Fall 2012 CSE Qualifying Exam
Core Subjects

September 29, 2012

Architecture

No one took this subject in Fall 2012.

Compilers

1. **LR-Parsing.**

   (a) Give definitions of FIRST(α) and FOLLOW(X).

   (b) Consider the following grammar G:

$$\begin{array}{rcl} S' & \rightarrow & S \\ S & \rightarrow & V \text{ `=' } E \\ S & \rightarrow & E \\ V & \rightarrow & \textbf{id} \\ V & \rightarrow & \text{`*' } E \\ E & \rightarrow & V \\ E & \rightarrow & \text{`\&' } V \end{array}$$

   For the grammar G above, generate the first six LR(1) sets of items.

   (c) Using the partial Sets-of-Items constructed in part (b) construct as much as you can of the parse table.

   (d) Describe in detail how the LR Parsing Algorithm will proceed when the state is s, the next token is t and the stack contents is $X_0, X_1, \ldots, X_{top-1}, X_{top}$.

2. **Semantic Rules.** Consider the usual bottom-up grammar for arithmetic expressions built from integer constants with + and $*$ and parentheses:

$$\begin{array}{rcl} \langle expr \rangle & ::= & \langle expr \rangle_1 + \langle term \rangle \\ \langle expr \rangle & ::= & \langle term \rangle \\ \langle term \rangle & ::= & \langle term \rangle_1 * \langle factor \rangle \\ \langle term \rangle & ::= & \langle factor \rangle \\ \langle factor \rangle & ::= & (\ \langle expr \rangle\ ) \\ \langle factor \rangle & ::= & \textbf{intconst} \end{array}$$

   If E is any expression given by this grammar, we define the *full operator parenthesization* (FOP) of E to be the same as E except that each proper subexpression of E that involves an operator is surrounded by exactly one pair of matching parentheses. Any other extraneous parentheses, such as those surrounding other parentheses, integer constants, or E itself, are not included. Thus we have the following examples:

E	FOP of E
2	2
$2+3$	$2+3$
$(2)+3$	$2+3$
$(2+3)$	$2+3$
$2+3+4$	$(2+3)+4$
$2+((((3+4))))$	$2+(3+4)$
$2+3*4$	$2+(3*4)$
$(2+3)*(4)$	$(2+3)*4$
$2*3*4$	$(2*3)*4$

Add semantic rules to the grammar above to compute the FOP of an expression as a string attribute $\langle expr \rangle.fop$. The *only* string-type expressions or operations you are allowed are the following:

- string literals (in double quotes),
- **intconst** *.str*, which you should assume is the string representation of the corresponding integer constant given by the lexical analyzer,
- the *.fop* attributes of any other nonterminals,
- the infix concatenation operator $\|$, e.g., $s_1 \| s_2 \| \cdots \| s_n$ returns the concatenation of strings $s_1, s_2, \ldots, s_n$,
- string assignment

You may define any other non-string attributes of nonterminals that you find helpful, and you may also use local variables within rules, but you are not allowed any global variables.

3. **Control Flow and Liveness Analysis.** The following fragment of 3-address code was produced by a nonoptimizing compiler:

```
 1  start:  x := 1
 2          y := 1
 3          sum := y
 4          sum := sum + 1
 5  loop1:  if x = n then goto out1
 6  loop2:  if y = x then goto out2
 7          t1 := a[y]
 8          t2 := y * t1
 9          t3 := t1 + x
10          if t3 < n then goto skip
11          t3 := t3 - n
12  skip:   sum := sum + t3
13          y := y + 1
14          goto loop2
15          goto loop2
16  out2:   a[x] := sum
17          sum := x + 1
18          x := x + 1
19          goto loop1
20  out1:   x := x - 1
21          t1 := a[x]
22          print t1
23          if x = 0 then goto out
24          goto out1
25  out:    ...
```

Assume that there are no entry points into the code from outside other than at `start`.

(a) (20% credit) Decompose the code into basic blocks $B_1, B_2, \ldots$, giving a range of line numbers for each.

(b) (20% credit) Draw the control flow graph, and describe any unreachable code.

(c) (40% credit) Fill in a 25-row table listing which variables are live at which control points. Treat the array `a` as a single variable. Assume that `n` and `sum` are the only live variables immediately before line 25. Your table should look like this:

Before line	**Live variables**
1	...
2	...
3	...
...	...

(d) (20% credit) Describe any simplifying transformations that can be performed on the code (i.e., transformations that preserve the semantics but reduce (i) the complexity of an instruction, (ii) the number of instructions, (iii) the number of branches, or (iv) the number of variables).

Algorithms

1. (A really lopsided recurrence.) Let some positive-valued function $T(n)$ satisfy the following recurrence for all sufficiently large integers n:

$$T(n) = T(\lceil n/2 \rceil) + T(n-1) + 1 \ .$$

Show using the substitution method that $T(n) = \Omega(n^k)$ for *any* constant k (no matter how large) as $n \to \infty$. Note that this is only an asymptotic *lower* bound on $T(n)$. You are only required to show the inductive step; you need not show how base cases are handled.

[Hint: You may use the fact (without proof) that for any $k \geq 0$,

$$\lim_{n\to\infty} \left(\frac{n-1}{n}\right)^k = 1 \ .]$$

2. Describe an algorithm that takes an array $A[1 \ldots n]$ of integers as input and determines whether or not there exists an integer a such that the elements of A form the set $\{a, a+1, \ldots, a+n-1\}$ in some order. Your algorithm should run in time $O(n)$.

You are not required to prove that your algorithm is correct or that it runs within the specified time bound.

3. The decision problem 1/4-PARTITION is defined as follows: Given a list $\langle a_1, a_2, \ldots, a_n \rangle$ of positive integers, does there exist a subset $S \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S} a_i = \frac{1}{4} \sum_{i=1}^{n} a_i \ ?$$

Give a direct, explicit, polynomial reduction from PARTITION to 1/4-PARTITION (thus showing that 1/4-PARTITION is NP-hard). [Recall that the PARTITION problem is the same except that the fraction on the right-hand side is 1/2 instead of 1/4. Also note that there is a trivial reduction from SUBSET-SUM to 1/4-PARTITION, but we are not asking for this.]

Theory

1. For any language $L \subseteq \{0,1\}^*$ of binary strings, define the *left set of* L as follows:

$$\text{left}(L) := \{x \in \{0,1\}^* \mid (\exists y \in L)[|x| = |y| \;\&\; x \leq y]\} \,,$$

   where $\leq$ is the standard numerical ordering restricted to binary strings of the same length.

   Show that if L is regular, then left(L) is also regular.

2. In this question we assume that every Turing machine has a single, one-way infinite tape with cells labeled $1, 2, 3, \ldots$ starting from the left end, and the machine's tape head is scanning cell 1 at the start of a computation.

   Let f be a function such that for every TM M and input string w, $f(\langle M, w\rangle)$ equals the maximum $i \geq 1$ such that M's tape head scans cell i at some point in M's computation on input w, provided such an i exists, and $f(\langle M, w\rangle) = 0$ if no such i exists. Show that f is not computable.

3. Recall the definition of left(L) in Problem 1.

   (a) (50%) Show for any $L \subseteq \{0,1\}^*$ that if $L \in$ NP, then left$(L) \in$ NP.

   (b) (50%) Describe an NP-complete language $A \subseteq \{0,1\}^*$ such that left$(A) \in$ P.