A06507 *No calculator permitted in this examination*

UNIVERSITY OF BIRMINGHAM

School of Computer Science

Third year – Degree of BSc with Honours
Artificial Intelligence and Computer Science
Computer Science

Third year – Degree of BEng with Honours
Computer Science/Software Engineering
Chemical Engineering with year in Computer Science

Fourth Year – Joint Degree of MEng with Honours
Electronic and Software Engineering

Undergraduate Occasional
Computer Science/Software Engineering
Electronic and Electrical Engineering

Degree of MSc
Computer Security

06 02578

Compilers and Languages

Summer Examinations 2011

Time allowed: 1 ½ hours

[Answer ALL Questions]

1. This question is about grammars and predictive parsing.

   (a) Consider the following grammar, where X is the start symbol and $\oplus, \ominus, 1, 2, 3, 4$ and 5 are terminal symbols:

$$\begin{array}{lcl} X & ::= & Y \oplus X \mid Y \\ Y & ::= & Y \ominus Z \mid Z \\ Z & ::= & 1 \mid 2 \mid 3 \mid 4 \mid 5 \end{array}$$

   Draw the concrete parse tree corresponding to this grammar that would be constructed for the string:

$$1 \ominus 2 \ominus 3 \oplus 4 \oplus 5$$

   [5%]

   (b) What associativity does your answer to part (a) above imply for $\ominus$ and $\oplus$ and what is the relative precedence of $\oplus$ and $\ominus$?

   [5%]

   (c) Write out the set of terminal symbols in $\mathbf{First}(Ba)$ for the following grammar, where A is the start symbol, lower case characters are terminal symbols, upper case characters are non-terminal symbols and ϵ is the empty string of symbols.

$$\begin{array}{lcl} A & ::= & C \mid Bd \mid fD \mid \epsilon \\ B & ::= & A \mid Cb \\ C & ::= & eCa \mid \epsilon \\ D & ::= & cA \mid A \end{array}$$

   [10%]

   (d) Assume a grammar for a programming language includes the following productions:

$$\begin{array}{lcl} \mathit{stmt} & ::= & \texttt{IF}\ \mathit{condition}\ \texttt{THEN}\ \mathit{stmt}\ \texttt{ELSE}\ \mathit{stmt} \\ \mathit{stmt} & ::= & \texttt{IF}\ \mathit{condition}\ \texttt{THEN}\ \mathit{stmt} \end{array}$$

   The resulting problem, called the *dangling else* problem, cannot be handled by a predictive parser. The standard solution is to left-factor this fragment of the grammar. However, left-factoring does not remove the underlying ambiguity in the grammar.

   Write out the left-factored version of the above grammar fragment and explain how a hand-written predictive parser can easily deal with the remaining ambiguity.

   [5%]

2. This question is about shift-reduce parsing.

 (a) Consider the following augmented grammar, where S is the start symbol, \$ is the end of file pseudo-token, and $*$, (,) and ID are terminal symbols:

$$
\begin{array}{lcl}
S & ::= & T\ \$ \\
T & ::= & T * F \mid F \\
F & ::= & (T) \mid ID
\end{array}
$$

 The start state, I_0, of the LR(0) automaton for this grammar consists of the following set of LR(0) items:

$$
\begin{array}{l}
S ::= \cdot T\ \$ \\
T ::= \cdot T * F \\
T ::= \cdot F \\
F ::= \cdot(T) \\
F ::= \cdot ID
\end{array}
$$

 Write out the set of items in **Goto**(I_0, "("), i.e. the set of items in the state that the LR(0) automaton transitions to from the start state under the input ")".

[10%]

 (b) Describe the difference between the conditions under which a reduction is triggered in an SLR(1) parser and in an LR(1) parser and explain why that leads to the LR(1) parser being strictly more powerful than an SLR(1) parser.

[10%]

 (c) Prove that the number of states in an LALR(1) parser is exactly the same as in the SLR(1) parser for the same grammar.

[5%]

3. This question is about the call stack. In all cases, assume that no arguments are passed in registers.

   (a) Draw the contents of the call stack when it is at its deepest point during the execution of `f(2)` in the following C program, identifying the activation frames and showing the positions and values of all parameters and local variables.

```
1.   int f(int n)
2.   {
3.         int v = 0;
4.
5.         if (n < 2)
6.               v = 1 ;
7.         else
8.               v = f(n-1) + f(n-2) ;
9.         return v;
10.  }
```

[10%]

   (b) Consider the following Gnu C code. Draw the contents of the call stack immediately after line 6 is executed but before the function `q()` returns. Then explain the mechanism for accessing the variable "a" during the execution.

```
1.   int p()
2.   {
3.         int a = 0;
4.         void q()
5.         {
6.               a = 1;
7.         }
8.         q() ;
9.         return a;
10.  }
```

[10%]

   (c) Explain why line 5 in the following C code does not, in fact, change the value of y.

```
1.   void f (int x) {  x = x + 2;  }
2.   void g()
3.   {
4.         int y = 0;
5.         f(y);
6.   }
```

[5%]

4. This question is about data flow analysis.

(a) Consider the following Java code excerpt:

```
a = b + a;
if (a > 0)
{
    c = a + d;
}
else
{
    e = a;
    return e;
}
return c + e;
```

Draw the control flow graph for this code and annotate it with the sets of live variables.

[10%]

(b) The data flow equations for live variable analysis are as follows:

$$in[n] = gen[n] \cup (out[n] - kill[n])$$
$$out[n] = \bigcup_{s \in succ[n]} in[s]$$

Explain why those data flow equations are correct and write down the values of $gen[n]$ and $kill[n]$ when n is the Java statement:

```
x = x + y
```

[10%]

(c) List three different optimisations or problem identifications that data flow analysis can be used for in a compiler.

[5%]