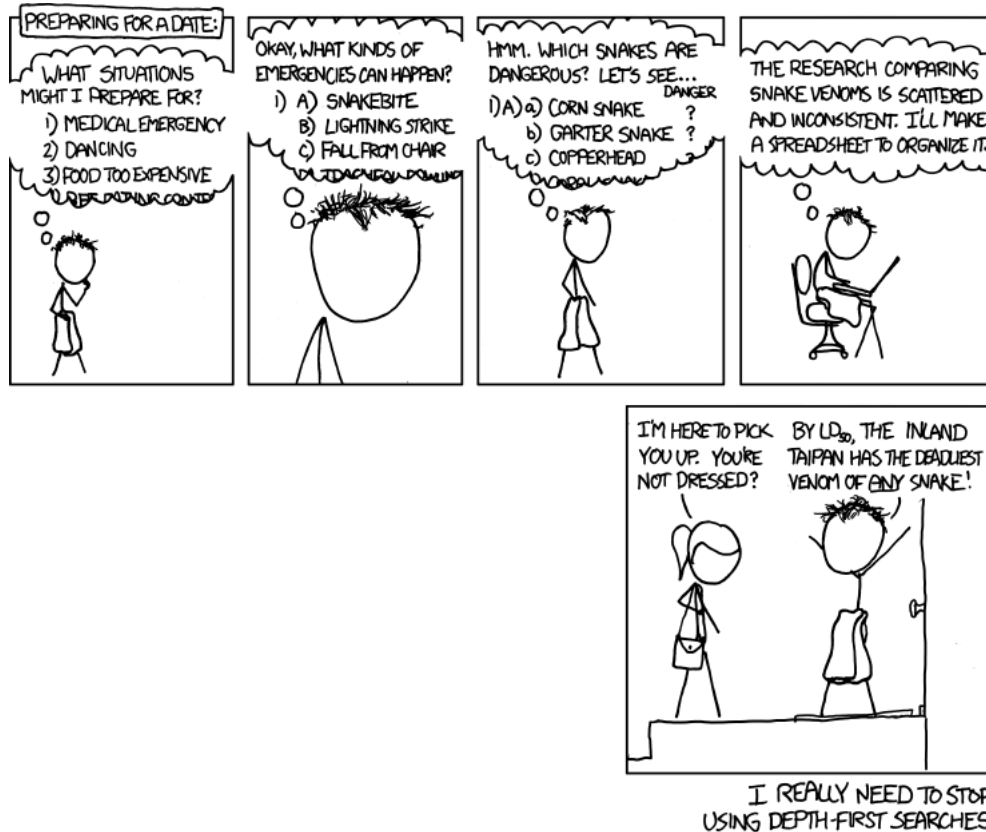


Compiler Construction 2016

— Series 7 —

Hand in until July 5th before the exercise class.



Credit: <https://xkcd.com/761/>

General Remarks

- It is allowed to hand in your solutions for the theoretical part via email **as a separately attached PDF file**. If you hand in solutions via email please make sure that the size is reasonable (< 1 MB). Solutions that our printer rejects to print within 30min due to large scanned images will not be corrected.
- Please hand in your solutions in groups of 3 or 4.

Exercise 1

(3 Points)

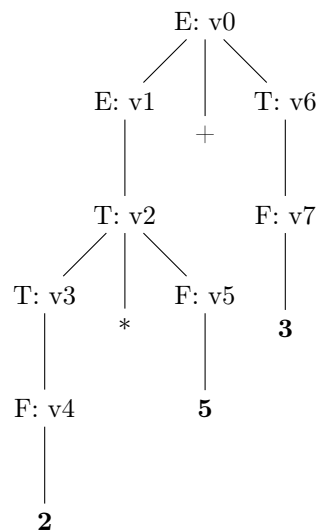
Write an unambiguous grammar for arithmetic expressions (containing addition, multiplication and parenthesis). Define an attribute *Val* to find the value of an expression. Evaluate $2 * 5 + 3$. For this, give the parse tree of this expression, set up the corresponding equation system and solve it.

Solution: _____

The attribute grammar for the arithmetic expression is as follows:

$$\begin{aligned}
 E &\rightarrow E + T & Val.0 &= Val.1 + Val.3 \\
 E &\rightarrow T & Val.0 &= Val.1 \\
 T &\rightarrow T * F & Val.0 &= Val.1 * Val.3 \\
 T &\rightarrow F & Val.0 &= Val.1 \\
 F &\rightarrow (E) & Val.0 &= Val.2 \\
 F &\rightarrow \mathbf{num} & Val.0 &= \mathbf{num}
 \end{aligned}$$

The annotated parse tree for $2 * 5 + 3$ as follows:



The set of variables is $\{v0, \dots, v7\}$. The equation system is defined as follows, where bold expressions on the right-hand side result from solving the system.

$$\begin{aligned}
 v0 &= v1 + v6 = \mathbf{13} \\
 v1 &= v2 = \mathbf{10} \\
 v2 &= v3 * v5 = \mathbf{10} \\
 v3 &= v4 = \mathbf{2} \\
 v4 &= 2 = \mathbf{2} \\
 v5 &= 5 = \mathbf{5} \\
 v6 &= v7 = \mathbf{3} \\
 v7 &= 3 = \mathbf{3}
 \end{aligned}$$

Exercise 2

(4 Points)

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

1. $L = \{a^{2^n} \mid n \in \mathbb{N}\}$.
2. $L = \{a^{n^2} \mid n > 0\}$.

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.

Solution: _____

Definition A word w is in the language of an attributed grammar G with Boolean attribute b iff $w \in L(G)$ and, at the root of a derivation tree, $b.0 = \text{true}$.

Consider the grammar G :

$$\begin{array}{lcl} S & \rightarrow & a S \\ S & \rightarrow & a \end{array}$$

The language of G is $\{a\}^+$. We have the following synthesized attributes $\{c, f, b\}$. The semantic rules for each production is give an as follows:

1. Intuitively, c stores the current power of 2 we have encountered so far. Then we store the additional number of a 's we have read after the last occurrence of a power of 2. Thus, we encounter a power of 2 iff $f \geq c - 1$.

$$S \rightarrow a S \quad \text{if } f.2 < c.2 - 1 \text{ then } f.0 = f.2 + 1; c.0 = c.2; b.0 = \text{false}; \\ \text{else } f.0 = 0; c.0 = c.2 + f.2 + 1; b.0 = \text{true}$$

$$S \rightarrow a \quad f.0 = 0; c.0 = 1; b.0 = \text{true};$$

2. Intuitively, c corresponds to the number of a read so far and f is the current value of n .

$$S \rightarrow a S \quad \text{if } c.2 = f.2 * f.2 - 1 \text{ then } f.0 = f.2 + 1; c.0 = c.2 + 1; b.0 = \text{true}; \\ \text{else } c.0 = c.2 + 1; f.0 = f.2; b.0 = \text{false}$$

$$S \rightarrow a \quad f.0 = 2; c.0 = 1; b.0 = \text{true};$$

Exercise 3

(4 Points)

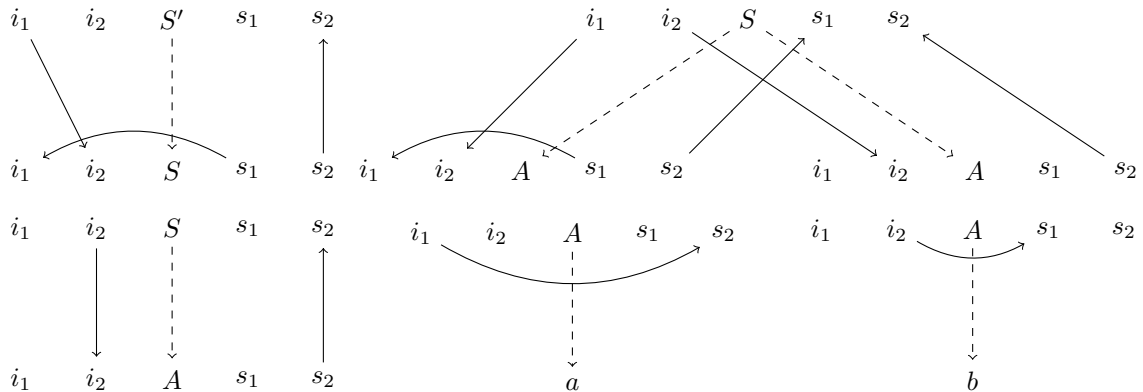
Consider the following grammar $G = (N, \Sigma, P, S)$ with inherited attributes $i1, i2$ and synthesised attributes $s1, s2$.

$$\begin{array}{ll}
 S' \rightarrow S & \begin{array}{l} i1.0 = 1 \\ i2.0 = 2 \\ i1.1 = s1.1 \\ i2.1 = i1.0 \\ s2.0 = s2.1 \end{array} \\
 S \rightarrow AA & \begin{array}{l} i1.1 = s1.1 \\ i2.1 = i1.0 \\ i1.2 = 0 \\ i2.2 = i2.0 \\ s1.0 = s2.1 \\ s2.0 = s2.2 \end{array} \\
 S \rightarrow A & \begin{array}{l} i1.1 = 0 \\ i2.1 = i2.0 \\ s2.0 = s2.1 \end{array} \\
 A \rightarrow a & \begin{array}{l} s1.0 = 0 \\ s2.0 = i1.0 \end{array} \\
 A \rightarrow b & \begin{array}{l} s2.0 = 0 \\ s1.0 = i2.0 \end{array}
 \end{array}$$

- Provide the dependency graph for each production in G .
- Apply the circularity test from the lecture to G .
 - Calculate the set $IS(A)$ for all $A \in N$.
 - Is G circular? Justify your answer.

Solution: _____

- For every production π in the attributed grammar, we draw the dependency graph $\rightarrow_\pi$.



- $IS(A) = \{is_1, is_2\}$ where

- $is_1 = is[A \rightarrow a] = \{(i1, s2)\}$
- $is_2 = is[A \rightarrow b] = \{(i2, s1)\}$

$IS(S) = \{is_3, \dots, is_8\}$ where

- $is_3 = is[S \rightarrow A, is_1] = \emptyset$

- $is_4 = is[S \rightarrow A, is_2] = \emptyset$
- $is_5 = is[S \rightarrow A A, is_1, is_1] = \emptyset$
- $is_6 = is[S \rightarrow A A, is_1, is_2] = \emptyset$
- $is_7 = is[S \rightarrow A A, is_2, is_1] = \emptyset$
- $is_8 = is[S \rightarrow A A, is_2, is_2] = \emptyset$

Now observe that

$$is[S' \rightarrow S, is_3] = \dots = is[S' \rightarrow S, is_8] = is[S' \rightarrow S, \emptyset] .$$

Hence, $IS(S') = \{is_9\}$ where

- $is_9 = is[S' \rightarrow S, \emptyset] = \emptyset$

G is not circular! There are two possible cover productions. In both productions $S' \rightarrow S$ and $S \rightarrow AA$, we have that i_1 depends on s_1 . However we have just shown that there is no subtree of S nor a subtree of A which has any dependencies. Therefore it is impossible to construct a dependency graph with a cycle.

Exercise 4

(* 3 Bonus Points)

In this task we implement a semantic check. In our language 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:

```
int x; int y;
if (x <= y) {
    write("Hello world.");
}
write(x);
```

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

```
int x;
if (x <= y) {
    int z;
}
write(z);
```

Implement `checker.DeclarationChecker.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.*

Solution: _____

For the solution see the code in `While2JasminCompilerSolution`.