Part I: Interpreters and operational semantics

Exercise I.1 (**) Prove theorem 2 (the unique decomposition theorem). Hint: use the fact that a value cannot reduce.

Exercise I.2 (*) Check that the reduction rules for derived forms (let, if/then/else, fst, snd) are valid.

Exercise I.3 (*) Consider \( a = 1 \ 2 \). Does there exist a value \( v \) such that \( a \Rightarrow v \)? Same question for \( a' = (\lambda x. x \ x) \ (\lambda x. x \ x) \). Do you see a difference between these two examples?

Exercise I.4 (**) As claimed in the proof of theorem 4, show that if \( a \rightarrow b \) and \( b \Rightarrow \) then \( a \Rightarrow \). Hint: proceed by induction on the derivation of \( a \rightarrow b \) in SOS and by case analysis on the last rule used to derive \( b \Rightarrow \).

Programming exercise I.5 (**) Implement a naive interpreter that follows exactly the Wright-Felleisen presentation of CBV semantics. The core function to implement is \texttt{decomp}, taking a term \( a \) as argument and returning a pair of a reduction context \( E \) and a subterm \( a' \) such that \( a \ = \ E[a'] \) and \( a \) reduces iff \( a' \) reduces at head. How will you represent contexts? Compare the efficiency of this interpreter with that of the naive interpreter based on the SOS presentation shown in the course.

Exercise I.6 (*) We relax the (app-r) reduction rule as follows:

\[
\frac{b \rightarrow b'}{a \ b \rightarrow a \ b'} \quad \text{(app-r')}
\]

That is, we allow reductions to take place on the right of an application, even if the left part of the application is not yet reduced to a value.

1. Show by way of an example that the reduction relation is no longer deterministic: give three terms \( a, a_1, a_2 \) such that \( a \rightarrow a_1 \) and \( a \rightarrow a_2 \) and \( a_1 \neq a_2 \).
2. Check that theorems 3 and 4 ($a \Rightarrow v$ if and only if $a \xrightarrow{*} v$) still hold for this relaxed, non-deterministic reduction relation. Conclude that the left-to-right evaluation order makes no difference for terminating terms.

3. Give an example of a term that evaluates differently under the two reduction strategies (left-to-right and non-deterministic).
Part II: Abstract machines

**Exercise II.1 (**)** A Krivine machine is hidden in the ZAM. Can you find it? More precisely, define a compilation scheme $N$ from $\lambda$-terms to ZAM instructions that implements call-by-name evaluation of the source term.

**Exercise II.2 (*)** The $C$ compilation schema for the ZAM is not mathematically precise: in the application case, it is not 100% clear what the $k$ in $\text{PUSHRETPADDR}(k)$ is, exactly. Reformulate the compilation schema more precisely as a 2-argument function $C(a,k)$, which should prepend to the code $k$ the instructions that evaluate the expression $a$ and deposit its value on the top of the stack.

**Exercise II.3 (**/***)** How would you extend the ZAM and its compilation scheme to handle \texttt{if/then/else} conditional expressions?

**Exercise II.4 (**)** Prove lemma 8 (the Final states lemma for Krivine’s machine).

**Exercise II.5 (***)** State and prove the analogues of the Simulation, Initial state and Final state lemmas (lemmas 6, 7, 8 in the case of Krivine’s machine) for the HP calculator. Use the notion of decompilation defined in the lecture notes and the following reduction semantics for arithmetic expressions:

\[
\begin{align*}
N_1 + N_2 & \rightarrow N \quad \text{(if $N = N_1 + N_2$)} \\
N_1 - N_2 & \rightarrow N \quad \text{(if $N = N_1 - N_2$)} \\
\quad a & \rightarrow a' \\
\quad b & \rightarrow b' \\
\quad a \text{ op } b & \rightarrow a' \text{ op } b \\
\quad N \text{ op } b & \rightarrow N \text{ op } b'
\end{align*}
\]