Exercise 1 (Procedures with Parameters): (2+2 Points)

Consider the following modification to our *WHILE* language, where procedures have (exactly) one parameter:

$$\begin{array}{rcl} p & ::= & \textbf{proc } P(x) \textbf{ is } c;\ p \mid \varepsilon \in \textbf{PDec} \\ c & ::= & \ldots \mid \textbf{call } p(a) \in \textbf{Cmd} \end{array}$$

Lift the operational semantics to meet the extended language, i.e. define new *call* and *block* rules

1. for a call by value parameter.
2. for a call by reference parameter. (Without restriction you can assume that a in **call** $p(a)$ is indeed a variable here.)

Exercise 2 (Denotational Semantics for Procedures): (3 Points)

Compute $\mathfrak{C}''[\![c]\!]\rho_1\pi_\varnothing\sigma$ of the following program

```
c  ≡  begin
        var y;
        proc P is
          y := 1;
          while ¬(x = 1)do
            y := y * x;
            x := x − 1;
        call P;
      end
```

where $\rho_1 := \rho_0[x \mapsto 0] \in$VEnv, $\pi_\varnothing \in$PEnv and $\sigma \in$Sto (with $\sigma(0) \neq \bot$).

Exercise 3 (Dynamic Scoping): (1+1 Points)

Considering dynamic scoping instead of static stoping leads to a simplification of operational semantics for blocks and procedures. What constitutes this simplification?
Does dynamic scoping lead to a simplification of denotational semantics, too? Why or why not?