Compilation of Head and Strong Reduction

Pascal Fradet

INRIA/IRISA

Campus de Beaulieu, 35042 Rennes Cedex, France

fradet@irisa.fr

Abstract

Functional language compilers implement only weak-head reduction. However, there are cases where head normal forms or full normal forms are needed. Here, we study how to use cps conversion for the compilation of head and strong reductions. We apply cps expressions to a special continuation so that their head or strong normal form can be obtained by the usual weak-head reduction. We remain within the functional framework and no special abstract machine is needed. Used as a preliminary step our method allows a standard compiler to evaluate under λ's.

1 Introduction

Functional language compilers consider only weak-head reduction and the evaluation stops when a weak head normal form (whnf), that is a constant or a λ-abstraction, is reached. In practice, whnf's are considered sufficient because printable results belong to basic domains. However, there are cases where one would like to reduce under λ's to get head normal forms (hnf) or even (strong) normal forms (nf). Specifically, head/strong reduction can be of interest in:

- program transformations (like partial evaluation) which need to reduce under λ's,
- higher order logic programming like λ-prolog [15] where unification involves reducing λ-terms to normal forms,
- evaluating data structures coded in λ-expressions,
- compiling more efficient evaluation strategies.

A well known tool used to compile (weak) evaluation strategies of functional programs is continuation-passing style (cps) conversion [6][16]. This program transformation makes the evaluation ordering explicit. We see it as a compiling tool since cps expressions can be reduced without any dynamic search for the next redex. Its main advantage is that it stays within the functional framework and thus does not preclude further transformations. Several compilers for strict and non-strict functional languages integrate a cps conversion as a preliminary step [1][7][11].

Here, we study how to use cps conversion for the implementation of head and strong reductions. To the best of our knowledge, the application of this transformation to such reduction strategies has not been investigated for far. A key property of cps expressions is that their (weak) evaluation is order independent: there is a unique (weak) redex at each reduction step. This property does not hold with strong or head reduction ; a cps expression may have several (strong) redexes. Our approach is to simulate head/strong reductions by weak reductions. Cps expressions are applied to special continuations so that their head/strong normal form can be obtained by the usual weak-head reduction. This way, we still use the only strategy known by compilers (weak reduction), and we retain the key property of cps. The advantage of this approach is that we do not have to introduce a special abstract machine and/or particular structures. It can be used to extend an existing compiler with head/strong reduction capabilities and it enables us to use classical implementation and optimization techniques.

In the following, we assume a basic familiarity with the λ-calculus and cps. In section 2, we introduce some notations, the definitions of the different reduction strategies and cps conversion. We consider in section 3 how to use standard cps conversion to simulate head-reduction of λ-expressions. Section 4 is devoted to strong reduction which involves a minor modification of the technique used for head reduction. In section 5, we envisage a restriction of λ-calculus with a flexible notion of typing which allows a better treatment of head reduction. Section 6 describes how this method could be used to compile more efficient reduction strategies, addresses implementation issues and discusses possible extensions.

2 Preliminaries

One of the application of head reduction being to avoid duplicated or useless computations (see section 6), we will focus on call-by-name. We consider pure λ-calculus and the global λ-expression to be reduced is always assumed to be closed. Given a reduction strategy x, $E \xrightarrow[x]{} F$ (resp. $E \xrightarrow[x]{i} F$) reads "E reduces to F after one (resp. i) reduction step by x". The transitive, reflexive closure of $\xrightarrow[x]{}$ is noted $\xrightarrow[x]{*}$. The three computation rules we are dealing with (i.e. weak head, head and strong reduction) are described in the form of deductive systems.

- Weak head reduction is noted $\xrightarrow[w]{}$ and is defined by

$$(\lambda x.E)\,F \xrightarrow[w]{} E[F/x] \qquad\qquad \frac{E \xrightarrow[w]{} E'}{E\,F \xrightarrow[w]{} E'\,F}$$

Closed whnf's are of the form λx.E.

- Head reduction is noted $\xrightarrow[h]{}$ and is defined by

$$\frac{E \xrightarrow[w]{} E'}{\lambda x_1.\ldots\lambda x_n.E \xrightarrow[h]{} \lambda x_1.\ldots\lambda x_n.E'}\ n\geq 0$$

Closed hnf's are of the form $\lambda x_1.\ldots\lambda x_n.x_i\,E_1\ldots E_p$ $(1\leq i\leq n,\ p\geq 0)$. x_i is called the head variable.

- Strong reduction is noted $\xrightarrow[s]{}$ and, with N_i's standing for normal forms, is defined by

$$\frac{E \xrightarrow[h]{} E'}{E \xrightarrow[s]{} E'} \qquad \frac{E_i \xrightarrow[s]{} E_i'}{\lambda x_1.\ldots\lambda x_n.x_i\,E_1\ldots E_i\,N_{i+1}\ldots N_p \xrightarrow[s]{} \lambda x_1.\ldots\lambda x_n.x_i\,E_1\ldots E'_i\,N_{i+1}\ldots N_p}$$

Strong reduction is described as a sequence of head reductions. When a hnf is reached, the arguments of the head variable are reduced in a right to left fashion. Closed normal forms are of the form $\lambda x_1.\ldots\lambda x_n.x_i\,N_1\ldots N_p$ with $1\leq i\leq n$ and with $N = x_j \mid \lambda x_1.\ldots\lambda x_n.x_k\,N_1\ldots N_p$

$\mathcal{N}$ stands for the standard cps conversion associated with call-by-name [16] and is defined in Figure 1. Call-by-name cps could also been defined *à la* Fischer where continuations occur first [6]. Our approach could be applied to this kind of cps expressions as well.

$$\mathcal{N}(x) = x$$

$$\mathcal{N}(\lambda x.E) = \lambda c.c\,(\lambda x.\mathcal{N}(E))$$

$$\mathcal{N}(E\,F) = \lambda c.\mathcal{N}(E)\,(\lambda f.f\,\mathcal{N}(F)\,c)$$

Figure 1 Standard Call-by-Name Cps

Variables c and f are supposed not to occur in the source term. The reduction of a cps term consists of a sequence of administrative reductions (i.e. reduction of redexes introduced by the transformation, here redexes of the form (λc.E) F or (λf.E) F), followed by a proper reduction (corresponding to a reduction of the source term), followed by administrative reductions, and so on. The relation induced by administrative reductions is noted $\overset{a}{\underset{w}{\rightarrow}}$, for example:

$$\mathcal{N}((\lambda x.E)\ F)\ \mathbf{I} \equiv (\lambda c.(\lambda c.c\ (\lambda x.\mathcal{N}(E)))\ (\lambda f.f\ \mathcal{N}(F)\ c))\ \mathbf{I} \overset{a}{\underset{w}{\rightarrow}} (\lambda x.\mathcal{N}(E))\ \mathcal{N}(F)\ \mathbf{I}$$

The following property states that evaluation of cps expressions simulates the reduction of source expressions ; it is proved in [16].

Property 1 *If E* $\overset{*}{\underset{w}{\rightarrow}}$ *W then* $\mathcal{N}(E)\ \mathbf{I} \overset{*}{\underset{w}{\rightarrow}} X \overset{a}{\underset{w}{\leftarrow}} \mathcal{N}(W)\ \mathbf{I}$ *and if W is a whnf then X is a whnf. Furthermore E does not have a whnf iff* $\mathcal{N}(E)\ \mathbf{I}$ *does not have a whnf.*

Cps conversion introduces many new λ-abstractions and affects the readability of expressions. In the remainder of the paper we use the following abbreviations

$$\lambda_c x.E \equiv \lambda c.c\ (\lambda x.E)$$

$$\lambda_c \vec{x}_n\ .\ E \equiv \lambda c.c\ (\lambda x_1. \ldots(\lambda c.c\ (\lambda x_n.E))\ldots)$$

$$\vec{X}_n\ \ E \equiv \lambda f.f\ X_1\ (\ldots(\lambda f.f\ X_n\ E)\ \ldots)$$

3 Head Reduction

Since we are interested in compiling, we consider only programs, i.e. closed expressions. A compiler does not know how to deal with free variables ; the expression to be reduced must remain closed throughout the evaluation. Furthermore, in order to use weak head reduction to evaluate hnf's, the leading λ's must be suppressed as soon as the whnf is reached. Our solution is to apply the whnf to combinators so that the associated variables are replaced with closed expressions. The head lambdas disappear, the expression remains closed and the evaluation can continue as before. After the body is reduced to hnf the expression must be reconstructed (i.e. the leading λ's must be reintroduced as well as their variables). We reach a hnf when the head variable is applied (a closed hnf is of the form $\lambda x_1. \ldots\lambda x_n.x_i\ E_1\ldots E_n$) so the combinators previously substituted for the leading variables should take care of the reconstruction process.

In general, it is not possible to know statically the number of leading λ's (sometimes called the binder length) of the hnf of an expression. We have to keep track of their number in order to eventually reintroduce them.This complicates the evaluation and reconstruction process. In section 5 we present a means of avoiding this need for counting.

We use the standard call-by-name cps conversion ($\mathcal{N}$). The global cps expression is applied to a recursive continuation Ω and an index $\overline{\mathbf{n}}$ such that $\Omega\ E\ \overline{\mathbf{n}} = E\ \mathbf{H}_n\ \Omega\ \overline{\mathbf{n+1}}$ (Ω, $\mathbf{H}_n$ and $\overline{\mathbf{n}}$ being combinators). Combinators $\overline{\mathbf{n}}$ represent the number of head abstractions already encountered. The weak head reduction of such expressions looks like

$\mathcal{N}(E)\ \Omega\ \overline{\mathbf{n}} \overset{*}{\underset{w}{\rightarrow}} (\lambda c.c\ (\lambda x.F))\ \Omega\ \overline{\mathbf{n}}$	when a cps expression E is evaluated by wh-reduction, its whnf (if any) will be of the form λc.c (λx.F)
$\underset{w}{\rightarrow} \Omega\ (\lambda x.F)\ \overline{\mathbf{n}}$	the continuation Ω is applied
$\underset{w}{\rightarrow} (\lambda x.F)\ \mathbf{H}_n\ \Omega\ \overline{\mathbf{n+1}}$	Ω applies the whnf to combinator $\mathbf{H}_n$, Ω and the new index
$\underset{w}{\rightarrow} F[\mathbf{H}_n/x]\ \Omega\ \overline{\mathbf{n+1}}$	$\mathbf{H}_n$ is substituted for x

The expression remains closed and the evaluation continues, performing the same steps if other whnf's are encountered. Eventually a hnf is reached, that is, a combinator $\mathbf{H}_i$ is in head position and this combinator is responsible for reconstructing the expression.

In fact, we do not apply the global expression directly to Ω but to combinator **A** (defined by A E F = F E) whose task is to apply the expression to Ω. This way Ω remains outside the expression and it makes its suppression during the reconstruction process easier. This technical trick is not absolutely necessary but it simplifies things when working within the pure λ-calculus. The reduction steps that occur when a whnf is reached actually are

$$(\lambda c.c\,(\lambda x.F))\ \mathbf{A}\ \Omega\ \overline{\mathbf{n}} \underset{w}{\rightarrow} \mathbf{A}\ (\lambda x.F)\ \Omega\ \overline{\mathbf{n}} \underset{w}{\rightarrow} \Omega\ (\lambda x.F)\ \overline{\mathbf{n}} \underset{w}{\rightarrow} (\lambda x.F)\ \mathbf{H}_n\ \mathbf{A}\ \Omega\ \overline{\mathbf{n+1}}$$

If E is a closed expression, its transformed form will be $\mathcal{N}(E)\ \mathbf{A}\ \Omega\ \overline{\mathbf{0}}$ with

$$\mathbf{A}\,\mathbf{M}\,\mathbf{N} \underset{w}{\rightarrow} \mathbf{N}\,\mathbf{M} \tag{A}$$

$$\Omega\,\mathbf{M}\,\overline{\mathbf{n}} \overset{*}{\underset{w}{\rightarrow}} \mathbf{M}\,\mathbf{H}_n\,\mathbf{A}\,\Omega\,\overline{\mathbf{n+1}} \tag{Ω}$$

The family of combinators $\mathbf{H}_i$ is defined by

$$\mathbf{H}_i\ \mathbf{M}\ \mathbf{N}\ \overline{\mathbf{n}} = \lambda_c\vec{x}_n\ .\ \lambda c.x_{i+1}\ (\mathbf{M}\ (\mathbf{R}\ \overline{\mathbf{n}}\ (\lambda c.\ \vec{x}_n\ \ c))\ (\mathbf{K}\ c)) \tag{H}$$

with
$$\mathbf{R}\,\mathbf{E}\,\mathbf{F}\,\mathbf{G}\,\mathbf{H}\,\mathbf{I} \underset{w}{\rightarrow} \lambda f.f\ (\lambda c.\mathbf{G}\ \mathbf{A}\ \Omega\ \mathbf{E}\ (\mathbf{F}\ c))\ (\mathbf{H}\ (\mathbf{R}\ \mathbf{E}\ \mathbf{F})\ \mathbf{I}) \tag{R}$$

The definitions **(H)** and **(R)** can be explained intuitively as follows. When the hnf is reached the expression is of the form $\mathbf{H}_i\ (\lambda f.f\ E_1 \ldots (\lambda f.f\ E_m\ \mathbf{A})\ldots)\ \Omega\ \overline{\mathbf{n}}$, $\overline{\mathbf{n}}$ representing the number of head abstractions of the hnf. The reduction rule of $\mathbf{H}_i$ deletes Ω, reintroduces the n leading λ's, the head variable and yields

$$\lambda_c\vec{x}_n\ .\ \lambda c.x_{i+1}\ ((\lambda f.f\ E_1 \ldots (\lambda f.f\ E_m\ \mathbf{A})\ldots)\ (\mathbf{R}\ \overline{\mathbf{n}}\ (\lambda c.\vec{x}_n\ \ c))\ (\mathbf{K}\ c))$$

Some $\mathbf{H}_i$'s may remain in the continuation of x_{i+1} and the role of **R** is to remove them by applying each E_i to suitable arguments. The reconstructing expression $\mathbf{R}\ \overline{\mathbf{n}}\ (\lambda c.\vec{x}_n\ \ c)$ will be recursively called by the argument "list" $(\lambda f.f\ E_1 \ldots (\lambda f.f\ E_m\ \mathbf{A})\ldots)$; the final continuation **A** will call **K** which removes $\mathbf{R}\ \overline{\mathbf{n}}\ (\lambda c.\vec{x}_n\ \ c)$. Meanwhile **R** applies each argument E_i to **A**, Ω, $\overline{\mathbf{n}}$, $(\vec{x}_n\ \ c)$ and reconstructs the argument "list". In summary

$$(\lambda f.f\ E_1 \ldots (\lambda f.f\ E_m \mathbf{A})\ldots)\ (\mathbf{R}\ \overline{\mathbf{n}}\ (\lambda c.\vec{x}_n\ \ c))\ (\mathbf{K}\ c) \overset{*}{\rightarrow} (\lambda f.f\ (\lambda c.E_1\ \mathbf{A}\ \Omega\ \overline{\mathbf{n}}\ (\vec{x}_n\ \ c))\ldots$$
$$\ldots(\lambda f.f\ (\lambda c.E_m\ \mathbf{A}\ \Omega\ \overline{\mathbf{n}}\ (\vec{x}_n\ \ c))\ c)\ldots)$$

Each E_i corresponds to an original cps expression F_i containing at most n free variables such that $E_i \equiv F_i[\vec{x}_n\ /\vec{\mathbf{H}}_n\ ]$. Let N_i be the normal form of F_i then

$$\lambda c.E_i\ \mathbf{A}\ \Omega\ \overline{\mathbf{n}}\ (\vec{x}_n\ \ c)\ = \lambda c.F_i[\vec{x}_n\ /\vec{\mathbf{H}}_n\ ]\ \mathbf{A}\ \Omega\ \overline{\mathbf{n}}\ (\vec{x}_n\ \ c) = \lambda c.(\lambda_c\vec{x}_n\ .F_i)\ \mathbf{A}\ \Omega\ \overline{\mathbf{0}}\ (\vec{x}_n\ \ c)$$

(and using Property 3) $\quad = \lambda c.(\lambda_c\vec{x}_n\ .N_i)\ (\vec{x}_n\ \ c) = \lambda c.N_i\ c = N_i$

So, the reduction of $\lambda c.E_i\ \mathbf{A}\ \Omega\ \overline{\mathbf{n}}\ (\vec{x}_n\ \ c)$ eventually yields the normal form of the argument, suppressing this way the combinators $\mathbf{H}_i$'s occurring in E_i.

Example: Let $E \equiv \lambda x.(\lambda w.\lambda y.\ w\ y\ x)\ (\lambda z.\ z)\ x$

Its head reduction is

$$\begin{aligned} E \ &\underset{h}{\rightarrow} \lambda x.(\lambda y.(\lambda z.\ z)\ y\ x)\ x \\ &\underset{h}{\rightarrow} \lambda x.(\lambda z.\ z)\ x\ x \\ &\underset{h}{\rightarrow} \lambda x.x\ x \end{aligned}$$

After cps conversion and simplification the expression becomes

$$\mathcal{N}(E) = \lambda_c x.\ \lambda c.(\lambda w.\ \lambda_c y.\lambda c.\ w\ (\lambda f.f\ y\ (\lambda f.f\ x\ c)))\ (\lambda_c z.z)\ (\lambda f.f\ x\ c)$$

The weak head reduction of $\mathcal{N}(E)\ A\ \Omega\ \bar{0}$ simulates the head reduction of E. Reductions corresponding to head reductions of the source expression are marked by ✢ ; the other being administrative reductions.

$$\begin{aligned}
\mathcal{N}(E)\ A\ \Omega\ \bar{0} \quad & \xrightarrow[w]{} A\ (\lambda x.\lambda c.(\lambda w.\ \lambda_c y.\lambda c.\ w\ (\lambda f.f\ y\ (\lambda f.f\ x\ c)))\ (\lambda_c z.z)\ (\lambda f.f\ x\ c))\ \Omega\ \bar{0} \\
& \xrightarrow[w]{} \Omega\ (\lambda x.\lambda c.(\lambda w.\ \lambda_c y.\lambda c.\ w\ (\lambda f.f\ y\ (\lambda f.f\ x\ c)))\ (\lambda_c z.z)\ (\lambda f.f\ x\ c))\ \bar{0} \\
& \xrightarrow[w]{} (\lambda x.\lambda c.(\lambda w.\ \lambda_c y.\lambda c.\ w\ (\lambda f.f\ y\ (\lambda f.f\ x\ c)))\ (\lambda_c z.z)\ (\lambda f.f\ x\ c))\ H_0\ A\ \Omega\ \bar{1} \\
& \xrightarrow[w]{2} (\lambda w.\ \lambda_c y.\lambda c.\ w\ (\lambda f.f\ y\ (\lambda f.f\ H_0\ c)))\ (\lambda_c z.z)\ (\lambda f.f\ H_0\ A)\ \Omega\ \bar{1} \\
& \xrightarrow[w]{} (\lambda_c y.\lambda c.\ (\lambda_c z.z)\ (\lambda f.f\ y\ (\lambda f.f\ H_0\ c)))\ (\lambda f.f\ H_0\ A)\ \Omega\ \bar{1} && ✢ \\
& \xrightarrow[w]{2} (\lambda y.\lambda c.\ (\lambda_c z.z)\ (\lambda f.f\ y\ (\lambda f.f\ H_0\ c)))\ H_0\ A\ \Omega\ \bar{1} \\
& \xrightarrow[w]{} (\lambda c.\ (\lambda_c z.z)\ (\lambda f.f\ H_0\ (\lambda f.f\ H_0\ c)))\ A\ \Omega\ \bar{1} && ✢ \\
& \xrightarrow[w]{3} (\lambda z.z)\ H_0\ (\lambda f.f\ H_0\ A)\ \Omega\ \bar{1} \\
& \xrightarrow[w]{} H_0\ (\lambda f.f\ H_0\ A)\ \Omega\ \bar{1} && ✢
\end{aligned}$$

The hnf is reached. Using the definition of H_0 we get

$$H_0\ (\lambda f.f\ H_0\ A)\ \Omega\ \bar{1} \rightarrow \lambda_c x.\lambda c.x\ ((\lambda f.f\ H_0\ A)\ (R\ \bar{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c)) \ \equiv \Delta \qquad (H)$$

Now, we show that this hnf Δ is equivalent to (or that the reconstruction yields) the principal hnf $(\lambda x.x\ x)$ in cps form $(\lambda_c x.\lambda c.x\ (\lambda f.f\ x\ c))$.

$$\begin{aligned}
\Delta & \rightarrow \lambda_c x.\lambda c.x\ (R\ \bar{1}\ (\lambda c.\lambda f.f\ x\ c)\ H_0\ A\ (K\ c)) \\
& \rightarrow \lambda_c x.\lambda c.x\ (\lambda f.f\ (\lambda c.H_0\ A\ \Omega\ \bar{1}\ ((\lambda c.\lambda f.f\ x\ c)\ c))\ (A\ (R\ \bar{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c))) \qquad (R)
\end{aligned}$$

$$\text{Since} \qquad \lambda c.H_0\ A\ \Omega\ \bar{1}\ ((\lambda c.\lambda f.f\ x\ c)\ c) \rightarrow \lambda c.(\lambda_c w.\lambda c.w\ c)\ ((\lambda c.\lambda f.f\ x\ c)\ c) \qquad (H),(A),(K)$$

$$\rightarrow \lambda c.x\ c =_\eta x$$

$$\text{and} \qquad A\ (R\ \bar{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c) \xrightarrow[w]{} c \qquad (A),(K)$$

then $\Delta = \lambda_c x.\lambda c.x\ (\lambda f.f\ x\ c)$ ❑

All reductions taking place in the head reduction of the source expression are performed on the transformed expression by weak head reduction. Here the resulting expression is interconvertible with the principal hnf in cps form. Note that the reconstruction process is not completed by weak head reduction. In a sense, the reconstruction process is lazy ; it can take place (by wh-reduction) only when the resulting expression is applied. Only the required subexpressions will be reconstituted.

The following property states that for any closed expression E the weak head reduction of $\mathcal{N}(E)\ A\ \Omega\ \bar{0}$ simulates the head reduction of E. If E has a hnf H then the wh-reduction of $\mathcal{N}(E)\ A\ \Omega\ \bar{0}$ yields an expression equal to $\mathcal{N}(H)\ A\ \Omega\ \bar{0}$ (after administrative reductions).

Property 2 *Let E be a closed expression, if* $E \xrightarrow[h]{*} H$ *then there exists an expression X such that* $\mathcal{N}(E)\ A\ \Omega\ \bar{0} \xrightarrow[w]{*} X \xleftarrow[w]{a} \mathcal{N}(H)\ A\ \Omega\ \bar{0}$ *and if H is a hnf then X is a whnf. Furthermore E does not have a hnf iff* $\mathcal{N}(E)\ A\ \Omega\ \bar{0}$ *does not have a whnf.*

Proof. [Sketch] We first show that the property holds for one reduction step. Two lemmas are needed: "$\mathcal{N}(E)[\mathcal{N}(F)/x] \equiv \mathcal{N}(E[F/x])$" which is shown in [16] and "if $x \not\equiv y$ and x does not occur free in G then $E[F/x][G/y] \equiv E[G/y][F[G/y]/x]$" which is shown in [3] (2.1.16 pp. 27). The property is then shown by induction on the number of reduction steps. Concerning the second part of the property: if an expression E_0 does not have a hnf there is an infinite reduction sequence $E_0 \xrightarrow[h]{} E_1 \xrightarrow[h]{} \ldots$. It is clear from the preceding proof that the corresponding weak head reduction on $\mathcal{N}(E_0)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ will also be infinite, so this expression does not have a whnf. If E has a hnf H then $E_0 \xrightarrow[h]{n} H$ so there is a X such that $\mathcal{N}(E_0)\mathbf{A}\ \Omega\ \bar{\mathbf{0}} \xrightarrow[w]{+} X \xleftarrow[w]{a} \mathcal{N}(H)\mathbf{A}\ \Omega\ \bar{\mathbf{0}}$. H being of the form $\lambda x_1. \ldots \lambda x_n.x_i\ E_1 \ldots E_p$, after administrative reductions, $\mathcal{N}(H)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ is of the form $H_i\ C\ \Omega\ \bar{n}$ and the reduction rule of $\mathbf{H}_i$ yields a whnf. ❑

In general $\mathcal{N}(H)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}} \leftrightarrow \mathcal{N}(H)$ does not hold, namely the result is not always interconvertible with the hnf in cps. This is usual with this kind of transformation ; the result is in compiled form and is convertible to its source version only under certain conditions. Still, $\mathcal{N}(H)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ and $\mathcal{N}(H)$ have a strong relationship. Let $H \equiv \lambda x_1. \ldots \lambda x_n.x_i\ E_1 \ldots E_p$ then

$$\mathcal{N}(H) = \lambda_c \vec{x}_n\ .\lambda c.\ x_i\ (\mathcal{N}(\vec{E}_p)\ \ c)$$

and $\mathcal{N}(H)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}} = \lambda_c \vec{x}_n\ \ \lambda c.\ x_i\ (\vec{X}_p\ \ E)$ with $X_i \equiv \lambda c.\mathcal{N}(\lambda \vec{x}_n\ \ .E_i)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}\ (\vec{x}_n\ \ c)$.

So, the head variable is the same and if the sub-expressions $\mathcal{N}(E_i)$ and X_i have a hnf they will also have the same head variable. Likewise, if a sub-expression $\mathcal{N}(E_i)$ does not have a hnf then the corresponding expression $\mathcal{N}(\lambda \vec{x}_n\ \ .E_i)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}\ (\vec{x}_n\ \ c)$ does not have a whnf ; they can then be considered equivalent. However we do not have a plain equivalence since there are expressions whose sub-expressions all have a hnf but have no nf themselves ; for example $(\lambda xy.y\ (x\ x))\ (\lambda xy.y\ (x\ x)) \rightarrow \ldots \rightarrow (\lambda y.y\ (\lambda y.y\ \ldots(\lambda xy.y\ (x\ x))\ (\lambda xy.y\ (x\ x)))$. For such expressions $\mathcal{N}(H)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ and $\mathcal{N}(H)$ are not interconvertible; the H_i's substituted for the leading variables may never be completely removed. However, for expressions with a normal form the following result holds.

Property 3 *Let E be a closed expression with a normal form then $\mathcal{N}(E) \leftrightarrow \mathcal{N}(E)$* $\mathbf{A}\ \Omega\ \bar{\mathbf{0}}$

Proof. [Sketch] If $E \rightarrow F$ then $\mathcal{N}(E) \xrightarrow{+} \mathcal{N}(F)$ and then obviously $\mathcal{N}(E)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}} \rightarrow \mathcal{N}(F)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ (just pick up the same redex). So if E has a normal form S then $E \xrightarrow{*} S$ and $\mathcal{N}(E)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}} \xrightarrow{*} \mathcal{N}(S)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$. We just have to show that for any normal form S, $\mathcal{N}(S) \leftrightarrow \mathcal{N}(S)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ which is proved by induction on the structure of nfs. ❑

Here, we propose one possible definition of combinators $\bar{\mathbf{n}}$, Ω, $\mathbf{H}_i$, $\mathbf{R}$ in terms of pure λ-expressions. We do not claim it is the best one ; we just want to show that such combinators can indeed be implemented in the same language. Simpler definitions could be conceived in a less rudimentary language (e.g. λ-calculus extended with constants).

We represent $\bar{\mathbf{n}}$ by Church integers, i.e. $\bar{\mathbf{0}} = \lambda fx.x$ and $\bar{\mathbf{n}} = \lambda fx.f^n\ x$. The successor function S^+ is defined by $S^+ = \lambda xyz.y\ (x\ y\ z)$.

$\mathbf{I} = \lambda x.x$, $\mathbf{K} = \lambda xy.x$, $\mathbf{A} = \lambda xy.y\ x$ and $\mathbf{Y} = (\lambda xy.y\ (x\ x\ y))(\lambda xy.y\ (x\ x\ y))$ (Turing's fixed point combinator)

$$\Omega = \mathbf{Y}\ (\lambda wen.\ e\ (\mathbf{H}\ n)\ \mathbf{A}\ w\ (S^+\ n))$$

The family $\mathbf{H}_i$ is represented by $\mathbf{H}\ \bar{\mathbf{i}}$ with

$$\mathbf{H} = \lambda ieon.\ n\ \mathbf{L}\ (\lambda ac.a\ \mathbf{I}\ (\mathbf{W}\ i)\ (e\ (\mathbf{R}\ n\ a)\ (\mathbf{K}\ c)))\ \mathbf{I}$$

where $\mathbf{W} = \lambda i.i\ (\lambda xyz.z\ x)\ \mathbf{K}$

$$\mathbf{L} = \lambda ab.\lambda_c x.a\ (\lambda c.b\ (\lambda f.f\ x\ c))$$

and $\mathbf{R} = \mathbf{Y}\ (\lambda ruvwxy.\lambda f.f\ (\lambda c.w\ \mathbf{A}\ \Omega\ u\ (v\ c))\ (x\ (r\ u\ v)\ y)$

We can easily check that these definitions imply the reduction rules previously assumed, for example $\Omega\, \mathbf{E}\, \overline{\mathbf{n}} \overset{*}{\underset{w}{\to}} \mathbf{E}\, \mathbf{H}_n\, \mathbf{A}\, \Omega\, \overline{\mathbf{n+1}}$ or $\mathbf{R}\, \mathbf{E}\, \mathbf{F}\, \mathbf{G}\, \mathbf{H}\, \mathbf{I} \overset{*}{\underset{w}{\to}} \lambda f.f\, (\lambda c.\mathbf{G}\, \mathbf{A}\, \Omega\, \mathbf{E}\, (\mathbf{F}\, c))\, (\mathbf{H}\, (\mathbf{R}\, \mathbf{E}\, \mathbf{F})\, \mathbf{I})$.

4 Strong Reduction

Full normal forms are evaluated by first reducing expressions to hnf and then reducing the arguments of the head variable. We follow the same idea as for head reduction. Instead of instantiating variables by combinators $\mathbf{H}_i$ we use the family $\mathbf{S}_i$ which will carry out the evaluation before reconstructing. The recursive continuation Ω is the same as before except that it applies the λ-abstraction to $\mathbf{S}_i$ instead of $\mathbf{H}_i$.

If E is a closed expression, its transformed form will be $\mathcal{N}(E)\, \mathbf{A}\, \Omega\, \overline{\mathbf{0}}$ with

$$\Omega\, M\, \overline{\mathbf{n}} \underset{w}{\to} M\, \mathbf{S}_n\, \mathbf{A}\, \Omega\, \overline{\mathbf{n+1}} \qquad (\Omega)$$

and

$$\mathbf{S}_i\, M\, N\, \overline{\mathbf{n}} \underset{w}{\to} M\, \mathbf{E}_n\, \mathbf{B}\, \mathbf{H}_i\, N\, \overline{\mathbf{n}} \qquad (\mathrm{S})$$

where

$$\mathbf{E}_i\, M\, N\, P \underset{w}{\to} N\, \mathbf{E}_i\, P\, (M\, \mathbf{A}\, \Omega\, \overline{\mathbf{i}}\, \mathbf{C}) \qquad (\mathrm{E})$$

$$\mathbf{B}\, M\, N \underset{w}{\to} N\, \mathbf{A} \qquad (\mathbf{B})$$

$$\mathbf{C}\, M\, N\, P \underset{w}{\to} P\, (\lambda f.f\, (\lambda c.\, c\, M)\, N) \qquad (\mathrm{C})$$

When the hnf is reached the head variable previously instantiated by $\mathbf{S}_i$ is called. It triggers the evaluation of its arguments via $\mathbf{E}_i$ and insert $\mathbf{H}_i$ as last continuation. $\mathbf{E}_i$ applies the arguments to $\mathbf{A}\, \Omega\, \overline{\mathbf{i}}$ which will be evaluated in a right to left order and inserts the continuation $\mathbf{C}$ needed to put back the evaluated arguments $X_1,\ldots,X_n$ in cps form (i.e. $\lambda f.f\, X_1\, (\ldots(\lambda f.f\, X_n\, E)\ldots)$). The role of $\mathbf{H}_i$'s is still to reconstruct the expression. Combinator $\mathbf{H}_i$ keeps the same definition except for $\mathbf{R}$ which have now the simplified reduction rule

$$\mathbf{R}\, \mathbf{E}\, \mathbf{F}\, \mathbf{G}\, \mathbf{H}\, \mathbf{I} = \lambda f.f\, (\lambda c.\mathbf{G}\, (\mathbf{F}\, c))\, (\mathbf{H}\, (\mathbf{R}\, \mathbf{E}\, \mathbf{F})\, \mathbf{I}) \qquad (\mathbf{R})$$

When $\mathbf{R}$ is applied, the arguments are already evaluated and reconstructed so there is no need to apply them to $\mathbf{A}\, \Omega\, \overline{\mathbf{i}}$ as before.

Example: Let $E = \lambda x.(\lambda w.\lambda y.w\ y\ ((\lambda v.v)\ x))\ (\lambda z.z)\ x$

Its strong reduction is

$$\begin{aligned} E &\underset{s}{\to} \lambda x.(\lambda y.(\lambda z.z)\ y\ ((\lambda v.v)\ x))\ x \\ &\underset{s}{\to} \lambda x.(\lambda z.z)\ x\ ((\lambda v.v)\ x) \\ &\underset{s}{\to} \lambda x.x\ ((\lambda v.v)\ x) \\ &\underset{s}{\to} \lambda x.x\ x \end{aligned}$$

After cps conversion and simplification the expression becomes

$$\mathcal{N}(E) = \lambda_c x.(\lambda w.\lambda_c y.\lambda c.w\ (\lambda f.f\ y\ (\lambda f.f\ ((\lambda v.\ v)\ x)\ c))\ (\lambda_c z.z)\ (\lambda f.f\ x\ c)$$

The weak head reduction of the cps expression is (reductions corresponding to strong reductions of the source expression are marked by $+$)

$\mathcal{N}(E)\ A\ \Omega\ \overline{0} \quad \xrightarrow[w]{3} (\lambda x.\ \lambda c.(\lambda w.\ \lambda_c y.\lambda c.\ w\ (\lambda f.f\ y\ (\lambda f.f\ ((\lambda v.\ v)\ x)\ c))\ (\lambda_c z.z)\ (\lambda f.f\ x\ c))\ S_0\ A\ \Omega\ \overline{1}$

$\xrightarrow[w]{2} (\lambda w.\ \lambda_c y.\lambda c.\ w\ (\lambda f.f\ y\ (\lambda f.f\ ((\lambda v.\ v)\ S_0)\ c))\ (\lambda_c z.z)\ (\lambda f.f\ S_0\ A)\ \Omega\ \overline{1}$

$\xrightarrow[w]{} (\lambda_c y.\lambda c.\ (\lambda_c z.z)\ (\lambda f.f\ y\ (\lambda f.f\ ((\lambda v.\ v)\ S_0)\ c))\ (\lambda f.f\ S_0\ A)\ \Omega\ \overline{1}$ $\quad +$

$\xrightarrow[w]{2} (\lambda y.\lambda c.\ (\lambda_c z.z)\ (\lambda f.f\ y\ (\lambda f.f\ ((\lambda v.\ v)\ S_0)\ c))\ S_0\ A\ \Omega\ \overline{1}$

$\xrightarrow[w]{} (\lambda c.\ (\lambda_c z.z)\ (\lambda f.f\ S_0\ (\lambda f.f\ ((\lambda v.\ v)\ S_0)\ c))\ A\ \Omega\ \overline{1}$ $\quad +$

$\xrightarrow[w]{3} (\lambda z.z)\ S_0\ (\lambda f.f\ ((\lambda v.\ v)\ S_0)\ A)\ \Omega\ \overline{1}$

$\xrightarrow[w]{} S_0\ (\lambda f.f\ ((\lambda v.\ v)\ S_0)\ A)\ \Omega\ \overline{1}$ the hnf is reached, the reduction rule of S_0 is used. $+$

$\xrightarrow[w]{} (\lambda f.f\ ((\lambda v.\ v)\ S_0)\ A)\ E_1\ B\ H_0\ \Omega\ \overline{1}$

$\xrightarrow[w]{} E_1\ ((\lambda v.\ v)\ S_0)\ A\ B\ H_0\ \Omega\ \overline{1}$

$\xrightarrow[w]{3} ((\lambda v.\ v)\ S_0\ A\ \Omega\ \overline{1}\ C)\ A\ H_0\ \Omega\ \overline{1}$

$\xrightarrow[w]{} S_0\ A\ \Omega\ \overline{1}\ C\ A\ H_0\ \Omega\ \overline{1}$ the nf is reached ; the reconstruction begins. $+$

$\xrightarrow[w]{3} H_0\ A\ \Omega\ \overline{1}\ C\ A\ H_0\ \Omega\ \overline{1}$

$\xrightarrow[w]{} (\lambda_c x.\lambda c.x\ (A\ (R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c)))\ C\ A\ H_0\ \Omega\ \overline{1}$

$\xrightarrow[w]{} C\ (\lambda x.\lambda c.x\ (A\ (R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c)))\ A\ H_0\ \Omega\ \overline{1}$

$\xrightarrow[w]{} H_0\ X\ \Omega\ \overline{1}$ with $X \equiv \lambda f.f\ (\lambda c.c(\lambda x.\lambda c.x\ (A\ (R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c))))\ A$

$\rightarrow \lambda_c x.\lambda c.x\ (X\ (R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c))$

The wh-reduction is completed. Now, we show that the result is equivalent to the normal form in cps form.

$X \xrightarrow{*} \lambda f.f\ (\lambda_c x.x)\ A$ (A),(K)

and $X\ (R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c) \xrightarrow{*} R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c)\ (\lambda_c x.x)\ A\ (K\ c)$

$\xrightarrow{*} \lambda f.f\ (\lambda c.(\lambda_c x.x)\ ((\lambda c.\lambda f.f\ x\ c)\ c))\ (A\ (R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c))$ (R)

$\xrightarrow{*} \lambda f.f\ x\ c$ since $A\ (R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c) \xrightarrow{*} c$ (A),(K)

and $\lambda c.(\lambda_c x.x)\ ((\lambda c.\lambda f.f\ x\ c)\ c) \rightarrow \lambda c.x\ c \rightarrow_\eta x$

So $\lambda_c x.\lambda c.x\ (X\ (R\ \overline{1}\ (\lambda c.\lambda f.f\ x\ c))\ (K\ c)) \xrightarrow{*} \lambda_c x.\lambda c.x\ (\lambda f.f\ x\ c)$ which is the normal form in cps form. ❑

All the reductions taking place during the strong reduction of the source expression are carried out by wh-reduction of the transformed expression. We do not really get the full normal form since the reconstruction can not be achieved completely by weak head reduction. As before the reconstruction is lazy. However the result is convertible to the normal form in cps and the complexity of this last step is bounded by the size of the normal form. If we were just interested in normal forms as a syntactic result, H_i's could be replaced by functions printing the nf instead of building a suspension representing it. In this case, the evaluation would be completely carried out by wh-reduction.

We have the analogues of Property 2 and Property 3. The following property states that for any closed expression E the weak head reduction of $\mathcal{N}(E)\ A\ \Omega\ \overline{0}$ simulates the strong reduction of E.

Property 4 *Let E be a closed expression, if* $E \xrightarrow[s]{*} S$ *then there exists an expression X such that* $\mathcal{N}(E)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}} \xrightarrow[w]{*} X \xleftarrow[w]{a} \mathcal{N}(S)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ *and if S is a nf then X is a whnf. Furthermore, E does not have a nf iff* $\mathcal{N}(E)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ *does not have a whnf.*

The result of the evaluation of $\mathcal{N}(E)\ \mathbf{A}\ \Omega\ \bar{\mathbf{0}}$ is interconvertible with the nf in cps.

Property 5 *If a closed expression E has a normal form then* $\mathcal{N}(E) \leftrightarrow \mathcal{N}(E)\ A\ \Omega\ \bar{0}$

Their proofs are similar to those of Property 2 and Property 3.

5 Head Reduction of Typed λ-Expressions

In the previous sections we needed to count the number of leading λ's during the evaluation. Using some form of typing it is possible to know the functionality of the expression prior to evaluation and thus get rid of this counter. We consider only head reduction ; typing does not seem to simplify the compilation of strong reduction.

Simply typed λ-calculus would suit our purposes but would harshly restrict the class of expressions. More flexible typing systems are sufficient. One candidate is reflexive reducing typing [2] which has already been used in [9] to determine the functionality of expressions. It is shown in [2] that we can restrict a language to reflexive reducing types without weakening its expressive power. Reflexive reducing types are defined by (possibly recursive) equations of the form $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \alpha$, $\sigma_1, \ldots, \sigma_n$ being themselves reflexive reducing types and α being a base type (not a reflexive type). This enables us to type recursive functions but not for example (λxy.xx)(λxy.xx) (this expression has the reflexive type $\rho \to \alpha \to \sigma$ with $\sigma = \alpha \to \sigma$ which is not reducing). We do not dwell here on the details of this typing system. The important point for us is that if a closed expression with type $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \alpha$ has a hnf then it is of the form $\lambda x_1. \ldots \lambda x_n. x_i\ E_1 \ldots E_p$.

If the expression to reduce to hnf has functionality n then the transformed expression is

$$\mathcal{N}(E)\ (\lambda f.f\ X^1_n \ldots (\lambda f.f\ X^n_n\ L_n)\ldots) \quad \text{and we note } \mathcal{N}(E)\ (\vec{X}_n\quad L_n).$$

That is, we apply the expression to n arguments in order to remove the n leading abstractions. Combinators X^i_n play the same role as the combinators $\mathbf{H}_i$ introduced in section 3. They will be substituted for variables and used to start the reconstruction process.

$$X^i_n\ \ E = \lambda_c\vec{x}_n\ .\ \lambda c.x_i\ (E\ (\mathbf{R}_n\ (\lambda c.\vec{x}_n\ \ c))\ (\mathbf{K}\ c)) \qquad \text{(X)}$$

Combinator $\mathbf{R}_n$ used in the definition of X^i_n plays the same role as **R** in the definition of $\mathbf{H}_i$.

$$\mathbf{R}_n\ E\ F\ G\ H = \lambda f.f\ (\lambda c.F\ \mathbf{L}_n\ (E\ c))\ (G\ (\mathbf{R}_n\ E)\ H) \qquad \text{(R)}$$

In the preceding sections, the reconstruction of subexpressions was based on the same technique as the reduction of the global expression: each subexpression was applied to continuation Ω and was rebuild after being reduced to hnf. Here, there is no type information available on the subexpressions and we cannot use the same method as for the global expression. In particular, a subexpression $(\lambda_c z.E)$ can not be reduced since we do not know its functionality. However, it may contain occurrences of combinators X^i_n which are to be removed. This case is treated using combinators $\mathbf{L}_n$ and $\mathbf{Z}_n$ which carry on the reconstruction inside the λ-abstraction.

$$\mathbf{L}_n\ E = \lambda_c\vec{x}_n\ .\ \lambda_c z.\lambda c.\ E\ (\mathbf{Z}_n\ z)\ \mathbf{L}_n\ (\vec{x}_n\ \ c) \qquad \text{(L)}$$

$$\mathbf{Z}_n\ E\ F = \lambda_c\vec{x}_n\ .\ \lambda c.\ E\ (F\ (\mathbf{R}_n\ (\lambda c.\vec{x}_n\ \ c))\ (\mathbf{K}\ c)) \qquad \text{(Z)}$$

For example, if the hnf is of the form $\lambda x_1 \ldots \lambda x_n . x_i \ldots (\lambda z.E) \ldots$ then $\mathbf{R}_n$ applies each subexpression to $\mathbf{L}_n$ and $(\vec{x}_n\ c)$ and we will get for the λ-abstraction $(\lambda z.E)$

$$\begin{aligned}\lambda c.(\lambda_c z.E)\ \mathbf{L}_n\ (\vec{x}_n\ \ c) &\rightarrow \lambda c.\ \mathbf{L}_n(\lambda z.E)\ (\vec{x}_n\ \ c)\\ &\rightarrow \lambda c.\ (\lambda_c \vec{x}_n\ .\ \lambda_c z.\lambda c.\ (\lambda z.E)\ (\mathbf{Z}_n\ z)\ \mathbf{L}_n\ (\vec{x}_n\ \ c))(\vec{x}_n\ \ c)\\ &\rightarrow \lambda_c z.\lambda c.\ E[\mathbf{Z}_n\ z/z]\ \mathbf{L}_n\ (\vec{x}_n\ \ c)\end{aligned}$$

The list of variables has been pushed inside the λ-abstraction and the reconstruction can continue. Variable z is replaced by $(\mathbf{Z}_n\ z)$ so that when it is applied to the list $(\vec{x}_n\ \ c)$ it returns z. Combinators $\mathbf{R}_n$, $\mathbf{L}_n$, $\mathbf{X}_n^i$, $\mathbf{Z}_n$ act very much like combinators used in abstraction algorithms. $\mathbf{X}_n^i$ is a selector (it selects the ith variable), $\mathbf{Z}_n$ is (like **K**) a destructor (it ignores the list and returns its first argument), $\mathbf{R}_n$ and $\mathbf{L}_n$ distribute the list of variables $(\lambda c.\vec{x}_n\ \ c)$ throughout the expression.

The head normal form (if any) of E will be of the form $(\lambda x_1 \ldots \lambda x_n . x_i\ E_1 \ldots E_p)$ so $\mathcal{N}(E)\ (\vec{\mathbf{X}}_n\ \ \mathbf{L}_n)$ will be reduced (by weak head reduction) to $\mathbf{X}_n^i\ (\vec{E}_p\ \ \mathbf{L}_n)$ and then, according to the definition of combinators $\mathbf{X}_n^i$, to $\lambda_c \vec{x}_n\ .\lambda c.x_i\ ((\vec{E}_p\ \ \mathbf{L}_n)\ (\mathbf{R}_n\ (\lambda c.\vec{x}_n\ \ c))\ (\mathbf{K}\ c))$. As before the continuation $(\mathbf{K}\ c)$ removes reconstructing expressions and returns the final continuation.

The following property states that for any closed expression E of type $\sigma_1 \rightarrow \ldots \rightarrow \sigma_n \rightarrow \alpha$ the weak head reduction of $\mathcal{N}(E)\ (\vec{\mathbf{X}}_n\ \ \mathbf{L}_n)$ simulates the head reduction of E.

Property 6 *Let E be a closed expression of functionality n, if* $E \overset{*}{\underset{h}{\Rightarrow}} H$ *then there exists an expression X such that* $\mathcal{N}(E)\ (\vec{\mathbf{X}}_n\ \ \mathbf{L}_n) \overset{*}{\underset{w}{\Rightarrow}} X \overset{a}{\underset{w}{\Leftarrow}} \mathcal{N}(H)\ (\vec{\mathbf{X}}_n\ \ \mathbf{L}_n)$ *and if H is a hnf then X is a whnf. Furthermore, E does not have a hnf iff* $\mathcal{N}(E)\ (\vec{\mathbf{X}}_n\ \ \mathbf{L}_n)$ *does not have a whnf.*

If E has a normal form the result of the evaluation of $\mathcal{N}(E)\ (\vec{\mathbf{X}}_n\ \ \mathbf{L}_n)$ is interconvertible with the principal hnf in cps form.

Property 7 *If a closed expression E of functionality n has a normal form then* $\mathcal{N}(E) \leftrightarrow \mathcal{N}(E)(\vec{\mathbf{X}}_n\ \ \mathbf{L}_n)$

Their proofs are similar to those of Property 2 and Property 3.

6 Applications

Among practical applications of head reduction listed in the introduction, one is to compile more efficient evaluation strategies. We describe better this question in the next section and suggest in section 6.2 how our approach can be used to compile such strategies. Implementation issues are discussed in 6.3.

6.1 Spine Strategies

Even when evaluating weak-head-normal forms it is sometimes better to reduce sub-terms in head normal forms. For example, in lazy graph reduction, the implementation of β-reduction $(\lambda x.E)F \rightarrow_\beta E[F/x]$ implies making a copy of the body E before the substitution. It is well known that this may lose sharing and work may be duplicated [18]. Program transformations, such as fully lazy lambda-lifting [10], aim at maximizing sharing but duplication of work can still occur. Another approach used to avoid recomputation is to consider alternative evaluation strategies. If the expression to reduce is $(\lambda x.E)F$ we know that the whnf of the body E will be needed and so it is safe to reduce E prior to the β-reduction. This computation rule belongs to the so-called spine-strategies [4]. It never takes more reductions than normal order and may prevent duplication of work.

A revealing example, taken from [8], is the reduction of $A_n\ \mathbf{I}$ where the family of λ-expressions A_i is defined by $A_0 = \lambda x.x\ \mathbf{I}$ and $A_n = \lambda h.(\lambda w.w\ h\ (w\ w))\ A_{n-1}$. The expression $A_n\ \mathbf{I}$ is reduced using the call-by-name weak head graph reduction as follows:

$$A_n\,\mathbf{I} = (\lambda h.(\lambda w.w\ h\ (w\ w))\ A_{n-1})\ \mathbf{I}$$

$$\to (\lambda w.w\ \mathbf{I}\ (w\ w))\ A_{n-1}$$

$$\to A_{n-1}\ \mathbf{I}\ (\bullet\ \bullet) \equiv (\lambda h.(\lambda w.w\ h\ (w\ w))\ A_{n-2})\ \mathbf{I}\ (\bullet\ \bullet) \qquad \text{(• representing the sharing of } A_{n-1}\text{)}$$

$$\to (\lambda w.w\ \mathbf{I}\ (w\ w))\ A_{n-2}\ (A_{n-1}\ \bullet)$$

The sharing is lost and the redexes inside A_{n-1} are duplicated. The complexity of the evaluation is $O(2^n)$. On the other hand, by reducing λ-abstractions to hnf before β-reductions the evaluation sequence becomes

$$A_n\,\mathbf{I} = (\lambda h.(\lambda w.w\ h\ (w\ w))\ A_{n-1})\ \mathbf{I}$$

$$\to \quad (\lambda h.A_{n-1}\ h\ (\bullet\ \bullet))\ \mathbf{I}$$

$$\xrightarrow{4(n-1)} \quad (\lambda h.A_0\ h\ (\bullet\ \bullet))\ \mathbf{I} \qquad A_{n-1} \text{ reduces to } A_0 \text{ in } 4(n-1) \text{ steps}$$

$$\to \quad (\lambda h.\ \mathbf{I}\ (A_0\ \bullet))\ \mathbf{I} \to (\lambda h.\ A_0\ \bullet)\ \mathbf{I} \to (\lambda h.\ \mathbf{I})\ \mathbf{I} \to \mathbf{I}$$

and the A_i's remain shared until they are reduced to their hnf A_0. The complexity of the evaluation drops from exponential to linear.

Of course this strategy alone is not optimal (optimal reduction of λ-expressions is more complex [12][13]) and work can still be duplicated. But in [17] Staples proposes a similar evaluation strategy with the additional rule that substitutions are not carried out inside redexes (they are suspended until the redex is needed and reduced to hnf). This reduction has been shown to be optimal for a λ-calculus with explicit substitutions.

6.2 Sharing Hnf's

We saw that evaluating the λ-abstraction to hnf before the β-reduction can save work by sharing hnf's instead of whnf's. Following this idea, maximal sharing is obtained by reducing every closure to hnf instead of whnf. The straightforward idea of applying closures to $\mathbf{A}\ \Omega\ \overline{\mathbf{0}}$ does not work. Our previous results were relying on the fact that the expression to be reduced was closed. Here, even if the global expression is closed, we may have to reduce to hnf sub-expressions containing free variables. For example, if $(\lambda x.\ \mathbf{I}\ (\lambda y.\ \mathbf{I}\ x))$ is cps converted and the two closures $(\lambda_c x.\ldots)$ and $(\lambda_c y.\ldots)$ are applied to $\mathbf{A}\ \Omega\ \overline{\mathbf{0}}$ then during the reduction of $(\lambda_c x.\ \ldots)$ we will have to reduce to hnf $(\lambda_c y.\ldots)\ \mathbf{A}\ \Omega\ \overline{\mathbf{0}}$. But x is already instantiated by $\mathbf{H}_0$ and we get $(\lambda_c y.\mathbf{H}_0)\ \mathbf{A}\ \Omega\ \overline{\mathbf{0}} \to \mathbf{H}_0\ \mathbf{A}\ \Omega\ \overline{\mathbf{1}} \to (\lambda_c y.y)$ which is false. The cps hnf of $(\lambda y.\ \mathbf{I}\ x)$ should have been $(\lambda_c y.\ \mathbf{H}_0)$ and the enclosing evaluation of $(\lambda_c x.\ \ldots)$ could continue. The problem comes from free variables already instantiated by combinators when a new head reduction begins.

One solution to the free variable problem is to use a second index as in [5]. In our framework, this technique is expressed by changing the rule of cps conversion for applications

$$\mathcal{N}^*(E\ F) = \lambda cwn.\mathcal{N}^*(E)\ (\lambda f.f\ (\mathcal{N}^*(F)\ \mathbf{A}\ \Omega\ n\ n)\ c)\ w\ n$$

Each closure is applied to two indexes, initially the current binder length. The first one will play the same role as before and will increase at each leading lambda encountered during the reduction of the closure. The second index, say k, remains fixed for each closure and is used to determine if a combinator $\mathbf{H}_i$ corresponds to a free variable ($i<k$) or a bound variable ($i \geq k$) in that context. The definitions of combinators $\mathbf{H}_i$ and $\mathbf{R}$ become

$$\mathbf{H}_i\, \mathrm{M\, N}\, \bar{\mathrm{n}}\, \bar{\mathrm{k}} \;= \lambda_c \overrightarrow{x_{n-k}}\,.\ \lambda cwn.x_{i-k+1}\ (\mathrm{M}\ (\mathbf{R}\ \bar{\mathrm{n}}\ \bar{\mathrm{k}}\ \mathrm{n}\ (\lambda c.\ \overrightarrow{x_{n-k}}\ c))\ (\mathrm{K}\ c))\ w\ n \quad \text{, if } i \geq k \qquad \textbf{(H1)}$$

$$= \lambda_c \overrightarrow{x_{n-k}}\,.\ \lambda cwn.\mathbf{H}_i\ (\mathrm{M}\ (\mathbf{R}\ \bar{\mathrm{n}}\ \bar{\mathrm{k}}\ \mathrm{n}\ (\lambda c.\ \overrightarrow{x_{n-k}}\ c))\ (\mathrm{K}\ c))\ w\ n \quad \text{, if } i < k \qquad \textbf{(H2)}$$

$$\mathbf{R}\ \mathrm{C\,D\,E\,F\,G\,H\,I} \underset{w}{\Rightarrow} \lambda f.f\ (\mathrm{G\,A}\,\Omega\,\mathrm{C\,D}\,(\mathrm{F\,A}))\ \Omega\ \mathrm{E\,E})\ (\mathrm{H}\ (\mathbf{R}\ \mathrm{C\,D\,E\,F})\ \mathrm{I})$$

Now, the reduction of the (cps form of the) closure (λx. **I** (λy. **I** x)) becomes

$$(\lambda_c x.\lambda cwn.\mathbf{I}\ ((\lambda_c y.\ \mathbf{I}\, x)\ \mathbf{A}\ \Omega\ n\ n)\ c)\ w\ n)\ \mathbf{A}\ \Omega\ \bar{0}\ \bar{0}$$

$$\overset{*}{\Rightarrow} (\lambda cwn.\mathbf{I}\ ((\lambda_c y.\ \mathbf{I}\ \mathbf{H}_0)\ \mathbf{A}\ \Omega\ n\ n)\ c\ w\ n)\ \mathbf{A}\ \Omega\ \bar{1}\ \bar{0}$$

$$\overset{*}{\Rightarrow} (\lambda_c y.\ \mathbf{I}\,\mathbf{H}_0)\ \mathbf{A}\ \Omega\ \bar{1}\ \bar{1}\ \mathbf{A}\ \Omega\ \bar{1}\ \bar{0}$$

$$\overset{*}{\Rightarrow} \mathbf{H}_0\ \mathbf{A}\ \Omega\ \bar{2}\ \bar{1}\ \mathbf{A}\ \Omega\ \bar{1}\ \bar{0} \qquad \mathbf{H}_0 \text{ corresponds to a free variable in } (\lambda_c y....)$$

$$\overset{*}{\Rightarrow} (\lambda_c y.\ \mathbf{H}_0)\ \mathbf{A}\ \Omega\ \bar{1}\ \bar{0} \qquad \textbf{(H2)}$$

$$\overset{*}{\Rightarrow} \mathbf{H}_0\ \mathbf{A}\ \Omega\ \bar{2}\ \bar{0} \ \rightarrow \lambda_c x.\lambda_c y.x \qquad \mathbf{H}_0 \text{ corresponds to a bound variable in } (\lambda_c x....) \quad \textbf{(H1)}$$

Several optimizations can be designed to avoid producing useless closures. An important one (that we used in the example above) is $\mathcal{N}^*(E\ x) = \lambda c.\mathcal{N}^*(E)\ (\lambda f.f\ x\ c)$. This rule holds because a variable is always instantiated either by a combinator $\mathbf{H}_i$ or by a closure $\mathcal{N}^*(F)$ **A** Ω m m. It can be shown that $\mathbf{H}_i$ **A** Ω n n = $\mathbf{H}_i$ (if i<n) and ($\mathcal{N}^*(F)$ **A** Ω m m) **A** Ω n n = $\mathcal{N}^*(F)$ **A** Ω m m if n≥m.

Sharing as much hnf's as possible is likely to be quite costly in practice. In [5] Crégut gives a function for which the reduction takes n^2 steps when sharing hnf's whereas it takes only n steps using standard wh-reduction. It is also shown that this is the worst case. Some less extreme strategies may be envisaged. For example, the imbrication of several head reductions could be forbidden (i.e. closures would be reduced to hnf by the top level reduction but only to whnf during the reduction of another closure). This would simplify the reduction but the price is a potential loss of sharing.

6.3 Implementation issues

The most obvious way to implement our approach is to transform expressions as previously described and give the result to a compiler. The combinators A, Ω,... are compiled like other functions and the reconstruction is naturally implemented by closure building. However, with compilers which already integrate a cps conversion, a more efficient way would be to directly use the cps phase. This is less trivial since the following steps expect only cps expressions and we have to introduce special combinators which are not in cps. One solution is to implement those combinators by hand and the compiler can use them as primitive functions. We plan such an integration in our cps-based compiler. Further work is still needed on different extensions:

- So far we have only considered call by name. As cps conversion can be used to compile different computation rules (call-by-value, call-by-name with strictness annotations, ...) it is likely that our method could be extended to treat those strategies as well.
- This method should be extended to a λ-calculus with constants and primitive operators.

If we just aim at reducing a program to hnf/nf and print the result then our approach will be very efficient. The whole evaluation is a weak reduction which can be completely compiled. The only slight overhead will be a few more reductions for each leading lambda and printing the result which should be proportional to the size of the expression.

The costly part of head/strong reduction is the reconstruction of expressions which happens when we actually use (i.e. apply) the hnf/nf. In particular, reconstructing uses a lot of memory space. In order to implement efficient evaluation strategies as described previously, it would be useful to develop the following points:

- Several analyses can detect expressions for which wh-reduction is better and should be implemented as well. For example, one policy could be that a closure will be reduced to hnf only if it is shared (using a sharing analysis), complex enough (using a complexity analysis) and of course not already in hnf.
- Computation can still be duplicated by performing substitutions inside redexes. It would be interesting to extend our work to compile Staples' method [17] which avoids this loss of sharing.

We did a few experiments using the trivial way (i.e. transforming source expressions before giving them to our compiler). We transformed the family of expressions A_n defined in section 6.1 into supercombinators and into cps form. The evaluation of A_{15} **I** takes around 1s using standard reduction and around 1ms when each supercombinator is applied to **A** Ω $\underline{\mathbf{0}}$. This result is not surprising since the theoretical complexity is exponential in one case and linear in the other. More interestingly, we redefined the family A_n by $A_0 = \lambda x.x\ \mathbf{I}$ and $A_n = \lambda h.(\lambda w.w\ h\ (A_0\ w))\ A_{n-1}$. Here, the wh-reduction of A_n **I** does not duplicate work (the second occurrence of w is not needed) and nothing is saved by using head reduction. We found that the head reduction of supercombinators made the evaluation 3 to 4 times slower than the standard wh-reduction. This example indicates the cost of reconstructing expressions. This cost is acceptable when the final goal is to implement symbolic evaluation. When the goal is to evaluate whnf's more efficiently by sharing hnf's then such examples should be avoided using analyses or (maybe more pragmatically) using user's annotations.

7 Conclusion

Implementation of head and strong reduction has also been studied by Crégut [5] and Nadathur and Wilson [14]. Crégut's abstract machine is based on De Bruijn's notation. Two versions have been developed. The first one evaluates the head or full normal form of the global expression. The second one implements a spine strategy and shares head normal forms. Terms are extended with formal variables and the machine state includes two indexes. One plays the role of our binder level as in section 3 and 4, the other one is needed (only in the second version of the machine) to deal with the problem of free variables in subexpressions as exposed in section 6.2. The algorithm presented in [14] was motivated by the implementation of λProlog [15]. It evaluates terms to hnf and, expressed as an abstract machine, this technique resembles Crégut's. It is also based on De Bruijn notation and the machine state includes two indexes.

We described in this paper how to use cps conversion to compile head and strong reduction. The hnf's or nf's of cps-expressions are evaluated by weak head reduction and at each step the unique (weak) redex is the leftmost application. The technique does not require to modify the standard cps cbn conversion. The cps expression is just applied to a special continuation and an index to keep track of the binder length. We presented a way to get rid of this index and suggested applications for our technique. Cps conversion was important to this work in several respects: special continuations could be used to suppress the leading λ's and the regular form of cps expressions helped the reconstruction.

Compared to [5] and [14] the main difference is that we proceed by program transformations and stay within the functional framework. Used as a preliminary step our technique allows a standard compiler to evaluate under λ's. Thus we can take advantage of all the classical compiling tools like analyses, transformations or simplifications. As already emphasized in [7], another advantage of this approach is that we do not have to introduce an abstract machine which makes correctness proofs simpler. Furthermore, optimizations of this compilation step can be easily expressed and justified in the functional framework.

Apart from the practical issues discussed in section 6.3, several others research directions like the application of this approach to partial evaluation or to the compilation of λ-prolog should be explored.

Acknowledgments. Thanks are due to Daniel Le Métayer for his comments on an earlier version of this paper.

References

[1] A. W. Appel. *Compiling with Continuations.* Cambridge University Press. 1992.

[2] E. Astesiano and G. Costa. Languages with reducing reflexive types. In *7th Coll. on Automata, Languages and Programming*, LNCS Vol. 85, pp. 38-50, 1980.

[3] H.P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* North-Holland, 1981.

[4] H.P. Barendregt, J.R. Kennaway, J.W. Klop and M.R. Sleep. Needed reduction and spine strategies for the lambda calculus. *Information and Computation*, Vol. 75, pp. 191-231, 1987.

[5] P. Crégut. An abstract machine for the normalization of λ-terms. In *Proc. of the ACM Conf. on Lisp and Functional Programming*, pp. 333-340, 1990.

[6] M. J. Fischer. Lambda-calculus schemata. In *Proc. of the ACM Conf. on Proving Properties about Programs*, Sigplan Notices, Vol. 7(1), pp. 104-109,1972. Revised version in *Lisp and Symb. Comp.*, Vol. 6, Nos. 3/4, 1993.

[7] P. Fradet and D. Le Métayer. Compilation of functional languages by program transformation. *ACM Trans. on Prog. Lang. and Sys.*, 13(1), pp. 21-51, 1991.

[8] G.S. Frandsen and C. Sturtivant. What is an efficient implementation of the λ-calculus? In *Proc. of the ACM Conf. on Functional Prog. Languages and Comp. Arch.*, LNCS Vol. 523, pp. 289-312, 1991.

[9] M. Georgeff. Transformations and reduction strategies for typed lambda expressions. *ACM Trans. on Prog. Lang. and Sys.*, 6(4), pp. 603-631, 1984.

[10] R.J.M. Hughes. Supercombinators, a new implementation method for applicative languages. In *Proc. of the ACM Conf. on Lisp and Functional Programming*, pp. 1-10, 1982.

[11] D. Kranz, R. Kelsey, J. Rees, P. Hudak, J. Philbin and N. Adams. Orbit: An optimizing compiler for scheme. In *proc. of 1986 ACM SIGPLAN Symp. on Comp. Construction*, 219-233, 1986.

[12] J. Lamping. An algorithm for lambda calculus optimal reductions. In *Proc. of the ACM Conf. on Princ. of Prog. Lang.*, pp. 16-30, 1990.

[13] J.-J. Lévy. *Réductions correctes et optimales dans le lambda calcul.* Doctorat d'état, Paris VII, 1978.

[14] G. Nadathur and D.S.Wilson. A representation of lambda terms suitable for operations on their intensions, In *Proc. of the ACM Conf. on Lisp and Functional Programming*, pp. 341-348, 1990.

[15] G. Nadathur and D. Miller. An overview of λProlog. In *Proc. of the 5th Int. Conf. on Logic Prog.*, MIT Press, pp. 810-827, 1988.

[16] G.D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, pp. 125-159, 1975.

[17] J. Staples. A graph-like lambda calculus for which leftmost-outermost reduction is optimal. In *Graph Grammars and their Application*, LNCS vol. 73, pp. 440-455, 1978.

[18] C.P. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus.* PhD thesis, Oxford, 1971.