CS3300

Quiz 2

Dept of CSE, IIT Madras

Total marks = 24

Time = 50 min

15 Oct 2019

Read the instructions and questions carefully. You can make any reasonably assumptions that you think are necessary; but state them clearly. There are total four questions (8 + 8 + 4 + 4 = 24 marks + 2 bonus marks). You will need approximately 15 minutes for answering an 8 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 8 pages (if you get a answer booklet with fewer pages then ask for a replacement). Leave the first page empty and start from Page#2. Start each question on a new page. Think about the question before you start writing and write briefly. **For any question, the answer (including the answers for 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. You mostly would NOT need any additional sheets.

1. [8+1] **Syntax Directed Translation**: Define inherited and synthesized attributes [2]. Given a production $A \to X_1 X_2 \cdots X_n$, state the order in which the inherited and synthesized attributes of A, X_1, X_2, $\ldots X_n$ may be evaluated in an L-attributed grammar [1]. Give an example grammar and semantic rules (not discussed in the slides) illustrating both inherited and synthesized attributes [Bonus 1] .

   Write a grammar to recognize positive decimal integer literals in C [1]. Use this grammar to write an SDT scheme to check if the number is divisible by six. [4]. You may not use division (or repeated subtraction for that matter).

2. [8] **Type checking and IR Generation**:

   Consider a subset of MiniJava with no loops and no conditional statements. To translate this subset, give a minimal list of instructions of 3-address-codes (TAC), with one line description for each [3]. For each such TAC instruction, give an example MiniJava code snippet (we don't require the complete program) explaining where we need that TAC instruction [2]. Assume that the IR need not have a statement for variable declarations. If you want you can also use 'bOp' as the generic binary operator.

   Give rules on how you will type-check a program written in your TAC [3].

3. [4+1] **Liveness Analysis and register allocation**

   Using a simple example code, explain `def` and `use` [1]. Say we have a program with no branch instructions then the liveness analysis algorithm can be simplified significantly. Write the simplified algorithm [3] and discuss its complexity [Bonus 1].

4. [4] **Answer True or False**

   (a) In an L-attributed grammar some attributes can be both synthesized and inherited.

   (b) Every compiler for a given language, for a given pass (for example, type checking) must use the same data-structure for the symbol table.

   (c) If MiniJava had method overloading but no method overriding, then the call resolution can happen at compile time.

   (d) LL parsers push semantic actions into the parse stack like grammar symbols.

   (e) In the procedure linkages, the prologue and epilogue code is exactly opposite of each other.

   (f) Given a program, during the discussed iterative liveness analysis algorithm, the `use` and `def` information may change, but only monotonically.

   (g) Given the set of live-in variables of a node, we cannot precisely compute its live-out variables using the set of variables used/defined at that node.

   (h) Liveness analysis cannot be performed as a forward analysis.