# Assignment 5 by Lucas Karlsson

Consider the following language over the alphabet $\Sigma = \{a, b\}$ :
$M = \{a^i b^{i+j} | i, j \in \mathbf{N}\}$

**1.** Give an ambiguous context-free grammar $G = (\{S\}, \Sigma, P, S)$ for $M$. Note that the grammar should have exactly one nonterminal, $S$.

**Solution 1:**
We have our language $M = \{a^i b^{i+j} | i, j \in \mathbf{N}\}$ over the alphabet $\Sigma = \{a, b\}$, our language will then only contain strings that starts with zero or more a's followed by the same amount of b's or more. Now we only need to define the productions for the language:

P $\rightarrow \epsilon$

P $\rightarrow Pb$

P $\rightarrow aPb$

These are all the rules we need to define for our CFG and we have
$G = \{\{P\}, \{a, b\}\, A, P\}$ where $A$ is the set of rules for the language.

**2.** Prove that $G$ is ambiguous.

**Solution 2:**
To prove that $G$ is ambiguous we have to be able to derive the same string in two or more different ways doing only left-most derivation or right-most derivation. We are going to use left-most derivation for this and doing it on the string $abb$.

$$P \underset{lm}{\Rightarrow} Pb \underset{lm}{\Rightarrow} aPbb \underset{lm}{\Rightarrow} abb$$

$$P \underset{lm}{\Rightarrow} aPb \underset{lm}{\Rightarrow} aPbb \underset{lm}{\Rightarrow} abb$$

□

**3.** Define a recursive function (in haskell ofc) that takes two natural numbers i and j and returns a proof showing that the string $a^i b^{i+j}$ can be derived from $S$. The proof should be represented by a list of derivation steps:

- The derivation step $\alpha A \beta \Rightarrow \alpha\gamma\beta$ should be represented by the four tuple $(\alpha, A, \gamma, \beta)$
- If the list contains $n$ elements, then there should be a derivation $S \overset{*}{\Rightarrow} a^i b^{i+j}$ of the length n, where k-th step of the derivation matches the k-th element of the list.

**Solution 3:**

```
type Set = (String, String, String, String)

run :: Int -> Int -> IO ()
run i j = mapM_ printset \$ derive i j [("","P","P","")]

derive :: Int -> Int -> [Set] -> [Set]
derive 0 0 sets = tail $ sets ++ [(removeP $ last sets)]
  where removeP (x,y,z,v) = (x,x++z++v,[c | c <- z, c /= 'P'], v)

derive i j sets
  | i > 0 = derive (i-1) j $ sets ++ [deriveset "a" $ last sets]
  | j > 0 = derive i (j-1) $ sets ++ [deriveset "b" $ last sets]

deriveset :: String -> Set -> Set
deriveset "a" (x,y,z,v) = ("a", old, old, "b") where old = x++y++v
deriveset "b" (x,y,z,v) = ("", old, old, "b") where old = x++y++v

printset :: Set -> IO ()
printset (x,y,z,v) = putStrLn (y++" => "++x++z++v)
```

Figure 1: Haskell solution to question 3.

```
*Main> run 2 1
P => aPb
aPb => aaPbb
aaPbb => aaPbbb
aaPbbb => aabbb
```

```
*Main> run 1 2
P => aPb
aPb => aPbb
aPbb => aPbbb
aPbbb => abbb
```

```
*Main> run 3 0
P => aPb
aPb => aaPbb
aaPbb => aaaPbbb
aaaPbbb => aaabbb
```

Figure 2: Sample outputs from running the haskell function **run**.

**4.** Prove $\forall w \in L(G, S). w \in M$, where $L(\_, \_)$ is the inductive construction introduced in the eleventh lecture under the heading of **Recursive inference**.

**5.** Give an unambiguous context-free grammar $G'$ for $M$.

**6.** (**BONUS**) Is there an algorithm for checking whether an arbitrary context-free grammar implements the language $M$?