apl. Prof. Dr. Thomas Noll

Friedrich Gretz, Souymodip Chakraborty

## Exercise 1 (Evaluation of Arithmetic Expression): (3 Points)

Write an unambiguous grammar for arithmetic expressions (containing addition, multiplication and parenthesis). Define an attribute *Val* to find the value of an expression. Evaluate $2 * 5 + 3$. For this, give the parse tree of this expression, set up the corresponding equation system and solve it.

## Exercise 2 (Attributed Grammars): (4 Points)

Give a context-free grammar for the language $\{a\}^+$. Extend that grammar with attributes so that the language of $G$ is the following set:

1. $L = \{a^{2^n} | n \in \mathbb{N}\}$.
2. $L = \{a^{n^2} | n \in \mathbb{N}\}$.

You may use any number of attributes, conditional updates, simple arithmetic and comparison between numbers. However, you may not use a predicate that directly checks whether a given number (in decimal or binary encoding) is a power of two or not.

## Exercise 3 (Circularity Test): (3 Points)

In this task we implement a semantic check. In our language *WHILE* we require that every variable identifier is *declared* before the variable is used (read or set). Additionally, a variable defined inside a scope like an if statement or a while loop is not visible outside this scope. We do not care whether a variable has been *initialised* before it is read. Examples:

This is valid:

```
1    int x; int y;
2    if (x <= y) {
3        write("...");
4    }
5    write(x);
6    $
```

This is not valid (`y` is undefined and `z` is undefined outside the if-statement):

```
1    int x;
2    if (x <= y) {
3        int z;
4        // ...
5    }
6    write(z);
7    $
```

Implement `Checker.checkDeclaredBeforeUsed()`. *Hint: for this you do not need to implement any attributed grammars and their evaluation. Instead simply walking the abstract syntax tree once and checking the required property suffices.*