Evaluation under $\lambda$-abstraction

Hongwei Xi
Department of Mathematical Sciences
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
Email: hwx+@cs.cmu.edu

Abstract

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

1. Introduction

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

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

where $I = (\lambda z.z)$, the $\beta$-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 $\lambda$-abstraction desirable.

Why is a value defined as a $\lambda$-abstraction or a variable in the (usual) call-by-value $\lambda$-calculus $\lambda_v$ [15]? One crucial observation is that this form of values is closed under value substitution.
This directly leads to the notion of residuals of $\beta_v$-redexes under $\beta_v$-reduction and the notion of parallel $\beta_v$-reduction, by which it follows that $\lambda_v$ enjoys the Church-Rosser property and a version of standardisation theorem. An evaluation function for $\lambda_v$ can then be defined and justified by the standardisation theorem in $\lambda_v$.

There exists another form of terms which is closed under substitution. A term in head normal form (HNF) is of form $\lambda x_1...\lambda x_m.x(M_1)...(M_n)$; 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 $\lambda$-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 $\lambda$-calculus $\lambda^v_{hd}$ based on this definition. After proving a version of normalisation theorem in $\lambda^v_{hd}$, 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 $\lambda$-calculus $\lambda^v_{hd}$ if and only if it can be reduced to a term in HNF in the usual $\lambda$-calculus $\lambda$. We also present an operational semantics for $\lambda^v_{hd}$ via a SECD machine, which can easily lead to a mechanical implementation.

Lazy functional programming languages implement the (call-by-name) $\lambda$-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 $\lambda$-calculus $\lambda^v_{hd}$ without compromising their semantics.

The next section presents some preliminaries such as head $\beta$-reductions and residuals in the $\lambda$-calculus $\lambda$. The third section introduces some proof techniques handling $\beta$-developments, which are used later to show some syntactic properties which $\lambda^v_{hd}$ enjoys. The fourth section presents the definitions of generalised head normal form and $\beta^v_{hd}$-redexes. The fifth section studies $\lambda^v_{hd}$, and the sixth section studies the relations between $\lambda^v_{hd}$ and $\lambda$. The seventh section presents an operational semantics of $\lambda^v_{hd}$ via a SECD machine. The eighth section deals with extensions of $\lambda^v_{hd}$ 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 $\lambda$-calculus $\lambda$ [4].

Definition 2.1.1 The set $\Lambda$ of $\lambda$-terms is defined inductively as follows.

- **(variable)** There are infinitely many variables $u, v, x, y, z, \ldots$ in $\Lambda$; 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 $\text{Var}$ 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 $\alpha$-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 $\Lambda$; $P$ ranges over programs; $R$ ranges over various redexes defined below; $\sigma, \tau$ range over various reduction sequences defined below, and $\emptyset$ stands for an empty reduction sequence.

We introduce a new symbol $\bullet$ as a placeholder and treat it as a variable. The body of a $\lambda$-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 $\lambda$-terms until Section 8.

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

Given two $\beta$-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 $\rightarrow$ in this paper, the corresponding decorated reduction notations of $\rightarrow^n$ and $\rightarrow^* \text{ stand for } n \text{ steps and some (possibly zero) steps of such a reduction, respectively.}$

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

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

for $1 \leq i \leq n$.

The following explicit notation of $\beta$-reduction sequences enables us to write out the contracted $\beta$-redexes.
**Definition 2.3** (Explicit β-reduction sequences) Given a β-redex \( R \) in \( M \); \( M \xrightarrow{R} N \) stands for the β-reduction step in which \( R \) gets contracted; \([R_1] + \cdots + [R_n]\) stands for a reduction sequence of the following form.

\[
M_0 \xrightarrow{R_1} M_1 \xrightarrow{R_2} \cdots \xrightarrow{R_n} M_n.
\]

**Notations** \( \sigma : M \xrightarrow{\beta} N \) or \( M \xrightarrow{\beta} N \) stands for a β-reduction sequence from \( M \) to \( N \); for a finite β-reduction sequence \( \sigma \) from \( M \), \( \sigma(M) \) stands for the term to which \( \sigma \) reduces \( M \); \( |\sigma| \) is the length of \( \sigma \), namely, the number of β-reduction steps in \( \sigma \); \( \sigma + \tau \) stands for a β-reduction sequence of form \( M_0 \xrightarrow{\beta} M_1 \xrightarrow{\beta} M_2 \); \( \sigma\{x := L\} \) stands for the β-reduction sequence obtained from \( \sigma \) by replacing every free occurrence of variable \( x \) in \( \sigma \) 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 \( \lambda x_1 \ldots \lambda x_m.(\lambda x_1.N_1)(N_2)(M_1) \ldots (M_n) \),

where \( m, n \geq 0 \). We write \( M \xrightarrow{h} N \) if \( M \xrightarrow{R} N \) 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 \( \lambda x_1 \ldots \lambda x_m.xM_1 \ldots M_n \), where \( m \geq 0 \), \( n \geq 0 \); the HNF is flexible if \( x \) occurs in \( x_1, \ldots, x_m \), 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 \( x_1, \ldots, x_n \); \( |\vec{x}| = n \) is the length of the sequence; \( \lambda \vec{x}.M \) is \( \lambda x_1 \ldots \lambda x_m.M_1 \); \( \vec{M} \) denotes a sequence (possibly empty) of terms \( M_1, \ldots, M_n \); \( |\vec{M}| = n \) is the length of \( M \); \( M(M) \) is \( M(M_1) \ldots (M_n) \); \( \vec{M}\{x := N\} = M_1^*, \ldots, M_n^* \), 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. \( M \xrightarrow{h} N \) implies \( M\{x := L\} \xrightarrow{h} N\{x := L\} \).
2. \( M \xrightarrow{h} \lambda x.M_1 \) implies \( M(N) \xrightarrow{h+1} M_1\{x := N\} \).
3. Given \( M \xrightarrow{h} \lambda x.\vec{x}(\vec{N}) \) where \( x \) is not in \( \vec{x} \) and \( N \xrightarrow{h} \bar{y}.z(\vec{N}) \) where \( z \) is not in \( \bar{y} \); then \( M(N) \xrightarrow{h+1} \lambda x.\vec{y}.z(\vec{L}) \), where \( \bar{L} = m + n + 1 + \max(|\bar{y}|, |\vec{N}|) \).

**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.
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 $M \overset{\beta}{\rightarrow}_N$. 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 $N$. The 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 $\sigma$ from $M$, we say that a β-redex $R$ in $M$ is involved in $\sigma$ if $R$ or one of its residuals gets contracted in $\sigma$.

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 $\beta_{hd}$-reduction defined in the next section.

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

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

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

where $\sigma_1$ is a development from $bd(M_1)$ and $\sigma_{2i}$ are developments from $M_2$ for $i = 1, \ldots, n$.

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

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

5
and we can easily verify that for every $R$ in $M$ if $R$ is involved in $\text{sp}(\sigma)$ then $R$ is involved in $\sigma$. ■

The finiteness of $\beta$-developments theorem (FD), which states that all $\beta$-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 $\beta$-developments) Given a $\beta$-development

$$\sigma : M_0 \mapsto_\beta M_1 \mapsto_\beta \cdots \mapsto_\beta M_n;$$

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

In other words, a $\beta$-development is canonical if it contains no inside-out $\beta$-reductions. Clearly, a standard $\beta$-development is canonical but a canonical $\beta$-development may not be standard. Let $\sigma = [R_1] + \cdots + [R_n]$ be a canonical $\beta$-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 $\sigma$ into

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

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

**Lemma 3.4** (Standardisation of $\beta$-developments) For every $\beta$-development $M \mapsto_\beta N$, there exists a standard development

$$\text{std}(\sigma) : M \mapsto \beta N$$

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

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

- $M$ 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 $\beta$-redex. Then we can assume $\sigma$ to be of form $\sigma_1 + \sigma_2$, where $\sigma_i$ are developments from $M_i$ for $i = 1, 2$. Hence $\text{cd}(\sigma) = \text{cd}(\sigma_1) + \text{cd}(\sigma_2)$ is a canonical development reducing $M$ to $N$. Assume that $R$ is involved in $\text{cd}(\sigma)$; then $R$ is involved in $\text{cd}(\sigma_i)$ for some $i \in \{1, 2\}$; this implies $R$ is involved in $\sigma_i$ by induction hypothesis, and therefore $R$ is involved in $\sigma$. 

6
\[ M = M_1(M_2), \text{ where } M \text{ is a } \beta\text{-redex. If } M \text{ has a residual in } N, \text{ then this case is the same as the previous one. Hence, let us assume that } M \text{ is involved in } \sigma. \text{ By Lemma 3.2, } sp(\sigma) \text{ is of form} \]
\[ [M] + \sigma_1 \{ \bullet := M_1 \} + \sigma_2 + \cdots + \sigma_{2n} \]
where \( \sigma_1 \) is a development from \( bd(M_1) \) and \( \sigma_{2i} \) are developments from \( M_2 \). Note that for every \( R, R \text{ is involved in } \sigma \) if \( R \text{ is involved in } sp(\sigma) \). Let us define \( cd(\sigma) \) as
\[ [M] + cd(\sigma_1) \{ \bullet := M_1 \} + cd(\sigma_{21}) + \cdots + cd(\sigma_{2n}). \]
It is a routine verification that \( cd(\sigma) \) is canonical since \( cd(\sigma_1) \) and \( cd(\sigma_{21}), \ldots, cd(\sigma_{2n}) \) are canonical. Assume that \( R \text{ is involved in } cd(\sigma) \); if \( R = M \) then \( R \text{ is involved in } sp(\sigma) \); if \( R \text{ is involved in } cd(\sigma_1) \{ \bullet := M_1 \} \) then \( R \text{ is involved in } \sigma_1 \{ \bullet := M_1 \} \) by induction hypothesis; if \( R \text{ is involved in } cd(\sigma_{2i}) \) for some \( 1 \leq i \leq n \), then \( R \text{ is involved in } \sigma_{2i} \) by induction hypothesis; therefore, \( R \text{ is always involved in } sp(\sigma), \) and this implies that \( R \text{ is involved in } \sigma. \]

Given a \( \beta \)-development \( \sigma \), Lemma 3.4 enables us to define a measure \( |std(\sigma)| \) for \( \sigma \). 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 \( \beta^*_{hd} \)-redexes and their residuals under \( \beta^*_{hd} \)-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 \( \text{Num} = \{0, 1, 2, \ldots\} \).

\[
\begin{align*}
\text{ld}(M) & = M & \text{if } M \in \text{Var}; \\
\text{ld}(\lambda x. M) & = \text{ld}(M) & \text{if } \text{ld}(M) \neq x; \\
\text{ld}(\lambda x. M) & = 0 & \text{if } \text{ld}(M) = x; \\
\text{ld}(\lambda x. M) & = n + 1 & \text{if } \text{ld}(M) = n; \\
\text{ld}(\lambda x. M) & = 0 & \text{if } \text{ld}(M) = \emptyset; \\
\text{ld}(M(N)) & = \text{ld}(M) & \text{if } M \in \text{Var}; \\
\text{ld}(M(N)) & = \text{ld}(N) & \text{if } \text{ld}(M) = 0 \text{ and } \text{ld}(N) \in \text{Var}; \\
\text{ld}(M(N)) & = 0 & \text{if } \text{ld}(M) = 0 \text{ and } \text{ld}(N) \in \text{Num}; \\
\text{ld}(M(N)) & = 0 & \text{if } \text{ld}(M) = 0 \text{ and } \text{ld}(N) = \emptyset; \\
\text{ld}(M(N)) & = n - 1 & \text{if } \text{ld}(M) = n > 0; \\
\text{ld}(M(N)) & = 0 & \text{if } \text{ld}(M) = \emptyset.
\end{align*}
\]

Given a term \( M; M \text{ is in } \text{GHNF} \text{ if } \text{ld}(M) \neq \emptyset; M \text{ is rigid if } \text{ld}(M) \text{ is a variable; } M \text{ is flexible if } \text{ld}(M) \text{ is a number; } M \text{ is indeterminate if } \text{ld}(M) = \emptyset; \text{ we say that } M \text{ has a general head } \text{ld}(N) \text{ if } M \rightarrow_{\beta^*_{hd}} N \text{ and } \text{ld}(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 $\lambda$-abstractions are closed under substitutions. This naturally yields the definitions of residuals in the call-by-value $\lambda$-calculus $\lambda_v$ [15] in which $\lambda$-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 $M \overset{R}{\rightarrow}_\beta N$ for some $R$. Let us proceed by a structural induction on $M$.

- $M = \lambda x. M_1$. Then $M_1$ is in GHNF, and $M_1 \overset{R}{\rightarrow}_\beta N_1$, 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 $N = M_{11}\{x := M_2\}$; $\text{hd}(M_{11}) \neq 0$ 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 $M \overset{L}{\rightarrow}_\beta \text{lhnf}(M)$ for some term $\text{lhnf}(M)$; if $\text{hd}(M) = x$ then $\text{lhnf}(M)$ is of form $\lambda \tilde{x}.x(\tilde{N})$, where $x$ does not occur in $\tilde{x}$; if $\text{hd}(M) = n$ then $\text{lhnf}(M)$ is of form $\lambda \tilde{x}\lambda x.\lambda y.x(\tilde{N})$, where $|\tilde{x}| = n$.

**Proof** Given a term $M$, we proceed by a structural induction to show the construction of $\text{lhnf}(M)$. 8
• $M$ is a variable. Then $\operatorname{hnf}(M) = M$.

• $M = \lambda x. M_1$. Then $\operatorname{hnf}(M) = \lambda x. \operatorname{hnf}(M_1)$. Since $M_1 \xrightarrow{\beta} \operatorname{hnf}(M_1)$ by induction hypothesis, $M \xrightarrow{\beta} \operatorname{hnf}(M)$.

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

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

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

**Definition 4.6 (General Body)** Let function $\operatorname{gbd}$ be defined on $\lambda$-terms $M$ with $\operatorname{hd}(M) \geq 0$.

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

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

**Definition 4.7 ($\beta_{hd}^*$-Redex)** A $\beta_{hd}^*$-redex is a term of form $\lambda x. N$ where $0 = \operatorname{hd}(M) \leq \operatorname{hd}(N)$, and $\beta_{hd}^*\{x := N\}$ is the contractum of the $\beta_{hd}^*$-redex; we write $M_1 \xrightarrow{\beta_{hd}^*} M_2$ if $M_2$ is obtained by replacing a $\beta_{hd}^*$-redex in $M_1$ with its contractum. $\beta_{hd}^*$-reduction sequence is a possibly infinite sequence of form $M_1 \xrightarrow{\beta_{hd}^*} M_2 \xrightarrow{\beta_{hd}^*} \ldots$

The explicit notation of $\beta_{hd}^*$-reduction sequences can be defined accordingly.

**Proposition 4.8** Given a term $M$ in GNHF and $M \xrightarrow{\beta_{hd}^*} N$; then $N$ in GNHF and $\operatorname{hd}(M) = \operatorname{hd}(N)$.

**Proof** This follows from a structural induction on $M$.

Hence, this partially justifies why a term in GNHF can be regarded as a sort of head normal form under $\beta_{hd}^*$-reduction.
Proposition 4.9 Given $M$ with $\text{hd}(M) \geq 0$, and $N = \text{hfn}(M)$; then $ML \rightarrow_{\beta} \beta(N, L)$ for every term $L$.

Proof By Proposition 4.5, $M \rightarrow_{\beta} N$ and $NL$ is a $\beta$-redex. Hence, $ML \rightarrow_{\beta} \beta(N, L)$.

Proposition 4.10 Given $M$ with $\text{hd}(M) \geq 0$, and $N = \text{hfn}(M)$; then $\beta_{\text{hd}}(M, L) \rightarrow_{\beta} \beta(N, L)$ 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 $\beta_{\text{hd}}(M, L)$ is undefined otherwise.

Corollary 4.11 Given a term $L$; if $L \xrightarrow{R} R_{\text{hd}} L_1$ then $L \rightarrow_{\beta} N$ and $L_1 \rightarrow_{\beta} N$ 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 $\beta_{\text{hd}}^v$-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 $\beta_{\text{hd}}^v$-redex.

Now we are ready to introduce the concept of residuals of $\beta_{\text{hd}}^v$-redexes under $\beta_{\text{hd}}$-reduction. Let $\mathcal{R}$ be a set of $\beta_{\text{hd}}^v$-redexes in a term $M$, $R = M_1(M_2) \in \mathcal{R}$ and $M \xrightarrow{R} \beta_{\text{hd}} R$. This $\beta_{\text{hd}}^v$-reduction step affects $\beta_{\text{hd}}$-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 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 gbd$(M_1)$, then $R''\{\bullet := M_2\}$ is the only residual of $R'$. This residual is a $\beta_{\text{hd}}^v$-redex by Corollary 4.3.
- $R'$ contains $R$. Then $R' \xrightarrow{R} \beta_{\text{hd}} R''$ for some $R''$ in $N$, and $R''$ is the only residual of $R$ in $N$. Proposition 4.8 implies that $R''$ is a $\beta_{\text{hd}}^v$-redex.
- $R'$ is disjoint with $R$. Then $R'$ is its own residual in $N$.

Note that we can also define $\beta_{\text{hd}}^v$-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 GNF or a variable. One disadvantage of adopting this definition is that the notion of residuals of $\beta_{\text{hd}}^v$-redexes under $\beta$-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 $\beta_{\text{hd}}^v$-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 $\lambda$-calculus $\lambda^v_{\text{bd}}$

In this section we present a call-by-value $\lambda$-calculus $\lambda^v_{\text{bd}}$ in which values are defined as terms in flexible GNHF. We show that $\lambda^v_{\text{bd}}$ is Church-Rosser and enjoys a version of normalisation theorem and a version of standardisation theorem.

$\lambda^v_{\text{bd}}$ studies $\beta^v_{\text{bd}}$-reduction. $\lambda^v_{\text{bd}} \vdash M \Rightarrow^*_{\beta^v_{\text{bd}}} N$ if there exists $M = M_0, M_1, \ldots, M_{2n-1}, M_n = N$ such that $M_{2i-1} \Rightarrow^*_{\beta^v_{\text{bd}}} M_{2i-2}$ and $M_{2i-1} \Rightarrow^*_{\beta^v_{\text{bd}}} M_{2i}$ for $1 \leq i \leq n$.

**Definition 5.1 ($\beta^v_{\text{bd}}$-development)** Given a term $M$ and a set $\mathcal{R}$ of $\beta^v_{\text{bd}}$-redexes in $M$; if $M \Rightarrow^*_{\beta^v_{\text{bd}}} N$ contracts only the $\beta^v_{\text{bd}}$-redexes in $\mathcal{R}$ or their residuals, then $\sigma$ is a $\beta^v_{\text{bd}}$-development (of $\mathcal{R}$).

Like $\beta$-development, $\beta^v_{\text{bd}}$-development enjoys the following property.

**Lemma 5.2 (Separation)** Given a $\beta^v_{\text{bd}}$-redex $M = M_1 M_2$; for every $\beta^v_{\text{bd}}$-development $\sigma$ from $M$ in which $M$ is involved, $\sigma(M)$ is of form $\sigma_1(\text{gbd}(M_1))[\sigma_2(M_2), \ldots, \sigma_{2n}(M_2)]$, where $\sigma_1$ is a $\beta^v_{\text{bd}}$-development from $\text{gbd}(M_1)$ and $\sigma_{2i}$ are $\beta^v_{\text{bd}}$-developments from $M_2$ for $i = 1, \ldots, n$; let $\text{sp}(\sigma)$ stand for $[M] + \sigma_1 \{\bullet := N\} + \sigma_{21} + \cdots + \sigma_{2n}$, then for every $R$ if $R$ is involved in $\text{sp}(\sigma)$ then $R$ is involved in $\sigma$.

**Proof** See the proof of Lemma 3.2. \(\blacksquare\)

The next lemma explains why $\lambda^v_{\text{bd}}$ enjoys the Church-Rosser property.

**Lemma 5.3** Given $\beta^v_{\text{bd}}$-developments $M \Rightarrow^*_{\beta^v_{\text{bd}}} M_1$ and $M \Rightarrow^*_{\beta^v_{\text{bd}}} M_2$; then there exist $\beta^v_{\text{bd}}$-developments $\tau_1$ and $\tau_2$ 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 $\beta^v_{\text{bd}}$-redex. For $i = 1, 2$, we can assume $\sigma_i = \sigma^1_i + \sigma^2_i$, where $\sigma^1_i$ are $\beta^v_{\text{bd}}$-developments from $M^1$ and $\sigma^2_i$ are $\beta^v_{\text{bd}}$-developments from $M^2$. By induction hypothesis, there exist $\tau^1_i$ and $\tau^2_i$ for $i = 1, 2$ such that $(\sigma^1_i + \tau^1_i)(M^1) = (\sigma^2_i + \tau^2_i)(M^1)$ and $(\sigma^1_i + \tau^1_i)(M^2) = (\sigma^2_i + \tau^2_i)(M^2)$. 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 $\beta_{hd}^\varepsilon$-redex. For $i = 1, 2$, we may assume that $\beta_{hd}^\varepsilon$-redex $M$ is involved in $\sigma_i$ since we can simply reduce its residual in $\sigma_i(M)$ otherwise. By Lemma 5.2, $\sigma_i(M)$ are of form $\sigma_i(\text{gbd}(M^1))[\sigma_{21}^i(M^2), \ldots, \sigma_{2n}^i(M^2)]$ for $i = 1, 2$, where $\sigma_{2i}^i$ are developments from $\text{gbd}(M^1)$ and $\sigma_{2i}^i, \ldots, \sigma_{2n_i}^i$ are developments from $M^2$. By the induction hypothesis on $M^2$, there exist $\tau_{21}^i, \ldots, \tau_{2n_i}^i$ for $i = 1, 2$ such that $(\sigma_{2j}^1 + \tau_{2j}^1) = N_2$ for $1 \leq j \leq n_2$, where $N_2$ is some term. By the induction hypothesis on $\text{gbd}(M^1)$, there exist $\tau_{2i}^i$ for $i = 1, 2$ such that $(\sigma_{2i}^1 + \tau_{2i}^1)(\text{gbd}(M^1)) = (\sigma_{2i}^2 + \tau_{2i}^2)(\text{gbd}(M^1)) = N_1$, where $N_1$ is some term. Let 

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

for $i = 1, 2$, then 

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

Therefore, $\beta_{hd}^\varepsilon$-development enjoys Church-Rosser property.

We say that a $\beta_{hd}^\varepsilon$-development $\sigma : M \rightarrow^{\beta_{hd}^\varepsilon} N$ of $\mathcal{R}$ is complete if $N$ contains no residuals of any $\beta_{hd}^\varepsilon$-redexes in $\mathcal{R}$. For those who know the notion of parallel reduction, it is easy to observe that a parallel $\beta_{hd}^\varepsilon$-reduction is always a complete $\beta_{hd}^\varepsilon$-development. A version of Lemma 5.3 in which $\beta_{hd}^\varepsilon$-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 $M \rightarrow^{\beta_{hd}^\varepsilon} M_1$ and $M \rightarrow^{\beta_{hd}^\varepsilon} M_2$; then there exists $N$ such that $M_1 \rightarrow^{\beta_{hd}^\varepsilon} N$ and $M_2 \rightarrow^{\beta_{hd}^\varepsilon} N$.

**Proof** This is a corollary of Lemma 5.3 since $\rightarrow^{\beta_{hd}^\varepsilon}$ is a transitive closure of $\beta_{hd}^\varepsilon$-developments.

**Corollary 5.5** Given $M \rightarrow^{\beta_{hd}^\varepsilon} M_i$ 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 $M_1 \rightarrow^{\beta_{hd}^\varepsilon} N$ and $M_2 \rightarrow^{\beta_{hd}^\varepsilon} N$. 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 $M \rightarrow^{\beta_{hd}^\varepsilon} N$ 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 $\lambda_{hd}^\varepsilon$ 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 $\mathcal{R}_{hd}^\varepsilon(M)$ be the set of all $\beta_{hd}^\varepsilon$-redexes in $M$ for every term $M$; a relation
\( \mathbf{\vartriangle}_{hd}(M) \) on \( \mathcal{R}_{hd}^v(M) \) is defined as follows.

\[
\begin{align*}
\mathbf{\vartriangle}_{hd}(M) & = \emptyset \quad \text{if } M \text{ is a variable;} \\
\mathbf{\vartriangle}_{hd}(\lambda x.M) & = \mathbf{\vartriangle}_{hd}(M) \\
\mathbf{\vartriangle}_{hd}(M(N)) & = \mathbf{\vartriangle}_{hd}(M) \cup \mathbf{\vartriangle}_{hd}(N) \cup \left( \mathcal{R}_{hd}^v(N) \times \mathcal{R}_{hd}^v(M) \right) \cup \{ (M(N), L) : L \in \mathcal{R}_{hd}^v(M) \times \mathcal{R}_{hd}^v(N) \} \quad \text{if } M(N) \text{ is a } \beta_{hd}^v \text{-redex;} \\
\mathbf{\vartriangle}_{hd}(M(N)) & = \mathbf{\vartriangle}_{hd}(M) \cup \mathbf{\vartriangle}_{hd}(N) \cup \left( \mathcal{R}_{hd}^v(N) \times \mathcal{R}_{hd}^v(M) \right) \quad \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) \not\in \text{Num}; \\
\mathbf{\vartriangle}_{hd}(M(N)) & = \mathbf{\vartriangle}_{hd}(M) \cup \mathbf{\vartriangle}_{hd}(N) \cup \left( \mathcal{R}_{hd}^v(M) \times \mathcal{R}_{hd}^v(N) \right) \quad \text{if } \text{hd}(M) \neq 0.
\end{align*}
\]

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

**Definition 5.7** Given a \( \beta_{hd}^v \)-development \( \sigma \)

\[
M_0 \xrightarrow{R_1} \beta_{hd}^v M_1 \xrightarrow{R_2} \beta_{hd}^v \cdots \xrightarrow{R_n} \beta_{hd}^v M_n; 
\]

\( \sigma \) is canonical if there exist no \( 1 \leq i < j \leq n \) such that \( R_j \) is a residual of some \( \beta_{hd}^v \)-redex \( R \) containing \( R_i \); \( \sigma \) is standard if there exist no \( 1 \leq i < j \leq n \) such that \( R_j \) is a residual of some \( \beta_{hd}^v \)-redex \( R \) with \( R \mathbf{\vartriangle}_{hd}^v R_i \) in \( M_{i-1} \).

Like the case of \( \beta \)-development, we can readily verify that a canonical \( \beta_{hd}^v \)-development can also be rearranged into a corresponding standard \( \beta_{hd}^v \)-developments.

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

\[
\text{std}_{hd}^v(\sigma) : M \rightarrow_{\beta_{hd}^v} N
\]

for every \( \beta_{hd}^v \)-development \( \sigma : M \rightarrow_{\beta_{hd}^v} N \).

**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 \( \lambda_{hd}^v \); we prove that the strategy is normalising which always contracts the \( \mathbf{\vartriangle}_{hd}^v \)-first \( \beta_{hd}^v \)-redexes in terms; we then define an evaluation function which always evaluates a term \( M \) to a term in GHNF if \( \lambda_{hd}^v \vdash M \Rightarrow_{\beta_{hd}^v} N \) for some \( N \) in GHNF.

If \( M \) is indeterminate then we call its \( \mathbf{\vartriangle}_{hd}^v \)-first \( \beta_{hd}^v \)-redexes the main \( \beta_{hd}^v \)-redex in \( M \). We present an equivalent definition of main \( \beta_{hd}^v \)-redexes.

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

\[
\begin{align*}
R_{hd}^v(M) & = M \quad \text{if } M \text{ is a } \beta_{hd}^v \text{-redex;} \\
R_{hd}^v(M) & = R_{hd}^v(M_1) \quad \text{if } M = \lambda x. M_1 \text{ and } \text{hd}(M_1) = 0; \\
R_{hd}^v(M) & = R_{hd}^v(M_1) \quad \text{if } M = M_1 M_2 \text{ and } \text{hd}(M_1) = 0; \\
R_{hd}^v(M) & = R_{hd}^v(M_2) \quad \text{if } M = M_1 M_2 \text{ and } \text{hd}(M_1) = 0 \text{ and } \text{hd}(M_2) = 0;
\end{align*}
\]
Clearly, $R^v_{h_d}(M)$ is well-defined on every indeterminate term $M$. It is a routine verification that $R^v_{h_d}(M)$ is the $<$-first in $M$.

**Proposition 5.10** Given $M$ with $\text{hd}(M) = \emptyset$, $R = R^v_{h_d}(M)$ and $M \rightarrow^v_{\beta^v_{h_d}} N$ for some $R_1 \neq R$; then $\text{hd}(N) = \emptyset$ and $R$ has only one residual in $N$ which is $R^v_{h_d}(N)$.

**Proof** This follows from a structural induction on $M$.

**Definition 5.11** If $M \rightarrow^v_{\beta^v_{h_d}} N$ and $R = R^v_{h_d}(M)$ then we write $M \rightarrow^v_{\beta^v_{h_d}} M N$. A $\beta^v_{h_d}$-normalising sequence is a possibly infinite sequence of the following form.

$$M = M_0 \rightarrow^v_{\beta^v_{h_d}} M_1 \rightarrow^v_{\beta^v_{h_d}} \cdots$$

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

Clearly, if $\nu(M)$ is finite then $\nu(M) : M \rightarrow^v_{\beta^v_{h_d}} N$ terminates with some $N$ in GHNF.

**Lemma 5.12** Given a $\beta^v_{h_d}$-development $M \rightarrow^v_{\beta^v_{h_d}} N$; 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 $(|\nu(N)|, |\sigma|)$, lexicographically ordered. If $M$ is in GHNF then $\nu(M) = \emptyset$. Now we assume $\text{hd}(M) = \emptyset$ and $M \rightarrow^v_{\beta^v_{h_d}} M_1$ where $R = R^v_{h_d}(M)$.

- $R$ is involved in $\sigma$. Since $\sigma$ is standard, $\sigma = [R] + \sigma_1$ for some standard $\beta^v_{h_d}$-development $\sigma_1$. $\nu(M_1)$ 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 $\sigma$. By Proposition 5.10, $R$ has only one residual $R_1$ in $N$, which is $R^v_{h_d}(N)$.

Let $N \rightarrow^v_{\beta^v_{h_d}} N_1$, then $M_1 \rightarrow^v_{\beta^v_{h_d}} N_1$. 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** ($\beta^v_{h_d}$-normalisation) Given a term $M$; if $M \rightarrow^v_{\beta^v_{h_d}} N$ 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 $M \rightarrow^v_{\beta^v_{h_d}} M_1$. 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 $\lambda_{hd} \vdash M =_{\beta_{hd}} N$, where $N$ is in GHNF; then $M \xrightarrow{\nu} N_1$ for some $N_1$ and $\text{hd}(N) = \text{hd}(N_1)$.

Proof Since $\lambda_{hd} \vdash M =_{\beta_{hd}} N$, $M \xrightarrow{\beta_{hd}} N_2$ and $N \xrightarrow{\beta_{hd}} N_2$ 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., $M \xrightarrow{\nu} N_1$ for some $N_1$ in GHNF. By Corollary 5.5, $\text{hd}(N) = \text{hd}(N_1)$. ■

Definition 5.15 ($\beta_{hd}$-evaluation function)

$\text{eval}_{hd}(M) = M$ if $\text{hd}(M) \neq \emptyset$;

$\text{eval}_{hd}(\lambda x.M) = (\lambda x.\text{eval}_{hd}(M))$ if $\text{hd}(M) = \emptyset$;

$\text{eval}_{hd}(M(N)) = \text{eval}_{hd}(\beta_{hd}(M, N))$ if $\text{hd}(M) = 0$ and $\text{hd}(N) \geq 0$;

$\text{eval}_{hd}(\text{eval}_{hd}(M(N))) = \text{eval}_{hd}(\text{eval}_{hd}(M)(\text{eval}_{hd}(N)))$ if $\text{hd}(M) = 0$ and $\text{hd}(N) = 0$;

$\text{eval}_{hd}(M(N)) = \text{eval}_{hd}(\text{eval}_{hd}(M)(\text{eval}_{hd}(N)))$ if $\text{hd}(M) = 0$;

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

1. $M \beta_{hd}$-evaluates to $M$ at time 0 if $\text{hd}(M) \neq \emptyset$.

2. $\lambda x.M \beta_{hd}$-evaluates to $\lambda x.N$ at time $t$ if $M \beta_{hd}$-evaluates to $N$ at time $t$.

3. If $M \beta_{hd}$-evaluates to $M'$ at time $t$ where $\text{hd}(M') \neq 0$, then $M(N) \beta_{hd}$-evaluates to $M'(N)$ at time $t$.

4. If $M \beta_{hd}$-evaluates to $M'$ at time $t$ where $\text{hd}(M') = 0$ and $N \beta_{hd}$-evaluates to $N'$ at time $t'$ where $\text{hd}(N') \in \text{Var}$, then $M(N) \beta_{hd}$-evaluates to $M'(N')$ at time $t + t'$.

5. If $M \beta_{hd}$-evaluates to $M'$ at time $t$ where $\text{hd}(M') = 0$, $N \beta_{hd}$-evaluates to $N'$ at time $t'$ where $\text{hd}(N') \geq 0$, and $\beta_{hd}(M', N') \beta_{hd}$-evaluates to $L$ are time $t''$, then $M(N) \beta_{hd}$-evaluates to $L$ are time $t + t' + t'' + 1$.

The following correspondence is obvious.

Proposition 5.16 $M \beta_{hd}$-evaluates to $N$ at time $t$ if and only if $\nu(M) : M \xrightarrow{\beta_{hd}} N$ and $|\nu(M)| = t$.

Proof This follows from an induction on $t$. ■

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

For the sake of completeness, we also prove a version of standardisation theorem in $\lambda_{hd}$, which implies the $\beta_{hd}$-normalisation theorem.

Definition 5.17 Given a $\beta_{hd}$-reduction sequence $\sigma$

$M_0 \xrightarrow{R_1} \beta_{hd}M_1 \xrightarrow{R_2} \beta_{hd} \cdots \xrightarrow{R_n} \beta_{hd}M_n$;

15
σ is standard if there exist no 1 \leq i < j \leq n such that \( R_j \) is a residual of some \( \beta_{hd}^e \)-redex \( R \) with \( R \prec R_i \) in \( M_{i-1} \).

Proposition 5.10 implies that the residuals of main \( \beta_{hd}^e \)-redexes are always main \( \beta_{hd}^e \)-redexes. This can be extended as follows.

**Proposition 5.18** Given \( R_1 \prec R_2 \) in \( M \) and \( M \overset{R_3}{\rightarrow} N \); then \( R_1 \) has only one residual \( R_1' \) in \( N \); if \( R' \) is not a residual of some redex in \( M \), then \( R_1' \prec R_i \) in \( N \); if \( R' \) is a residual of some redex \( R \) in \( M \) with \( R_1 \prec R_{i1} \), then \( R_1' \prec R_{i1} \) in \( N \).

**Proof** This simply follows from a structural induction on \( M \).

**Lemma 5.19** Given \( \sigma + \tau : \text{from } M \), where \( \sigma \) is a standard \( \beta_{hd}^e \)-development and \( \tau \) is a standard \( \beta_{hd}^e \)-reduction sequence; then there exists a standard \( \beta_{hd}^e \)-reduction sequence

\[
S(\sigma, \tau) : M \rightarrow \beta_{hd}^e(\sigma + \tau)(M)
\]

such that for every \( R \) if \( R \) is involved in \( S(\sigma, \tau) \) 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 \( S(\sigma, \emptyset) = \sigma \).
- \( \sigma = \emptyset \). Then let \( S(\emptyset, \tau) = \tau \).
- \( \sigma = [R_1] + \sigma_1 \) and \( \tau = [R_2] + \tau_1 \). Now we have two subcases.
  1. \( R_2 \) is a residual of some redex in \( M \). Then let \( S(\sigma, \tau) = S(\text{std}_{hd}^e(\sigma + [R_2]), \tau_1) \). Note that for every \( R \) if \( R \) is involved in \( \text{std}_{hd}^e(\sigma + [R_2]) \) then \( R \) is involved in \( \sigma + [R_2] \). By induction hypothesis, \( R \) is involved in \( \text{std}_{hd}^e(\sigma + [R_2]) + \tau \) if \( R \) is involved in \( S(\text{std}_{hd}^e(\sigma + [R_2]), \tau_1) \). Hence, with Lemma 5.8, \( R \) is involved in \( \sigma + \tau \) if \( R \) is involved in \( S(\sigma, \tau) \).
  2. \( R_2 \) is not a residual of any redex in \( M \). Then let \( S(\sigma, \tau) = [R_1] + S(\sigma_1, \tau) \). By induction hypothesis, for every \( R \) if \( R \) is involved in \( S(\sigma_1, \tau) \) then \( R \) is involved in \( \sigma_1 + \tau \). Hence, if \( R \) is involved in \( S(\sigma, \tau) \) then \( R \) is involved in \( \sigma + \tau \). Let us assume \( R \prec Q_{hd}^e \) in \( M \); then \( R \) is not involved in \( \sigma \) since \( \sigma \) is standard; by Proposition 5.18, \( R \) has only one residual \( R' \) in \( \sigma(M) \) and \( R' \prec Q_{hd}^e R_2 \) in \( \sigma(M) \); then \( R' \) is not involved in \( \tau \) since \( \tau \) is standard; therefore, \( R \) is not involved in \( \sigma + \tau \), and this yields that \( S(\sigma, \tau) \) is standard since \( S(\sigma_1, \tau) \) is standard by induction hypothesis.

**Theorem 5.20** (\( \beta_{hd}^e \)-standardisation) Given a \( \beta_{hd}^e \)-reduction sequence \( \sigma : M \rightarrow \beta_{hd}^e N \); then there exists a standard \( \beta_{hd}^e \)-reduction sequence \( \text{std}_{hd}^e(\sigma) : M \rightarrow \beta_{hd}^e N \).

**Proof** We proceed by induction on \( |\sigma| \). If \( \sigma = \emptyset \) then \( \text{std}_{hd}^e(\sigma) = \emptyset \); if \( \sigma = [R] + \sigma_1 \) then \( \text{std}_{hd}^e(\sigma) = S([R], \text{std}_{hd}^e(\sigma_1)) \), which is standard by Lemma 5.19.

16
6. Relations between $\lambda_{hd}^v$ and $\lambda$

We prove that a term can be reduced in $\lambda_{hd}^v$ to a term in GHNF if and only if it can be reduced in the $\lambda$ 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 $M \rightarrow_{h}^{h} N_2$ for some $N_2$ in HNF.

We can then define an evaluation function corresponding to head $\beta$-reduction.

**Definition 6.2** ($\beta$-evaluation function)

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

We define predicate $M$ $\beta$-evaluates to $N$ at time $t$ similarly; with Proposition 2.5 (1), we have that $M$ $\beta$-evaluates to $N$ at time $t$ if and only if $M \rightarrow_{h}^{h} N$ with $N$ in HNF.

**Lemma 6.3** Given a term $L$; if $L \beta_{hd}^v$-evaluates to $M$, then $L \rightarrow_\beta N$ for some $N$ in HNF and $\text{hd}(N) = \text{hd}(M)$.

**Proof** Assume $L \beta_{hd}^v$-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 $L \rightarrow_{\beta_{hd}}^{\beta_{hd}} L_1$ and $L_1 \beta_{hd}^v$-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 $\lambda$ is Church-Rosser, and this yields $L \rightarrow_\beta N$. Clearly, $\text{hd}(N) = \text{hd}(N_1) = \text{hd}(M)$.

**Lemma 6.4** Given terms $M, M_1, N$, where

\[M \rightarrow_{\beta_{hd}}^{m} M_1 \text{ and } M \rightarrow_{h}^{h} N;\]

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

**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 $M^2 \rightarrow_{h}^{h} N^2$, and
\[ N = \beta(M^1, M^2); \text{ thus, we have } M_1 \xrightarrow{\beta} N_1 \text{ for } N_1 = \beta(M^1, N^2); \text{ it can be verified that a } R_e \text{ in some occurrence of } M^2 \text{ in } N \text{ is the main } \beta^v_{hd} \text{-redex in } N \text{ since } \text{hd}(M^1) = 0; \text{ we can contract the } R_e \text{ in this occurrence of } M^2 \text{ in } N \text{ to } N^2; \text{ therefore, } N \xrightarrow{\beta^v_{hd}} N' \xrightarrow{\beta^v_{hd}} N_1. \]

**Lemma 6.5** Given terms \( M, N \), where \( M \xrightarrow{\beta} N \) and \( N \xrightarrow{\beta^v_{hd}} \) evaluates to a term at time \( t \); then \( M \xrightarrow{\beta^v_{hd}} \) 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 \( \beta^v_{hd} \)-redex, then \( R \) is the main \( \beta^v_{hd} \)-redex in \( M \) and this becomes trivial. Now we assume that \( M \xrightarrow{\beta^v_{hd}} M_1 \) and \( R \) is not a \( \beta^v_{hd} \)-redex. Then \( M_1 \xrightarrow{\beta} N_1 \) and \( N_1 \xrightarrow{\beta^v_{hd}} N' \xrightarrow{\beta^v_{hd}} N_1 \) for some \( N_1 \) by Lemma 6.4. Since \( N' \xrightarrow{\beta^v_{hd}} \) evaluates to a term at time \( t - 1 \), \( N_1 \xrightarrow{\beta^v_{hd}} \) 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, \( M_1 \xrightarrow{\beta^v_{hd}} \) evaluates to \( L \) for some \( L \) in GHNF at some time \( t'_1 \leq t_1 + 1 \). Hence \( M \xrightarrow{\beta^v_{hd}} \) evaluates to \( L \) at time \( t'_1 + 1 \leq t_1 + 2 \leq t + 1 \).

**Corollary 6.6** Given a term \( M \) which \( \beta \)-evaluates to a term at time \( t \); then \( M \xrightarrow{\beta^v_{hd}} \) 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, \( \text{eval}^v_{hd} \) never takes more time to terminate than \( \text{eval} \) does.

**Theorem 6.7** Given a term \( M \); \( \text{eval}(M) \) is defined if and only if \( \text{eval}^v_{hd}(M) \) is defined.

**Proof** If \( \text{eval}^v_{hd}(M) \) is defined then \( M \xrightarrow{\#} N_1 \) for some \( N_1 \) in HNF by Lemma 6.3, which implies that \( \text{eval}(M) \) is defined. If \( \text{eval}(M) \) is defined then \( \text{eval}^v_{hd}(M) \) is defined by Corollary 6.6.

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

### 7. Operational Semantics

This section presents an operational semantics for \( \lambda^v_{hd} \) 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. \((\text{CL} : S, E, \emptyset, \langle S', E', C', D' \rangle) \Rightarrow (\text{CL} : S', E', C', D')\)
2. \((S, E, x : C, D) \Rightarrow (E(x)/E : S, E, C, D)\)
3. \((S, E, (\lambda x.B) : C, D) \Rightarrow (S, E, B : \lambda x : C, D)\)
4. \((S, E, \circ (B, B') : C, D) \Rightarrow (S, E, B : \circ B' : C, D)\)
5. \((S, E, \circ (\text{CL}, B) : C, D) \Rightarrow (\text{CL}/E : S, E, \circ B : C, D)\)
6. \((x, B, E') : S, E, \lambda x : C, D) \Rightarrow ((0, \lambda x.B \{x := \text{var}\}, E') : S, E, C, D)\)
7. \((x, B, E') : S, E, \lambda y : C, D) \Rightarrow ((x, \lambda y.B, E') : S, E, C, D)\)
8. \((n, \lambda x.B, E') : S, E, \lambda y : C, D) \Rightarrow ((n + 1, \lambda x.B, E') : S, E, C, D)\)
9. \((x, B, E') : S, E, \circ B' : C, D) \Rightarrow ((x, \circ (B, B'), E') : S, E, C, D)\)
10. \((0, \lambda x.B, E') : S, E, \circ B' : C, D) \Rightarrow ((0, \lambda x.B, E') : S, E, B' : \circ : C, D)\)
11. \((x, B, E') : \text{CL} : S, E, \circ : C, D) \Rightarrow ((x, \circ (\text{CL}, B), E') : S, E, C, D)\)
12. \((\text{CL} : 0, \lambda x.B, E') : S, E, \circ : C, D) \Rightarrow ((\emptyset, E'[x \mapsto \text{CL}], B, \circ (S, E, C, D))\)

Table 1: The SECD machine state transitions

- Given distinct variables \(x_i\) and closures \(\text{CL}_i\) for \(i = 1, \ldots, n\); the following ordered sequence \(E\)
  \[(x_1 \mapsto \text{CL}_1, \ldots, x_n \mapsto \text{CL}_n)\]
is an environment; \(\text{Dom}(E) = \{x_1, \ldots, x_n\}\); \(E = ()\) is an empty environment when \(n = 0\).

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

Given an environment \(E\):
- if \(x \notin E\) then \((x, x, E)\) is a closure;
- \((x, \circ (B, B'), E)\) is a closure for any closure body \(B'\) if \((x, B, E)\) is a closure;
- \((x, \circ (\text{CL}, B), E)\) is a closure for any closure \(\text{CL}\) with \(hd(\text{CL}) = 0\) 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;
- \((n, \circ (B, B'), E)\) is a closure for any closure body \(B'\) if \((n + 1, B, E)\) is a closure.

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

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

Given \(E = (x_1 \mapsto \text{CL}_1, \ldots, x_n \mapsto \text{CL}_n)\); 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
\[(x_1 \mapsto \text{CL}_1, \ldots, x_n \mapsto \text{CL}_n, x \mapsto \text{CL})\].
We define functions \( B_\emptyset : \text{bodies} \to \text{terms} \) and \( \text{BE} : \text{bodies} \times \text{environments} \to \text{terms} \) inductively.

\[
\begin{align*}
B_\emptyset(x) & = x \\
B_\emptyset(\@ (B, B')) & = \@ (B_\emptyset(B), B_\emptyset(B')) \\
B_\emptyset(\@ (\text{CL}, B)) & = \@ (\text{BE}(\text{bd} (\text{CL}), \text{env} (\text{CL})), B_\emptyset(B)) \\
B_\emptyset(\lambda x. B) & = \lambda x. B_\emptyset(B) \\
\text{BE}(B, E) & = [M_1/x_1] ([M_2/x_2] (\cdots ([M_n/x_n] B_\emptyset(B)) \cdots ))
\end{align*}
\]

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

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

**Lemma 7.1** Given closures \( \text{CL}_1 = (\text{hd}_1, \lambda x. B_1, E_1) \) 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:

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

We write \( \text{CL}/E' \) 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. \texttt{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) = ([], (\), [], []) ,
\]

where \( [] \) stands for an empty list; let

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

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

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 \( \text{Eval}^\text{hd} \) and \( \text{eval}^\text{hd} \).

**Theorem 7.2** \( \text{Eval}^\text{hd} (P) = \text{eval}^\text{hd} (P) \) 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 and implement it. This is a subject of future work.

8. Extensions

We first add a recursion combinator to β-hd-calculus eliminating some syntactic overhead. Then we extend β-hd-calculus with constructors and primitive functions.

8.1. Recursion

Many fixed point operators have been constructed. For instance,

\[ Y = \lambda f. (\lambda x. f(x))(\lambda x. f(x)) \]

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

\[ \text{fix}(f) \rightarrow f(\text{fix}(f)) \].

Let \( \text{hd} (\text{fix}) = 0 \); if \( \text{hd}(f) \geq 0 \) then \( \text{fix} f \) is a \( \beta^v_{hd} \)-redex and \( \beta^v_{hd} (\text{fix}, f) = f(\text{fix} f) \). Another possibility is to introduce letrec, which will be explored when we study implementations of \( \lambda^v_{hd} \).

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

\[
\begin{align*}
\text{hd}(M) &= c^n & \text{if } M = c^n(M_1, \ldots, M_n); \\
\text{hd}(M(N)) &= c^n & \text{if } \text{hd}(M) \in \text{Const}; \\
\text{hd}(M(N)) &= \text{hd}(N) & \text{if } \text{hd}(M) = 0 \text{ and } \text{hd}(N) \in \text{Const}.
\end{align*}
\]

Let \( S = \text{fix}(\lambda x. \text{cons}(0, x)) \), then \( \text{hd}(S) = \text{cons} \); so \( S \) is in GHNF. This is justified in most actual implementations which allocate only one cell for \( \text{cons} \), representing \( S \) as a cyclic data structure. Therefore, the \( \lambda^v_{hd} \)-calculus does not have the deficiency of the call-by-need λ-calculus [3], where \( S \) evaluates to \( \text{cons}(0, S) \) containing two distinct \( \text{cons} \) 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 $\Delta_n$ represent integer $n$ for $n = 0, 1, \ldots$ and $t, f$ stand for truth values. Let $\text{Int} = \{\Delta_0, \Delta_1, \ldots\}$ and $\text{Bool} = \{t, f\}$.

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 $\delta$-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 $\text{eval}_{\text{hd}}$ 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 $\langle 0, \text{Int} \rangle$; $M(\text{N})$ is regarded as a $\beta_{\text{hd}}$-redex if $M = \langle 0, \text{Int} \rangle$ and $\text{hd}(\text{N}) \in \text{Int}$. Clearly, Int can be replaced with other sets of constants. Also it is easy to see how to adjust the definition of $\text{hd}$ 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 $\delta$-redex if $\text{hd}(M), \text{hd}(N) \in \text{Int}$. We also extend the definition of $\text{eval}_{\text{hd}}$ as follows.

$$\text{eval}_{\text{hd}}(+(M, N)) = \Delta_m + \Delta_n \quad \text{if } \text{hd}(M) = \Delta_m \text{ and } \text{hd}(N) = \Delta_n;$$

$$\text{eval}_{\text{hd}}(+(M, N)) = \text{eval}_{\text{hd}}(+(\text{eval}_{\text{hd}}(M), N)) \quad \text{if } \text{hd}(M) = \emptyset;$$

$$\text{eval}_{\text{hd}}(+(M, N)) = \text{eval}_{\text{hd}}(+(M, \text{eval}_{\text{hd}}(N))) \quad \text{if } \text{hd}(M) \neq \emptyset \text{ and } \text{hd}(N) = \emptyset;$$

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.

$$\text{hd}(\text{IF}(M, N_1, N_2)) = \begin{cases} 
\emptyset & \text{if } \text{hd}(M) \in \text{Bool}; \\
\langle \text{hd}(M), \text{Bool} \rangle & \text{if } \text{hd}(M) \in \text{Var}; \\
\text{hd}(M) & \text{if } \text{hd}(M) \notin \text{Var} \cup \text{Bool}; 
\end{cases}$$
IF($M, N_1, N_2$) is a $\delta$-redex if $\text{hd}(M) \in \text{Bool}$. We extend the definition of $\text{eval}^d_{\text{hd}}$, defining

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

8.4. Potential Improvements

Let $\text{BIG}$ 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 $\text{hd}$ 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 $\text{eval}^d_{\text{hd}}$ to accommodate the change. We are currently studying on this subject.

9. Related Work

$\lambda^v_{\text{hd}}$ is closely related to the weak $\lambda$-calculus in [18] and the call-by-need $\lambda$-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 $\lambda$-abstraction; it can be readily verified that such a form is closed under value substitution if values are defined as variables or $\lambda$-abstractions; we can then define a kind of $\lambda$-calculus corresponding to such redexes; a close correspondence between this $\lambda$-calculus and the $\lambda$-calculi in [18] and [3] can be established accordingly. It is this observation that motivates the paper.

The problem of sharing evaluation under $\lambda$-abstraction has lead to a great deal of study on $\lambda$-lifting [12] and supercombinators [9] under the title full laziness or maximal laziness. Because of the ability of evaluating under $\lambda$ directly, $\text{eval}^d_{\text{hd}}$ can achieve what is beyond the scope of either $\lambda$-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

$$P = \lambda u. (\ldots u \ldots u \ldots ) (M(t)(\Delta_n)),$$

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

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

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

23
term $\text{IF}(z, x, y)$ contains variable $y$ at compile time. In some sense, full laziness is really not full when evaluation under $\lambda$ is allowed.

Another related subject is partial evaluation. We show that $\text{eval}^{\nu}_{hd}$ works well with staged computation [5]. Let us define the power function $\text{pow}$ as

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

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

\[
\begin{align*}
\text{eval}^{\nu}_{hd}(\text{pow}(\Delta_0)) &= \lambda n. (\lambda f \lambda n. \Delta_1)(\text{pow})(n) \\
\text{eval}^{\nu}_{hd}(\text{pow}(\Delta_1)) &= \lambda n. (\lambda n. \times((\lambda f \lambda n. \Delta_1)(\text{pow})(n), n))(n) \\
\text{eval}^{\nu}_{hd}(\text{pow}(\Delta_2)) &= \lambda n. (\lambda n. \times((\lambda n. \times((\lambda f \lambda n. \Delta_1)(\text{pow})(n), n))(n), n))(n)
\end{align*}
\]

This is very close to Example 2.4 in [5]. Suppose that we implement $\text{eval}^{\nu}_{hd}$ using the above presented SECD machine; if a term $\text{pow}(\Delta_0)$ 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 $\text{dotprod}$ which computes the inner product of two vectors of some given length [11], the reader can also verify that $\text{eval}^{\nu}_{hd}(\text{dotprod}(\Delta_0))$ is adequately expanded for every $\Delta_0 \in \text{Int}$. Note that $\text{vec}[n]$ yields the $n$th element in vector $\text{vec}$.

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

Also $\text{eval}^{\nu}_{hd}$ 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

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

Given any $\Delta_m \in \text{Int}$, $\text{eval}^{\nu}_{hd}(\text{acker}(\Delta_m))$ 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 $\text{eval}^{\nu}_{hd}$ 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 $\beta$-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 $\text{eval}^{\nu}_{hd}$ be a evaluation function which is able to perform some degree of on-line partial evaluation. This favors that a polished implementation of $\text{eval}^{\nu}_{hd}$ is promising to enhance the performance of lazy functional programming languages.

10. Conclusions and Future Work

We have presented a call-by-value $\lambda$-calculus $\lambda^n_{hd}$ in which values are defined as terms in flexible generalised head normal form. $\lambda^n_{hd}$ enjoys many similar properties as the $\lambda$-calculus $\lambda$ does.
Given a program $P$ which outputs base values such as integers, $\lambda \vdash P \equiv_{b} b$ for some base value $b$ if and only if $\lambda_{hd}^{b} \vdash P \equiv_{\gamma_{hd}^{b}} M$ for some $M$ in GHNF with $b$ as its general head. Therefore, lazy functional programming languages can implement $\lambda_{hd}^{b}$ without compromising their semantics. A call-by-need implementation of $\lambda_{hd}^{b}$ suggests a higher degree of sharing since evaluation can take place under $\lambda$-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 $\mathbf{eval}_{hd}^{b}$ for $\lambda_{hd}^{b}$.

We intend to extend $\lambda_{hd}^{b}$ with explicit substitutions and study approaches to implementing $\lambda_{hd}^{b}$ efficiently. We would also like to establish a $\lambda$-calculus corresponding to the usual call-by-value $\lambda$-calculus $\lambda_{c}$, in which evaluation under $\lambda$ can be performed. We believe that studies on $\lambda_{hd}^{b}$ 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


