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

1 Introduction

The meaning of programs
How can we tell if a program is correct?

1. The meaning of a program is ideally independent of its actual implementation.
2. A rigorous specification of language is essential. Everything must be defined without any ambiguities. No undefined behaviour.
3. A structural approach to semantics. The meaning of a program is built from its parts, so verification is possible.
4. The precise definitions of notions, such as strict and lazy evaluation strategies.

Two approaches to be taught

1. **Operational approach**: How values and functions are computed? E.g., for $\mathtt{add} : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}$ and numerals $\underline{2}$, $\underline{4}$

$$\begin{aligned} &\mathtt{add}\ \underline{2}\ \underline{4} \rightsquigarrow \mathtt{suc}\ (\mathtt{add}\ \underline{1}\ \underline{4}) \\ \rightsquigarrow\ &\mathtt{suc}\ \mathtt{suc}(\mathtt{add}\ \underline{0}\ \underline{4}) \rightsquigarrow \mathtt{suc}\ \mathtt{suc}\ \underline{4} \equiv \underline{6} \end{aligned}$$

2. **Denotational approach**: What the values and functions are? The set $\mathbb{N}_\perp$ of natural numbers with *divergence* $\perp$ is the denotation of the type $\mathtt{nat}$, e.g.,

$$\begin{aligned} [\![\mathtt{add}\ \underline{2}\ \underline{4}]\!] &= [\![\mathtt{add}\ \underline{2}]\!]\ [\![\underline{4}]\!] = ([\![\mathtt{add}]\!]\ [\![\underline{2}]\!])\ 4 \\ &= (x \mapsto 2 + x)\ 4 = 2 + 4 = 6 \end{aligned}$$

where $[\![\mathtt{add}]\!] \colon \mathbb{N}_\perp \to (\mathbb{N}_\perp \to \mathbb{N}_\perp)$.

2 Programming in PCF

2.1 Syntax and typing rules for PCF

What is PCF?
PCF stands for **Programming Computable Functionals**,

1. an extension of simply typed lambda calculus with $\mathtt{nat}$ and general recursion;
2. extremely simple compared to modern programming languages;
3. Turing complete, i.e. every computable function on natural numbers can be defined in **PCF**.

Syntax of PCF

Definition 1. Types in **PCF** are defined by the inference rules

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

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

or equivalently by the grammar $\tau \coloneqq \mathtt{nat} \mid \tau \to \tau$.

Definition 2. The collection of terms in **PCF** is defined inductively:

$$\begin{aligned} \mathsf{M} \coloneqq\ & x \mid \lambda x.\, \mathsf{M} \mid \mathsf{M}\ \mathsf{N} \mid \mathtt{zero} \mid \mathtt{suc}\ \mathsf{M} \\ & \mid \mathtt{ifz}(\mathsf{M}; \mathsf{M}; x.\, \mathsf{M}) \mid \mathtt{Y}x.\, \mathsf{M} \end{aligned}$$

where x is a variable.

The operator $\mathtt{Y}$ is called the **fixpoint operator**, or **general recursion**.

Typing Rules for PCF
A **judgement** $\Gamma \vdash \mathsf{M} : \tau$ denotes that M has type τ under the context Γ. **PCF** consists of

- simply typed lambda calculus (with $\to$ only):

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

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

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

- the type of natural numbers:

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

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

- **if zero** test: it is meant to be the *case analysis* on natural numbers:

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

- **general recursion** (to be explained):

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

Definition 3. A term M of type τ is called a **program** of type τ in **PCF** if it is derivable under an empty context, i.e. the judgement

$$() \vdash \mathsf{M} : \tau$$

is derivable where $()$ denotes the empty context for emphasis.

E.g. $\mathtt{Y}x.\, \mathtt{suc}\ x$ and $\mathtt{ifz}(\mathtt{zero}; \lambda x.\, \mathtt{zero}; y.\, \lambda z.\, y)$ are programs, but $\lambda y.\, x\ y$ or $\mathtt{suc}\ (\lambda x.\, \mathtt{suc}\ x)$ are not.

Example: Predecessor

The predecessor of natural numbers can be defined as

$$\mathtt{pred} \coloneqq \lambda x.\, \mathtt{ifz}(x; \mathtt{zero}; y.\, y) : \mathtt{nat} \to \mathtt{nat}$$

with the following typing derivation:

$$\frac{\dfrac{\dfrac{}{\Gamma \vdash x : \mathtt{nat}} \quad \dfrac{}{\Gamma \vdash \mathtt{zero} : \mathtt{nat}} \quad \dfrac{}{\Gamma, y : \mathtt{nat} \vdash y : \mathtt{nat}}}{\Gamma \vdash \mathtt{ifz}(x; \mathtt{zero}; y.\, y) : \mathtt{nat}}}{\vdash \lambda x.\, \mathtt{ifz}(x; \mathtt{zero}; y.\, y) : \mathtt{nat} \to \mathtt{nat}}$$

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

2.2 Operational semantics

One-step reduction

Definition 4. A **closed value** denoted **val** is one of the following:

$$\frac{}{\mathtt{zero}\ \textbf{val}}$$

$$\frac{\mathsf{M}\ \textbf{val}}{\mathtt{suc}\ \mathsf{M}\ \textbf{val}}$$

$$\frac{}{\lambda x.\, \mathsf{M}\ \textbf{val}}$$

A value is meant to be the final result of computation. For example, natural numbers $\mathtt{zero}, \mathtt{suc}\ \mathtt{zero}$ and lambda functions $\lambda x.\, x$ etc. This formulation also includes ill-typed terms such as $\mathtt{suc}\ (\lambda x.\, \mathsf{M})$.

Notation

The notation $\rightsquigarrow$ is a relation between terms, denoted

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

which means that the term M reduces to M' in *one step.*

Reduction of general recursion and natural numbers

For general recursion, each occurrence of $\mathtt{Y}.\, \mathsf{M}$ reduces to an substitution of x in M by itself:

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

For the *eager evaluation*, $\mathtt{suc}\ \mathsf{M}$ reduces to $\mathtt{suc}\ \mathsf{M}'$ if M reduces to M'

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

On the other hand, it is possible to defer the evaluation of natural numbers, and this evaluation is known as the *lazy evaluation.* To do so, we simply remove this reduction rule $\rightsquigarrow$-$\mathtt{suc}$ and modify the definition of closed values for $\mathtt{suc}$ to

$$\frac{}{\mathtt{suc}\ \mathsf{M}\ \textbf{val}}$$

without any assumptions.

Reduction of ifz

For the if-zero test, the first argument must be reduced to a closed value before branching, but branching can be done before the evaluation on branches:

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

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

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

Reduction for application: call-by-name and call-by-value

In call-by-name evaluation, arguments are substituted directly into the function body. It is a *non-strict* evaluation strategy, because application with non-terminating arguments can be terminating.

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

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

In call-by-value evaluation, each argument is evaluated before application, so we replace ($\rightsquigarrow$-by-name) by the following two rules. It is a *strict* evaluation strategy, as non-terminating arguments always lead to non-terminating terms.

$$\frac{\mathsf{M}\ \mathbf{val} \qquad \mathsf{N} \rightsquigarrow \mathsf{N}'}{\mathsf{M}\ \mathsf{N} \rightsquigarrow \mathsf{M}'\ \mathsf{N}}\ (\rightsquigarrow\text{-by-value-1})$$

$$\frac{\mathsf{N}\ \mathbf{val}}{(\lambda x.\, \mathsf{M})\ \mathsf{N} \rightsquigarrow \mathsf{M}[\mathsf{N}/x]}\ (\rightsquigarrow\text{-by-value-2})$$

In the following context, we adopt the *call-by-name* interpretation.

Many-step reduction

Definition 5. The relation $\rightsquigarrow^*$ between terms is defined inductively:

$$\frac{}{\mathsf{M} \rightsquigarrow^* \mathsf{M}}$$

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

Note that $M \rightsquigarrow^* \mathsf{M}'$ if M' is reachable from M after finitely many steps of reduction, i.e. $\mathsf{M} = \mathsf{M}_0 \rightsquigarrow \mathsf{M}_1 \rightsquigarrow \cdots \mathsf{M}_k = \mathsf{M}'$.

Proposition 6. *The relation $\rightsquigarrow^*$ is reflexive and transitive.*

Proof. Easy exercise. □

Example: Calculating the factorials

To define the factorials, we are seeking for a function `fact` satisfying

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

and this can be understood as a fixpoint of the functional F mapping $f\colon \mathbb{N} \to \mathbb{N}$ to $f'\colon \mathbb{N} \to \mathbb{N}$ defined by

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

where f' does not depend on itself.

Under the context of $\Gamma \coloneqq f\colon \mathtt{nat} \to \mathtt{nat}$, we define a function depending on f as follows:

$$\Gamma \vdash \lambda n.\, \mathtt{ifz}(n; \mathtt{suc}\ \mathtt{zero}; m.\, n \times (f\ m)) : \mathtt{nat} \to \mathtt{nat}$$

and thus we derive its fixpoint by $\mathtt{Y}$:

$$\mathtt{fact} \coloneqq \mathtt{Y}f.\, \lambda n.\, \mathtt{ifz}(n; \mathtt{suc}\ \mathtt{zero}; m.\, n \times (f\ m))$$

where the term `suc zero` represents the natural number 1.

Example 7. Let $\underline{0} \coloneqq \mathtt{zero}$ and $\underline{n+1} \coloneqq \mathtt{suc}\ \underline{n}$. We calculate $\mathtt{fact}\ \underline{2}$:

$$\begin{aligned} \mathtt{fact}\ \underline{2} &\rightsquigarrow (\lambda n.\, \mathtt{ifz}(n; \underline{1}; m.\, n \times (\mathtt{fact}\ m)))\ \underline{2} \\ &\rightsquigarrow \mathtt{ifz}(\underline{2}; \underline{1}; m.\, \underline{2} \times (\mathtt{fact}\ m)) \\ &\rightsquigarrow \underline{2} \times (\mathtt{fact}\ \underline{1}) \\ &\rightsquigarrow \underline{2} \times (\lambda n.\, \mathtt{ifz}(n; \underline{1}; m.\, n \times (\mathtt{fact}\ m))\ \underline{1}) \\ &\rightsquigarrow \cdots \rightsquigarrow \underline{2} \times (\underline{1} \times \underline{1}) \rightsquigarrow^* \underline{2} \end{aligned}$$

where the definition of $\times : \mathtt{nat} \to \mathtt{nat} \to nat$ is left as an exercise.

In-class exercise

Try to be familiar with `ifz`.

1. Calculate $\mathtt{pred}\ \mathsf{M}$ for $\mathsf{M}\ \mathbf{val}$ to closed values with their derivations: For the base case `zero`:

   $$\mathtt{pred}\ \mathtt{zero} \rightsquigarrow^* ?$$

   For the inductive case $\mathsf{M} = \mathtt{suc}\ \mathsf{N}$:

   $$\frac{\mathtt{suc}\ \mathsf{N}\ \mathbf{val}}{\mathtt{pred}\ (\mathtt{suc}\ \mathsf{N}) \rightsquigarrow^* ?}$$

2. Define $\mathtt{flip}\colon \mathtt{nat} \to \mathtt{nat}$ such that $\mathtt{flip}\ \mathtt{zero} \rightsquigarrow^* \mathtt{suc}\ \mathtt{zero}$ and $\mathtt{flip}\ (\mathtt{suc}\ \mathsf{M}) \rightsquigarrow^* \mathtt{zero}$.

In-class exercise: `fold` on natural numbers

`fold` on natural numbers is defined in Haskell as follows:

```
fold :: (a -> a) -> a -> Integer -> a
fold f e 0 = e
fold f e n = f (fold f e (n - 1))
```

By modifying the definition of `fact`, give the corresponding term of `fold` in **PCF**.

3 Type safety

Progress Theorem

Every well-typed is either a closed value or a reducible term.

Theorem 8. *If* $\vdash \mathsf{M} : \tau$ *then either* M *is a closed value or there exists* M' *such that* $\mathsf{M} \rightsquigarrow \mathsf{M}'$.

Proof. By induction on the derivation of $\vdash \mathsf{M} : \tau$. For the case that

$$\frac{\mathsf{M} : \mathtt{nat}}{\vdash \mathtt{suc}\ \mathsf{M} : \mathtt{nat}}$$

M is either a closed value or a reducible term by induction hypothesis:

1. If M is a closed value, then $\mathtt{suc}\ \mathsf{M}$ is also a closed value by definition.

2. Suppose that $\mathsf{M} \rightsquigarrow \mathsf{M}'$ for some M'. Then, by the rule ($\rightsquigarrow$-$\mathtt{suc}$), we also have $\mathtt{suc}\ \mathsf{M} \rightsquigarrow \mathtt{suc}\ \mathsf{M}'$.

□

Proofs are programs

Remark 3.1. Note that given a program $\mathsf{M} : \tau$, the previous proof produces either a closed value or a **PCF** term M' with a proof that M reduces to M'. Forgetting the proof, the proof itself is indeed a program which asks a **PCF** term, preforms a single reduction and return a term tagged either `done` or `not yet`.

Substitution Lemma

If a variable $x : \tau$ in a term M is substituted by another term N of the same type, then the type of the resulting term remains.

Lemma 9. *If* $\Gamma, x : \sigma \vdash \mathsf{M} : \tau$ *and* $\Gamma \vdash \mathsf{N} : \sigma$*, then* $\Gamma \vdash \mathsf{M}[\mathsf{N}/x] : \tau$.

Proof. Induction on the derivation of $\Gamma, x : \sigma \vdash \mathsf{M} : \tau$. Suppose that $\Gamma, x : \sigma \vdash \mathsf{M} : \tau$ is derived from

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

that is, $\Gamma = \Delta, y : \tau$ and $\mathsf{M} = y$ for some variable y. Then, we need to show that $\Delta, y : \tau \vdash y[\mathsf{N}/x] : \tau$.

1. If $x = y$, then $y[\mathsf{N}/x] = \mathsf{N} : \sigma$ and $\sigma = \tau$.

2. Otherwise, $x \neq y$, then $y[\mathsf{N}/x] = y : \tau$.

Other cases follow similarly. □

Preservation Theorem

The one-step evaluation preserves types. This property is also called **Subject Reduction**.

Theorem 10. *If* $\vdash \mathsf{M} : \tau$ *and* $\mathsf{M} \rightsquigarrow \mathsf{N}$ *then* $\vdash \mathsf{N} : \tau$.

Proof. We prove it by induction on the derivation of $\vdash \mathsf{M} : \tau$ and $\mathsf{M} \rightsquigarrow \mathsf{M}'$. For the case that

$$\frac{x : \sigma \vdash \mathsf{M} : \sigma}{\vdash \mathtt{Y}x.\,\mathsf{M} : \sigma}$$

we do induction on $\rightsquigarrow$, but there is exactly one rule applicable:

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

By Substitution Lemma, it follows that $\vdash M[\mathtt{Y}x.\,M/x] : \sigma$, and other cases follow similarly. □

4 Big-step semantics

Call-by-name big-step semantics

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

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

$$\frac{\mathsf{M} \Downarrow \underline{n}}{\mathtt{suc}\ \mathsf{M} \Downarrow \mathtt{suc}\ \underline{n}}\ (\Downarrow\text{-suc})$$

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

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

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

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

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

Closed values

We shell justify the intended meaning: whenever $\mathsf{M} \Downarrow \mathsf{V}$, the term V is always a closed value:

Lemma 11. *For every terms* M *and* V*, the term* V *is a closed value if* $\mathsf{M} \Downarrow \mathsf{V}$*.*

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

Moreover, a closed value reduces to itself:

Lemma 12. *If* V *is a closed value, then* $\mathsf{V} \Downarrow \mathsf{V}$*.*

Proof. By structural induction on V **val**. That is, it is sufficient to check that $\mathtt{zero} \Downarrow \mathtt{zero}$, $\lambda x.\,\mathsf{M} \Downarrow \lambda x.\,\mathsf{M}$; $\mathtt{suc}\ \mathsf{M} \Downarrow \mathtt{suc}\ \mathsf{M}$ if $\mathsf{M} \Downarrow \mathsf{M}$ by induction hypothesis. □

Agreement of big-step and one-step semantics

Indeed, the big-step reduction can be characterised with respect to the one-step step reduction as follows:

Theorem 13. *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. First we show that if $\mathsf{M} \Downarrow \mathsf{V}$ then $\mathsf{M} \rightsquigarrow^* \mathsf{V}$ by induction on $\Downarrow$ and $\rightsquigarrow^*$.
2. Second, by induction on $\rightsquigarrow$ and $\Downarrow$ we show that

$$\frac{\mathsf{M} \rightsquigarrow \mathsf{N} \Downarrow \mathsf{V}}{\mathsf{M} \Downarrow \mathsf{V}}$$

3. Finally, by induction on $\rightsquigarrow^*$ we show that

$$\frac{\mathsf{M} \rightsquigarrow^* \mathsf{N} \Downarrow \mathsf{V}}{\mathsf{M} \Downarrow \mathsf{V}}$$

In particular, for every $\mathsf{M} \rightsquigarrow^* \mathsf{V}$ with V **val**, we always have $\mathsf{V} \Downarrow \mathsf{V}$, so it follows $\mathsf{M} \Downarrow \mathsf{V}$.

Proof. 1. We show the case ($\Downarrow$-fix), which is similar to other cases:

$$\frac{\overline{\mathtt{Y}x.\,\mathsf{M} \rightsquigarrow \mathsf{M}[\mathtt{Y}x.\,\mathsf{M}/x]} \qquad \overline{M[\mathtt{Y}x.\,\mathsf{M}/x] \rightsquigarrow^* \mathsf{V}}}{\mathtt{Y}x.\,\mathsf{M} \rightsquigarrow *\mathsf{V}}$$

and by assumption V has no further reduction.

2. We show the case ($\rightsquigarrow$-fix), which is similar to other cases. By hypothesis, we have $\mathtt{Y}x.\,\mathsf{M} \rightsquigarrow \mathsf{M}[\mathtt{Y}x.\,\mathsf{M}/x]$. If $\mathsf{M}[\mathtt{Y}x.\,\mathsf{M}/x] \Downarrow \mathsf{V}$, then by ($\Downarrow$-fix) it follows that $\mathtt{Y}x.\,\mathsf{M} \Downarrow \mathsf{V}$.

3. Induction on $\rightsquigarrow^*$.

□

By the agreement of big-step and one-step semantics, we easily conclude that the Subject Reduction also holds for big-step semantics:

Corollary 14. *If* $\vdash \mathsf{M} : \tau$ *and* $\mathsf{M} \Downarrow \mathsf{V}$ *then* $\vdash \mathsf{V} : \tau$*.*

Exercises

Basic

1. Define the following programs in **PCF**.
   (a) Multiplication of natural numbers *Hint.* Define addition first;
   (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:

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

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

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

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

   (b) Provide its one-step semantics.

Advanced

3. Define primitive recursion in **PCF**

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

   such that it reduces to

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

   respectively

4. Show that $\mathtt{rec}\ e_0\ f\ \underline{n} \Downarrow \mathsf{V}$ for a closed value V if f terminating and $\underline{n}$ a closed value.

Consider Gödel's **T**, simply typed lambda calculus with natural numbers and *primitive recursion*:

$$\frac{\Gamma \vdash e_0 : \tau \quad \Gamma \vdash \mathsf{M} : \mathtt{nat} \quad \Gamma, x : \mathtt{nat}, y : \tau \vdash e_1 : \tau}{\Gamma \vdash \mathtt{rec}(e_0; x.\,y.\,e_1; \mathsf{M}) : \tau}$$

5. Provide the one-step and big-step reductions for `rec`.

6. Show that for every program M in $\mathbf{T}$, there is a closed value V with $\mathsf{M} \Downarrow \mathsf{V}$. This property is called *totality*. *Hint.* Use the structural induction principle on the typing rules for $\mathbf{T}$.

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.