For a change, this exercise should be solved on paper only, not using Isabelle.

**Exercise 13.1  Procedure Definedness Check**

We consider a language with statically scoped procedures but (for simplicity) without local variables. For this language we can define a small-step semantics that does not require a special procedure environment. Instead, the context of procedure declarations is managed by gradually transforming the program itself. The rules for the basic commands remain unchanged:

\[
(x ::= a, s) \rightarrow (\text{SKIP}, s(x := \text{aval} a s))
\]

\[
(\text{SKIP}; c_2, s) \rightarrow (c_2, s)
\]

\[
(c_1, s) \rightarrow (c_1', s') \Rightarrow (c_1; c_2, s) \rightarrow (c_1', c_2, s')
\]

\[
\text{bval} b s \Rightarrow (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, s) \rightarrow (c_1, s)
\]

\[
\neg \text{bval} b s \Rightarrow (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, s) \rightarrow (c_2, s)
\]

\[
(\text{WHILE } b \text{ DO } c, s) \rightarrow (\text{IF } b \text{ THEN } c; \text{WHILE } b \text{ DO } c \text{ ELSE } \text{SKIP}, s)
\]

Now, procedure declarations distribute over semicolons, and disappear when they surround a \text{SKIP}. Moreover, we may make an arbitrary step under a procedure declaration:

\[
(\{\text{PROC } p = cp; c_1, c_2\}, s) \rightarrow (\{\text{PROC } p = cp; c_1\}; \{\text{PROC } p = cp; c_2\}, s)
\]

\[
(\{\text{PROC } p = cp; \text{SKIP}\}, s) \rightarrow (\text{SKIP}, s)
\]

\[
(c, s) \rightarrow (c', t) \Rightarrow (\{\text{PROC } p = cp; c\}, s) \rightarrow (\{\text{PROC } p = cp; c'\}, t)
\]

(a) Complete the small-step semantics by formulating the missing rules for CALL.

(b) Define a recursive function that checks if a program is well-formed, that is, it contains no calls to procedures that were not defined.

(c) Prove that the evaluation of a well-formed program cannot get stuck: If \(c\) is well-formed and \((c, s) \rightarrow^* (c', t)\) and final \((c', t)\) then \(c' = \text{SKIP}\).

Recall that final \(cs\) is defined as \(\neg (\exists cs'. \, cs \rightarrow cs')\).