

Evaluation under λ -abstraction

Hongwei Xi

Department of Mathematical Sciences
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Email: hwxi+@cs.cmu.edu

Abstract

In light of the usual definition of values [15] as terms in weak head normal form (WHNF), a λ -abstraction is regarded as a value, and therefore no expressions under λ -abstraction can get evaluated and the sharing of computation under λ has to be achieved through program transformations such as λ -lifting and supercombinators. In this paper we generalise the notion of head normal form (HNF) and introduce the definition of *generalised head normal form* (GHNF). We then define values as terms in GHNF with flexible heads, and study a call-by-value λ -calculus λ_{hd}^v corresponding to this new notion of values. After establishing a version of normalisation theorem in λ_{hd}^v , we construct an evaluation function \mathbf{eval}_{hd}^v for λ_{hd}^v which evaluates under λ -abstraction. We prove that a program can be evaluated in λ_{hd}^v to a term in GHNF if and only if it can be evaluated in the usual λ -calculus to a term in HNF. We also present an operational semantics for λ_{hd}^v via a SECD machine. We argue that lazy functional programming languages can implement λ_{hd}^v and a call-by-need implementation of λ_{hd}^v can significantly enhance the degree of sharing in evaluation.

1. Introduction

SECD machines never evaluate redexes under a λ -abstraction, and this has some potential disadvantages. For instance, given a program

$$(\lambda x. + (x(0), x(1)))(\lambda y. I(I)(y)),$$

where $I = (\lambda z. z)$, the β -redex $I(I)$ gets reduced twice by either a call-by-value or a call-by-need SECD machine. This kind of duplication of reductions can be avoided by extracting the maximal-free terms in a function at compile-time [14]. However, since evaluation can change the set of free variables in a term, the situation becomes complicated if such a redex is generated at run-time. This makes direct evaluation under λ -abstraction desirable.

Why is a value defined as a λ -abstraction or a variable in the (usual) call-by-value λ -calculus λ_v [15]? One crucial observation is that this form of values is closed under *value* substitution.

This directly leads to the notion of residuals of β_v -redexes under β_v -reduction and the notion of parallel β_v -reduction, by which it follows that λ_v enjoys the Church-Rosser property and a version of standardisation theorem. An evaluation function for λ_v can then be defined and justified by the standardisation theorem in λ_v .

There exists another form of terms which is closed under substitution. A term in head normal form (HNF) is of form $\lambda x_1 \dots \lambda x_m. x(M_1) \dots (M_n)$; if x is x_i for some $1 \leq i \leq n$ then the term is in flexible HNF; flexible HNF is clearly closed under substitution.

If we define values as terms in flexible HNF, then evaluation under λ -abstraction has to be performed in order to reduce $(\lambda y. I(I)(y))$ to a value, which is $(\lambda y. y)$ in this case. Hence, call-by-value in this setting can avoid duplicating reductions. Unfortunately, the new definition of values has some serious drawbacks as illustrated in Section 4. Modifying the notion of head normal form (HNF), we present the definition of generalised head normal form (GHNF). We then define values as terms in flexible GHNF and study a call-by-value λ -calculus λ_{hd}^v based on this definition. After proving a version of normalisation theorem in λ_{hd}^v , we define an evaluation function which always evaluates a program to a term in GHNF if there exists one. Our main contribution is showing that a term can be reduced to a term in GHNF in λ -calculus λ_{hd}^v if and only if it can be reduced to a term in HNF in the usual λ -calculus λ . We also present an operational semantics for λ_{hd}^v via a SECD machine, which can easily lead to a mechanical implementation.

Lazy functional programming languages implement the (call-by-name) λ -calculus. Executions of programs aim at returning observable values such as integers and booleans in realistic programming languages. Observable values are in HNF, and therefore, lazy functional programming languages can implement the call-by-value λ -calculus λ_{hd}^v without compromising their semantics.

The next section presents some preliminaries such as head β -reductions and residuals in the λ -calculus λ . The third section introduces some proof techniques handling β -developments, which are used later to show some syntactic properties which λ_{hd}^v enjoys. The fourth section presents the definitions of generalised head normal form and β_{hd}^v -redexes. The fifth section studies λ_{hd}^v , and the sixth section studies the relations between λ_{hd}^v and λ . The seventh section presents an operational semantics of λ_{hd}^v via a SECD machine. The eighth section deals with extensions of λ_{hd}^v with recursion combinators, constructors and primitive operators. The rest of the paper discusses some related work and future directions.

2. Preliminaries

We assume a basic familiarity of the reader with the λ -calculus λ [4].

Definition 2.1 *The set Λ of λ -terms is defined inductively as follows.*

- (variable) *There are infinitely many variables u, v, x, y, z, \dots in Λ ; variables are the only subterms of themselves.*
- (abstraction) *If $M \in \Lambda$ then $(\lambda x. M) \in \Lambda$; N is a subterm of $(\lambda x. M)$ if N is $(\lambda x. M)$ or a subterm of M .*

- (application) If $M_1, M_2 \in \Lambda$ then $M_1(M_2) \in \Lambda$; N is a subterm of $M_1(M_2)$ if N is $M_1(M_2)$ or a subterm of M_i for some $i \in \{1, 2\}$.

Let Var be the set of all variables. The set $\text{FV}(M)$ of free variables in a term M is defined as follows.

$$\text{FV}(M) = \begin{cases} \{M\} & \text{if } M \in \text{Var}; \\ \text{FV}(M_1) \setminus \{x\} & \text{if } M = (\lambda x.M_1); \\ \text{FV}(M_1) \cup \text{FV}(M_2) & \text{if } M = M_1(M_2). \end{cases}$$

An occurrence of a variable x in a term M is bound if it occurs in some term M_1 where $\lambda x.M_1$ is a subterm of M ; an occurrence of a variable is free if it is not bound. A term M is called a program if $\text{FV}(M) = \emptyset$.

The notion $M\{x := N\}$ denotes the result of substituting N for all free occurrences of x in M . We assume Barendregt's variable convention to avoid name collisions and treat α -conversions implicitly. We also assume some elementary properties of substitution, e.g. the substitution lemma (Lemma 2.1.16 [4]).

Convention L, M, N range over terms in Λ ; P ranges over programs; R ranges over various redexes defined below; σ, τ range over various reduction sequences defined below, and \emptyset stands for an empty reduction sequence.

We introduce a new symbol \bullet as a place holder and treat it as a variable. The body of a λ -abstraction $M = \lambda x.M_1$ is defined as $\text{bd}(M) = M_1\{x := \bullet\}$. We may use integers in our examples, but we only study pure λ -terms until Section 8.

Definition 2.2 (β -redex and β -reduction) $M(N)$ is a β -redex if M is a λ -abstraction; $\beta(M, N) = \text{bd}(M)\{\bullet := N\}$ is the contractum of the β -redex; we write $M_1 \rightarrow_\beta M_2$ if M_2 is obtained by replacing some β -redex in M_1 with its contractum. A β -reduction sequence is a possibly infinite sequence of form $M_1 \rightarrow_\beta M_2 \rightarrow_\beta \dots$

Given two β -redexes R_1 and R_2 in a term M ; R_1 is to the left of R_2 if the first symbol of R_1 is to the left of that of R_2 .

For any decorated reduction notation of \rightarrow in this paper, the corresponding decorated reduction notations of \rightarrow^n and \twoheadrightarrow stand for n steps and some (possibly zero) steps of such a reduction, respectively.

The (usual) λ -calculus λ is a theory which studies β -reduction. $\lambda \vdash M =_\beta N$ if there exists $M = M_0, M_1, \dots, M_{2n-1}, M_{2n} = N$ such that

$$M_{2i-1} \twoheadrightarrow_\beta M_{2i-2} \text{ and } M_{2i-1} \twoheadrightarrow_\beta M_{2i}$$

for $1 \leq i \leq n$.

The following explicit notation of β -reduction sequences enables us to write out the contracted β -redexes.

Definition 2.3 (*Explicit β -reduction sequences*) Given a β -redex R in M ; $M \xrightarrow{R}_\beta N$ stands for the β -reduction step in which R gets contracted; $[R_1] + \dots + [R_n]$ stands for a reduction sequence of the following form.

$$M_0 \xrightarrow{R_1}_\beta M_1 \xrightarrow{R_2}_\beta \dots \xrightarrow{R_n}_\beta M_n.$$

Notations $\sigma : M \rightarrow_\beta N$ or $M \xrightarrow{\sigma}_\beta N$ stands for a β -reduction sequence from M to N ; for a finite β -reduction sequence σ from M , $\sigma(M)$ stands for the term to which σ reduces M ; $|\sigma|$ is the length of σ , namely, the number of β -reduction steps in σ ; $\sigma + \tau$ stands for a β -reduction sequence of form $M_0 \xrightarrow{\sigma}_\beta M_1 \xrightarrow{\tau}_\beta M_2$; $\sigma\{x := L\}$ stands for the β -reduction sequence obtained from σ by replacing every free occurrence of variable x in σ with L .

All these notations generalise to other notions of reduction defined later if applicable.

Definition 2.4 (*HNF*) Given a term M ; M has a head β -redex $(\lambda x.N_1)(N_2)$ if M is of form

$$\lambda x_1 \dots \lambda x_m. (\lambda x.N_1)(N_2)(M_1) \dots (M_n),$$

where $m, n \geq 0$. We write $M \xrightarrow{h}_\beta N$ if $M \xrightarrow{R}_\beta N$ where R is the head β -redex in M . M is in HNF if M has no head β -redex, i.e., M is of form $\lambda x_1 \dots \lambda x_m. x M_1 \dots M_n$, where $m \geq 0$, $n \geq 0$; the HNF is flexible if x occurs in x_1, \dots, x_m , and it is rigid otherwise.

Note that flexible HNF is closed under substitution; if we define values as terms in flexible HNF, then a redex in the corresponding call-by-value calculus is of form $M(N)$, where M, N are in flexible HNF; we can prove that this λ -calculus is Church-Rosser and enjoys a version of standardisation theorem; unfortunately, normal forms in this λ -calculus may seem inappropriate to be regarded as outputs of programs; for instance, $(\lambda x.I(x(I)))(I)(0)$, where $I = \lambda y.y$, is a normal form in this λ -calculus but we really wish to reduce it to 0. This obstacle will be overcome in the Section 4.

Notations \vec{x} denotes a sequence (possibly empty) of variables x_1, \dots, x_n ; $|\vec{x}| = n$ is the length of the sequence; $\lambda \vec{x}.M$ is $\lambda x_1 \dots \lambda x_n.M$; \vec{M} denotes a sequence (possibly empty) of terms M_1, \dots, M_n ; $|\vec{M}| = n$ is the length of \vec{M} ; $M(\vec{M})$ is $M(M_1) \dots (M_n)$; $\vec{M}\{x := N\} = M_1^*, \dots, M_n^*$, where $M_i^* = M_i\{x := N\}$ for $1 \leq i \leq n$.

Proposition 2.5 *We have the following properties on head reduction.*

1. $M \xrightarrow{h}_\beta^m N$ implies $M\{x := L\} \xrightarrow{h}_\beta^m N\{x := L\}$.
2. $M \xrightarrow{h}_\beta^m \lambda x.M_1$ implies $M(N) \xrightarrow{h}_\beta^{m+1} M_1\{x := N\}$.
3. Given $M \xrightarrow{h}_\beta^m \lambda x \lambda \vec{x}. x(\vec{M})$ where x is not in \vec{x} and $N \xrightarrow{h}_\beta^n \lambda \vec{y}. z(\vec{N})$ where z is not in \vec{y} ; then $M(N) \xrightarrow{h}_\beta^l \lambda \vec{x} \lambda \vec{y}. z(\vec{L})$, where $l = m + n + 1 + \max\{|\vec{y}|, |\vec{M}|\}$.

Proof (1) follows from the observation that $R\{x := L\}$ is the head redex in $M\{x := L\}$ if R is the head redex in M ; (2) follows from (1); (3) follows from (2). See [4] for details. \blacksquare

Let us introduce the concept of *residuals* of β -redexes. The rigorous definition of this notion can be found in [8]. Let \mathcal{R} be a set of β -redexes in a term M , $R = (\lambda x.M_1)M_2$ in \mathcal{R} and $M \xrightarrow{\beta} N$. This β -reduction step affects β -redexes R' in \mathcal{R} in the following ways. We assume that bound variables are chosen distinctly from free variables.

- R' is R . Then R' has no residual in N .
- R' is in M_2 . All copies of R' in $M_1\{x := M_2\}$ are residuals of R' in N .
- R' is in M_1 . Then $R'\{x := M_2\}$ in $M_1\{x := M_2\}$ is the only residual of R' in N . This is the step where we need that the form of β -redexes is closed under substitution.
- R' contains R . Then the residual of R' is the term obtained by replacing R in R' with $M_1\{x := M_2\}$.
- Otherwise, R' is not affected, and is its own residual in N .

The residual relation is *transitive*. Given a β -reduction sequence σ from M , we say that a β -redex R in M is *involved* in σ if R or one of its residuals gets contracted in σ .

3. β -developments

In this section, we present some proof techniques handling β -developments. More details can be found in [17]. We will use these techniques to study β_{hd}^v -reduction defined in the next section.

Notations Let $M[\bullet, \dots, \bullet]$ be a representation of M in which all occurrences of \bullet in M have been enumerated from left to right in $[\bullet, \dots, \bullet]$. If there exist n occurrences of \bullet in $M[\bullet, \dots, \bullet]$, then $M[M_1, \dots, M_n]$ stands for the term obtained from substituting M_i for the i th occurrence of \bullet in $M[\bullet, \dots, \bullet]$ for $i = 1, \dots, n$. Given a context $C[\]$ and a β -reduction sequence $\sigma : M \xrightarrow{\beta} N$, we denote by $C[\sigma]$ the corresponding β -reduction sequence from $C[M]$ to $C[N]$. We often use σ to stand for $C[\sigma]$ if this causes no confusion. A consequence of this is as follows. Given $\sigma : M \xrightarrow{\beta} M_1 = C[N]$ and $\tau : N \xrightarrow{\beta} N_1$, $\sigma + C[\tau]$ is often denoted by $\sigma + \tau$.

Definition 3.1 (*β -development*) Given a term M and a set \mathcal{R} of β -redexes in M ; if $\sigma : M \xrightarrow{\beta} N$ contracts only the β -redexes in \mathcal{R} or their residuals, then σ is a β -development (of \mathcal{R}).

Lemma 3.2 (*Separation*) Given a β -redex $M = M_1(M_2)$; for each β -development σ from M in which M is involved, $\sigma(M)$ is of form

$$\sigma_1(\text{bd}(M_1))[\sigma_{21}(M_2), \dots, \sigma_{2n}(M_2)],$$

where σ_1 is a development from $\text{bd}(M_1)$ and σ_{2i} are developments from M_2 for $i = 1, \dots, n$.

Proof This can be proven by an induction on $|\sigma|$, yielding a construction of $\sigma_1, \sigma_{21}, \dots, \sigma_{2n}$. A detailed proof can be found in [17]. Let $\mathbf{sp}(\sigma)$ stand for

$$[M] + \sigma_1\{\bullet := N\} + \sigma_{21} + \dots + \sigma_{2n},$$

and we can easily verify that for every R in M if R is involved in $\mathbf{sp}(\sigma)$ then R is involved in σ . ■

The *finiteness of β -developments* theorem (FD), which states that all β -developments are finite, can be readily proven using the separation lemma, and such a proof is essentially the same as the one given by Hindley [7].

Definition 3..3 (*Canonical and Standard β -developments*) Given a β -development

$$\sigma : M_0 \xrightarrow{R_1} \beta M_1 \xrightarrow{R_2} \beta \cdots \xrightarrow{R_n} \beta M_n;$$

σ is canonical if there exist no $1 \leq i < j \leq n$ such that R_j is a residual of some β -redex R containing R_i ; σ is standard if there exist no $1 \leq i < j \leq n$ such that R_j is a residual of some β -redex R which is to the left of R_i .

In other words, a β -development is canonical if it contains no *inside-out* β -reductions. Clearly, a standard β -development is canonical but a canonical β -development may not be standard. Let $\sigma = [R_1] + \cdots + [R_n]$ be a canonical β -development of \mathcal{R} , and R be the leftmost $R \in \mathcal{R}$ such that R or one of its residuals is some R_j ; it can be readily verified that R_i are disjoint from R_j for $1 \leq i < j$; therefore, we can rearrange σ into

$$[R_j] + [R_1] + \cdots + [R_{j-1}] + [R_{j+1}] + \cdots + [R_n].$$

In this way, we can rearrange any canonical β -development into a standard β -development corresponding to it. Note that for every R , R is involved in a canonical β -development if and only if R is involved in its corresponding standard β -development.

Lemma 3..4 (*Standardisation of β -developments*) For every β -development $M \xrightarrow{\sigma} \beta N$, there exists a standard development

$$\mathbf{std}(\sigma) : M \rightarrow \beta N$$

such that for every redex R in M if R is involved in $\mathbf{std}(\sigma)$ then R is involved in σ .

Proof Let us proceed by a structural induction on M to show that there exists a canonical development $\mathbf{cd}(\sigma) : M \rightarrow \beta N$ for every $\sigma : M \rightarrow \beta N$ and R is involved in σ if R is involved in $\mathbf{cd}(\sigma)$. Then we can rearrange $\mathbf{cd}(\sigma)$ into its corresponding standard β -development $\mathbf{std}(\sigma)$.

- M is a variable. Then $\sigma = \emptyset$ is canonical.
- $M = (\lambda x.M_1)$. This case follows from the induction hypothesis on M_1 .
- $M = M_1(M_2)$, where M is not a β -redex. Then we can assume σ to be of form $\sigma_1 + \sigma_2$, where σ_i are developments from M_i for $i = 1, 2$. Hence $\mathbf{cd}(\sigma) = \mathbf{cd}(\sigma_1) + \mathbf{cd}(\sigma_2)$ is a canonical development reducing M to N . Assume that R is involved in $\mathbf{cd}(\sigma)$; then R is involved in $\mathbf{cd}(\sigma_i)$ for some $i \in \{1, 2\}$; this implies R is involved in σ_i by induction hypothesis, and therefore R is involved in σ .

- $M = M_1(M_2)$, where M is a β -redex. If M has a residual in N , then this case is the same as the previous one. Hence, let us assume that M is involved in σ . By Lemma 3..2, $\mathbf{sp}(\sigma)$ is of form

$$[M] + \sigma_1\{\bullet := M_1\} + \sigma_{21} + \cdots + \sigma_{2n}$$

where σ_1 is a development from $\text{bd}(M_1)$ and σ_{2i} are developments from M_2 . Note that for every R , R is involved in σ if R is involved in $\mathbf{sp}(\sigma)$. Let us define $\mathbf{cd}(\sigma)$ as

$$[M] + \mathbf{cd}(\sigma_1)\{\bullet := M_1\} + \mathbf{cd}(\sigma_{21}) + \cdots + \mathbf{cd}(\sigma_{2n}).$$

It is a routine verification that $\mathbf{cd}(\sigma)$ is canonical since $\mathbf{cd}(\sigma_1)$ and $\mathbf{cd}(\sigma_{21}), \dots, \mathbf{cd}(\sigma_{2n})$ are canonical. Assume that R is involved in $\mathbf{cd}(\sigma)$; if R is M then R is involved in $\mathbf{sp}(\sigma)$; if R is involved in $\mathbf{cd}(\sigma_1)\{\bullet := M_1\}$ then R is involved in $\sigma_1\{\bullet := M_1\}$ by induction hypothesis; if R is involved in $\mathbf{cd}(\sigma_{2i})$ for some $1 \leq i \leq n$, then R is involved in σ_{2i} by induction hypothesis; therefore, R is always involved in $\mathbf{sp}(\sigma)$, and this implies that R is involved in σ . \blacksquare

Given a β -development σ , Lemma 3..4 enables us to define a measure $|\mathbf{std}(\sigma)|$ for σ . We will see that this measure can often be used in inductive proofs, obviating the need of FD. One potential advantage of doing so is that proofs without using FD are usually easier to be mechanised in certain theorem proving systems or logical frameworks.

4. The generalised head normal form

In this section we first present the definition of GHNF; We then prove a few important properties on GHNF and introduce β_{hd}^v -redexes and their residuals under β_{hd}^v -reduction.

In the previous example, we cannot reduce the term $(\lambda x.I(x(I)))(I)(0)$ in the call-by-value calculus where values are defined as terms in flexible HNF. This problem can be resolved if we treat $I(x(I))$ as a term with head x . Then $(\lambda x.I(x(I)))$ has a flexible head, and $(\lambda x.I(x(I)))(I)(0)$ can be reduced to 0, which is of the form we expect.

Definition 4..1 (*Head and GHNF*) *The (general) heads of terms are defined below. Let $\text{Num} = \{0, 1, 2, \dots\}$.*

$$\begin{aligned} \text{hd}(M) &= M && \text{if } M \in \text{Var}; \\ \text{hd}(\lambda x.M) &= \text{hd}(M) && \text{if } \text{hd}(M) \neq x; \\ \text{hd}(\lambda x.M) &= 0 && \text{if } \text{hd}(M) = x; \\ \text{hd}(\lambda x.M) &= n + 1 && \text{if } \text{hd}(M) = n; \\ \text{hd}(\lambda x.M) &= \emptyset && \text{if } \text{hd}(M) = \emptyset; \\ \text{hd}(M(N)) &= \text{hd}(M) && \text{if } \text{hd}(M) \in \text{Var}; \\ \text{hd}(M(N)) &= \text{hd}(N) && \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) \in \text{Var}; \\ \text{hd}(M(N)) &= \emptyset && \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) \in \text{Num}; \\ \text{hd}(M(N)) &= \emptyset && \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) = \emptyset; \\ \text{hd}(M(N)) &= n - 1 && \text{if } \text{hd}(M) = n > 0; \\ \text{hd}(M(N)) &= \emptyset && \text{if } \text{hd}(M) = \emptyset. \end{aligned}$$

Given a term M ; M is in GHNF if $\text{hd}(M) \neq \emptyset$; M is rigid if $\text{hd}(M)$ is a variable; M is flexible if $\text{hd}(M)$ is a number; M is indeterminate if $\text{hd}(M) = \emptyset$; we say that M has a general head $\text{hd}(N)$ if $M \twoheadrightarrow_{\beta_{hd}^v} N$ and $\text{hd}(N) \neq \emptyset$.

It can be readily verified that $\text{hd}(M)$ is well-defined on every term $M \in \Lambda$. Therefore, GHNF is well-defined. If M is in HNF then M is in GHNF. The term $M = \lambda x.I(x(I))$ is in GHNF since $\text{hd}(M) = 0$, but M is not in HNF.

The notion of residuals can play a key rôle in proofs of Church-Rosser theorem and standardisation theorem. Note that λ -abstractions are closed under substitutions. This naturally yields the definitions of residuals in the call-by-value λ -calculus λ_v [15] in which λ -abstractions are regarded as values. Fortunately, flexible GHNF is also closed under substitution as proven below, and therefore, is suitable for defining values in this respect.

Proposition 4..2 *Given a term M ; if $\text{hd}(M) \neq x$ then $\text{hd}(M\{x := N\}) = \text{hd}(M)$ for every term N ; if $\text{hd}(M) = x$ then $\text{hd}(M\{x := N\}) = \text{hd}(N)$ if $\text{hd}(N) \in \text{Var}$.*

Proof The proposition follows from a structural induction on M . ■

Therefore, we have the following corollary stating that flexible GHNF is closed under substitution.

Corollary 4..3 *Given terms M, N ; if $\text{hd}(M) \geq 0$ then $\text{hd}(M\{x := N\}) = \text{hd}(M)$.*

The following gives some justification on why a term in GHNF need not be reduced further.

Proposition 4..4 *Given a term M in GHNF and $M \rightarrow_{\beta} N$; then N is in GHNF and $\text{hd}(M) = \text{hd}(N)$.*

Proof It suffices to show $\text{hd}(M) = \text{hd}(N)$ when $M \xrightarrow{R}_{\beta} N$ for some R . Let us proceed by a structural induction on M .

- $M = \lambda x.M_1$. Then M_1 is in GHNF, and $M_1 \xrightarrow{R}_{\beta} N_1$, where $N = \lambda x.N_1$. By induction hypothesis, $\text{hd}(N_1) = \text{hd}(M_1)$. Hence $\text{hd}(N) = \text{hd}(M)$.
- $M = M_1(M_2)$. If R is in M_i for some $i \in \{1, 2\}$, then the case simply follows induction hypothesis. Now we assume R to be M . Then $M_1 = \lambda x.M_{11}$ and $N = M_{11}\{x := M_2\}$; $\text{hd}(M_{11}) \neq \emptyset$ since M is in GHNF; if $x \neq \text{hd}(M_{11}) \in \text{Var}$ then $\text{hd}(M_1) = \text{hd}(M_{11}) = \text{hd}(N)$ by Proposition 4..2; if $x = \text{hd}(M_{11})$ then M in GHNF implies $\text{hd}(M_2) \in \text{Var}$ and $\text{hd}(M) = \text{hd}(M_2) = \text{hd}(N)$ by Proposition 4..2; if $\text{hd}(M_{11}) = n \geq 0$ then $\text{hd}(M) = n = \text{hd}(N)$ by Proposition 4..2. Therefore, $\text{hd}(M) = \text{hd}(N)$ in this case. ■

Proposition 4..5 *Given a term M in GHNF; then $M \xrightarrow{h}_{\beta} \text{hnf}(M)$ for some term $\text{hnf}(M)$; if $\text{hd}(M) = x$ then $\text{hnf}(M)$ is of form $\lambda \vec{x}.x(\vec{N})$, where x does not occur in \vec{x} ; if $\text{hd}(M) = n$ then $\text{hnf}(M)$ is of form $\lambda \vec{x}\lambda x\lambda \vec{y}.x(\vec{N})$, where $|\vec{x}| = n$.*

Proof Given a term M , we proceed by a structural induction to show the construction of $\text{hnf}(M)$.

- M is a variable. Then $\text{hnf}(M) = M$.
- $M = \lambda x.M_1$. Then $\text{hnf}(M) = \lambda x.\text{hnf}(M_1)$. Since $M_1 \xrightarrow{\text{h}_\beta} \text{hnf}(M_1)$ by induction hypothesis, $M \xrightarrow{\text{h}_\beta} \text{hnf}(M)$.
- $M = M_1(M_2)$. Then we have three subcases.
 - $\text{hd}(M_1) \in \text{Var}$. Then by induction hypothesis $\text{hnf}(M_1)$ is of form $\lambda \vec{x}.x(\vec{M})$, where x does not occur in \vec{x} ; if $|\vec{x}| = 0$ then let $\text{hnf}(M) = \text{hnf}(M_1)M_2$; if $|\vec{x}| > 0$ then let $\text{hnf}(M) = \lambda \vec{y}.x(\vec{M}^*)$, where $\vec{x} = x_1\vec{y}$ and $\vec{M}^* = \vec{M}\{x_1 := M_2\}$.
 - $\text{hd}(M_1) \geq 1$. Then by induction hypothesis $\text{hnf}(M_1)$ is of form $\lambda x_1\lambda \vec{x}\lambda x\lambda \vec{y}.x(\vec{M})$. Let $\text{hnf}(M) = \lambda \vec{x}\lambda x\lambda \vec{y}.x(\vec{M}^*)$, where $\vec{M}^* = \vec{M}\{x_1 := M_2\}$.
 - $\text{hd}(M_1) = 0$ and $\text{hd}(M_2) \in \text{Var}$. Then by induction hypothesis $\text{hnf}(M_1)$ is of form $\lambda x\lambda \vec{x}.x(\vec{M})$ and $\text{hnf}(M_2)$ is of form $\lambda \vec{y}.y\vec{N}$, where y is not in \vec{y} . Then $\text{hnf}(M)$ can be defined according to Proposition 2.5 (3).

With Proposition 2.5, it can be readily verified that the definition of $\text{hnf}(M)$ in every subcase satisfies the needs. \blacksquare

Let values be defined as terms in flexible GHNF; given terms $I = \lambda u.u$ and $M = (\lambda x.\lambda y.Iyx)N$; $\text{hd}(M) = 0$ and $\text{hd}(MI) = \emptyset$; since MI is indeterminate, we need to reduce it to a term in GHNF; M should not be reduced since it is a value; hence, we reduce MI to $(\lambda x.IIx)N$; this leads to the following definition.

Definition 4..6 (*General Body*) Let function gbd be defined on λ -terms M with $\text{hd}(M) \geq 0$.

$$\begin{aligned} \text{gbd}(M_1(M_2)) &= \text{gbd}(M_1)M_2 \\ \text{gbd}(\lambda x.M) &= M\{x := \bullet\} \quad \text{if } \text{hd}(M) = x; \\ \text{gbd}(\lambda x.M) &= \lambda x.\text{gbd}(M) \quad \text{if } \text{hd}(M) \in \text{Num}; \end{aligned}$$

For example, $\text{gbd}(\lambda x.\lambda y.Iyx) = \lambda x.I\bullet x$. It is a routine verification that $\text{gbd}(M)$ is well-defined if $\text{hd}(M) \geq 0$.

Definition 4..7 (β_{hd}^v -redex) A β_{hd}^v -redex is a term of form $M(N)$ where $0 = \text{hd}(M) \leq \text{hd}(N)$, and $\beta_{\text{hd}}^v(M, N) = \text{gbd}(M)\{\bullet := N\}$ is the contractum of the β_{hd}^v -redex; we write $M_1 \rightarrow_{\beta_{\text{hd}}^v} M_2$ if M_2 is obtained by replacing a β_{hd}^v -redex in M_1 with its contractum. A β_{hd}^v -reduction sequence is a possibly infinite sequence of form $M_1 \rightarrow_{\beta_{\text{hd}}^v} M_2 \rightarrow_{\beta_{\text{hd}}^v} \dots$

The explicit notation of β_{hd}^v -reduction sequences can be defined accordingly.

Proposition 4..8 Given a term M in GHNF and $M \rightarrow_{\beta_{\text{hd}}^v} N$; then N in GHNF and $\text{hd}(M) = \text{hd}(N)$.

Proof This follows from a structural induction on M . \blacksquare

Hence, this partially justifies why a term in GHNF can be regarded as a sort of head normal form under β_{hd}^v -reduction.

Proposition 4..9 *Given M with $\text{hd}(M) \geq 0$, and $N = \text{hnf}(M)$; then $ML \twoheadrightarrow_{\beta} \beta(N, L)$ for every term L .*

Proof By Proposition 4..5, $M \twoheadrightarrow_{\beta} N$ and NL is a β -redex. Hence, $ML \twoheadrightarrow_{\beta} \beta(N, L)$. ■

Proposition 4..10 *Given M with $\text{hd}(M) \geq 0$, and $N = \text{hnf}(M)$; then $\beta_{\text{hd}}^v(M, L) \twoheadrightarrow_{\beta} \beta(N, L)$ for every L with $\text{hd}(L) \geq 0$.*

Proof This follows from a structural induction on M . We need $\text{hd}(L) \geq 0$ since $\beta_{\text{hd}}^v(M, L)$ is undefined otherwise. ■

Corollary 4..11 *Given a term L ; if $L \xrightarrow{R}_{\beta_{\text{hd}}^v} L_1$ then $L \twoheadrightarrow_{\beta} N$ and $L_1 \twoheadrightarrow_{\beta} N$ for some N .*

Proof Note that R is of form $M_1(M_2)$, where $\text{hd}(M_1) = 0$ and $\text{hd}(M_2) \geq 0$. Hence, this follows from Proposition 4..9 and Proposition 4..10. ■

Given a term M with $\text{hd}(M) \geq 0$ and a β_{hd}^v -redex R in M ; a subterm $\text{img}(R; M)$ in $\text{gbd}(M)$ is regarded as the image of R in $\text{gbd}(M)$; if $M = \lambda x.M_1$ and $\text{hd}(M_1) = x$ then $\text{img}(R; M)$ is $R\{x := \bullet\}$; if $M = \lambda x.M_1$ and $\text{hd}(M_1) \geq 0$ then $\text{img}(R; M)$ is $\text{img}(R; M_1)$; if $M = M_1(M_2)$ and R is in M_1 then $\text{img}(R; M)$ is $\text{img}(R; M_1)$; if $M = M_1(M_2)$ and R is in M_2 then $\text{img}(R; M)$ is R . Clearly, $\text{img}(R)$ is a β_{hd}^v -redex.

Now we are ready to introduce the concept of residuals of β_{hd}^v -redexes under β_{hd}^v -reduction. Let \mathcal{R} be a set of β_{hd}^v -redexes in a term M , $R = M_1(M_2) \in \mathcal{R}$ and $M \xrightarrow{R}_{\beta_{\text{hd}}^v} N$. This β_{hd}^v -reduction step affects β_{hd}^v -redexes R' in \mathcal{R} in the following ways.

- R' is R . Then R' has no residual in N .
- R' is in M_2 . Then all copies of R' in the contractum $\text{gbd}(M_1)\{\bullet := M_2\}$ of R are residuals of R' .
- R' is in M_1 . Let R'' be the image of R' in $\text{gbd}(M_1)$, then $R''\{\bullet := M_2\}$ is the only residual of R' . This residual is a β_{hd}^v -redex by Corollary 4..3.
- R' contains R . Then $R' \xrightarrow{R}_{\beta_{\text{hd}}^v} R''$ for some R'' in N , and R'' is the only residual of R in N . Proposition 4..8 implies that R'' is a β_{hd}^v -redex.
- R' is disjoint with R . Then R' is its own residual in N .

Note that we can also define β_{hd}^v -redexes as terms of form $M(N)$, where $0 = \text{hd}(M) \leq \text{hd}(N)$, or $0 = \text{hd}(M)$ and N is a variable. The obvious reason is that such a form is closed under *value* substitution if a value is defined as a term in flexible GHNF or a variable. One disadvantage of adopting this definition is that the notion of residuals of β_{hd}^v -redexes under β -reduction is difficult to define. Besides, we will clearly see that if we can reduce a program to a GHNF via such an extended notion of β_{hd}^v -reduction then we can always do so without reducing any redex of form $M(N)$, where $\text{hd}(M) = 0$ and N is a variable.

5. The λ -calculus λ_{hd}^v

In this section we present a call-by-value λ -calculus λ_{hd}^v in which values are defined as terms in flexible GHNF. We show that λ_{hd}^v is Church-Rosser and enjoys a version of normalisation theorem and a version of standardisation theorem.

λ_{hd}^v studies β_{hd}^v -reduction. $\lambda_{hd}^v \vdash M =_{\beta_{hd}^v} N$ if there exists $M = M_0, M_1, \dots, M_{2n-1}, M_{2n} = N$ such that $M_{2i-1} \rightarrow_{\beta_{hd}^v} M_{2i-2}$ and $M_{2i-1} \rightarrow_{\beta_{hd}^v} M_{2i}$ for $1 \leq i \leq n$.

Definition 5.1 (*β_{hd}^v -development*) *Given a term M and a set \mathcal{R} of β_{hd}^v -redexes in M ; if $M \xrightarrow{\sigma}_{\beta_{hd}^v} N$ contracts only the β_{hd}^v -redexes in \mathcal{R} or their residuals, then σ is a β_{hd}^v -development (of \mathcal{R}).*

Like β -development, β_{hd}^v -development enjoys the following property.

Lemma 5.2 (*Separation*) *Given a β_{hd}^v -redex $M = M_1 M_2$; for every β_{hd}^v -development σ from M in which M is involved, $\sigma(M)$ is of form*

$$\sigma_1(\text{gbd}(M_1))[\sigma_{21}(M_2), \dots, \sigma_{2n}(M_2)],$$

where σ_1 is a β_{hd}^v -development from $\text{gbd}(M_1)$ and σ_{2i} are β_{hd}^v -developments from M_2 for $i = 1, \dots, n$; let $\mathbf{sp}(\sigma)$ stand for

$$[M] + \sigma_1\{\bullet := N\} + \sigma_{21} + \dots + \sigma_{2n},$$

then for every R if R is involved in $\mathbf{sp}(\sigma)$ then R is involved in σ .

Proof See the proof of Lemma 3.2. ■

The next lemma explains why λ_{hd}^v enjoys the Church-Rosser property.

Lemma 5.3 *Given β_{hd}^v -developments*

$$M \xrightarrow{\sigma_1}_{\beta_{hd}^v} M_1 \quad \text{and} \quad M \xrightarrow{\sigma_2}_{\beta_{hd}^v} M_2;$$

then there exist β_{hd}^v -developments τ_1 and τ_2 such that $(\sigma_1 + \tau_1)(M) = (\sigma_2 + \tau_2)(M)$.

Proof The proof proceeds by a structural induction on M .

- M is a variable. This is trivial.
- $M = \lambda x.M^1$. This case follows from the induction hypothesis on M^1 .
- $M = M^1(M^2)$, where M is not a β_{hd}^v -redex. For $i = 1, 2$, we can assume $\sigma_i = \sigma_1^i + \sigma_2^i$, where σ_1^i are β_{hd}^v -developments from M^1 and σ_2^i are β_{hd}^v -developments from M^2 . By induction hypothesis, there exist τ_1^i and τ_2^i for $i = 1, 2$ such that $(\sigma_1^1 + \tau_1^1)(M^1) = (\sigma_1^2 + \tau_1^2)(M^1)$ and $(\sigma_2^1 + \tau_2^1)(M^2) = (\sigma_2^2 + \tau_2^2)(M^2)$. Let $\tau_i = \tau_1^i + \tau_2^i$ for $i = 1, 2$, and we are done.

- $M = M^1(M^2)$, where M is a β_{hd}^v -redex. For $i = 1, 2$, we may assume that β_{hd}^v -redex M is involved in σ_i since we can simply reduce its residual in $\sigma_i(M)$ otherwise. By Lemma 5.2, $\sigma_i(M)$ are of form $\sigma_1^i(\text{gbd}(M^1))[\sigma_{21}^i(M^2), \dots, \sigma_{2n_i}^i(M^2)]$ for $i = 1, 2$, where σ_1^i are developments from $\text{gbd}(M^1)$ and $\sigma_{21}^i, \dots, \sigma_{2n_i}^i$ are developments from M^2 . By the induction hypothesis on M^2 , there exist $\tau_{21}^i, \dots, \tau_{2n_i}^i$ for $i = 1, 2$ such that $(\sigma_{2j}^i + \tau_{2j}^i) = N_2$ for $1 \leq j \leq n_i$, where N_2 is some term. By the induction hypothesis on $\text{gbd}(M^1)$, there exist τ_1^i for $i = 1, 2$ such that $(\sigma_1^i + \tau_1^i)(\text{gbd}(M^1)) = (\sigma_1^2 + \tau_1^2)(\text{gbd}(M^1)) = N_1$, where N_1 is some term. Let

$$\tau_i = \tau_{21}^i + \dots + \tau_{2n_i}^i + \tau_1^i\{\bullet := N_2\}$$

for $i = 1, 2$, then

$$(\sigma_1 + \tau_1)(M) = N_1\{\bullet := N_2\} = (\sigma_2 + \tau_2)(M).$$

Therefore, β_{hd}^v -development enjoys Church-Rosser property. ■

We say that a β_{hd}^v -development $\sigma : M \twoheadrightarrow_{\beta_{hd}^v} N$ of \mathcal{R} is complete if N contains no residuals of any β_{hd}^v -redexes in \mathcal{R} . For those who know the notion of parallel reduction, it is easy to observe that a parallel β_{hd}^v -reduction is always a complete β_{hd}^v -development. A version of Lemma 5.3 in which β_{hd}^v -developments are replaced with parallel reductions can also be proven using an argument due to Tait/Martin-Löf [4], which is also the main proof strategy in [15].

Theorem 5.4 (*Church-Rosser*) *Given $M \twoheadrightarrow_{\beta_{hd}^v} M_1$ and $M \twoheadrightarrow_{\beta_{hd}^v} M_2$; then there exists N such that $M_1 \twoheadrightarrow_{\beta_{hd}^v} N$ and $M_2 \twoheadrightarrow_{\beta_{hd}^v} N$.*

Proof This is a corollary of Lemma 5.3 since $\twoheadrightarrow_{\beta_{hd}^v}$ is a transitive closure of β_{hd}^v -developments. ■

Corollary 5.5 *Given $M \twoheadrightarrow_{\beta_{hd}^v} M_i$ for $i = 1, 2$; if M_1 and M_2 are in GHNF then $\text{hd}(M_1) = \text{hd}(M_2)$.*

Proof By Theorem 5.4, there exists N such that $M_1 \twoheadrightarrow_{\beta_{hd}^v} N$ and $M_2 \twoheadrightarrow_{\beta_{hd}^v} N$. For $i = 1, 2$, $\text{hd}(M_i) = \text{hd}(N)$ by Proposition 4.8 since $\text{hd}(M_i) \neq \emptyset$. Therefore, $\text{hd}(M_1) = \text{hd}(M_2)$. ■

We say M has a general head $\text{hd}(N)$ if $M \twoheadrightarrow_{\beta_{hd}^v} N$ for some N in GHNF, By Corollary 5.5, a term has at most one general head. Clearly, $(\lambda x.x(x))(\lambda x.x(x))$ has no general head.

We need the following definition to prove in λ_{hd}^v a version of normalisation theorem and a version of standardisation theorem. If one is only interested in the former, a proper definition of evaluation context would suffice.

Definition 5.6 *Let $\mathcal{R}_{hd}^v(M)$ be the set of all β_{hd}^v -redexes in M for every term M ; a relation*

$\triangleleft_{hd}^v(M)$ on $\mathcal{R}_{hd}^v(M)$ is defined as follows.

$$\begin{aligned}
\triangleleft_{hd}^v(M) &= \emptyset && \text{if } M \text{ is a variable;} \\
\triangleleft_{hd}^v(\lambda x.M) &= \triangleleft_{hd}^v(M) && ; \\
\triangleleft_{hd}^v(M(N)) &= \triangleleft_{hd}^v(M) \cup \triangleleft_{hd}^v(N) \cup (\mathcal{R}_{hd}^v(N) \times \mathcal{R}_{hd}^v(M)) \\
&\quad \cup \{ \langle M(N), L \rangle : L \in \mathcal{R}_{hd}^v(M) \cup \mathcal{R}_{hd}^v(N) \} && \text{if } M(N) \text{ is a } \beta_{hd}^v\text{-redex;} \\
\triangleleft_{hd}^v(M(N)) &= \triangleleft_{hd}^v(M) \cup \triangleleft_{hd}^v(N) \cup (\mathcal{R}_{hd}^v(N) \times \mathcal{R}_{hd}^v(M)) && \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) \notin \text{Num;} \\
\triangleleft_{hd}^v(M(N)) &= \triangleleft_{hd}^v(M) \cup \triangleleft_{hd}^v(N) \cup (\mathcal{R}_{hd}^v(M) \times \mathcal{R}_{hd}^v(N)) && \text{if } \text{hd}(M) \neq 0.
\end{aligned}$$

$\triangleleft_{hd}^v(M)$ is linear for every term M . We write $M \triangleleft_{hd}^v N$ in L if $\langle M, N \rangle \in \triangleleft_{hd}^v(L)$; we often leave L out if this causes no confusion; we say $R \in \mathcal{R}_{hd}^v(M)$ is the \triangleleft_{hd}^v -first β_{hd}^v -redex in M satisfying some property if R in M satisfies the property and there exists no $R' \triangleleft_{hd}^v R$ in M satisfying the same property.

Definition 5.7 Given a β_{hd}^v -development σ

$$M_0 \xrightarrow{R_1}_{\beta_{hd}^v} M_1 \xrightarrow{R_2}_{\beta_{hd}^v} \dots \xrightarrow{R_n}_{\beta_{hd}^v} M_n;$$

σ is canonical if there exist no $1 \leq i < j \leq n$ such that R_j is a residual of some β_{hd}^v -redex R containing R_i ; σ is standard if there exist no $1 \leq i < j \leq n$ such that R_j is a residual of some β_{hd}^v -redex R with $R \triangleleft_{hd}^v R_i$ in M_{i-1} .

Like the case of β -development, we can readily verify that a canonical β_{hd}^v -development can also be rearranged into a corresponding standard β_{hd}^v -developments.

Lemma 5.8 (Standardisation of β_{hd}^v -developments) There exists a standard development

$$\mathbf{std}_{hd}^v(\sigma) : M \twoheadrightarrow_{\beta_{hd}^v} N$$

for every β_{hd}^v -development $\sigma : M \twoheadrightarrow_{\beta_{hd}^v} N$.

Proof This lemma can be proven in the same way as Lemma 3.4 is proven. ■

Now we are ready to establish a version of normalisation theorem in λ_{hd}^v ; we prove that the strategy is normalising which always contracts the \triangleleft_{hd}^v -first β_{hd}^v -redexes in terms; we then define an evaluation function which always evaluates a term M to a term in GHNF if $\lambda_{hd}^v \vdash M =_{\beta_{hd}^v} N$ for some N in GHNF.

If M is indeterminate then we call its \triangleleft_{hd}^v -first β_{hd}^v -redexes the *main* β_{hd}^v -redex in M . We present an equivalent definition of main β_{hd}^v -redexes.

Definition 5.9 Given M with $\text{hd}(M) = \emptyset$; the main β_{hd}^v -redex $R_{hd}^v(M)$ in M is defined as follows.

$$\begin{aligned}
R_{hd}^v(M) &= M && \text{if } M \text{ is a } \beta_{hd}^v\text{-redex;} \\
R_{hd}^v(M) &= R_{hd}^v(M_1) && \text{if } M = \lambda x.M_1 \text{ and } \text{hd}(M_1) = \emptyset; \\
R_{hd}^v(M) &= R_{hd}^v(M_1) && \text{if } M = M_1 M_2 \text{ and } \text{hd}(M_1) = \emptyset; \\
R_{hd}^v(M) &= R_{hd}^v(M_2) && \text{if } M = M_1 M_2 \text{ and } \text{hd}(M_1) = 0 \text{ and } \text{hd}(M_2) = \emptyset;
\end{aligned}$$

Clearly, $R_{hd}^v(M)$ is well-defined on every indeterminate term M . It is a routine verification that $R_{hd}^v(M)$ is the \triangleleft_{hd}^v -first in M .

Proposition 5..10 *Given M with $hd(M) = \emptyset$, $R = R_{hd}^v(M)$ and $M \xrightarrow{R_1} \beta_{hd}^v N$ for some $R_1 \neq R$; then $hd(N) = \emptyset$ and R has only one residual in N which is $R_{hd}^v(N)$.*

Proof This follows from a structural induction on M . ■

Definition 5..11 *If $M \xrightarrow{R} \beta_{hd}^v N$ and $R = R_{hd}^v(M)$ then we write $M \xrightarrow{m} \beta_{hd}^v MN$. A β_{hd}^v -normalising sequence is a possibly infinite sequence of the following form.*

$$M = M_0 \xrightarrow{m} \beta_{hd}^v M_1 \xrightarrow{m} \beta_{hd}^v \cdots$$

Let $\nu(M)$ denote the longest β_{hd}^v -normalising sequence from M , which can be of infinite length.

Clearly, if $\nu(M)$ is finite then $\nu(M) : M \xrightarrow{m} \beta_{hd}^v N$ terminates with some N in GHNF.

Lemma 5..12 *Given a β_{hd}^v -development $M \xrightarrow{\sigma} \beta_{hd}^v N$; if $\nu(N)$ is finite then $\nu(M)$ is also finite and $|\nu(N)| \leq |\nu(M)|$.*

Proof The proof proceeds by an induction on $\langle |\nu(N)|, |\sigma| \rangle$, lexicographically ordered. If M is in GHNF then $\nu(M) = \emptyset$. Now we assume $hd(M) = \emptyset$ and $M \xrightarrow{R} \beta_{hd}^v M_1$ where $R = R_{hd}^v(M)$.

- R is involved in σ . Since σ is standard, $\sigma = [R] + \sigma_1$ for some standard β_{hd}^v -development σ_1 . $\nu(M_1)$ is finite and $\nu(N) \leq \nu(M_1)$ by induction hypothesis, and this implies that $\nu(M) = [R] + \nu(M_1)$ is finite, and $\nu(N) < \nu(M)$.
- R is not involved in σ . By Proposition 5..10, R has only one residual R_1 in N , which is $R_{hd}^v(N)$. Then $\mathbf{std}(\sigma + [R_1]) = [R] + \sigma_1$ for some standard β_{hd}^v -development σ_1 . Let $N \xrightarrow{R_1} \beta_{hd}^v N_1$, then $M_1 \xrightarrow{\sigma_1} \beta_{hd}^v N_1$. Since $|\nu(N_1)| < |\nu(N)|$, $|\nu(M_1)|$ is finite and $|\nu(N_1)| \leq |\nu(M_1)|$ by induction hypothesis. This implies that $\nu(M) = [R] + \nu(M_1)$ is finite and $|\nu(N)| = 1 + |\nu(N_1)| \leq 1 + |\nu(M_1)| = |\nu(M)|$.

■

Theorem 5..13 (β_{hd}^v -normalisation) *Given a term M ; if $M \xrightarrow{\sigma} \beta_{hd}^v N$ and N is in GHNF, then $\nu(M)$ is finite.*

Proof We proceed by induction on $|\sigma|$. If $\sigma = \emptyset$ then $\nu(M) = \emptyset$. Now we assume $\sigma = [R] + \sigma_1$ and $M \xrightarrow{R} \beta_{hd}^v M_1$. Since $|\sigma_1| < |\sigma|$, $\nu(M_1)$ is finite by induction hypothesis. Therefore, $\nu(M)$ is finite by Lemma 5..12. ■

Corollary 5..14 *Given $\lambda_{hd}^v \vdash M =_{\beta_{hd}^v} N$, where N is in GHNF; then $M \xrightarrow{m}_{\beta_{hd}^v} N_1$ for some N_1 and $\text{hd}(N) = \text{hd}(N_1)$.*

Proof Since $\lambda_{hd}^v \vdash M =_{\beta_{hd}^v} N$, $M \rightarrow_{\beta_{hd}^v} N_2$ and $N \rightarrow_{\beta_{hd}^v} N_2$ for some N_2 by Theorem 5..4. Since N is in GHNF, N_2 is also in GHNF by Proposition 4..8. Hence, Theorem 5..13 implies $\nu(M)$ is finite, i.e., $M \xrightarrow{m}_{\beta_{hd}^v} N_1$ for some N_1 in GHNF. By Corollary 5..5, $\text{hd}(N) = \text{hd}(N_1)$. ■

Definition 5..15 (β_{hd}^v -evaluation function)

$$\begin{aligned} \text{eval}_{hd}^v(M) &= M && \text{if } \text{hd}(M) \neq \emptyset; \\ \text{eval}_{hd}^v(\lambda x.M) &= (\lambda x.\text{eval}_{hd}^v(M)) && \text{if } \text{hd}(M) = \emptyset; \\ \text{eval}_{hd}^v(M(N)) &= \text{eval}_{hd}^v(\beta_{hd}^v(M, N)) && \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) \geq 0; \\ \text{eval}_{hd}^v(M(N)) &= \text{eval}_{hd}^v(M(\text{eval}_{hd}^v(N))) && \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) = \emptyset; \\ \text{eval}_{hd}^v(M(N)) &= \text{eval}_{hd}^v(\text{eval}_{hd}^v(M)(N)) && \text{if } \text{hd}(M) = \emptyset; \end{aligned}$$

Now we define the predicate $M \beta_{hd}^v$ -evaluates to N at time t as follows.

1. $M \beta_{hd}^v$ -evaluates to M at time 0 if $\text{hd}(M) \neq \emptyset$.
2. $\lambda x.M \beta_{hd}^v$ -evaluates to $\lambda x.N$ at time t if $M \beta_{hd}^v$ -evaluates to N at time t .
3. If $M \beta_{hd}^v$ -evaluates to M' at time t where $\text{hd}(M') \neq 0$, then $M(N) \beta_{hd}^v$ -evaluates to $M'(N)$ at time t .
4. If $M \beta_{hd}^v$ -evaluates to M' at time t where $\text{hd}(M') = 0$ and $N \beta_{hd}^v$ -evaluates to N' at time t' where $\text{hd}(N') \in \text{Var}$, then $M(N) \beta_{hd}^v$ -evaluates to $M'(N)'$ at time $t + t'$;
5. If $M \beta_{hd}^v$ -evaluates to M' at time t where $\text{hd}(M') = 0$, $N \beta_{hd}^v$ -evaluates to N' at time t' where $\text{hd}(N') \geq 0$, and $\beta_{hd}^v(M', N') \beta_{hd}^v$ -evaluates to L at time t'' , then $M(N) \beta_{hd}^v$ -evaluates to L at time $t + t' + t'' + 1$.

The following correspondence is obvious.

Proposition 5..16 *$M \beta_{hd}^v$ -evaluates to N at time t if and only if $\nu(M) : M \xrightarrow{m}_{\beta_{hd}^v} N$ and $|\nu(M)| = t$.*

Proof This follows from an induction on t . ■

Therefore, eval_{hd}^v always evaluates a program P to a value if $\lambda_{hd}^v \vdash P =_{\beta_{hd}^v} M$ for some value M . This justifies the definition of eval_{hd}^v .

For the sake of completeness, we also prove a version of standardisation theorem in λ_{hd}^v , which implies the β_{hd}^v -normalisation theorem.

Definition 5..17 *Given a β_{hd}^v -reduction sequence σ*

$$M_0 \xrightarrow{R_1}_{\beta_{hd}^v} M_1 \xrightarrow{R_2}_{\beta_{hd}^v} \cdots \xrightarrow{R_x}_{\beta_{hd}^v} M_n;$$

σ is standard if there exist no $1 \leq i < j \leq n$ such that R_j is a residual of some β_{hd}^v -redex R with $R \triangleleft_{hd}^v R_i$ in M_{i-1} .

Proposition 5..10 implies that the residuals of main β_{hd}^v -redexes are always main β_{hd}^v -redexes. This can be extended as follows.

Proposition 5..18 *Given $R_1 \triangleleft_{hd}^v R_2$ in M and $M \xrightarrow{\beta} N$; then R_1 has only one residual R'_1 in N ; if R' is not a residual of some redex in M , then $R'_1 \triangleleft_{hd}^v R'$ in N ; if R' is a residual of some redex R in M with $R_1 \triangleleft_{hd}^v R$, then $R'_1 \triangleleft_{hd}^v R'$ in N .*

Proof This simply follows from a structural induction on M . ■

Lemma 5..19 *Given $\sigma + \tau$: from M , where σ is a standard β_{hd}^v -development and τ is a standard β_{hd}^v -reduction sequence; then there exists a standard β_{hd}^v -reduction sequence*

$$\mathcal{S}(\sigma, \tau) : M \twoheadrightarrow_{\beta_{hd}^v} (\sigma + \tau)(M)$$

such that for every R if R is involved in $\mathcal{S}(\sigma, \tau)$ then R is involved in $\sigma + \tau$

Proof Let us proceed by an induction on $\langle |\tau|, |\sigma| \rangle$, lexicographically ordered.

- $\tau = \emptyset$. Then let $\mathcal{S}(\sigma, \emptyset) = \sigma$.
- $\sigma = \emptyset$. Then let $\mathcal{S}(\emptyset, \tau) = \tau$.
- $\sigma = [R_1] + \sigma_1$ and $\tau = [R_2] + \tau_1$. Now we have two subcases.
 - (1) R_2 is a residual of some redex in M . Then let $\mathcal{S}(\sigma, \tau) = \mathcal{S}(\mathbf{std}_{hd}^v(\sigma + [R_2]), \tau_1)$. Note that for every R if R is involved in $\mathbf{std}_{hd}^v(\sigma + [R_2])$ then R is involved in $\sigma + [R_2]$. By induction hypothesis, R is involved in $\mathbf{std}_{hd}^v(\sigma + [R_2]) + \tau$ if R is involved in $\mathcal{S}(\mathbf{std}_{hd}^v(\sigma + [R_2]), \tau_1)$. Hence, with Lemma 5..8, R is involved in $\sigma + \tau$ if R is involved in $\mathcal{S}(\sigma, \tau)$. $\mathcal{S}(\sigma, \tau)$ is standard by induction hypothesis.
 - (2) R_2 is not a residual of any redex in M . Then let $\mathcal{S}(\sigma, \tau) = [R_1] + \mathcal{S}(\sigma_1, \tau)$. By induction hypothesis, for every R if R is involved in $\mathcal{S}(\sigma_1, \tau)$ then R is involved in $\sigma_1 + \tau$. Hence, if R is involved in $\mathcal{S}(\sigma, \tau)$ then R is involved in $\sigma + \tau$. Let us assume $R \triangleleft_{hd}^v R_1$ in M ; then R is not involved in σ since σ is standard; by Proposition 5..18, R has only one residual R' in $\sigma(M)$ and $R' \triangleleft_{hd}^v R_2$ in $\sigma(M)$; then R' is not involved in τ since τ is standard; therefore, R is not involved in $\sigma + \tau$, and this yields that $\mathcal{S}(\sigma, \tau)$ is standard since $\mathcal{S}(\sigma_1, \tau)$ is standard by induction hypothesis. ■

Theorem 5..20 (β_{hd}^v -standardisation) *Given a β_{hd}^v -reduction sequence $\sigma : M \twoheadrightarrow_{\beta_{hd}^v} N$; then there exists a standard β_{hd}^v -reduction sequence $\mathbf{std}_{hd}^v(\sigma) : M \twoheadrightarrow_{\beta_{hd}^v} N$.*

Proof We proceed by induction on $|\sigma|$. If $\sigma = \emptyset$ then $\mathbf{std}_{hd}^v(\sigma) = \emptyset$; if $\sigma = [R] + \sigma_1$ then $\mathbf{std}_{hd}^v(\sigma) = \mathcal{S}([R], \mathbf{std}_{hd}^v(\sigma_1))$, which is standard by Lemma 5..19. ■

6. Relations between λ_{hd}^v and λ

We prove that a term can be reduced in λ_{hd}^v to a term in GHNF if and only if it can be reduced in the λ to a term in HNF. The following is Theorem 8.3.11 in [4].

Theorem 6..1 *Given a term M ; if $M \rightarrow_{\beta} N_1$ for some N_1 in HNF then $M \xrightarrow{h}_{\beta} N_2$ for some N_2 in HNF.*

We can then define an evaluation function corresponding to head β -reduction.

Definition 6..2 (*β -evaluation function*)

$$\begin{aligned} \text{eval}(M) &= M && \text{if } M \text{ in HNF;} \\ \text{eval}(\lambda x.M) &= (\lambda x.\text{eval}(M)) && \text{if } M \text{ is not in HNF;} \\ \text{eval}(M(N)) &= \text{eval}(\text{eval}(M)(N)) && \text{if } M \text{ is not in HNF.} \\ \text{eval}(M(N)) &= \text{eval}(\beta(M, N)) && \text{if } M \text{ is a } \lambda\text{-abstraction in HNF;} \end{aligned}$$

We define predicate M β -evaluates to N at time t similarly; with Proposition 2..5 (1), we have that M β -evaluates to N at time t if and only if $M \xrightarrow{h}_{\beta}^t N$ with N in HNF.

Lemma 6..3 *Given a term L ; if L β_{hd}^v -evaluates to M , then $L \rightarrow_{\beta} N$ for some N in HNF and $\text{hd}(N) = \text{hd}(M)$.*

Proof Assume L β_{hd}^v -evaluates to M at time t . We proceed by induction on t . If $t = 0$, then L is in GHNF and this follows from Proposition 4..5. Now assume $t > 0$. Then there exists L_1 such that $L \rightarrow_{\beta_{hd}^v} L_1$ and L_1 β_{hd}^v -evaluates to M at $t - 1$. By induction hypothesis, $L_1 \rightarrow_{\beta} N_1$ for some term N_1 in HNF with $\text{hd}(N_1) = \text{hd}(M)$. Also $L \rightarrow_{\beta} N_2$ and $L_1 \rightarrow_{\beta} N_2$ for some N_2 by Corollary 4..11. Hence, $N_1 \rightarrow_{\beta} N$ and $N_2 \rightarrow_{\beta} N$ for some N since λ is Church-Rosser, and this yields $L \rightarrow_{\beta} N$. Clearly, $\text{hd}(N) = \text{hd}(N_1) = \text{hd}(M)$. \blacksquare

Lemma 6..4 *Given terms M, M_1, N , where*

$$M \xrightarrow{m}_{\beta_{hd}^v} M_1 \text{ and } M \xrightarrow{h}_{\beta} N;$$

Let R be the head β -redex in M and R_v be the main β_{hd}^v -redex in M ; if R is not R_v , then $M_1 \xrightarrow{h}_{\beta} N_1$ and $N \xrightarrow{m}_{\beta_{hd}^v} N' \rightarrow_{\beta_{hd}^v} N_1$ for some N' and N_1 .

Proof Let us proceed by a structural induction on M .

- $M = \lambda x.M^1$. Then this follows from the induction hypothesis on M^1 .
- $M = M^1(M^2)$. We only handle one nontrivial subcase in which $R = M$ and R_v is in M^2 ; then $\text{hd}(M^1) = 0$ since R_v is in M^2 , and $M_1 = M^1(N^2)$ where $M^2 \xrightarrow{R_v}_{\beta_{hd}^v} N^2$, and

$N = \beta(M^1, M^2)$; thus, we have $M_1 \xrightarrow{h} \beta N_1$ for $N_1 = \beta(M^1, N^2)$; it can be verified that a R_v in some occurrence of M^2 in N is the main β_{hd}^v -redex in N since $\text{hd}(M^1) = 0$; we can contract the R_v in this occurrence of M^2 , and then β_{hd}^v -reduce all other occurrences of M^2 in N to N^2 ; therefore, $N \xrightarrow{m} \beta_{hd}^v N' \twoheadrightarrow \beta_{hd}^v N_1$. ■

Lemma 6..5 *Given terms M, N , where $M \xrightarrow{h} \beta N$ and N β_{hd}^v -evaluates to a term at time t ; then M β_{hd}^v -evaluates to L for some L in GHNF at some time $t' \leq t + 1$.*

Proof We proceed by induction on t . Let R be the head redex in M . If M is in GHNF then this is obvious. If R is a β_{hd}^v -redex, then R is the main β_{hd}^v -redex in M and this becomes trivial. Now we assume that $M \xrightarrow{m} \beta_{hd}^v M_1$ and R is not a β_{hd}^v -redex. Then $M_1 \xrightarrow{h} \beta N_1$ and $N \xrightarrow{m} \beta_{hd}^v N' \twoheadrightarrow \beta_{hd}^v N_1$ for some N_1 by Lemma 6..4. Since N' β_{hd}^v -evaluates to a term at time $t - 1$, N_1 β_{hd}^v -evaluates to a term at some time t_1 by Corollary 5..14, and $t_1 \leq t - 1$ by Lemma 5..12. By induction hypothesis, M_1 β_{hd}^v -evaluates to L for some L in GHNF at some time $t'_1 \leq t_1 + 1$. Hence M β_{hd}^v -evaluates to L at time $t'_1 + 1 \leq t_1 + 2 \leq t + 1$. ■

Corollary 6..6 *Given a term M which β -evaluates to a term at time t ; then M β_{hd}^v -evaluates to N for some N in GHNF at some time $t' \leq t$.*

Proof By Lemma 6..5, this follows from an induction on t . ■

Hence, \mathbf{eval}_{hd}^v never takes more time to terminate than \mathbf{eval} does.

Theorem 6..7 *Given a term M ; $\mathbf{eval}(M)$ is defined if and only if $\mathbf{eval}_{hd}^v(M)$ is defined.*

Proof If $\mathbf{eval}_{hd}^v(M)$ is defined then $M \twoheadrightarrow \beta N_1$ for some N_1 in HNF by Lemma 6..3, which implies that $\mathbf{eval}(M)$ is defined. If $\mathbf{eval}(M)$ is defined then $\mathbf{eval}_{hd}^v(M)$ is defined by Corollary 6..6. ■

We have shown that a program P β_{hd}^v -evaluates to a term in GHNF if and only if P β -evaluates to a term in HNF. If we extend λ_{hd}^v and λ with some base values such as integers and treat them as terms in HNF, then $\mathbf{eval}_{hd}^v(P)$ and $\mathbf{eval}(P)$ are well-defined for every program P which outputs a base value b , and $\text{hd}(\mathbf{eval}_{hd}^v(P)) = \mathbf{eval}(P) = b$. This suggests a new approach to implementing functional programming languages with call-by-name semantics, namely implementing λ_{hd}^v .

7. Operational Semantics

This section presents an operational semantics for λ_{hd}^v in the style of [15]. An application (MN) is written as $@(M, N)$ in this section. We begin with a description of our SECD machine.

The state of the SECD machine is a 4-tuple, $\langle S, E, C, D \rangle$, where S is a stack of closures, E is an environment, C is the control string and D is the dump. We define closures and environments inductively.

1.	$\langle \text{CL} : S, E, \emptyset, \langle S', E', C', D' \rangle \rangle$	\Rightarrow	$\langle \text{CL} : S', E', C', D' \rangle$
2.	$\langle S, E, x : C, D \rangle$	\Rightarrow	$\langle E(x)/E : S, E, C, D \rangle$
3.	$\langle S, E, (\lambda x.B) : C, D \rangle$	\Rightarrow	$\langle S, E, B : \lambda x : C, D \rangle$
4.	$\langle S, E, @(B, B') : C, D \rangle$	\Rightarrow	$\langle S, E, B : @B' : C, D \rangle$
5.	$\langle S, E, @(\text{CL}, B) : C, D \rangle$	\Rightarrow	$\langle \text{CL}/E : S, E, @B : C, D \rangle$
6.	$\langle (x, B, E') : S, E, \lambda x : C, D \rangle$	\Rightarrow	$\langle (0, \lambda \text{var}.B\{x := \text{var}\}, E') : S, E, C, D \rangle$
7.	$\langle (x, B, E') : S, E, \lambda y : C, D \rangle$	\Rightarrow	$\langle (x, \lambda y.B, E') : S, E, C, D \rangle$
8.	$\langle (n, \lambda x.B, E') : S, E, \lambda y : C, D \rangle$	\Rightarrow	$\langle (n+1, \lambda x.\lambda y.B, E') : S, E, C, D \rangle$
9.	$\langle (x, B, E') : S, E, @B' : C, D \rangle$	\Rightarrow	$\langle (x, @(B, B'), E') : S, E, C, D \rangle$
10.	$\langle (n+1, \lambda x.B, E') : S, E, @B' : C, D \rangle$	\Rightarrow	$\langle (n, \lambda x.@(B, B'), E') : S, E, C, D \rangle$
11.	$\langle (0, \lambda x.B, E') : S, E, @B' : C, D \rangle$	\Rightarrow	$\langle (0, \lambda x.B, E') : S, E, B' : @ : C, D \rangle$
12.	$\langle (x, B, E') : \text{CL} : S, E, @ : C, D \rangle$	\Rightarrow	$\langle (x, @(\text{CL}, B), E') : S, E, C, D \rangle$
13.	$\langle \text{CL} : (0, \lambda x.B, E') : S, E, @ : C, D \rangle$	\Rightarrow	$\langle \emptyset, E'[x \mapsto \text{CL}], B, \langle S, E, C, D \rangle \rangle$

Table 1: The SECD machine state transitions

- Given distinct variables x_i and closures CL_i for $i = 1, \dots, n$; the following ordered sequence E

$$(x_1 \mapsto \text{CL}_1, \dots, x_n \mapsto \text{CL}_n)$$

is an environment; $\text{Dom}(E) = \{x_1, \dots, x_n\}$; $E = ()$ is an empty environment when $n = 0$.

- A closure CL is of form (hd, bd, env) ; the closure head $\text{hd}(\text{CL})$ of CL is hd ; the closure body $\text{bd}(\text{CL})$ of CL is bd ; the closure environment $\text{env}(\text{CL})$ of CL is env ; let B range over closure bodies.

Given an environment E ;

- if $x \notin E$ then (x, x, E) is a closure;
- $(x, @(B, B'), E)$ is a closure for any closure body B' if (x, B, E) is a closure.
- $(x, @(\text{CL}, B), E)$ is a closure for any closure CL with $\text{hd}(\text{CL}) = 0$ if (x, B, E) is a closure;
- $(0, \lambda x.B, E)$ is a closure if (x, B, E) is a closure.
- $(x, \lambda y.B, E)$ is a closure for $y \neq x$ if (x, B, E) is a closure;
- $(n+1, \lambda x.B, E)$ is a closure if (n, B, E) is a closure;
- $(n, @(B, B'), E)$ is a closure for any closure body B' if $(n+1, B, E)$ is a closure.

It is straightforward to define α -conversion on closure bodies, and the equality between bodies is modular α -conversion. Notice that only terms in GHNF can form closures. This is similar to the definition of closures given by Landin [13], where only λ -abstractions can form closures.

Notice that every environment we can formulate below is of form $(x_1 \mapsto \text{CL}_1, \dots, x_n \mapsto \text{CL}_n)$, where CL_i does not contain any free occurrence of x_j for $1 \leq j \leq i$.

Given $E = (x_1 \mapsto \text{CL}_1, \dots, x_n \mapsto \text{CL}_n)$; if $x = x_i$ for some $1 \leq i \leq n$ then $E(x) = \text{CL}_i$; if $x \notin \text{Dom}(E)$ then $E(x) = (x, x, ())$; if $x \notin \text{Dom}(E)$, we write $E[x \mapsto \text{CL}]$ for

$$(x_1 \mapsto \text{CL}_1, \dots, x_n \mapsto \text{CL}_n, x \mapsto \text{CL}).$$

We define functions $B_\emptyset : \text{bodies} \rightarrow \text{terms}$ and $\text{BE} : \text{bodies} \times \text{environments} \rightarrow \text{terms}$ inductively.

$$\begin{aligned} B_\emptyset(x) &= x \\ B_\emptyset(@ (B, B')) &= @(B_\emptyset(B), B_\emptyset(B')) \\ B_\emptyset(@ (\text{CL}, B)) &= @(\text{BE}(\text{bd}(\text{CL}), \text{env}(\text{CL})), B_\emptyset(B)) \\ B_\emptyset(\lambda x. B) &= \lambda x. B_\emptyset(B) \\ \text{BE}(B, E) &= [M_n/x_n](\dots([M_1/x_1]B_\emptyset(B))\dots) \end{aligned}$$

where $E = (x_1 \mapsto \text{CL}_1, \dots, x_n \mapsto \text{CL}_n)$ and $M_i = \text{BE}(\text{CL}_i)$ for $i = 1, \dots, n$. Note that is *not* a simultaneous substitution. Let function $\text{Real} : \text{closures} \rightarrow \text{terms}$ be defined as

$$\text{Real}(\text{CL}) = \text{BE}(\text{hd}(\text{CL}), \text{env}(\text{CL})).$$

Lemma 7..1 *Given closures*

$$\text{CL}_1 = (\text{hd}_1, \lambda x. B_1, E_1) \text{ and } \text{CL}_2 = (\text{hd}_2, B_2, E_2),$$

and terms $M_i = \text{Real}(\text{CL}_i)$ for $i = 1, 2$; then $\text{BE}(B_1, E[x \mapsto \text{CL}_2]) = \beta(M_1, M_2)$.

Proof This follows from a straightforward structural induction on B_1 . ■

Given two environments E and E' ; an environment E/E' is defined inductively as follows;

$$\begin{aligned} E/() &= E; \\ E/(x \mapsto \text{CL}) &= E \quad \text{if } x \in \text{Dom}(E); \\ E/(x \mapsto \text{CL}) &= E[x \mapsto \text{CL}] \quad \text{if } x \notin \text{Dom}(E); \\ E/E'[x \mapsto \text{CL}] &= (E/E')/(x \mapsto \text{CL}) \text{ if } E' \neq (). \end{aligned}$$

We write CL/E' for $(n, B, E/E')$ if $\text{CL} = (n, B, E)$.

Now we are ready to present the state transitions of our SECD machine, which are listed in Table 1. **var** always stands for some fresh variable when Transition 6 is applied. We assume the familiarity of the reader with the notation, which is adopted from [15]. We first define some functions; let

$$\text{Load}(M) = \langle [], (), [], [] \rangle,$$

where $[]$ stands for an empty list; let

$$\text{Unload}(\langle [\text{CL}], (), [], [] \rangle) = \text{Real}(\text{CL});$$

$\text{Eval}_{hd}^v(M) = N$ if and only if $\text{Load}(M) \Rightarrow^* D$ and $N = \text{Unload}(D)$ for some dump D .

Note that a fresh bound variable x is introduced whenever a closure of form $(0, \lambda x. B, E)$ is constructed through Transition 6 in Table 1. This means that x does not occur in the domains of any environments when $x \mapsto \text{CL}$ is added to some environment through Transition 13. This easily ensures that that no name clashes can occur in Transition 2 and Transition 5 where $/$ operator is applied. With this observation, we can establish the expected relation between Eval_{hd}^v and eval_{hd}^v .

Theorem 7..2 $\text{Eval}_{hd}^v(P) = \text{eval}_{hd}^v(P)$ for every program P .

Proof See the proof of Theorem 1 in [15]. We omit a long but straightforward argument here. ■

Like the usual call-by-value SECD machine, the presented SECD machine can be improved in numerous ways. Transition 6 is a serious obstacle to efficiency because of the renaming. We are planning to fix this problem soon. We also plan to design a call-by-need λ -calculus corresponding to λ_{hd}^v and implement it. This is a subject of future work.

8. Extensions

We first add a recursion combinator to β_{hd}^v -calculus eliminating some syntactic overhead. Then we extend β_{hd}^v -calculus with constructors and primitive functions.

8.1. Recursion

Many fixed point operators have been constructed. For instance,

$$Y = \lambda f.(\lambda x.f(x(x)))(\lambda x.f(x(x)))$$

is a fixed point operator. There is a great deal of syntactic overhead involved if we use fixed point operators to do recursion directly. This suggests that we introduce **fix** as a recursion combinator with the following rule

$$\mathbf{fix}(f) \rightarrow f(\mathbf{fix}(f)).$$

Let $\text{hd}(\mathbf{fix}) = 0$; if $\text{hd}(f) \geq 0$ then $\mathbf{fix}f$ is a β_{hd}^v -redex and $\beta_{hd}^v(\mathbf{fix}, f) = f(\mathbf{fix}f)$. Another possibility is to introduce *letrec*, which will be explored when we study implementations of λ_{hd}^v .

8.2. Constructors

We treat base values such as integers and boolean values as constructors with 0 arity. We need extend the definition of terms and the definition of hd .

Definition 8.1 $c^n(M_1, \dots, M_n)$ is a term if c^n is constructor with arity n and M_1, \dots, M_n are terms. Let Const be the set of all constructors.

$$\begin{aligned} \text{hd}(M) &= c^n && \text{if } M = c^n(M_1, \dots, M_n); \\ \text{hd}(M(N)) &= c^n && \text{if } \text{hd}(M) \in \text{Const}; \\ \text{hd}(M(N)) &= \text{hd}(N) && \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) \in \text{Const}. \end{aligned}$$

Let $S = \mathbf{fix}(\lambda x.\mathbf{cons}(0, x))$, then $\text{hd}(S) = \mathbf{cons}$; so S is in GHNF. This is justified in most actual implementations which allocate only one cell for **cons**, representing S as a cyclic data structure. Therefore, the λ_{hd}^v -calculus does not have the deficiency of the call-by-need λ -calculus [3], where S evaluates to $\mathbf{cons}(0, S)$ containing two distinct **cons** cells.

8.3. Primitive Functions

Primitive functions have to be handled individually according to their semantics. We use a few examples illustrating our points. Let Δ_n represent integer n for $n = 0, 1, \dots$ and \mathbf{t}, \mathbf{f} stand for truth values. Let $\text{Int} = \{\Delta_0, \Delta_1, \dots\}$ and $\text{Bool} = \{\mathbf{t}, \mathbf{f}\}$.

Let fun be a primitive function on integers with arity 1. We intend to define $\text{hd}(\text{fun}) = \text{fun}$ and $\text{hd}(\text{fun}(M)) = M$ if M is a variable, but this definition has a serious flaw; assume that $\text{fun}(M)$ is a δ -redex if $\text{hd}(M)$ is some integer; then

$$\text{hd}((\lambda x.\text{fun}(x))M) = \text{hd}(M)$$

implies that $(\lambda x.\text{fun}(x))M$ is in GHNF; this prevents eval_{hd}^v from evaluating $(\lambda x.\text{fun}(x))M$ to the value of $\text{fun}(M)$. Our solution to this dilemma is to modify the definition of general head; let

$$\text{hd}(\text{fun}(x)) = \langle x, \text{Int} \rangle$$

and $\lambda x.\text{fun}(x)$ have head $\langle 0, \text{Int} \rangle$; $M(N)$ is regarded as a β_{hd}^v -redex if $M = \langle 0, \text{Int} \rangle$ and $\text{hd}(N) \in \text{Int}$. Clearly, Int can be replaced with other sets of constants. Also it is easy to see how to adjust the definition of hd to handle such pairs. We write $\text{hd}(M) \in \text{Pair}$ if M has a head which is a pair.

8.3.1. Basic Operations on Integers

We demonstrate how addition (+) can be handled. The general head of $+(M, N)$ is given as follows.

$$\text{hd}(+(M, N)) = \begin{cases} \emptyset & \text{if } \text{hd}(M) = \emptyset; \\ \langle \text{hd}(M), \text{Int} \rangle & \text{if } \text{hd}(M) \in \text{Var}; \\ \text{hd}(M) & \text{if } \text{hd}(M) \in \text{Pair}; \\ \emptyset & \text{if } \text{hd}(M) = \text{Int} \text{ and } \text{hd}(N) = \emptyset; \\ \langle \text{hd}(N), \text{Int} \rangle & \text{if } \text{hd}(M) \in \text{Int} \text{ and } \text{hd}(N) \in \text{Var}; \\ \text{hd}(N) & \text{if } \text{hd}(M) = \text{Int} \text{ and } \text{hd}(N) \in \text{Pair}; \\ \emptyset & \text{if } \text{hd}(M) \in \text{Int} \text{ and } \text{hd}(N) \in \text{Int}. \end{cases}$$

$+(M, N)$ is a δ -redex if $\text{hd}(M), \text{hd}(N) \in \text{Int}$. We also extend the definition of eval_{hd}^v as follows.

$$\begin{aligned} \text{eval}_{hd}^v(+(M, N)) &= \Delta_{m+n} && \text{if } \text{hd}(M) = \Delta_m \text{ and } \text{hd}(N) = \Delta_n; \\ \text{eval}_{hd}^v(+(M, N)) &= \text{eval}_{hd}^v(+(\text{eval}_{hd}^v(M), N)) && \text{if } \text{hd}(M) = \emptyset; \\ \text{eval}_{hd}^v(+(M, N)) &= \text{eval}_{hd}^v(+(M, \text{eval}_{hd}^v(N))) && \text{if } \text{hd}(M) \neq \emptyset \text{ and } \text{hd}(N) = \emptyset; \end{aligned}$$

Other operations, such as subtraction ($-$), multiplication (\times) and equality ($=$), can be handled in a similar fashion.

8.3.2. Conditional

We introduce a conditional IF; $\text{IF}(M, N_1, N_2)$ is a term if M, N_1, N_2 are terms; the general head of $\text{IF}(M, N_1, N_2)$ is defined as follows.

$$\text{hd}(\text{IF}(M, N_1, N_2)) = \begin{cases} \emptyset & \text{if } \text{hd}(M) \in \text{Bool}; \\ \langle \text{hd}(M), \text{Bool} \rangle & \text{if } \text{hd}(M) \in \text{Var}; \\ \text{hd}(M) & \text{if } \text{hd}(M) \notin \text{Var} \cup \text{Bool}; \end{cases}$$

$\text{IF}(M, N_1, N_2)$ is a δ -redex if $\text{hd}(M) \in \text{Bool}$. We extend the definition of eval_{hd}^v , defining

$$\text{eval}_{hd}^v(\text{IF}(M, N_1, N_2)) = \begin{cases} \text{eval}_{hd}^v(N_1) & \text{if } \text{hd}(M) = \mathbf{t}; \\ \text{eval}_{hd}^v(N_2) & \text{if } \text{hd}(M) = \mathbf{f}; \\ \text{eval}_{hd}^v(\text{IF}(\text{eval}_{hd}^v(M), N_1, N_2)) & \text{if } \text{hd}(M) = \emptyset. \end{cases}$$

8.4. Potential Improvements

Let BIG be some computational complex function; for every n ,

$$M = \lambda x. + (x, (\text{BIG } \Delta_n))$$

is regarded as a value since $\text{hd}(M) = \langle 0, \text{Int} \rangle$ in the above extension; this may lead to repeated evaluation of $(\text{BIG } \Delta_n)$. A potential solution to this problem is to introduce a notion of multiple general heads; we can adjust the definition of hd and define

$$\text{hd}(+(M, N)) = \{\text{hd}(M), \text{hd}(N)\};$$

$\text{hd}(+(M, N))$ is in GHNF if both M and N are in GHNF ; we then need modify eval_{hd}^v to accommodate the change. We are currently studying on this subject.

9. Related Work

λ_{hd}^v is closely related to the weak λ -calculus in [18] and the call-by-need λ -calculus in [3] in the following sense; we can define a kind of redex as a term of form $M(N)$ and its contractum as $\text{sub}N \bullet \text{gbd}(M)$, where M is a term with $\text{hd}(M) = 0$ and N is a variable or a λ -abstraction; it can be readily verified that such a form is closed under value substitution if values are defined as variables or λ -abstractions; we can then define a kind of λ -calculus corresponding to such redexes; a close correspondence between this λ -calculus and the λ -calculi in [18] and [3] can be established accordingly. It is this observation that motivates the paper.

The problem of sharing evaluation under λ -abstraction has lead to a great deal of study on λ -lifting [12] and *supercombinators* [9] under the title *full laziness* or *maximal laziness*. Because of the ability of evaluating under λ directly, eval_{hd}^v can achieve what is beyond the scope of either λ -lifting or supercombinators. Given a term

$$M = \lambda z. \lambda x. \lambda y. + (\text{BIG}(\text{IF}(z, x, y)), \times(x, y))$$

and a program

$$P = \lambda u. (\dots u \dots u \dots)(M(\mathbf{t})(\Delta_n)),$$

where BIG stands for some computationally complex strict function, and $\lambda u. (u \dots u \dots)$ is a term with general head 0 . Note

$$\text{eval}_{hd}^v(M(\mathbf{t})(\Delta_n)) = (\lambda y. + (\text{BIG}_n, \times(x, y))),$$

where BIG_n is the value of $\text{BIG}(\Delta_n)$. Hence, eval_{hd}^v evaluates $\text{BIG}(\Delta_n)$ only once when evaluating P . Such a sharing of evaluation cannot be done using λ -lifting or supercombinators since

term $\text{IF}(z, x, y)$ contains variable y at compile time. In some sense, full laziness is really not full when evaluation under λ is allowed.

Another related subject is partial evaluation. We show that eval_{hd}^v works well with staged computation [5]. Let us define the power function **pow** as

$$\mathbf{fix}(\lambda f \lambda p \lambda n. \text{IF}(= (p, \Delta_0), \Delta_1, \times(f(n) (- (p, 1)), n))).$$

Note $\text{hd}(\mathbf{pow}) = \langle \emptyset, \text{Int} \rangle$. The followings can be readily verified.

$$\begin{aligned} \text{eval}_{hd}^v(\mathbf{pow}(\Delta_0)) &= \lambda n. (\lambda f. \lambda n. \Delta_1)(\mathbf{pow})(n) \\ \text{eval}_{hd}^v(\mathbf{pow}(\Delta_1)) &= \lambda n. (\lambda n. \times ((\lambda f. \lambda n. \Delta_1)(\mathbf{pow})(n), n))(n) \\ \text{eval}_{hd}^v(\mathbf{pow}(\Delta_2)) &= \lambda n. (\lambda n. \times ((\lambda n. \times ((\lambda f. \lambda n. \Delta_1)(\mathbf{pow})(n), n))(n), n))(n) \end{aligned}$$

This is very close to Example 2.4 in [5]. Suppose that we implement eval_{hd}^v using the above presented SECD machine; if a term $\mathbf{pow}(\Delta_n)$ is generated at *run-time* and needs to be evaluated, the machine always evaluates it to a GHNF before forming a closure; this is quite desirable since this amounts to some sort of partial evaluation at run-time. For the following function *dotprod* which computes the inner product of two vectors of some given length [11], the reader can also verify that $\text{eval}_{hd}^v(\text{dotprod}(\Delta_n))$ is adequately expanded for every $\Delta_n \in \text{Int}$. Note that $\text{vec}[n]$ yields the n th element in vector *vec*.

$$\mathbf{fix}(\lambda f \lambda n \lambda u \lambda v. \text{IF}(= (n, 0), 0, +(f(- (n, 1))(u)(v), \times(u[n], v[n]))))$$

Also eval_{hd}^v does not suffer from any termination problems, which on the other hand, significantly limit the use of partial evaluators. Let us define Ackermann's function *acker* as

$$\mathbf{fix}(\lambda f \lambda m \lambda n. \text{IF}(= (m, 0), +(n, 1), f(- (m, 1))(\text{IF}(= (n, 0), +(n, 1), f(m) (- (n, 1))))))$$

Given any $\Delta_m \in \text{Int}$, $\text{eval}_{hd}^v(\text{acker}(\Delta_m))$ always terminates since no terms under the second IF can be evaluated when n is unknown.

Our work also relates to [10]. We show that eval_{hd}^v can achieve *complete laziness* for the following example taken from [10] if we form closures instead of performing substitutions. This cannot be done by an evaluation strategy corresponding to full laziness as mentioned in [10]. Here lower case letters are variables and uppercase letters are closed expressions.

$$(\lambda f. f(B)(f(C)))((\lambda a. \lambda z. (\lambda g. g(a))(\lambda x. x(x)(z)))(A))$$

We assume that A is already in GHNF. Note that $(\lambda a. \lambda z. (\lambda g. g(a))(\lambda x. x(x)(z)))(A)$ reduces to $(\lambda z. x(x)(z))$ with x bound to A ; this term is not in flexible GHNF; therefore, the β -redex $x(x)$ with x bound to A needs to be reduced before we can bind z to B and C , avoiding evaluating it twice.

These examples suggest that eval_{hd}^v be a evaluation function which is able to perform some degree of *on-line* partial evaluation. This favors that a polished implementation of eval_{hd}^v is promising to enhance the performance of lazy functional programming languages.

10. Conclusions and Future Work

We have presented a call-by-value λ -calculus λ_{hd}^v in which values are defined as terms in flexible generalised head normal form. λ_{hd}^v enjoys many similar properties as the λ -calculus λ does.

Given a program P which outputs base values such as integers, $\lambda \vdash P =_{\beta} b$ for some base value b if and only if $\lambda_{hd}^v \vdash P =_{\beta_{hd}^v} M$ for some M in GHNF with b as its general head. Therefore, lazy functional programming languages can implement λ_{hd}^v without compromising their semantics. A call-by-need implementation of λ_{hd}^v suggests a higher degree of sharing since evaluation can take place under λ -abstraction, viz. in the bodies of functions. We have also designed a SECD machine which can easily lead to an implementation of the evaluation function \mathbf{eval}_{hd}^v for λ_{hd}^v .

We intend to extend λ_{hd}^v with explicit substitutions and study approaches to implementing λ_{hd}^v efficiently. We would also like to establish a λ -calculus corresponding to the usual call-by-value λ -calculus λ_v , in which evaluation under λ can be performed. We believe that studies on λ_{hd}^v can help enhance the performance of functional programming languages.

11. Acknowledgements

I gratefully acknowledge my discussion with Frank Pfenning regarding the subject of paper, and thank him for his many constructive comments on this work. I also thank Peter Andrews for providing me such a nice work environment.

References

- [1] M. Abadi, L. Cardelli, P.-L. Curien, and I.-I. Lévy (1991), Explicit substitutions, *Journal of Functional Programming*, 4(1), pp. 375-416.
- [2] S. Abramsky (1990), The lazy lambda calculus. In D. Turner, editor, *Declarative Programming*, Addison-Wesley Publishing Company.
- [3] Z.M. Ariola, M. Felleisen, J. Maraist, M. Odersky and P. Wadler (1996), A Call-by-Need Lambda Calculus, *Proceedings of the 22nd ACM Symposium on Principles of Programming Languages*, San Francisco, pp. 233-246
- [4] H.P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, North-Holland, Amsterdam, 1984.
- [5] R. Davies and F. Pfenning (1996), A Modal Analysis of Staged Computation, In Proceedings of the 23rd ACM Symposium on Principles of Programming Languages, pp. 206-209
- [6] A.J. Field and P.G. Harrison (1988), *Functional Programming*, Addison-Wesley Publishing Company.
- [7] J.R. Hindley (1978), Reductions of residuals are finite, *Trans. Amer. Math. Soc.* 240, pp. 345-361.
- [8] G. Huét (1994), Residual Theory in λ -Calculus: A Formal Development, *Journal of Functional Programming Vol. 4*, pp. 371-394.
- [9] R.J.M. Hughes (1984), The design and implementation of programming languages, *Ph.D. thesis, University of Oxford*.

- [10] C.K. Holst and C.K. Gomard (1991), Partial Evaluation is Fuller Laziness, in *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pp. 223-233.
- [11] R. Glück and J. Jørgensen (1995), Efficient multi-level generating extensions for program specialisation. In S.D. Swierstra and M. Hermenegildo, editors, *Programming Languages, Implementations Logics and Programs, Springer-Verlag LNCS 982*, pp. 259-278.
- [12] T. Johnsson (1985), Lambda lifting: transforming programs to recursive equations, In *Proceedings of the Conference on Functional programming Languages and Computer Architecture*, Nancy, pp. 190-203.
- [13] P.J. Landin (1964), The mechanical evaluation of expressions, *BCS Computing Journal* 6(4), pp. 308-320.
- [14] S.L. Peyton Jones (1991), A fully-lazy lambda lifter in Haskell, *Software practice and experience* 21.
- [15] G.D. Plotkin (1975), Call-by-name, call-by-value and the lambda-calculus, *Theoretical Computer Science* 1, pp. 125-159.
- [16] Chris Reade (1989), *Elements of Functional Programming*, Addison-Wesley Publishing Company.
- [17] H. Xi (1996), Separating developments in λ -calculus, *Technical report, Department of Mathematical Sciences, Carnegie Mellon University*, Pittsburgh.
- [18] N. Yoshida (1993), Optimal reduction in weak λ -calculus with shared environments. In *Proc. ACM conference on Functional Programming Languages and Computer Architecture*, Copenhagen.