The University of Melbourne

Assessment for Semester 1, 2012

Department: Computer Science
Subject number: COMP90045
Subject name: Programming Language Implementation
Exam date: June 2012
Exam duration: 3 hours
Reading time: 15 minutes
This paper has four pages.

Authorized materials

No materials are authorized.

Instructions to invigilators

Supply students with standard script books.

Exam papers may be taken from the exam room.

This paper should be lodged with the Baillieu Library.

Instructions to students

You should attempt all questions. You should use the number of marks allocated to a question to judge the level of detail required in your answer.

The maximum number of marks for this exam is 70.

(1) (a) In C++, two consecutive forward slash characters signal the start of a comment, which continues to the end of the line, as in this code fragment:

```
p = malloc(size);
if (p == 0) {      // check if malloc failed
        // we should print an error message here
        exit(1);
}
```

Write a lex pattern/action pair for recognizing such comments.

(b) Consider the lex specification consisting of these rules:

```
[a-z][a-z]*          { return ID; }
"xxx"                { return XXX; }
.                    { return OTHER; }
" "                  { return SPACE; }
```

Give the token sequences returned by yylex for the following two inputs.

(i) "xxxxx abc89"
(ii) "xxx xyz xxx"

Consider the input to consist of only the characters between the quotation marks.
(6 marks)

(2) (a) Convert the regular expression (a*b)|(ba?b) into an NFA using the algorithm taught in lectures, and draw that NFA.

(b) Give the algorithm for converting an NFA to a DFA. Describe the meaning of any primitive operations the algorithm relies on.
(9 marks)

(3) Consider the following grammar:

```
item -> BOX                 /* production 1 */
item -> CIRCLE attrs        /* production 2 */

attr -> TEXT                /* production 3 */
attr -> RADIUS NUM          /* production 4 */

attrs ->                    /* production 5 */
attrs -> attrs attr         /* production 6 */
```

Compute the FIRST and FOLLOW sets of each nonterminal. Is the grammar LL(1)? Why or why not?
(5 marks)

(4) Consider the following grammar:

```
item -> BOX                    /* production 1 */
item -> CIRCLE attrs           /* production 2 */

attr -> TEXT                   /* production 3 */
attr -> RADIUS NUM             /* production 4 */

attrs ->                       /* production 5 */
attrs -> attrs attr            /* production 6 */
```

The SLR parsing table for this grammar is the following.

state number	Action table						Goto table		
	BOX	CIRCLE	NUM	RADIUS	TEXT	EOF	item	attr	attrs
0	s2	s3					1		
1						accept			
2						r1			
3				r5	r5	r5			4
4				s6	s7	r2		5	
5				r6	r6	r6			
6			s8						
7				r3	r3	r3			
8				r4	r4	r4			

Show how an SLR parser works by showing the contents of the parser's stack after each action as it parses the following token sequence, and draw the resulting parse tree.

CIRCLE TEXT RADIUS NUM EOF

(5 marks) 

(5) Describe briefly the main differences between LL(1) parsers and LR(1) parsers in terms of what kinds of derivations they build and how they construct the parse tree. Give one reason why a compiler writer may prefer to use an LL(1) parser, and one reason why a compiler writer may prefer to use an LR(1) parser.
(4 marks)

(6) Write down yacc/bison token declarations and productions for recognizing expressions using the four usual arithmetic operations (addition, subtraction, multiplication and division) on integer constants. (You don't have to write lex rules to recognize the tokens themselves.)
(8 marks)

(7) (a) What is the distinction between S-attributed definitions and L-attributed definitions? Which one is more expressive? Is there any significant difference in how easily they can be implemented with a recursive descent parser? Why or why not?

(b) Is there a natural mechanism to implement synthesized attributes in an LR(1) parser? If yes, what is it? Is there a natural mechanism to implement inherited attributes in an LR(1) parser? If yes, what is it?
(7 marks)

(8) (a) What is *short circuit evaluation*?

(b) Give a short piece of code that works in the presence of short circuit evaluation and does not work in its absence. Say *why* it does not work without short circuit evaluation.

(c) Is it natural to use short circuit evaluation in a code generator targeting an instruction set which puts the result of all comparisons into the condition code register? Justify your answer.

(d) Is it natural to use short circuit evaluation in a code generator targeting an instruction set which puts the result of comparisons into general purpose registers as booleans? Justify your answer.
(8 marks)

(9) Write down semantic rules for generating code for if-then-else statements, i.e. for the production

stmt -> IF expr THEN stmt ELSE stmt

Assume that the target machine uses a condition code register.
(5 marks)

(10) System designers designate some registers to be caller save and other registers to be callee save.
(a) Why don't they make all registers caller save?
(b) Why don't they make all registers callee save?
(4 marks)

(11) Dynamic programming approaches to instruction selection work on instruction sequences represented by a directed acyclic graph or DAG. What are the nodes and the edges of this DAG?
(4 marks)

directed acyclic graph

(12) (a) What makes an optimization a peephole optimization?
(b) Give an example of a peephole optimization.
(5 marks)

A cheap way to optimize code sequences is to peep at them through a small sliding window, and try to fit the instructions in that window into known patterns. Code that fits a source pattern is then replaced with equivalent code that is shorter and/or faster, created from the target pattern associated with the source pattern.

END OF EXAM PAPER