where \textit{succ} is the successor function with type \textit{int} \to \textit{int}. One possible "typing" under the QST typing rules is:

\[(\lambda x : \textit{int}. \textit{succ} x) \vdash \textit{false} \to \textit{true} \]

while the \textit{Type} rules correctly rejects the expression as ill typed.

3 Type Inference

In this section we present a typing system based on the \textit{Type} algorithm of section 2. Our system takes terms that may have incomplete type information, and infers a Partial Type for the term.

The object language we use is the same except that \textit{\lambda}-bound variables need not be given a type. The idea is to integrate plausibility checking into a typing system that does type inference, extending the work in [12]. An expression in this language is called a \textit{Qstein} expression.

Types need to be extended to include type variables, usually denoted by the first letters in the Greek alphabet. We denote with \textit{TV} the set of all type variables:

\[\tau ::= \alpha \mid \text{int} \mid \text{bool} \mid \tau \rightarrow \tau\]

The definition of the subtype relation is the same as in QST. We denote the set of all type expressions as \textit{TX}.

As mentioned above, we choose to base our type inference system on the \textit{Type} algorithm. The reason for this is the fact that \textit{Type} is more restrictive, as shown in section 2.

To derive a type for an expression, the typing rules use an "environment", in the same way ML does, and also a set of constraints of the form \(\tau_1 \leq \tau_2\). The set of constraints is modified by some of the typing rules. We call our system \textit{Qstein}.

\textbf{Definition 2} A type substitution is a function \(\sigma : \textit{TV} \rightarrow \textit{TX}\) such that \(\sigma(\alpha) = \alpha\) for almost all \(\alpha \in \textit{TV}\).

\textbf{Definition 3} Let \(\textit{C}\) be a set of type constraints. A solution of \(\textit{C}\) is a type substitution \(\sigma : \textit{V} \rightarrow \textit{TX}\) such that \(\textit{C} contains valid inequalities only.\)

\textbf{Definition 4} Let \(\textit{M}\) be an expression of the object language. We say that \(\textit{M}\) is typable under environment \(\textit{TE}\) and has type \(\tau\) in \textit{Qstein}, when \(\textit{TE}, \textit{C} \vdash \textit{Qstein} \textit{M} : \tau\), if and only if the set of constraints \(\textit{C}\) has a solution.

A set of constraints can have more than one solution, and a particular solution determines the type given to the expression. We are looking for the solution that gives a type as precise and "general" as possible. The typing rules are:

\[
\begin{align*}
\text{TE}, \; \alpha \vdash x : \text{TE}(\alpha) \\
\text{TE}, \; C \vdash M : \tau_1 \\
\text{TE}, \; C \vdash \lambda x : \tau_1. M : \tau_2
\end{align*}
\]

\[
\begin{align*}
\text{TE}[\alpha \equiv \tau_1], \; C \vdash M : \tau_2 \\
\text{TE}, \; C \vdash \lambda x : \tau_1. M : \tau_2
\end{align*}
\]

\[
\begin{align*}
\text{TE}, \; C_1 \vdash M : \tau_1, \; \text{TE}, \; C_2 \vdash N : \tau_2 \\
\text{TE}, \; C_1 \cup C_2 \cup \{\tau_1 \leq \tau_2, \; \tau_2 \leq \tau_1\} \vdash MN : \tau_1 \leq \tau_2
\end{align*}
\]

We illustrate how these rules work with two examples:

\textbf{Example 3} ((\lambda x . x)3)4

Consider the following derivation for the expression ((\lambda x . x)3)4:

\[
\begin{align*}
\{x : \alpha_x\} \vdash x : \alpha_x \\
\vdash \lambda x : \alpha_x. x : \alpha_x \\
\{\tau_1 \leq \text{int}, \; \tau_2 \leq \alpha_x\} \vdash (\lambda x : \alpha_x. x) : \alpha_x \\
\{\tau_1 \leq \text{int}, \; \tau_2 \leq \alpha_x, \; \tau_3 \leq \tau_2, \; \tau_4 \leq \text{int}\} \vdash ((\lambda x : \alpha_x. x) : \alpha_x) : \tau_3
\end{align*}
\]

The set of constraints for the given expression is:

\[
\textit{C} = \{\tau_1 \leq \alpha_x, \; \tau_2 \leq \text{int}, \; \tau_3 \leq \alpha_x, \; \tau_4 \leq \tau_2, \; \tau_4 \leq \text{int}\}
\]

which has, among others, the following solution \(\sigma\):

\[
\sigma(\alpha_x) = \Omega, \; \sigma(\tau_1) = \text{int}, \; \sigma(\tau_2) = \alpha_x, \; \sigma(\tau_3) = \Omega, \; \sigma(\tau_4) = \text{int}
\]

Using this typing together with Thalke's \textit{Type} algorithm we get the following expression:

\[(((\lambda x : \Omega . x)3)1) : \Omega
\]

\textbf{Example 4} (\lambda x . x)(\lambda y . y)

The set of constraints for the expression (\lambda x . x)(\lambda y . y) is: