CSCI3390-Lecture 4: Closure Properties; Some additional computational problems

September 13, 2018

1 Summary

- The classes of Turing-decidable and Turing-recognizable languages are both closed under union and under intersection.
- We discuss a class of problems about string-rewriting that illustrates the distinction between Turing-decidable and Turing-recognizable languages.

2 Closure properties of recognizable and decidable languages.

Theorem 1 *Let Σ be a finite alphabet and let $L_1, L_2 \subseteq \Sigma^*$ be languages.*

- *If L_1 is Turing-decidable, then the complement $\Sigma^* \setminus L_1$ is Turing-decidable.*
- *If L_1, L_2 are both Turing-decidable, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are Turing-decidable.*
- *If L_1, L_2 are both Turing-recognizable, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are Turing-decidable.*

Proof. We already saw how to prove the first item in the previous lecture: If L_1 is decided by a TM $\mathcal{M}_1$, then we get a TM deciding $\Sigma^* \setminus L_1$ just by interchanging the accept and reject states. (This is a low-level proof, because we describe exactly how to modify the Turing machine.)

For the second item, there is a simple high-level argument: If we have (one-tape) machines $\mathcal{M}_1$ and $\mathcal{M}_2$ deciding L_1 and L_2 respectively, then we can create a 2-tape machine that decides $L_1 \cup L_2$. first copies the input on the first tape to the second tape, then runs $\mathcal{M}_1$ on the first tape. If this reaches an accept state of $\mathcal{M}_1$, then we halt and accept. If it reaches a reject state of $\mathcal{M}_1$, then we switch to running $\mathcal{M}_2$ on the second tape, and accept or reject according to whether $\mathcal{M}_2$ accepts or rejects. Note how decidability comes in: The machines $\mathcal{M}_1$ and $\mathcal{M}_2$ are guaranteed to halt in either accept or reject states on every input, which makes this construction work. By the results in the previous lecture, we can now turn this 2-tape machine into a 1-tape machine. The argument for intersection is similar.

This argument does not work for the third item, precisely because we cannot run $\mathcal{M}_1$ 'until it halts', if all we know is that $\mathcal{M}_1$ recognizes L_1: It might not halt at all. Instead we adopt the strategy used in the proof at the end of the preceding lecture: We still use a 2-tape machine, and we still copy the input from the first tape to the second. But now we run the two machines in parallel, doing a step of $\mathcal{M}_1$ on the first tape, then a step of $\mathcal{M}_2$ on the second tape. If either of these two accepts, our 2-tape machine accepts. Once again, the argument for intersection is similar.

3 A string-rewriting problem

We'll consider words over a finite alphabet Σ. You're given the following input:

- A finite collection of pairs of words $v \mapsto v'$. We'll call these *substitution rules*.
- A pair w_1, w_2 of words, which we'll call the *start word* and the *target word*, respectively.

If $w, w' \in \Sigma^*$ then we write $w \Rightarrow w'$ if you can find the left-hand side of a rule within w, and w' results by replacing this by the right-hand side of the rule. That is,

$$w = xvy, w' = x'v'y,$$

where $v \mapsto v'$ is a substitution rule.

For example, if $ab \mapsto bba$ is a rule, then $aabbab \Rightarrow abbabab$, and also $aabbab \Rightarrow aabbbba$. The problem is to determine whether the target word can be derived from the start word in a finite sequence of steps.

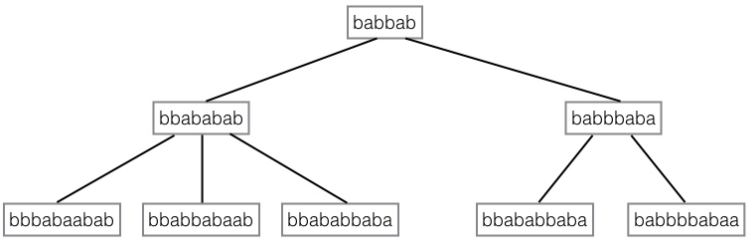

Figure 1: The first few levels of the tree of words derived from $babbab$ using the rule $ab \mapsto baba$.

This is a decision problem. Is it Turing-recognizable? Is it decidable?

First let's look at a special restricted case of the problem. Suppose all the rules $v \mapsto v'$ have the property that $|v| < |v'|$, in other words, all the rules are length-increasing. Then we can proceed as follows. We will illustrate this with the example of a single rule $ab \mapsto baba$ with start word $babbab$, but the argument is the same for any finite collection of length-increasing rules. Beginning from the start word, we can construct, layer by layer, the tree of all words that can be derived from the start word. The first few levels are illustrated in Figure 1.

Let us suppose that the target word has length 20. Since the source word has length 6, and since the length of the word grows by 2 with each application of the substitution rule, any word of length 20 that can be derived from the source word will appear in the seventh generation. Thus this problem (for length-increasing rules) is *decidable:* The algorithm for deciding it is to extend this tree, layer by layer, until no more words of length less than the target word appear at the leaves. If the target word appears in the tree, then the answer is 'yes', otherwise 'no'.

Now suppose that instead of length-increasing rules, all the rules keep the length the same: For example, suppose the substitution rules are $abab \mapsto bbab, baba \mapsto abaa$. If we are given a source and a target word, then obviously the answer is 'no' if the words do not have the same length. If they do have the same length, then we can proceed as above, constructing the tree of words derivable from the source word—but when can we stop?

Consider this: Let us suppose that source and target words both have length 8, and that there is a very long derivation, *e.g.,*

$$abbababb \Rightarrow ababaabb \Rightarrow \cdots \Rightarrow baabbbab.$$

If this sequence of words contains more than $2^8 = 256$ elements, then some word must appear twice in the derivation sequence. Thus there is a *shorter* derivation, because we can simply cut out the loop. So now we have a decision algorithm: If the source and the target word both have length n, construct the tree of derivable words to a maximum depth of 2^n. If the target word ever appears, stop and answer 'Yes'. If the target word does not appear in the first 2^n levels, answer 'No'.

We can combine the two arguments above to similarly bound the depth of the tree when every rule is either length-increasing or length-preserving.

What happens in the general case, where substitution rules can either increase, decrease, or preserve the lengths? Here the only thing we can say at the outset is that the decision problem is Turing-recognizable: The algorithm constructs the tree of derivable words layer by layer, and if we ever encounter the target word, we halt and answer 'Yes'. But it is not clear when we can answer 'No'. In fact we will see later that the general problem is not decidable. There is no way, given the substitution rules, and the source and target words, that we can *a priori* bound the length of the derivation.