U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação | 3º YEAR
EIC0028 | Compilers | 2018/2019 – 2º Semester

Open book exam. Duration: 2h30m. ***Exame de Recurso* (second round exam)**

Students that intend to replace the grade of a single mini-test (midterm exam) must answer PART I (replacement of mini-test MT1) or PART II (replacement of mini-test MT2). In that case the duration of the exam is **1h15m**. Students enrolled in grade improvement must answer the complete exam.

PART I (10 pts)

Group 1. (6 pts)

Considering the G1 grammar below (parentheses and numbers on the right identify the productions):

Grammar G1:

$S \rightarrow w$	(2)
$S \rightarrow a\ C\ b$	(3)
$C \rightarrow \varepsilon$	(4)
$C \rightarrow S\ x\ C$	(5)

1.a) [1pt] Indicate the First and Follow sets for the grammar variables;

1.b) [1pt] Show the LL(1) parsing table. Is the grammar LL(1)?

1.c) [1pt] Is the grammar ambiguous? Justify your answer,

1.d) [1pt] Determine and show the LR(0) automaton (consider that for the LR parser, S' is the new start variable and the production $S' \rightarrow S\ \$$ (1) is added to the grammar);

1.e) [1pt] Is the grammar LR(0)? Justify your answer indicating the LR(0) parsing table corresponding to the automaton;

1.f) [1pt] Explain in which circumstances a value of k > 1 can make a non LL(1) CFG into an LL(k) CFG. If needed, give examples to support your explanations.

Group 2. (2 pts)

Consider the G2 grammar below.

Some of the Tokens for G2:

```
IDENT = [a-zA-Z][0-9a-zA-Z]*
WHILE = while
ENDWHILE = endwhile
DO = do
IF = if
ELSE = else
THEN = then
ENDIF = endif
CMP = == | < | > | <= | >= | !=
CONST = [0-9]+
OP = + | - | * | /
```

Grammar G2:

```
1.  S → (Stmt)*
2.  Stmt → While | Assign | If
3.  While → WHILE Cond DO (Stmt)* ENDWHILE
4.  Assign → Lhs = Operand (OP Operand)? ;
5.  Operand → IDENT | CONST | ArrayAcc
6.  If → IF Cond THEN Stmt (ELSE Stmt)? ENDIF
7.  Cond → Operand CMP Operand
8.  Lhs → IDENT | ArrayAcc
9.  ArrayAcc → IDENT "[" Index "]" | IDENT "[]"
10. Index → IDENT | CONST
```

2.a) [1pt] Indicate extensions to the grammar to allow typical FOR…ENDFOR and DO…WHILE loops;

2.b) [1pt] Analyze if those extensions contribute to a non LL(1) grammar and/or to an ambiguous grammar: if not, justify your answer, and if so change the new productions in order to avoid that.

Group 3. (2 pts)

3.a) [2pt] Comment the following sentence: "The only way to achieve a high-level representation with trees representing expressions according to the rules of precedence of the operations of the input language is to have the grammar of the language taking care of the precedence rules."

Open book exam. Duration: 2h30m. ***Exame de Recurso* (second round exam)**

Students that intend to replace the grade of a single mini-test (midterm exam) must answer PART I (replacement of mini-test MT1) or PART II (replacement of mini-test MT2). In that case the duration of the exam is **1h15m**. Students enrolled in grade improvement must answer the complete exam.

PART II (10 pts)

Group 4. (8 pts)

```
//in={}
1.   i = 0;
2.   m = A[0];
     do {
3.      i=i+1;
4.      x = A[i];
5.      if (x > m)
6.          m = x;
7.   } while(i<N);
// out = {m}
```

4.a) [2pt] Indicate a low-level intermediate representation (LLIR) for the section of the code in the example on the left (where N represents a 32-bit *int* constant) based on expression trees, but considering the processor with instruction set presented below, and the type and storage of the variables given by the following table.

4.b) [1pt] Considering that a new version of the processor includes an instruction able to implement the code in lines 5 and 6, show the pattern that can be used by the instruction selection algorithm (e.g., Maximal Munch) in order to have the option to associate that instruction to the CFG representation of those lines of code.

Var.	Type	Storage
A	int8 A[N] (array of bytes)	Base array address stored in the stack at SP + 4
i	int i (32-bit scalar var.)	Variable stored in register r2
m	int m (32-bit scalar var.)	Variable stored in the stack at SP + 8
x	int x (32-bit scalar var.)	Variable stored in register r1

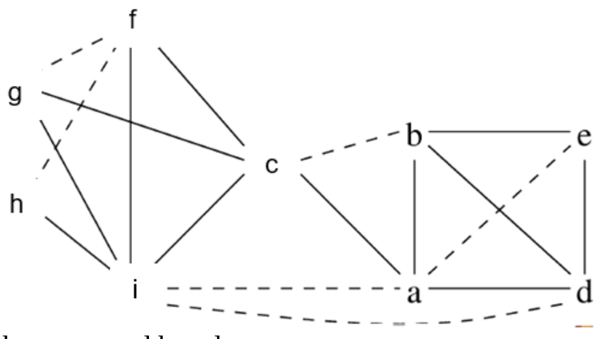
4.c) [2.5pt] In the context of liveness analysis, and based on the dataflow analysis equations that are responsible to propagate information between nodes, present modifications to the dataflow analysis iterative algorithm (maintaining the input CFG) that may make it faster.

4.d) [2.5pt] Consider the interference graph (IG) presented below. Indicate a possible allocation of registers using the graph coloring based register allocation algorithm presented in the lectures and considering a maximum of 3 registers (*R1*, *R2, and R3*). Show the steps performed and the content of the stack after the complete simplification of the IG. In the case of having to perform *spilling*, use the degree of the nodes to decide. In the case of having to perform *coalescing*, indicate the heuristic used and why you *coalesce* or *freeze*. Show the result of the first iteration of the register allocation algorithm (i.e., without repeating the process).

Instruction	Operation
load, Ri, Rj, C	Ri ← Mem[Rj+C]
store Rj, C, Ri	Mem[Rj+C] ← Ri
add Ri, Rj, Rk	Ri ← Rj + Rk
addi Ri, Rj, C	Ri ← Rj + C
mul Ri, Rj, Rk	Ri ← Rj * Rk
lth Ri, Rj, label1	If(Ri < Rj) goto label1
lte Ri, Rj, label1	If(Ri <= Rj) goto label1
gth Ri, Rj, label1	If(Ri > Rj) goto label1
gte Ri, Rj, label1	If(Ri >= Rj) goto label1
goto label1	jump to label1

Group 5. (2 pts)

5.a) [2 pt] Considering the liveness analysis using dataflow analysis, members of a compiler team suggested the following two options: (a) merging CFG nodes that have a single predecessor and a single successor into a new node; (b) Instead of computing dataflow information for all variables at once using sets, compute the analysis for each variable separately. Comment on these two options and explain why you think each of the options might make sense and be advantageous or not.



(End.)