Recitation 7: Dynamic Typing and Refinements

15-312: Principles of Programming Languages

Wednesday, February 26, 2014

For this handout, we consider PCF without sums and products but with an additional failure construct \texttt{fail} as the sole source of runtime errors (which are propagated as usual).

\[ \Gamma \vdash \texttt{fail} : \tau \]

We might want to use \texttt{fail} to handle cases like division by zero as a better alternative to nontermination or returning a bogus answer:

\[
\text{fix div:nat} \rightarrow \text{nat} \rightarrow \text{nat is} \nonumber \\n (m:\text{nat}) \fn (n:\text{nat}) \text{ifz } n \{ \text{z } \Rightarrow \text{fail } | \text{s(\_)} \Rightarrow \ldots \}
\]

We will use refinement types to exclude the possibility of failure. Under the refinement type system, we can give \texttt{div} the refinement \( T_{\text{nat}} \rightarrow \text{pos} \rightarrow T_{\text{nat}} \), which requires the second argument to be non-zero.

1 Refinements refine types

\[
\begin{array}{c}
T_{\tau} \leq \tau \\
\text{zero} \leq \text{nat} \\
\text{pos} \leq \text{nat} \\
\text{odd} \leq \text{nat} \\
\text{even} \leq \text{nat}
\end{array}
\]

\[
\frac{\varphi_1 \leq \tau_1 \quad \varphi_2 \leq \tau_2}{\varphi_1 \rightarrow \varphi_2 \leq \tau_1 \rightarrow \tau_2}
\]

\[
\frac{\varphi_1 \leq \tau \\ \varphi_2 \leq \tau}{\varphi_1 \land \varphi_2 \leq \tau}
\]

2 Refinement entailment

Any \texttt{nat} that satisfies refinement \texttt{zero} also satisfies refinement \texttt{even}, any refinement that satisfies refinement \texttt{odd} also satisfies refinement \texttt{pos}. This is captured by refinement entailment \( \varphi \leq \varphi' \).

\[
\begin{array}{c}
\varphi \leq \tau \quad \varphi \leq \tau \quad \text{zero} \leq \text{even} \\
\varphi \leq \varphi \\
\varphi \leq T_{\tau} \\
\text{odd} \leq \text{pos}
\end{array}
\]

\[
\begin{array}{c}
\varphi_1 \leq \varphi \\
\varphi_2 \leq \varphi
\end{array}
\]

\[
\begin{array}{c}
\varphi \leq \varphi_1 \\
\varphi \leq \varphi_2
\end{array}
\]

We want refinement entailment to be reflexive and transitive. We can either define these properties as rules or we can just require them to hold as theorems (or admissible rules) of our system. The subrefinement definition above has reflexivity as an explicit rule and has transivity as an admissible rule; the subrefinement definition in the homework has both as admissible rules.
3 Refinement checking

By writing the judgement \( x_1 \in \varphi_1, \ldots, x_n \in \varphi_n \vdash e \in \varphi \), we assert that we already know that \( x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau \), where \( \varphi \sqsubseteq \tau, \varphi_1 \sqsubseteq \tau_1, \ldots, \) and \( \varphi_n \sqsubseteq \tau_n \).

The rules for variables, functions, and fixpoints just match the type system, because we don’t have any interesting refinements directly applied to functions in this system.

\[
\frac{\sum, x \in \varphi_1 \vdash e \in \varphi_2}{\sum \vdash \text{fn}(x:\tau)e \in \varphi_1 \rightarrow \varphi_2} \quad \frac{\sum \vdash e_1 \in \varphi' \rightarrow \varphi}{\sum \vdash e_2 \in \varphi'} \quad \frac{\sum, x \in \varphi \vdash e \in \varphi}{\sum \vdash \text{fix} \; x : \tau \text{ is } e \in \varphi}
\]

There are two rules specific to refinement systems: refinement entailment and conjunction. Note that we do not have a rule allowing us to prove \( \sum \vdash e \in \top_\tau \), because this would mean we couldn’t exclude bad programs from our language.

\[
\frac{\sum \vdash e \in \varphi_1 \quad \sum \vdash e \in \varphi_2}{\sum \vdash e \in \varphi_1 \wedge \varphi_2} \quad \frac{\sum \vdash e \in \varphi_1 \quad \varphi_1 \leq \varphi_2}{\sum \vdash e \in \varphi_2}
\]

The interesting bit of this particular refinement system is in the introduction and elimination forms for \( \text{nat} \):

\[
\sum \vdash z \in \text{zero} \quad \sum \vdash e \in \text{even} \quad \sum \vdash e \in \text{odd} \\
\sum \vdash s(e) \in \text{pos} \quad \sum \vdash s(e) \in \text{odd} \quad \sum \vdash s(e) \in \text{even} \\
\sum \vdash \text{ifz} \; e \{ z \Rightarrow e_0 \mid s(x) \Rightarrow e_1 \} \in \varphi \\
\sum \vdash e \in \text{odd} \quad \sum, x \in \text{even} \vdash e_1 \in \varphi \\
\sum \vdash \text{ifz} \; e \{ z \Rightarrow e_0 \mid s(x) \Rightarrow e_1 \} \in \varphi \\
\sum \vdash e \in \text{even} \quad \sum \vdash e_0 \in \varphi \quad \sum, x \in \text{odd} \vdash e_1 \in \varphi \\
\sum \vdash \text{ifz} \; e \{ z \Rightarrow e_0 \mid s(x) \Rightarrow e_1 \} \in \varphi \\
\sum \vdash e \in \top_\text{nat} \\
\sum \vdash e_0 \in \varphi \\
\sum, x \in \top_\text{nat} \vdash e_1 \in \varphi \\
\sum \vdash \text{ifz} \; e \{ z \Rightarrow e_0 \mid s(x) \Rightarrow e_1 \} \in \varphi
\]