Types for Pratical Programming

(Started on the 6th of October, 2006)

(Working Draft of October 25, 2011)

Hongwei Xi

Boston University

Chapter 1

Introduction

The paradigm of *programming with theorem programming* is advocated in the programming language ATS. In this book, we present a theoretical foundation for this programming paradigm.

Chapter 2

Lambda Calculus

We study pure untyped lambda-calculus in this chapter as a theory of substitution. We assume the existence of a denumerable set **VAR** of (object) variables $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$, and use x, y, z to range over these variables. Given two variables x_1 and x_2, we write $x_1 = x_2$ if both x_1 and x_2 denote the same $\mathbf{x}_n$ for some natural number n; similarly, we write $x_1 < x_2$ ($x_1 \leq x_2$) if x_1 and x_2 denote $\mathbf{x}_{n_1}$ and $\mathbf{x}_{n_2}$, respectively, for some natural numbers n_1 and n_2 satisfying $n_1 < n_2$ ($n_1 \leq n_2$); we write $x_1 > x_2$ ($x_1 \geq x_2$) to mean $x_2 < x_1$ ($x_2 \leq x_1$).

Definition 2.0.1 (λ-terms) *The (pure) λ-terms are formally defined below:*

$$\textit{terms} \quad t \quad ::= \quad x \mid \lambda x.t \mid t_1(t_2)$$

We use **TERM** for the set of all λ-terms. Given a λ-term t, t is either a variable, or a λ-abstraction of the form $\lambda x.t_1$, or an application of the form $t_1(t_2)$. We write $t_1 \equiv t_2$ to mean that t_1 and t_2 are syntactically the same.

When giving examples, we often use I for $\lambda x.x$, K for $\lambda x.\lambda y.x$, K' for $\lambda x.\lambda y.y$, and S for $\lambda x.\lambda y.\lambda z.x(z)(y(z))$. A few other special λ-terms are to be introduced later.

Definition 2.0.2 (Size of λ-terms) *We define a unary function $size(\cdot)$ to compute the size of a given λ-term:*

$$\begin{array}{rcl} size(x) & = & 0 \\ size(\lambda x.t) & = & 1 + size(t) \\ size(t_1(t_2)) & = & 1 + size(t_1) + size(t_2) \end{array}$$

Clearly, we have $size(I) = 1$, $size(K) = 2$ and $size(S) = 6$.

There is often a need to refer to a subterm in a given λ-term. For this purpose, we introduce *paths* defined as finite sequences of natural numbers:

$$\text{paths} \quad p \quad ::= \quad \emptyset \mid n.p$$

We use $\emptyset$ for the empty sequence and $n.p$ for the sequence whose head and tail are n and p, respectively, where n ranges over natural numbers. Given two paths p_1 and p_2, we write $p_1 @ p_2$ for the concatenation of p_1 and p_2. We say that p_1 is a prefix of p_2 if $p_2 = p_1 @ p_3$ for some path p_3; this prefix is proper if p_3 is not empty. We say that p_1 and p_2 are incompatible if neither of them is the prefix of the other.

We use **PATH** for the set of all paths and $\overline{p}$ to range over finite sets of paths. Given n and $\overline{p}$, we use $n.\overline{p}$ for the set $\{n.p \mid p \in \overline{p}\}$. Given p_0 and $\overline{p}$, the sets $p_0@\overline{p}$ and $\overline{p}@p_0$ are $\{p_0@p \mid p \in \overline{p}\}$ and $\{p@p_0 \mid p \in \overline{p}\}$, respectively.

Definition 2.0.3 *We define as follows a partial binary function* $subterm(\cdot,\cdot)$ *from* (**TERM**, **PATH**) *to* **TERM***:*

$$\begin{array}{rcl} subterm(t,\emptyset) & = & t \\ subterm(t_1(t_2),0.p) & = & subterm(t_1,p) \\ subterm(t_1(t_2),1.p) & = & subterm(t_2,p) \\ subterm(\lambda x.t,0.p) & = & subterm(t,p) \end{array}$$

Given two λ-terms t_1, t_2 and a path p, we say that t_1 is a *subterm of* t_2 *at* p if $subterm(t_2,p) = t_1$; this substerm is proper if p is not empty. We may simply say that t_1 is a subterm of t_2 if $subterm(t_2,p) = t_1$ for some path p. Also, we may say that t_1 has an occurrence in t_2 (at p) if t_1 is a subterm of t_2 (at p). Note that for a λ-term of the form $\lambda x.t$, the variable x following the binder λ does not count as an occurrence (in the formal sense).

Given a λ-term, we use $paths(t)$ for the set of paths such that $p \in paths(t)$ if and only if $subterm(t,p)$ is defined. Clearly, for every λ-term t, we have

- $\emptyset \in paths(t)$, and
- $p_0 \in paths(t)$ implies that $p \in paths(t)$ holds for every prefix p of p_0.

Note that for every λ-term t, $p \in paths(t)$ implies p being a sequence of 0's and 1's.

Definition 2.0.4 (Variable Set) *We define a function vars as follows that maps* λ*-terms to finite sets of variables:*

$$\begin{array}{rcl} \mathit{vars}(x) & = & \{x\} \\ \mathit{vars}(\lambda x.t) & = & \mathit{vars}(t) \cup \{x\} \\ \mathit{vars}(t_1(t_2)) & = & \mathit{vars}(t_1) \cup \mathit{vars}(t_2) \end{array}$$

Clearly, for every λ-term t_0, $x \in \mathit{vars}(t_0)$ if and only if t_0 has a subterm of the form x or $\lambda x.t$.

Definition 2.0.5 (Free Variable Set) *We define a function FV as follows that maps* λ*-terms to finite sets of variables:*

$$\begin{array}{rcl} \mathit{FV}(x) & = & \{x\} \\ \mathit{FV}(\lambda x.t) & = & \mathit{FV}(t) \backslash \{x\} \\ \mathit{FV}(t_1(t_2)) & = & \mathit{FV}(t_1) \cup \mathit{FV}(t_2) \end{array}$$

Given a λ*-term* t*, we refer to* $\mathit{FV}(t)$ *as the set of free variables in* t*. We say that a variable* x *is free in* t *if and only if* $x \in \mathit{FV}(t)$ *holds.*

Given a λ-term t_0 and a variable x, an occurrence of x in t_0 at p_0 is a free occurrence if $subterm(t_0,p)$ is not of the form $\lambda x.t$ for any prefix p of p_0. It is clear from the definition of *FV* that $x \in \mathit{FV}(t)$ if and only if x has at least one free occurrence in t.

2.1 α-Equivalence

Definition 2.1.1 (Variable Replacement) *Given a λ-term t and two variables x and y, we define $t[y/x]$ as follows by structural induction on t:*

$$\begin{array}{rcl} x[y/x] & ::= & y \\ x'[y/x] & ::= & x' \ \textit{if } x' \textit{ is not } x \\ t_1(t_2)[y/x] & ::= & t_1[y/x](t_2[y/x]) \\ (\lambda x.t)[y/x] & ::= & \lambda x.t \\ (\lambda x'.t)[y/x] & ::= & \lambda x'.t[y/x] \ \textit{if } x \neq x' \end{array}$$

We refer to $t[y/x]$ as the λ-term obtained from replacing (free occurrences) of x with y in t.

Clearly, $size(t[y/x]) = size(t)$ for all λ-terms t and variables x and y.

Proposition 2.1.2 *We have the following.*

1. $t[x/x] \equiv t$.
2. $t[y/x] \equiv t$ *if* $x \notin FV(t)$.
3. $t[y/x][z/y] \equiv t[z/x]$ *if* $y \notin vars(t)$.

Proof Both (1) and (2) are straightforward. We prove (3) by structural induction on t.

- t is x. Then both $t[y/x][z/y] \equiv z$ and $t[z/x] \equiv z$ hold, and we are done.
- t is x' for some variable $x' \neq x$. Then $x' \neq y$ also holds as $y \notin vars(t)$. So $t[y/x][z/y] \equiv x'$ and $t[z/x] \equiv x'$, and we are done.
- t is $t_1(t_2)$. For $i = 1, 2$, we have $t_i[y/x][z/y] \equiv t_i[z/x]$ by induction hypotheses on t_i. Therefore, $t[y/x][z/y] \equiv t[z/x]$ holds as well.
- t is $\lambda x.t_0$. Then $t[y/x][z/y] \equiv t[z/y]$, and $t[z/x] \equiv t$. By (2), $t[z/y] \equiv t$ holds, and we are done.
- t is $\lambda x'.t_0$ for some $x' \neq x$. We have $t_0[y/x][z/y] \equiv t_0[z/x]$ by induction hypthesis on t_0. Note that $x' \neq y$ since $y \notin vars(t)$. So we have $t[y/x][z/y] \equiv t[z/x]$.

We conclude the proof as all the cases are covered. ∎

Definition 2.1.3 *We use Γ for a sequence of variables defined as follows:*

$$\Gamma \ ::= \ \emptyset \mid \Gamma, x$$

We write $x \in \Gamma$ to indicate that x occurs in Γ, and $|\Gamma|$ for the length of Γ, that is, the number of variables in Γ. If $x \in \Gamma$ holds, we define $\Gamma(x)$ as follows: $\Gamma(x) = |\Gamma|$ if $\Gamma = \Gamma_1, x$ and $\Gamma(x) = \Gamma_1(x)$ if $\Gamma = \Gamma_1, x_1$ for some $x_1 \neq x$.

Definition 2.1.4 (α-normal forms) *We use $\underline{t}$ for α-normal forms defined as follows:*

$$\alpha\text{-normal forms} \quad \underline{t} \quad ::= \quad x \mid n \mid \lambda(\underline{t}) \mid \underline{t}_1(\underline{t}_2)$$

where n ranges over positive integers.

Given an α-normal form $\underline{t}$, $\mathit{shift}(\underline{t})$ is the α-normal form obtained from increasing each n in $\underline{t}$ by 1. Formally, we have

$$\mathit{shift}(x) = x; \mathit{shift}(n) = n+1; \mathit{shift}(\underline{t}_1(\underline{t}_2)) = \mathit{shift}(\underline{t}_1)(\mathit{shift}(\underline{t}_2)); \mathit{shift}(\lambda(\underline{t}_1)) = \lambda(\mathit{shift}(\underline{t}_1))$$

Definition 2.1.5 (α-equivalence) *Given a sequence Γ of variables and a term t, $\mathit{NF}_\alpha(\Gamma; t)$ is defined inductively as follows:*

$$\mathit{NF}_\alpha(\Gamma; t) = \begin{cases} x & \text{if } t = x \text{ for some } x \notin \Gamma; \\ \Gamma(x) & \text{if } t = x \text{ for some } x \in \Gamma; \\ \lambda(\underline{t}_0) & \text{if } t = \lambda x.t_0 \text{ and } \underline{t}_0 = \mathit{NF}_\alpha(\Gamma, x; t_0); \\ \underline{t}_1(\underline{t}_2) & \text{if } t = t_1(t_2) \text{ and } \underline{t}_1 = \mathit{NF}_\alpha(\Gamma; t_1) \text{ and } \underline{t}_2 = \mathit{NF}_\alpha(\Gamma; t_2). \end{cases}$$

We use $\mathit{NF}_\alpha(t)$ as a shorthand for $\mathit{NF}_\alpha(\emptyset; t)$. Given two terms t_1 and t_2, we say that t_1 and t_2 are α-equivalent if $\mathit{NF}_\alpha(t_1) \equiv \mathit{NF}_\alpha(t_2)$ holds, and we use $t_1 \equiv_\alpha t_2$ to indicate that t_1 and t_2 are α-equivalent. Note that $\equiv_\alpha$ is an equivalence relation, that is, $\equiv_\alpha$ is reflexive, symmetric and transitive.

Clearly, we have $\mathit{NF}_\alpha(I) = \lambda(1)$, $\mathit{NF}_\alpha(K) = \lambda(\lambda(1))$, and $\mathit{NF}_\alpha(S) = \lambda(\lambda(\lambda(1(3)(2(3)))))$, and note that $\lambda(\mathit{shift}(\mathit{NF}_\alpha(I))) = \lambda(\lambda(2)) = \mathit{NF}_\alpha(K')$.

Given $\underline{t}$ and x, we use $\underline{t}[1/x]$ and $\underline{t}[y/x]$ for the α-normal forms obtained from replacing each occurrence of x in $\underline{t}$ with 1 and y, respectively. For brevity, the formal defintions for these replacements are omitted.

Proposition 2.1.6 *For every λ-abstraction $\lambda x.t$, we have*

$$\mathit{NF}_\alpha(\lambda x.t) = \lambda(\mathit{shift}(\mathit{NF}_\alpha(t))[1/x])$$

Proof Let us first establish the following equation for all sequences Γ:

$$\mathit{NF}_\alpha(x, \Gamma; t) = \mathit{shift}(\mathit{NF}_\alpha(\Gamma, t))[1/x]$$

We proceed by structural induction on t:

- t is x. If $x \in \Gamma$, then both sides of the equation equal $\Gamma(x) + 1$. Otherwise, both sides of the equation equal 1.
- t is some variable y that is distinct from x. If $y \in \Gamma$, then both sides of the equation equal $\Gamma(y) + 1$. Otherwise, both sides of the equation equal y.
- t is of the form $\lambda x_1.t_1$. By definition, $\mathit{NF}_\alpha(x, \Gamma; t) = \lambda(\mathit{NF}_\alpha(x, \Gamma, x_1; t_1))$. By induction hypothesis on t_1, we have the following:

  $$\mathit{NF}_\alpha(x, \Gamma, x_1; t_1) = \mathit{shift}(\mathit{NF}_\alpha(\Gamma, x_1; t_1))[1/x]$$

  Note that we have:

  $$\lambda(\mathit{shift}(\mathit{NF}_\alpha(\Gamma, x_1; t_1))[1/x]) = \mathit{shift}(\lambda(\mathit{NF}_\alpha(\Gamma, x_1; t_1)))[1/x]$$

  Hence, $\mathit{NF}_\alpha(x, \Gamma; t) = \mathit{shift}(\mathit{NF}_\alpha(\Gamma; t))[1/x]$ holds.

- t is of the form $t_1(t_2)$. This is straightforward based on the properties of $NF_\alpha(\cdot)$, $shift(\cdot)$ and variable replacement.

We conclude the inductive proof as all the cases are covered. Let Γ be $\emptyset$, and we have $NF_\alpha(x;t) = shift(NF_\alpha(t))[1/x]$. Therefore, we have $NF_\alpha(\lambda x.t) = \lambda(shift(NF_\alpha(t))[1/x])$. ∎

Given a λ-abstraction $\lambda x.t$, let us choose a variable y not in $vars(t)$. By Proposition 2.1.6, we have

$$NF_\alpha(\lambda x.t) = \lambda(shift(NF_\alpha(t))[1/x]) \text{ and } NF_\alpha(\lambda y.t[y/x]) = \lambda(shift(NF_\alpha(t[y/x]))[1/y])$$

It should be easy to note that $shift(NF_\alpha(t))[1/x] = shift(NF_\alpha(t[y/x]))[1/y]$. Therefore, $\lambda x.t$ and $\lambda y.t[y/x]$ are α-equivalent.

2.2 Substitution

Definition 2.2.1 *(Substiutions) We use θ for substituitions, which are finite mappings from variables to λ-terms:*

$$substitutions \quad \theta \quad ::= \quad [] \mid \theta[x \mapsto t]$$

We may use $[]$ for the empty mapping and $\theta[x \mapsto t]$ for the mapping that extends θ with a link from x to t, where x is assumed to be not in $\mathbf{dom}(\theta)$. We use $\mathbf{dom}(\theta)$ for the (finite) domain of θ and $vars(\theta)$ for the following (finite) set of variables:

$$\mathbf{dom}(\theta) \cup (\cup_{x \in \mathbf{dom}(\theta)} vars(\theta(x)))$$

We may use $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ for the subtitution θ such that $\mathbf{dom}(\theta) = \{x_1, \ldots, x_n\}$ and $\theta(x_i) = t_i$ for $1 \leq i \leq n$, where $x_1, \ldots, x_n$ are assumed to be distinct variables.

Definition 2.2.2 *Given a λ-term t and a substitution θ, we use $t[\theta]$ for the result of applying the substitution θ to t, which is formally defined below as a function by induction on the size of t:*

- $t[\theta] = \theta(x)$ *if t is some x in* $\mathbf{dom}(\theta)$.
- $t[\theta] = x$ *if t is some x not in* $\mathbf{dom}(\theta)$.
- $t[\theta] = \lambda y.(t_1[y/x])[\theta]$ *if t is $\lambda x.t_1$, where y is the first variable not in $vars(t_1) \cup vars(\theta)$. Note that the reason for choosing y in such a manner is to guarantee that applying a substitution θ to a term t can be done deterministically.*
- $t[\theta] = t_1[\theta](t_2[\theta])$ *if* $t = t_1(t_2)$.

Given two substitutions θ_1 and θ_2, we write $\theta_1 \equiv_\alpha \theta_2$ to mean that $\theta_1(x) \equiv_\alpha \theta_2(x)$ holds for every $x \in \mathbf{dom}(\theta_1) = \mathbf{dom}(\theta_2)$. We are to prove that $t[\theta] \equiv_\alpha t'[\theta']$ whenever $\theta \equiv_\alpha \theta'$ and $t \equiv_\alpha t'$, that is, the operation of applying a substitution to a term is well-defined modulo α-equivalence.

Let us use $\underline{\theta}$ for finite mappings from variables to α-normal forms and $shift(\underline{\theta})$ be the mapping $\underline{\theta}'$ such that $\mathbf{dom}(\underline{\theta}') = \mathbf{dom}(\underline{\theta})$ and $\underline{\theta}'(x) = shift(\underline{\theta}(x))$ for each $x \in \mathbf{dom}(\underline{\theta}')$. Given $\underline{t}$, we define $\underline{t}[\underline{\theta}]$ as follows:

$$\underline{t}[\underline{\theta}] = \begin{cases} \underline{\theta}(x) & \text{if } t = x; \\ n & \text{if } t = n; \\ \lambda(t_1[\underline{\theta}']) & \text{if } t = \lambda(\underline{t}_1) \text{ and } \underline{\theta}' = shift(\underline{\theta}); \\ t_1[\underline{\theta}](t_2[\underline{\theta}]) & \text{if } t = t_1(t_2). \end{cases}$$

Proposition 2.2.3 *Given x, t and θ, if y is a variable not in vars$(t) \cup$ vars(θ), then we have the following equation:*

$$shift(\underline{t}[y/x][\underline{\theta}])[1/y] = shift(\underline{t})[1/x][shift(\underline{\theta})]$$

Proof We proceed by structural induction on $\underline{t}$. For brevity, we only consider the case where t is of the form $\lambda(\underline{t}_1)$. Note that $shift(\underline{t}[y/x][\underline{\theta}]) = \lambda(shift(\underline{t}_1[y/x][\underline{\theta}']))$ in this case, where $\underline{\theta}' = shift(\underline{\theta})$. By induction hypothesis on $\underline{t}_1$, we have:

$$shift(\underline{t}_1[y/x][\underline{\theta}'])[1/y] = shift(\underline{t}_1)[1/x][shift(\underline{\theta}')]$$

Note that $shift(\underline{t})[1/x][shift(\underline{\theta})] = \lambda(shift(\underline{t}_1)[1/x])[\underline{\theta}'] = \lambda(shift(\underline{t}_1)[1/x][shift(\underline{\theta}')])$, and we have

$$shift(\underline{t}[y/x][\underline{\theta}])[1/y] = \lambda(shift(\underline{t}_1[y/x][\underline{\theta}'])[1/y]) = shift(\underline{t})[1/x][shift(\underline{\theta})]$$

All of the other cases can be readily handled. ∎

Proposition 2.2.4 *We have the following equation:*

$$NF_\alpha(t[\theta]) = NF_\alpha(t)[NF_\alpha(\theta)]$$

In other words, the subsitution function given in Definition 2.2.2 is well-defined modulo the α-equivalence relation.

Proof Let $\underline{\theta} = NF_\alpha(\theta)$. We proceed by induction on the size of t. The only interesting case is the one where t is of the form $\lambda x_1.t_1$. By definition, $t[\theta] = \lambda y.t_1[y/x_1][\theta]$, where y is some variable not appearing in vars$(t_1) \cup$ vars(θ). By induction hypothesis on $t_1[y/x_1]$, $NF_\alpha(t_1[y/x_1][\theta]) = NF_\alpha(t_1[y/x_1])[\underline{\theta}]$ holds. Let $\underline{t}_1 = NF_\alpha(t_1)$, and we have $NF_\alpha(t_1[y/x_1]) = \underline{t}_1[y/x_1]$. By Proposition 2.2.3, we have

$$\lambda(shift(NF_\alpha(t_1[y/x_1][\theta]))[1/y]) = \lambda(shift(\underline{t}_1[y/x_1][\underline{\theta}])[1/y]) = \lambda(shift(\underline{t}_1)[1/x_1][shift(\underline{\theta})])$$

Note that $NF_\alpha(t) = \lambda(shift(\underline{t}_1)[1/x1])$, which leads to $NF_\alpha(t)[\underline{\theta}] = \lambda(shift(\underline{t}_1)[1/x1][shift(\underline{\theta})])$. So we have $NF_\alpha(t[\theta]) = NF_\alpha(t)[NF_\alpha(\theta)]$ in this case. All of the other cases can be readily handled. ∎

Given θ_1 and θ_2, we use $\theta_2 \circ \theta_1$ for the substitution θ such that $\mathbf{dom}(\theta) = \mathbf{dom}(\theta_1) \cup \mathbf{dom}(\theta_2)$, and for each $x \in \mathbf{dom}(\theta)$, $\theta(x) = x[\theta_1][\theta_2]$.

Lemma 2.2.5 $(t[\theta_1])[\theta_2] \equiv_\alpha t[\theta_2 \circ \theta_1]$.

Proof As an exercise. ∎

Given a λ-abstraction $\lambda x.t$ and a finite set of variables, we can also choose another λ-abstraction $\lambda x'.t'$ that is α-equivalent to $\lambda x.t$ while guaranteeing that x' does not occur in the given finite set of variables. This is often called α-*conversion* or α-*renaming* (of a bound variable).

2.3 β-Reduction

Definition 2.3.1 (β-redexes) *A λ-term t is a β-redex if it is of the form $\lambda x.t_1(t_2)$, and its contractum is $t_1[x := t_2]$. We may also refer to the contractum of a β-redex as the reduct of the β-redex.*

Given a λ-term t, $\mathcal{R}$ is a set of β-redexes in t if $\mathcal{R}$ a finite set of paths such that $subterm(t, p)$ is a β-redex for each $p \in \mathcal{R}$.

Definition 2.3.2 (λ-term Contexts)

$$\textit{contexts} \quad C \quad ::= \quad [] \mid \lambda x.C \mid C(t) \mid t(C)$$

Given a context C and a λ-term t, we use $C[t]$ for the λ-term obtained from replacing the hole [] in C, which is formally defined below:

$$C[t] = \begin{cases} t & \text{if } C \text{ is } []; \\ \lambda x.(C_0[t]) & \text{if } C \text{ is } \lambda x.C_0; \\ C_1[t](t_2) & \text{if } C \text{ is } C_1(t_2); \\ t_1((C_2[t])) & \text{if } C \text{ is } t_1(C_2). \end{cases}$$

Given a context C and a path p, we use $subterm(C, p)$ for either a context or a term defined below:

$$\begin{array}{rcl} subterm(C, \emptyset) & = & C \\ subterm(C(t), 0.p) & = & subterm(C, p) \\ subterm(t(C), 0.p) & = & subterm(t, p) \\ subterm(C(t), 1.p) & = & subterm(t, p) \\ subterm(t(C), 1.p) & = & subterm(C, p) \\ subterm(\lambda x.C, 0.p) & = & subterm(C, p) \end{array}$$

Definition 2.3.3 (β-reduction) *Given two λ-terms t_1, t_2 and a path p, we write $[p] : t_1 \to_\beta t_2$ if $t_1 \equiv C[t]$ for some context C and β-redex t, where $subterm(C, p) = []$, and $t_2 \equiv C[t']$ for the reduct t' of t. A reduction $[p]$ is a top reduction if $p = \emptyset$, and it is a head reduction if $p = 0 \ldots 0.\emptyset$.*

Let $\omega = \lambda x.x(x)$ and $\Omega = \omega(\omega)$. Clearly, we have $[\emptyset] : \Omega \to_\beta \Omega$, which is a top reduction.

We may write $t_1 \to_\beta t_2$ to mean $[p] : t_1 \to_\beta t_2$ for some p. We refer to the binary relation $\to_\beta$ as (one-step) β-reduction, and use $\to_\beta^+$ and $\to_\beta^*$ for the transitive closure and the reflexive and transitive closure of $\to_\beta$, respectively. We may also refer to $\to_\beta^*$ as multi-step β-reduction. In addtion, we use $\equiv_\beta$ for the minimal equivalence relation containing $\to_\beta$.

Definition 2.3.4 (β-reduction Sequences) *We use σ for (finite) β-reduction sequences defined as follows:*

$$\textit{β-reduction sequences} \quad \sigma \quad ::= \quad \emptyset \mid [p] + \sigma$$

where $\emptyset$ stands for the empty β-reduction sequence. Note that we may omit writing the trailing $\emptyset$ in β-reduction sequence.

We write $\sigma : t \to_\beta^* t'$ to mean that σ is a β-reduction sequence from t to t', that is, σ is of the form $[p_1] + \ldots + [p_n] + \emptyset$ and there are λ-terms $t = t_1, \ldots, t_{n+1} = t'$ such that $[p_i] : t_i \to_\beta t_{i+1}$ holds for each $1 \leq i \leq n$. We write $\sigma : t$ to mean $\sigma : t \to_\beta^* t'$ for some t', which can be denoted by $\sigma(t)$.

Proposition 2.3.5 *Assume $\sigma : t \to^*_\beta t'$ and $\sigma = [p_1] + [p_2]$. If p_1 and p_2 are incompatible, then we have $\sigma' : t \to^*_\beta t'$ for $\sigma' = [p_2] + [p_1]$.*

Proof By structural induction on t. ∎

Let σ be a β-reduction sequence from t such that each $[p]$ in σ implies $p = 0.p_1$ or $p = 1.p_1$ for some p_1. By applying Proposition 2.3.5 repeatedly, we can arrange σ into σ' of the form $0.\sigma'_1 + 1.\sigma'$ such that $\sigma(t) = \sigma'(t)$.

Proposition 2.3.6 *Assume $\sigma : t \to^*_\beta t'$. Then for every substitution θ, we also have $\sigma : t[\theta] \to^*_\beta t'[\theta]$.*

Proof It is straightforward to verify that $[p] : t \to_\beta t'$ implies $[p] : t[\theta] \to_\beta t'[\theta]$. Then the proposition follows immediately. ∎

Lemma 2.3.7 *Assume that $\sigma : t$. If $\sigma = \sigma_1 + [\emptyset]$ for some σ_1 containing no head reduction, that is, $[\emptyset]$ does not appear in σ_1, then there exists $\sigma' : t$ such that $\sigma' = [\emptyset] + \sigma'_1$ for some σ'_1 and $\sigma'(t) = \sigma(t)$.*

Proof Since σ contains a head reduction, t must be of the form $\lambda x.t_1(t_2)$. Since σ_1 contains no head reduction, we can assume by Proposition 2.3.5 that $\sigma_1 = 0.0.\sigma_{11} + 1.\sigma_{12}$ for some $\sigma_{11} : t_1$ and $\sigma_{12} : t_2$. Clearly, $\sigma_1(t) = \lambda x.\sigma_{11}(t_1)(\sigma_{12}(t_2))$, and $\sigma(t) = \sigma_{11}(t_1)[x := \sigma_{12}(t_2)]$. Let $p^x_1, \ldots, p^x_n$ be an enumeration of the possitions of all free occurrences of x in $\sigma(t_1)$, and

$$\sigma' = [\emptyset] + \sigma_{11} + p^x_1@\sigma_{12} + \ldots + p^x_n@\sigma_{12}$$

Clearly, we have $\sigma' : t$ and $\sigma'(t) = \sigma(t)$. ∎

2.4 Developments

Definition 2.4.1 (Residuals under β-reduction) *Let t_0 be $C[(\lambda x.t_1)t_2]$, where the hole $[]$ in C is at some position p_0, and $t'_0 = C[t_1[x := t_2]]$. For each β-redex in t_0 at some position p, the residuals of the β-redex, denoted by* **Res**(t_0, p_0, p)*, is a set of subterms in t'_0 defined as follows:*

1. *If $p = p_0$, then* **Res**$(t_0, p_0, p) = \emptyset$.
2. *If $p = (p_0.0)@p'$, then* **Res**$(t_0, p_0, p) = \{p_0@p'\}$.
3. *If $p = (p_0.1)@p'$, then* **Res**$(t_0, p_0, p) = \{p_0@p_x@p'\}$*, where p_x ranges over all paths such that $subterm(t_1, p_x)$ is a free occurrence of x in t_1.*
4. *Otherwise, p_0 is not a prefix of p. In this case,* **Res**$(t_0, p_0, p) = \{p\}$.

Clearly, the residuals of a β-redex are β-redexes themselves.

Definition 2.4.2 (λ-terms with marked β-redexes) *Given a λ-term t and a set $\mathcal{R}$ of β-redexes in t, we write $t/\mathcal{R}$ a λ-term with marked β-redexes, or a marked λ-term for short.*

Definition 2.4.3 (Developments) *Assume that t is a λ-term and $\mathcal{R}$ is a set of β-redexes in t. A β-reduction sequence σ is from $t/\mathcal{R}$ is called a development if σ is empty, or $\sigma = [p] + \sigma_1$ for some $p \in \mathcal{R}$ and $[p]$ reduces $t/\mathcal{R}$ to $t_1/\mathcal{R}_1$ and σ_1 is a development of $t_1/\mathcal{R}_1$.*

A finite development σ of $t/\mathcal{R}$ is complete if $\sigma(t/\mathcal{R}) = t'/\emptyset$ for some t'.

Lemma 2.4.4 *Let P be a unary predicate on marked λ-terms. Assume that P is modulo α-equivalence, that is, $P(t_1/\mathcal{R})$ implies $P(t_2/\mathcal{R})$ whenever $t_1 \equiv_\alpha t_2$ holds, and*

1. *$P(x/\emptyset)$ holds for every variable x.*
2. *For $t/\mathcal{R}$, $P(t/\mathcal{R})$ implies $P(\lambda x.t/0.\mathcal{R})$.*
3. *For $t_1/\mathcal{R}_1$ and $t_2/\mathcal{R}_1$, $P(t_1/\mathcal{R}_1)$ and $P(t_2/\mathcal{R}_2)$ implies $P(t/\mathcal{R})$, where $t = t_1(t_2)$ and $\mathcal{R} = 0.\mathcal{R}_1 \cup 1.\mathcal{R}_2$*
4. *For $t_1/\mathcal{R}_1$ and $t_2/\mathcal{R}_2$, $P(t_1/\mathcal{R}_1)$, $P(t_2/\mathcal{R}_2)$ and $P(t_1/\mathcal{R}_1[x := t_2/\mathcal{R}_2])$ implies $P(t/\mathcal{R} \cup \{\emptyset\})$, where $t = (\lambda x.t_1)(t_2)$ and $\mathcal{R} = \{\emptyset\} \cup 0.0.\mathcal{R}_1 \cup 1.\mathcal{R}_2$.*

Then $P(t/\mathcal{R})$ holds for every marked λ-term $t/\mathcal{R}$.

Proof We first prove that for every marked λ-term $t/\mathcal{R}$, $P(t/\mathcal{R}[\theta])$ holds for every substitution θ that maps variables to marked λ-terms satisfying P, that is, $P(\theta(x))$ for each $x \in \mathbf{dom}(\theta)$. We proceed by structural induction on t.

- t is some variable x. Then $t[\theta]$ is either x or $\theta(x)$. So $P(t[\theta])$ holds.
- t is $\lambda x_0.t_0$ for some λ-term t_0. Then $\mathcal{R} = 0.\mathcal{R}_0$ for some set $\mathcal{R}_0$ of redexes in t_0. Given that P is modulo α-equivalence, we may assume $t[\theta] \equiv \lambda x_0.t_0[\theta]$ without loss of generality. By induction hypothesis on t_0, we have $P(t_0[\theta])$. By (2), we have $P(t[\theta]/\mathcal{R})$.
- t is $t_1(t_2)$ and $\emptyset \notin \mathcal{R}$. Then $\mathcal{R} = 0.\mathcal{R}_1 \cup 1.\mathcal{R}_2$ for some sets $\mathcal{R}_0$ and $\mathcal{R}_1$ of redexes in t_1 and t_2, respectively. Clearly, $t[\theta] = t_1[\theta]((t_2[\theta]))$. By induction hypothesis, both $P(t_1/\mathcal{R}_0[\theta])$ and $P(t_2/\mathcal{R}_1[\theta])$ hold. By (3), we have $P(t/\mathcal{R}[\theta])$.
- t is $(\lambda x.t_1)(t_2)$ and $\emptyset \in \mathcal{R}$. Then $\mathcal{R} = \{\emptyset\} \cup 0.0.\mathcal{R}_1 \cup 1.\mathcal{R}_2$ for some $\mathcal{R}_1$ and $\mathcal{R}_2$. By induction hypothesis on t_2, $P(t_2/\mathcal{R}_2[\theta])$ holds. Let θ' be $\theta[x \mapsto t_2/\mathcal{R}_2[\theta]]$. By induction hypothesis on t_1, $P(t_1/\mathcal{R}_1[\theta'])$ holds. Note that $t_1/\mathcal{R}_1[\theta']$ is α-equivalent to $t_1/\mathcal{R}_1[\theta][x := t_2/\mathcal{R}_2[\theta]]$. By (4), $P(t/\mathcal{R}[\theta])$ holds.

Therefore, by structural induction, $P(t/\mathcal{R}[\theta])$ for all λ-terms t and all substitutions θ such that $P(\theta(x))$ holds for each $x \in \mathbf{dom}(\theta)$. Let θ be the empty substitution, and we have $P(t/\mathcal{R})$ for all λ-terms t. ∎

Theorem 2.4.5 *(Finite Developments) For each λ-term t, there exists a number n such that the length of every development from t is less than or equal to n.*

Proof Let $P(t/\mathcal{R})$ be the statement that there is a number n such that the length of every development from $t/\mathcal{R}$ is less than or equal to n.

- Assume $t = x$ for some variable x. Then we can choose n to be 0.
- Assume $t = \lambda x.t_1$. Then $\mathcal{R} = 0.\mathcal{R}_1$ for some set $\mathcal{R}_1$ of β-redexes in t_1. Note in this case that each developlement from $t/\mathcal{R}$ is of the form $0.\sigma_1$ for some development σ_1 from $t_1/\mathcal{R}_1$. Hence, $P(t_1/\mathcal{R}_1)$ implies $P(t/\mathcal{R})$.

- Assume $t = t_1(t_2)$ and $\emptyset \notin \mathcal{R}$. Then $\mathcal{R} = 0.\mathcal{R}_1 \cup 1.\mathcal{R}_2$, and each development from $t/\mathcal{R}$ can essentially be written as $0.\sigma_1 + 1.\sigma_2$, where σ_1 and σ_2 are some developments from $t_1/\mathcal{R}_1$ and $t_2/\mathcal{R}_2$, respectively. Clearly, $P(t_1/\mathcal{R}_1)$ and $P(t_2/\mathcal{R}_2)$ implies $P(t/\mathcal{R})$ in this case.

- Assume $t = (\lambda x.t_1)(t_2)$ and $\emptyset \in \mathcal{R}$. Let σ be any development from $t/\mathcal{R}$. We may assume that $[\emptyset]$ appears in σ for otherwise we can simply take $\sigma + [\emptyset]$ instead. Let $\sigma = \sigma_1 + [\emptyset] + \sigma_2$. By studying the proof of Lemma 2.3.7, we see that there is a development σ' from $t/\mathcal{R}$ that is of the form $[\emptyset] + \sigma_1' + \sigma_2$. Assume $[\emptyset] : t/\mathcal{R} \rightarrow_\beta t'/\mathcal{R}'$. Then we clearly have $P(t_1/\mathcal{R}_1)$, $P(t_2/\mathcal{R}_2)$ and $P(t'/\mathcal{R}')$ implies $P(t/\mathcal{R})$.

By Lemma 2.4.4, we have $P(t/\mathcal{R})$ for all $t/\mathcal{R}$. Given t, let $\mathcal{R}_t$ be the set of all β-redexes in t. Then every development from t is a development from $t/\mathcal{R}_t$. Hence, we are done. ∎

Given $t/\mathcal{R}$, let $\mu_0(t/\mathcal{R})$ be the maximum of $length(\sigma)$, where σ ranges over all the developments of $t/\mathcal{R}$. Let $\mu_0(t)$ be $\mu_0(t/\mathcal{R}_t)$, where $\mathcal{R}_t$ is the set of all β-redexes in t. Theorem 2.4.5 simply states that $\mu_0(t) < \infty$ for all t.

Lemma 2.4.6 *Assume that σ_1 and σ_2 are two complete developments from $t/\mathcal{R}$. Then $\sigma_1(t/\mathcal{R}) = \sigma_2(t/\mathcal{R})$.*

Proof Let $P(t/\mathcal{R})$ be the statement that $\sigma_1(t/\mathcal{R}) = \sigma_2(t/\mathcal{R})$ for every pair of complete developments σ_1 and σ_2 from $t/\mathcal{R}$.

- Assume $t = x$ for some variable x. Clearly, $P(t/\mathcal{R})$ holds.

- Assume $t = \lambda x.t_1$. Then $\mathcal{R} = 0.\mathcal{R}_1$ for some set β-redexes in t_1. Clearly $P(t_1/\mathcal{R}_1)$ implies $P(t/\mathcal{R})$.

- Assume $t = t_1(t_2)$ and $\emptyset \notin \mathcal{R}$. Then $\mathcal{R} = 0.\mathcal{R}_1 \cup 1.\mathcal{R}_2$, where $\mathcal{R}_1$ and $\mathcal{R}_2$ are some sets of β-redexes in t_1 and t_2, respectively. Clearly, $P(t_1/\mathcal{R}_1)$ and $P(t_2/\mathcal{R}_2)$ implies $P(t/\mathcal{R})$.

- Assume $t = (\lambda x.t_1)(t_2)$ and $\emptyset \in \mathcal{R}$. Let $\sigma_1 = \sigma_{11} + [\emptyset] + \sigma_{12}$ and $\sigma_2 = \sigma_{21} + [\emptyset] + \sigma_{22}$. By studying the proof of Lemma 2.3.7, we see that there is a complete development σ_1' from $t/\mathcal{R}$ that is of the form $[\emptyset] + \sigma_{11}' + \sigma_{12}$, and $\sigma_1'(t) = \sigma_1(t)$. Similarly, there is a complete development σ_2' from $t/\mathcal{R}$ that is of the form $[\emptyset] + \sigma_{21}' + \sigma_{22}$, and $\sigma_2'(t) = \sigma_2(t)$. Assume $[\emptyset] : t/\mathcal{R} \rightarrow_\beta t'/\mathcal{R}'$. Note that $\sigma_{11}' + \sigma_{12}$ and $\sigma_{21}' + \sigma_{22}$ are complete developments from $t'/\mathcal{R}'$. Hence, $P(t'/\mathcal{R}')$ implies $P(t/\mathcal{R})$.

By Lemma 2.4.4, we have $P(t/\mathcal{R})$ for all $t/\mathcal{R}$, which yields this lemma. ∎

Lemma 2.4.7 *Assume σ_1 and σ_2 are developments from t. Then there exists σ_1' and σ_2' such that $\sigma_1 + \sigma_2'$ and $\sigma_2 + \sigma_1'$ are developments from t to some term t'.*

Proof Assume σ_1 and σ_2 are developments from $t/\mathcal{R}_1$ and $t/\mathcal{R}_2$, respectively. Then σ_1 and σ_2 are also developments from $t/\mathcal{R}$ for $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. By Theorem 2.4.5, there exists σ_1' and σ_2' such that both $\sigma_1 + \sigma_2'$ and $\sigma_2 + \sigma_1'$ are complete developments from $t/\mathcal{R}$. By Lemma 2.4.6, we have

$$(\sigma_1 + \sigma_2')(t/\mathcal{R}) = (\sigma_2 + \sigma_1')(t/\mathcal{R})$$

This concludes the proof. ∎

Definition 2.4.8 *(Standard Developments) A standard development σ from $t/\mathcal{R}$ is standard if*

1. *σ is empty, or*
2. *$\sigma = [p] + \sigma_1$ for the leftmost p in $\mathcal{R}$ and σ_1 is a standard devlopment of $[p](t/\mathcal{R})$.*

2.5 Fundamental Theorems of λ-calculus

Lemma 2.5.1 *Assume that σ_1 is a development from t. For every finite β-reduction sequence σ_2 from t, there exists a development σ_1' and a β-reduction sequence σ_2' such that $(\sigma_1 + \sigma_2')(t) = (\sigma_2 + \sigma_1')(t)$.*

Proof We proceed by induction on the length of σ_2.

- $\sigma_2 = \emptyset$. Let $\sigma_1' = \sigma_1$ and $\sigma_2' = \emptyset$, and we are done.
- $\sigma_2 = \sigma_{20} + \sigma_{21}$, where σ_{20} is a nonempty development. Let σ_{10} be σ_1. By Lemma 2.4.7, there exists two developments σ_{10}' and σ_{20}' such that $(\sigma_{10} + \sigma_{20}')(t) = (\sigma_{20} + \sigma_{10}')(t)$. By induction hypthesis on $\sigma(t)$, there exist a development σ_{10}'' and a β-reduction sequence σ_{21}' such that $(\sigma_{10}' + \sigma_{21}')(\sigma_{20}(t)) = (\sigma_{21} + \sigma_{10}'')(\sigma_{20}(t))$. Let $\sigma_1' = \sigma_{10}''$ and $\sigma_2' = \sigma_{20}' + \sigma_{21}'$, and we are done.

If $\sigma_2 = \sigma_{21} + \ldots + \sigma_{2n}$ for some developments $\sigma_{21}, \ldots, \sigma_{2n}$, then it is clear from the proof that σ_2' can be written in the form of $\sigma_{21}' + \ldots + \sigma_{2n}'$, where σ_{2i}' are all development for $1 \leq i \leq n$. In other words, if σ_2 is a concatenation of n developments, then σ_2' can also be chosen to be a concatenation of n developments. ∎

Lemma 2.5.1 is often referred to as *Strip Lemma* for the obvious reason.

Theorem 2.5.2 *(Church-Rosser) Assume that $t \equiv_\beta t'$, where $\equiv_\beta$ is the minimal equivalence relation containing $\rightarrow_\beta$. Then there exists $\sigma_1 : t$ and $\sigma_2 : t'$ such that $\sigma_1(t) = \sigma_2(t')$.*

Proof $t \equiv_\beta t'$ implies the existence of λ-terms $t_0, t_1 \ldots, t_n$ for some $n \geq 1$ such that $t = t_0$ and $t' = t_n$ and for each $0 \leq i < n$, either $t_i \rightarrow_\beta t_{i+1}$ or $t_{i+1} \rightarrow_\beta t_i$ holds. We proceed by induction on n.

- Assume $n = 1$. We omit this trivial case for brevity.
- Assume $n > 0$. By induction hypothesis, we have σ_{11} and σ_{12} such that $\sigma_{11}(t_1) = \sigma_{12}(t_n)$.
  - Assume $[p] : t_0 \rightarrow_\beta t_1$ for some p. Let $\sigma_1 = [p] + \sigma_{11}$ and $\sigma_2 = \sigma_{12}$, and we are done.
  - Assume $[p] : t_1 \rightarrow_\beta t_0$ for some p. Clearly, $[p]$ is a development, By Lemma 2.5.1, we have σ_{10} and σ_{20} such that

    $$([p] + \sigma_{10})(t_1) = (\sigma_{11} + \sigma_{20})(t_1)$$

    Let $\sigma_1 = \sigma_{10}$ and $\sigma_2 = \sigma_{12} + \sigma_{20}$, and we are done.

We conclude the induction proof as all the cases are covered. ∎

Lemma 2.5.3 *Assume $\sigma = \sigma_1 + \sigma_2$ is finite β-reduction sequence for a λ-term t, where σ_1 is a standard development and σ_2 is a standard β-reduction sequence. Then we can construct a standard (finite) β-reduction sequence σ' from t such that $\sigma(t) = \sigma'(t)$.*

Proof We are to define a binary function std_2 that takes the arguments σ_1 and σ_2 and returns σ'. ■

Theorem 2.5.4 *(Standardization)*

Given a λ-term t, we use $norm_\beta(t)$ for the (possibly infinite) reduction sequence σ from t such that each β-reduction step in σ is leftmost and $\sigma(t)$ is in normal form σ is finite.

Theorem 2.5.5 *(Normalization) Assume $\nu(t) < \infty$. Then $norm_\beta(t)$ is finite.*

Lemma 2.5.6 *Assume $\mu(u[x := v](t_1) \ldots (t_n)) < \infty$ and $\mu(v) < \infty$. Then we have:*

$$\mu((\lambda x.u)(v)(t_1) \ldots (t_n)) \leq 1 + \mu(u[x := v](t_1) \ldots (t_u)) + \mu(v)$$

Proof Let $t^* = u[x := v](t_1) \ldots (t_n)$, Clearly, $\mu(t^*) < \infty$ implies that $\mu(u), \mu(t_1), \ldots, \mu(t_n)$ are all finite. We proceed by induction on $\mu(u)+\mu(v)+\mu(t_1)+\ldots+\mu(t_n)$. Let $t = \mu((\lambda x.u)(v)(t_1) \ldots (t_n))$. Assume $[p] : t \rightarrow_\beta t'$, and we do a case analysis on p.

- Assume p in u. Then $t' = (\lambda x.u')(v)(t_1) \ldots (t_n)$ for some u' such that $u \rightarrow_\beta u'$ holds. By induction hypothesis,

$$\mu(t') \leq 1 + \mu(u'[x := v](t_1) \ldots (t_u)) + \mu(v) \leq \mu(t^*) + \mu(v)$$

- Assume p in v. Then $t' = (\lambda x.u)(v')(t_1) \ldots (t_n)$ for some v' such that $v \rightarrow_\beta v'$ holds. By induction hypothesis,

$$\mu(t') \leq 1 + \mu(u[x := v'](t_1) \ldots (t_u)) + \mu(v') \leq \mu(t^*) + \mu(v)$$

- Assume p in t_i for some $1 \leq i \leq i$. Then $t' = (\lambda x.u)(v)(t_1) \ldots (t_i') \ldots (t_n)$ for some t_i' such that $t_i \rightarrow_\beta t_i'$ holds. By induction hypothesis,

$$\mu(t') \leq 1 + \mu(u[x := v](t_1) \ldots (t_i') \ldots (t_u)) + \mu(v) \leq \mu(t^*) + \mu(v)$$

- Assume that p is the outmost β-redex $(\lambda x.u)(v)$. Then $t^* = t'$. So, $\mu(t') \leq \mu(t^*) + \mu(v)$.

So $\mu(t') \leq \mu(t^*) + \mu(v)$ for each t' such that $t \rightarrow_\beta t'$ holds, and this yields $\mu(t) \leq 1 + \mu(t^*) + \mu(v)$. ■

Definition 2.5.7 (λ_I-terms and β_I-redexes) *A λ-term t_0 is a λ_I-term if for every subterm t of t_0, t being of the form $\lambda x.t_1$ implies $x \in FV(t_1)$. Moreover, a β-redex $\lambda x.t_1(t_2)$ is a β_I-redex if $x \in FV(t_1)$.*

Lemma 2.5.8 *Assume that σ is a development of $t/\mathcal{R}$, where $\mathcal{R}$ is a set of β_I-redexes. Then $\mu(\sigma(t)) < \infty$ implies $\mu(t) < \infty$.*

Proof Let $t' = \sigma(t)$. Assume $\mu(t') < \infty$, and we proceed to prove $\mu(t) < \infty$ by induction on $\langle \mu(t'), size(t') \rangle$, lexicographically ordered.

- Assume $t = \lambda x.t_1$. Then $\sigma = 0.\sigma_1$ for some $\sigma_1 : t_1$. Clearly, $t' = \lambda x.t'_1$, where $t'_1 = \sigma_1(t_1)$. Note $\mu(t'_1) = \mu(\lambda x.t'_1)$ and $size(t'_1) < size(t')$. By induction hypothesis on t'_1, $\mu(t_1) < \infty$ holds. Hence, $\mu(t) < \infty$ holds as well.
- Assume $t = x(t_1)\ldots(t_n)$.
- Assume $t = (\lambda x.u)(v)(t_1)\ldots(t_n)$.

■

Corollary 2.5.9 *(Conservation) Assume $[p] : t \to_\beta t'$ and $\mu(t') < \infty$. If $subterm(t, p)$ is β_I-redex, then $\mu(t) < \infty$.*

Proof By Lemma 2.5.8 immediately. ■

Corollary 2.5.10 *A λ_I-term is strongly normalizing if and only if it is weakly normalizing.*

Proof If t is strongly normalizing, then it is obviously weakly normalizing. If t is weakly normalizing, then there exists a finite reduction sequence $\sigma : t \to^*_\beta t'$ such that t' is in β-normal form. It is trivial to verify that each β-redex reduced in σ is a β_I-redex. By Theorem 2.5.9, t is strongly normalizing. ■

2.6 Exercises

Exercise 1 *Assume $t \equiv_\alpha t'$. Please show that FV(t) = FV(t') holds.*

Exercise 2 *Assume that y is a variable not contained in vars(t). Let t' be $t[y/x]$, that is, the term obtained from replacing (not substituting) y for x in t. Please show that FV(t') = FV(t) if $x \notin$ FV(t), and FV(t') = (FV$(t)\backslash\{x\}) \cup \{y\}$ if $x \in$ FV(t).*

Exercise 3 *Assume $t \equiv_\alpha t'$. Please show $t[y/x] \equiv_\alpha t'[y/x]$ holds for any x if y does not occur in vars$(t) \cup$ vars(t').*

Exercise 4 *Please show $\lambda x.t \equiv_\alpha \lambda y.t[x := y]$ if $y \notin$ FV(t).*

Exercise 5 *Assume that σ is a development of t. Please show $size(\sigma(t)) < 2^{|t|}$, where $|\cdot|$ be a unary function defined as follows:*

$$|x| = 1; |\lambda x.t| = |t|; |(t_1)t_2| = |t_1| + |t_2|$$

Exercise 6 *Prove Lemma 2.4.7 by employing Lemma 2.4.4 directly.*

Exercise 7 *Assume that σ is a standard development of $t/\mathcal{R}$ that is also complete. Please show that $length(\sigma) = \mu_0(t/\mathcal{R})$ if $\mathcal{R}$ contains only β_I-redexes.*

Chapter 3

Simple Types

We study simple types in this chapter. We first formalize a simply-typed programming language $\mathcal{L}_0$ and then establish its type soundness, setting some machinery for development in the following chapters. We also prove a classic result based on the reducibilty method (Tait 1967) that simply-typed λ-calculus is strongly normalizing, namely, there is no infinite β-reduction sequence starting from any simply-typed λ-term. This proof method is to be employed later for building a theorem-proving system in which proofs are constructed as total functions.

3.1 A Simply-Typed Programming Language

We present in Figure 3 the syntax for a simply-typed language $\mathcal{L}_0$. We use δ for base types, which include int for integers and $bool$ for booleans. We use c for a constant, which is either a constant constructor cc or a constant function cf. For instance, we have $true$ and $false$ for boolean constants, and $0, -1, 1, -2, 2, \ldots$ for integer constants, and functions such as $+$ and $-$ on integers. We can simply write c for the expression $c()$ if the constant c is of arity 0. We also have lam-variables x and fix-variables f, and may use xf for either a lam-variable or a fix-variable. Note that each lam-variable following **lam** is a bound variable and so is each fix-variable following **fix**. We use v for values, which are a special form of expressions. Note that a lam-variable is a value but a fix-variable is not. We use $\vec{e}$ for a (possibly empty) sequence of expressions, and this notation may also be applied to variables, values, etc.

types	T	$::=$	$\delta \mid T_1 * T_2 \mid T_1 \rightarrow T_2$
expressions	e	$::=$	$x \mid c(\vec{e}) \mid \mathbf{if}(e_0, e_1, e_2) \mid$
			$\langle e_1, e_2 \rangle \mid \mathbf{fst}(e) \mid \mathbf{snd}(e) \mid \mathbf{lam}\, x.e \mid \mathbf{app}(e_1, e_2) \mid \mathbf{fix}\, f.e$
values	v	$::=$	$x \mid cc(\vec{v}) \mid \langle v_1, v_2 \rangle \mid \mathbf{lam}\, x.e$
typing contexts	Γ	$::=$	$\emptyset \mid \Gamma, x : T$
evaluation contexts	E	$::=$	$[] \mid c(\vec{v}, E, \vec{e}) \mid \mathbf{if}(E, e_1, e_2) \mid$
			$\langle E, e \rangle \mid \langle v, E \rangle \mid \mathbf{fst}(E) \mid \mathbf{snd}(E) \mid \mathbf{app}(E, e) \mid \mathbf{app}(v, E)$

Figure 3.1: The syntax for $\mathcal{L}_0$

Various functions on λ-terms can be readily defined on expressions in $\mathcal{L}_0$ as well. For instance, it should be obvious how *FV*$(\cdot)$ can be defined in $\mathcal{L}_0$. In particular, we have *FV*$(\mathbf{lam}\, x.e) =$ *FV*$(e)\backslash\{x\}$ *FV*$(\mathbf{fix}\, f.e) =$ *FV*$(e)\backslash\{f\}$. Also, it should be clear how α-renaming can be performed on expressions in $\mathcal{L}_0$.

We are to present the dynamic semantics of $\mathcal{L}_0$ based on redexes and evaluation contexts. A redex $\mathcal{L}_0$ is a special form of an expression, which can be reduced to another expression referred to as a reduct of the redex. In $\mathcal{L}_0$, each redex has at least one reduct.

Definition 3.1.1 (Redexes) *The redexes in $\mathcal{L}_0$ and their reducts are defined below.*

- $cf(\vec{v})$ *is a redex, and each defined value of* $cf(\vec{v})$ *is a reduct of* $cf(\vec{v})$.
- $\mathbf{if}(true, e_1, e_2)$ *is a redex, and its reduct is* e_1.
- $\mathbf{if}(false, e_1, e_2)$ *is a redex, and its reduct is* e_2.
- $\mathbf{fst}(\langle v_1, v_2\rangle)$ *is a redex, and its reduct is* v_1.
- $\mathbf{snd}(\langle v_1, v_2\rangle)$ *is a redex, and its reduct is* v_2.
- $\mathbf{app}(\mathbf{lam}\, x.e, v)$ *is a redex, and its reduct is* $e[x := v]$.
- $\mathbf{fix}\, f.e$ *is a redex, and its reduct is* $e[f := \mathbf{fix}\, f.e]$.

For instance, if we assume that $+$ represents the usual addition function on integers, then $1 + 2$ is a redex and 3 is the only reduct of $1 + 2$. More interestingly, we may also assume the existence of a nullary constant function $random$ such that $random()$ is a redex and every natural number is a reduct of $random()$.

Definition 3.1.2 (Evaluation) *We use $\rightarrow$ for the binary evaluation relation on expressions. Given e_1 and e_2, $e_1 \rightarrow e_2$ holds if $e_1 = E[e]$ and $e_2 = E[e']$ for some evaluation context E, redex e and a reduct e' of e. We use $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$.*

There are clearly (closed) non-value expressions in $\mathcal{L}_0$ that can not be further evaluated, and we refer to as such expressions as being stuck. For instance, expresssions like $\mathbf{fst}(0)$ and $\mathbf{app}(1, true)$ are stuck.

Given a program e, that is, a closed expression, we expect to avoid the scenario where evaluating the program may reach a stuck expression. To address this issue, we set out to enforce a type discipline in $\mathcal{L}_0$ so that for each well-typed program, that is, program constructed following the type discipline, evaluating the program can never lead to a stuck expression.

A typing judgment in $\mathcal{L}_0$ is of the form $\Gamma \vdash e : T$, meaning that e can be assigned the type T under the typing context Γ. The rules for deriving such judgments are given in Figure 3.2. In $\mathcal{L}_0$, each constant constructor cc is given a type of the form $(T_1, \ldots, T_n) \rightarrow \delta$, and we say that cc is associated with δ. For instance, $true$ and $false$ are given the type $() \rightarrow bool$, and each integer constant is given the type $() \rightarrow int$.

Lemma 3.1.3 (Canonical Forms) *Assume that $\emptyset \vdash v : T$ is derivable.*

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \textbf{ (ty-var)}$$

$$\frac{\vdash c : (T_1, \ldots, T_n) \to T \quad \Gamma \vdash e_i : T_i \text{ for } 1 \leq i \leq n}{\Gamma \vdash c(e_1, \ldots, e_n) : T} \textbf{ (ty-cst)}$$

$$\frac{\Gamma \vdash e_0 : \mathit{bool} \quad \Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathbf{if}(e_0, e_1, e_2) : T} \textbf{ (ty-if)}$$

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash \langle e_1, e_2 \rangle : T_1 * T_2} \textbf{ (ty-tup)}$$

$$\frac{\Gamma \vdash e : T_1 * T_2}{\Gamma \vdash \mathbf{fst}(e) : T_1} \textbf{ (ty-fst)} \qquad \frac{\Gamma \vdash e : T_1 * T_2}{\Gamma \vdash \mathbf{snd}(e) : T_2} \textbf{ (ty-snd)}$$

$$\frac{\Gamma, x : T_1 \vdash e : T_2}{\Gamma \vdash \mathbf{lam}\, x.e : T_1 \to T_2} \textbf{ (ty-lam)}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash \mathbf{app}(e_1, e_2) : T_2} \textbf{ (ty-app)}$$

$$\frac{\Gamma, f : T \vdash e : T}{\Gamma \vdash \mathbf{fix}\, f.e : T} \textbf{ (ty-fix)}$$

Figure 3.2: The typing rules for expressions in $\mathcal{L}_0$

- *If $T = \delta$ for some base type δ, then v is of the form $cc(\vec{v})$, where cc is a constructor associated with δ.*
- *If $T = T_1 * T_2$ for some types T_1 and T_2, then v is of the form $\langle v_1, v_2 \rangle$.*
- *If $T = T_1 \to T_2$ for some types T_1 and T_2, then v is of the form $\mathbf{lam}\, x.e_0$*

Proof By an inspection of the typing rules in Figure 3.2. ∎

We use θ for substitutions in $\mathcal{L}_0$, which are finte mappings from variables to expressions. Given a substitution θ and an expression, $e[\theta]$ is the result from applying θ to e, which can be formally defined as is done in Chapter **??**. In particular, when applying θ to $\mathbf{lam}\, x.e$, we may need to first α-rename $\mathbf{lam}\, x.e$ to some $\mathbf{lam}\, x'.e'$ so that x' does not occur in *vars*(θ) and then obtain $\mathbf{lam}\, x'.e'[\theta]$. Similarly, when applying θ to $\mathbf{fix}\, f.e$, we may need to α-rename the bound variable f.

Lemma 3.1.4 (Substitution) *Assume that $\Gamma_0, \Gamma \vdash e : T$ is derivable and $\Gamma_0 \vdash \theta : \Gamma$ holds. Then $\Gamma_0 \vdash e[\theta] : T$ is also derivable.*

Proof We proceed by induction on the height of the typing derivation $\mathcal{D}$ of $\Gamma_0, \Gamma \vdash e : T$.

- e is some variable x. Then $x \in \mathbf{dom}(\Gamma_0, \Gamma)$. If $x \in \mathbf{dom}(\Gamma)$, then $\Gamma_0 \vdash \theta : \Gamma$ implies that $\Gamma_0 \vdash e[\theta] : \Gamma(x)$ is derivable since $e[\theta] = \theta(x)$. If $x \notin \mathbf{dom}(\Gamma)$, then $x \in \mathbf{dom}(\Gamma_0)$ and thus $\Gamma_0 \vdash x : T$ is derivable.

- e is of the form $\langle e_1, e_2 \rangle$. Then $\mathcal{D}$ must be of the following form:

$$\frac{\mathcal{D}_1 :: \Gamma_0, \Gamma \vdash e_1 : T_1 \quad \mathcal{D}_2 :: \Gamma_0, \Gamma \vdash e_2 : T_2}{\Gamma_0, \Gamma \vdash e : T}$$

  where $T = T_1 * T_2$. By induction hypotheses on $\mathcal{D}_1$ and $\mathcal{D}_2$, both $\Gamma_0 \vdash e_1[\theta] : T_1$ and $\Gamma_0 \vdash e_2[\theta] : T_2$ are derivable. Hence, $\Gamma_0 \vdash \langle e_1[\theta], e_2[\theta] \rangle : T$ is derivable. Note that $e[\theta] = \langle e_1[\theta], e_2[\theta] \rangle$, and we are done.

- e is of the form $\mathbf{lam}\, x.e_1$. Then $\mathcal{D}$ must be of the following form:

$$\frac{\mathcal{D}_1 :: \Gamma_0, \Gamma, x : T_1 \vdash e_1 : T_2}{\Gamma_0, \Gamma \vdash \mathbf{lam}\, x.e_1 : T}$$

  We may assume that x is distinct from each variable in Γ. Clearly, we can construct a derivation $\mathcal{D}_1'$ of $\Gamma_0, x : T_1, \Gamma \vdash e_1 : T_2$ that is of the same height as $\mathcal{D}_1$. By induction hypothesis on $\mathcal{D}_1'$, we can derive $\Gamma_0, x : T_1 \vdash e_1[\theta] : T_2$, which leads to a derivation of $\Gamma_0 \vdash \mathbf{lam}\, x.e_1[\theta] : T_2$. Note that $e[\theta] = \mathbf{lam}\, x.e_1[\theta]$, and we are done.

The rest of the cases are omitted, which can all be handled similarly, ∎

Lemma 3.1.5 *Assume that $\Gamma \vdash e : T$ is derivable. If e is a redex and e' is a reduct of e, then $\Gamma \vdash e' : T$ is also derivable.*

Proof As an exercise. ∎

A typing judgment for assigning types to evaluation contexts in $\mathcal{L}_0$ is of the form $\Gamma \vdash E : T_0/T$, meaning that E can be assigned the type T under Γ if the hole $[]$ in E is given the type T_0. The rules for deriving such judgments are given in Figure 3.3. As these rules can be easily constructed by studying the typing rules in Figure 3.2, we may omit presenting such rules in the future.

Lemma 3.1.6 *If both $\Gamma \vdash E : T_0/T$ and $\Gamma \vdash e : T_0$ are derivable, then $\Gamma \vdash E[e] : T$ is also derivable.*

Proof As an exercise. ∎

Lemma 3.1.7 *If $\Gamma \vdash E[e] : T$ is derivable, then there exists a type T_0 such that both $\Gamma \vdash E : T_0/T$ and $\Gamma \vdash e : T_0$ are both derivable.*

Proof As an exercise. ∎

Theorem 3.1.8 (Subject Reduction) *Assume that $\Gamma \vdash e_1 : T$ is derivable and $e_1 \rightarrow e_2$ holds. Then $\Gamma \vdash e_2 : T$ is also derivable.*

Proof Assume that $e_1 = E[e]$ for some evaluation context E and redex e, and $e_2 = E[e']$ for some reduct e' of e. By Lemma 3.1.7, there exists a type T_0 such that $\Gamma \vdash E : T_0 \rightarrow T$ and $\Gamma \vdash e : T_0$ are derivable. By Lemma 3.1.5, $\Gamma \vdash e' : T_0$ is derivable, and by Lemma 3.1.6, $\Gamma \vdash e_2 : T$ is derivable since $e_2 = E[e']$. ∎

Lemma 3.1.9 *Assume that $\emptyset \vdash e : T$ is derivable. If e is not a value, then $e = E[e^r]$ for some evaluation context E and redex e^r.*

$$\frac{}{\Gamma \vdash [] : T_0/T_0} \textbf{ (tc-id)}$$

$$\frac{\vdash c : (T_1, \ldots, T_n) \to T \quad \Gamma \vdash E : T_0/T_i \quad \Gamma \vdash v_k : T_k \text{ for } 1 \leq k < i \quad \Gamma \vdash e_k : T_k \text{ for } i < k \leq n}{\Gamma \vdash c(v_1, \ldots, v_{i-1}, E, e_{i+1}, \ldots, e_n) : T_0/T} \textbf{ (tc-cst)}$$

$$\frac{\Gamma \vdash E : T_0/bool \quad \Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \textbf{if}(E, e_1, e_2) : T_0/T} \textbf{ (tc-if)}$$

$$\frac{\Gamma \vdash E : T_0/T_1 \quad \Gamma \vdash e : T_2}{\Gamma \vdash \langle E, e \rangle : T_0/T_1 * T_2} \textbf{ (tc-tup-1)}$$

$$\frac{\Gamma \vdash e : T_1 \quad \Gamma \vdash E : T_0/T_2}{\Gamma \vdash \langle e, E \rangle : T_0/T_1 * T_2} \textbf{ (tc-tup-2)}$$

$$\frac{\Gamma \vdash E : T_0/T_1 * T_2}{\Gamma \vdash \textbf{fst}(E) : T_0/T_1} \textbf{ (tc-fst)} \qquad \frac{\Gamma \vdash E : T_0/T_1 * T_2}{\Gamma \vdash \textbf{snd}(E) : T_0/T_2} \textbf{ (tc-snd)}$$

$$\frac{\Gamma \vdash e : T_1 \to T_2 \quad \Gamma \vdash E : T_0/T_1}{\Gamma \vdash \textbf{app}(e, E) : T_0/T_2} \textbf{ (tc-app)}$$

$$\frac{\Gamma \vdash E : T_0/T_1 \to T_2 \quad \Gamma \vdash e : T_1}{\Gamma \vdash \textbf{app}(E, e) : T_0/T_2} \textbf{ (tc-app)}$$

Figure 3.3: The typing rules for evaluation contexts in $\mathcal{L}_0$

Proof We proceed by structural induction on the typing derivation $\mathcal{D}$ of $\emptyset \vdash e : T$.

- The last rule applied in $\mathcal{D}$ is **(ty-cst)**. Then $\mathcal{D}$ is of the following form:

$$\frac{\vdash c : (T_1, \ldots, T_n) \to T \quad \mathcal{D}_i :: \emptyset \vdash e_i : T_i \text{ for } 1 \leq i \leq n}{\emptyset \vdash c(e_1, \ldots, e_n) : T} \textbf{ (ty-cst)}$$

  where $e = c(e_1, \ldots, e_n)$. We have two subcases.

  - There exists e_i for some $1 \leq i \leq n$ that is not a value but e_j are values for all $1 \leq j < i$. By induction hypothesis on $\mathcal{D}_i$, $e_i = E_1[e^r]$ for some evaluation context E_1 and redex e^r. Let $E = c(e_1, \ldots, e_{i-1}, E_1, e_{i+1}, \ldots, e_n)$, and we are done.
  - All e_i are values for $1 \leq i \leq n$. Since e is not a value, c must be a constant function and thus e is a redex. Let E be $[]$ and e^r be e, and we are done.

- The last rule applied in $\mathcal{D}$ is **(ty-if)**. Then $\mathcal{D}$ is of the following form:

$$\frac{\mathcal{D}_0 :: \emptyset \vdash e_0 : \mathit{bool} \quad \mathcal{D}_1 :: \emptyset \vdash e_1 : T_1 \quad \mathcal{D}_2 :: \emptyset \vdash e_2 : T_2}{\emptyset \vdash \textbf{if}(e_0, e_1, e_2) : T} \textbf{ (ty-if)}$$

  where $e = \textbf{if}(e_0, e_1, e_2)$. We have two subcases.

  - e_0 is not a value. Then by induction hypothesis on $\mathcal{D}_0$, $e_0 = E_1[e^r]$ for some evaluation context E_1 and redex e^r. Let $E = \textbf{if}(E_1, e_1, e_2)$, and we are done.
  - e_0 is a value. By Lemma 3.1.3, e_0 is either true or false. Hence, e is a redex. Let $E = []$ and e^r, and we are done.

- The last rule applied in $\mathcal{D}$ is **(ty-fst)**. Then $\mathcal{D}$ is of the following form:

$$\frac{\mathcal{D}_0 :: \emptyset \vdash e_0 : T_1 * T_2}{\emptyset \vdash \textbf{fst}(e_0) : T_1} \textbf{ (ty-fst)}$$

  where $e = \textbf{fst}(e_0)$ and $T = T_1$. We have two subcases.

  - e_0 is not a value. By induction hypothesis on $\mathcal{D}_0$, $e_0 = E_1[e^r]$ for some evaluation context E_1 and redex e^r. Let $E = \textbf{fst}(E_1)$, and we are done.
  - e_0 is a value. By Lemma 3.1.3, e_0 is of the form $\langle v_1, v_2 \rangle$. So e is a redex. Let $E = []$ and e^r, and we are done.

- The last rule applied in $\mathcal{D}$ is **(ty-snd)**. Then this case is symmetric to the previous one.

- The last rule applied in $\mathcal{D}$ is **(ty-app)**. Then $\mathcal{D}$ is of the following form:

$$\frac{\mathcal{D}_1 :: \emptyset \vdash e_1 : T_1 \to T_2 \quad \mathcal{D}_2 :: \emptyset \vdash e_2 : T_1}{\emptyset \vdash \textbf{app}(e_1, e_2) : T_2} \textbf{ (ty-app)}$$

  where $e = \textbf{app}(e_1, e_2)$ and $T = T_2$. We have a few subcases.

  - e_1 is not a value. By induction hypothesis on $\mathcal{D}_1$, $e_1 = E_1[e^r]$. Let $E = \mathbf{app}(E_1, e_2)$ and we are done.
  - e_1 is a value but e_2 is not a value. By induction hypothesis on $\mathcal{D}_2$, $e_2 = E_1[e^r]$. Let $E = \mathbf{app}(e_1, E_1)$ and we are done.
  - e_1 and e_2 are values. By Lemma 3.1.3, e_1 is of the form $\mathbf{lam}\, x.e_{10}$. Hence, e is a redex. Let $E = []$ and e^r, and we are done.

- The last rule applied in $\mathcal{D}$ is **(ty-fix)**. Then $\mathcal{D}$ is of the following form:

$$\frac{\emptyset, f : T \vdash e_0 : T}{\emptyset \vdash \mathbf{fix}\, f.e_0 : T}\ \textbf{(ty-fix)}$$

  where $e = \mathbf{fix}\, f.e_0$. Then e is a redex. Let $E = []$ and e^r, and we are done.

We conclude the proof as all the possible cases are covered. ∎

Theorem 3.1.10 (Progress) *Assume that $\emptyset \vdash e : T$ is derivable. Then either e is value or $e \to e'$ for some e'.*

Proof Assume that e is not a value. By Lemma 3.1.9, $e = E[e^r]$ for some evaluation context E and redex e^r. Hence, $e \to E[e^c]$ holds for each e^c that is a reduct of e^r. ∎

By Theorem 3.1.8 and Theorem 3.1.10, it is clear that for every well-typed (closed) expression e in $\mathcal{L}_0$, an evaluation starting from e either terminates with a value or continues forever. In particular, e can never be evaluated to a non-value expression that cannot be further evaluated. This is phrased by Robin Milner in the slogan: *a well-typed program can never go wrong*.

3.2 Pattern Matching

3.3 Simply-Typed Lambda-Calculus

The types in simply-typed lambda-calculus are given as follows, where we use δ for some base types.

$$\text{types} \quad T \quad ::= \quad \delta \mid T_1 \to T_2$$

These types are often referred to as *simply types*. For each type T, we assume the existence of a denumerable set of (typed) variables $\mathbf{x}_1^T, \mathbf{x}_2^T, \ldots$ and use x^T, y^T, z^T to ranges over this set. We may write x to mean x^T for some T.

Definition 3.3.1 *The simply-typed λ-terms are defined as follows.*

- *A typed variable x^T is a term of type T.*
- *$\lambda x^{T_1}.t$ is a term of type $T_1 \to T_2$ if t is a term of type T_2.*
- *$t_1(t_2)$ is a term of type T_2 if t_1 and t_2 are terms of types $T_1 \to T_2$ and T_1, respectively.*

We say that a term is neutral if it is a variable or an application. We may write t^T to indicate that t is a term of type T.

Definition 3.3.2 (Reducibility) *Given a term t of type T, t is reducible of type T if $\mu(t) < \infty$ and*

- *$T = \delta$ for some δ, or*
- *$T = T_1 \to T_2$ and for every term t_1 that is reducible of type T_1, $t_0[x := t_1]$ is reducible of type T_2 whenever $t \to_\beta^* \lambda x.t_0$ holds.*

Given a term of type T, we say that t is reducible if t is reducible of type T.

Clearly, the notion of being reducible of type T is well-defined based on structural induction on T. This notion is often referred to as reducibility of type T. In Definition 3.3.2, the notion of reducibility is presented in a style we refer to as *introduction-major*. This is in contrast to the definition given in (Girard, Lafont, and Taylor 1989), which is of a style we refer to as *elimination-major*.

Lemma 3.3.3 (Forward Property) *Assume that t is reducible and $t \to_\beta^* t'$. Then t' is also reducible.*

Proof The lemma follows from the defintion of reducibility immediately. ∎

Lemma 3.3.4 (Backward Property) *Assume that t is neutral. If t' is reducible whenever $t \to_\beta t'$ holds, then t is also reducible.*

Proof Assume that t is of type T. Clearly, $\mu(t)$ is finite as $\mu(t') < \infty$ whenever $t \to_\beta t'$ holds. If T is some δ, we are done. Otherwise, assume $T = T_1 \to T_2$. If $t \to_\beta^* \lambda x.t_0$, then there must be t' such that $t \to_\beta t'$ and $t' \to_\beta^* \lambda x.t_0$. Since $t's$ is reducible, $t_0[x := t_1]$ is reducible for every t_1 that is reducible of type T_1. By definition, t is reducible of type T. ∎

Corollary 3.3.5 *Every variable x^T is reducible of type T.*

Proof By Lemma 3.3.4 immediately. ∎

Lemma 3.3.6 *Let $t = \lambda x^{T_1}.t_0$. If $t_0[x := t_1]$ is reducible for every t_1 that is reducible of type T_1, then t is reducible.*

Proof Assume $t \to_\beta^* \lambda x.t_0'$. Then we have $t_0 \to_\beta^* t_0'$, which implies $t_0[x := t_1] \to_\beta^* t_0'[x := t_1]$. By Lemma 3.3.3, $t_0'[x := t_1]$ is reducible. By definition, t is reducible. ∎

Lemma 3.3.7 *Assume that t_1 and t_2 are reducible of types $T_1 \to T_2$ and T_2, respectively. Then $t_1(t_2)$ is reducible of type T_2.*

Proof By definition, $\mu(t_i) < \infty$ for $i = 1, 2$. Let $t = t_1(t_2)$, and we proceed by induction on $\mu(t_1) + \mu(t_2)$. Assume $t \to_\beta t'$, and we have the following cases.

- $t_1 \to_\beta t_1'$ and $t' = t_1'(t_2)$. By Lemma 3.3.3, t_1' is reducible. Note that $\mu(t_1') + \mu(t_2) < \mu(t_1) + \mu(t_2)$. By induction hypothesis, t' is reducible.

- $t_2 \rightarrow_\beta t_2'$ and $t' = t_1(t_2')$. This case is similar to the previous one.
- $t_1 = \lambda x.t_{10}$ and $t' = t_{10}[x := t_2]$. By definition, t_1 and t_2 being reducible implies that t' is reducible.

Therefore, t' is reducible whenever $t \rightarrow_\beta t'$ holds. By Lemma 3.3.4, t is reducible. ■

Theorem 3.3.8 *Every simply-typed λ-term t is strongly normalizing, that is, $\mu(t) < \infty$.*

Proof We say that a substution θ is reducible of it maps each variable x^T in its domain to a term reducible of type T. Assume that t is of type T_0. We prove that $t[\theta]$ is reducible of type T_0 whenever θ is reducible. The proof proceeds by structural induction on t.

- Assume $t = x$. If $x \in \mathbf{dom}(\theta)$, then $t[\theta] = \theta(x)$, which is reducible since θ is reducible. If $x \notin \mathbf{dom}(\theta)$, then $t[\theta] = x$, which is reducbile by Corollary 3.3.5.
- Assume $t = \lambda x^{T_1}.t_1$, where t_1 is a term of type T_2. We can choose $x^{T_1} \notin \mathit{vars}(\theta)$ so that $t[\theta] = \lambda x^{T_1}.t_1[\theta]$ holds. Let t_2 be a reducible term of type T_1. Note that $t_1[\theta][x^{T_1} := t_2] = t_1[\theta']$, where $\theta' = \theta[x \mapsto t_2]$ is reducible. By induction hypothesis on t_1, $t_1[\theta][x^T := t_2]$ is reducible. By Lemma 3.3.6, $t[\theta]$ is reducible.
- Assume $t = t_1(t_2)$. By induction hypothesis, $t_i[\theta]$ is reducible for $i = 1, 2$. By Lemma 3.3.7, $t_1[\theta](t_2[\theta])$ is reducible. Note $t[\theta] = t_1[\theta](t_2[\theta])$, and we are done.

As all the cases are covered, we conclude that $t[\theta]$ is reducible whenever θ is reducible. Let $\theta = []$, and we know that t is reducible. By the definition of reducibility, t is strongly normalizing. ■

3.4 Exercises

Exercise 1 *Extend the simply-typed λ-calculus with product types. Prove SN.*

Exercise 2 *Extend the simply-typed λ-calculus with sum types. Prove SN.*

Chapter 4

Polymorphic Types

4.1 A Polymorphically-Typed Programming Language

4.2 Second-Order Polymorphic Lambda-Calculus

The types in second-order polymorphic lambda-calculus λ_2, which is also referred to as System F, are given as follows, where we use α for type variables, that is, variables ranging over types.

$$\text{types} \quad T \quad ::= \quad \alpha \mid T_1 \to T_2 \mid \forall \alpha.T$$

Note that there are no base types in λ_2. For each type T, we assume the existence of a denumerable set of (typed) variables $\mathbf{x}_1^T, \mathbf{x}_2^T, \ldots$ and use x^T, y^T, z^T to ranges over this set. We may write x to mean x^T for some T.

Definition 4.2.1 *The polymorphically-typed λ-terms are defined as follows.*

- *A typed variable x^T is a term of type T.*
- *$\lambda x^{T_1}.t$ is a term of type $T_1 \to T_2$ if t is a term of type T_2.*
- *$t_1(t_2)$ is a term of type T_2 if t_1 and t_2 are terms of types $T_1 \to T_2$ and T_1, respectively.*
- *$\lambda\alpha.t$ is a term of type $\forall\alpha.T_0$ if t is a term of type T_0 and α does not have free occurrences in T for any free variable x^T in t.*

types	T	::=	$\alpha \mid \delta(T_1, \ldots, T_n)$
type schemes	$\hat{T}$	::=	$\forall\vec{\alpha}.\ \hat{T}$
typing contexts	Γ	::=	$\emptyset \mid \Gamma, x : \hat{T}$
expressions	e	::=	$\ldots \mid \textbf{let}\ x = e_1\ \textbf{in}\ e_2\ \textbf{end}$
evaluation contexts	E	::=	$\ldots \mid \textbf{let}\ x = E\ \textbf{in}\ e\ \textbf{end}$

Figure 4.1: The syntax for $\mathcal{L}_{poly}^{pred}$

- *$t(T)$ is a term of type $T_0[\alpha := T]$ if t is a term of type $\forall\alpha.T_0$.*

We say that a term t is neutral if it is of one of the following forms x^T, $t_1(t_2)$ and $t_1(T_2)$. We may write t^T to indicate that t is a term of type T.

Definition 4.2.2 (Reducibility Candidate) *A reducibility candidate of type T is a set R of terms of type T satisfying the following conditions:*

1. *$\mu(t) < \infty$ for each $t \in R$*
2. *(Forward Condition) If $t \in R$ and $t \rightarrow^*_\beta t'$, then $t' \in R$.*
3. *(Backward Condition) If t is neutral and $t' \in R$ whenever $t \rightarrow_\beta t'$ holds, then $t \in R$.*

Definition 4.2.3 *Let T be a type and Θ be a type substition such that $\mathbf{dom}(\Theta) \subseteq FV(T)$. Let $\hat{\Theta}$ be a mapping such that $\mathbf{dom}(\hat{\Theta}) = \mathbf{dom}(\Theta)$ and $\hat{\Theta}(\alpha)$ is a reducibility candidate of type $\Theta(\alpha)$ for each $\alpha \in \mathbf{dom}(\hat{\Theta})$.*

- *$T = \alpha$ for some α. Then $\mathbf{RED}(\hat{\Theta}; T) = \hat{\Theta}(\alpha)$.*
- *$T = T_1 \rightarrow T_2$. Then $\mathbf{RED}(\hat{\Theta}; T)$ is the set R such that $t \in R$ if t is of type $T[\Theta]$ and $\mu(t) < \infty$ and for every term t_1 in $\mathbf{RED}(\hat{\Theta}; T_1)$, $t_0[x := t_1] \in \mathbf{RED}(\hat{\Theta}; T_2)$ whenever $t \rightarrow^*_\beta \lambda x.t_0$ holds.*
- *$T = \forall\alpha.T_0$. Then $\mathbf{RED}(\hat{\Theta}; T)$ is the set R such that $t \in R$ if t is of type $T[\Theta]$ and $\mu(t) < \infty$ and for every type T_1 and reducibility candiate R_1 of type T_1, $t_0[\alpha := T_1] \in \mathbf{RED}(\hat{\Theta}'; T_0)$ whenever $t \rightarrow^*_\beta \lambda\alpha.t_0$ holds, where $\hat{\Theta}'$ extends $\hat{\Theta}$ with a link from α to R_1.*

Clearly, $\mathbf{RED}(\hat{\Theta}; T)$ is well-defined based on structural induction on T.

Proposition 4.2.4 *Let $T, \Theta, \hat{\Theta}$ be the same as in Definition 4.2.3. Then $\mathbf{RED}(\hat{\Theta}; T)$ is a reducibility candidate of type $T[\Theta]$.*

Proof ∎

Lemma 4.2.5 *Let $T, \Theta, \hat{\Theta}$ be the same as in Definition 4.2.3. If $T = \forall\alpha.T_0$ and $t \in \mathbf{RED}(\hat{\Theta}; T)$, then $t(T_1[\Theta]) \in \mathbf{RED}(\hat{\Theta}; T_0[\alpha := T_1])$ holds for every type T_1 satisfying $FV(T_1) \subseteq \mathbf{dom}(\Theta)$.*

Lemma 4.2.6

Chapter 5

Types with Effects

Chapter 6

Predicatization

Chapter 7

Programming with Theorem Proving

Chapter 8

Linear Types

Contents

List of Figures

Bibliography

Girard, J.-Y., Y. Lafont, and P. Taylor (1989). *Proofs and Types*, Volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge, England: Cambridge University Press.

Tait, W. W. (1967, June). Intensional Interpretations of Functionals of Finite Type I. *Journal of Symbolic Logic 32*(2), 198–212.