CIS 425

Spring 2019

Assignment 1

October 2019

1 Language Generated by Grammar

1. Consider the grammar

   $\langle S \rangle$ ::= aS | T

   $\langle T \rangle$ ::= bT | U

   $\langle U \rangle$ ::= cU | ϵ

   (a) Which of the following strings is generated by this grammar?

   i. [✓ ccc]
   ii. aba
   iii. bab
   iv. ca

   (b) Which of the following is a derivation of the string bbc?

   i. $S \rightarrow T \rightarrow U \rightarrow bU \rightarrow bbU \rightarrow bbU \rightarrow bbc$
   ii. $S \rightarrow bT \rightarrow bbT \rightarrow bbU \rightarrow bbcU \rightarrow bbc$
   iii. [✓ $S \rightarrow T \rightarrow bT \rightarrow bbT \rightarrow bbU \rightarrow bbcU \rightarrow bbc$]
   iv. $S \rightarrow T \rightarrow bT \rightarrow bTbT \rightarrow bbT \rightarrow bbcU \rightarrow bbc$

   (c) Which of the following regular expressions accepts the same language as this grammar?

   i. $(a|b|c)^*$
   ii. abc^*
   iii. [✓ $a^*b^*c^*$]
   iv. $(a|ab|abc)$

2. Which of the following grammars describes the same language as 0^n1^m where $m \leq n$?

   (a) $S ::= 0S1 \ | \ \epsilon$
   (b) $S ::= 0S1 \ | \ S1 \ | \ \epsilon$
   (c) [✓ $S ::= 0S1 \ | \ 0S \ | \ \epsilon$]
   (d) $S ::= SS \ | \ 0 \ | \ 1 \ | \ \epsilon$

3. What language does this grammar generate?

   $\langle S \rangle$ ::= aSa | aBa

   $\langle B \rangle$ ::= bB | b

Solution: $L(G) = \{a^k b^m a^k | m, k > 0\}$ □

4. What language does this grammar generate?

   $\langle S \rangle$ ::= abScB | ϵ

   $\langle B \rangle$ ::= bB | b

   Solution: $L(G) = \{(ab)^n (cb^+)^n | n \geq 0\}$ □

2 Designing Grammars

Follows some hints to generate grammars.
1. Use recursive productions to generate an arbitrary number of symbols.
2. To generate languages with matching, balanced, or related numbers of symbols, write productions which generate strings form the middle.
3. For a language that is the union of other languages, use separate nonterminals for each part of the union and then combine.

1. Design a grammar that generates zero or more a's *Solution:* $S ::= aS \mid \epsilon$ □
2. Design a grammar that generates one or more b's *Solution:* $S ::= bS \mid b$ □
3. Design a grammar that generates the language a^*b^*

   Solution: $S ::= A$
   $A ::= aA \mid B$
   $B ::= bB \mid \epsilon$ □
4. Design a grammer that generates the language $\{a^n b^n | n \geq 0\}$

   Solution: $S ::= aSb \mid \epsilon$ □
5. Design a grammar that generates the language $\{a^n b^{2n} | n \geq 0\}$

   Solution: $S ::= aSbb \mid \epsilon$ □
6. Design a grammar that generates the language $\{a^n b^m | 0 \leq n \leq m \leq 2n\}$

   Solution: $S ::= aSb \mid aSbb \mid \epsilon$ □
7. Design a grammar that generates words formed from $\{0, 1\}$ that begin and end with the same symbol.

Solution:

$$\begin{aligned} S &::= 0C0 \mid 1C1 \\ C &::= 0C \mid 1C \mid \epsilon \end{aligned}$$

□

8. Design a grammar that generates the language $\{a^n(b^m \mid c^m) \mid m > n \geq 0\}$
(**Note** that this can be rewritten as $\{a^nb^m|m > n \geq 0\} \cup \{a^nc^m|m > n \geq 0\}$)
Solution:

$$\begin{aligned} S &::= T \mid V \\ T &::= aTb \mid U \\ U &::= Ub \mid b \\ V &::= aVc \mid W \\ W &::= Wc \mid c \end{aligned}$$

□

9. Design a grammar for the language L which is over over the alphabet $\{a, b, c\}$, which consists of at least three consecutive b's (For example, L includes the strings bbb and $abcbbba$, but not $abbab$).
Solution:

$$\begin{aligned} S &::= XbbbX \\ X &::= ABC \\ A &::= aA \mid \epsilon \\ B &::= aB \mid \epsilon \\ C &::= aC \mid \epsilon \end{aligned}$$

□

10. For each of the above grammars, state if the grammar is regular or context-free
Solution: 1 regular, 2 regular, 3 regular, 4 context-free , 5 context-free, 6 context-free, 7 context-free, 8 context-free, 9 context-free □

3 Associativity and Priority

Rewrite the ambiguous grammar:

$$E ::= E + E \mid E * E \mid (E) \mid a \mid b$$

1. To reflect the fact that $+$ and $*$ are left associative and $*$ has higher precedence than $+$
2. To reflect the fact that $+$ and $*$ are right associative and $*$ has higher precedence than $+$

Solution:

1. $E ::= E + T \mid T$
   $T ::= T * P \mid P$
   $P ::= a \mid b \mid (E)$
2. $E ::= T + E \mid T$
   $T ::= P * T \mid P$
   $P ::= a \mid b \mid (E)$

$\square$

4 Parse Trees and Ambiguity

1. Show that the following grammar is ambiguous:

   $S ::= SS \mid () \mid (S)$

   Solution: This grammar has two distinct left-most derivations for some strings. For example: $()()()$

   - $S \to \mathbf{S}S \to \mathbf{S}SS \to ()\mathbf{S}S \to ()()\mathbf{S} \to ()()()$
   - $S \to \mathbf{S}S \to ()\mathbf{S} \to ()\mathbf{S}S \to ()()\mathbf{S} \to ()()()$

   $\square$

2. Which of the following grammars is ambiguous?

   (a) $S ::= 0SS1 \mid 0S1 \mid \epsilon$
   (b) $S ::= A1S1A \mid \epsilon$
       $A ::= 0$
   (c) $S ::= (S, S, S) \mid 1$
   (d) None of the above

3. Consider the following grammar:

$\langle stmt \rangle$ ::= $\langle assignment \rangle$ | $\langle if\text{-}stmt \rangle$

$\langle assignment \rangle$ ::= ‘`a = 1`’ | ‘`a = 2`’

$\langle expr \rangle$ ::= ‘`x > y`’ | ‘`x < z`’

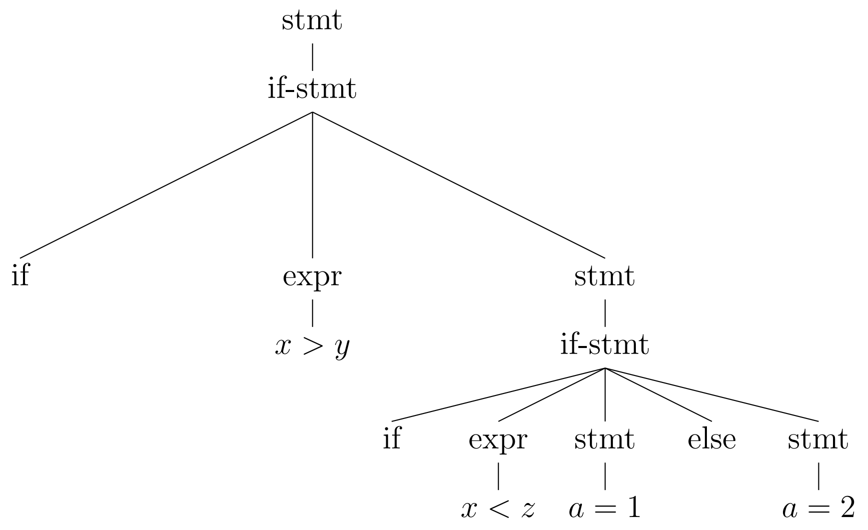
$\langle if\text{-}stmt \rangle$ ::= ‘`if`’ ($\langle expr \rangle$) $\langle stmt \rangle$ | ‘`if`’ ($\langle expr \rangle$) $\langle stmt \rangle$ ‘`else`’ $\langle stmt \rangle$

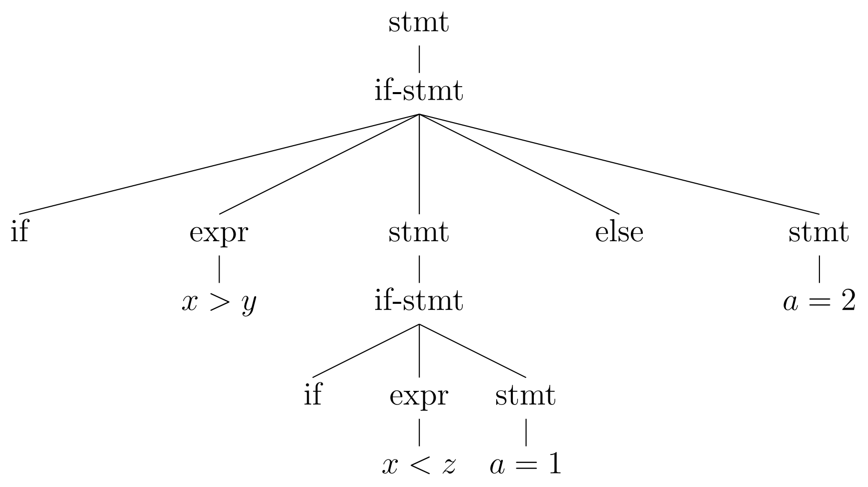
Note that <> is used to denote non-terminals and *stmt* is the root symbol

Draw two parse trees for the following program fragment:

if ($x > y$) if ($x < z$) a = 1 else a = 2

Solution:





□

4. Consider the following grammar for arithmetic expressions:

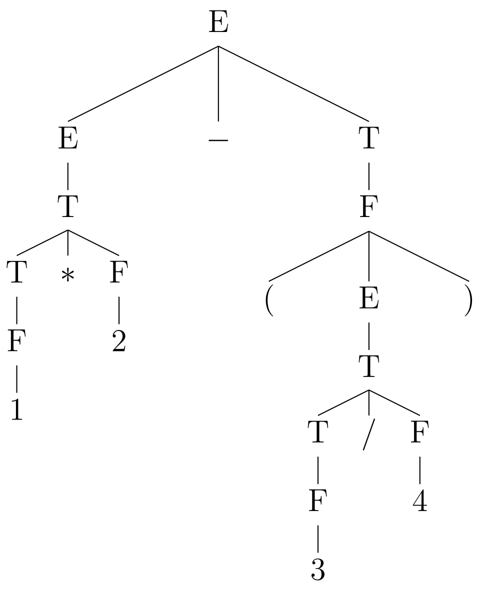
$\langle E \rangle$ ::= E + T | E - T | T

$\langle T \rangle$ ::= T * F | T / F | F

$\langle F \rangle$ ::= 1 | 2 | 3 | 4 | (E)

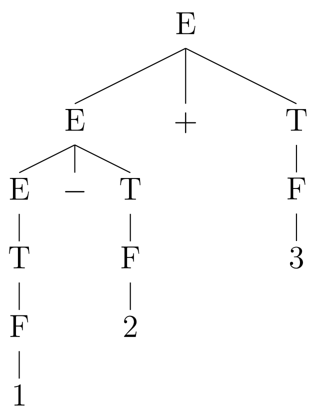
5. Draw a parse tree for the following strings:

   (a) $1 * 2 - (3/4)$
   Solution:



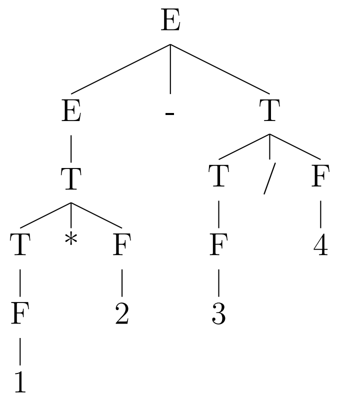
$\square$

   (b) $1 - 2 + 3$
   Solution:



$\square$

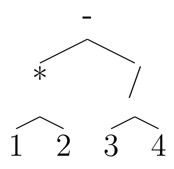
(c) $1 * 2 - 3/4$
Solution:



□

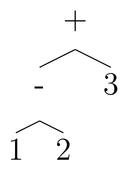
6. Draw an abstract syntax tree for the following strings:

(a) $1 * 2 - (3/4)$
Solution:



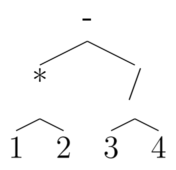
□

(b) $1 - 2 + 3$
Solution:



□

(c) $1 * 2 - 3/4$
Solution:



□

5 Operational Semantics

This problem asks you about operational semantics for a simple language with assignment and addition. The expressions of this language are given by the grammar

$$e ::= n \mid x \mid x := e \mid e + e$$

where n can be any number $(0, 1, 2, 3, \ldots)$. We can define the operational semantics with respect to a function $\sigma : \ Var \ \rightarrow \mathcal{N}$, where Var is the set of variables that may appear in expressions and $\mathcal{N}$ is the set of numbers that can be values of variables. To have some reasonable terminology, we will call σ a store and a pair e, σ a state.

When a program executes, we hope to reach a final state e, σ where e is a number, which we will call a value.

1. Let's look at arithmetic expressions. Two rules for evaluating summands of a sum are

$$\frac{e_1, \sigma \ \longrightarrow \ e_1', \sigma'}{e_1 + e_2, \sigma \ \longrightarrow \ e_1' + e_2, \sigma'}$$

$$\frac{e_2, \sigma \ \longrightarrow \ e_2', \sigma'}{n + e_2, \sigma \ \longrightarrow \ n + e_2', \sigma'}$$

We assume that it is a single execution step to add two numbers, so we have the rule

$$\frac{n, m, p \text{ are numbers with } n + m = p}{n + m, \sigma \quad \longrightarrow \quad p, \sigma}$$

The value of a variable depends on the store, as specified by this evaluation rule

$$x, \sigma \quad \longrightarrow \quad \sigma(x), \sigma$$

Show how these rules let you evaluate an expression $x+y$ to a sum of numbers. Assume σ is a store with $\sigma(x) = 2$ and $\sigma(y) = 3$. Write your answer as an execution sequence of the form below, with an explanation.

$$x + y, \sigma \longrightarrow \square + \square, \sigma \longrightarrow \square + \square, \sigma \longrightarrow \square, \sigma$$

Solution:

$$x + y, \sigma \longrightarrow 2 + y, \sigma \longrightarrow 2 + 3, \sigma \longrightarrow 5, \sigma$$

That is, first we evaluate x by looking it up from the store to get 2. Then we evaluate y in a similar manner and finally we evaluate $2 + 3$ to get 5. $\square$

2. *Put* is the function on stores with $Put(\sigma, x, n) = \sigma'$ with $\sigma'(x) = n$ and $\sigma'(y) = \sigma(y)$ for all variables y other than x. Using *Put*, the execution rule for assignment can be written

$$x := n, \sigma \quad \longrightarrow \quad n, Put(\sigma, x, n)$$

In this particular language, an assignment changes the store, as usual. In addition, as you can see from the operational semantics of assignment, an assignment is an expression whose value is the value assigned.

If we have an assignment with an expression that has not been evaluated to a number, then we can use this evaluation rule:

$$\frac{e, \sigma \quad \longrightarrow \quad e', \sigma'}{x := e, \sigma \quad \longrightarrow \quad x := e', \sigma'}$$

Combining these rules with the rules from part a, show how to execute $x := x + 3, \sigma$ when σ is a store with $\sigma(x) = 1$. Write your answer as an execution sequence of the form below, with an explanation.

$$x := x + 3, \sigma \longrightarrow x := \square + \square, \sigma \longrightarrow x := \square, \sigma \longrightarrow \square, \square$$

Solution:

$$x := x + 3, \sigma \longrightarrow x := 1 + 3, \sigma \longrightarrow x := 4, \sigma \longrightarrow 4, Put(\sigma, x, 4)$$

That is, first we lookup x from the store to get 1, then we evaluate $1+3$ and get 4. We then proceed to evaluate the assignment statement which in our case means updating the store with $x = 4$. $\square$

3. Show how to execute $(x := 3) + x, \sigma$ in the same level of detail, including an explanation.

   Solution:

$$(x := 3) + x, \sigma \longrightarrow 3 + x, Put(\sigma, x, 3) \longrightarrow 3 + 3, Put(\sigma, x, 3) \longrightarrow 6, Put(\sigma, x, 3)$$

   That is, first we evaluate the assignment statement (as the first rule for addition dictates). This means, we have to update the store with $x = 3$, this returns 3. Therefore, we then proceed to evaluate $3 + x$ with the newly updated store, which gives us 6. $\square$

4. Show how to execute $x := (x := x + 3) + (x := x + 5), \sigma$ when σ is a store with $\sigma(x) = 1$ in the same level of detail, including an explanation. *Solution:*

$$\begin{aligned}
&x := (x := x + 3) + (x := x + 5), \sigma \longrightarrow x := (x := 1 + 3) + (x := x + 5), \sigma \\
&\quad \longrightarrow x := (x := 4) + (x := x + 5), \sigma \longrightarrow x := 4 + (x := x + 5), \alpha \\
&\quad \longrightarrow x := 4 + (x := 4 + 5), \alpha \longrightarrow x := 4 + (x := 9), \alpha \\
&\quad \longrightarrow x := 4 + 9, \gamma \longrightarrow x := 13, \gamma \\
&\quad \longrightarrow 13, \omega
\end{aligned}$$

   Where :

$$\begin{aligned}
\alpha &= Put(\sigma, x, 4) \\
\gamma &= Put(\alpha, x, 9) \\
\omega &= Put(\gamma, x, 13)
\end{aligned}$$

   That is, as the addition rule dictates, we first evaluate the left hand side of the addition, namely $(x := x + 3)$. This leads us to look up the value of x which we find to be 1. We add that to 3 and which evaluates to 4. To fully evaluate the expression then, we update the store with $x = 4$ and return it. Next, we evaluate the left hand side of the addition $x := x + 5$. But now, looking up x returns 4, so $x + 5 = 9$. As before, we fully evaluate the assignment, updating the store with $x = 9$ and returning it. Therefore, the last operation is to update the store with $x = 13$, and returning 13, which we do. We update the store three times, and so we have 4 different stores during the lifetime of our execution. That is σ, the initial store, and α, γ, ω. $\square$