

# 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:

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

- Introduction:

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

**Exercise.** Define types `Term`, `Type`, `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

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

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

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

## 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  $\text{even} : \mathbb{N} \rightarrow \mathcal{U}$  by

- Formation:

$$\frac{}{\Gamma \vdash \text{even} : \mathbb{N} \rightarrow \mathcal{U}}$$

- Introduction:

$$\frac{}{\Gamma \vdash \text{zero} : \text{even zero}} \qquad \frac{\Gamma \vdash p : \text{even } n}{\Gamma \vdash \text{suc } p : \text{even (suc (suc } n))}$$

where the elimination rule and the computational rule are omitted.

**Exercise.** Define **Val** 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

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

A relation is

- **reflexive** if and only if

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

- **transitive** if and only if

$$\prod [x : A] \prod [y : A] \prod [z : A] R \ x \ y \rightarrow R \ y \ z \rightarrow R \ x \ z$$

**Exercise.** Define the one-step reduction  $\_ \rightsquigarrow \_$  over  $\text{Term}$ .

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

■ Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash R : A \rightarrow A \rightarrow \mathcal{U}}{\Gamma \vdash R^* : A \rightarrow A \rightarrow \mathcal{U}}$$

■ Introduction:

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

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

$$\_ \vdash \_ : \_ : \text{Cxt} \rightarrow \text{Term} \rightarrow \text{Type} \rightarrow \mathcal{U}.$$

in type theory.

The introduction rule for `suc` in **PCF** is formalised as

$$\begin{array}{c} \text{suc} : \forall \{ \Gamma \ e \} \\ \rightarrow \Gamma \vdash e : \text{nat} \\ \hline \rightarrow \Gamma \vdash \text{suc } e : \text{nat} \end{array} \quad (\text{s})$$

**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  $M \rightsquigarrow M'$ .*

which corresponds to a witness of

$$\begin{aligned} & \Pi[M : \text{Term}] \Pi[\tau : \text{Type}] \\ & [\ ] \vdash M : \tau \rightarrow (\mathbf{Val} \ M) + \Sigma[M' : \text{Term}] \ M \rightsquigarrow M' \end{aligned}$$

# Preservation Theorem in type theory

Similarly, Preservation Theorem

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

is translated to a term of type

$$\prod[M : \text{Term}] \prod[N : \text{Term}] \prod[\tau : \text{Type}] \\ [] \vdash M : \tau \rightarrow M \rightsquigarrow N \rightarrow [] \vdash N : \tau$$

**Exercise.** Finish `PCF_blank.agda`.