CS 515 Programming Languages and Compilers I

Problem Set 1

Sample Solution

1 Problem — Regular Expressions and Finite State Machines

1. You want to define identifiers in a new programming language using a regular expression. An identifiers in your programming language has to satisfy all of the following properties:

   - The first symbol must not be a special symbol; all other symbols may be letters, digits, or other special symbols
   - An identifier contains at most two special symbols
   - If an identifier contains two special symbols, the second special symbol must be followed by any positive number (greater than 0) of digits. Letters must not follow the second special symbol.

   Here is an example for and identifier in the language: $Hello1\$ag34ain\_32$. The following strings are not identifiers in our language: $\$Hello1$ and 1_what_is3.

   (a) Give a regular expression for identifiers using the following regular expressions as part of your answer:

   - $letter \rightarrow (a \mid b \mid c \mid ... \mid z \mid A \mid B \mid C \mid ... \mid Z)$
   - $digit \rightarrow (0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)$
   - $special \rightarrow (\$ \mid \_ \mid \#)$

   ```
   {\bf Answer}:

     (letter | digit)* |
     (letter | digit)+ special (letter | digit)* |
     (letter | digit)+ special (letter | digit)* special digit+
   ```

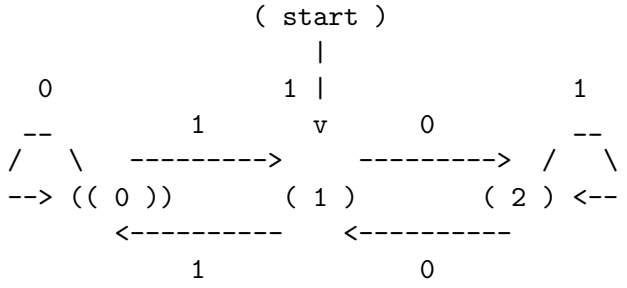
   (b) Construct the transition diagram of a nondeterministic finite automaton (NFA) **without** ϵ transitions that recognizes the language described by your regular expression for identifiers. Use the basic regular expressions *letter*, *digit*, and *special* to mark the edges in the transition diagram. Don't forget to identify the start state, and final state(s).

2. Give a DFA for the following languages over the aphabet {0, 1}:

   The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 3, i.e., <bitstring> mod 3 = 0.

   The challenge with constructing a DFA is to decide what information each state should represent. Here, each state represents the modulo value 0, 1, and 2, respectively. The states are shown as (0), (1), and (2) below. State (start) is the start state. This is needed to ensure that the binary string begins with a 1. (0) is the accepting state and thereforemarked as ((0)).

   For the transition, assume you are in state x. Then reading a 0 means that you need to make a transition to state $(2 * x)$ modulo 3, and reading a 1 means a transition to state $(2 * x + 1)$ modulo 3. If after reading the entire string you end up in state ((0)), the modulo 3 of the binary number represented by the string is zero.



2 Problem — Context-Free Languages

Are the following languages context-free or not? If yes, specify a context-free grammar in BNF notation that generates the language. If not, give an **informal** argument.

All of these languages are context free. Sample sets of rules are given, but other rules may also work.

1. { $a^n b^m c^o$ | m > n $\geq$ 0, o > 0} , with alphabet Σ = {a, b, c}

   $< S >::=< A >< B >< C >$

   $< A >::= a < A > b \mid \epsilon$

   $< B >::= b < B > \mid b$

   $< C >::= c < C > \mid c$

2. { $a^n b^{2n}$ | n $\geq$ 0 }, with alphabet Σ = {a, b}

   $< S >::= a < S > bb \mid \epsilon$

3. { ww^R | w $\in \Sigma^*$ and w^R is w in reverse }, with alphabet Σ ={a, b}

   $< S >::= a < S > a \mid b < S > b \mid \epsilon$

4. { $a^n b^m c^m d^n$ | n $\geq$ 0, m $\geq$ 0 }, with alphabet Σ = {a, b, c, d}
   $< S >::= a < S > d \mid < A > \mid \epsilon$
   $< A >::= b < A > c \mid \epsilon$
5. { w | w has no more than 5 symbols}, with alphabet Σ={a, b}
   $< S >::=< A >< A >< A >< A >< A >$
   $< A >::= a \mid b \mid \epsilon$

3 Problem — Derivation, Parse Tree, Ambiguity, Precedence & Associativity

A language that is a subset of the language of propositional logic may be defined as follows:

<start> ::= <expr>

<expr> :: = <expr> $\vee$ <expr> |
<expr> $\wedge$ <expr> |
<expr> $\rightarrow$ <expr> |
<const> | <var>

<const> :: = true | false

<var> :: = a | b | c | . . . | z

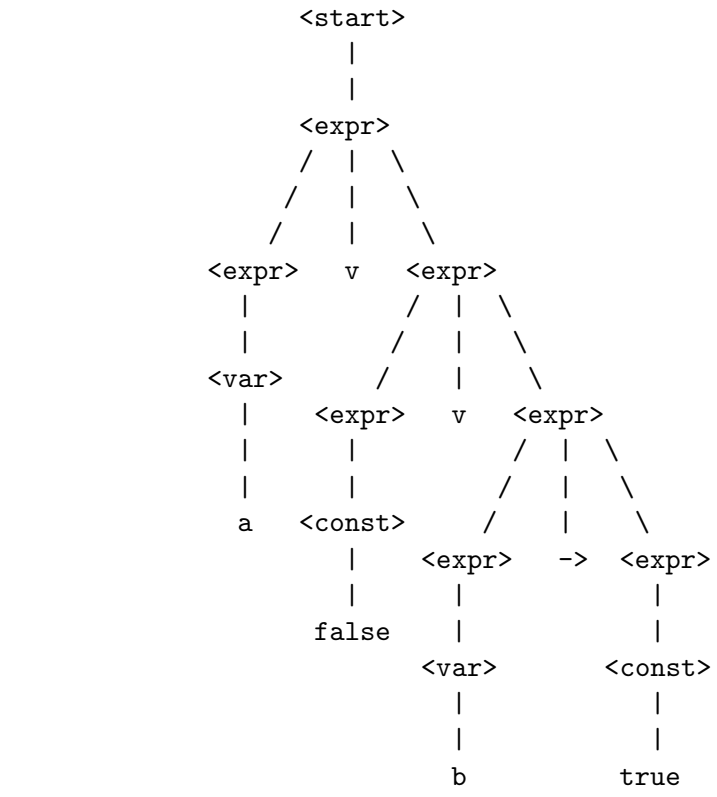
1. Give a leftmost and a rightmost derivation for the sentence
   a $\vee$ false $\vee$ b $\rightarrow$ true.

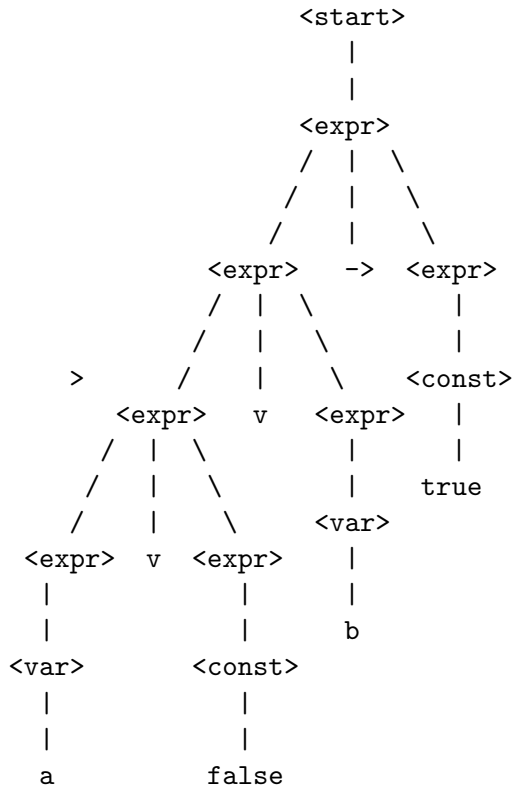
<start>
$\Rightarrow_{LM}$ <expr>
$\Rightarrow_{LM}$ <expr> $\vee$ <expr>
$\Rightarrow_{LM}$ <var> $\vee$ <expr>
$\Rightarrow_{LM}$ a $\vee$ <expr>
$\Rightarrow_{LM}$ a $\vee$ <expr> $\vee$ <expr>
$\Rightarrow_{LM}$ a $\vee$ <const> $\vee$ <expr>
$\Rightarrow_{LM}$ a $\vee$ false $\vee$ <expr>
$\Rightarrow_{LM}$ a $\vee$ false $\vee$ <expr> $\rightarrow$ <expr>
$\Rightarrow_{LM}$ a $\vee$ false $\vee$ <var> $\rightarrow$ <expr>
$\Rightarrow_{LM}$ a $\vee$ false $\vee$ b $\rightarrow$ <expr>
$\Rightarrow_{LM}$ a $\vee$ false $\vee$ b $\rightarrow$ <const>
$\Rightarrow_{LM}$ a $\vee$ false $\vee$ b $\rightarrow$ true

<start>
$\Rightarrow_{RM}$ <expr>
$\Rightarrow_{RM}$ <expr> $\rightarrow$ <expr>

$\Rightarrow_{RM}$ <expr> $\rightarrow$ <const>
$\Rightarrow_{RM}$ <expr> $\rightarrow$ true
$\Rightarrow_{RM}$ <expr> $\vee$ <expr> $\rightarrow$ true
$\Rightarrow_{RM}$ <expr> $\vee$ <var> $\rightarrow$ true
$\Rightarrow_{RM}$ <expr> $\vee$ b $\rightarrow$ true
$\Rightarrow_{RM}$ <expr> $\vee$ <expr> $\vee$ b $\rightarrow$ true
$\Rightarrow_{RM}$ <expr> $\vee$ <const> $\vee$ b $\rightarrow$ true
$\Rightarrow_{RM}$ <expr> $\vee$ false $\vee$ b $\rightarrow$ true
$\Rightarrow_{RM}$ <var> $\vee$ false $\vee$ b $\rightarrow$ true
$\Rightarrow_{RM}$ a $\vee$ false $\vee$ b $\rightarrow$ true

2. Show the corresponding parse trees for the derivations





3. Show that the above grammar is ambiguous.

   Since the sentence "a$\vee$false$\vee$b $\rightarrow$ true" has multiple possible parse trees, the grammar is ambiguous. Many other sentences also demonstrate this.

4. Give an unambiguous grammar for the same language that enforces the following precedence and associativity:

   - $\wedge$ has highest precedence (binds strongest), followed by $\vee$, and then $\rightarrow$
   - $\wedge$ and $\vee$ are left associative, and $\rightarrow$ is right associative

   $< start >::=< expr >$

   $< expr >::=< orexpr > \rightarrow < expr > \mid < orexpr >$

   $< orexpr >::=< orexpr > \vee < andexpr > \mid < andexpr >$

   $< andexpr >::=< andexpr > \wedge < term > \mid < term >$

   $< term >::=< const > \mid < var >$

   $< const >::= true \mid false$

   $< var >::= a \mid b \mid c \mid \ldots \mid z$