T-106.4200 Introduction to Compiling
Exam Oct. 24, 2011

Examiner: Jorma Tarhio

No written material is allowed in this exam. Submit at least one answer sheet, even if an empty one! Write on *each* answer sheet you submit the code of the course, the date, your name, and your student ID number.

---

1. Answer shortly to the following questions:

(a) What is a display?

(b) What is a lookahead symbol?

(c) What is an LR(0) item?

(d) To which direction do semantic values go in an S-attributed grammar?

(e) What is bottom-up parsing?

(f) Which of the following methods handles the largest set of grammars and which one the smallest set: LR(1), SLR, LR(2)?

(6 p)

2. Consider the following ambiguous grammar:

$$S \rightarrow \mathtt{a}\ S\ \mathtt{b} \mid S\ \mathtt{x}\ S \mid S\ \mathtt{y} \mid \mathtt{z}$$

For each of the three strings below, Indicate whether the rightmost parse of the string is ambiguous or unambiguous according to this grammar. Show all of the possible parse trees for that string.

(a) `z x z y`

(b) `z y y`

(c) `a z x z b y x z`

(3+3+3 p)

3. (a) Give a description as a regular expression for identifier, which consists of letters, digits, and underscores (_). The last character must be a letter.

(b) Give a description as a regular expression for number, which has allowed forms
`5 +7.8 -0.0821E-77 7E+4`
and which has disallowed forms
`34. +.23 -8.92E28 1E-491 1E11`

(5+5 p)

P.T.O.

4. Study the following grammar: $\{P \rightarrow P \texttt{ => } P \mid P \texttt{ and } P \mid \texttt{not } P \mid (P) \mid atom\}$. Its LR parsing table is given below. Remove the parse conflicts by assuming that no operator is associative, and the precedence of the operators (higher first) is `not`, `=>`, and `and`.

	=>	and	not	(	)	*atom*	$	P
0			s2	s3		s4		1
1	s5	s6					acc	
2			s2	s3		s4		7
3			s2	s3		s4		8
4	r5	r5			r5		r5	
5			s2	s3		s4		9
6			s2	s3		s4		10
7	r3/s5	r3/s6			r3		r3	
8	s5	s6			s11			
9	r1/s5	r1/s6			r1		r1	
10	r2/s5	r2/s6			r2		r2	
11	r4	r4			r4		r4	

(10 p)

5. Compute the domination relation of the following piece of intermediate code:

```
(S1)   L0: a <- M[100]
(S2)       b <- M[200]
(S3)       i <- M[300]
(S4)       if a > b goto L1
(S5)       c <- b * b
(S6)       goto L2
(S7)   L1: c <- M[i]
(S8)       i <- i + 1
(S9)   L2: d <- a + c
(S10)      if d < 1000 goto L0
```

(10 p)

6. An RR(1) parser is similar to an LL(1) parser except that it works from right-to-left, tracing out a top-down, rightmost derivation. The parser selects the next production based on nonterminal on the top of the parsing stack and on the look-ahead terminal. The construction of the parsing table is analogous to the LL(1) case, except that instead of having FIRST and FOLLOW sets we have LAST and PRECEDE sets. Similarly, instead of having an end-of-input marker, we have a beginning-of-input marker, which we denote by £.

Is there a grammar that is RR(1) but not LL(1)? If so, give an example and explain both why it is not LL(1) and why it is RR(1). If not, explain why not.

(10 p)