, in order to obtain a contradiction, that E_{TM} were decidable, and suppose the TM R decides E_{TM}. How might we use this "black box" machine R to build a decider for A_{TM}?

Build a decider S for A_{TM} as follows: on input $\langle M, w\rangle$, *construct* the description for the following TM M_1: on input x, M_1 checks whether $x = w$, and immediately *rejects* if not; if $x = w$, then M_1 simulates M on w, and behaves the same as M on w (i.e. it *accepts* if M accepts w, *rejects* if M rejects w, and does not halt if M does not either on w). We then feed in the description of this machine M_1 as an input to our presumed decider R, which will either accept or reject. If R accepts, we ask S to **reject**, and if R rejects, we ask S to **accept**.

Why would this be a correct decider for A_{TM}? Observe that we used our inputs M and w and constructed an auxiliary machine M_1, *not* in order to run it on any input, but just in order to obtain a description of it in order to run it as an input to our supposed decider R. What is the purpose of this machine M_1? Well, M_1 rejects all inputs that are not w, and behaves the same as M on the input w. So at most, the only string M_1 can possibly accept is the string w, and even that is only when M accepts it as well. In other words, there are only *two* possibilities for the language recognized by M_1: it is $\varnothing$ precisely when M does not accept w, and it is the singleton set $\{w\}$ precisely when M accepts w. So if we had a way of distinguishing these two possibilities for the language recognized by M_1, we would know whether M would accept w or not. One way of distinguishing the two possibilities is to check whether $\mathcal{L}(M_1) = \varnothing$ or not. But fortunately, we *have* a decider – R – that checks precisely that, and always returns us a yes/no answer! So we have just found a deciding protocol that tells us whether M accepts w or not, and in effect we have built a decider for A_{TM}, which we know from the previous lecture is impossible.

It follows that we must have made a false assumption somewhere. Since the *only* assumption we made was the existence of the decider R, it follows that such an R cannot exist, or in other words, E_{TM} is *undecidable.* □

Claim 2. *The problem $REGULAR_{TM} = \{\langle M\rangle : M$ is a TM; $\mathcal{L}(M)$ is regular$\}$ is undecidable.*

Proof. This is similar in spirit to the last reduction. We reduce from A_{TM}. Suppose, in order to obtain a contradiction, that $REGULAR_{TM}$ were decidable, and suppose the TM R decides it. How might we use this "black box" machine R to build a decider for A_{TM}?

Build a decider S for A_{TM} as follows: on input $\langle M, w\rangle$, *construct* the description for the following TM M_1: on input x, M_1 checks whether x is of the form 0^n1^n for some n, and if so, it immediately *accepts*; if x is not of that form, M_1 simulates M on w, and behaves the same as M on w (i.e. it *accepts* if M accepts w, *rejects* if M rejects w, and does not halt if M does not either on w). We then feed in the description of this machine M_1 as an input to our presumed decider R, which will either accept or reject. If R accepts, we ask S to **accept**, and if R rejects, we ask S to **reject**.

Why would this be a correct decider for A_{TM}? Again we constructed the auxiliary machine M_1, *not* in order to run it on any input, but just in order to obtain a description of it in order to run it as an input to our supposed decider R. You can convince yourself that there are only two possibilities for the language recognized by M_1: M_1 recognizes the nonregular language $\{0^n1^n : n \in \mathbb{N}\}$ precisely when M does not accept w, and M_1 recognizes the regular language Σ^* precisely when M does accept w. Again, it follows that if we had a way of distinguishing these two possibilities for the language recognized by M_1, we would know whether M would accept w or not. The decider R gives us a way of distinguishing this, and gives us a deciding protocol that tells us whether M accepts w or not. So once again, we have built a decider for A_{TM}, which we know from the previous lecture is impossible, proving that $REGULAR_{TM}$ must be undecidable as well. □

Sometimes it makes sense to reduce from other languages known to be undecidable, though in most cases A_{TM} suffices. Here is an example where it is easier to reduce from a different language (though an A_{TM}-reduction is also possible).

Claim 3. *The problem* $EQ_{TM} = \{\langle M_1, M_2\rangle : M_1, M_2$ *are TMs;* $\mathcal{L}(M_1) = \mathcal{L}(M_2)\}$ *is undecidable.*

Proof. We reduce from E_{TM}. Suppose, in order to obtain a contradiction, that EQ_{TM} were decidable, and suppose the TM R decides it. How might we use this "black box" machine R to build a decider for E_{TM}?

Build a decider S for E_{TM} as follows: on input $\langle M\rangle$, just construct the trivial TM $M_\varnothing$ that *rejects* all inputs (this is easy to construct: do you see what it would look like?). The decider S will simply run R on the input $\langle M, M_\varnothing\rangle$. Since R is a decider, it will either accept or reject. If R accepts, it means that the language recognized by M is the same as the language recognized by $M_\varnothing$, which is just $\varnothing$, and is the same as saying M is the empty language. If R rejects, this means the language is not $\varnothing$, and so $\mathcal{L}(M)$ is nonempty. Either way, we have a definitive answer, so S will output the same as what R does. It is important to convince yourself of the validity of this argument. $\square$

We now switch gears for a bit, and talk about computation histories, in the lead-up to discussing LBAs.

Definition 1. *Given a TM M and an input string w, an* accepting computation history *for M on w is a finite sequence*

$$C_1, C_2, \ldots, C_\ell,$$

where each C_i is a valid configuration of M, C_1 is the correct initial configuration of M on the input w, C_ℓ is an accepting configuration, and each C_i follows correctly from C_{i-1} under the transition function of M.

A rejecting computation history *is defined exactly analogously, except that we require C_ℓ to be a rejecting configuration.*

With all this in mind, let's look at yet another model of computation that we have not seen so far in this course.

Definition 2. *A* linear-bounded automaton*, or an LBA for short, is a TM with an additional constraint: the tape head is not allowed to move past the input in* either *direction – left or right.*

You can think about an LBA as essentially the same as a TM, with one critically important difference: limited memory. Just because LBAs have limited memory, though, does not mean they are not a powerful model. Every decider you have encountered in the class so far as been an LBA. However, as you might imagine, the limitation on its memory means that it is quite easy to verify whether some particular string w is accepted by an LBA. We formalize this below.

Claim 4. *The LBA-acceptance problem* $A_{LBA} = \{\langle M, w\rangle : M$ *is an LBA that accepts input* $w\}$ *is decidable.*

Proof. Let M be an LBA with a given input w. We know the tape head is not allowed to move past the left or right end of w, so the entire computation has to happen in the $n := |w|$ cells defined by the input position. How many different configurations are there of M, given this constraint? If M has q states, and the size of the tape alphabet $|\Gamma| = g$, then at any "snapshot" of the computation, we can be in any of the q states, the tape head can be in any of the n positions, and each cell can have any of the g symbols, leading to a total of qng^n possibilities. Because there are only this many distinct possibilities, it means that if M halts on w, it had better do so within qng^n steps; otherwise, we will have repeated a configuration, and then we can never halt, since we will always keep looping. This observation means we have the following decider for A_{LBA}. On input $\langle M, w\rangle$, simulate M on w for qng^n steps, as defined above. If M halts in that time, then output whatever M outputs. If M does not halt, then **reject**. □

Interestingly, however, it turns out that the *emptiness* problem for LBAs is undecidable. If this seems counterintuitive, note that you cannot run the same technique as the other easier emptiness problem deciders, because those essentially use graph search, but here you can't use graph search without having a bound on what a potential input size might be. We have the following claim.

Claim 5. *The LBA-emptiness problem $E_{LBA} = \{\langle M\rangle : M \text{ is an LBA; } \mathcal{L}(M) = \varnothing\}$ is undecidable.*

Proof. As is typical, the reduction is from A_{TM}. We wish for a proof along the same lines. If E_{LBA} were decidable, suppose some TM R decides it. For the purposes of our proof, we will be more interested in R', the TM that decides the *complement* of E_{LBA} (which must exist if R exists). We want to use this to construct a TM S that decides A_{TM} for the contradiction. We want to build S such that, on input $\langle M, w\rangle$, we can construct some LBA M_1 such that R' accepts $\langle M_1\rangle$ if and only if M accepts w. In other words, we want the language of M_1 to be *nonempty* if and only if M accepts w.

What might the language recognized by this M_1 be? We claim that this is precisely the set of *accepting computation histories* of the machine M on the input w. If M accepts w, there is such a valid accepting computation history, so $\mathcal{L}(M_1) \neq \varnothing$. If M does not accept w, similarly, we must have $\mathcal{L}(M_1) = \varnothing$. It only remains for us to show that, given an input to M_1, it can in fact check whether it is a valid accepting computation history for M on w in the space bound it has.

We can easily define M_1: on input x, it first checks whether x is in the right format, namely that x looks like

$$\#C_1\#C_2\#\ldots\#C_\ell\#$$

where the symbol # is a delimiter between adjacent configurations. Next, M_1 verifies that C_1 is the correct start configuration of M on w, i.e. that C_1 looks like $q_0w_1w_2\ldots w_k$, and that C_ℓ has q_{accept} in it. Finally, it zigzags across adjacent configurations C_{i-1} and C_i, and verifies that C_i follows from C_{i-1} using consistent transition rules from M. All of this can be done without the tape head ever moving past the ends of the input, and so M_1 is indeed a valid LBA.

You should convince yourself that this indeed gives you the contradiction you want, namely that it enables you to build a decider for A_{TM}. For specific details as well as minute but important checks, please refer to the textbook. □

Using very similar techniques, we can even prove the following claim, about an interesting language involving CFLs that at first glance has nothing to do with accepting computation histories.

Claim 6. *The CFG-universality problem $ALL_{CFG} = \{\langle G\rangle : G \text{ is a CFG; } \mathcal{L}(G) = \Sigma^*\}$ is undecidable.*

We omit the proof here, and refer to the textbook for the details, including a very important observation on how to represent inputs as alternating between a configuration represented normally and a configuration represented in reverse, to account for the structure of the stack. We do note the following important corollary, which we mentioned briefly at the beginning of class.

Claim 7. *The CFG-equivalence problem* $EQ_{CFG} = \{\langle G_1, G_2\rangle : G_1, G_2 \text{ are CFGs; } \mathcal{L}(G_1) = \mathcal{L}(G_2)\}$ *is undecidable.*

Proof. This follows immediately from the previous claim. If EQ_{CFG} were decidable, say with decider D, then we could use this to build a decider for ALL_{CFG} as follows. On input $\langle G\rangle$, we can construct the following trivial grammar G_{ALL}, defined by the simple rules $\{S \to \sigma S | \varepsilon : \sigma \in \Sigma \cup \varepsilon\}$. Once we do this, we can run D on the input $\langle G, G_{ALL}$. If D accepts, our decider **accepts**; if D rejects, our decider **rejects** as well. Convince yourself that this is a valid reduction, and note that this means we have a contradiction (since ALL_{CFG} is undecidable), proving that EQ_{CFG} must be undecidable as well. □