CS 352 – Compilers: Principles and Practice
Mid-term Examination, 10/14/2002

Instructions: Read carefully through the whole exam first and plan your time. Note the relative weight of each question and part (as a percentage of the score for the whole exam). The total points is 100 (your grade will be the percentage of your answers that are correct).

This exam is **closed book, closed notes**. You may *not* refer to any book or other materials.

You have **one hour** to complete all four (4) questions. Write your answers on this paper (use both sides if necessary).

Name:

Student Number:

Signature

1. (Scanning; 25%) C integer literals are formed in the following way:

- *Octal* literals consist of a leading `0` followed by zero or more octal digits (`0` through `7`)
- *Hexadecimal* literals consist of a leading `0` followed by `x` or `X` followed by one or more hexadecimal digits ((`0` through `9`, `a` through `f`, or `A` through `F`)
- *Decimal* literals consist of one non-zero decimal digit (`1` through `9`) followed by zero or more decimal digits (`0` through `9`).

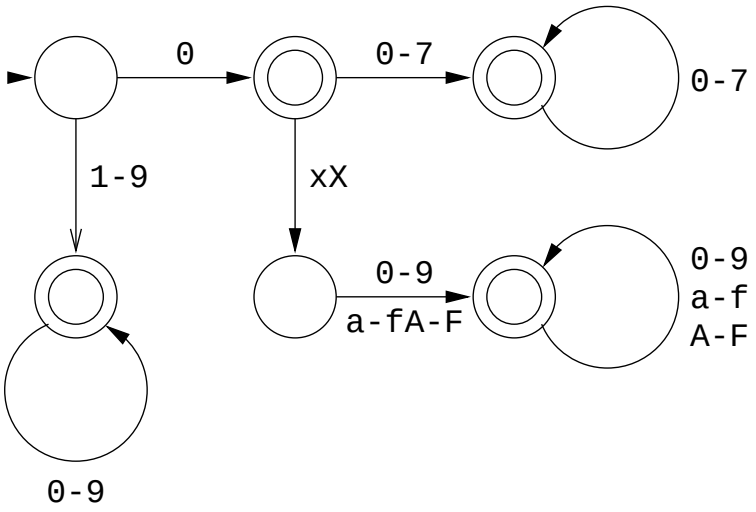
(a) (10%) Give a regular expression for C integer literals.

Answer:

$$0(0-7)^* \mid 0(x|X)(0-9|a-f|A-F)^+ \mid (1-9)(0-9)^*$$

(b) (15%) By inspection (I don't expect you to go through any NFA-DFA construction to do this), draw the state diagram of a DFA (not an NFA!) for this literal form.

Answer:

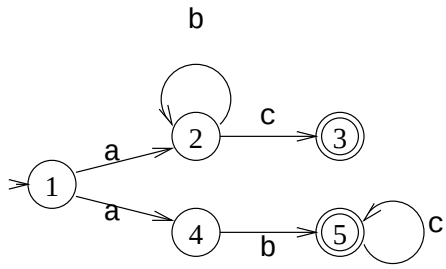


2. (Finite automata; 35%)

(a) (10%) Draw an NFA that recognizes the same language as defined by the following regular expression:

$$(ab^*c) \mid (abc^*)$$

Answer:



(b) (5%) Show the sequence of moves made by your NFA in accepting the input string *abcc*

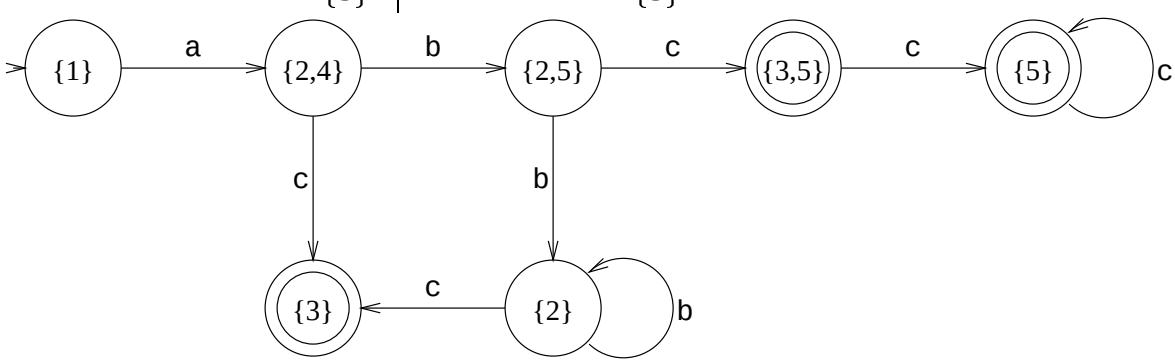
Answer:

$$(1, abcc) \vdash (4, bcc) \vdash (5, cc) \vdash (5, c) \vdash (5, \varepsilon)$$

(c) (15%) Using the subset construction, convert the NFA into a DFA. Optimize the resulting DFA by merging equivalent states (if any).

Answer:

	a	b	c
{1}	{2,4}	-	-
{2,4}	-	{2,5}	{3}
{2,5}	-	{2}	{3,5}
{3}	-	-	-
{2}	- {2}	{3}	
{3,5}	-	-	{5}
{5}	-	-	{5}



States {3,5} and {5} can be merged.

(d) (5%) Show the sequence of moves made by your DFA in accepting the input string *abcc*

Answer:

$$(\{1\}, abcc) \vdash (\{2,4\}, bcc) \vdash (\{2,5\}, cc) \vdash (\{3,5\}, c) \vdash (\{5\}, \varepsilon)$$

3. (Parsing, context free grammars, LL parsing; 30%) Consider the following simple context free grammars:

Grammar G_1	Grammar G_2	Grammar G_3
$S \rightarrow A\$$	$S \rightarrow A\$$	$S \rightarrow A\$$
$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$
$A \rightarrow bbA$	$A \rightarrow bAb$	$A \rightarrow Abb$

The start symbols are S, the non-terminals are S and A, and the terminal symbols are b and \$. Note that these grammars all generate the same language: strings consisting of even numbers of b symbols (including 0 of them).

(a) (15%) Demonstrate that G_1 is LL(1) by constructing its LL(1) parse table.

Answer:

$\text{FIRST}(G) = \{b, \$\}$
$\text{FIRST}(A) = \{b, \varepsilon\}$

$\text{FOLLOW}(G) = \{\varepsilon\}$
$\text{FOLLOW}(A) = \{\$\}$

LL(1) parse table

	b	\$
G	$G \rightarrow A\$$	$G \rightarrow A\$$
A	$A \rightarrow bbA$	$A \rightarrow \varepsilon$

Since there are no multiply defined entries in its LL(1) parse table G_1 is LL(1).

(b) (5%) Is G_2 LL(1)? You need not construct its LL(1) parse table, but may argue from other properties.

Answer:

G_2 is not LL(1) since b predicts both $A \rightarrow \varepsilon$ and $A \rightarrow bAb$.

(c) (5%) Is G_3 LL(1)? Again, you need not construct its LL(1) parse table, but may argue from other properties.

Answer:

G_3 is not LL(1) since it is left-recursive.

(d) (5%) Of the *language* classes we have discussed in the course, what is the *smallest* category into which $L(G_1)$ fits? Justify your answer?

Answer:

The language is regular and can be defined by the regular expression $(bb)^*$.

4. (Symbol tables; 10%) Compared to the full Java programming language, MiniJava has a relatively simple type system. Suppose MiniJava is extended to support overloading as in full Java (i.e., allowing procedures with the same name but different argument lists and result types). Describe the modifications (if any) you would need to make to the symbol table to support this feature. Based on these modifications (if any) describe how you would perform type checking. Use an example in your answer.

   Answer:

   Both the parameter and return types must be used to distinguish procedures, and incorporated into the symbol table lookup mechanism. These could form part of the hash for lookup in a hash table. Type checking must use the types of the actual parameters to lookup the procedure in the symbol table.