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 $\underline{2}$, $\underline{4}$

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

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.,

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

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

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 `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

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

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

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

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

$$\mathsf{M} ::= x \mid \lambda x.\, \mathsf{M} \mid \mathsf{M}\ \mathsf{N} \mid \mathtt{zero} \mid \mathtt{suc}\ \mathsf{M}$$
$$\mid \mathtt{ifz}(\mathsf{M}; \mathsf{M}; x.\, \mathsf{M}) \mid \mathtt{Y}x.\, \mathsf{M}$$

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):

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

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

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

- the type of natural numbers:

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

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

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

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

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

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

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:

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

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:

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

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

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

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

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

without any assumptions.

Reduction of $\mathtt{ifz}$

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:

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

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

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

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.

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

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

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.

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

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

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

Many-step reduction

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

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

$$\frac{M_1 \rightsquigarrow M_2 \qquad M_2 \rightsquigarrow^* M_3}{M_1 \rightsquigarrow^* M_3}$$

Note that $M \rightsquigarrow^* M'$ if M' is reachable from M after finitely many steps of reduction, i.e. $M = M_0 \rightsquigarrow M_1 \rightsquigarrow \cdots M_k = M'$.

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

Proof. Easy exercise. $\qquad\qquad\square$

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:

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

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

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

where the term $\mathtt{suc\ zero}$ 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 $\times\colon \mathtt{nat} \to \mathtt{nat} \to nat$ is left as an exercise.

In-class exercise

Try to be familiar with `ifz`.

1. Calculate `pred M` for `M` **val** to closed values with their derivations: For the base case `zero`:

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

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

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

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

$\square$

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

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

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. $\square$

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. $\square$

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.

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

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

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

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

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

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

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

Closed values

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

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

Proof. By induction on the formulation of $M \Downarrow V$. $\square$

Moreover, a closed value reduces to itself:

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

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

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, $M \Downarrow V$ if and only if $M \rightsquigarrow^* V$ with V **val**.*

Proof Sketch.

1. First we show that if $M \Downarrow V$ then $M \rightsquigarrow^* V$ by induction on $\Downarrow$ and $\rightsquigarrow^*$.

2. Second, by induction on $\rightsquigarrow$ and $\Downarrow$ we show that

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

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

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

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

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

$$\frac{\overline{Yx. M \rightsquigarrow M[Yx. M/x]} \quad \overline{M[Yx. M/x] \rightsquigarrow^* V}}{Yx. M \rightsquigarrow^* V}$$

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 $Yx. M \rightsquigarrow M[Yx. M/x]$. If $M[Yx. M/x] \Downarrow V$, then by ($\Downarrow$-fix) it follows that $Yx. M \Downarrow V$.

3. Induction on $\rightsquigarrow^*$.

$\square$

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 $\vdash M : \tau$ and $M \Downarrow V$ then $\vdash V : \tau$.*

Exercises

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 $\texttt{zero}$ if even, $\texttt{suc zero}$ otherwise.

2. Let $\texttt{bool}$ be a type with two constructors:

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

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

   (a) Provide the typing rule for the conditional construct $\texttt{if}$:

$$\frac{?}{\Gamma \vdash \texttt{if}(M_0; M_1; M_2) : \tau}$$

   (b) Provide its one-step semantics.

3. Define primitive recursion in **PCF**

$$\texttt{rec} : \tau \rightarrow (\texttt{nat} \rightarrow \tau \rightarrow \tau) \rightarrow \texttt{nat} \rightarrow \tau$$

   such that

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

   respectively

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

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

4. Provide the one-step and big-step reductions for $\texttt{rec}$.

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.