Semantics of Functional Programming

PCF and its Operational Semantics

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

Formosan Summer School on Logic, Language, and Computation 2014

Why semantics?

> The hitch is that defining a language *a posteriori*, i.e. after its design has been frozen by the existence of implementations and uses, can hardly improve it. To create a good programming language, semantics must be used *a priori*, as a design tool that embodies and extends the intuitive notion of uniformity. — John C. Reynolds

C++

- Implementation-led design.
- ***C++** The International Standard, 1338 pp.*, 2012. Note that the committee consists of *200+ people.*
- *1900+* language issues!

Standard ML

- Semantics-led design.
- R. Milner, M. Tofte, R. Harper, and D. MacQueen *The Definition of **Standard ML** (Revised), 128 pp.*, 1997.
- Standard ML is a safe, modular, strict, functional, polymorphic programming language ... and *a formal definition with a proof of soundness.*

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 typed lambda calculus where terms and typing rules are introduced separately.

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

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

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

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

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

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

The existence of ill-typed terms

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

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

$$\frac{x \ \textbf{var} \qquad \dfrac{\dfrac{x \ \textbf{var}}{x \ \textbf{term}} \qquad \dfrac{x \ \textbf{var}}{x \ \textbf{term}}}{x \ x \ \textbf{term}}}{\lambda x.\, x \ x \ \textbf{term}}$$

However, $(\lambda x.\, x \ x)$ 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. We introduce two rules for applications:

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

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

These two rules formalise what we call *call-by-name* evaluation strategy, where its arguments are evaluated only if used at least once. This allows us to feed a non-terminating argument and produce a terminating result.

In most of programming languages such as **C**, arguments are evaluated to values before applications, and this evaluation strategy is called *call-by-value*.

Example 2. $(\lambda x.\, \lambda y.\, x)\ \mathsf{M}\ \mathsf{N}$ can reduce 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

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

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

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

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 the cases, so we can analyse its structure case by case.

Proposition 5 (Transitivity of $\rightsquigarrow^*$)**.** *For every three terms* M_1*,* M_2*, and* M_3*, 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 $\dfrac{}{\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_1}$, 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 $\dfrac{\mathsf{M}_1 \rightsquigarrow \mathsf{M} \qquad \mathsf{M} \rightsquigarrow^* \mathsf{M}_2}{\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_2}$, we infer that $\mathsf{M} \rightsquigarrow^* \mathsf{M}_3$ by induction hypothesis, so we derive the goal
$$\frac{\mathsf{M}_1 \rightsquigarrow \mathsf{M} \qquad \mathsf{M} \rightsquigarrow^* \mathsf{M}_3}{\mathsf{M}_1 \rightsquigarrow^* \mathsf{M}_3}$$

□

Similarly, we can do induction on the formation 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 $\mathsf{M} \rightsquigarrow^* \mathsf{M}'$.

Type safety: well-typed programs don't go wrong

Well-typed closed terms have some nice properties. First, every well-typed closed term can reduce 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 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. The context of $\dfrac{}{\Gamma, x : \sigma, \Delta \vdash x : \sigma}$, is non-empty, so it does not satisfy the assumption.
2. For that case $\dfrac{x : \sigma \vdash \mathsf{M} : \tau}{\vdash \lambda x.\, \mathsf{M} : \sigma \to \tau}\ (\text{abs})$, we have already given a term in this form $\lambda x.\, \mathsf{M}$.
3. For $\dfrac{\vdash \mathsf{M} : \sigma \to \tau \qquad \vdash \mathsf{N} : \sigma}{\vdash \mathsf{M}\ \mathsf{N} : \tau}\ (\text{app})$, 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):

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

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

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

$\square$

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

$\square$

Reductions on ill-typed terms

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

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

On the other hand, the reduction of ill-typed terms may not reduce to a value at all

$$x\ x \not\rightsquigarrow$$

Summary

To define a language, we specify following sets of rules

Syntax type, term, and typing rules.

Semantics reduction rules.

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

Progress Theorem for every well-typed closed term, it either can reduce 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 latter two features step by step.

It has two rules of type formation:

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

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

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

Term formation, typing, and reduction for natural numbers

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

$$\frac{}{\texttt{zero}\ \textbf{term}}$$

$$\frac{\mathsf{M}\ \textbf{term}}{\texttt{suc}\ \mathsf{M}\ \textbf{term}}$$

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

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

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

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

Values: canonical elements

Values are basic forms of terms 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:

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

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

Example 10. By this formation, 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.

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

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

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

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

with the following typing derivation:

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

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

Exercise.

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

Term formation, typing rule, and reduction for general recursion

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

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

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

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

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

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

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

Example: calculating the factorials

The factorial of n is usually defined recursively

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

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 $F(\mathtt{fact}) = \mathtt{fact}$.

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

$$F \coloneqq \lambda f.\, F'$$

with

$$F' \coloneqq \lambda n.\, \mathtt{ifz}(n; \underline{1}; m.\, n \times (f\ m)).$$

$\mathtt{Y}f.\, F$" is a a fixpoint of F

$$\begin{aligned}(\lambda f.\, F')\ (\mathtt{Y}f.\, F') &\rightsquigarrow F'[(\mathtt{Y}f.\, F')/f] \\ &= \lambda n.\, \mathtt{ifz}(n; \underline{1}; m.\, n \times (\mathtt{Y}f.\, F')\ m)\end{aligned}$$

We will explain this in more details in the lectures on denotational semantics. **Exercise**. Show that $\mathtt{fact}\ \underline{n} \rightsquigarrow^* \underline{n!}$ by induction on $\underline{n}$.

Type safety for PCF

Theorem 12 (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 13 (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

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

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

- natural numbers

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

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

- if-zero test

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

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

- general recursion

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

Exercise.

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

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

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

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

Reduction on values
We shall 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 14. *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 15. *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 16. *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 $\mathsf{M} \rightsquigarrow \mathsf{N}$ and $\mathsf{N} \Downarrow \mathsf{V}$ then $\mathsf{M} \Downarrow \mathsf{V}$.
3. Show that if $\mathsf{M} \rightsquigarrow^* \mathsf{N}$ and $\mathsf{N} \Downarrow \mathsf{V}$ 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 17 (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:

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

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

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

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

(b) Provide its one-step semantics such that $\texttt{if}(\mathsf{M}_0, \mathsf{M}_1, \mathsf{M}_2)$ reduces to M_1 if M_0 is `true`; or M_2 otherwise.

(c) Show that Progress Theorem and Preservation Theorem hold for **PCF** with `bool`.

3. Define primitive recursion in **PCF**

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

such that

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

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.

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

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

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

$$\frac{}{\texttt{zero}\ \textbf{term}}$$

$$\frac{\mathsf{M}\ \textbf{term}}{\texttt{suc}\ \mathsf{M}\ \textbf{term}}$$

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

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

Figure 2: Term formation rules for **PCF**

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

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

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

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

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

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

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

Figure 3: Typing rules for **PCF**

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

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

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

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

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

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

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

Figure 4: Reduction rules for **PCF**