CS 320
Computer Language Processing
Exercises: Week 3

March 7, 2025

Exercise 1 Recall the pumping lemma for regular languages:

For any language $L \subseteq \Sigma^*$, if L is regular, there exists a strictly positive constant $p \in \mathbb{N}$ such that every word $w \in L$ with $|w| \geq p$ can be written as $w = xyz$ such that:

- $x, y, z \in \Sigma^*$
- $|y| > 0$
- $|xy| \leq p$, and
- $\forall i \in \mathbb{N}.\ xy^iz \in L$

Consider the language $L = \{w \in \{a\}^* \mid |w| \text{ is prime}\}$. Show that L is not regular by using the pumping lemma.

Solution $L = \{w \in \{a\}^* \mid |w| \text{ is prime}\}$ is not a regular language.

To the contrary, assume it is regular, and so there exists a constant p such that the pumping conditions hold for this language.

Consider the word $w = a^n \in L$, for some prime $n \geq p$. By the pumping lemma, we can write $w = xyz$ such that $|y| > 0$, $|xy| \leq p$, and $xy^iz \in L$ for all $i \geq 0$.

Assume that $|xz| = m$ and $|y| = k$ for some natural numbers m and k. Thus, $|xy^iz| = m + ik$ for all i. Since by the pumping lemma $xy^iz \in L$ for every i, it follows that for every i, the length $m + ik$ is prime. However, if $m \neq 0$, then m divides $m + mk$, and if $m = 0$, then $m + 2k$ is not prime. In either case, we have a contradiction.

Thus, this language is not regular.

□

Exercise 2 For each of the following languages, give a context-free grammar that generates it:

1. $L_1 = \{a^nb^m \mid n, m \in \mathbb{N} \wedge n \geq 0 \wedge m \geq n\}$
2. $L_2 = \{a^nb^mc^{n+m} \mid n, m \in \mathbb{N}\}$
3. $L_3 = \{w \in \{a, b\}^* \mid \exists m \in \mathbb{N}.\ |w| = 2m + 1 \wedge w_{(m+1)} = a\}$ (w is of odd length, has a in the middle)

Solution

1. $L_1 = \{a^n b^m \mid n, m \in \mathbb{N} \wedge n \geq 0 \wedge m \geq n\}$

$$S ::= aSb \mid B$$
$$B ::= bB \mid \epsilon$$

2. $L_2 = \{a^n b^m c^{n+m} \mid n, m \in \mathbb{N}\}$

$$S ::= aSc \mid B$$
$$B ::= bBc \mid \epsilon$$

   A small tweak to L_1's grammar allows us to keep track of addition precisely here. Could we do something similar for $\{a^n b^n c^n \mid n \in \mathbb{N}\}$? (open-ended discussion)

3. $L_3 = \{w \in \{a, b\}^* \mid \exists m \in \mathbb{N}.\ |w| = 2m + 1 \wedge w_{(m+1)} = a\}$

$$S ::= aSb \mid bSa \mid aSa \mid bSb \mid a$$

   Note that after each recursive step, the length of the inner string has the same parity (i.e. odd).

□

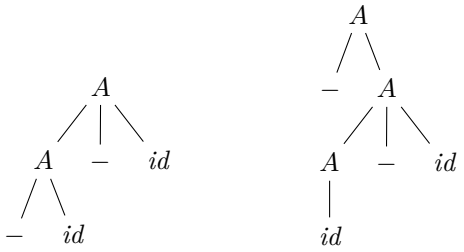
Exercise 3 Consider the following context-free grammar G:

$$A ::= -A$$
$$A ::= A - id$$
$$A ::= id$$

1. Show that G is ambiguous, i.e., there is a string that has two different possible parse trees with respect to G.
2. Make two different unambiguous grammars recognizing the same words, G_p, where prefix-minus binds more tightly, and G_i, where infix-minus binds more tightly.
3. Show the parse trees for the string you produced in (1) with respect to G_p and G_i.
4. Produce a regular expression that recognizes the same language as G.

Solution

1. An example string is $-id - id$. It can be parsed as either $-(id - id)$ or $(-id) - id$. The corresponding parse trees are:



Left: prefix binds tighter, right: infix binds tighter.

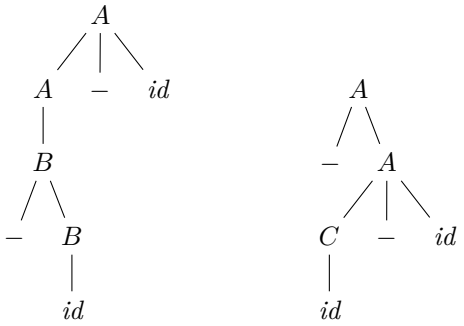
2. G_p:

$$A ::= B \mid A - id$$
$$B ::= -B \mid id$$

G_i:

$$A ::= C \mid -A$$
$$C ::= id \mid C - id$$

3. Parse trees for $-id - id$ with respect to G_p (left) and G_i (right):



4. $L(G) = L(-^* id(-id)^*)$. Note: () are part of the regular expression syntax, not parentheses in the string.

□

Exercise 4 Consider the two following grammars G_1 and G_2:

$$G_1 :$$
$$S ::= S(S)S \mid \epsilon$$
$$G_2 :$$
$$R ::= RR \mid (R) \mid \epsilon$$

Prove that:

1. $L(G_1) \subseteq L(G_2)$, by showing that for every parse tree in G_1, there exists a parse tree yielding the same word in G_2.
2. (Bonus) $L(G_2) \subseteq L(G_1)$, by showing that there exist equivalent parse trees or derivations.

Solution

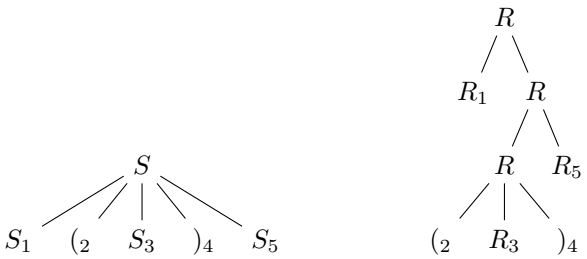
1. $L(G_1) \subseteq L(G_2)$.

   We give a recursive transformation of parse trees in G_1 producing parse trees in G_2.

   (a) **Base case:** The smallest parse tree is the ϵ production, which can be transformed as (left to right):

   (b) **Recursive case:** Rule $S ::= S(S)S$. The parse tree transformation is:

   The nodes are numbered to check that the order of children (left to right) does not change. This ensures that the word yielded by the tree is the same. The transformation is applied recursively to the children S_1, S_3, S_5 to obtain R_1, R_3, R_5.

   Verify that the tree on the right is indeed a parse tree in G_2.

2. $L(G_2) \subseteq L(G_1)$.

   Straightforward induction on parse trees does not work easily. The rule $R ::= RR$ in G_2 is not directly expressible in G_1 by a simple transformation of parse trees. However, we can note that, in fact, adding this rule to G_1 does not change the language!

   Consider the grammar G_1' defined by $S ::= SS \mid S(S)S \mid \epsilon$. We must show that for every two words v and w in $L(G_1)$, vw is in $L(G_1)$, and so adding the rule $S ::= SS$ does not change the language.

   We induct on the length $|v| + |w|$.

   (a) **Base case:** $|v| + |w| = 0$. $v = w = vw = \epsilon \in L(G_1)$. QED.

   (b) **Inductive case:** $|v| + |w| = n + 1$. The induction hypothesis is that for every v', w' with $|v'| + |w'| = n$, $v'w' \in L(G_1)$.

   From the grammar, we know that either $v = \epsilon$ or $v = x(y)z$ for $x, y, z \in L(G_1)$. If $v = \epsilon$, then $w = vw \in L(G_1)$. In the second case, $vw = x(y)zw$. However, $zw \in L(G_1)$ by the inductive hypothesis, as $|z| + |w| < n$.

   Thus, $vw = x(y)z'$ for $z' \in L(G_1)$. Finally, since $x, y, z' \in L(G_1)$, it follows from the grammar rules that $vw = x(y)z' \in L(G_1)$.

Thus, $L(G_1) = L(G_1')$. It can now be shown just as in the first part, that $L(G_2) \subseteq L(G_1')$.

□

Exercise 5 Consider a context-free grammar $G = (A, N, S, R)$. Define the reversed grammar $rev(G) = (A, N, S, rev(R))$, where $rev(R)$ is the set of rules is produced from R by reversing the right-hand side of each rule, i.e., for each rule $n ::= p_1 \dots p_n$ in R, there is a rule $n ::= p_n \dots p_1$ in $rev(R)$, and vice versa. The terminals, non-terminals, and start symbol of the language remain the same.

For example, $S ::= abS \mid \epsilon$ becomes $S ::= Sba \mid \epsilon$.

Is it the case that for every context-free grammar G defining a language L, the language defined by $rev(G)$ is the same as the language of reversed strings of L, $rev(L) = \{rev(w) \mid w \in L\}$? Give a proof or a counterexample.

Solution Consider any word w in the original language. Looking at the definition of a language $L(G)$ defined by a grammar G:

$$w \in L(G) \iff \exists T.\, w = yield(T) \wedge isParseTree(G, T)$$

There must exist a parse tree T for w with respect to G. We must show that there exists a parse tree for $rev(w)$ with respect to the reversed grammar $G_r = rev(G)$ as well.

We propose that this is precisely the tree $T_r = mirror(T)$. Thus, we need to show that $rev(w) = yield(T_r)$ and that $isParseTree(G_r, T_r)$.

1. $rev(w) = yield(T_r)$: $yield(\cdot)$ of a tree is the word obtained by reading its leaves from left to right. Thus, the yield of the mirror of a tree $yield(mirror(\cdot))$ is the word obtained by reading the leaves of the original tree from right to left. Thus, $yield(T_r) = yield(mirror(T)) = rev(yield(T)) = rev(w)$.

2. $isParseTree(G_r, T_r)$: We need to show that T_r is a parse tree with respect to G_r. Consider the definition of a parse tree:

   (a) The root of T_r is the start symbol of G_r: the root of $T_r = mirror(T)$ is the same as that of T. Since T's root node must be the start symbol of G, it is also the root symbol of T_r. G and G_r share the same start symbol in our transformation.

   (b) The leaves are labelled by the elements of A: the mirror transformation does not alter the set or the label of leaves, only their order. This property transfers from T to T_r as well.

   (c) Each non-leaf node is labelled by a non-terminal symbol: the mirror transformation does not alter the label of non-leaf nodes either, so this property transfers from T to T_r as well.

   (d) If a non-leaf node has children that are labelled $p_1, \dots, p_n$ left-to-right, then there is a rule $(n ::= p_1 \dots p_n)$ in the grammar: consider any non-leaf node in T_r, labelled n, with children labelled left-to-right $p_1, \dots, p_n$. By the definition of $mirror$, the original tree T must have

the same node labelled n, with the reversed list of children left-to-right, $p_n, \ldots, p_1$. Since T is a parse tree for G, $n ::= p_n \ldots p_1$ is a valid rule in G, and by the reverse transformation, $n ::= p_1 \ldots p_n$ must be a rule in G_r. Thus, the property is satisfied.

Thus, both properties are satisfied. Therefore, the language defined by the reversed grammar is the reversed language of the original grammar.

□