Coláiste na Tríonóide, Baile Átha Cliath
Trinity College Dublin
Ollscoil Átha Cliath | The University of Dublin

TRINITY COLLEGE DUBLIN
THE UNIVERSITY OF DUBLIN

Faculty of Engineering, Mathematics and Science
School of Computer Science and Statistics

Integrated Computer Science
BA (MOD), CSLL
Year Three Annual Examination

Trinity Term 2016

Compiler Design (CS3071)

11 May, 2016 Examination Hall 14:00 – 16:00

Dr DM Abrahamson

Instructions to Candidates

Attempt question number 1 and one other question

Materials permitted for this examination

None

1. i. By computing selection sets for the productions from the following context free grammars, determine which (if any) of them are LL(1):

a.
```
<S> → <X> Y
<X> → <Y>
<X> → X
<X> → ε
<Y> → y
<Y> → ε
```

b.
```
<S> → <X> <Y> z
<X> → x
<X> → ε
<Y> → y
<Y> → ε
```

c.
```
<S> → <X> <Y> <Y> <X>
<X> → x
<X> → ε
<Y> → y
<Y> → ε
```

[15 Marks]

ii. In relation to error processing, describe the prefix property and outline the differences between local and global error recovery, and explain in detail the function of the recursive descent procedure `SKIP_TO` and its use in global error processing.

[15 Marks: prefix property–2; differences–4; and `SKIP_TO`–9]

iii. Given the l-attributed translation grammar

$$\begin{aligned}
&\texttt{<statement>} && \rightarrow && \texttt{<expression>}_p\ \{\texttt{print}\}_p \\
&\texttt{<expression>}_p && \rightarrow && \texttt{+ <expression>}_q\ \texttt{<expression>}_r\ \{\texttt{add}\}_{q,r,p} \\
& && && \{\texttt{add}\}_{p,q,r}\ \ r \leftarrow p+q \\
&\texttt{<expression>}_p && \rightarrow && \texttt{* <expression>}_q\ \texttt{<expression>}_r\ \{\texttt{mult}\}_{q,r,p} \\
& && && \{\texttt{mult}\}_{p,q,r}\ \ r \leftarrow p*q \\
&\texttt{<expression>}_p && \rightarrow && \texttt{const}_q
\end{aligned}$$

where $\texttt{<expression>}_p$ – synthesized $_p$, all action symbol attributes are inherited and `const` is a lexical token for an integer constant, design an augmented pushdown machine (including all stack replacements) for an interpreter to compute the value of an arithmetic expression and show its operation using a stack movie of it parsing the simple prefix expression

```
* 2 + 16 32.
```

[30 Marks: pushdown machine–12; stack replacements—10; and stack movie–8]

[Total 60 Marks]

2. Design some form of switch (or case) statement and explain the underlying reason(s) behind your choice of syntax. Describe the flow of control and issues that might arise in the generation of object code for such statements. Finally, design l-attributed production(s) for your switch statement.

[40 Marks: language design (including basis for syntax)–16; flow of control–8; attributed translation grammar–8; description of attributes–2; and description of action symbols–6]

3. Discuss the relationship between the output action symbols {label$_p$} and {jump$_p$} [or {jumpt$_{p,q}$}/{jumpf$_{p,q}$}], and demonstrate their use by converting the context free productions

```
<statement> → DO <statements> WHILE <condition>
<statement> → WHILE <condition> DO <statements>
```

into equivalent l-attributed translation productions. For each production sketch the flow of control through the generated object code as well as the sequence of actions performed during the parse. Describe the purpose of the function "`newl`" and show, by example, how the processing of the address field in a generated branch instruction is dependent on the relative position of a {label} and its corresponding {jump}, {jumpt} or {jumpf} action.

[40 Marks: attributed translation grammar–12; relationship between action symbols–2; sequence and flow of control–8; purpose of function `newl`–2; and discussion about relative position of action symbols and address field–16]