Weak Reduction and Garbage Collection in Interaction Nets

Jorge Sousa Pinto [1]

Departamento de Informática
Universidade do Minho
4710-057 Braga, Portugal

Abstract

This paper presents an implementation device for the *weak reduction of interaction nets to interface normal form*. The results produced by running several benchmarks are given, suggesting that weak reduction greatly improves the performance of the interaction combinators-based implementation of the λ-calculus.

1 Introduction

The main purpose of this paper is to present an implementation mechanism for weak interaction net reduction. This strategy never triggers reductions in disconnected parts of the net, and in fact it may stop before all the redexes in the connected component (with respect to the interface) are reduced. This mechanism is particularly useful for the evaluation of functional programs.

An interaction net [6] is an undirected graph in which edges are connected to specific positions (called *ports*) in the nodes (with at most one edge connected to each port). Each node is labeled with a symbol and has a distinguished *principal port*. Redexes (here called *active pairs*) are pairs of adjacent nodes connected by their principal ports. The remaining ports of a node are called *auxiliary*. The *interface* of the net is the set of its free ports (i.e. ports with no edge connected).

An *interaction rule* is a graph-rewriting rule which replaces an active pair by a net such that all the connections (i.e. the interface of the active pair) are preserved. Figure 1(a) contains an example of such a rule. An *interaction system* is a set of rules such that there is at most one applicable rule for each pair of symbols. The main property of this rewriting framework is *strong local confluence* (diamond property), a consequence of the fact that each node is involved in no more than one active pair. The number of steps required for fully reducing a net is thus fixed.

[1] Email:`jsp@di.uminho.pt`

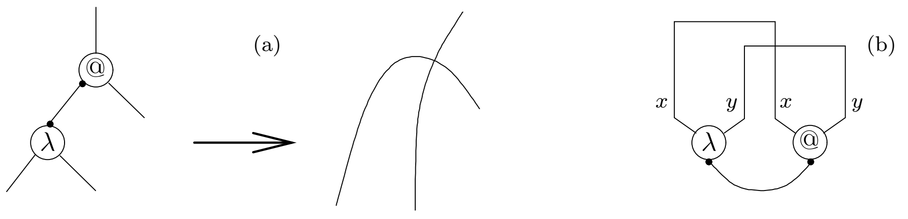

Fig. 1. (a) Example interaction rule, (b) redrawn for conversion to text format

INs and Functional Programming.

Interaction nets have been used as a technique for the implementation of functional programs, characterized by a high degree of *sharing* of computations – interaction nets are notably used when *optimal β-reduction* is sought [1,5,8].

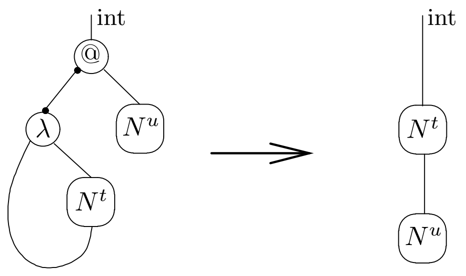

Fig. 2. A net representing the β-redex $(\lambda x.t)u$ – 'int' is its interface; principal ports are marked with a dot; N^t and N^u are the nets representing terms t and u, and the curved wire in the left-hand side represents the bound variable x. A step of reduction (using the rule of figure 1) results in the net on the right

β-redexes are in general encoded as active pairs consisting of a binary @ agent (standing for application) and a binary λ agent (for abstractions), as shown in figure 2. To understand why interaction nets are good at sharing, observe that in the λ-calculus computations are duplicated when a bound variable is substituted in a term where it occurs more than once. These multiple occurrences would typically be encoded with duplicating agents (denoted by δ) which copy the substituted term to obtain the required number of copies.

It is then easy to understand that *copying forces evaluation*, in the sense that, since the principal port of the @ agent is facing the λ agent, it is not possible to duplicate a redex by connecting a δ agent to it: the redex must first be reduced (see figure 3). We remark that this discussion is too simplistic, since the real problem has to do with the duplication of virtual, rather than explicit, redexes.

We remark that it is not obvious from the above discussion how to encode λ-terms as interaction nets so that net reduction soundly corresponds to β-reduction – when variables occur non-linearly, some mechanism must be used to correctly handle the duplication of terms. For instance, each of the encodings (of expressions into nets) reviewed in section 5 handles this problem differently. Studying how such an encoding behaves with respect to the sharing of computations is an extremely complex issue.

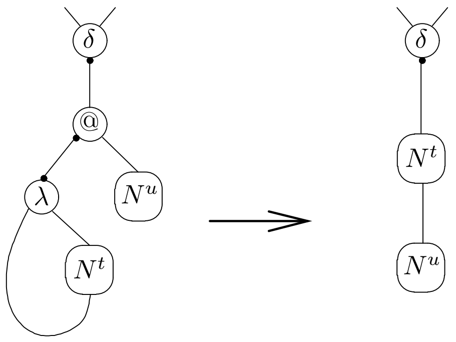

Fig. 3. An attempt to copy a β-redex, forcing evaluation

Garbage collection with Interaction Nets.

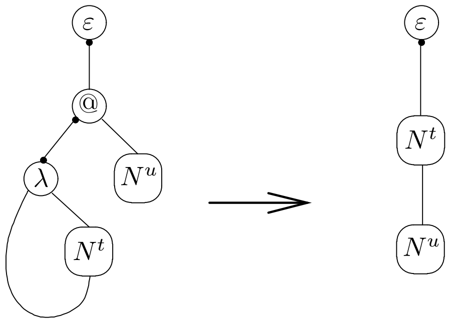

Fig. 4. Erasing a β-redex

In the λ-calculus garbage collection occurs when a bound variable is substituted in a term where it does not occur; this non-occurrence is encoded by an erasing agent (ε). Similarly to duplication, erasing a redex implies first reducing it (see figure 4).

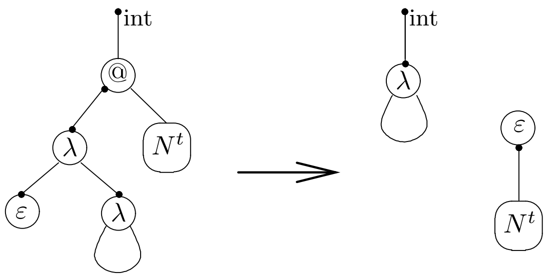

Fig. 5. Reduction of $(\lambda xy.y)t$. The term $\lambda y.y$ can be read at the interface after one step of reduction

To take a concrete example, consider $(\lambda xy.y)t \longrightarrow \lambda y.y$ where t is a big term. Figure 5 shows the corresponding net after one step of reduction, where the net for t becomes disconnected (unreachable from the interface). Reducing this net implies reducing (the net representing) t unnecessarily. If t is non-terminating, this will produce a non-terminating interaction net.

Weak Reduction and Interface Normal Forms.

The above examples show that in some situations full reduction is too strong since it forces the reduction of disconnected nets which may not terminate. Fernández and Mackie [2] have proposed *weak reduction to interface normal form* (WRINF) as an alternative strategy. As in functional programming, where reduction usually stops with weak head normal forms, WRINF stops as soon as a principal port reaches the interface of the net, doing in fact the minimum amount of work necessary for that. In particular, no reductions are performed in disconnected components. If the net encodes a λ-term, reduction stops with the principal port of a λ agent at the interface.

This Paper.

The author has proposed an abstract machine for full interaction net reduction [12]. This machine addressed the implementation issues left open by the calculus (notably those concerning concurrency), by decomposing interaction steps into finer-grained operations. The present paper continues that work by proposing a mechanism which implements the WRINF strategy, in the same spirit of implementation mechanisms for the λ-calculus such as the SECD machine or Krivine's machine, that also force specific reduction strategies.

An important issue to consider when λ-terms are encoded as interaction nets is that the nets that become disconnected may be generated by the encoding, rather than intrinsic to the term (as in the previous examples). Moreover, these disconnected nets are potentially dangerous in the sense that their reduction may generate bigger intermediate nets to be garbage-collected. Section 5 contains a major application of weak reduction: a set of experimental results allow to study the nature of the garbage generated by different encodings of the λ-calculus, leading to a better understanding of the encodings (and in particular their time and space behaviour). The results also offer convincing evidence that weak reduction may dramatically improve the behaviour of interaction net-based program evaluators.

2 Background

In this paper we use a simpler form of nets for which the interface consists of a single free port. These are sufficient for representing closed functional expressions (programs), and moreover all the work here can be easily carried over to the general case. Rather than elaborating on the graphical formalism we now turn to the calculus for interaction nets of Fernández and Mackie [2], simplified for the class of nets we consider.

The Calculus for Interaction Nets.

Nets are reduced in the context of an interaction system $\langle \Sigma, \mathcal{R} \rangle$, where Σ is a set of symbols ranged over by $\alpha, \beta, \ldots$, each symbol α with an arity $\mathsf{ar}(\alpha) \in \mathbf{N}$; and $\mathcal{R}$ is a set of interaction rules. Additionally we use a set of names or variables x, y, z, etc. Terms (ranged over by $t, u, \ldots$) in our language are either names or of the form $\alpha(\boldsymbol{t})$ where $\boldsymbol{t}$ is a sequence of terms $t_1, \ldots, t_n$ and $\mathsf{ar}(\alpha) = n$.

The occurrence of α in $\alpha(\boldsymbol{t})$ is called an *agent* (a node in the graph) and $\mathsf{ar}(\alpha)$ corresponds to the number of its auxiliary ports. A net is represented as a pair $\langle t \mid \Delta \rangle$ consisting of a multiset Δ of unordered pairs of terms (equations) together with a distinguished term t corresponding to the interface of the net. Each variable occurs exactly twice in a net.

Representing Nets in the Calculus.

The following observations explain the relation between the language of the calculus and (graphical) nets:

- A term $t = \alpha(t_1, \ldots, t_n)$ corresponds to a *tree*, with the principal port of α at the root, and each t_i of the form $\beta(u_1, \ldots, u_m)$ a sub-tree (with the principal port of β connected to the ith auxiliary port of α). Variables in t correspond to auxiliary ports which are leaves in the tree.
- The pair of leaves represented by the two occurrences of the same variable (either in the same or in different trees) is connected by an edge.
- Equations of the form $\alpha(\boldsymbol{t}) = \beta(\boldsymbol{u})$ correspond to active pairs.
- The remaining equations (such as $x = t$) are simply edges.

As an example, consider again the net of figure 5. This will be written as $\langle z \mid @(z, u^t) = \lambda(\lambda(y, y), \varepsilon), \Delta^t \rangle$, with $\Sigma = \{@, \lambda\}$. Δ^t is the multiset of equations corresponding to the term t, and u^t its interface. Observe that alternative representations of this net exist, such as the more bureaucratic $\langle z \mid b = \lambda(\lambda(a, a), \varepsilon), @(z, u^t) = b, \Delta^t \rangle$.

We use the notation Δ_1, Δ_2 for $\Delta_1 \cup \Delta_2$ and e, Δ for $\{e\} \cup \Delta$ with e an equation. The set of names $\mathcal{N}(t)$ occurring in a term t is defined in the obvious way and extended to equations and multisets. Substitution is also defined as usual; it will be clear from rules **Indirection** and **Collect** below that substitutions are performed to remove pairs of variables from the net, and as such they can be implemented as assignment (no terms can be erased or duplicated); $t[x := u]$ denotes the term t where x is substituted by u; even though no explicit binding operator is present, we assume α-equivalence of nets for names (denoted by $\equiv$).

An interaction rule $r \in \mathcal{R}$ is written as an equation $t \bowtie u$, where names occur exactly twice. Rules are converted to this format by connecting together corresponding free ports in the left- and right-hand sides; one obtains a net with an active pair that can be written as an equation as explained above. For instance, the rule in figure 1(a) can be written as $@(x, y) \bowtie \lambda(x, y)$ after being redrawn as in figure 1(b).

We now give the three rules of the calculus, allowing to reduce nets:

Interaction $\langle w \mid \alpha(\boldsymbol{t}) = \beta(\boldsymbol{u}), \Delta\rangle \longrightarrow \langle w \mid T, U, \Delta\rangle$, where

$$\begin{aligned}
&\boldsymbol{t} = t_1, \ldots, t_j \text{ and } \boldsymbol{u} = u_1, \ldots, u_k \\
&\alpha(t'_1, \ldots, t'_j) \bowtie \beta(u'_1, \ldots, u'_k) \in \mathcal{R} \\
&T = \{t_1 = t'_1, \ldots, t_j = t'_j\} \text{ and } U = \{u_1 = u'_1, \ldots, u_k = u'_k\}
\end{aligned}$$

Indirection $\langle w \mid x = t, u = v, \Delta\rangle \longrightarrow \langle w \mid u[x := t] = v, \Delta\rangle$ if $x \in \mathcal{N}(u)$

Collect $\langle w \mid x = t, \Delta\rangle \longrightarrow \langle w[x := t] \mid \Delta\rangle$ if $x \in \mathcal{N}(w)$

The first rule of the calculus applies the graphical interaction rules, and the remaining rules handle the bureaucracy introduced by the textual notation.

Remark 2.1 [Variable Convention] In the interaction rule, α-equivalence is used to produce a copy of the rule $\alpha(t'_1, \ldots, t'_j) \bowtie \beta(u'_1, \ldots, u'_k)$ where all variables are fresh.

Properties.

The calculus (as well as the graphical formalism) enjoys strong confluence and uniqueness of normal forms. The latter are either of the form $\langle w \mid \emptyset\rangle$ (called *reduced nets*) or $\langle w \mid x = u\rangle$ where $x \in \mathcal{N}(u)$, corresponding to a *cycle* – a tree together with an edge connecting its root to one of its leaves. Cycles represent *deadlocked* computations; they do not arise in the context of functional programming (or in general whenever typed nets are used).

Example.

Assuming Δ^t empty in our example net, the following reduction sequence would be possible in the interaction system $\langle\{@, \lambda\}, \{@(x, y) \bowtie \lambda(x, y)\}\rangle$:

$$\begin{aligned}
\langle z \mid @(z, u^t) = \lambda(\lambda(y, y), \varepsilon)\rangle \equiv\ & \langle z \mid @(z, u^t) = \lambda(\lambda(a, a), \varepsilon)\rangle \\
\longrightarrow\ & \langle z \mid z = x, u^t = y, \lambda(a, a) = x, \varepsilon = y\rangle \\
\longrightarrow\ & \langle z \mid z = x, \lambda(a, a) = x, \varepsilon = u^t\rangle \\
\longrightarrow\ & \langle z \mid \lambda(a, a) = z, \varepsilon = u^t\rangle \\
\longrightarrow\ & \langle \lambda(a, a) \mid \varepsilon = u^t\rangle
\end{aligned}$$

where the last net corresponds to the right-hand side of figure 5. The above reduction sequence corresponds to an application of the **Interaction** rule, followed by two applications of **Indirection** and finally **Collect**. Reduction would then proceed to garbage collect u^t.

3 Interface Normal Forms and Weak Reduction

A net is in *interface normal form* [2] when one of the following is connected to the interface:

- the principal port of an agent, corresponding to a net of the form $\langle \alpha(\boldsymbol{t}) \mid \Delta \rangle$;
- an auxiliary port of an agent which is part of some cycle, corresponding to a net like $\langle z \mid x = \alpha(\boldsymbol{t}), \Delta \rangle$ where $x, z \in \mathcal{N}(\boldsymbol{t})$.

Observe that reduced nets are interface normal forms: if the equation multiset is empty, the interface cannot be a variable (otherwise its other occurrence would have to be in some equation). We shall take interface normal forms as *canonical* (i.e., no further reductions are performed). *Weak reduction* (which we denote by $\longrightarrow_w$) is obtained by restricting reduction as follows:

> *A net $\langle z \mid \Delta \rangle$ can only be rewritten by applying rules which involve the unique equation $(t = u) \in \Delta$ such that $z \in \mathcal{N}(t) \cup \mathcal{N}(u)$.*

The following examples show that weak reduction is not deterministic:

(i) Let $N = \langle z \mid y = t, z = \alpha(y) \rangle$. Then $N \longrightarrow_w \langle \alpha(y) \mid y = t \rangle$ and $N \longrightarrow_w \langle z \mid z = \alpha(t) \rangle$.

(ii) Let $N' = \langle z \mid y = t, \alpha(z) = \beta(y) \rangle$. Then $N' \longrightarrow_w \langle z \mid \alpha(z) = \beta(t) \rangle$ and a second reduction of N' is possible using the **Interaction** rule.

(iii) Let $N'' = \langle z \mid y = t, x = \alpha(z, y), u = \beta(x) \rangle$. Then $N'' \longrightarrow_w \langle z \mid x = \alpha(z, t), u = \beta(x) \rangle$ and $N'' \longrightarrow_w \langle z \mid y = t, u = \beta(\alpha(z, y)) \rangle$.

Let $\Downarrow$ denote weak reduction to canonical form. The first example above shows that weak reduction does not yield unique canonical forms, for we have $N \Downarrow \langle \alpha(y) \mid y = t \rangle$ and $N \Downarrow \langle \alpha(t) \mid \emptyset \rangle$. Although non-determinism is caused by the indirection rule (a bureaucratic artifact of the calculus), it cannot be eliminated by giving lower priority to this rule (see example iii. above).

Remark 3.1 Canonical forms are unique in *graphical* interaction nets, since weak reduction corresponds to always reducing the active pair which is closest to the interface (a deterministic strategy).

Deterministic Calculus for WRINF.

The calculus stands closer to the implementation level than the graphical formalism (nets in the calculus can easily be seen as data-structures). The first step towards the design of an abstract machine is then to force determinism for weak reduction. We do this by introducing the following modified calculus:

Interaction $\langle z \mid \alpha(\boldsymbol{t}) = \beta(\boldsymbol{u}), \Delta \rangle \longrightarrow_w \langle z \mid T, U, \Delta \rangle$ if $z \in \mathcal{N}(\boldsymbol{t}) \cup \mathcal{N}(\boldsymbol{u})$, where

$$\boldsymbol{t} = t_1, \dots, t_j \text{ and } \boldsymbol{u} = u_1, \dots, u_k$$

$$\alpha(t'_1, \dots, t'_j) \bowtie \beta(u'_1, \dots, u'_k) \in \mathcal{R}$$

$$T = \{t_1 = t'_1, \dots, t_j = t'_j\} \text{ and } U = \{u_1 = u'_1, \dots, u_k = u'_k\}$$

Indirection $\langle z \mid x = \alpha(\boldsymbol{t}), u = v, \Delta\rangle \longrightarrow_w \langle z \mid u[x := \alpha(\boldsymbol{t})] = v, \Delta\rangle$ if $z \in \mathcal{N}(\boldsymbol{t})$ and $x \in \mathcal{N}(u)$

Collect $\langle z \mid z = t, \Delta\rangle \longrightarrow_w \langle t \mid \Delta\rangle$

These rules apply only to configurations whose interface is a variable; moreover there are now additional conditions for the first two rules. The key point is that reduction is directed by where the interface variable occurs.

It is immediate to see that there is a unique reduction for each of the previous examples using this calculus. Henceforth the symbol $\longrightarrow_w$ will refer to weak reduction as defined by the calculus.

Properties.

The calculus performs weak reductions only, and clearly a single rule applies to each net: if the variable in the interface occurs directly as a member in some equation, rule **Collect** applies; otherwise it must occur in a term of the form $\alpha(\boldsymbol{t})$ and only one of the rules **Interaction** or **Indirection** will apply. This immediately yields uniqueness of normal forms.

The calculus stops exactly with interface normal forms: if the interface is not a variable or the interface variable occurs in a cycle, no rule applies, and some rule always applies to a net that is not in interface normal form.

Example.

The following is an example reduction in this calculus:

$$\begin{aligned}
\langle z \mid b = \lambda(\lambda(a,a),\varepsilon), @(z,u^t) = b\rangle &\longrightarrow_w \langle z \mid @(z,u^t) = \lambda(\lambda(a,a),\varepsilon)\rangle \\
&\longrightarrow_w \langle z \mid z = x, u^t = y, \lambda(a,a) = x, \varepsilon = y\rangle \\
&\longrightarrow_w \langle x \mid u^t = y, \lambda(a,a) = x, \varepsilon = y\rangle \\
&\longrightarrow_w \langle \lambda(a,a) \mid u^t = y, \varepsilon = y\rangle
\end{aligned}$$

This reduction sequence is unique and ends with an interface normal form.

4 The Abstract Machine

An efficient algorithm for implementing WRINF can be read from the deterministic calculus. We will express the algorithm as an abstract machine acting on configurations – the data-structures used to represent interaction nets. For any net $\langle z \mid \Delta\rangle$ the multiset of equations Δ will be represented by a sequence; we will enforce the following invariant:

The unique equation where the interface variable z occurs always occupies the first position in the sequence L representing Δ.

In fact L will contain *annotated equations* – pairs (e, n), which we will write simply as $e^{\mathbf{n}}$, where e is an equation and $\mathbf{n}$ is a positive integer, the *address* of the equation in the sequence. The algorithm (specifically the **Indirection** rule) requires access to equations other than that where the interface variable occurs; the notion of *address* allows for access in constant time. We remark that an address is simply an integer uniquely assigned to an equation, which does not necessarily correspond to the position of the equation in the sequence.

Definition 4.1 A *configuration* is a tuple $\langle t \mid L \mid \phi \mid \mathcal{P}\rangle$, where t is a term and L a finite sequence of annotated equations. Let $\mathsf{Names} = \mathcal{N}(t) \cup \mathcal{N}(L)$; then $\phi : \mathsf{Names} \longrightarrow \mathsf{Names}$ is an involutive fixpoint-free function (i.e, $\phi(x) = y$ implies $y \neq x$ and $\phi(y) = x$), and the partial map $\mathcal{P} : \mathsf{Names} \longrightarrow \mathbf{N}$ assigns addresses to names. Each $x \in \mathsf{Names}$ occurs once in the configuration (either in t or in L).

The two occurrences of each name x in a net are replaced by a pair of names x_1, x_2 such that $\phi(x_1) = x_2$, following the abstract machine for full reduction [12]. This allows for nets to be represented as sets of *trees*, together with the additional linking information in ϕ. This splitting of names also applies to interaction rules.

The invariant on a configuration $\langle z \mid L \mid \phi \mid \mathcal{P}\rangle$ can be restated as follows: the variable $\phi(z)$ always occurs in the head equation in L (see lemma 4.5). $\mathcal{P}$ associates to each variable the address of the equation where it occurs, which will be essential for preserving the invariant.

Notation and Conventions.

Interaction rules are written as $t \overset{\Phi}{\bowtie} u$ where Φ is the involution collecting the linking information for the split variables. We write $\mathcal{P}[x \mapsto m]$ for $\mathcal{P} \cup \{(x, m)\}$ and $\phi[x \leftrightarrow y]$ for $\phi \cup \{(x, y), (y, x)\}$. The symbol ',' is used as a separator in sequences; the same symbol is overloaded to represent sequence concatenation. $\mathcal{P}_{\backslash [x_1,\ldots,x_n]}$ denotes the function obtained by excluding $x_1, \ldots, x_n$ from the domain of $\mathcal{P}$. The operator $\mathrm{tl}(\cdot)$ returns the tail of a sequence. In the presentation of the abstract machine, we assume that pairs of terms are unordered, i.e., if necessary members are swapped before a rule is applied. The variable convention applies as in section 2.

The abstract machine rules used for rewriting configurations are given in table 1. We make here some key observations. First of all, we observe that the rule to apply to a configuration is given by the form of the head equation in the list (where $\phi(z)$ occurs).

- The **AgAg** rule performs an interaction, since the equation where $\phi(z)$ occurs is an active pair. This rule adds to the involution in the current configuration the Φ information in the interaction rule.
- The **VarVar** rule handles the case where the head equation has the form $x = y$, where $\phi(y)$ is the interface variable z and $\phi(x)$ occurs in some equation whose address (given by $\mathcal{P}$) is m. The rule manipulates the involution changing $\phi(z)$; consequently the equation with address m must be moved

AgAg $\langle z \mid \alpha(t_1,\dots,t_j) = \beta(u_1,\dots,u_k)^{\mathbf{n}}, S \mid \phi \mid \mathcal{P}\rangle$
$\quad\longrightarrow \langle z \mid \hat{L}, S \mid \phi \cup \Phi \mid \mathcal{P} \setminus \mathcal{P}^0 \cup \hat{\mathcal{P}}\rangle$

where $\alpha(t'_1,\dots,t'_j) \overset{\Phi}{\bowtie} \beta(u'_1,\dots,u'_k) \in \mathcal{R}$, $\phi(z)$ occurs in t_i $(1 \le i \le j)$, and

$$\hat{L} = t_i = t_i'^{\mathbf{n+k+j-1}}, t_j = t_j'^{\mathbf{n+k+j-2}}, \dots, t_{i+1} = t_{i+1}'^{\mathbf{n+k+i-1}}, t_{i-1} = t_{i-1}'^{\mathbf{n+k+i-2}},$$
$$\dots, t_1 = t_1'^{\mathbf{n+k}}, u_k = u_k'^{\mathbf{n+k-1}}, \dots, u_1 = u_1'^{\mathbf{n}}$$
$$\mathcal{P}^0 = \{x \mapsto n \mid x \in \mathcal{N}(t_1) \cup \cdots \cup \mathcal{N}(t_j) \cup \mathcal{N}(u_1) \cup \cdots \cup \mathcal{N}(u_k)\}$$
$$\hat{\mathcal{P}} = \{x \mapsto n \mid x \in \mathcal{N}(e), e^{\mathbf{n}} \in \mathrm{tl}(\hat{L})\}$$

VarVar $\langle z \mid x = y^{\mathbf{n}}, S_1, e^{\mathbf{m}}, S_2 \mid \phi[x \leftrightarrow x', y \leftrightarrow z] \mid \mathcal{P}[x' \mapsto m]\rangle$
$\quad\longrightarrow \langle z \mid e^{\mathbf{n}}, S_1, S_2 \mid \phi[x' \leftrightarrow z] \mid \mathcal{P}_{\setminus[x,y]}\rangle$

VarAg $\langle z \mid x = \alpha(\boldsymbol{t})^{\mathbf{n}}, S_1, e^{\mathbf{m}}, S_2 \mid \phi[x \leftrightarrow x'] \mid \mathcal{P}[x' \mapsto m]\rangle$
$\quad\longrightarrow \langle z \mid e[x' := \alpha(\boldsymbol{t})]^{\mathbf{n}}, S_1, S_2 \mid \phi \mid \mathcal{P}_{\setminus[x,x']}\rangle \qquad$ if $x' \neq z$

Interface $\langle z \mid x = \alpha(\boldsymbol{t})^{\mathbf{n}}, S \mid \phi[x \leftrightarrow z] \mid \mathcal{P}\rangle \longrightarrow \langle \alpha(\boldsymbol{t}) \mid S \mid \phi \mid \mathcal{P}_{\setminus[x,z]}\rangle$

Table 1
Abstract Machine Rules

to the head of the list (to preserve the invariant).

- **VarAg** corresponds to the remaining case of the Indirection rule of the calculus. Observe that $\phi(z)$ occurs in $\alpha(\boldsymbol{t})$, which will substitute some variable in another equation, that must thus be moved to the head of the list.
- **Interface** is a restricted version of the Collect rule of the calculus.

With respect to how the $\mathcal{P}$ component is updated and used, we remark that it is not necessary for $\mathcal{P}$ to store the addresses of variables occurring in the first equation in the list. $\mathcal{P}$ will not be consulted for variables in this equation (since we assume cycles do not arise) and it may even contain incorrect addresses for these variables. For this reason, rules **VarVar** and **VarAg** need not update in $\mathcal{P}$ the addresses of variables that are moved to the head of the list. The **AgAg** rule on the other hand must update information in $\mathcal{P}$ corresponding to all the equations (but the first) added to the configuration.

We now show how an initial configuration is constructed:

Definition 4.2 From any (non-canonical) interaction net $N = \langle z \mid \Delta\rangle$ one obtains a corresponding *initial configuration* $\mathcal{C}_{(N)} = \langle z \mid L \mid \phi \mid \mathcal{P}\rangle$ by replacing (specializing) the two occurrences of each variable x by x_1 and x_2 in z and Δ; then $\phi \doteq \{(x_1,x_2),(x_2,x_1) \mid x \in \mathcal{N}(\Delta)\}$; let $e_1, e_2, e_3, \dots e_n$ be any enumeration of the elements of Δ, then $L \doteq e_i^{\mathbf{n}}, e_n^{\mathbf{n-1}}, \dots, e_{i+1}^{\mathbf{i}}, e_{i-1}^{\mathbf{i-1}}, \dots, e_1^{\mathbf{1}}$ where $\phi(z) \in \mathcal{N}(e_i)$; $\mathcal{P} \doteq \{(x,n) \mid x \in \mathcal{N}(e), e^{\mathbf{n}} \in \mathrm{tl}\,L\}$.

Definition 4.3 The *interpretation* of a configuration $\mathcal{C} = \langle t \mid L \mid \phi \mid \mathcal{P} \rangle$ is an interaction net $[\![\mathcal{C}]\!] \doteq \langle t' \mid \Delta \rangle$ where t', Δ are obtained by replacing in t and L all pairs of variables x, y such that $\phi(x) = y$ by a single (fresh) name $v_{x,y}$.

The interpretation of a configuration allows to read back a net from it. The following lemma is straightforward to prove:

Lemma 4.4 *Let N be a net; then $[\![\mathcal{C}_{(N)}]\!] \equiv N$.*

We now give a series of results concerning properties of the abstract machine. The reader is referred to the full version of this paper for the proofs of these results.

Lemma 4.5 *Let $\mathcal{C}_0$ be an initial configuration and $\mathcal{C}_0 \longrightarrow^* \mathcal{C} = \langle z \mid e_0^{\mathbf{n}}, L \mid \phi \mid \mathcal{P} \rangle$; then*

(i) *For $m \neq n$, $\mathcal{P}(x) = m$ iff $x \in \mathcal{N}(e)$ with $e^{\mathbf{m}} \in L$; moreover if $e'^{\mathbf{m}} \in L$ then $e' = e$.*

(ii) *$\phi(z) \in \mathcal{N}(e_0)$;*

Lemma 4.6 *Let $\mathcal{C}_0$, $\mathcal{C}$, $\mathcal{C}'$ be configurations such that $\mathcal{C}_0$ is initial, $\mathcal{C}_0 \longrightarrow^* \mathcal{C}$, and $\mathcal{C} \longrightarrow \mathcal{C}'$; then $[\![\mathcal{C}]\!] \longrightarrow_w N$ and $N \equiv [\![\mathcal{C}']\!]$.*

Lemma 4.7 *Let N, N' be nets such that $N \longrightarrow_w N'$, and $\mathcal{C}$ a configuration such that $[\![\mathcal{C}]\!] \equiv N$ and $\mathcal{C}_0 \longrightarrow^* \mathcal{C}$ for some initial configuration $\mathcal{C}_0$; then $\mathcal{C} \longrightarrow \mathcal{C}'$ and $[\![\mathcal{C}']\!] \equiv N'$.*

The following correctness result follows in a straightforward way from lemmas 4.4, 4.6, and 4.7.

Proposition 4.8 (Correctness) *Let N_0 be an interaction net; then*

$$\mathcal{C}_{(N_0)} \longrightarrow^* \mathcal{C} \quad \textit{iff} \quad N_0 \longrightarrow_w^* N$$

and $N \equiv [\![\mathcal{C}]\!]$. Here $\longrightarrow^*$ denotes the reflexive and transitive closure of a reduction relation (modulo equivalence of nets in the case of net reduction).

Example.

We take the example interaction net used in section 3:

$$N = \langle z \mid b = \lambda(\lambda(a,a),\varepsilon), @(z,u^t) = b \rangle$$

An initial configuration for this net is:

$$\mathcal{C}_{(N)} = \langle z_1 \mid @(z_2,u^t) = b1^{\mathbf{2}}, b_2 = \lambda(\lambda(a_1,a_2),\varepsilon)^{\mathbf{1}} \mid a_1 \leftrightarrow a_2, b_1 \leftrightarrow b_2, z_1 \leftrightarrow z_2 \mid a_1 \mapsto 1, a_2 \mapsto 1, b_2 \mapsto 1 \rangle$$

and the following is the interaction rule for $@ \bowtie \lambda$, after variables have been split:

$$@(x_1,y_1) \overset{\Phi}{\bowtie} \lambda(x_2,y_2), \text{ with } \Phi = \{x_1 \leftrightarrow x_2, y_1 \leftrightarrow y_2\}$$

The unique reduction sequence for this initial configuration is:

$$
\begin{aligned}
\mathcal{C}_{(N)} \longrightarrow\ & \langle z_1 \mid @(z_2, u^t) = \lambda(\lambda(a_1, a_2), \varepsilon)^{\mathbf{2}} \mid \\
& a_1 \leftrightarrow a_2, z_1 \leftrightarrow z_2 \mid \\
& a_1 \mapsto 1, a_2 \mapsto 1\rangle \\
\longrightarrow\ & \langle z_1 \mid z_2 = x_1^{\mathbf{5}}, u^t = y_1^{\mathbf{4}}, \lambda(a_1, a_2) = x_2^{\mathbf{3}}, \varepsilon = y_2^{\mathbf{2}} \mid \\
& a_1 \leftrightarrow a_2, z_1 \leftrightarrow z_2, x_1 \leftrightarrow x_2, y_1 \leftrightarrow y_2 \mid \\
& a_1 \mapsto 3, a_2 \mapsto 3, x_2 \mapsto 3, y_1 \mapsto 4, y_2 \mapsto 2\rangle \\
\longrightarrow\ & \langle z_1 \mid \lambda(a_1, a_2) = x_2^{\mathbf{5}}, u^t = y_1^{\mathbf{4}}, \varepsilon = y_2^{\mathbf{2}} \mid \\
& a_1 \leftrightarrow a_2, z_1 \leftrightarrow x_2, y_1 \leftrightarrow y_2 \mid \\
& a_1 \mapsto 3, a_2 \mapsto 3, x_2 \mapsto 3, y_1 \mapsto 4, y_2 \mapsto 2\rangle \\
\longrightarrow\ & \langle \lambda(a_1, a_2) \mid u^t = y_1^{\mathbf{4}}, \varepsilon = y_2^{\mathbf{2}} \mid \\
& a_1 \leftrightarrow a_2, y_1 \leftrightarrow y_2 \mid \\
& a_1 \mapsto 3, a_2 \mapsto 3, y_1 \mapsto 4, y_2 \mapsto 2\rangle
\end{aligned}
$$

Where rules **VarAg**, **AgAg**, **VarVar**, and **Interface** were used, in this order. The last net is a normal form of the abstract machine and has as interpretation the following net:

$$\overline{N} = \langle \lambda(a, a) \mid u^t = y, \varepsilon = y\rangle$$

We recall that throughout reduction, the map $\mathcal{P}$ may contain "wrong" addresses for variables occurring in the head of the list.

Implementation Issues.

Consider a configuration $\langle t \mid L \mid \phi \mid \mathcal{P}\rangle$. The sequence of equations L has an obvious linked implementation, where each node is a pair of trees corresponding to terms with uniquely occurring variables (the leaves in the trees). t is yet one such term. The involution ϕ is kept locally as pointers between pairs of variables (leaves in trees). The $\mathcal{P}$ component is also implemented locally by pointers (stored in variables) to nodes in the linked list.

As an example we explain how the **VarAg** rule is implemented: the ϕ pointer is stored as a field in the structure representing variable x, which contains the address of the structure representing x'; the $\mathcal{P}$ pointer is stored as a field in x', and contains the address $\mathbf{m}$ of a node in the linked list of equations; simple pointer operations allow to (i) move the node with address $\mathbf{m}$ to the head of the list and (ii) substitute the term $\alpha(\boldsymbol{t})$ for x' in the equation stored in this node. Since no searching is required, all these operations are done in constant time.

The **AgAg** rule is computationally heavier than its equivalent in the full-reduction version of the abstract machine. In particular, the work involved in building the $\hat{\mathcal{P}}$ mapping represents the overhead of implementing weak reduction. This can be done in linear time in the number of variables occurring in the head equation in the configuration.

5 Application to Encodings of the λ-calculus

In the optimal reduction systems [5,8] the number of graphical rewrite steps which actually correspond to β-reductions is optimal in the sense of Lévy [9]; however many other steps are needed to support that optimality – the overhead is quite significant. Asperti and colleagues have addressed this problem by introducing rewrite rules which do not fit in the strict definition of interaction but can dramatically improve performance.

Other systems have been proposed that give up optimality, opting instead to reduce the total number of interactions performed – since interaction steps are performed in constant time, this is a measure of the actual cost of evaluating an expression. The following two encodings follow the latter approach; both result from work on translations of Linear Logic [4] proofs into INs:

Yale Encoding [10] This encoding forces *closed reduction* of the encoded terms; it captures locally the notion of an exponential box (from Linear Logic proof nets) by means of boundary agents which allow substitutions to be carried over inside abstractions. The encoding requires garbage collection of the boundary agents when a box is opened.

Combinators Encoding [11] This was motivated by the work of Lafont [7], who introduced a set of agents capable of simulating every interaction net system. This is the simplest encoding (the combinators interaction system consists of 3 agents and 6 rules) but surprisingly it results in a high level of sharing – the number of β-reductions is lower than for Yale. The drawback is that the total number of interactions is in general very high, and for this reason the combinators system has not been considered to be useful in practice. One of the three agents in this system is precisely an eraser agent, which can easily generate disconnected nets.

This section contains a direct application of weak reduction to these two encodings. We saw in section 1 that garbage collection is triggered in λ-terms containing non-occurring bound variables. We now turn to a different sort of garbage: our goal here is to study the nature of the disconnected nets generated by the *dynamics of the encoding*. The process of garbage collecting these nets may be harmful in the sense that it may generate bigger intermediate nets – a situation which would clearly benefit from weak reduction.

This issue has been studied in BOHM, where the encoding of terms generates garbage which should be collected as soon as possible [1]. This not only prevents a space explosion but it also affects efficiency. The data given below allows for a better understanding of how garbage collection should be handled in the Yale and Combinators encodings, and of whether the encodings behave significantly better when WRINF is used rather than full reduction.

We give in table 2 experimental results concerning the reduction of two sequences of λ-terms: **N2II** and **N22II**, for **N** between **1** and **10**, where the numbers correspond to Church numerals. These sequences generate computations which greatly benefit from the sharing allowed by interaction nets.

The results are given for full reduction and for WRINF implemented by the abstract machine of section 4. We remark that for these terms both strategies result in full normal forms (the term $\lambda x.x$ can be read from the interface). The differences between the two strategies concern reduction in the disconnected parts of the nets only.

For each term, we give the total number of interactions obtained with each strategy; the number of interactions which correspond to β-reductions steps; and finally (under the heading "Space"), the number of cells contained in the disconnected components of the nets when WRINF is used. Observe that the machine used here does not perform garbage collection during reduction; the blocked disconnected nets are simply erased once an interface normal form has been reached. Their size is thus a measure of the work involved in this post-reduction garbage collection.

	Yale				Comb			
	Full Red.	WRINF	Space	β-steps	Full Red.	WRINF	Space	β-steps
22II	43	27	21	9	253	42	65	9
32II	67	44	28	12	559	70	90	12
42II	91	61	35	15	1121	98	115	15
52II	115	78	42	18	2195	126	140	18
'10'2II	235	163	77	33	65933	266	265	33
222II	127	88	46	20	1269	149	145	18
322II	383	280	110	51	17031	319	270	29
422II	4395	3292	1110	550	4195705	625	507	48
522II	1051207	788464	262750	131369		1203	968	83
'10'22II						35101	28809	2082

Table 2
Experimental Results

We make the following observations:

(i) The two sequences differ in that (for both encodings) the number of β steps is linear in **N** for **N2II**, and exponential in **N** for **N22II**.

(ii) In the latter case, the exponential growth in the number of β steps is much faster for the Yale encoding than for the Combinators encoding.

(iii) For the Yale encoding the total number of interactions performed is linear (for both sequences) in the number of β steps; the effect of weak reduction is to lower the linearity constant.

(iv) For the Combinators encoding the total number of interactions using full reduction is exponential in **N** for both sequences; the effect of weak reduction is to restore its linearity.

With respect to the size of the disconnected nets when WRINF is used, we remark that for Yale the size of the disconnected nets is equal to the difference between the number of steps obtained using full reduction and weak reduction (minus a constant corresponding to the number of eraser agents in the disconnected net). This means that reduction of the disconnected nets does not generate more garbage: exactly the same work must be done cleaning up the disconnected nets (after WRINF) as if full reduction is used. We conclude that weak reduction is not useful with respect to the garbage generated by Yale (although it *would* still be useful for terms generating intrinsic garbage).

In the combinators system weak reduction brings a major revelation: most of the work involved in (fully) reducing a term is in fact useless for the purpose of evaluation. The size of the disconnected nets is linear in the number of β-reductions, while the work saved is exponential; the disconnected nets contain many potential interactions which are blocked by weak reduction.

Finally, we remark that when weak reduction is used, the Combinators encoding behaves asymptotically better than Yale, since both the number of interactions and the size of the disconnected net are linear in the number of β-steps, which is asymptotically lower than for Yale. In fact the Combinators encoding seems to behave even better than BOHM for the second sequence (even though the optimal number of β steps is linear in **N**, the total work needed to support that optimality is asymptotically higher than for the Combinators encoding).

6 Conclusions and Further Work

We have proposed a simple device for implementing weak reduction of interaction nets, thus correcting an operational problem of interaction net-based implementations of functional languages, which arises when disconnected nets are generated. In this context, the present work shows that the interaction combinators are an excellent choice for encoding λ-terms as nets, with respect to both the number of β-reduction steps and the total number of interactions.

Weak reduction certainly reveals something about the dynamics of the Yale and Combinators encodings; however, no theoretical results exist allowing to compare these with the optimal reducer (specifically, concerning the global work required to support the underlying strategies).

Our experimental results indicate that the size of the disconnected nets may be quite significant; in particular for the sequence **N22II** this size grows exponentially with **N**, which may cause a space explosion of the evaluator program. The solution is of course to somehow erase the disconnected nets as they are generated, without waiting for weak reduction to finish. The examples in section 1 show that this cannot be done without leaving the interaction framework: if an eraser agent ε is to erase an active pair, then it must be capable of interacting with the application agent through one of its auxiliary ports. The full version of this paper includes some preliminary work on how the abstract machine can handle non-interaction rules (inspired by the BOHM garbage collection rules, which are always executed as soon as possible).

References

[1] A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[2] M. Fernández and I. Mackie. A calculus for interaction nets. In G. Nadathur, editor, *Proceedings of the International Conference on Principles and Practice of Declarative Programming (PPDP'99)*, number 1702 in Lecture Notes in Computer Science, pages 170–187. Springer-Verlag, September 1999.

[3] M. Fernández, I. Mackie, and J. S. Pinto. A higher-order calculus for graph transformation. In *Proceedings of the International Workshop on Term Graph Rewriting, TERMGRAPH 2002*, volume 72 of *Electronic Notes in Theoretical Computer Science*.

[4] J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[5] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages (POPL'92)*, pages 15–26. ACM Press, Jan. 1992.

[6] Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, Jan. 1990.

[7] Y. Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997.

[8] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 16–30. ACM Press, Jan. 1990.

[9] J.-J. Lévy. Optimal reductions in the lambda calculus. In J. P. Hindley and J. R. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, 1980.

[10] I. Mackie. YALE: Yet another lambda evaluator based on interaction nets. In *Proceedings of the 3rd ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*, pages 117–128. ACM Press, September 1998.

[11] I. Mackie and J. S. Pinto. Encoding linear logic with interaction combinators. *Information and Computation*, 176(2):153–186, 2002.

[12] J. S. Pinto. Sequential and Concurrent Abstract Machines for Interaction Nets. In J. Tiuryn, editor, *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*, number 1784 in Lecture Notes in Computer Science, pages 267–282. Springer-Verlag, 2000.