1 Showing a language is not regular

After this tutorial you should be able to:

1. Show a language is not regular.

You may find the following terminology helpful. For a language L, we say that two strings x and y are *distinguished by* L if there is a string z such that only one of xz and yz are in L (and in this case we say that z *distinguishes* x and y). The method we saw in lectures to prove that a language is not regular, is to find (define) infinitely many strings $x_1, x_2, \cdots$ such that every pair of them is distinguished by L.

Problem 1. Fix $\Sigma = \{a, b\}$. Using the technique from lectures, show the following languages are not regular (for the tutorial pick one).

1. $\{a^i b^j : i > j\}$.
2. $\{a^n b^m : n$ divides m, or m divides $n\}$.
3. $\{a^{n^2} : n \geq 0\}$.
4. All strings $a^i b^j$ such that (a) i is even, or (b) $j < i$ and j is even. (hard)

Solution 1.

1. Let $x_n = a^n$ for every n. If $i > j$ then x_i, x_j can be distinguished by $z = b^j$. Indeed, $x_i z = a^i b^j \in L$ while $x_j z = a^j b^j \notin L$.
2. Recall that an integer $n > 1$ is prime if its only divisors are 1 and n. Let $p(i)$ be the ith prime number, e.g., $p(1) = 2, p(2) = 3, p(3) = 5, p(4) = 7, p(5) = 11, \ldots$.
   Let $x_n = a^{p(n)}$ for every n. Then if $i \neq j$ then x_i, x_j are distinguishable by $z = b^{p(i)}$. Indeed, $x_i z = a^{p(i)} b^{p(i)} \in L$ (since every number divides itself), while $x_j z = a^{p(j)} b^{p(i)} \notin L$ (since no prime divides any other prime).
3. Let $x_n = a^{n^2}$ for every n. If $i < j$ then x_i, x_j are distinguishable by $z = a^{2i+1}$. Indeed, $|x_i z| = i^2 + 2i + 1 = (i+1)^2$ and so $x_i z \in L$, while $j^2 < |x_j z| = j^2 + 2i + 1 < j^2 + 2j + 1 = (j+1)^2$ and so $x_j z \notin L$.
4. Let $x_n = a^{2n+1}$ for every n. If $j < i$ then x_i, x_j are distinguishable by $z = b^{2i}$. Indeed, $x_i z = a^{2i+1} b^{2i} \in L$ while $x_j z = a^{2j+1} b^{2i} \notin L$ since $j < i$ implies $2j + 1 < 2i + 1$ and in particular that $2j + 1 \leq 2i$.

2 Context free grammars

In the lecture we saw a new model of computation called 'context-free grammar' (CFG). A CFG generates strings by rewriting variables (starting with the 'start' variable) until there are

none left. The language of a CFG is called a 'context-free language'. These include all the regular languages, and more. CFGs are central to describing the syntax of many programming languages. We ended the lecture with the fact that some CFGs can generate a string with more than one rightmost derivation (= more than one leftmost derivation, = more than one parse tree). Such strings are called ambiguous, and correspond, intuitively, to different "meanings" of the string.

After this tutorial you should be able to:

1. Convert an English or mathematical description of a context-free language into a CFG, and argue why your grammar is correct.
2. Describe in English or mathematics the language of a CFG, and argue why your description is correct.
3. Argue if a CFG is ambiguous or not.

Problem 2. For each of the following grammars:

Indicate the set of variables, terminals, production rules, and the start variable.

Give two strings derivable by this grammar, and two strings that are not.

Describe in one sentence the language generated by the grammar.

Is the language regular?

1.

$$S \to 0S0 \mid 1$$

2.

$$S \to X1X$$
$$X \to 0X \mid \epsilon$$

Solution 2.

1. The terminal set is $\{0, 1\}$, the variable set is $\{S\}$, the start variable is S, and the rules are as shown.

   In: $0001000, 1$; Out: $011, 10$.

   The language of the grammar is the set of binary strings of the form 0^n10^n for $n \geq 0$.

   The language is not regular. To see this, note that if $x_n = 0^n1$ then x_i and x_j are distinguished by $z = 0^i$.

2. The terminal set is $\{0, 1\}$, the variable set is $\{S, X\}$, the start variable is S, and the rules are as shown.

   In: $10, 0100$; Out: $11, 0101$.

   The language of the grammar is the set of binary strings of the form 0^n10^m for $n, m \geq 0$.

The language is regular, it is the language of the regular expression 0^*10^*.

Aside. Although X stores the information "any number of zero", the two occurrence of X in $S \to X1X$ are independent of each other.

Problem 3. Consider the following grammar:

$$\begin{aligned} E &\to E + T \mid E - T \mid T \\ T &\to F \times T \mid T/T \mid F \\ F &\to (E) \mid V \mid C \\ V &\to a \mid b \mid c \\ C &\to 1 \mid 2 \mid 3 \end{aligned}$$

1. Indicate the set of variables, terminals, production rules, and the start variable.
2. Give a left-most derivation of the string $a + b \times c$
3. Give a right-most derivation of the string $a + b \times c$
4. Give a parse tree for $a \times b - 2 \times c$
5. Give a parse tree for $a \times (b - 2 \times c)$

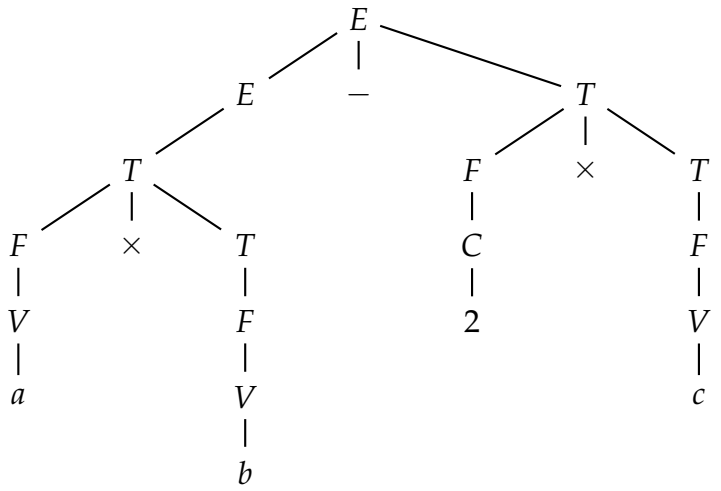
Solution 3.

1. $V = \{E, T, F, V, C\}$
   $T = \{a, b, c, 1, 2, 3, (,), \times, /, +, -\}$
   Production rules are as shown (note there are 15 rules!)
   Start variable as E (if it's not stated we assume it's the first one, or S)
2. $$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow T + T \\ &\Rightarrow F + T \\ &\Rightarrow V + T \\ &\Rightarrow a + T \\ &\Rightarrow a + F \times T \\ &\Rightarrow a + V \times T \\ &\Rightarrow a + b \times T \\ &\Rightarrow a + b \times F \\ &\Rightarrow a + b \times V \\ &\Rightarrow a + b \times c \end{aligned}$$

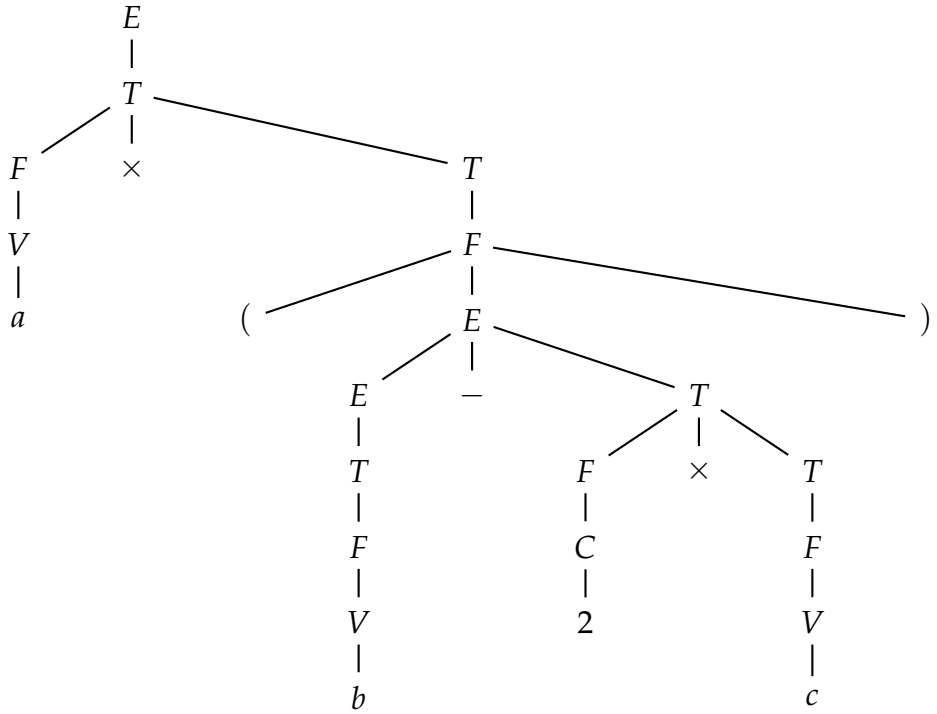
3.

$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow E + F \times T \\
&\Rightarrow E + F \times F \\
&\Rightarrow E + F \times V \\
&\Rightarrow E + F \times c \\
&\Rightarrow E + V \times c \\
&\Rightarrow E + b \times c \\
&\Rightarrow T + b \times c \\
&\Rightarrow F + b \times c \\
&\Rightarrow V + b \times c \\
&\Rightarrow a + b \times c
\end{aligned}
$$

4.



5.



Problem 4. For each of the following languages, find a CFG that generates it (for the tutorial, pick two of these):

1. $\{bb, bbbb, bbbbbb, ...\} = \{(bb)^{n+1} \mid n \in \mathbb{Z}_0^+\}$
2. $\{a, ba, bba, bbba, ...\} = \{b^n a \mid n \in \mathbb{Z}_0^+\}$
3. $\{\varepsilon, ab, abab, ...\} = \{(ab)^n \mid n \in \mathbb{Z}_0^+\}$
4. $\{ac, abc, abbbc, ...\} = \{ab^n c \mid n \in \mathbb{Z}_0^+\}$
5. $\{a^m b^n \mid n, m \in \mathbb{Z}_0^+\}$
6. $\{a^m b^n \mid n, m \in \mathbb{Z}_0^+, m > 0\}$
7. $\{a^m b^n \mid n, m \in \mathbb{Z}_0^+, m > 0, n > 0\}$
8. $\{a^n b^n : n > 0\}$

Solution 4.

1. $\{bb, bbbb, bbbbbb, ...\} = \{(bb)^{n+1} \mid n \in \mathbb{Z}_0^+\}$

   $S \to bb \mid bbS$ (bSb, Sbb would also work fine)

2. $\{a, ba, bba, bbba, ...\} = \{b^n a \mid n \in \mathbb{Z}_0^+\}$

   $S \to a \mid bS$

3. $\{\varepsilon, ab, abab, ...\} = \{(ab)^n \mid n \in \mathbb{Z}_0^+\}$

$$S \to \varepsilon \mid abS \qquad (Sab \text{ would also work fine})$$

4. $\{ac, abc, abbbc, ...\} = \{ab^n c \mid n \in \mathbb{Z}_0^+\}$

$$S \to aBc \qquad (\text{this puts the a and c on the ends})$$
$$B \to bB \mid \varepsilon \qquad (\text{derives any number of b's})$$

5. $\{a^m b^n \mid n, m \in \mathbb{Z}_0^+\}$

$$S \to AB$$
$$A \to aA \mid \varepsilon \qquad (\text{derives any number of a's})$$
$$B \to bB \mid \varepsilon \qquad (\text{derives any number of b's})$$

6. $\{a^m b^n \mid n, m \in \mathbb{Z}_0^+, m > 0\}$

$$S \to AB$$
$$A \to aA \mid a \qquad (\text{derives at least one a})$$
$$B \to bB \mid \varepsilon \qquad (\text{derives any number of b's})$$

7. $\{a^m b^n \mid n, m \in \mathbb{Z}_0^+, m > 0, n > 0\}$

$$S \to AB$$
$$A \to aA \mid a \qquad (\text{derives at least one a})$$
$$B \to bB \mid b \qquad (\text{derives at least one b})$$

8. $\{a^n b^n : n > 0\}$

$$S \to aTb$$
$$T \to aTb|\epsilon$$

Problem 5.

1. Write a CFG for the language of balanced parentheses. So, e.g., the following strings are in the language: $()(())$ and $((())$ and ϵ, while the following are not: $)()$ and $((()))$. Explain each rule in at most two sentences.
2. Show this language is not regular.

Solution 5.

1. Consider the CFG

$$S \to \epsilon \mid (S) \mid SS$$

The first rule says that the empty empty string is balanced, the second that if w is balanced then so is (w), the third that if v, w are balanced then so is vw.

This can also be written as a recursive definition of what it means for a string to have 'balanced parentheses':

(a) The empty string has balanced parenths.

(b) If w has balanced parenths then so does (w).

(c) If v, w have balanced parents then so does vw.

2. Call this language L. To see that L is not regular, let L' be the intersection of L with the language of the regular expression $(^*)^*$. Then L' is the set of strings of the form $(^n)^n$ for $n \geq 0$. So if L were regular, also L' would be regular, but we have shown it is not — actually we showed that $\{a^n b^n : n \geq 0\}$ is regular, but the identical proof will work replacing a by '(' and b by ')'.

Practice problems

Problem 6. (Exam 2022) Fix $\Sigma = \{0, 1\}$. Provide a context-free grammar for the language

$$\{u0v1w \in \Sigma^* : \text{len}(v) = \text{len}(u) + \text{len}(w), u, v, w \in \Sigma^*\}$$

For instance, taking $u = 01, w = 11, v = 1100$ we get that $u0v1w = 0101100111$ is in the language.

Give a short explanation/justification of your answer.

Note that u, v, w in the expression above are string variables; while in context-free grammars we use the word 'variable' to refer to a symbol that can be rewritten during a derivation.

Solution 6.

$$\begin{aligned} S &\to AB \\ A &\to 0A0 \mid 0A1 \mid 1A0 \mid 1A1 \mid 0 \\ B &\to 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid 1 \end{aligned}$$

In words,

- The S rule splits the string into two AB.
- The A rule generates strings of the form $x0y$ where $\text{len}(x) = \text{len}(y)$.
- The B rule generates strings of the form $x'1y'$ where $\text{len}(x') = \text{len}(y')$.

To see that S generates only strings in the language, note that S generates strings of the form $x0yx'1y'$ with $|x| = |y|$ and $|x'| = |y'|$, and in particular, $|yx'| = |x| + |y'|$.

To see that every string $u0v1w$ with $|v| = |u| + |w|$ is generated by S have A generate $u0v'$ where v' is the prefix of v of length u, and have B generate $v''1w$ where v'' is the suffix of v of length w.

Problem 7. Some programming languages define the `if` statement in similar ways to the following grammar:

$$\begin{aligned} \textit{Conditional} &\rightarrow \text{ if } \textit{Condition} \text{ then } \textit{Statement} \\ \textit{Conditional} &\rightarrow \text{ if } \textit{Condition} \text{ then } \textit{Statement} \text{ else } \textit{Statement} \\ \textit{Statement} &\rightarrow \textit{Conditional} \mid S_1 \mid S_2 \\ \textit{Condition} &\rightarrow C_1 \mid C_2 \end{aligned}$$

1. Show that this grammar is ambiguous.
2. Show that the string you provided can be interpreted in two different ways, i.e., resulting in programs with different behaviours. You may want to write the programs corresponding to each derivation (hint: you can read these off from the parse trees).
3. Write a CFG that captures `if` statements but is not ambiguous.

Solution 7. The string "if C1 then if C2 then S1 else S2" has two leftmost derivations. Here is one leftmost derivation:

Conditional

$\Rightarrow$ if *Condition* then *Statement*

$\Rightarrow$ if C1 then *Statement*

$\Rightarrow$ if C1 then *Conditional*

$\Rightarrow$ if C1 then if *Condition* then *Statement* else *Statement*

$\Rightarrow$ if C1 then if C2 then *Statement* else *Statement*

$\Rightarrow$ if C1 then if C2 then S1 else *Statement*

$\Rightarrow$ if C1 then if C2 then S1 else S2

Here is another:

Conditional

$\Rightarrow$ if *Condition* then *Statement* else *Statement*

$\Rightarrow$ if C1 then *Statement* else *Statement*

$\Rightarrow$ if C1 then *Conditional* else *Statement*

$\Rightarrow$ if C1 then if *Condition* then *Statement* else *Statement*

$\Rightarrow$ if C1 then if C2 then *Statement* else *Statement*

$\Rightarrow$ if C1 then if C2 then S1 else *Statement*

$\Rightarrow$ if C1 then if C2 then S1 else S2

Consider the code in the Figure. Each program corresponds to one of the derivations. If $C1$ is true and C_2 is false, then the first does S_2, and the other doesn't do S_2.

Here is an unambiguous CFG that captures "if-then-else" statements.

$$\begin{aligned} \textit{Conditional} &\rightarrow \text{ if } \textit{Condition} \text{ then } \textit{Statement} \text{ endif} \\ \textit{Conditional} &\rightarrow \text{ if } \textit{Condition} \text{ then } \textit{Statement} \text{ else } \textit{Statement} \text{ endif} \end{aligned}$$

```
1  def first(C1,C2):
2      if C1:
3          if C2:
4              return("S1")
5          else:
6              return("S2")
7
8  def second(C1,C2):
9      if C1:
10         if C2:
11             return("S1")
12     else:
13         return("S2")
14
15 print("First  does", first(True,False))
16 print("Second does", second(True,False))
```

Problem 8. Consider the CFG G':

$$S \rightarrow Sab \mid aSb \mid abS \mid Sba \mid bSa \mid baS \mid \epsilon$$

1. Show that if $x \in L(G')$ then x has the same number of as as bs.
2. Show that there is a string with the same number of as as bs that is not in $L(G')$.

The moral of this problem is that it is easy to design a CFG that misses some intended strings (in this case, the intended language is "same number of as as bs").

Solution 8.

1. The base rule ($S \rightarrow \epsilon$) produces a string with the same number of as as bs (i.e., zero many), and each application of any of the rest of the rules adds 1 to this number. A more precise proof would go by induction on the length of a derivation for x.
2. The string $x = aabbbbaa$ has four as and four bs. However, it is not generated by S. Why? Simply because there is no first rule that can be applied to give

this string, e.g., $S \to Sab$ generates strings ending in ab (which x does not), $S \to aSb$ generates strings starting with a and ending with b (which x does not), etc.

Problem 9. I asked GPT-4 to write recursive code for the language of well-formed parentheses.

It gave me this:

```
def is_well_formed(s):
    # Base case: An empty string is well-formed
    if not s:
        return True
    # If string starts with '(' and ends with ')'
    if s[0] == '(' and s[-1] == ')':
        return is_well_formed(s[1:-1])
    return False
```

Is this correct?

Solution 9. This depends on what we mean by "well-formed parentheses". The procedure given generates the same language as the grammar $S \to (S)|\epsilon$. This only generates well-formed parentheses. But it misses strings like $()()$ which we would consider well-formed.

Problem 10. Consider the following context-free grammar G:

$$S \to ST$$
$$S \to a$$
$$T \to BS$$
$$B \to +$$

For each of the following strings, say whether it is generated by the grammar, and if so, give a derivation, and if not give a reason:

1. $+a+a$
2. $a+a$
3. $a+a+a$
4. $a+a++$

Solution 10.

1. No, since strings in $L(G)$ must start with an a since every string of terminals and nonterminals in a derivation from S must either begin with S (via $S \to ST$) or with a (via $S \to a$).

2. $S \Rightarrow ST \Rightarrow aT \Rightarrow aBS \Rightarrow a + S \Rightarrow a + a$.
3. $S \Rightarrow ST \Rightarrow aT \Rightarrow aBS \Rightarrow a + S \Rightarrow a + ST \Rightarrow a + aT \Rightarrow a + aBS \Rightarrow a + a + S \Rightarrow a + a + a$.
4. No, since every $+$ that is derived must be followed by a string derived from S (via $T \to BS$ and $B \to +$), and strings derived from S must start with an a (as argued before).

Problem 11. Describe the language generated by the following context-free grammar:

$$S \to X1Y$$
$$X \to \epsilon \mid X0$$
$$Y \to \epsilon \mid 1Y \mid Y0$$

Briefly explain why your answer is correct.

Is the grammar ambiguous?

Solution 11. Language of the regexp $0^*11^*0^*$.

Informally, the reason is that S generates $X1Y$; X generates 0^*; Y generates 1^*0^*.

More precisely, we will show two things: that every string matching the regexp $0^*11^*0^*$. can be generated by this grammar; and that every leftmost derivation generates a string matching this grammar.

Every string of the form $0^n11^m0^l$ can be generated as follows: $S \Rightarrow X1Y \Rightarrow^n 0^n1Y \Rightarrow^m 0^n11^mY \Rightarrow^l 0^n11^m0^l$. On the other hand, a leftmost derivation must look as follows $S \Rightarrow X1Y \Rightarrow^n 0^n1Y$ for some $n \geq 0$, and $Y \Rightarrow^* 1^m0^l$ for some $m, l \geq 0$.

It is ambiguous because, e.g., $S \Rightarrow X1Y \Rightarrow 1Y \Rightarrow 11Y \Rightarrow 11Y \Rightarrow 11Y0 \Rightarrow 110$ and $S \Rightarrow X1Y \Rightarrow 1Y \Rightarrow 1Y0 \Rightarrow 11Y0 \Rightarrow 110$ are two leftmost derivations of the string 110.

Problem 12. Consider the following context-free grammar:

$$S \to 0S \mid 0S1S \mid \epsilon$$

For each of the following words, say whether or not it is generated by the grammar: 001, 0101, 0110. The only justification that is needed is a derivation for those words that are generated by the grammar.

Show that the grammar is ambiguous.

Solution 12.

1. $S \Rightarrow 0S1S \Rightarrow 00S1S \Rightarrow 001S \Rightarrow 001$, and another leftmost $S \Rightarrow 0S \Rightarrow 00S1S \Rightarrow 001S \Rightarrow 001$, thus grammar is ambiguous.

2. $S \Rightarrow 0S1S \Rightarrow 01S \Rightarrow 010S1S \Rightarrow 0101S \Rightarrow 0101$.
3. Not generated.

Problem 13. (Practice Exam) Fix $\Sigma = \{0, 1, \#\}$. Provide a context-free grammar for the language of strings of the form $u\#v$ where $u, v \in \{0,1\}^*$ and $|u| = |v|$ and the reverse of u is not equal to v.

For instance, $01\#11, 011\#011$ is in the language, while $\#, 0\#0, 01\#10$ are not.

Give a short explanation/justification of your answer.

Solution 13. A string is in the language if and only if it is of the form $x'0x\#y1y'$ or $x'1x\#y0y'$ with $x, y, x', y' \in \{0,1\}^*$, $|x'| = |y'|$ and $|x| = |y|$.

$$
\begin{aligned}
S &\to ASA \mid T \\
T &\to 0U1 \mid 1U0 \\
U &\to AUA \mid \# \\
A &\to 0 \mid 1
\end{aligned}
$$

1. A generates 0 or 1
2. U generates strings of the form $x\#y$ where $|x| = |y|$
3. T generates strings of the form $0x\#y1$ or $1x\#0y$ where $|x| = |u|$
4. S generates strings of the form $x'0x\#y1y'$ or $x'1x\#y0y'$ where $|x'| = |y'|$, $|x| = |y|$.

Problem 14. A CFG is called *right-regular* if every rule is of the form $A \to a$ or $A \to aB$ where A, B are arbitrary nonterminals and a is an arbitrary terminal or ϵ. A CFG is called *left-regular* if every rule is of the form $A \to a$ or $A \to Ba$ where A, B are arbitrary nonterminals and a is an arbitrary terminal or ϵ. A CFG is *regular* if it is left-regular or right-regular.

1. Write a right-regular grammar generating the language $L(a\,|(b^*c))$.
2. Write a left-regular grammar generating the language $L(a\,|(b^*c)$.
3. Find a grammar that only has rules of the form $A \to a$ or $A \to aB$ or $A \to Ba$, and that generates a language that is not the language of any RE.
4. Show that the regular grammars generated exactly the languages of regular expressions.

Solution 14.

1. We write a right-regular grammar as follows:

$$S \to a$$
$$S \to bT$$
$$S \to c$$
$$T \to bT$$
$$T \to c$$

2. We write a left-regular grammar as follows:

$$S \to a$$
$$S \to Tc$$
$$T \to \epsilon$$
$$T \to Tb$$

3. We write a grammar as follows:

$$S \to aT$$
$$T \to Sb$$
$$S \to c$$

This grammar generates the language $\{a^n cb^n : n \geq 1\}$, which is not regular.

Problem 15. Give a context-free grammar which generates just the valid identifiers. A valid identifier begins with a letter or underscore, contains only letters, underscores, or digits, and is ended by a blank.

Solution 15.

$\langle Identifier \rangle \to \langle First \rangle \langle Rest \rangle \langle Space \rangle$ (i.e. the first char, the rest, then the terminating space)
$\langle First \rangle \to \langle Underscore \rangle \mid \langle Letter \rangle$
$\langle Rest \rangle \to \varepsilon \mid \langle Character \rangle \langle Rest \rangle$ (Recursively derive the middle of the string)
$\langle Character \rangle \to \langle Underscore \rangle \mid \langle Letter \rangle \mid \langle Digit \rangle$
$\langle Underscore \rangle \to \_$
$\langle Letter \rangle \to a \mid b \mid ... \mid Z$
$\langle Digit \rangle \to 0 \mid 1 \mid 2 \mid 3 \mid ... \mid 9$
$\langle Space \rangle \to \textvisiblespace$

Problem 16. Give a CFG that generates the language of propositional formulas over the atoms p, q. Make sure to state what the variables and terminals are.

Solution 16.

$$S \to p \mid q \mid (S \wedge S) \mid (S \vee S) \mid \neg S$$

- Variable S
- Terminals $p, q, (,), \wedge, \vee, \neg$

NB. Note that the terminals can be non alpha-numeric symbols.

Problem 17. Provide a context-free grammar for the following language L: the set of well-bracketed strings that use two different types of brackets, i.e., $()$ and $\{\}$. For instance, $(\{\})()$ is in the language, as is $(())$, as is $()\{\}$, but $(\}$ is not. To justify your answer you should briefly explain a) why your grammar only generates strings in L, and b) why your grammar generates all strings in L.

Solution 17.

$$S \to (S) \mid \{S\} \mid SS \mid \epsilon$$

Every well-bracketed string is generated: if u is well-bracketed, it is of the form $u = (s)$ or $u = \{s\}$ or $u = st$ where s, t are well-brackted. Then (by induction on the length of the string), each of s, t is generated, and then so is u, e.g., if $u = (s)$ then start with $S \to (S)$ and proceed with a derivation of s.

Conversely, if a string s is generated, then either the derivation starts $S \to (S)$, in which case $s = (u)$ where u is well-bracketed (by induction on the length of the derivation), and so s is well bracketed, or it starts with $S \to \{S\}$ (and the reason is similar), or it starts with $S \to SS$, and so $s = uv$ where u, v are well-bracketed (by induction on the length of the derivation), and so is s.

Problem 18. Consider the following grammars.

1. $S \to a \mid SbS$
2. $S \to aS \mid Sa \mid a$
3. $S \to S[S]S \mid \varepsilon$
4. $S \to 0S1 \mid 01$
5. $S \to +SS \mid -SS \mid a$
6. $S \to SS+ \mid SS* \mid a$
7. $S \to S(S)S \mid \epsilon$

8. $S \to aSbS \mid bSaS \mid \epsilon$
9. $S \to a \mid S + S \mid SS \mid S* \mid (S)$

For each:

- Describe in English the language generated, and justify your answer.
- Say if the language is ambiguous, and justify your answer.
- If ambiguous, try find an equivalent unambiguous grammar.

Solution 18.

1.
   - The grammar generates the language described by $(ab)^\star a$.
   - "ababa" is ambiguous. (i.e. it has two parse trees (you need to show them, or give two leftmost derivations, or two rightmost deriviations)) e.g. these are both leftmost derivations of "ababa"
     $S \Rightarrow SbS \Rightarrow abS \Rightarrow abSbS \Rightarrow ababS \Rightarrow ababa$
     $S \Rightarrow SbS \Rightarrow SbSbS \Rightarrow abSbS \Rightarrow ababS \Rightarrow ababa$
   - $S \to a \mid abS$ is equivalent (and is not ambiguous)
2.
   - Strings of one or more a's.
   - "aa" has two parse trees, because it has two left derivations:
     $$S \Rightarrow aS \Rightarrow aa$$
     $$S \Rightarrow Sa \Rightarrow aa$$
   - $S \to aS \mid a$
3.
   - Balanced square brackets.
   - "[][]" has two parse trees, because it has two left derivations:
     $$S \Rightarrow S[S]S \Rightarrow S[S]S[S]S \Rightarrow ... \Rightarrow [][]$$
     $$S \Rightarrow S[S]S \Rightarrow [S]S \Rightarrow []S \Rightarrow []S[S]S \Rightarrow ... \Rightarrow [][]$$
   - $S \to [S]S \mid \epsilon$
4.
   - Strings of the form 0^n1^n for $n \geq 1$.
   - The grammar itself is unambiguous because at every step of the derivation just there is just one possible rule to fire.
5. Arithmetic expressions with variable a, operators $+, -$, in prefix notation (aka Polish notation).
   - It does seem unambiguous (todo: add a proof of this fact).
6. Similar to the previous item, but in postfix notation.
7. All balanced parentheses, similar to item 3 above.

8. Same number of *a*s as *b*s.

Problem 19. Construct unambiguous CFGs for each of the following languages. In each case show that your grammar is correct.

1. Arithmetic expressions in postfix notation.
2. Left-associative lists of identifiers separated by commas.
3. Right-associative lists of identifies separated by commas.
4. Arithmetic expressions of integers and identifiers with the four binary operations $+, -, *, /$.
5. Add unary plus and minus to the arithmetic operations of the previous language.

Problem 20. Find a syntactic condition on the structure of a grammar that guarantees that the language generated by the grammar is infinite (i.e., contains infinitely many strings).

Problem 21. Here is an application of grammars to Compilers.

Syntax-directed translation is done by attaching rules or programs fragments to productions in a grammar.

Consider the CFG generating the set of binary strings:

$$\begin{aligned} S &\to 0 \\ S &\to 1 \\ S &\to S0 \\ S &\to S1 \end{aligned}$$

Attach to each rule the following pseudocode:

$$\begin{aligned} S \to\ & 0 && S.val = 0 \\ S \to\ & 1 && S.val = 1 \\ S \to\ & S0 && S.val = S.val \times 2 \\ S \to\ & S1 && S.val = S.val \times 2 + 1 \end{aligned}$$

Write an algorithm that will take a parse-tree for a string as input, do a bottom-up traversal of the tree (i.e., visit a node only after visiting all its children), and when visiting an internal node of the tree, executes the corresponding command. Once the traversal is complete, what is the content of $S.val$?

Problem 22. Construct syntax-guided translations between notations of arithmetic expressions:

1. From infix notation to prefix notation.
2. From infix notation to prefix notation.
3. From postfix notation to infix notation.

Problem 23. In lectures we said that the following CFG G generates the set L of strings with the same number of as as bs.

$$S \to \epsilon \mid aSbS \mid bSaS$$

Why is this true?

Solution 23. We must show that $L(G) = L$.

This requires that we do two things.

(1) Show $L(G) \subseteq L$. In words, we must show that G only generates strings in L. To see this, note that ϵ is generated and is in L, and that the other rules add exactly one a and exactly one b.

[A more precise argument is by induction on the length of the derivations. i.e., Suppose $x \in L(G)$. Suppose there is a derivation of x of length n, we write this as $S \stackrel{n}{\Rightarrow} x$. If $n = 1$ then $x = \epsilon$ which is in L. Suppose $n > 1$. The inductive hypothesis states that if a string has a derivation of length $< n$ then that string is in L. Suppose x starts with a (the other case is symmetric). Then $S \stackrel{n}{\Rightarrow} x$ can be written $S \Rightarrow aSbS \stackrel{n_1}{\Rightarrow} aubS \stackrel{n_2}{\Rightarrow} aubv$. But here $n_1, n_2 < n$, and so by the inductive hypothesis, $u, v \in L$. I.e., x has the same number of as as bs, and y has the same number of as as bs. So $x = aubv$ has the same number of as as bs.]

(2) Show $L \subseteq L(G)$. In words, we must show that every word in L is generated by G. To see this, let n be the number of zeros and ones in a string $x \in L$. If $n = 0$ then x is the empty string which has a derivation, i.e., $S \to \epsilon$. Otherwise, if $n > 0$, then suppose x starts with 0 (the other case is symmetric). Now, scan x from left to right until we get to the first position, say i, in which the number of zeros is equal to the number of ones. This position must end in a 1, and so we can write $x = 0u1v$ where both u has the same number of 0s and 1s, and since it is shorter than x we already know it is in $L(G)$, and the same for v.

[A more precise argument is as follows. Let $x \in L$. If $x = \epsilon$ simply note that $S \to \epsilon$ derives it. So suppose x starts with a (the case that it starts with b is symmetric). For a string z let $c(z)$ be the number of as in z minus the number of bs in z. For $i \leq |x|$ let $f(i) = c(x_1 x_2 \cdots x_i)$. Note that (*): $x \in L$ if and only if $f(|x|) = 0$. So, suppose $x \in L$. Then $f(|x|) = 0$ and $f(1) > 0$. Let $j \leq |x|$ be the smallest integer such that $f(j) = 0$. Then $x_1 = a, x_j = b$. Also $f(x_2 x_3 \cdots x_{j-1}) = 0$ and so by (*) we have $x_2 x_3 \cdots x_{j-1} \in L$, and similarly $f(x_{j+1} \cdots x_m) = 0$ and so by (*) we have $x_{j+1} \cdots x_m \in L$.]

Problem 24. Show that the set of strings over the alphabet $\{(,)\}$ accepted by the

function *check* is the set of strings generated by the grammar $S \to SS \mid (S) \mid \varepsilon$.

Figure 1: Pseudocode

```
def check(myStr):
    counter = 0
    for i in myStr:
        if i == "(":
            counter += 1
        elif i == ")":
            if counter > 0:
                counter -= 1
            else:
                return "Reject"
    if counter == 0:
        return "Accept"
    else:
        return "Reject"
```

Solution 24. We will show two things: (1) every string generated by the grammar is accepted by the function, (2) that every string accepted by the function is generated by the grammar.

We need two important properties of the function:

1. If w is accepted by the function then (w) is accepted by the function. This is true because the new string only increases the number of open parenths at the start, so the count position i of w is one more for (w) than it was for w, and decreases the number of closed parenths at the end, so the total number of open is equal to the total number of closed parenths.
2. If v, w are accepted by the function then vw is accepted by the function. This is true because while reading v the difference is at least 0, after v is read the difference is equal to 0, so while reading w the difference is at least 0, and after reading w the difference is equal to 0.

To show that every string generated by the grammar is accepted by the function we can argue as follows. Look at the first step of any derivation. If it is $S \Rightarrow \varepsilon$ then the derived string is the emptystring, which is accepted by the function. If it is $S \Rightarrow (S)$ then the derived string is of the form (w) where w is accepted by the function. But then by the first property, also the derived string (w) is accepted by the function. If it is $S \to SS$ then the derived string is of the form vw where both v and w are accepted by the function. Then by the second property, also the derived string vw is accepted by the function.

To show that every accepted by the function string is generated by the grammar we can argue as follows. The empty string is generated. So take a non-empty accepted by the function string x. Suppose as you read x left to right that the difference between the number of open parenths and closed parenths is always > 0, except at the end when it must be $= 0$. Then the string is of the form (w)

for some accepted by the function string w, and so we can derive it by starting with the step $S \Rightarrow (S)$. Suppose that the number is $= 0$ somewhere in the string, say after reading v. Then $x = vw$ where v, w are accepted by the function. So we can derive x by starting with the step $S \Rightarrow SS$.

Although the above proof is good enough to convince someone that knows what is going on that the statement is correct, here is a more formal proof for those that want to convince even the most skeptical person.

Proof. The first direction is proved by strong induction on the length of the derivations. There is only one derivation of length 1, i.e., $S \Rightarrow \varepsilon$, and ε is accepted by the function. So, suppose every derivation of length $\leq k$ generates a string accepted by the function string. Consider a derivation of length $k+1$, suppose it starts with the rule $S \Rightarrow (S)$. Then, the derived string is of the form (w) where w has a derivation of length k. So, by our inductive assumption, w is accepted by the function, and so by the first property so is the derived string (w). Suppose it starts with the rule $S \Rightarrow SS$. Then the derived string is of the form vw where each of v and w has a derivation of length $\leq k$ (why?). So by our inductive hypothesis v is accepted by the function and w is accepted by the function. So by the second property, also the derived string vw is accepted by the function.

The second direction is proved using strong induction on the length of the string x accepted by the function. If the length is zero, then $x = \epsilon$ which is derivable. Otherwise, suppose that every string accepted by the function of length at most k is generated, and let x be a string accepted by the function of length $k+1$. Then there are two cases. If the number of open minus closed in x is always > 0 (except at the end), then $x = (w)$ where w is accepted by the function. But since w has length $< k$, by our inductive hypothesis it has a derivation in our grammar. Thus, we can derive x by starting with $S \Rightarrow (S)$ and then applying the derivation of w. For the other case, say after reading v the difference is 0, then $x = vw$ where v, w are accepted by the function. Since $|v|, |w| < k$, by our inductive hypothesis, both can be derived. And so to derive x start with $S \to SS$ and then apply the derivation of v to the first S and then the derivation of w to the second. □