University of California, San Diego
Department of Computer Science and Engineering
Course: CSE 105 – Theory of Computability
2004 Summer Session I
July 21, 2004

Instructor: Adriana Palacio
TAs: Ragesh Jaiswal
Vadim Lyubashevsky
Saurabh Panjwani

Problem Set 2 Solutions

1. Exercise 2.1.

The parse trees are in Figure 1.

a. $E \Rightarrow T \Rightarrow F \Rightarrow \mathtt{a}$

b. $E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow \mathtt{a}+T \Rightarrow \mathtt{a}+F \Rightarrow \mathtt{a}+\mathtt{a}$.

c. $E \Rightarrow E+T \Rightarrow E+T+T \Rightarrow T+T+T \Rightarrow F+T+T \Rightarrow \mathtt{a}+T+T \Rightarrow \mathtt{a}+F+T \Rightarrow \mathtt{a}+\mathtt{a}+T \Rightarrow \mathtt{a}+\mathtt{a}+F \Rightarrow \mathtt{a}+\mathtt{a}+\mathtt{a}$

d. $E \Rightarrow T \Rightarrow F \Rightarrow (E) \Rightarrow (T) \Rightarrow (F) \Rightarrow ((E)) \Rightarrow ((T)) \Rightarrow ((F)) \Rightarrow ((\mathtt{a}))$.

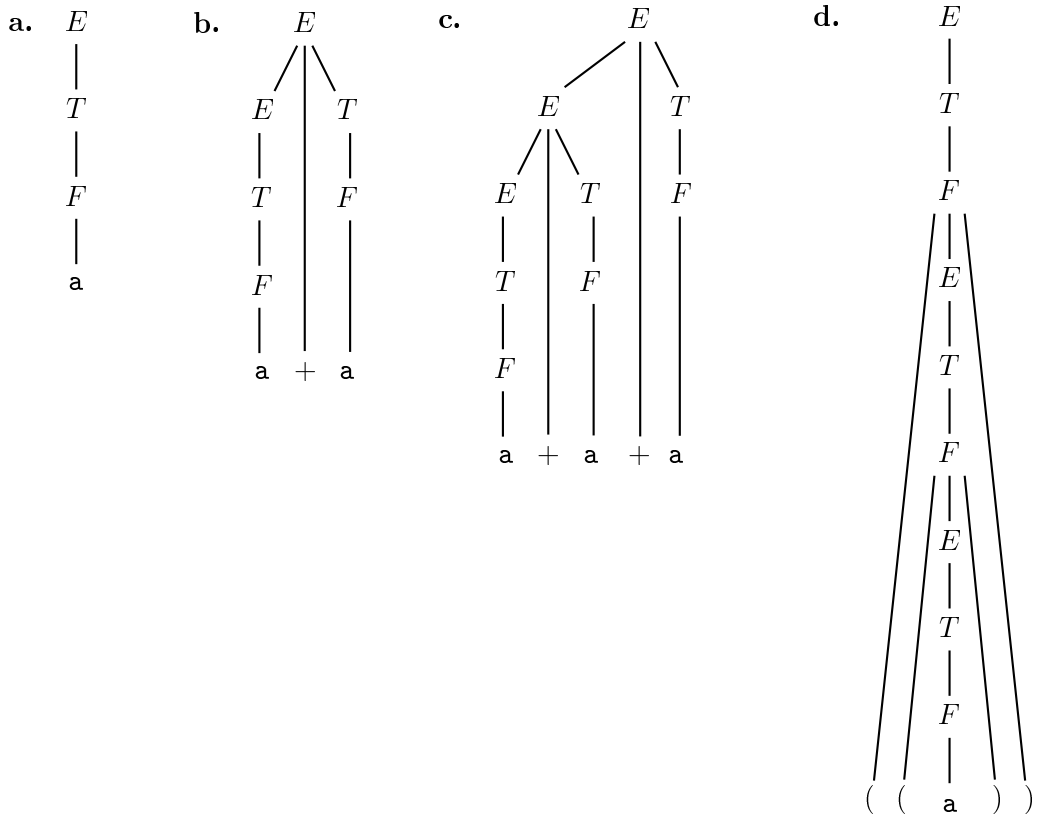

Figure 1: Parse trees for Exercise 2.1 **a**, **b**, **c**, **d**.

2. Exercise 2.3 a, b, ... , m.

a. The variables are $\{R, S, T, X\}$. The terminals are $\{\mathtt{a}, \mathtt{b}\}$. The start variable is R.

b. $\{\mathtt{ab}, \mathtt{ba}, \mathtt{aab}\}$

c. $\{\epsilon, \mathtt{aa}, \mathtt{ba}\}$

d. False

e. True

f. False

g. True

h. True

i. False

j. True

k. True

l. False

m. All strings over $\{\mathtt{a}, \mathtt{b}\}$ that are not palindromes.

---

3. Exercise 2.4 b, f.

b. Let $L = \{\, w \mid w$ starts and ends with the same symbol $\}$. Let $G = (V, \Sigma, R, S)$ where

- $V = \{S, T\}$ (there are two variables, one of which is the start variable S)
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ (the problem says to assume this for all parts)
- The set R of rules contains the following eight rules:

$$\begin{aligned} S &\rightarrow \mathtt{0}T\mathtt{0} \mid \mathtt{1}T\mathtt{1} \mid \mathtt{0} \mid \mathtt{1} \\ T &\rightarrow TT \mid \mathtt{0} \mid \mathtt{1} \mid \varepsilon \end{aligned}$$

Grammar G generates L.

Comment: *From the variable T, one can derive any string in Σ^*. Therefore, from S one can derive any string whose first and last symbols are the same. Note that the first and last symbols of the string* `0` *are the same, and likewise for the string* `1`*. Therefore, these must be derivable. On the other hand, the empty string has no symbols and thus its first and last symbols are not the same and it should not be derivable.*

When you provide grammars in your solutions, use the above template for the specification. Do not merely list the rules; you must also say what is the set V of variables, the set Σ of terminals, and the start variable, and say that the grammar is the 4-tuple of these.

f. Let $L = \{\, w \mid w = w^{\mathcal{R}},$ that is, w is a palindrome $\}$. Let $G = (V, \Sigma, R, S)$ where

- $V = \{S\}$ (there is only one variable, namely the start variable S)

- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ (the problem says to assume this for all parts)
- The set R of rules contains the following five rules:

$$S \;\rightarrow\; \mathtt{0}S\mathtt{0} \mid \mathtt{1}S\mathtt{1} \mid \mathtt{0} \mid \mathtt{1} \mid \varepsilon$$

Grammar G generates L.

---

4. Exercise 2.6 b, c.

b. Let $L = \overline{\{\, \mathtt{a}^n\mathtt{b}^n \mid n \geq 0 \,\}}$. Let $G = (V, \Sigma, R, S)$ where

- $V = \{S, S_1, S_2, S_3, T\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- The set R of rules contains the following 14 rules:

$$\begin{aligned} S &\rightarrow S_1 \mid S_2 \mid S_3 \\ S_1 &\rightarrow \mathtt{a} \mid \mathtt{a}S_1 \mid \mathtt{a}S_1\mathtt{b} \\ S_2 &\rightarrow \mathtt{b} \mid S_2\mathtt{b} \mid \mathtt{a}S_2\mathtt{b} \\ S_3 &\rightarrow T\mathtt{ba}T \\ T &\rightarrow TT \mid \mathtt{a} \mid \mathtt{b} \mid \varepsilon \end{aligned}$$

Grammar G generates L.

Comment: *We want the grammar to generate all strings $w \in \{\mathtt{a}, \mathtt{b}\}^*$* ***not*** *consisting of some number of* a*s followed by the same number of* b*s. To simplify the task, we break it up into parts depending on the structure of w. We note that w is not of the form $\mathtt{a}^n\mathtt{b}^n$ if and only if one of the following is true: (1) w is of the form $\mathtt{a}^i\mathtt{b}^j$ for some $i > j \geq 0$, or (2) w is of the form $\mathtt{a}^i\mathtt{b}^j$ for some $j > i \geq 0$, or (3) w contains* ba *as a substring. In other words, the language in question is the union of the three languages L_1, L_2, L_3, where*

(1) $L_1 = \{\, \mathtt{a}^i\mathtt{b}^j \mid i > j \geq 0 \,\}$

(2) $L_2 = \{\, \mathtt{a}^i\mathtt{b}^j \mid j > i \geq 0 \,\}$

(3) $L_3 = \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid \mathtt{ba} \text{ is a substring of } w \,\}$.

The grammar G above was designed so that

$$\begin{aligned} \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid S_1 \Rightarrow^* w \,\} &= L_1 \\ \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid S_2 \Rightarrow^* w \,\} &= L_2 \\ \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid S_3 \Rightarrow^* w \,\} &= L_3\,. \end{aligned}$$

The first rule of the grammar G implies that $L(G) = L_1 \cup L_2 \cup L_3$.

We remark that the high-level idea of the construction follows the idea behind the proof of the closure of the class of CFLs under union given in lecture. Also notice that the last rule of this grammar is very similar to the last rule of the grammar of Problem 3. **b.** *above, and has the same functionality, namely to generate any string. This illustrates how one can reuse pieces of known grammars in building new ones.*

c. Let $L = \{\, w\#x \mid w^{\mathcal{R}}$ is a substring of x for $w, x \in \{\mathtt{0},\mathtt{1}\}^* \,\}$. Let $G = (V, \Sigma, R, S)$ where

- $V = \{S, T, U\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}, \#\}$
- The set R of rules contains the following 8 rules:

$$\begin{array}{lcl} S & \rightarrow & TU \\ T & \rightarrow & \mathtt{0}T\mathtt{0} \mid \mathtt{1}T\mathtt{1} \mid \#U \\ U & \rightarrow & UU \mid \mathtt{0} \mid \mathtt{1} \mid \varepsilon \end{array}$$

Grammar G generates L.

Comment: *We want the grammar to generate all strings $s \in \{\mathtt{0},\mathtt{1},\#\}^*$ such that $s = w\#uw^{\mathcal{R}}v$ for some $w, u, v \in \{\mathtt{0},\mathtt{1}\}^*$. From variable U one can derive any string in $\{\mathtt{0},\mathtt{1}\}^*$. Observe that the rule $X \rightarrow \mathtt{0}X\mathtt{0} \mid \mathtt{1}X\mathtt{1} \mid \varepsilon$ allows to derive all strings in $\{\, ww^{\mathcal{R}} \mid w \in \{\mathtt{0},\mathtt{1}\}^* \,\}$. Thus from variable T one can derive all strings of the form $w\#uw^{\mathcal{R}}$, where $w, u \in \{\mathtt{0},\mathtt{1}\}^*$. From variable S one can then derive all strings of the form $w\#uw^{\mathcal{R}}v$, where $w, u, v \in \{\mathtt{0},\mathtt{1}\}^*$.*

---

5. Exercise 2.8

The first derivation of the sentence is as follows:

$$\begin{array}{rcl} \langle SENTENCE\rangle & \Rightarrow & \langle NOUN-PHRASE\rangle\langle VERB-PHRASE\rangle \\ & \Rightarrow & \langle CMPLX-NOUN\rangle\langle VERB-PHRASE\rangle \\ & \Rightarrow & \langle ARTICLE\rangle\langle NOUN\rangle\langle VERB-PHRASE\rangle \\ & \Rightarrow & \texttt{the}\langle NOUN\rangle\langle VERB-PHRASE\rangle \\ & \Rightarrow & \texttt{the girl}\langle VERB-PHRASE\rangle \\ & \Rightarrow & \texttt{the girl}\langle CMPLX-VERB\rangle \\ & \Rightarrow & \texttt{the girl}\langle VERB\rangle\langle NOUN-PHRASE\rangle \\ & \Rightarrow & \texttt{the girl touches}\langle NOUN-PHRASE\rangle \\ & \Rightarrow & \texttt{the girl touches}\langle CMPLX-NOUN\rangle\langle PREP-PHRASE\rangle \\ & \Rightarrow & \texttt{the girl touches}\langle ARTICLE\rangle\langle NOUN\rangle\langle PREP-PHRASE\rangle \\ & \Rightarrow & \texttt{the girl touches the}\langle NOUN\rangle\langle PREP-PHRASE\rangle \\ & \Rightarrow & \texttt{the girl touches the boy}\langle PREP-PHRASE\rangle \\ & \Rightarrow & \texttt{the girl touches the boy}\langle PREP\rangle\langle CMPLX-NOUN\rangle \\ & \Rightarrow & \texttt{the girl touches the boy with}\langle CMPLX-NOUN\rangle \\ & \Rightarrow & \texttt{the girl touches the boy with}\langle ARTICLE\rangle\langle NOUN\rangle \\ & \Rightarrow & \texttt{the girl touches the boy with the}\langle NOUN\rangle \\ & \Rightarrow & \texttt{the girl touches the boy with the flower} \end{array}$$

This derivation corresponds to the interpretation that the flower is something that the "boy" has (for example, the boy could be holding the flower in his hand). How the girl touches him (with her hand, with her elbow, with another flower etc) is unspecified by this interpretation.

The second derivation works as follows:

$$
\begin{aligned}
\langle SENTENCE\rangle &\Rightarrow \langle NOUN-PHRASE\rangle\langle VERB-PHRASE\rangle \\
&\Rightarrow \langle CMPLX-NOUN\rangle\langle VERB-PHRASE\rangle \\
&\Rightarrow \langle ARTICLE\rangle\langle NOUN\rangle\langle VERB-PHRASE\rangle \\
&\Rightarrow \texttt{the}\langle NOUN\rangle\langle VERB-PHRASE\rangle \\
&\Rightarrow \texttt{the girl}\langle VERB-PHRASE\rangle \\
&\Rightarrow \texttt{the girl}\langle CMPLX-VERB\rangle\langle PREP-PHRASE\rangle \\
&\Rightarrow \texttt{the girl}\langle VERB\rangle\langle NOUN-PHRASE\rangle\langle PREP-PHRASE\rangle \\
&\Rightarrow \texttt{the girl touches}\langle NOUN-PHRASE\rangle\langle PREP-PHRASE\rangle \\
&\Rightarrow \texttt{the girl touches}\langle CMPLX-NOUN\rangle\langle PREP-PHRASE\rangle \\
&\Rightarrow \texttt{the girl touches}\langle ARTICLE\rangle\langle NOUN\rangle\langle PREP-PHRASE\rangle \\
&\Rightarrow \texttt{the girl touches the}\langle NOUN\rangle\langle PREP-PHRASE\rangle \\
&\Rightarrow \texttt{the girl touches the boy}\langle PREP-PHRASE\rangle \\
&\Rightarrow \texttt{the girl touches the boy}\langle PREP\rangle\langle CMPLX-NOUN\rangle \\
&\Rightarrow \texttt{the girl touches the boy with}\langle CMPLX-NOUN\rangle \\
&\Rightarrow \texttt{the girl touches the boy with}\langle ARTICLE\rangle\langle NOUN\rangle \\
&\Rightarrow \texttt{the girl touches the boy with the}\langle NOUN\rangle \\
&\Rightarrow \texttt{the girl touches the boy with the flower}
\end{aligned}
$$

This derivation corresponds to the interpretation that the flower is something that the "girl" is using to carry out her action (i.e., to touch the boy).

---

6. Problem 2.25

Comment: *The solution for this problem has been written in discrete steps that could be followed in proceeding towards the solution. Our recommendation is that a reader who has only partially succeeded in solving this problem should view each step as a hint and try to get the complete solution hint by hint.*

(1) Your first step should be to figure out the kind of strings that can be derived from the variable Y alone (this is because the strings that are contained in the language are related to whatever can be derived from Y). If you spend some time thinking, you should be able to see that *every* possible string in the set $\{\texttt{a},\texttt{b}\}^*$ (including ε) can be derived from Y (try writing out the strings that can be obtained by applying the rules for Y and you'll see this).

(2) Using the above observation, figure out the kind of strings that can be derived from S. Look

at the simpler rules first. Any string starting in a $\mathtt{b}$ can be derived from S (using the rule $S \rightarrow \mathtt{b}Y$). Any string ending in an $\mathtt{a}$ can be derived from S (using the rule $S \rightarrow Y\mathtt{a}$). What are the strings in $\{\mathtt{a}, \mathtt{b}\}^*$ that are not covered by these two kinds of strings ? The strings which both start in an $\mathtt{a}$ and end in a $\mathtt{b}$ and the string ε. We conclude that the language $L(G)$ must *at least* contain any string which is not of the form $\mathtt{a}x\mathtt{b}$ (where $x \in \{\mathtt{a}, \mathtt{b}\}^*$). Also observe that the string ε cannot be derived from S by the application of any of the 3 rules for S.

(3) But hey! Look at the first rule for S. It says $S \rightarrow \mathtt{a}S\mathtt{b}$, meaning that some strings starting in an $\mathtt{a}$ and ending in a $\mathtt{b}$ *can* be derived from S. But can all such strings in $\{\mathtt{a}, \mathtt{b}\}^*$ be derived from it? That in turn depends on what S can derive. We saw in step (2) above that any string that isn't ε and that doesn't start in an $\mathtt{a}$ and end in a $\mathtt{b}$ can surely be derived from S. The rule $S \rightarrow \mathtt{a}S\mathtt{b}$ gives us the impression that many strings beginning with an $\mathtt{a}$ and ending in a $\mathtt{b}$ can also be derived from S. Which ones can't be derived? Well, can you derive the string $\mathtt{ab}$ from S? NO. Since we can't derive the empty string from S, we can't use the rule $S \rightarrow \mathtt{a}S\mathtt{b}$ in deriving the string $\mathtt{ab}$. Similarly, can you derive the string $\mathtt{aabb}$ from S? NO. Since we can't derive $\mathtt{ab}$ from S, we can't use the rule $S \rightarrow \mathtt{a}S\mathtt{b}$ in order to derive $\mathtt{aabb}$.

(4) Applying this argument recursively, we conclude that any string that is formed by concatenating a sequence of $\mathtt{a}$'s with a sequence of $\mathtt{b}$'s of *equal length* cannot be derived from S. This means that $L(G) = \{\mathtt{a}, \mathtt{b}\}^* \setminus \{\, \mathtt{a}^n\mathtt{b}^n \mid n \geq 0 \,\}$. Note that this notation captures the fact that ε is not in the language.

(5) What is the complement of $L(G)$? The language $\{\, \mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$. We've seen this (and similar) language(s) many times in class. The grammar for it is simple and the rules are as follows:

$$S \rightarrow \mathtt{a}S\mathtt{b} \mid \varepsilon$$

---

7. Prove that the class of context-free languages is closed under the concatenation operation.

Given: Languages L_1 and L_2 are context-free. Therefore, there is a CFG $G_1 = (V_1, \Sigma_1, R_1, S_1)$ that generates L_1 and a CFG $G_2 = (V_2, \Sigma_2, R_2, S_2)$ that generates L_2.

Want: To construct a CFG $G = (V, \Sigma, R, S)$ that generates the language $L_1 \cdot L_2$, showing that $L_1 \cdot L_2$ is a context-free language.

Construction: Grammar $G = (V, \Sigma, R, S)$ is defined as follows:

- $V = V_1 \cup V_2 \cup \{S\}$ where S is a new variable
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- $R = R_1 \cup R_2 \cup \{S \rightarrow S_1S_2\}$ (the set of rules contains the rules of G_1, the rules of G_2, and one new rule, namely the rule $S \rightarrow S_1S_2$)

In this construction, we assume that $V_1 \cap V_2 = \emptyset$, meaning that the two given grammars have no variables in common. This can always be made true by renaming variables in one of the grammars, if necessary.

Correctness: We claim that $L(G) = L(G_1) \cdot L(G_2)$. (Since $L(G_1) = L_1$ and $L(G_2) = L_2$ this means that $L(G) = L_1 \cdot L_2$, so we are done.) In other words, we claim that $S \Rightarrow^* w$ in G if and

only if w has the form w_1w_2 for some w_1, w_2 such that $S_1 \Rightarrow^* w_1$ in G_1 and $S_2 \Rightarrow^* w_2$ in G_2. This is true because the first rule applied in a derivation of any w via G is $S \rightarrow S_1S_2$, and after that, the only way the derivation can proceed is to expand S_1 according to G_1 and S_2 according to G_2. The fact that $V_1 \cap V_2 = \emptyset$ is important to ensure that a derivation starting from, say, S_1, can only use rules in G_1 and not use rules in G_2, and vice-versa.

---

8. Problem 2.16

First we will show how to convert a regular expression R directly to an equivalent CFG G. Then we will use this to give another proof that every regular language is a CFL.

- $R = s$ for s in the alphabet Σ
  $G = (\{S\}, \Sigma, \{S \rightarrow s\}, S)$
- $R = \varepsilon$
  $G = (\{S\}, \Sigma, \{S \rightarrow \varepsilon\}, S)$
- $R = \emptyset$
  $G = (\{S\}, \Sigma, \{S \rightarrow S\}, S)$ OR $G = (\{S\}, \Sigma, \emptyset, S)$
- $(R_A \cup R_B)$ for reg exps R_A, R_B
  If $L(G_1) = L(R_A)$, where $G_1 = (V_1, \Sigma, R_1, S_1)$, and $L(G_2) = L(R_B)$, where $G_2 = (V_2, \Sigma, R_2, S_2)$ and $V_1 \cap V_2 = \emptyset$, then
  $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, R_1 \cup R_2 \cup \{S \rightarrow S_1 \mid S_2\}, S)$
- $(R_A \cdot R_B)$ for reg exps R_A, R_B
  If $L(G_1) = L(R_A)$, where $G_1 = (V_1, \Sigma, R_1, S_1)$, and $L(G_2) = L(R_B)$, where $G_2 = (V_2, \Sigma, R_2, S_2)$ and $V_1 \cap V_2 = \emptyset$, then
  $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, R_1 \cup R_2 \cup \{S \rightarrow S_1S_2\}, S)$
- (R_A^*) for reg exp R_A
  If $L(G_1) = L(R_A)$, where $G_1 = (V_1, \Sigma, R_1, S_1)$, then
  $G = (V_1 \cup \{S\}, \Sigma, R_1 \cup \{S \rightarrow SS \mid S_1 \mid \varepsilon\}, S)$

Let L be a regular language. Then there exists a regular expression R such that $L(R) = L$. Let G be a CFG equivalent to R, constructed as shown above. Then $L(G) = L(R) = L$. Thus, there is a CFG that generates L, i.e., L is a CFL.

---

9. Problem 2.18 a, c.

a. We use a template similar to the one we used to prove that a language is nonregular via the Pumping Lemma for regular languages. You should use this template in your own solutions.

The proof is by contradiction.

Assume that $L = \{ \mathtt{0}^n\mathtt{1}^n\mathtt{0}^n\mathtt{1}^n \mid n \geq 0 \}$ is a CFL. This assumption means that the Pumping Lemma for CFLs (Theorem 2.19, page 115 of the textbook) applies to L. Let p be the pumping length given by the P.L. Let $s = \mathtt{0}^p\mathtt{1}^p\mathtt{0}^p\mathtt{1}^p$. Then $s \in L$ and $|s| \geq p$. By the P.L., s can be

split into $uvxyz$ such that the following conditions hold:

(1) $|vy| > 0$

(2) $|vxy| \leq p$

(3) For all $i \geq 0$: $uv^ixy^iz \in L$

Comment: *Recall that we can choose s, but we have no control over u, v, x, y, z. We know, however, that they satisfy the three conditions. Now our goal is to show that conditions* (1) *and* (2) *imply that for some $i \geq 0$: uv^ixy^iz is not in L, contradicting condition* (3) *above. This means our assumption that L was a CFL is false.*

Since we are not given information about the length of u, we do not know where vxy lies in the string $s = uvxyz = \mathtt{0}^p\mathtt{1}^p\mathtt{0}^p\mathtt{1}^p$. We have to consider several possibilities.

We know that $s = uvxyz = \mathtt{0}^p\mathtt{1}^p\mathtt{0}^p\mathtt{1}^p$. Condition (2) says that the length of vxy is at most p. This means that vxy is a substring of one of the following: 1) the first $\mathtt{0}^p\mathtt{1}^p$ segment, or 2) $\mathtt{1}^p\mathtt{0}^p$, or 3) the last $\mathtt{0}^p\mathtt{1}^p$ segment. We now consider these three cases one by one, and in each case show that $uv^2xy^2z \notin L$.

Case 1) If vxy is a substring of the first $\mathtt{0}^p\mathtt{1}^p$ segment of $s = uvxyz = \mathtt{0}^p\mathtt{1}^p\mathtt{0}^p\mathtt{1}^p$, then by condition (1) (which says that vy has non-zero length), uv^2xy^2z has the form $w\mathtt{0}^p\mathtt{1}^p$, where w has more than p 0s or more than p 1s. In either case, $w \neq \mathtt{0}^p\mathtt{1}^p$ and hence $uv^2xy^2z \notin L$.

Case 2) If vxy is a substring of $\mathtt{1}^p\mathtt{0}^p$, then, by condition (1), uv^2xy^2z has the form $\mathtt{0}^pw\mathtt{1}^p$, where w has more than p 0s or more than p 1s. In either case, $w \neq \mathtt{1}^p\mathtt{0}^p$ so $uv^2xy^2z \notin L$.

Case 3) The last case is that vxy is a substring of the second $\mathtt{0}^p\mathtt{1}^p$ segment of $s = uvxyz = \mathtt{0}^p\mathtt{1}^p\mathtt{0}^p\mathtt{1}^p$. By condition (1), uv^2xy^2z has the form $\mathtt{0}^p\mathtt{1}^pw$, where w has more than p 0s or more than p 1s. In either case, $w \neq 0^p1^p$ and hence $uv^2xy^2z \notin L$.

c. The proof is by contradiction.

Assume that $L = \{\ w\#x \mid w \text{ is a substring of } x, \text{ where } w, x \in \{\mathtt{a}, \mathtt{b}\}^*\ \}$ is a CFL. Then the Pumping Lemma for CFLs (Theorem 2.19, page 115 of the textbook) applies to L. Let p be the pumping length given by the P.L. Let $s = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. Then $s \in L$ and $|s| \geq p$. By the P.L., s can be split into $uvxyz$ such that the following conditions hold:

(1) $|vy| > 0$

(2) $|vxy| \leq p$

(3) For all $i \geq 0$: $uv^ixy^iz \in L$

Comment: *Recall that we can choose s, but we have no control over u, v, x, y, z. We know, however, that they satisfy the three conditions. Now our goal is to show that conditions* (1) *and* (2) *imply that for some $i \geq 0$: uv^ixy^iz is not in L, contradicting condition* (3) *above. This means our assumption that L was a CFL is false.*

Since we are not given information about the length of u, we do not know where vxy lies in the string $s = uvxyz = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. We have to consider several possibilities.

We know that $s = uvxyz = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. Condition (2) says that the length of vxy is at most

p. This means that one of the following is true: either 1) vxy is a substring of the first $\mathtt{a}^p\mathtt{b}^p$ segment of s, or 2) vxy contains #, or 3) vxy is a substring of the last $\mathtt{a}^p\mathtt{b}^p$ segment of s. We now consider these three cases one by one. For the first case, we show that $uv^2xy^2z \notin L$. For the second case, we show that $uxz \notin L$ or $uv^2xy^2z \notin L$. For the third case, we show that $uxz \notin L$.

Comment: *Notice that this argument is a little more involved than the ones we used in class and the one in part* **a.** *above. The first and third cases are standard (and somewhat similar to the first and third cases in the proof discussed in class for the non-CFL language* $\{ ww \mid w \in \{\mathtt{0},\mathtt{1}\}^* \}$*). But, for the second case, we show that* $uxz \notin L$ **or** $uv^2xy^2z \notin L$*. This means that for all cases there is* **some** $i \geq 0$ *such that* uv^ixy^iz *is not in* L*. (For the first case,* $i = 2$*. For the second case, either* $i = 0$ *or* $i = 2$*. For the third case,* $i = 0$*.) Unlike previous problems, for this one we cannot always say exactly which* i *will yield a string that is not in* L*. But we show that in every case there is an* $i \geq 0$ *that yields such a string. This is enough to contradict condition* (3) *above.*

Case 1) The first case is that vxy is a substring of the first $\mathtt{a}^p\mathtt{b}^p$ segment of $s = uvxyz = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. Condition (1) says that vy has non-zero length. This means that uv^2xy^2z has the form $w\#\mathtt{a}^p\mathtt{b}^p$, where w has more than p a's or more than p b's. In either case, w is not a substring of $\mathtt{a}^p\mathtt{b}^p$ and hence $uv^2xy^2z \notin L$.

Case 2) The second case is that vxy contains #. If v or y contain # then uv^2xy^2z has more than one #, so $uv^2xy^2z \notin L$. If neither v nor y contain #, then x must contain #. By condition (1), $v \neq \varepsilon$ or $y \neq \varepsilon$. If $v \neq \varepsilon$, then, by condition (2), v consists of one or more b's and y consists of zero or more a's. Therefore, uv^2xy^2z has the form $w\#\mathtt{a}^{p+j}\mathtt{b}^p$, where $j \geq 0$ and w has more than p b's. Thus w is not a substring of $\mathtt{a}^{p+j}\mathtt{b}^p$, and hence $uv^2xy^2z \notin L$. If $v = \varepsilon$, then, by Conditions (1) and (2), y consists of one or more a's. Therefore, uxz has the form $\mathtt{a}^p\mathtt{b}^p\#w$ where w has less than p a's. Thus $\mathtt{a}^p\mathtt{b}^p$ is not a substring of w and hence $uxz \notin L$. This shows that if vxy contains #, then $uxz \notin L$ or $uv^2xy^2z \notin L$.

Case 3) The last case is that vxy is a substring of the second $\mathtt{a}^p\mathtt{b}^p$ segment of $s = uvxyz = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. By condition (1), uxz has the form $\mathtt{a}^p\mathtt{b}^p\#w$, where w has less than p a's or less than p b's. In either case, $\mathtt{a}^p\mathtt{b}^p$ is not a substring of w and hence $uxz \notin L$.

---

10. Problem 2.26

Let $C = \{ x\#y \mid x, y \in \{\mathtt{0},\mathtt{1}\}^* \text{ and } x \neq y \}$. We show that C is a CFL by constructing a PDA that recognizes C. The state diagram of this PDA is in Figure 2.

Comment: *Notice that to conclude that strings* x *and* y *are different, it is sufficient to show that their lengths are different or find one position in which* x *and* y *differ, whereas to conclude that the two strings are equal, it is necessary to check that all symbols in* x *and* y *are identical. PDAs cannot test for equality of strings. They can, however, test for inequality. The PDA we construct to recognize* C *guesses whether to check if strings* x *and* y *have different lengths, or to first guess a position in which these strings differ and then check that the symbols in this position of* x *and* y *are indeed different. To keep track of the position, the PDA pushes some symbol, say* a *onto the stack while it reads its input until it makes the guess. It then reads the following symbols in the input*

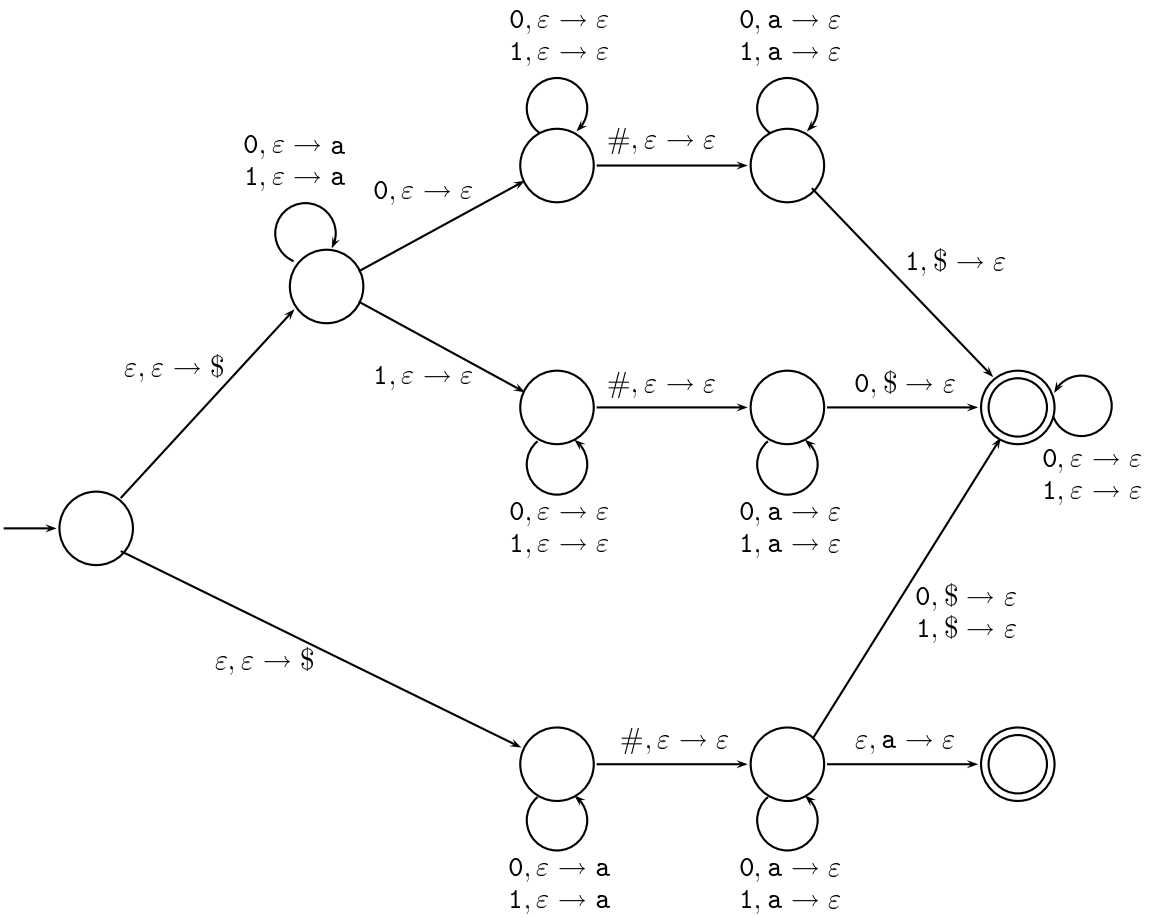

Figure 2: PDA for Problem 2.26.

---

string until it reaches the symbol #. Then it pops a*'s from the stack while it continues to read its input, until the stack is empty. If the symbol read when the marker* $ *is popped is different from the symbol read when the PDA made its guess, then the machine accepts. When the PDA makes a guess to check if strings* x *and* y *have different lengths, it pushes* a*'s onto the stack while it reads its input until it reaches the symbol #. Then it pops* a*'s from the stack while it continues to read its input, until the stack is empty or the input is exhausted. If the input has not been exhausted when the marker* $ *is popped (meaning* $|y| > |x|$*), then the PDA accepts. If the input is exhausted before the stack is empty (meaning* $|x| > |y|$*), then the PDA also accepts. Notice that we use the symbol* a *to emphasize that the PDA is only keeping track of the number of symbols and not the symbols themselves. We could have just used, say* 1*, instead. Notice that if the input string does not contain exactly one occurrence of the symbol #, then the PDA will reject. If the input string is of the form* $x\#y$*, where* $x, y \in \{0,1\}^*$ *and* $|x| \neq |y|$*, then the PDA will accept. If the input string is of the form* $x\#y$*, where* $x, y \in \{0,1\}^*$ *and* $|x| = |y|$*, then the PDA will accept if and only if there is some position where* x *and* y *differ (i.e., iff* $x \neq y$*).*

---

16. Exercise 3.5

a. Can a Turing machine ever write the blank symbol on its tape?

YES. By the definition of the transition function, a TM can write any symbol from the tape alphabet to the tape. The blank symbol is part of the tape alphabet (although it is not part of the input alphabet).

b. Can the tape alphabet Γ be the same as the input alphabet Σ?

NO. The tape alphabet always contains the blank symbol, but the input alphabet cannot contain this symbol. If the blank symbol were part of the input alphabet, the TM would never know when the input actually ends!

c. Can a Turing machine's read head *ever* be in the same location in two successive steps?

YES. The only time this can happen is when the TM is on the first tape square and it tries to move left. It will stay in place instead of falling off the tape.

d. Can a Turing machine contain just a single state?

NO. A TM must contain at least two states: an accept state and a reject state. Notice that since being in either of these states halts the computation, a different start state is necessary in order for the TM to read any input whatsoever.

---

17. Exercise 3.7

M_{bad} = "The input is a polynomial p over variables $x_1, \ldots, x_k$.

1. Try all possible settings of $x_1, \ldots, x_k$ to integer values.
2. Evaluate p on all of these settings.
3. If any of these settings evaluates to 0, *accept*; otherwise, *reject*."

This description has several problems. The loop structure is too vague to really specify what it does. Does the machine first write all possible values of the variables to the tape and then evaluate each one of them? Or does it produce some values of the variables and evaluate that assignment before considering another assignment? The most serious problem is in the last step. There is no condition when the machine will reject. The machine cannot try all possible assignments of the variables, because there are an infinite number of them.

Note: If you are careful about how the loop is structured, you can make a TM that *recognizes* the language thet this machine attempts to decide.

---

18. Problem 3.14 a, b, c, d, e.

Comment: *For simplicity, in each of these problems we consider languages over the same alphabet Σ. All of the proofs can be modified to account for languages over distinct alphabets.*

a. Prove that the class of decidable languages is closed under union.

Given: Decidable languages L_1 and L_2 over some alphabet Σ. Since L_1 is decidable, there is a TM M_1 that decides it, i.e., there is a TM M_1 such that for every $w \in \Sigma^*$:

$w \in L_1 \implies M_1(w)$ *accepts*
$w \notin L_1 \implies M_1(w)$ *rejects*

Likewise, there is a TM M_2 such that for every $x \in \Sigma^*$:

$x \in L_2 \implies M_2(x)$ *accepts*
$x \notin L_2 \implies M_2(x)$ *rejects*

Want: A TM M that decides $L_1 \cup L_2$. (This would mean that $L_1 \cup L_2$ is decidable, and thus that the class of decidable languages is closed under union.) M must be such that for every $y \in \Sigma^*$:

$y \in L_1 \cup L_2 \implies M(y)$ *accepts*
$y \notin L_1 \cup L_2 \implies M(y)$ *rejects*

Construction:

$M =$ On input a string y
Run $M_1(y)$; If it accepts then *accept* EndIf
Run $M_2(y)$; If it accepts then *accept* EndIf
Reject

Correctness:

If $y \in L_1 \cup L_2$, then $y \in L_1$ OR $y \in L_2$. If $y \in L_1$ then $M_1(y)$ accepts and hence $M(y)$ accepts. If $y \in L_2$ then $M_2(y)$ accepts and hence $M(y)$ accepts.

If $y \notin L_1 \cup L_2$, then $y \notin L_1$ AND $y \notin L_2$. Therefore, $M_1(y)$ rejects and $M_2(y)$ rejects, and hence $M(y)$ rejects.

b. Prove that the class of decidable languages is closed under concatenation.

Given: Decidable languages L_1 and L_2 over some alphabet Σ. Since L_1 is decidable, there is a TM M_1 that decides it, i.e., there is a TM M_1 such that for every $w \in \Sigma^*$:

$w \in L_1 \implies M_1(w)$ *accepts*
$w \notin L_1 \implies M_1(w)$ *rejects*

Likewise, there is a TM M_2 such that for every $x \in \Sigma^*$:

$x \in L_2 \implies M_2(x)$ *accepts*
$x \notin L_2 \implies M_2(x)$ *rejects*

Want: A TM M that decides $L_1 \cdot L_2$. (This would mean that $L_1 \cdot L_2$ is decidable, and thus that the class of decidable languages is closed under concatenation.) M must be such that for every $y \in \Sigma^*$:

$y \in L_1 \cdot L_2 \implies M(y)$ *accepts*
$y \notin L_1 \cdot L_2 \implies M(y)$ *rejects*

Construction:

$M =$ On input a string y
$n \leftarrow |y|$

For $i = 1, \ldots, n$ do
Run $M_1(y[1] \cdots y[i])$
Run $M_2(y[i+1] \cdots y[n])$
If **both** of these computations accept then *accept* EndIf
EndFor
Reject

Comment: *This can also be written as follows (for example):*

$M =$ On input a string y
For each pair of strings $w, x \in \Sigma^*$ such that $y = wx$ do
Run $M_1(w)$
Run $M_2(x)$
If **both** of these computations accept then *accept* EndIf
EndFor
Reject

Comment: *Observe that the number of pairs of strings $w, x \in \Sigma^*$ such that $y = wx$ is finite. Thus the computation of M on any input halts.*

Correctness:

If $y \in L_1 \cdot L_2$, then there exist $w \in L_1$ and $x \in L_2$ such that $y = wx$. Since M checks all pairs $w, x \in \Sigma^*$ for which $y = wx$, it will find a pair such that $w \in L_1$ and $x \in L_2$. Thus it will find a pair such that $M_1(w)$ accepts and $M_2(x)$ accepts. Therefore, $M(y)$ accepts.

If $y \notin L_1 \cdot L_2$, then there do not exist $w \in L_1$ and $x \in L_2$ such that $y = wx$. Therefore, for all pairs $w, x \in \Sigma^*$ that M checks, $w \notin L_1$ or $x \notin L_2$. Thus for all pairs $w, x \in \Sigma^*$ that M checks, $M_1(w)$ rejects or $M_2(x)$ rejects, and hence $M(y)$ rejects.

c. Prove that the class of decidable languages is closed under star.

Given: A decidable language L over some alphabet Σ. Since L is decidable, there is a TM M that decides it, i.e., there is a TM M such that for every $w \in \Sigma^*$:

$$w \in L \implies M(w) \quad \textit{accepts}$$
$$w \notin L \implies M(w) \quad \textit{rejects}$$

Want: A TM M' that decides L^*. (This would mean that L^* is decidable, and thus that the class of decidable languages is closed under star.) M' must be such that for every $y \in \Sigma^*$:

$$y \in L^* \implies M'(y) \quad \textit{accepts}$$
$$y \notin L^* \implies M'(y) \quad \textit{rejects}$$

Construction:

$M' =$ On input a string y
If $y = \varepsilon$ then *accept* EndIf
$n \leftarrow |y|$
For $k = 1, \ldots, n$ do
For each sequence of strings $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ such that $y = w_1 \cdots w_k$ do
Run $M(w_i)$ for $i = 1, \ldots, k$

If **all** of these computations accept then *accept*
EndFor
EndFor
Reject

Comment: *Observe that the number of sequences of strings $w_1, \ldots, w_k \in \Sigma^*$ such that $y = w_1 \cdots w_k$ is finite. Thus the computation of M' on any input halts.*

Correctness:

If $y \in L^*$, then $y = \varepsilon$ or there exist $w_1, \ldots, w_k \in L \setminus \{\varepsilon\}$ such that $y = w_1 \cdots w_k$. If $y = \varepsilon$, then $M'(y)$ accepts by construction. If $y \neq \varepsilon$, then since M' checks all sequences $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ for which $y = w_1 \cdots w_k$, it will find a sequence such that $w_1, \ldots, w_k \in L \setminus \{\varepsilon\}$. Since M' is a decider for L, for $i = 1, \ldots, k$, $M(w_i)$ accepts. Therefore, $M'(y)$ accepts.

If $y \notin L^*$, then $y \neq \varepsilon$ and there does not exist a sequence $w_1, \ldots, w_k \in L \setminus \{\varepsilon\}$ such that $y = w_1 \cdots w_k$. Therefore, for all sequences $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ that M checks, there is some $i \in \{1, \ldots, k\}$ such that $w_i \notin L$. Since M is a decider for L, for all sequences $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ that M checks, there is some $i \in \{1, \ldots, k\}$ such that $M(w_i)$ rejects. Therefore, $M(y)$ rejects.

d. Prove that the class of decidable languages is closed under complementation.

Given: A decidable language L over some alphabet Σ. Since L is decidable, there is a TM M that decides it, i.e., there is a TM M such that for every $w \in \Sigma^*$:

$$w \in L \implies M(w) \quad \textit{accepts}$$
$$w \notin L \implies M(w) \quad \textit{rejects}$$

Want: A TM M' that decides $\overline{L}$. (This would mean that $\overline{L}$ is decidable, and thus that the class of decidable languages is closed under complementation.) M' must be such that for every $y \in \Sigma^*$:

$$y \in \overline{L} \implies M'(y) \quad \textit{accepts}$$
$$y \notin \overline{L} \implies M'(y) \quad \textit{rejects}$$

Construction:

M' = On input a string y
Run $M(y)$
If it accepts then *reject* else *accept* EndIf

Correctness:

If $y \in \overline{L}$, then $y \notin L$. Therefore, $M(y)$ rejects and hence $M'(y)$ accepts.

If $y \notin \overline{L}$, then $y \in L$. Therefore, $M(y)$ accepts and $M'(y)$ rejects.

e. Prove that the class of decidable languages is closed under intersection.

Given: Decidable languages L_1 and L_2 over some alphabet Σ. Since L_1 is decidable, there is a TM M_1 that decides it, i.e., there is a TM M_1 such that for every $w \in \Sigma^*$:

$w \in L_1 \implies M_1(w)$ *accepts*
$w \notin L_1 \implies M_1(w)$ *rejects*

Likewise, there is a TM M_2 such that for every $x \in \Sigma^*$:

$x \in L_2 \implies M_2(x)$ *accepts*
$x \notin L_2 \implies M_2(x)$ *rejects*

Want: A TM M that decides $L_1 \cap L_2$. (This would mean that $L_1 \cap L_2$ is decidable, and thus that the class of decidable languages is closed under intersection.) M must be such that for every $y \in \Sigma^*$:

$y \in L_1 \cap L_2 \implies M(y)$ *accepts*
$y \notin L_1 \cap L_2 \implies M(y)$ *rejects*

Construction:

$M =$ On input a string y
Run $M_1(y)$
Run $M_2(y)$
If **both** accept then *accept* else *reject* EndIf

Correctness:

If $y \in L_1 \cap L_2$, then $y \in L_1$ AND $y \in L_2$. Therefore, $M_1(y)$ accepts and $M_2(y)$ accepts, and hence $M(y)$ accepts.

If $y \notin L_1 \cap L_2$, then $y \notin L_1$ OR $y \notin L_2$. Therefore, $M_1(y)$ rejects OR $M_2(y)$ rejects, and hence $M(y)$ rejects.

---

19. Problem 3.15 a, b, c, d.

Comment: *For simplicity, in each of these problems we consider languages over the same alphabet Σ. All of the proofs can be modified to account for languages over distinct alphabets.*

a. Prove that the class of recognizable languages is closed under union.

Given: Recognizable languages L_1 and L_2 over some alphabet Σ. Since L_1 is recognizable, there is a TM M_1 that recognizes it, i.e., there is a TM M_1 such that for every $w \in \Sigma^*$:

$w \in L_1 \implies M_1(w)$ *accepts*
$w \notin L_1 \implies M_1(w)$ *rejects* or *does not halt*

Likewise, there is a TM M_2 such that for every $x \in \Sigma^*$:

$x \in L_2 \implies M_2(x)$ *accepts*
$x \notin L_2 \implies M_2(x)$ *rejects* or *does not halt*

Want: A TM M that recognizes $L_1 \cup L_2$. (This would mean that $L_1 \cup L_2$ is recognizable, and thus that the class of recognizable languages is closed under union.) M must be such that for every $y \in \Sigma^*$:

$y \in L_1 \cup L_2 \implies M(y)$ *accepts*

$y \notin L_1 \cup L_2 \implies M(y)$ *rejects* or *does not halt*

Construction:

$M =$ On input a string y
Run $M_1(y)$ and $M_2(y)$ in parallel
If either accepts then *accept* EndIf
Reject

Comment: *Notice that if M runs $M_1(y)$ before running $M_2(y)$ and M_1 does not halt on input y, then even if $M_2(y)$ accepts, machine M will not halt. This is why it is necessary to run M_1 and M_2 in parallel.*

Correctness:

If $y \in L_1 \cup L_2$, then $y \in L_1$ OR $y \in L_2$. If $y \in L_1$ then $M_1(y)$ accepts and hence $M(y)$ accepts. If $y \in L_2$ then $M_2(y)$ accepts and hence $M(y)$ accepts.

If $y \notin L_1 \cup L_2$, then $y \notin L_1$ AND $y \notin L_2$. Therefore, $M_1(y)$ rejects or does not halt and $M_2(y)$ rejects or does not halt. Hence $M(y)$ either rejects or does not halt.

b. Prove that the class of recognizable languages is closed under concatenation.

Given: Recognizable languages L_1 and L_2 over some alphabet Σ. Since L_1 is recognizable, there is a TM M_1 that recognizes it, i.e., there is a TM M_1 such that for every $w \in \Sigma^*$:

$w \in L_1 \implies M_1(w)$ *accepts*

$w \notin L_1 \implies M_1(w)$ *rejects* or *does not halt*

Likewise, there is a TM M_2 such that for every $x \in \Sigma^*$:

$x \in L_2 \implies M_2(x)$ *accepts*

$x \notin L_2 \implies M_2(x)$ *rejects* or *does not halt*

Want: A TM M that recognizes $L_1 \cdot L_2$. (This would mean that $L_1 \cdot L_2$ is recognizable, and thus that the class of recognizable languages is closed under concatenation.) M must be such that for every $y \in \Sigma^*$:

$y \in L_1 \cdot L_2 \implies M(y)$ *accepts*

$y \notin L_1 \cdot L_2 \implies M(y)$ *rejects* or *does not halt*

Construction:

$M =$ On input a string y
$n \leftarrow |y|$; numsteps $\leftarrow 1$
While $(0 = 0)$ do
For each pair of strings $w, x \in \Sigma^*$ such that $y = wx$ do
Run $M_1(w)$ for numsteps steps
Run $M_2(x)$ for numsteps steps
If **both** of these computations have accepted then *accept* EndIf
EndFor

numsteps $\leftarrow$ numsteps $+ 1$
EndWhile

Correctness:

If $y \in L_1 \cdot L_2$, then there exist $w \in L_1$ and $x \in L_2$ such that $y = wx$. M considers all pairs $w, x \in \Sigma^*$ for which $y = wx$. For each of them, it runs M_1 on input w and M_2 on input x for numsteps steps to see if they both accept. Eventually, numsteps becomes large enough that for some pair $w, x \in \Sigma^*$ such that $y = wx$, $M_1(w)$ and $M_2(x)$ both accept within numsteps steps. At that point, M accepts.

If $y \notin L_1 \cdot L_2$, then there do not exist $w \in L_1$ and $x \in L_2$ such that $y = wx$. Therefore, for all pairs $w, x \in \Sigma^*$ that M checks, $w \notin L_1$ or $x \notin L_2$. Thus for all pairs $w, x \in \Sigma^*$ that M checks, regardless of the value of numsteps, $M_1(w)$ does not accept (i.e., it rejects or does not halt) or $M_2(x)$ does not accept (i.e., it rejects or does not halt). Therefore, $M(y)$ does not halt.

Comment: *Notice that M never rejects. It accepts if $y \in L_1 \cdot L_2$ and it does not halt if $y \notin L_1 \cdot L_2$. That is ok because we just want a machine that recognizes $L_1 \cdot L_2$. The formal definition of M includes a reject state, but M never enters this state.*

c. Prove that the class of recognizable languages is closed under star.

Given: A recognizable language L over some alphabet Σ. Since L is recognizable, there is a TM M that recognizes it, i.e., there is a TM M such that for every $w \in \Sigma^*$:

$w \in L \implies M(w)$ *accepts*
$w \notin L \implies M(w)$ *rejects* or *does not halt*

Want: A TM M' that recognizes L^*. (This would mean that L^* is recognizable, and thus that the class of recognizable languages is closed under star.) M' must be such that for every $y \in \Sigma^*$

$y \in L^* \implies M'(y)$ *accepts*
$y \notin L^* \implies M'(y)$ *rejects* or *does not halt*

Construction:

$M' =$ On input a string y
If $y = \varepsilon$ then *accept* EndIf
$n \leftarrow |y|$; numsteps $\leftarrow 1$; acc $\leftarrow 1$
While $(0 = 0)$ do
For $k = 1, \ldots, n$ do
For each sequence of strings $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ such that $y = w_1 \cdots w_k$ do
acc $\leftarrow 1$
For $i = 1, \ldots, k$ do
Run $M(w_i)$ for numsteps steps
If this computation has not accepted then acc $\leftarrow 0$ EndIf
EndFor
If acc $= 1$ then *accept*
EndFor
EndFor
numsteps $\leftarrow$ numsteps $+ 1$

EndWhile

Correctness:

If $y \in L^*$, then $y = \varepsilon$ or there exist $w_1, \ldots, w_k \in L \setminus \{\varepsilon\}$ such that $y = w_1 \cdots w_k$. If $y = \varepsilon$, then $M'(y)$ accepts by construction. If $y \neq \varepsilon$, then M' considers all sequences $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ for which $y = w_1 \cdots w_k$. For each of them, it runs M on each w_i for numsteps steps to see if M accepts all of these inputs. Eventually, numsteps becomes large enough that for some sequence $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ such that $y = w_1 \cdots w_k$, for $i = 1, \ldots, k$, $M(w_i)$ accepts within numsteps steps. At that point, M accepts.

If $y \notin L^*$, then $y \neq \varepsilon$ and there does not exist a sequence $w_1, \ldots, w_k \in L \setminus \{\varepsilon\}$ such that $y = w_1 \cdots w_k$. Therefore, for all sequences $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ that M checks, there is some $i \in \{1, \ldots, k\}$ such that $w_i \notin L$. Thus for all sequences $w_1, \ldots, w_k \in \Sigma^* \setminus \{\varepsilon\}$ that M checks, regardless of the value of numsteps, there is some $i \in \{1, \ldots, k\}$ such that $M(w_i)$ does not accept (i.e., it rejects or does not halt). Therefore, $M(y)$ does not halt.

d. Prove that the class of recognizable languages is closed under intersection.

Given: Recognizable languages L_1 and L_2 over some alphabet Σ. Since L_1 is recognizable, there is a TM M_1 that recognizes it, i.e., there is a TM M_1 such that for every $w \in \Sigma^*$:

$w \in L_1 \implies M_1(w)$ *accepts*
$w \notin L_1 \implies M_1(w)$ *rejects* or *does not halt*

Likewise, there is a TM M_2 such that for every $x \in \Sigma^*$:

$x \in L_2 \implies M_2(x)$ *accepts*
$x \notin L_2 \implies M_2(x)$ *rejects* or *does not halt*

Want: A TM M that recognizes $L_1 \cap L_2$. (This would mean that $L_1 \cap L_2$ is recognizable, and thus that the class of recognizable languages is closed under intersection.) M must be such that for every $y \in \Sigma^*$:

$y \in L_1 \cap L_2 \implies M(y)$ *accepts*
$y \notin L_1 \cap L_2 \implies M(y)$ *rejects* or *does not halt*

Construction:

$M =$ On input a string y
Run $M_1(y)$
If it accepts then
Run $M_2(y)$
If it accepts then *accept* EndIf
EndIf
Reject

Correctness:

If $y \in L_1 \cap L_2$, then $y \in L_1$ AND $y \in L_2$. Therefore, $M_1(y)$ accepts and $M_2(y)$ accepts, and hence $M(y)$ accepts.

If $y \notin L_1 \cap L_2$, then $y \notin L_1$ OR $y \notin L_2$. Therefore, $M_1(y)$ rejects or does not halt OR $M_2(y)$

rejects or does not halt, and hence $M(y)$ rejects or does not halt.

---

20. Exercise 4.1

a. Is $\langle M, \mathtt{0100} \rangle \in A_{\mathsf{DFA}}$?

YES. DFA M accepts $\mathtt{0100}$.

b. Is $\langle M, \mathtt{011} \rangle \in A_{\mathsf{DFA}}$?

NO. DFA M rejects $\mathtt{011}$.

c. Is $\langle M \rangle \in A_{\mathsf{DFA}}$?

NO. $\langle M \rangle$ is not a valid encoding of a DFA and a string.

d. Is $\langle M, \mathtt{0100} \rangle \in A_{\mathsf{REX}}$?

NO. $\langle M, \mathtt{0100} \rangle$ is not a valid encoding of a regular expression and a string.

e. Is $\langle M \rangle \in E_{\mathsf{DFA}}$?

NO. $L(M) \neq \emptyset$ (e.g., M accepts $\mathtt{0100}$)

f. Is $\langle M, M \rangle \in EQ_{\mathsf{DFA}}$?

YES. $L(M) = L(M)$.

---

21. Exercise 4.2

The problem can be expressed as the following language:

$EQ_{\mathsf{DR}} = \{ \langle B, E \rangle \mid B \text{ is a DFA equivalent to regular expression } E \}$

We prove that EQ_{DR} is decidable as follows.

Want: A TM M that decides EQ_{DR}, i.e., such that for all DFAs B and all regular expressions E:

$$\begin{array}{lcll} L(B) = L(E) & \Longrightarrow & M(\langle B, E \rangle) & \textit{accepts} \\ L(B) \neq L(E) & \Longrightarrow & M(\langle B, E \rangle) & \textit{rejects} \end{array}$$

Construction: Here we exploit a fact already proved about decidability. Theorem 4.5 says that the following language is decidable:

$$EQ_{\mathsf{DFA}} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}$$

Let M_{ED} be a TM that decides EQ_{DFA}. The following TM decides EQ_{DR}.

$M =$ On input $\langle B, E \rangle$
Convert E into an equivalent DFA C
Run $M_{\mathsf{ED}}(\langle B, C \rangle)$
If it accepts then *accept* else *reject*

Comment: *Note that we follow Sipser's convention of not including explicitly in the description of a TM the verification of the format of the input. If M is given an input that isn't a proper encoding of a DFA and a regular expression, it rejects.*

Correctness:

If $L(E) = L(B)$ (i.e., $\langle B, E \rangle \in EQ_{\mathsf{DR}}$), then $L(C) = L(E) = L(B)$. Hence $M_{\mathsf{ED}}(\langle B, C \rangle)$ accepts and M accepts.

If $L(E) \neq L(B)$ (i.e., $\langle B, E \rangle \notin EQ_{\mathsf{DR}}$), then $L(C) = L(E) \neq L(B)$. Hence $M_{\mathsf{ED}}(\langle B, C \rangle)$ rejects and M rejects.

Thus M decides EQ_{DR}.

---

22. Exercise 4.4

Consider the language $A\varepsilon_{\mathsf{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG that generates } \varepsilon \}$.

Want: A TM M that decides $A\varepsilon_{\mathsf{CFG}}$, i.e., such that for all CFGs G:

$$\begin{array}{lcll} G \text{ generates } \varepsilon & \Longrightarrow & M(\langle G \rangle) & \textit{accepts} \\ G \text{ does not generate } \varepsilon & \Longrightarrow & M(\langle G \rangle) & \textit{rejects} \end{array}$$

Construction: Theorem 4.6 says that the following language is decidable:

$$A_{\mathsf{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates } w \}$$

Let M_{CFG} be a TM that decides A_{CFG}. The following TM decides $A\varepsilon_{\mathsf{CFG}}$.

$M =$ On input $\langle G \rangle$
Run $M_{\mathsf{CFG}}(\langle G, \varepsilon \rangle)$
If it accepts then *accept* else *reject*

Correctness:

If G generates ε (i.e., $\langle G \rangle \in A\varepsilon_{\mathsf{CFG}}$), then $M_{\mathsf{CFG}}(\langle G, \varepsilon \rangle)$ accepts and M accepts.

If G does not generate ε (i.e., $\langle G \rangle \notin A\varepsilon_{\mathsf{CFG}}$), then $M_{\mathsf{CFG}}(\langle G, \varepsilon \rangle)$ rejects and M rejects.

Thus M decides $A\varepsilon_{\mathsf{CFG}}$.

---

23. Exercise 4.5

Consider the language $\mathit{INFINITE}_{\mathsf{DFA}} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) \text{ is an infinite language} \}$.

Want: A TM M that decides $\mathit{INFINITE}_{\mathsf{DFA}}$, i.e., such that for all DFAs A:

$$\begin{array}{lcll} L(A) \text{ is an infinite language} & \Longrightarrow & M(\langle A \rangle) & \textit{accepts} \\ L(A) \text{ is a finite language} & \Longrightarrow & M(\langle A \rangle) & \textit{rejects} \end{array}$$

Construction: First we make a claim. Then we construct a TM M that decides $\mathit{INFINITE}_{\mathsf{DFA}}$ based on this claim. Finally, we prove the claim.

Claim: A regular language L is infinite if and only if the DFA that recognizes L has some state which, through a series of transitions, can reach itself and an accept state, and can also be reached from the start state. (i.e., the DFA that recognizes L has a loop reachable from the start state that can reach some accept state).

The following TM decides $\mathit{INFINITE}_{\mathsf{DFA}}$.

M = On input $\langle A \rangle$
    For every state s of A do
        Clear all marks
        Mark all states that have transitions coming into them from s
        Repeat until no new state gets marked
            Mark any state that has a transition coming into it from any state that is already marked
            If state s and some accept state got marked then
                Clear all marks
                Mark the start state
                Repeat until no new state gets marked
                    Mark any state that has a transition coming into it from any state that is already marked
                    If state s got marked, then *accept* EndIf
                EndRepeat
            EndIf
        EndRepeat
    EndFor
    Reject

What the above algorithm does is, for every state, it checks whether the state can reach itself through one or more transitions and can reach an accept state. If so, then it lies on a loop that can reach the accept state. We then need to check whether this state can be reached from the start state. If so, then we accept because we found a loop that can be reached from the start state and can reach the accept state. If, after we go through all the states, none of them satisfy the above conditions, we conclude that the language recognized by DFA A is finite, and we reject.

Correctness: We need to prove the above claim.

($\Leftarrow$) If the DFA has a loop that can be reached from the start state and can reach an accept state, then we can just cycle through the loop as many times as we want, and get an arbitrarily large string recognized by the DFA.

($\Rightarrow$) Say that a state is *pointless* it cannot be reached from the start state or cannot reach an accept state. Let's remove all these *pointless* states and all the transitions associated with them. Notice that we are now left with an NFA (it could still be a DFA, but not necessarily) that recognizes the same language as our initial DFA and all the states of the NFA can reach an accept state and are reachble from the start state. Say this new NFA has p states. If the NFA has no loops, then it cannot possibly accept a string of length greater than p. (Because accepting a string of length greater than p would mean visiting some state twice, but that would require a loop!) Thus if this NFA has no loops, then the language it accepts is finite. And the Claim is proved.

---

24. Problem 4.10

Consider the language

$A = \{ \langle M \rangle \mid M$ is a DFA which doesn't accept any string containing an odd number of `1`s $\}$.

Want: A TM M_A that decides A, i.e., such that for all DFAs M:

M doesn't accept any string containing an odd number of 1s $\implies$ $M_A(\langle M \rangle)$ *accepts*
M accepts some string containing an odd number of 1s $\implies$ $M_A(\langle M \rangle)$ *rejects*

Construction: Theorem 4.4 says that the following language is decidable:

$$E_{\mathsf{DFA}} = \{ \langle B \rangle \mid B \text{ is a DFA and } L(B) = \emptyset \}$$

Let M_{E} be a TM that decides E_{DFA}.

Note that the language

$$B = \{ w \in \{0,1\}^* \mid w \text{ contains an odd number of 1s} \}$$

is regular. Let D_B be a DFA that recognizes B.

Let T be a TM that on input $\langle M_1, M_2 \rangle$, where M_1 and M_2 are DFAs, returns $\langle N \rangle$, where N is a DFA such that $L(N) = L(M_1) \cap L(M_2)$. Notice that this TM exists since the transformation underlying the proof that the class of regular languages is closed under intersection can be automated.

The following TM decides A.

M_A = On input $\langle M \rangle$
Let $\langle N \rangle$ be the output of $T(\langle M, D_B \rangle)$
Run $M_{\mathsf{E}}(\langle N \rangle)$
If it accepts then *accept* else *reject*

Correctness:

If M is a DFA that doesn't accept any string containing an odd number of 1s (i.e., $\langle M \rangle \in A$), then $L(M) \cap L(D_B) = \emptyset$. Hence $M_{\mathsf{E}}(T(\langle M, D_B \rangle))$ accepts and M_A accepts.

If M is a DFA that accepts some string containing an odd number of 1s (i.e., $\langle M \rangle \notin A$), then $L(M) \cap L(D_B) \neq \emptyset$. Hence $M_{\mathsf{E}}(T(\langle M, D_B \rangle))$ rejects and M_A rejects.

Thus M_A decides A.

---

24. Problem 5.2

DEF: A language is **co-recognizable** if its complement is recognizable.

Consider the language $EQ_{\mathsf{CFG}} = \{ \langle G_1, G_2 \rangle \mid G_1, G_2 \text{ are CFGs and } L(G_1) = L(G_2) \}$.

We show that EQ_{CFG} is co-recognizable by constructing a recognizer for its complement, namely, $\overline{EQ_{\mathsf{CFG}}} = \{ \langle G_1, G_2 \rangle \mid G_1, G_2 \text{ are CFGs and } L(G_1) \neq L(G_2) \}$.

Want: A TM M that recognizes $\overline{EQ_{\mathsf{CFG}}}$, i.e., such that for all CFGs G_1, G_2:

$L(G_1) \neq L(G_2)$ $\implies$ $M(\langle G_1, G_2 \rangle)$ *accepts*
$L(G_1) = L(G_2)$ $\implies$ $M(\langle G_1, G_2 \rangle)$ *rejects*

Construction: Theorem 4.6 says that the following language is decidable:

$$A_{\mathsf{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates } w \}$$

Let M_{CFG} be a TM that decides A_{CFG}. The following TM recognizes $\overline{EQ_{\mathsf{CFG}}}$.

M = On input $\langle G_1, G_2 \rangle$

Let Σ be the union of the terminal alphabets of G_1 and G_2
For all $w \in \Sigma^*$ do
    If $M_{\mathsf{CFG}}(\langle G_1, w\rangle)$ accepts then $\mathsf{acc}_1 \leftarrow 1$ else $\mathsf{acc}_1 \leftarrow 0$ EndIf
    If $M_{\mathsf{CFG}}(\langle G_2, w\rangle)$ accepts then $\mathsf{acc}_2 \leftarrow 1$ else $\mathsf{acc}_2 \leftarrow 0$ EndIf
    If $\mathsf{acc}_1 \neq \mathsf{acc}_2$ then *accept* EndIf
EndFor

Correctness:

If $L(G_1) \neq L(G_2)$ then there is some $w \in \Sigma^*$ such that 1) $w \in L(G_1)$ and $w \notin L(G_2)$ OR 2) $w \notin L(G_1)$ and $w \in L(G_2)$. M searches for a string w satisfying this property. When it finds one, it accepts.

If $L(G_1) = L(G_2)$ then for all $w \in \Sigma^*$ 1) $w \in L(G_1)$ and $w \in L(G_2)$ OR 2) $w \notin L(G_1)$ and $w \notin L(G_2)$. M searches for a string w such that 1) $w \in L(G_1)$ and $w \notin L(G_2)$ OR 2) $w \notin L(G_1)$ and $w \in L(G_2)$. It does not find one, so it does not halt.