Final Exam
CS3300

Maximum marks = 80, Time: 3.00 hrs

17-Nov-2016

Read all the instructions and questions carefully. You can make any reasonable assumptions that you think are necessary; but state them clearly. There are total five questions, each sixteen marks worth. You will need approximately 30-35 minutes for answering a sixteen marks question (plan your time accordingly). For questions with sub-parts, the division for the sub-parts are given in square brackets.

You will get an answer sheet with 12 pages (if you get a answer sheet with fewer pages then ask for a replacement sheet). Leave the first page empty. Start each question on a new page. Think about the question before you start writing and write briefly. **The answer for any question (including all the sub-parts) should NOT cross more than two pages.** If the answer for any question is spanning more than two pages, we will strictly ignore the spill-over text. If you scratch/cross some part of the answer, you can use space from the next page.

1. [16] **Parsing**:
(A) Write a grammar to parse all possible expressions in C that involve (only) variables of pointer datatype [6]. Assume that you don't have to handle pointer-arithmetic, array-subscript notation, addressof operator, or brackets. Assume that only available operators are * and =.

Note I: In C language assignment (of the form a=b) is also an expression.

Note II: In C language `**x` is a valid expression.

Check if the grammar is SLR(1)? [8]

Multiple choice questions [1+1]:

(B) Which of the following derivations does a bottom-up parser use while parsing an input string? The input is assumed to be scanned in left to right order.

(a) Leftmost derivation
(b) Leftmost derivation in reverse
(c) Rightmost derivation
(d) Rightmost derivation in reverse
(e) None of the above.

(C) Which of the following statements is false?

(a) An unambiguous grammar has same leftmost and rightmost derivation
(b) An LL(1) parser is a top-down parser
(c) LALR is more powerful than SLR
(d) An ambiguous grammar can never be LR(k) for any k
(e) None of the above.

2. [16] **IR Generation**:

(A) Recall the semantic actions discussed in the class to generate IR for booleans, conditional statements, and the discussion about implementing objects. Write similar rules to translate statements generated by the grammar shown on the right hand side to three address codes [10].

Note I: A object of a derived class is an instance of both the derived class and the parent class.

Note II: Assume that the IR has an instruction `alloc`; for example, `t=alloc`(4) allocates 4 bytes of memory and stores in `t`.

```
P -> C*; M // sequence of class decl
           // followed by main func.
C -> class Id; | class Id extends Id;
M -> S*
S -> Id = new Id
S -> Id = Id instanceof Id
S -> return Id
S -> if (Id) S* else S*
```

Use your stated rules to generate IR for the code shown in the right hand side [4].

```
class A;
class B extends A;
class C;

x = new A
y = new B

p = x instanceof A
if (p)
    q = y instanceof C
    return q
else
    r = y instanceof A
    return r
```

Multiple choice questions [1+1]:

(B) Choose the right answer:

(a) A compiler has a fixed number of IR instructions.

(b) Every compiler for a given language must have the same set of IR instructions.

(c) IR instructions do not carry type information.

(d) All IRs must have the same expressive power.

(e) None of the above.

(C) Choose the right answer:

(a) Java and C cannot be compiled to the same IR.

(b) Assembly language can be an IR.

(c) Java bytecode cannot be an IR.

(d) No IR has representation for loops.

(e) All of the above

3. [16] **Control flow**:

(A) Let us define an extended basic block (EBB) as a maximal sequence of single entry multiple exit nodes in the control-flow-graph (CFG) such that except the first-node in the EBB (leader) no other node can be a "join" block. Naturally, an EBB can have one or more basic blocks. And EBB identification starts with basic block identification.

Give an algorithm to compute all the EBBs for a function (= list of instructions) [8]. Assume the input to your algorithm is a CFG. For the code shown on right, draw the CFG [3], and super impose the EBBs (using dashed boxes) on the CFG [3].

```
void bar(){
  i=1;
  L0: if (i > n) goto L1;
  j=1;
  goto L2;
  L1: j=2;
  L2: k = 3;
  if (n > 0) goto L3
  L4: p++
  if (p > 10) return;
  goto L4
  L3: if (m > 0) return;
  goto L0
}
```

Multiple choice questions [1+1]:

(B) Choose the wrong answer:

(a) Dominator information can be used to identify all loops.

(b) A given procedure cannot have different CFGs.

(c) There is no limit on the number of instructions present in a basic block.

(d) Control flow analysis can identify loops that are otherwise not obvious in the code.

(e) None of the above.

(C) Choose the right answer:

(a) A basic block can have at most two predecessors.

(b) A basic block can have at most two successors.

(c) A basic block must have a successor.

(d) It is possible to find a basic block (other than 'entry') with no predecessor.

(e) None of the above.

4. [16] **Runtime Management**:
(A) Say you are building a simulator (in Java) to simulate code written in a procedural language like C. Show the (per function call) data structures you will maintain to perform runtime stack management. [6]. Show how/when you will allocate, initialize and free the data structures. [6]

Hint I: These data-structures only concern the runtime management and nothing else. Hint II: Pay special attention to function call, function begin, and function end.

GCC has an optimization called tail-call optimization that replaces a tail call with a jump. It does so, only if the caller and callee have the same number of arguments. Give a reason. [2]

Multiple choice questions [1+1]:

(B) Choose the right answer:

(a) In Java you can pass arguments by reference.

(b) For a language like C there is no reason to separate (on the call-stack) the locals from the compiler generated temporaries.

(c) We cannot store the stack frames on the heap.

(d) The epilogue code (executed, just before the return as part of runtime management) can first restore $sp register and then load the return address.

(e) None of the above.

(C) Choose the wrong answer:

(a) All registers can be declared caller-save.

(b) All registers can be declared callee-save.

(c) It is sensible to mark some registers as neither caller-save, nor callee-save.

(d) It is sensible to mark some registers as both caller-save and callee-save.

(e) None of the above.

5. [16] **Optimizations and Register allocation**:
(A) Show two unique example (pattern) codes each, where you can apply the following optimizations: i) instruction scheduling for avoiding RAW hazard, ii) instruction scheduling for avoiding control hazard, iii) instruction scheduling to fill branch delay slot, iv) fuse jump statements, v) eliminate redundant stores. For each example, show the input - output code and conditions under which you can apply the optimization. [3+3+3+3+3].

Multiple choice questions [0.5+0.5]:

(B) Choose the right answer:

(a) The complexity of the optimum register allocation algorithm is $O(N^3)$, where N is the number of live ranges in the program.

(b) Spilling two live ranges (say A and B) can never be better compared to spilling one (say C). Assume A, B, and C are different.

(c) It is inefficient to keep dedicated registers for handling spill code.

(d) Linear scan is guaranteed to be better than Kempe's heuristic.

(e) None of the above.

(C) Choose the wrong answer:

(a) An interference graph with a K clique needs at least K registers for a spill-free register allocation.

(b) If the size of the max-clique in an interference graph is K, then we need at most K registers for spill free register allocation.

(c) The complexity of linear-scan algorithm is $O(N^3)$, where N is the number of live-ranges.

(d) The register $ra is a callee save register.

(e) None of the above.