# General feedback

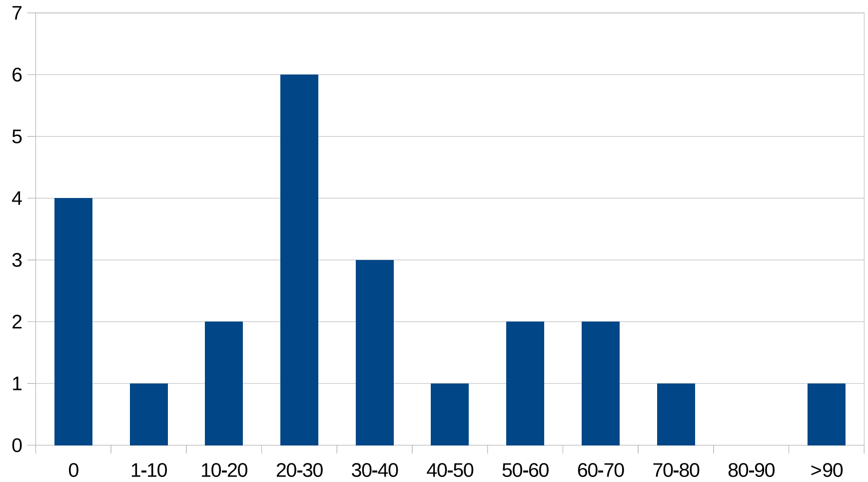
**CS3470 Assessed Coursework 3**

## The marking process

At the lecture of the 7th of December the students have received a copy of their scripts with handwritten annotations and corrections. The lecturer identified recurring patterns. These patterns are discussed in this document. The most problematic exercises are discussed — and solutions given — in the lecture on the 7th of December.

## General feedback

The assignment consisted of two parts: (S)LR parsing and GLL parsing. The first part was done better than the second part, mostly because not many students attempted to execute the GLL parse algorithms. The average mark was not higher than 40%, because the majority of points was to be obtained in part 2 and executing the GLL algorithms in particular. The histogram below shows how the marks were divided among students.



**Question 1** Almost all students constructed the NFA nearly correctly, but only some perfectly. Common mistakes include forgetting an $\epsilon$-edge towards a square node and to include two nodes for $\epsilon$, one labelled $\cdot\epsilon$ and the other $\epsilon\cdot$.

A large number of students constructed the DFA with no or but a few errors. However, a large number of students did not apply the subset construction technique.

To place the reductions in the SLR parse table it is necessary to compute the FOLLOW-sets of the nonterminals of the grammar. Many students made small errors in doing so, resulting in incorrect parse tables.

**Question 2** The DFA to be constructed for this exercise was significantly easier (or at least smaller). Many students got the DFA correct although some did not compute closures correctly.

A large number of students did not construct the SLR(1) table correctly. For example, some tables had rows without entries. This indicates a mistake, as a DFA state cannot simultaneously have no outgoing edges and no reductions. Many students forgot to place reductions, or placed them incorrectly.

It would have been easy for students to check their own answer by determining whether the input string is in the language defined by the grammar (which it is) and then working out how the parser is supposed to discover this fact.

To show that a grammar is ambiguous we need to give two leftmost (or rightmost) derivations of a particular string. Non-ambiguous grammars exist for which the SLR(1) parse table contains conflicts, for example:

```
S ::= aab | Aa
A ::= a
```

**Question 3** Many students applied the GLL templates correctly or only with few errors. Applying the GLL templates correctly should have been straightforward and would have given you 12 marks relatively easily. Very few students attempted to execute the parser.

**Question 4** More students attempted the execute the parser of question 4. Most made clerical errors resulting in missing descriptors, GSS-nodes, or pop-set elements. Simply attempting this exercise and getting reasonably far would have earned a lot of marks. Even students that did not really attempt question 3/4 recognised part (ii) of question 4 as a relatively easy way to score some additional points, and did this without error.

## Strategic Tips

Especially when under time pressure, make sure that you familiarise yourself with the entire assignment before you commence. There is no reason to do the exercise in the order specified. There are several ways in which you could have checked your answers.

Most importantly, the assignment asked you to execute three parse algorithms. Beforehand, ask yourself the question whether the parse is supposed to succeed by trying to find a derivation yourself (without the help of an algorithm — you are still more creative than a computer in finding it). If you find no derivations, you will probably know why the parser should fail. If you do find derivations, you can use them to check the parser's steps. Any mismatch means that the generated parser / parse table is incorrect, or that you are applying the algorithm incorrectly.

Note that a DFA is a *deterministic* automaton. It can therefore not have two or more outgoing edges when one of these is labelled $\epsilon$ or when two of these are labelled with the same symbol (otherwise there is a choice to make and the need for backtracking).

In an LR(0) NFA there is a "branch" of nodes for each production $X ::= \alpha$ of the grammar, ending in a reduction state with the $\alpha\cdot$. There is thus exactly one reduction state in the NFA for each production. An LR(0) DFA — constructed from an LR(0) NFA via the subset construction or constructed directly — will thus have the LR-item $X ::= \alpha\cdot$ appearing at least once. All the states in which it occurs will be reduction states (even if $\alpha = \epsilon$).

Question 3 asked you to write down a GLL parser, whilst question 4 gave you a GLL parser to execute. You could have compared the parser you wrote down for question 3 with the one given as part of question 4. Any difference (apart from the underlying grammar) indicates a mistake on your part in question 3 (assuming question 4 is correct).