Semantics of Functional Programming

Formalising PCF in Dependent Type Theory

Chen, Liang-Ting

lxc@iis.sinica.edu.tw

Institute of Information Science, Academia Sinica

Formosan Summer School on Logic, Language, and Computation 2014

Formalising **PCF**

Terms, types, and lists are introduced as (non-dependent) types. For example, the type for **PCF** types are introduced as:

- Formation:

$$\frac{}{\vdash \mathtt{Type} : \mathcal{U}}$$

- Introduction:

$$\frac{}{\vdash \mathtt{nat} : \mathtt{Type}} \qquad \frac{\vdash \tau_1 : \mathtt{Type} \qquad \vdash \tau_2 : \mathtt{Type}}{\vdash \tau_1 \Rightarrow \tau_2 : \mathtt{Type}}$$

Exercise. Define types $\mathtt{Term}, \mathtt{Type}, \mathtt{Cxt}$ for **PCF** terms, **PCF** types, and contexts respectively in **Agda**.

Predicates

In case that you have been polluted by set theory, we distinguish a few set-theoretic and type-theoretic notions.

In set theory

A **predicate** P over a set X is a subset $P \subseteq X$.

In type theory

A term P is a **predicate** over a type A if and only if

$$\Gamma \vdash P : A \to \mathcal{U}$$

A term f is a **membership function** if and only if

$$\Gamma \vdash p : A \to \mathbf{Bool}$$

An example of predicates

In set theory, an **even number** n is commonly defined as a natural number satisfying $n = 2k$ for some natural number k, i.e.

$$E_{\mathbb{N}} = \{\, n \in \mathbb{N} \mid \exists k \in \mathbb{N}.\, n = 2k \,\}.$$

In type theory, it can be defined inductively as a predicate $\texttt{even} : \mathbb{N} \to \mathcal{U}$ by

- Formation:

$$\frac{}{\Gamma \vdash \textbf{even} : \mathbb{N} \to \mathcal{U}}$$

- Introduction:

$$\frac{}{\Gamma \vdash \texttt{e-zero} : \textbf{even}\ \texttt{zero}} \qquad \frac{\Gamma \vdash p : \texttt{even}\ n}{\Gamma \vdash \texttt{e-suc}\ p : \textbf{even}\ (\texttt{suc}\ (\texttt{suc}\ n))}$$

where the elimination rule and the computational rule are omitted.

Exercise. Define **Val** : $\texttt{Term} \to \mathcal{U}$ for values of **PCF** terms.

Set-theoretic relations

A **relation** over a set X is a subset $R \subseteq X \times X$, and $(x_1, x_2) \in R$ is written as

$$x_1 \; R \; x_2.$$

A relation $R \subseteq X \times X$ is

- **reflexive** if $x \, R \, x$ for every $x \in X$.
- **transitive** if $x \, R \, z$ whenever $x \, R \, y$ and $y \, R \, z$

A **reflexive transitive closure** of a relation R is the smallest reflexive transitive relation R^* containing R:

$$R^* := \bigcap \{\, S \subseteq X \times X \mid R \subseteq S \text{ and } S \text{ is reflexive and transitive} \,\}$$

Type-theoretic relations

A **relation** over a type (set) A is a judgement

$$\Gamma \vdash R : A \to A \to \mathcal{U}.$$

A relation is

- **reflexive** if and only if

$$\Pi[x : A]\ R\ x\ x$$

- **transitive** if and only if

$$\Pi[x : A]\ \Pi[y : A]\ \Pi[z : A]\ R\ x\ y \to R\ y\ z \to R\ x\ z$$

Exercise. Define the one-step reduction $\_\rightsquigarrow\_$ over `Term`.

A **reflexive transitive closure** R^* of a relation R over A:

- Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash R : A \to A \to \mathcal{U}}{\Gamma \vdash R^* : A \to A \to \mathcal{U}}$$

- Introduction:

$$\frac{\Gamma \vdash x : A}{\Gamma \vdash \texttt{refl}\ x : R^*\ x\ x}$$

$$\frac{\begin{array}{l}\Gamma \vdash x : A \\ \Gamma \vdash y : A \qquad \Gamma \vdash t : R\ x\ y \qquad \Gamma \vdash u : R^*\ y\ z \\ \Gamma \vdash z : A\end{array}}{\Gamma \vdash \texttt{trans}\ t\ u : R^*\ x\ z}$$

where the elimination rule and the computation rule are omitted.

Exercise. Show the following statements in `Transitive-Closure.agda`.

1. R^* is reflexive and transitive for every relation R over A.
2. R^* is the "smallest" transitive reflexive relation containing R.

Judgements in type theory

A **judgement** is a ternary predicate

$$\_\vdash\_:\_ : \mathtt{Cxt} \to \mathtt{Term} \to \mathtt{Type} \to \mathcal{U}.$$

in type theory.

The introduction rule for $\mathtt{suc}$ in **PCF** is formalised as

```
suc : ∀ {Γ e}
      → Γ ⊢ e : nat
      ------------------- (s)
      → Γ ⊢ suc e : nat
```

Exercise. Define a type of the typing rules of **PCF**.

Progress Theorem in type theory

Recall Progress Theorem in **PCF**:

> *Every closed well-typed **PCF** term* M *is either a value or there exists another term* M' *such that* $\mathsf{M} \rightsquigarrow \mathsf{M}'$.

which corresponds to a witness of

$$\Pi[\mathsf{M} : \mathtt{Term}]\Pi[\tau : \mathtt{Type}]$$
$$[\,] \vdash \mathsf{M} : \tau \to (\textbf{Val}\ \mathsf{M}) + \Sigma[\mathsf{M}' : \mathtt{Term}]\ \mathsf{M} \rightsquigarrow \mathsf{M}'$$

Preservation Theorem in type theory

Similarly, Preservation Theorem

> *For every closed well-typed **PCF** term* M *of type* τ *with* M $\rightsquigarrow$ N, *the term* N *is also of type* τ.

is translated to a term of type

$$\Pi[\mathsf{M} : \mathtt{Term}]\Pi[\mathsf{N} : \mathtt{Term}]\Pi[\tau : \mathtt{Type}]$$
$$[\,] \vdash \mathsf{M} : \tau \to \mathsf{M} \rightsquigarrow \mathsf{N} \to [\,] \vdash \mathsf{N} : \tau$$

Exercise. Finish `PCF_blank.agda`.