Semantics of Functional Programming

Lecture I: **PCF** and its Operational Semantics

Chen, Liang-Ting
lxc@iis.sinica.edu.tw

Formosan Summer School on Logic, Language, and Computation 2014

Overview

In this lecture, we will present simply typed lambda calculus in a different manner, where terms and typing rules are introduced separately. In this approach, terms might not be well-typed at all.

Then, we discuss its computational meaning by **one-step reduction** and define many-step reduction. Later we introduce the concept of **type safety**.

Finally, we extend simply typed lambda calculus with natural numbers and general recursion. This extension is called **PCF**, *Programming Computable Functional*. We formalise new features by what we have learnt later.

1 Simply typed lambda calculus à la Curry

The approach *à la* Curry

We introduce a different approach to simply lambda calculus where terms and typing rules are introduced separately.

$$\frac{x \ \mathbf{var}}{x \ \mathbf{term}}$$

$$\frac{}{\Gamma, x : \sigma, \Delta \vdash x : \sigma} \ (\text{var})$$

$$\frac{x \ \mathbf{var} \qquad \mathsf{M} \ \mathbf{term}}{\lambda x.\, \mathsf{M} \ \mathbf{term}}$$

$$\frac{\Gamma, x : \sigma \vdash \mathsf{M} : \tau}{\Gamma \vdash \lambda x.\, \mathsf{M} : \sigma \to \tau} \ (\text{abs})$$

$$\frac{\mathsf{M} \ \mathbf{term} \qquad \mathsf{N} \ \mathbf{term}}{\mathsf{M}\ \mathsf{N} \ \mathbf{term}}$$

$$\frac{\Gamma \vdash \mathsf{M} : \sigma \to \tau \qquad \Gamma \vdash \mathsf{N} : \sigma}{\Gamma \vdash \mathsf{M}\ \mathsf{N} : \tau} \ (\text{app})$$

The existence of ill-typed terms

In contrast the approach *á la* Church where every term is introduced with a type, there are ill-typed terms in the approach *à la* Curry:

Example 1. $(\lambda x.\, x)\ (\lambda x.\, x)$ is a term if x is a variable, because

$$\frac{\dfrac{x \ \mathbf{var} \qquad \dfrac{x \ \mathbf{var}}{x \ \mathbf{term}}}{\lambda x.\, x \ \mathbf{term}} \qquad \dfrac{x \ \mathbf{var} \qquad \dfrac{x \ \mathbf{var}}{x \ \mathbf{term}}}{\lambda x.\, x \ \mathbf{term}}}{(\lambda x.\, x)\ (\lambda x.\, x) \ \mathbf{term}}$$

However, $(\lambda x.\, x)\ (\lambda x.\, x)$ cannot be assigned a type unless $\sigma \to \sigma = \sigma$.

Reduction

One-step reduction relation $\rightsquigarrow$ between terms is introduced to describe the flow of computation from a term to another term in a single step, regardless of types.

$$\frac{\mathsf{M} \rightsquigarrow \mathsf{M}'}{\mathsf{M}\ \mathsf{N} \rightsquigarrow \mathsf{M}'\ \mathsf{N}} \ (\rightsquigarrow\text{-lapp})$$

$$\frac{}{(\lambda x.\, \mathsf{M})\ \mathsf{N} \rightsquigarrow \mathsf{M}[\mathsf{N}/x]} \ (\rightsquigarrow\text{-app})$$

Example 2. $(\lambda x.\, \lambda y.\, x)\ \mathsf{M}\ \mathsf{N}$ can be reduced to M by the following derivation

$$\frac{\dfrac{}{(\lambda x.\, \lambda y.\, x)\ \mathsf{M} \rightsquigarrow (\lambda y.\, \mathsf{M})} \ (\rightsquigarrow\text{-app})}{((\lambda x.\, \lambda y.\, x)\ \mathsf{M})\ \mathsf{N} \rightsquigarrow (\lambda y.\, \mathsf{M})\ \mathsf{N}} \ (\rightsquigarrow\text{-lapp})$$

Many-step reduction

As we will mostly discuss a sequence of reductions, it is convenient to define another relation $\rightsquigarrow^*$ so that $\mathsf{M} \rightsquigarrow^* \mathsf{N}$ means M reduces to N in finitely many steps.

Definition 3. The many-step reduction relation $\rightsquigarrow^*$ is defined inductively by

$$\frac{}{\mathsf{M} \rightsquigarrow^* \mathsf{M}} \qquad \frac{\mathsf{M}_1 \rightsquigarrow \mathsf{M}_2 \qquad \mathsf{M}_2 \rightsquigarrow^* \mathsf{M}_3}{\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_3}$$

Proposition 4 (Reflexivity of $\rightsquigarrow^*$)**.** *For every term* M, $\mathsf{M} \rightsquigarrow^* \mathsf{M}$.

For example, one has

$$(\lambda x.\, \lambda y.\, x)\ \mathsf{M}\ \mathsf{N} \rightsquigarrow^* (\lambda y.\, \mathsf{M})\ \mathsf{N}$$

by the derivation

$$\cfrac{\cfrac{\cfrac{}{(\lambda x.\, \lambda y.\, x)\ \mathsf{M} \rightsquigarrow (\lambda y.\, \mathsf{M})}}{((\lambda x.\, \lambda y.\, x)\ \mathsf{M})\ \mathsf{N} \rightsquigarrow (\lambda y.\, \mathsf{M})\ \mathsf{N}} \quad (\lambda y.\, \mathsf{M})\ \mathsf{N} \rightsquigarrow^* (\lambda y.\, \mathsf{M})\ \mathsf{N}}{(\lambda x.\, y.\, x)\ \mathsf{M}\ \mathsf{N} \rightsquigarrow^* (\lambda y.\, \mathsf{M})\ \mathsf{N}}$$

Exercise. Evaluate the following terms (formally or informally).

1. $(\lambda x.\, x)\ y$
2. $(\lambda x.\, x\ x)\ (\lambda x.\, x\ x)$
3. $(\lambda x.\, \lambda y.\, \lambda z.\, y)\ \mathsf{M}_0\ \mathsf{M}_1\ \mathsf{M}_2$

Induction on derivation

Every instance of $\mathsf{M} \rightsquigarrow^* \mathsf{N}$ must be constructed by one of cases, so we can analyse its structure case by case.

Proposition 5 (Transitivity of $\rightsquigarrow^*$)**.** *For every three terms* M_0, M_1, *and* M_2, *if* $\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_2$ *and* $\mathsf{M}_2 \rightsquigarrow^* \mathsf{M}_3$, *then* $\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_3$.

Given derivations of $\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_2$ and $\mathsf{M}_2 \rightsquigarrow^* \mathsf{M}_3$, we do case analysis on the derivation of $\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_2$. Also, we can assume that the premise satisfy this property, that is, the induction hypothesis.

Proof. 1. For $\cfrac{}{\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_1}$, it unifies M_2 to M_1, so the given derivation $\mathsf{M}_2 \rightsquigarrow^* \mathsf{M}_3$ is just the goal derivation as $\mathsf{M}_1 = \mathsf{M}_2$.

2. For $\cfrac{\mathsf{M}_1 \rightsquigarrow \mathsf{M} \qquad \mathsf{M} \rightsquigarrow^* \mathsf{M}_2}{\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_2}$, we infer that $\mathsf{M} \rightsquigarrow^* \mathsf{M}_3$ by induction hypothesis, so we derive the goal

$$\cfrac{\mathsf{M}_1 \rightsquigarrow \mathsf{M} \qquad \mathsf{M} \rightsquigarrow^* \mathsf{M}_3}{\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_3}$$

□

Similarly, we can do induction on the formulation of terms, typing rules, and any other inductive definitions.

Exercise. Show that if $\mathsf{M} \rightsquigarrow^* \mathsf{M}'$ then $\mathsf{M}\ \mathsf{N} \rightsquigarrow^* \mathsf{M}'\ \mathsf{N}$ for any term N by induction on the derivation of $\mathsf{M} \rightsquigarrow^* \mathsf{M}'$.

Reductions on ill-typed terms

Reductions can be applied to ill-typed terms and sometimes it reduces to a well-typed closed term!

$$(\lambda x.\, x)\ (\lambda x.\, x) \rightsquigarrow^* (\lambda x.\, x)$$

On the other hand, the reduction of ill-typed terms may not stop at all.

$$\begin{aligned}(\lambda x.\, x\ x)\ (\lambda x.\, x\ x) &\rightsquigarrow (x\ x)[(\lambda x.\, x\ x)/x] \\ &= (\lambda x.\, x\ x)\ (\lambda x.\, x\ x) \rightsquigarrow \cdots\end{aligned}$$

Type safety

In contrast to ill-typed terms, well-typed closed terms have some nice properties. First, every well-typed closed term can be reduced further or it is a *value*.

Theorem 6 (Progress Theorem)**.** *If* $\vdash \mathsf{M} : \tau$, *then either* $\mathsf{M} \rightsquigarrow \mathsf{M}'$ *for some* M' *or* $\mathsf{M} = \lambda x.\, \mathsf{M}'$.

To show this property, we do the structural induction on the derivation of $\vdash \mathsf{M} : \tau$ and either produce a derivation of $\mathsf{M} \rightsquigarrow \mathsf{M}'$ or show that $\mathsf{M} = \lambda x.\, \mathsf{M}'$.

Proof. 1. $\vdash \mathsf{M} : \tau$ cannot be given by $\cfrac{}{\Gamma, x : \sigma, \Delta \vdash x : \sigma}$, since the context is empty.

2. For that case $\cfrac{x : \sigma \vdash \mathsf{M} : \tau}{\vdash \lambda x.\, \mathsf{M} : \sigma \to \tau}$ (abs), $(\lambda x.\, \mathsf{M}') \rightsquigarrow^* (\lambda x.\, \mathsf{M})$ we have already given a term in this form $\lambda x.\, \mathsf{M}$.

3. For $\cfrac{\vdash \mathsf{M} : \sigma \to \tau \qquad \vdash \mathsf{N} : \sigma}{\vdash \mathsf{M}\ \mathsf{N} : \tau}$ (app), by introduction hypothesis either $\mathsf{M} \rightsquigarrow \mathsf{M}'$ for some M' or $\mathsf{M} = \lambda x.\, \mathsf{M}'$. For the former case, we apply ($\rightsquigarrow$-lapp):

$$\cfrac{\mathsf{M} \rightsquigarrow \mathsf{M}'}{\mathsf{M}\ \mathsf{N} \rightsquigarrow \mathsf{M}'\ \mathsf{N}}$$

For the later case, we apply ($\rightsquigarrow$-app)

$$\cfrac{}{(\lambda x.\, \mathsf{M}')\ \mathsf{N} \rightsquigarrow \mathsf{M}'[\mathsf{N}/x]}$$

□

Moreover, the type of a well-typed closed term is always preserved by reductions:

Theorem 7 (Preservation Theorem)**.** *If* $\vdash \mathsf{M} : \tau$ *and* $\mathsf{M} \rightsquigarrow \mathsf{M}'$, *then* $\vdash \mathsf{M}' : \tau$.

However, to show this property, we need the following lemma saying that types are preserved by substitution.

Lemma 8 (Substitution Lemma)**.** *If* $\Gamma, x : \sigma \vdash \mathsf{M} : \tau$ *and* $\Gamma \vdash \mathsf{N} : \sigma$*, then* $\Gamma \vdash \mathsf{M}[\mathsf{N}/x] : \tau$*.*

By the introduction on the derivation of $\vdash \mathsf{M} : \tau$ and $\mathsf{M} \rightsquigarrow \mathsf{M}'$ at the same time.

Proof of Preservation Theorem.
1. $\vdash \mathsf{M} : \tau$ cannot be constructed by (var), since the context is empty.
2. For $\dfrac{x : \sigma \vdash \mathsf{M} : \tau}{\vdash \lambda x.\, \mathsf{M} : \sigma \to \tau}$, there is no reduction rule for $\lambda x.\, \mathsf{M}$, so a derivation $(\lambda x.\, \mathsf{M}) \rightsquigarrow \mathsf{M}'$ cannot exist.
3. For $\dfrac{\vdash \mathsf{M} : \sigma \to \tau \qquad \vdash \mathsf{N} : \sigma}{\vdash \mathsf{M}\ \mathsf{N} : \tau}$, we do induction on the derivation of $\mathsf{M}\ \mathsf{N} \rightsquigarrow \mathsf{M}'$.

□

Summary

A functional programming language consists of

1. type formulation rules,
2. term formulation rules,
3. typing rules, and
4. one-step reduction rules.

In particular, well-typed closed terms share type safety:

Progress Theorem for every well-typed closed term, it either can be reduced further or is a value;

Preservation Theorem for every well-typed closed term, its type is preserved by reduction.

Next, we add some features to simply typed lambda calculus and type safety remains.

2 Programming with typed recursion

Introduction to PCF

PCF, which stands for **Programming Computable Functionals**, is a functional programming language and it consists of

1. simply typed lambda calculus,
2. natural numbers, and
3. general recursion (to be explained).

We will introduce the later two features step by step.

It has two rules of type formulation:

$$\frac{}{\mathtt{nat}\ \mathbf{set}}$$

$$\frac{\tau_1\ \mathbf{set} \qquad \tau_2\ \mathbf{set}}{\tau_1 \to \tau_2\ \mathbf{set}}$$

Still, 'set' is a synonyms of 'type'.

Term formulation, typing, and reduction for natural numbers

Every natural number is either `zero` or a successor of some natural number.

$$\frac{}{\mathtt{zero}\ \mathbf{term}}$$

$$\frac{\mathsf{M}\ \mathbf{term}}{\mathtt{suc}\ \mathsf{M}\ \mathbf{term}}$$

$$\frac{}{\Gamma \vdash \mathtt{zero} : \mathtt{nat}}\ (\mathrm{z})$$

$$\frac{\Gamma \vdash \mathsf{M} : \mathtt{nat}}{\Gamma \vdash \mathtt{suc}\ \mathsf{M} : \mathtt{nat}}\ (\mathrm{s})$$

The reduction of $(\mathtt{suc}\ \mathsf{M})$ is given by its subterm M:

$$\frac{\mathsf{M} \rightsquigarrow \mathsf{M}'}{\mathtt{suc}\ \mathsf{M} \rightsquigarrow \mathtt{suc}\ \mathsf{M}'}\ (\rightsquigarrow\text{-}\mathtt{suc})$$

Values: canonical elements

Value are basic forms of term of each kind of types and they are defined independent of their types in the approach *à la* Curry.

Definition 9. A **value** is a term of the following form:

$$\frac{}{\mathtt{zero}\ \mathbf{val}} \qquad \frac{\mathsf{M}\ \mathbf{val}}{\mathtt{suc}\ \mathsf{M}\ \mathbf{val}} \qquad \frac{\mathsf{M}\ \mathbf{term}}{\lambda x.\, \mathsf{M}\ \mathbf{val}}$$

Define **numerals** $\underline{0}$ for `zero` and $\underline{n+1}$ for $\mathtt{suc}\ \underline{n}$ inductively.

Example 10. By this formulation, we have well-typed values $\mathtt{suc}\ (\mathtt{suc}\ \mathtt{zero})$, $\lambda x.\, \mathtt{suc}\ x$, and $\lambda x.\, x$, and also ill-typed values $\mathtt{suc}\ \lambda x.\, x$, $\lambda y.\, y\ y$.

Moreover, we can do branching according to the argument is zero or not.

$$\frac{\mathsf{M}\ \mathbf{term} \qquad \mathsf{M}_0\ \mathbf{term} \qquad x\ \mathbf{var} \qquad \mathsf{M}_1\ \mathbf{term}}{\mathtt{ifz}(\mathsf{M}; \mathsf{M}_0; x.\, \mathsf{M}_1)\ \mathbf{term}}$$

$$\frac{\Gamma \vdash \mathsf{M} : \mathtt{nat} \qquad \Gamma \vdash \mathsf{M}_0 : \tau \qquad \Gamma, x : \mathtt{nat} \vdash \mathsf{M}_1 : \tau}{\Gamma \vdash \mathtt{ifz}(\mathsf{M}; \mathsf{M}_0; x.\, \mathsf{M}_1) : \tau}\ (\mathrm{ifz})$$

accompanying with three reductions rules

$$\frac{\mathsf{M} \rightsquigarrow \mathsf{M}'}{\mathtt{ifz}(\mathsf{M}; \mathsf{M}_0; x.\, \mathsf{M}_1) \rightsquigarrow \mathtt{ifz}(\mathsf{M}'; \mathsf{M}_0; x.\, \mathsf{M}_1)}\ (\rightsquigarrow\text{-}\mathtt{ifz})$$

$$\frac{}{\mathtt{ifz}(\mathtt{zero}; \mathsf{M}_0; x.\, \mathsf{M}_1) \rightsquigarrow \mathsf{M}_0}\ (\rightsquigarrow\text{-}\mathtt{ifz}_0)$$

$$\frac{\mathtt{suc}\ \mathsf{M}\ \mathbf{val}}{\mathtt{ifz}(\mathtt{suc}\ \mathsf{M}; \mathsf{M}_0; x.\, \mathsf{M}_1) \rightsquigarrow \mathsf{M}_1[\mathsf{M}/x]}\ (\rightsquigarrow\text{-}\mathtt{ifz}_1)$$

Example: predecessor

The predecessor of natural numbers can be defined as

$$\texttt{pred} \coloneqq \lambda x.\, \texttt{ifz}(x; \underline{0}; y.\, y) : \texttt{nat} \to \texttt{nat}$$

with the following typing derivation:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : \texttt{nat}} \quad \dfrac{}{\Gamma \vdash \underline{0} : \texttt{nat}} \quad \dfrac{}{\Gamma, y : \texttt{nat} \vdash y : \texttt{nat}}}{\Gamma \vdash \texttt{ifz}(x; \underline{0}; y.\, y) : \texttt{nat}}}{\vdash \lambda x.\, \texttt{ifz}(x; \underline{0}; y.\, y) : \texttt{nat} \to \texttt{nat}}$$

where $\Gamma \coloneqq x : \texttt{nat}$.

Exercise.

1. Show that $\texttt{pred}\ \underline{0} \rightsquigarrow^* \underline{0}$ and $\texttt{pred}\ \underline{n+1} \rightsquigarrow^* \underline{n}$ by induction on $\underline{0}$.
2. Define $\texttt{flip} \colon \texttt{nat} \to \texttt{nat}$ such that $\texttt{flip}\ \underline{0} \rightsquigarrow^* \underline{1}$ and $\texttt{flip}\ \underline{n+1} \rightsquigarrow^* \underline{0}$.

Term formulation, typing rule, and reduction for general recursion

The $\texttt{Y}$ operator, used to do general recursion, has the same term formulation as λ-abstraction and a similar typing rules.

$$\frac{x \ \textbf{var} \qquad \mathsf{M} \ \textbf{term}}{\texttt{Y}x.\, \mathsf{M} \ \textbf{term}}$$

$$\frac{\Gamma, x : \sigma \vdash \mathsf{M} : \sigma}{\Gamma \vdash \texttt{Y}x.\, \mathsf{M} : \sigma} \ (\text{Y})$$

Each occurrence of $\texttt{Y}x.\, \mathsf{M}$ reduces to an substitution of x in M by itself:

$$\frac{}{\texttt{Y}x.\, \mathsf{M} \rightsquigarrow \mathsf{M}[\texttt{Y}x.\, M/x]} \ (\rightsquigarrow\text{-fix})$$

Example 11 (Divergent term). Consider the term $\texttt{Y}x.\, x$ which never reduces to any value

$$\texttt{Y}x.\, x \rightsquigarrow x[\texttt{Y}x.\, x] = \texttt{Y}x.\, x \rightsquigarrow \texttt{Y}x.\, x \rightsquigarrow \cdots$$

Example: calculating the factorials

The factorial of n is usually defined recursively

$$\texttt{fact} \colon n \mapsto \begin{cases} 1 & \text{if } n = 0 \\ n \times \texttt{fact}(n') & \text{if } n = n' + 1 \end{cases}$$

This is a *fixpoint* of the higher-order function $F \colon (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ defined by

$$F(f) \colon n \mapsto \begin{cases} 1 & \text{if } n = 0 \\ n \times f(n') & \text{if } n = n' + 1 \end{cases}$$

for any $f \colon \mathbb{N} \to \mathbb{N}$, satisfying $F(\texttt{fact}) = \texttt{fact}$.

The higher-order function $F \colon (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ can be presented in **PCF** as

$$\begin{aligned} \lambda.f\, F \coloneqq \lambda f.& \\ &\lambda n. \\ &\quad \texttt{ifz}(n; \underline{1}; m.\, n \times (f\ m)) \end{aligned}$$

with the type $(\texttt{nat} \to \texttt{nat}) \to (\texttt{nat} \to \texttt{nat})$. A fixpoint of $\lambda.f\, F$ can be given by $\texttt{Y}.f\, F$ as the evaluation of $(\lambda f.\, F)(\texttt{Y}f.\, F)$ and $\texttt{Y}f.\, F$

$$\begin{aligned} (\lambda f.\, F)(\texttt{Y}f.\, F) &\rightsquigarrow F[(\texttt{Y}f.\, F)/f] \\ \texttt{Y}f.\, F &\rightsquigarrow F[(\texttt{Y}f.\, F)/f] \end{aligned}$$

shows that they reduce to the same term.

Exercise. Show that $\texttt{fact}\ \underline{n} \rightsquigarrow^* \underline{n!}$ by induction on $\underline{n}$.

Example: greatest common divisor

Example 12. The Euclidean algorithm for the greatest common divisor of two natural numbers can be defined recursively as follows: where $\texttt{mod}\ x\ y$ is the reminader of x/y.

Type safety for PCF

Theorem 13 (Progress Theorem)**.** *If* $\vdash \mathsf{M} : \tau$ *then either* M *is a value or there exists* M' *such that* $\mathsf{M} \rightsquigarrow \mathsf{M}'$*.*

Theorem 14 (Preservation Theorem)**.** *If* $\vdash \mathsf{M} : \tau$ *and* $\mathsf{M} \rightsquigarrow \mathsf{N}$ *then* $\vdash \mathsf{N} : \tau$*.*

All follow the same pattern in the situtaiton for simply typed lambda calculus.[1]

3 Big-step semantics

Another reduction relation

Instead of the one-step reduction relation $\rightsquigarrow$, we turn to the **big-step** reduction relation $\Downarrow$ between terms, formulating the notion that a term M reduce to a value V eventually.

- simply typed lambda calculus

$$\frac{}{\lambda x.\, \mathsf{M} \Downarrow \lambda x.\, \mathsf{M}} \ (\Downarrow\text{-lam})$$

$$\frac{\mathsf{M} \Downarrow \lambda x.\, \mathsf{E} \qquad \mathsf{E}[\mathsf{N}/x] \Downarrow \mathsf{V}}{\mathsf{M}\ \mathsf{N} \Downarrow \mathsf{V}} \ (\Downarrow\text{-app})$$

- natural numbers

[1] To be proved in **Agda** formally.

$$\dfrac{\dfrac{}{\lambda x.\,\mathtt{ifz}(x;\underline{0};y.\,y) \Downarrow \lambda x.\,\mathtt{ifz}(x;\underline{0};y.\,y)} \qquad \dfrac{\dfrac{\vdots}{\underline{3} \Downarrow \mathtt{suc}\ \underline{2}} \qquad \dfrac{\vdots}{y[\underline{2}/y] \Downarrow \underline{2}}}{\mathtt{ifz}(x;\underline{0};y.y)[\underline{3}/x] \Downarrow \underline{2}}}{\lambda x.\,\mathtt{ifz}(x;\underline{0};y.\,y)\ \underline{3} \Downarrow \underline{2}}$$

Figure 1: Derivation of $\mathtt{pred}\ \underline{3} \Downarrow \underline{2}$

$$\dfrac{}{\mathtt{zero} \Downarrow \mathtt{zero}}\ (\Downarrow\text{-zero})$$

$$\dfrac{\mathsf{M} \Downarrow \mathsf{V}}{\mathtt{suc}\ \mathsf{M} \Downarrow \mathtt{suc}\ \mathsf{V}}\ (\Downarrow\text{-suc})$$

- if-zero test

$$\dfrac{\mathsf{M} \Downarrow \mathtt{zero} \qquad \mathsf{M}_0 \Downarrow \mathsf{V}}{\mathtt{ifz}(\mathsf{M};\mathsf{M}_0;x.\,\mathsf{M}_1) \Downarrow \mathsf{V}}\ (\Downarrow\text{-}\mathtt{ifz}_0)$$

$$\dfrac{\mathsf{M} \Downarrow \mathtt{suc}\ \mathsf{N} \qquad \mathsf{M}_1[\mathsf{N}/x] \Downarrow \mathsf{V}}{\mathtt{ifz}(\mathsf{M};\mathsf{M}_0;x.\,\mathsf{M}_1) \Downarrow \mathsf{V}}\ (\Downarrow\text{-}\mathtt{ifz}_1)$$

- general recursion

$$\dfrac{\mathsf{M}[\mathtt{Y}x.\,\mathsf{M}/x] \Downarrow \mathsf{V}}{\mathtt{Y}x.\,\mathsf{M} \Downarrow \mathsf{V}}\ (\Downarrow\text{-fix})$$

Exercise.

1. Show that $\mathtt{fact}\ \underline{0} \Downarrow \underline{1}$.
2. Show that $\mathtt{flip}\ \underline{0} \Downarrow \underline{1}$ and $\mathtt{flip}\ \underline{n+1} \Downarrow \underline{0}$.

Reduction on values

We shell justify the intended meaning. Whenever $\mathsf{M} \Downarrow \mathsf{V}$, the term V is always a value; every value is in its simplest form.

Lemma 15. *For every terms* M *and* V*, the term* V *is a value if* $\mathsf{M} \Downarrow \mathsf{V}$*.*

Proof. By induction on the derivation of $\mathsf{M} \Downarrow \mathsf{V}$. □

Lemma 16. *If* V *is a value, then* $\mathsf{V} \Downarrow \mathsf{V}$*.*

Proof. By induction on the derivation of V **val**. □

Agreement of big-step and one-step semantics

Theorem 17. *For every term* M *and* V*,* $\mathsf{M} \Downarrow \mathsf{V}$ *if and only if* $\mathsf{M} \rightsquigarrow^* \mathsf{V}$ *with* V **val***.*

Proof sketch.
1. Show that if $\mathsf{M} \Downarrow \mathsf{V}$ then $\mathsf{M} \rightsquigarrow^* \mathsf{V}$ by induction on $\Downarrow$ and $\rightsquigarrow^*$.
2. Show that if $\mathsf{M} \rightsquigarrow \mathsf{N} \Downarrow \mathsf{V}$ then $\mathsf{M} \Downarrow \mathsf{V}$.
3. Show that if $\mathsf{M} \rightsquigarrow^* \mathsf{N} \Downarrow \mathsf{V}$ then $\mathsf{M} \Downarrow \mathsf{V}$.

In particular, every $\mathsf{M} \rightsquigarrow^* \mathsf{V}$ with V **val**, has $\mathsf{V} \Downarrow \mathsf{V}$, so it follows that $\mathsf{M} \Downarrow \mathsf{V}$. □

Corollary 18 (Preservation Theorem for $\Downarrow$)**.** *If* $\vdash \mathsf{M} : \tau$ *and* $\mathsf{M} \Downarrow \mathsf{V}$ *then* $\vdash \mathsf{V} : \tau$*.*

Exercises

1. Define the following programs in **PCF**.
   (a) Addition and multiplication of natural numbers
   (b) Fibonacci numbers;
   (c) Parity test, i.e. a function determines whether the given argument is an odd or even number. Return `zero` if even, `suc zero` otherwise.
2. Let `bool` be a type with two constructors:

$$\dfrac{}{\mathtt{true} : \mathtt{bool}}$$

$$\dfrac{}{\mathtt{false} : \mathtt{bool}}$$

   (a) Provide the typing rule for the conditional construct `if`:

$$\dfrac{?}{\Gamma \vdash \mathtt{if}(\mathsf{M}_0;\mathsf{M}_1;\mathsf{M}_2) : \tau}$$

   (b) Provide its one-step semantics.

3. Define primitive recursion in **PCF**

$$\mathtt{rec} : \tau \to (\mathtt{nat} \to \tau \to \tau) \to \mathtt{nat} \to \tau$$

such that

$$\begin{aligned} \mathtt{rec}\ e_0\ f\ \mathtt{zero} &\rightsquigarrow^* e_0 \\ \mathtt{rec}\ e_0\ f\ (\mathtt{suc}\ \mathsf{M}) &\rightsquigarrow^* f\ \mathsf{M}\ (\mathtt{rec}\ e_0\ f\ \mathsf{M}) \end{aligned}$$

respectively

Reference

Denotational Semantics and this lecture are based on the following two books:

1. Thomas Streicher, *Domain-Theoretic Foundations of Functional Programming*, World Scientific, 2006
2. Robert Harper, *Practical Foundations for Programming Languages*, Cambridge University Press, 2012

Their preprints are available on the Internet.