Semantics of Functional Programming

PCF and its Operational Semantics

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

Institute of Information Science, Academia Sinica

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!

Revision 89, 2014-05-27: Issues 1351 1356 1465, 1590, 1639 1708, and 1810 were returned to "review" status for further discussion. Restored issue 1397 to "ready"; it had incorrectly been moved back to "drafting" because of a misunderstood comment. Added new issues 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, and 1931.

Standard ML

- Semantics-led design.
- R. Milner, M. Tofte, and R. Harper, *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.

The approach *à la* Curry

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

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

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

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

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

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

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

The 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

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

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:

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

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

Example 2

$(\lambda x.\, \lambda y.\, x)$ M N can be reduced to M by the following derivation

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

Many-step reduction

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

Definition 3

The many-step reduction relation $\leadsto^*$ is defined inductively by

$$\frac{}{M \leadsto^* M} \qquad \frac{M_1 \leadsto M_2 \quad M_2 \leadsto^* M_3}{M_1 \leadsto^* M_3}$$

Proposition 4 (Reflexivity of $\leadsto^*$)

For every term M, $M \leadsto^* M$.

For example, one has

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

by the derivation

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

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 $M \rightsquigarrow^* N$ 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* $M_1 \rightsquigarrow^* M_2$ *and* $M_2 \rightsquigarrow^* M_3$, *then* $M_1 \rightsquigarrow^* M_3$.

Given derivations of $M_1 \rightsquigarrow^* M_2$ and $M_2 \rightsquigarrow^* M_3$, we do case analysis on the derivation of $M_1 \rightsquigarrow^* M_2$. Also, we can assume that the premise satisfy this property, that is, the induction hypothesis.

Proof.

1. For $\overline{M_1 \rightsquigarrow^* M_1}$, it unifies M_2 to M_1, so the given derivation $M_2 \rightsquigarrow^* M_3$ is just the goal derivation as $M_1 = M_2$.
2. For $\dfrac{M_1 \rightsquigarrow M \qquad M \rightsquigarrow^* M_2}{M_1 \rightsquigarrow^* M_2}$, we infer that $M \rightsquigarrow^* M_3$ by induction hypothesis, so we derive the goal

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

□

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

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

Reductions on ill-typed terms

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

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

On the other hand, the reduction of ill-typed terms may not 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: well-typed programs don't go wrong

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 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 $\dfrac{}{\Gamma, x : \sigma, \Delta \vdash x : \sigma}$, since the context is empty.
2. For that case $\dfrac{x : \sigma \vdash \mathsf{M} : \tau}{\vdash \lambda x.\, \mathsf{M} : \sigma \to \tau}$ (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 $\dfrac{\vdash \mathsf{M} : \sigma \to \tau \qquad \vdash \mathsf{N} : \sigma}{\vdash \mathsf{M}\ \mathsf{N} : \tau}$ (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):

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

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

□

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 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.

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 formulation:

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

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

Term formulation, typing, and reduction for natural numbers

Every natural number is either $\texttt{zero}$ or a successor of some natural number.

$$\frac{}{\texttt{zero}\ \textbf{term}} \qquad\qquad \frac{}{\Gamma \vdash \texttt{zero} : \texttt{nat}}\ (\mathsf{z})$$

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

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:

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

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

Example 10

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

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

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

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

accompanying with three reductions rules

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

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

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

Example: predecessor

The predecessor of natural numbers can be defined as

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

with the following typing derivation:

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

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

Exercise.

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

Term formulation, typing rule, and reduction for general recursion

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

$$\frac{x \ \textbf{var} \qquad \mathsf{M} \ \textbf{term}}{\mathtt{Y}x.\,\mathsf{M} \ \textbf{term}} \qquad\qquad \frac{\Gamma, x : \sigma \vdash \mathsf{M} : \sigma}{\Gamma \vdash \mathtt{Y}x.\,\mathsf{M} : \sigma}\,(\mathsf{Y})$$

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

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

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

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

with the type $(\mathtt{nat} \to \mathtt{nat}) \to (\mathtt{nat} \to \mathtt{nat})$.

A fixpoint of $\lambda.f\ F$ can be given by $\mathtt{Y}.f\ F$ as the evaluation of $(\lambda f.\, F)(\mathtt{Y}f.\, F)$ and $\mathtt{Y}f.\, F$

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

shows that they reduce to the same term.

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]

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

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

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

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

- if-zero test

$$\frac{M \Downarrow \texttt{zero} \qquad M_0 \Downarrow V}{\texttt{ifz}(M; M_0; x.\, M_1) \Downarrow V} \; (\Downarrow\text{-}\texttt{ifz}_0)$$

$$\frac{M \Downarrow \texttt{suc}\; N \qquad M_1[N/x] \Downarrow V}{\texttt{ifz}(M; M_0; x.\, M_1) \Downarrow V} \; (\Downarrow\text{-}\texttt{ifz}_1)$$

- general recursion

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

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

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

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 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* $\mathsf{M} \leadsto^{*} \mathsf{V}$ *with* V **val**.

Proof sketch.

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

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

Corollary 17 (Preservation Theorem for $\Downarrow$)

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