Q1: Grammars, Inductive Definitions and Induction

Consider the following grammar:

Variables \( x, y, z \)

Terms \( M, N \) ::= \( x \) variable
\| \( (\lambda x. N) \) abstraction
\| \( M (M) \) application

Definition for Questions 1(c)–(d). The rules below define an unused variable judgment:

\( V \) unused-in \( M \) means that the variables listed in \( V \) are not used in \( M \)

For example:

\( z \) unused-in \( x(y) \)
\( z, y \) unused-in \( x \)
\( z, y \) unused-in \( \lambda z. z \)

Note that in \( \lambda z. z \), the use of \( z \) refers to the \( z \) bound by the \( \lambda \), so \( z \) is considered to be unused. The list of variables does not have to be complete, so \( \emptyset \) unused-in \( M \) is derivable for every \( M \).

However, if a variable is listed in \( V \) it must really be unused, so \( x \) unused-in \( x \) is not derivable.

Variable lists \( V ::= \emptyset \mid V, x \)

\( V \) unused-in \( M \) Variables listed in \( V \) are not used in \( M \)

\[ \frac{x \notin V}{V \text{ unused-in } x} \quad \frac{V \setminus \{x\} \text{ unused-in } N}{V \text{ unused-in } (\lambda x. N)} \quad \frac{V \text{ unused-in } M}{V \text{ unused-in } M(N)} \]

In rule ui-abs, \( V \setminus \{x\} \) removes \( x \) from \( V \). For example, if \( V = \emptyset, y, x, z \) then \( V \setminus \{x\} = \emptyset, y, z \).
§1 Q1: Grammars, Inductive Definitions and Induction

2 Q2: Type Preservation

The language of terms from Q1 is essentially the original \(\lambda\)-calculus. In this question, we add integer constants, but no operations on integers.

For this question, you will complete a proof of type preservation. Instead of the call-by-value semantics used in the assignments and most of the lecture notes, in this question we will use a call-by-name semantics—obtained by adjusting the syntax of evaluation contexts.

First, we give a call-by-name semantics for the language of terms from Q1. We write \(Q\) for evaluation contexts to emphasize that the evaluation contexts in this question are not the same as call-by-value evaluation contexts \(C\), which would include a production \(v(C)\).

Values
\[ v ::= x \mid n \mid (\lambda x. N) \]

Evaluation contexts
\[ Q ::= \square \mid Q(M) \]

\(M \mapsto_R M'\) Term \(M\) reduces to \(M'\)
\[(\lambda x. M)(N) \mapsto_R [N/x]M \quad \text{red-app} \]

\(M \mapsto M'\) Term \(M\) takes one step to \(M'\)
\[Q[M] \mapsto Q[M'] \quad \text{step-context} \]

Second, we define a type system for the language of terms:

Types
\[ S, T ::= \text{int} \mid S \to S \]

Typing contexts
\[ \Gamma ::= \emptyset \mid \Gamma, x : S \]

\(\Gamma \vdash M : S\) Under typing context \(\Gamma\), term \(M\) has type \(S\)
\[(x : S) \in \Gamma \quad \text{ty-assum} \quad \Gamma \vdash x : S \quad \text{ty-int} \quad \Gamma, x : S \vdash N : T \quad \text{ty-abs} \quad \Gamma \vdash M : S \to T \quad \text{ty-app} \quad \Gamma \vdash N : S \quad \Gamma \vdash M(N) : T \]
Q3: Choice and Progress

Consider the language of terms extended with a choice operator \( \text{choose}(M_1, M_2) \), which steps to either \( M_1 \) or \( M_2 \), unpredictably. The new grammar is:

\[
\text{Variables} \quad x, y, z \\
\text{(Q3) Terms} \quad M, N ::= x \quad \text{variable} \\
\quad \quad \quad | (\lambda x. N) \quad \text{abstraction} \\
\quad \quad \quad | M(M) \quad \text{application} \\
\quad \quad \quad | n \quad \text{integer constant} \\
\quad \quad \quad | \text{choose}(M, M) \quad \text{choice}
\]

Adding the choice operator to the small-step semantics requires only that we add two reduction rules. The evaluation contexts don’t change: think of \( \text{choose}(M_1, M_2) \) as “if \( \text{random-bit}() \) is 0 then \( M_1 \) else \( M_2 \)”. We didn’t reduce inside an \( \text{Ite} \), so we don’t reduce inside \( \text{choose} \).

\[
\begin{align*}
\text{(Same as Q2) Values} & \quad v ::= x \mid n \mid (\lambda x. N) \\
\text{(Same as Q2) Evaluation contexts} & \quad Q ::= [] \\
& \quad \quad \quad | Q(M)
\end{align*}
\]

\[M \rightarrow_R M'\] Term \( M \) reduces to \( M' \)

\[
\begin{align*}
(\lambda x. M)(N) & \rightarrow_R [N/x]M & \text{red-app} \\
\text{choose}(M_1, M_2) & \rightarrow_R M_1 & \text{red-choose1} \\
\text{choose}(M_1, M_2) & \rightarrow_R M_2 & \text{red-choose2}
\end{align*}
\]