ECE351 Sample Questions (Third Set)

Question 1: True/False questions (20 points)

Please specify if the following assertions are True or False, accompanied with appropriate reasoning. Each sub-question is worth 2 points. To get full points you must supply the correct reason for your answer.

The alphabet for the languages specified below will typically be $\{a, b\}$, unless otherwise specified.

1. Consider the class of languages that are context-free but not regular. This class is closed under intersection, i.e., the intersection of two such languages is context-free but not regular.

   Answer: False. Consider the two context-free languages $a^n b^n$ and $b^n a^n$ (where $n \geq 0$), that are not regular. It is easy to see that they are not closed under intersection, i.e., the resultant language contains only the empty string which is certainly regular.

2. Associativity rules help overcome left-recursion in parsers for context-free grammars.

   Answer: False.

3. The following grammar represent the language of all strings with equal number of a's and b's.

   $$a^n b^n \bigcup b^n a^n$$

   Answer: False. Consider the string abab.

4. Context-free languages are closed under concatenation operation.

   Answer: True. Suppose we have grammars for two languages, with start symbols S_1 and S_2. Rename variables as needed to ensure that the two grammars don't share any variables. Then construct a grammar for the union of the languages, with start symbol N, by taking all the rules from both grammars and adding a new rule $N \rightarrow S_1 S_2$.

5. Every finite language has a corresponding NFA that has no "looping" transitions.

   Answer: True. For every finite language F, you can construct an NFA such that each string in F is accepted by a sequence of states with no "looping" transitions.

6. Consider an NFA F such that the ϵ-closure from the start state of F (i.e., set of all states that can be visited by ϵ-transitions from the start state) contains an accepting state. Let $L(F)$ represent the language accepted by F. The intersection of $L(F)$ with the language $\{a^n b^n \mid n > 0\}$ contains the empty string ϵ.

   Answer: False.

7. The use of canaries in call-stacks ensure that buffer overflow attacks can never be launched.

   Answer: False. Canaries can only help detect whether an attack occured. They cannot prevent it.

8. It is known that the register-allocation problem is NP-hard.

   Answer: True.

9. In modern garbage collection mark-and-sweep algorithms, unreachable memory is guaranteed to be garbage.

   Answer: True.

10. The following grammar is not left-recursive.

$$A \rightarrow B\alpha$$
$$B \rightarrow AC$$

Answer: False.

Question 2: Short answer questions (28 points)

For each of the questions below, provide a short correct and complete answer. Each question carries a maximum of 7 points.

1. Use the local optimization discussed in class (algebraic simplification, dead code elimination) to optimally simplify the code below. (Assume only 'g' is alive outside the given basic block.) You must show your steps to get full points:

```
b := 3
c := x
d := c * c
e := b * 2
f := e + b
g := e * f
```

Answer: The optimized code is:

```
g := 54
```

2. Propose a simple local optimization technique to simplify the code below. Note that you must be able to implement your technique as an optimization pass in a compiler. Explain your answer.

```
X := B;
if( B + X == 7) {
Y := f(B);
}
else{
X := 4;
Y := f(X);
}
return Y;
```

Answer: Detect if an expression is always even. If your local optimization pass can detect that, then the if-conditional will always evaluate to FALSE, and the if-else will be simplified to just f(4); The resultant simplified code is "return f(4);"

3. Short questions on bottom-up and top-down parsing

(a) Briefly describe why bottom-up parsers are typically considered as more powerful than top-down parsers.

(b) Does the following grammar, as given, have a recursive-descent parser?

$$S \rightarrow Sa \mid \epsilon$$

Answer: The answers are:

(a) For bottom-up parsers, the grammar doesn't need to be left-factored, and supports a larger class of CFLs than top-down.

(b) No, the above-mentioned grammar has a left recursion.

4. Prove or disprove the following identity over regular expressions r and s:

$$(rs + r)^* r = r(sr + r)^*$$

Answer: Transform the LHS as follows to obtain the RHS (crucial rule: $(rs)^* r \equiv r(sr)^*$).

$$(rs + r)^* r \equiv (rs + r\epsilon)^* r \equiv (r(s + \epsilon))^* r \equiv r(sr + r)^*$$

Question 3: Liveness Analysis (32 points)

Give detailed answers to the following questions. Each question is worth a maximum of 8 points. Justify your answers.

In class we defined liveness as follows:

We say that a variable X is live at a statement S if there is a subsequent statement S' where X is used and there are no intervening paths that assign values to X different from the value at S.

1. Would the variable be considered live at X, if X is assigned different values along different paths from S to S'?

   Answer:

   If X is assigned different values along different paths from S to S', then it is not live at S anymore.

2. What if X is assigned the same value in every path from S to S', but this value is different from the value at S?

   Answer: Again, in this case X is not live at S.

3. What if X is assigned the same value along all paths from S to S' and this value happens to be the value assigned at S, but the static analysis implemented in the compiler is not powerful enough to detect this fact?

   Answer: The analysis is conservative, i.e., if it can't determine the value along even a single path from S to S' then it will assume that the value is different from the value at S. Thus, the analysis will infer that X is not live at S.

4. What would the analysis conclude about X if the compiler detects the fact that all paths from S to S' assign the same value to X and that this value happens to be the same as the value assigned at S.

   Answer: This satisfies the definition of liveness, and hence the compiler will correctly conclude that X is live at S.

Question 4: Register Allocation (20 points)

Give detailed answers to the following questions. Each question is worth a maximum of 5 points. Justify your answers.

1. Why does the register allocation algorithm perform a static liveness analysis step?

   Answer: In order to determine which temporaries are simultaneously live at a particular statement in the program during any run of the program. This enables the algorithm to determine, in the worst-case, the maximum number of different registers are needed simultaneously at that particular statement in the program for the purposes of register allocation.

2. Under what conditions would a register allocation alogrithm spill, assuming the hardware has a fixed number K of registers?

   Answer: If the register interference graph (RIG) is not K-colorable.

3. You may be given a code snippet and asked to perform liveness analysis for the purposes of register allocation.

4. You may be given a code snippet whose register-interference graph may not be K-colorable. You may be asked to spill a register and recompute the liveness analysis and draw the RIG.