

# CS5363 Final Exam

- (10pts) Write a regular expression for each of the following languages.
  - The set of floating point numbers over the alphabet  $\{0-9, .\}$ . For example, strings "0.52", "35." and "100.63" are in the language, but "1.01.5", "35" (an integer number with no floating point) are not.
  - The set of strings over  $\{1,0\}$  that contain no substring 101. For example, strings "0100" and "10010" are in the language, but string "0101" is not.

- (12pts) Give a context-free grammar for a small graph description language. The terminals of your grammar include numbers (denoted by the token *NUM*), '(', ')', ';' and '→'. Each node of the graph is denoted by a number, and each edge is denoted by a '→' connecting two nodes (eg., "3→14" represents an edge from node 3 to node 14). Each graph description starts with '(', ends with ')', and includes a sequence of paths (a single node is considered a special path) separated by ';'.

The following gives an example graph description

```
( 1→2→5→1; 2 → 1)
```

Here the graph has three nodes (1, 2 and 5) and four edges ( 1→2, 2→5, 5→1, and 2 → 1). Similarly, the description "(1; 2, 3)" gives another graph, which has three nodes (1, 2 and 3) but no edges. Note that "(1;3;)" is not a valid graph description (the second ";" should not be there), neither is the description "(1→)" (missing end point of the edge).Using your grammar, write a parse tree and an abstract syntax tree for "( 1→2→5; 1→1)".

- (10pts) For the following Pascal program, draw the set of activation records that are on the stack just prior to the return from function F1. Use access links for non-local data access and use line numbers for return addresses. Draw directed arcs for control and access links. Label the values of local variables and parameters. Label each AR (activation record) with its procedure name.

```
1 program main(input, output);
2   var x : integer;
3   function F1(a : integer ) : integer;
4     begin
5       x := a + 3;
6       F1 = x;
7     end;
8   procedure P;
```

```

9   var a: integer;
10  function F2(b : integer) : integer;
11      begin
12          F2 := a + F1(b)
13      end
20  begin
21      a := 5;
22      writeln(F2(a));
23  end;
24  begin
25      P
26  End.

```

4. (8pts) Generate three-address ILOC for the following C statements.

```

a = b * c * a + b - (c - 2);
while (b < c || a < c) {
    b = b * b;
    a = a * a;
}

```

Use short-circuit evaluation for translating boolean expressions. Assume we have an infinite number of registers and that all variables can be allocated to registers without a memory address. Therefore no memory load or store operations are necessary. You can use  $ra$ ,  $rb$ ,  $rc$  to denote the registers for variables  $a$ ,  $b$ , and  $c$  respectively, and use  $r1, r2, \dots$  to denote registers allocated for temporary values. You can use the following operations in your translation, where  $r1, r2, r3$  are registers,  $num$  is an integer number, and  $L1, L2$  are labels.

```

PLUS r1, r2 => r3
MULT r1, r2 => r3
SUB  r1, r2 => r3
SUBI r1, num => r3
COMP r1, r2 => r3
COMPI r1, num => r3
CBR_LT r1 => L1, L2
JUMPI L1

```

5. (24pts) Suppose the following pseudo code is given.

```

a := b + c
b := a - d
L1: c := c + 1
if (c > d) goto L3
a := a + d
d := d * c
b := a - b

```

```

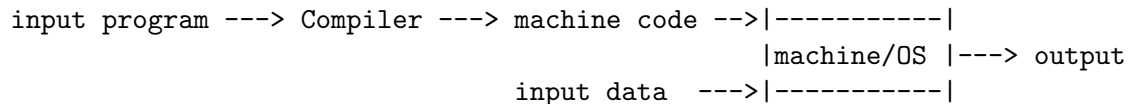
    if (b == d) goto L2
    e := b
    goto L1
L2:e := a + b
    b := d - 1
L3:k := a - e
    c := d + b

```

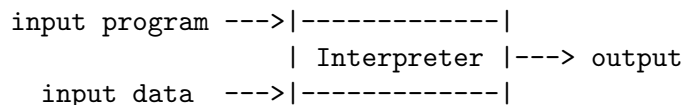
- (a) Draw its control flow graph.
  - (b) Write the data-flow analysis equation for analyzing the set of reaching variable definitions at the entry of each basic block. A variable definition, e.g., “a = 3”, can reach the entry of a basic block B if there is a control flow path from “a = 3” to block B along which there is no intervening definition to *a*. Show both the intermediate and final result of performing reaching definition analysis. Hint: label each definition site in the code with a unique identifier.
  - (c) convert the CFG into SSA form.
6. (18pts) Use a common definition to explain each of the following pairs of concepts. Briefly summarize how the two concepts in each pair differ and use diagrams or examples to illustrate the difference.

For example, the following explains the concepts “compilation and interpretation”. They are two different approaches to implement a programming language. Compilation translates the input program to machine code before evaluating the program, while interpretation evaluates the input computation directly without going through translation. The following diagrams illustrate the compilation and interpretation processes respectively.

Compilation:



Interpretation:



- (a) NFA and DFA. Give an example of each type of automata.
  - (b) Synthesized attribute and inherited attribute. Give an example context-free grammar with syntax directed definitions for evaluating both types of attributes.
  - (c) Type checking and type inference. Use an example expression to illustrate the information required for each analysis.
7. (18pts) Answer the following questions.

- (a) List three program optimizations that you've studied in class. Briefly summarize the optimization in 2-5 sentences. Specifically, what are they used for? What program analysis (i.e., what information regarding the input program) do they require?
- (b) List three components that belong to the backend of most modern compilers. Summarize each component in 2-5 sentences. Specifically, what is the functionality of each component? Specify a popular algorithm used in each component.
- (c) What are the definition(meaning) of LL(1) and LR(1) parsers? Summarize the parsing steps in 3-6 sentences for each kind of parsers. What are the respective rows, columns, and entries of a LL(1) or LR(1) parse table?