Aufgabe 10.1.  [Product of PDS and NFA] (10 points)
In order to define the language of a pushdown system, we add a set $F$ of final states. That is, a pushdown system with final states is a tuple $M = (P, \Gamma, \Sigma, p_0, \gamma_0, F, \Delta)$. The language of $M$ is defined as the label sequences of all runs to final states (and arbitrary stack):

$$L(M) := \{l \mid \exists p' \in F. \exists w. p_0 \gamma_0 \xrightarrow{ l^* } p' w\}$$

Now let $M = (P, \Gamma, \Sigma, p_0, \gamma_0, F_M, \Delta_M)$ be a pushdown system, and $A = (Q, \Sigma, I, F_A, \Delta_A)$ be a nondeterministic word automaton. Construct a pushdown system with final states $M'$, such that $L(M') = L(M) \cap L(A)$. Prove your construction correct.

Hint: Use a product construction. You do not need tree automata for this exercise.
Aufgabe 10.2.  [Program Analysis] (10 points)

1. Translate the following program into a PDS. The actions are \{\(x == 0, y = 1, x = y * x, x = x - 1, x = 5, \tau\)\} where \(\tau\) labels call and return transitions. When reaching the \textit{halt}-command, the PDS should get stuck. This ensures that \textit{main} cannot return.

   \begin{verbatim}
   int x, y;
   void factorial() {
     if (x == 0) {
       y = 1;
     } else {
       y = y * x;
       x = x - 1;
       factorial();
     }
   }
   void main() {
     x = 5;
     factorial();
     halt;
   }
   \end{verbatim}

2. What is wrong with the above program? Does it really compute the factorial of 5?

3. Let’s focus on variable \(y\). There are actions that initialize \(y\), actions that read \(y\) (and thus require it to be initialized), and actions that do not touch \(y\). Specify a regular tree language that characterizes all non-returning execution trees where \(y\) is read before it is initialized. Note: Your automaton shall specify all those execution trees from \(XN\) (cf. Slide 101), not only the ones of the PDS from (1). Use the following sets:

   - \(Base = Read \cup Init \cup Unrelated\) Set of base rules, composed of those actions that read the variable, initialize the variable, and do not touch the variable.
   - \(Call\) Set of call rules. They do not touch the variable.
   - \(Return\) Set of return rules. They do not touch the variable.
   - \(P, \Gamma\) States and stack alphabet of the PDS.

   Note: You are not required to prove your construction correct.

4. Assume that you have a function \textit{program-to-PDS(prog)}, which translates a program to a PDS, a function \textit{PDS-to-NFTA(pds)} which converts a PDS to a tree automaton describing its execution trees, and a function \textit{bad-execs-NFTA(pds)} that uses the construction from (3) to return a tree automaton that describes all execution trees over the rules from \(pds\) which read an uninitialized variable. Moreover, assume you have already implemented the standard algorithms on tree automata from this lecture.

   Specify, in pseudocode, and algorithm \textit{may-read-uninit(prog)}, which takes as input a program, and outputs \textit{true} if the program may access an uninitialized variable. Use the ideas of this exercise!