# Recitation 7:<br>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 `fail` as the sole source of runtime errors (which are propogated as usual).

$$\frac{}{\Gamma \vdash \texttt{fail} : \tau} \qquad\qquad \frac{}{\texttt{fail}\ \mathsf{err}}$$

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

$$\begin{array}{l}\texttt{fix}\ \mathit{div}{:}\texttt{nat} \rightharpoonup \texttt{nat} \rightharpoonup \texttt{nat}\ \texttt{is} \\ \quad \texttt{fn}\,(m{:}\texttt{nat})\,\texttt{fn}\,(n{:}\texttt{nat})\,\texttt{ifz}\ n\ \{\texttt{z} \Rightarrow \texttt{fail} \mid \texttt{s}(\_) \Rightarrow \ldots\}\end{array}$$

We will use refinement types to *exclude* the possibility of failure. Under the refinement type system, we can give *div* the refinement $\top_{\texttt{nat}} \rightharpoonup \texttt{pos} \rightharpoonup \top_{\texttt{nat}}$, which requires the second argument to be non-zero.

## 1 Refinements refine types

$$\frac{}{\top_\tau \trianglelefteq \tau} \qquad \frac{}{\texttt{zero} \trianglelefteq \texttt{nat}} \qquad \frac{}{\texttt{pos} \trianglelefteq \texttt{nat}} \qquad \frac{}{\texttt{odd} \trianglelefteq \texttt{nat}} \qquad \frac{}{\texttt{even} \trianglelefteq \texttt{nat}}$$

$$\frac{\varphi_1 \trianglelefteq \tau_1 \qquad \varphi_2 \trianglelefteq \tau_2}{\varphi_1 \rightharpoonup \varphi_2 \trianglelefteq \tau_1 \rightharpoonup \tau_2} \qquad\qquad \frac{\varphi_1 \trianglelefteq \tau \qquad \varphi_2 \trianglelefteq \tau}{\varphi_1 \wedge \varphi_2 \trianglelefteq \tau}$$

## 2 Refinement entailment

Any `nat` that satisfies refinement `zero` also satisfies refinement `even`, any refinement that satisfies refinement `odd` also satisfies refinment `pos`. This is captured by refinement entailment $\varphi \leq \varphi'$.

$$\frac{\varphi \trianglelefteq \tau}{\varphi \leq \varphi} \qquad \frac{\varphi \trianglelefteq \tau}{\varphi \leq \top_\tau} \qquad \frac{}{\texttt{zero} \leq \texttt{even}} \qquad \frac{}{\texttt{odd} \leq \texttt{pos}} \qquad \frac{\varphi_1' \leq \varphi_1 \qquad \varphi_2 \leq \varphi_2'}{\varphi_1 \rightharpoonup \varphi_2 \leq \varphi_1' \rightharpoonup \varphi_2'}$$

$$\frac{\varphi_1 \leq \varphi}{\varphi_1 \wedge \varphi_2 \leq \varphi} \qquad \frac{\varphi_2 \leq \varphi}{\varphi_1 \wedge \varphi_2 \leq \varphi} \qquad \frac{\varphi \leq \varphi_1 \qquad \varphi \leq \varphi_2}{\varphi \leq \varphi_1 \wedge \varphi_2}$$

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 $\varphi \trianglelefteq \tau$, $\varphi_1 \trianglelefteq \tau_1$, …, and $\varphi_n \trianglelefteq \tau_n$.

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.

$$\frac{\Sigma, x \in \varphi_1 \vdash e \in \varphi_2}{\Sigma \vdash \mathtt{fn}\,(x{:}\tau)\, e \in \varphi_1 \rightharpoonup \varphi_2} \qquad \frac{\Sigma \vdash e_1 \in \varphi' \rightharpoonup \varphi \qquad \Sigma \vdash e_2 \in \varphi'}{\Sigma \vdash e_1\ (e_2) \in \varphi} \qquad \frac{\Sigma, x \in \varphi \vdash e \in \varphi}{\Sigma \vdash \mathtt{fix}\; x{:}\tau\; \mathtt{is}\; e \in \varphi}$$

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

$$\frac{\Sigma \vdash e \in \varphi_1 \qquad \Sigma \vdash e \in \varphi_2}{\Sigma \vdash e \in \varphi_1 \wedge \varphi_2} \qquad \frac{\Sigma \vdash e \in \varphi_1 \qquad \varphi_1 \leq \varphi_2}{\Sigma \vdash e \in \varphi_2}$$

The interesting bit of this particular refinement system is in the introduction and elimination forms for `nat`:

$$\frac{}{\Sigma \vdash \mathtt{z} \in \mathtt{zero}} \qquad \frac{\Sigma \vdash e \in \top_{\mathtt{nat}}}{\Sigma \vdash \mathtt{s}(e) \in \mathtt{pos}} \qquad \frac{\Sigma \vdash e \in \mathtt{even}}{\Sigma \vdash \mathtt{s}(e) \in \mathtt{odd}} \qquad \frac{\Sigma \vdash e \in \mathtt{odd}}{\Sigma \vdash \mathtt{s}(e) \in \mathtt{even}}$$

$$\frac{\Sigma \vdash e \in \mathtt{zero} \qquad \Sigma \vdash e_0 \in \varphi}{\Sigma \vdash \mathtt{ifz}\, e\, \{\mathtt{z} \Rightarrow e_0 \mid \mathtt{s}(x) \Rightarrow e_1\} \in \varphi} \qquad \frac{\Sigma \vdash e \in \mathtt{pos} \qquad \Sigma, x \in \top_{\mathtt{nat}} \vdash e_1 \in \varphi}{\Sigma \vdash \mathtt{ifz}\, e\, \{\mathtt{z} \Rightarrow e_0 \mid \mathtt{s}(x) \Rightarrow e_1\} \in \varphi}$$

$$\frac{\Sigma \vdash e \in \mathtt{odd} \qquad \Sigma, x \in \mathtt{even} \vdash e_1 \in \varphi}{\Sigma \vdash \mathtt{ifz}\, e\, \{\mathtt{z} \Rightarrow e_0 \mid \mathtt{s}(x) \Rightarrow e_1\} \in \varphi} \qquad \frac{\Sigma \vdash e \in \mathtt{even} \qquad \Sigma \vdash e_0 \in \varphi \qquad \Sigma, x \in \mathtt{odd} \vdash e_1 \in \varphi}{\Sigma \vdash \mathtt{ifz}\, e\, \{\mathtt{z} \Rightarrow e_0 \mid \mathtt{s}(x) \Rightarrow e_1\} \in \varphi}$$

$$\frac{\Sigma \vdash e \in \top_{\mathtt{nat}} \qquad \Sigma \vdash e_0 \in \varphi \qquad \Sigma, x \in \top_{\mathtt{nat}} \vdash e_1 \in \varphi}{\Sigma \vdash \mathtt{ifz}\, e\, \{\mathtt{z} \Rightarrow e_0 \mid \mathtt{s}(x) \Rightarrow e_1\} \in \varphi}$$