The University of Melbourne

Semester 2 Assessment, 2006

Department: Computer Science

Subject number: 433-361

Subject name: Programming Language Implementation

Exam date: 16 November 2006

Exam duration: 3 hours

Reading time: 15 minutes

This paper has 5 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 75.

(1) (a) In many programming languages, a floating point constant is represented lexically as a string of digits containing a decimal point in which there must be at least one digit both before and after the decimal point. Write a lex pattern/action pair for recognizing such constants.

(b) If an action in a lex specification uses a C function *f* that is supposed to be private to the lex specification, where in the specification should you declare *f*, and where should you define *f*?
(4 marks)

(2) (a) Some scanners store in a string table all the strings they return to the parser in a token. Describe briefly two advantages of this approach compared to simply making a copy of each string as it is being returned.

(b) Give brief descriptions of two distinct approaches that a lexical analyzer can use to recognize the keywords of a programming language. For each approach, say why one may want to choose that approach.
(6 marks)

(3) Give an algorithm for minimizing DFAs. Given an input DFA, the algorithm should generate an equivalent DFA that has as few states as possible.
(6 marks)

(4) Consider the following grammar:

```
makefile ->
makefile ->       entry makefile

entry ->          definition
entry ->          rule

definition ->     ID EQUAL ids NEWLINE

ids ->
ids ->            ID ids

rule ->           targets COLON sources NEWLINE actions

targets ->        ID
targets ->        ID targets

sources ->
sources ->        ID sources

actions ->
actions ->        ACTION NEWLINE actions
```

Consider the following sequence of tokens, which is a sentence in the above grammar:

ID EQUAL ID NEWLINE ID COLON ID ID NEWLINE ACTION NEWLINE

Give the leftmost derivation for this token sequence, and draw the corresponding parse tree.
(5 marks)

(5) Consider the following left-recursive grammar fragment:

item -> item enda
item -> item endb endc
item -> basic1
item -> basic2

where *enda*, *endb*, *endc*, *basic1* and *basic2* are all terminals. Transform this grammar fragment to an equivalent grammar fragment that is not left recursive.
(4 marks)

(6) 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.

CIRCLE TEXT RADIUS NUM EOF

(4 marks)

(7) (a) What is the distinction between parse trees and abstract syntax trees?

(b) Give an example of an attribute included in abstract syntax trees in typical compilers purely for use in error messages.
(4 marks)

(8) (a) LR(1) parsers are generated by the sets-of-items construction algorithm. That algorithm generates states that each have one or more items. Each item suggests an action that the LR(1) parser should take when in that state and looking at a certain lookahead token.

Give the rules that LR(1) parsers use to determine what action table entry (if any) and what goto table entry (if any) is suggested by each item.

(b) When bison reports a shift/reduce conflict, what can the programmer do to find out the cause of that conflict?
(10 marks)

(9) Bottom-up parsers execute actions only when they perform a reduction, yet yacc allows specifications to include actions in the middle of productions, like this:

```
rest :    ADD_OP term { printf("+"); } rest
     |    SUB_OP term { printf("-"); } rest
     |    /* empty */
     ;
```

How does yacc implement such actions?
(3 marks)

(10) Describe one advantage and one disadvantage of a Java compiler generating code for the JVM versus generating code for the x86.
(4 marks)

(11) Give all the semantic rules required for code generation for the following three productions in PLI06, the source language used in the project. Make them generate code for T06, the target language used in the project. Do not assume anything about the types of variables and expressions beyond what is required by the rules of PLI06. Do assume that semantic checking has already been done, and that the values of any attributes that would be computed during semantic checking are in fact available.

```
stmt ->  ID ASSIGN_OP expr SEMICOLON
expr ->  ID
expr ->  expr ADD expr
```

(8 marks)

(12) When generating T06 code for performing a function call such as sqrt(2.0), what tasks does the generated code have to perform before the transfer of control to the called function, and what tasks does it have to perform after the transfer of control back from the called function?
(6 marks)

(13) Give an example of a C structure whose size depends on the order of its fields. Give the reason *why* the size depends on the order.
(4 marks)

(14) (a) Give an example of a peephole optimization.

(b) Give an algorithm for computing the set of basic blocks in the intermediate representation (IR) code of a function.
(7 marks)

END OF EXAM PAPER