Electronic Communications of the EASST
Volume 29 (2010)

Proceedings of the Ninth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2010)

A lightweight abstract machine for interaction nets

Abubakar Hassan, Ian Mackie and Shinya Sato

12 pages

Guest Editors: Jochen Küster, Emilio Tuosto
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A lightweight abstract machine for interaction nets

Abubakar Hassan[1], **Ian Mackie**[2] **and Shinya Sato**[3]

[1] Department of Informatics, University of Sussex, Falmer, Brighton BN1 9QJ, UK

[2] LIX, CNRS UMR 7161, École Polytechnique, 91128 Palaiseau Cedex, France

[3] Himeji Dokkyo University, Faculty of Econoinformatics, 7-2-1 Kamiohno, Himeji-shi, Hyogo 670-8524, Japan

Abstract: We present a new abstract machine for interaction nets and demonstrate that an implementation based on the ideas is significantly more efficient than existing interaction net evaluators. The machine, which is founded on a chemical abstract machine formulation of interaction nets, is a simplification of a previous abstract machine for interaction nets. This machine, together with an implementation, is at the heart of current work on using interaction nets as a new foundation as an intermediate language for compiler technology.

Keywords: Interaction nets, programming languages, abstract machine

1 Introduction

Interaction nets [Laf90] are a graphical model of computation. It is possible to program with interaction nets [HMS09, Mac05] and they also serve as an intermediate language for implementing other programming languages. Some examples are encodings of λ-calculus, and simple functional programming languages (amongst others, see for instance [AG98, GAL92, Mac98]).

One reason why they have been very successful at implementing other programming languages is that a compilation must explain all the components of a computation. What is rare, is that the compilation can give something back, and this has been observed with the encodings on the λ-calculus where new strategies for reduction have been found. One of the reasons for this is because interaction nets naturally capture sharing, indeed one has to work hard to simulate reduction strategies where duplication of work takes place.

In [FM99] a calculus was given which provided a foundation for the operational understanding of interaction nets. This calculus led to the development of an abstract machine [Pin00], which in turn led to a very efficient implementation of interaction nets.

Recently, there have been new developments in the foundations for a calculus of interaction nets. The purpose of this paper is to outline these ideas which led to the main contribution of the paper which is an abstract machine founded on the new calculus. This in turn has led to the development of new implementations of interaction nets which are the most efficient that we are aware of to date.

One of the main hopes of this work is that it provides a new foundation for a research programme to build implementations of programming languages through interaction nets: an improvement in the implementation technology for nets will have an impact on all the compilers

developed.

The main contributions of this paper are:

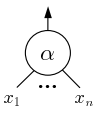
- We define a new term calculus of interaction nets. The novelty is that the notion of substitution is simplified in that it just replaces a name.
- We simplify and improve Pinto's abstract machine [Pin00] by using this calculus. The main improvement is due to the fact that we no longer need to maintain lists of names, and consequently the transition rules become significantly more simple.
- We have built a prototype implementation based on the ideas. We demonstrate that we get a factor of ten improvement over previous implementations, and this implementation is thus the most efficient evaluator to date.

Overview. The rest of this paper is structured as follows. In the next section we review what we need about interaction nets. In Section 3 we give our new calculus. Section 4 gives the abstract machine, and gives studied properties of it. We conclude the paper in Section 5.

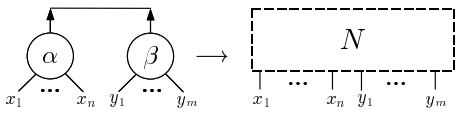
2 Interaction nets

Here we review the basic notions of interaction nets. We refer the reader to [Laf90] for a more detailed presentation. Interaction nets are specified by the following data:

- A set Σ of *symbols*. Elements of Σ serve as *agent* (node) labels. Each symbol has an associated arity *ar* that determines the number of its *auxiliary ports*. If $ar(\alpha) = n$ for $\alpha \in \Sigma$, then α has $n+1$ *ports: n* auxiliary ports and a distinguished one called the *principal port*.



- A *net* built on Σ is an undirected graph with agents at the vertices. The edges of the net connect agents together at the ports such that there is only one edge at every port. A port which is not connected is called a *free port*. A set of free ports is called an *interface*.
- Two agents $(\alpha, \beta) \in \Sigma \times \Sigma$ connected via their principal ports form an *active pair* (analogous to a redex). An interaction rule $((\alpha, \beta) \longrightarrow N) \in \mathscr{R}_{in}$ replaces the pair (α, β) by the net N. All the free ports are preserved during reduction, and there is at most one rule for each pair of agents. The following diagram illustrates the idea, where N is any net built from Σ.



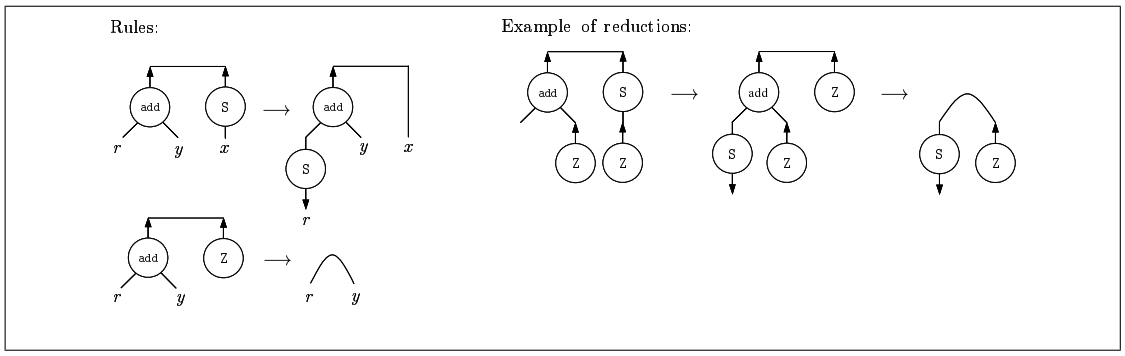


Figure 1: An example of a system of interaction nets

We use the relation $\longrightarrow$ for the one step reduction and $\longrightarrow^*$ for its transitive and reflexive closure. Interaction nets have the following property [Laf90]:

Proposition 1 (Strong Confluence) *Let N be a net. If $N \longrightarrow N_1$ and $N \longrightarrow N_2$ with $N_1 \neq N_2$, then there is a net N_3 such that $N_1 \longrightarrow N_3$ and $N_2 \longrightarrow N_3$.*

Figure 1 shows a classical example of an interaction net system that encodes the addition operation. We can represent numbers using the agents S to represent the successor function $(n \mapsto n+1)$ and Z to represent the number 0. The left of the figure contains the two addition rules which we leave the reader to relate to the standard equational term rewriting system definition of addition. The right of the figure gives an example reduction sequence which shows how a net representing $0+1$ is reduced to 1 using the given rules.

2.1 The calculus for interaction nets

In this section we review the calculus for interaction nets proposed by Fernández and Mackie [FM99]. We begin by introducing a number of syntactic categories:

Names Let $\mathcal{N}$ be a set of *names* ranged over by $x, y, z, x_1, x_2, \ldots$. We write $\bar{x}, \bar{y}, \ldots$ for sequences of names. We assume $\mathcal{N}$ and Σ are disjoint.

Terms are built from Σ and $\mathcal{N}$ using the grammar: $t ::= x \mid \alpha(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms, $\alpha \in \Sigma$ and $ar(\alpha) = n$. If $ar(\alpha) = 0$, then we omit brackets and write just α. We use $t, s, u, \ldots$ to range over terms and $\bar{t}, \bar{s}, \bar{u}, \ldots$ over sequences of terms.

Equations have the form: $t = s$, where t and s are terms. Equations are elements of computation. Given $\bar{t} = t_1, \ldots, t_k$ and $\bar{s} = s_1, \ldots, s_k$, we write $\bar{t} = \bar{s}$ to denote the list $t_1 = s_1, \ldots, t_k = s_k$. We use $\Delta, \Theta, \ldots$ to range over multisets of equations.

Configurations have the form: $\langle\ \bar{t} \mid \Delta\ \rangle$, where $\bar{t}$ is a sequence of terms representing the interface of the net and Δ is a sequence of equations. All names occur at most twice in a configuration. We use $C_1, C_2, \ldots$ to range over configurations. Configurations that differ only on names are considered equivalent.

Interaction rules have the form: $\alpha(t_1,...,t_n) \bowtie \beta(s_1,...,s_k)$, where $\alpha(t_1,...,t_n)$ and $\beta(s_1,...,s_k)$ are terms. This notation for rules was introduced by Lafont [Laf90] and we refer to it as Lafont's style. All names occur exactly twice in a rule, and there should be at most one rule between any pair of agents in $\mathscr{R}$. $\mathscr{R}$ is closed under symmetry, thus if $\alpha(\bar{t}) \bowtie \beta(\bar{s}) \in \mathscr{R}$ then $\beta(\bar{s}) \bowtie \alpha(\bar{t}) \in \mathscr{R}$.

Definition 1 (Bound names) If a name x occurs twice in a term t, then we say x is *bound*. We extend this notion to equations, sequences of terms, and multiset of equations.

The calculus consists of three reduction rules which reduce (valid) configurations.

Indirection:
$\langle\, \bar{t} \mid x = t, u = s, \Delta \,\rangle \longrightarrow_i \langle\, \bar{t} \mid u[t/x] = s, \Delta \,\rangle$ where x occurs in u,

Collect:
$\langle\, \bar{t} \mid x = t, \Delta \,\rangle \longrightarrow_c \langle\, \bar{t}[t/x] \mid \Delta \,\rangle$ where x occurs in $\bar{t}$,

Interaction:
$\langle\, \bar{t} \mid \alpha(\bar{t_1}) = \beta(\bar{t_2}), \Delta \,\rangle \longrightarrow_{\bowtie} \langle\, \bar{t} \mid \bar{t_1} = \bar{s'}, \bar{t_2} = \bar{u'}, \Delta \,\rangle$
where $\alpha(\bar{s}) \bowtie \beta(\bar{u}) \in \mathscr{R}$ and $\bar{s'}$ and $\bar{u'}$ are the result of replacing each occurrence of a bound name x for $\alpha(\bar{s}) \bowtie \beta(\bar{u})$ by a fresh name x' respectively.

Example 1 *The example rules in Figure 1 can be represented using Lafont's style* [1] *as:*

$$\mathtt{add}(S(x), y) \bowtie S(\mathtt{add}(x, y)), \qquad \mathtt{add}(x, x) \bowtie Z$$

The example net in Figure 1 can be represented using the configuration:

$$\langle\, a \mid \mathtt{add}(a, Z) = S(Z) \,\rangle$$

and the following is a possible reduction sequence using the calculus rules above:

$$\begin{aligned}
\langle\, a \mid \mathtt{add}(a, Z) = S(Z) \,\rangle \quad &\longrightarrow_{\bowtie} \quad \langle\, a \mid a = S(x'), Z = y', Z = \mathtt{add}(x', y') \,\rangle \\
&\longrightarrow_c \quad \langle\, S(x') \mid Z = y', Z = \mathtt{add}(x', y') \,\rangle \\
&\longrightarrow_i \quad \langle\, S(x') \mid Z = \mathtt{add}(x', Z) \,\rangle \\
&\longrightarrow_{\bowtie} \quad \langle\, S(x') \mid x' = x'', Z = x'' \,\rangle \\
&\longrightarrow_c \quad \langle\, S(x'') \mid Z = x'' \,\rangle \\
&\longrightarrow_c \quad \langle\, S(Z) \mid \,\rangle
\end{aligned}$$

3 Refining the calculus

The calculus given in the previous section has nice properties and provides a simple static and dynamic semantics for interaction nets. However, the calculus introduces extra computational steps to reduce a given net to normal form. For example, the example net in Figure 1 reduces in two steps using the graphical setting while the same net reduces in six steps using the textual calculus (see Example 1). In this section, we answer the following question in the positive: can we optimise the calculus to obtain more efficient computations? The result of this question is our *lightweight* calculus which will form the basis of the *lightweight* abstract machine.

[1] see [Laf90, HMS08] for a more detailed description of Lafont's style syntax

Interaction rules. The notation of Lafont's style generates (redundant) equations which will be reduced by the Indirection rule. In particular, if an auxiliary port of an interacting agent in a rule is connected to another auxiliary port, the application of an Interaction rule will generate an equation with a variable x on one side of the equation. Since all variables appear twice in a rule, x will eventually be eliminated using the Indirection rule. For example, this can be traced in Example 1 where the equation $\mathtt{Z} = y'$ is generated in the configuration after applying the first rule $\mathtt{add}(\mathtt{S}(x), y) \bowtie \mathtt{S}(\mathtt{add}(x, y))$. In other words, the application of an Interaction rule to an active pair (α, β) where $\alpha(\bar{t_1}, x, \bar{t_2}) \bowtie \beta(\bar{s_1}) \in \mathscr{R}$ will generate a configuration where an Indirection rule is applicable.

In order to eliminate the generation of redundant equations we introduce an alternative notation to represent interaction rules. We represent rules using the syntax: $lhs \longrightarrow rhs$ where lhs consists of an equation between the two interacting agents and rhs is a list of equations which represent the right-hand side net. All rules $\alpha(\bar{t}) \bowtie \beta(\bar{s})$ in Lafont's style can be written using our notation:

$$\alpha(\bar{t_1}) = \beta(\bar{s_1}) \longrightarrow \bar{t_1} = \bar{t}, \bar{s_1} = \bar{s} \qquad \text{where } \bar{t_1}, \bar{s_1} \text{ are meta-variables for terms.}$$

As a concrete example, the rule $\mathtt{add}(\mathtt{S}(x), y) \bowtie \mathtt{S}(\mathtt{add}(x, y))$ can be represented as

$$\mathtt{add}(t_1, t_2) = \mathtt{S}(u_1) \longrightarrow t_1 = \mathtt{S}(x), t_2 = y, u_1 = \mathtt{add}(x, y)$$

moreover we can simplify rules by replacing equals for equals. The above rule can be simplified to:

$$\mathtt{add}(t_1, t_2) = \mathtt{S}(u_1) \longrightarrow t_1 = \mathtt{S}(x), u_1 = \mathtt{add}(x, t_2)$$

Therefore we obtain a more efficient computation by using the notation of term rewriting systems.

Definition 2 (Lightweight interaction rules) A lightweight rule $r \in \mathscr{R}_{\text{lt}}$ is of the form:

$$\alpha(t_1, ..., t_n) = \beta(s_1, ..., s_k) \longrightarrow \Delta$$

where $\alpha, \beta \in \Sigma$, $\text{ar}(\alpha) = n, \text{ar}(\beta) = k$, and $t_1, ..., t_n, s_1, ..., s_k$ are meta-variables for terms. Each meta-variable occurs exactly twice in a rule: once on the lhs and once on the rhs. The set $\mathscr{R}_{\text{lt}}$ contains at most one rule between any pair of agents; $\mathscr{R}_{\text{lt}}$ is closed under symmetry — if $\alpha(\bar{t}) = \beta(\bar{s}) \longrightarrow \Delta \in \mathscr{R}_{\text{lt}}$ then $\beta(\bar{s}) = \alpha(\bar{t}) \longrightarrow \Delta \in \mathscr{R}_{\text{lt}}$.

Indirection rules. Let us now examine the Indirection rule of the calculus which eliminates bound variables by means of variable substitution. The application of this rule will search through the list of terms to locate a term which contains an occurrence of a particular variable. In order to reduce the searching costs, Pinto's abstract machine [Pin00], which is based on this calculus, attaches a list of variables to the head of every term. This again introduces management overheads, hence the increase in the number of operations required to perform rewirings.

Taking into consideration that every change of connection does not affect interactions directly, it turns out that we do not have to perform all substitutions eagerly. Therefore we decompose the Indirection rule into: *communication rules* that will replace just a name, and *substitution rule* that will perform other substitutions.

Definition 3 (Lightweight reduction rules) We define Lightweight reduction rules as follows:

Communication:
$\langle\, \bar{t} \mid x = t, x = u, \Delta \,\rangle \xrightarrow{com} \langle\, \bar{t} \mid t = u, \Delta \,\rangle$,

Substitution:
$\langle\, \bar{t} \mid x = t, u = s, \Delta \,\rangle \xrightarrow{sub} \langle\, \bar{t} \mid u[t/x] = s, \Delta \,\rangle$ where u is not a name and x occurs in u,

Collect:
$\langle\, \bar{t} \mid x = t, \Delta \,\rangle \xrightarrow{col} \langle\, \bar{t}[t/x] \mid \Delta \,\rangle$ where x occurs in $\bar{t}$,

Interaction:
$\langle\, \bar{t} \mid \alpha(\bar{t_1}) = \beta(\bar{t_2}), \Delta \,\rangle \xrightarrow{int} \langle\, \bar{t} \mid \Theta^l, \Delta \,\rangle$

where $\alpha(\bar{s}) = \beta(\bar{u}) \longrightarrow \Theta \in \mathscr{R}_{\mathrm{lt}}$ and Θ^l is the result of replacing each occurrence of a bound name x for Θ by a fresh name x^l and replacing each occurrence of $\bar{s}, \bar{u}$ by $\bar{t_1}, \bar{t_2}$ respectively.

We use just $\longrightarrow$ instead of $\xrightarrow{com}, \xrightarrow{sub}, \xrightarrow{col}, \xrightarrow{int}$ when there is no ambiguity. We define $C_1 \Downarrow C_2$ by $C_1 \longrightarrow^* C_2$ where C_2 is in normal form. From now on, we use $T, S, U, \ldots$ for non-variable terms.

Example 2 *Rules in Figure 1 can be represented as follows:*

$$\begin{array}{rcl} \mathtt{add}(x_1, x_2) = \mathtt{S}(y) & \longrightarrow & x_1 = \mathtt{S}(w), y = \mathtt{add}(w, x_2) \\ \mathtt{add}(x_1, x_2) = \mathtt{Z} & \longrightarrow & x_1 = x_2 \end{array}$$

and the following computation can be performed:

$$\begin{array}{rcl} \langle\, a \mid \mathtt{add}(a, \mathtt{Z}) = \mathtt{S}(\mathtt{Z}) \,\rangle & \xrightarrow{int} & \langle\, a \mid a = \mathtt{S}(w'), \mathtt{Z} = \mathtt{add}(w', \mathtt{Z}) \,\rangle \\ & \xrightarrow{col} & \langle\, \mathtt{S}(w') \mid \mathtt{Z} = \mathtt{add}(w', \mathtt{Z}) \,\rangle \\ & \xrightarrow{int} & \langle\, \mathtt{S}(w') \mid w' = \mathtt{Z} \,\rangle \\ & \xrightarrow{col} & \langle\, \mathtt{S}(\mathtt{Z}) \mid \,\rangle \end{array}$$

3.1 Properties of lightweight reduction rules

In this section, we present some properties of the lightweight reduction rules. First, we show that we can postpone the application of Collect rules as in Abramsky's Computational interpretations of linear logic [Abr93].

Lemma 1 *If* $C_1 \xrightarrow{col} \cdot \xrightarrow{com} C_2$ *then* $C_1 \xrightarrow{com} \cdot \xrightarrow{col} C_2$.

Proof. Let $C_1 = \langle\, \bar{t} \mid x = t, u = y, y = v, \Delta \,\rangle \xrightarrow{col} \langle\, \bar{t}[t/x] \mid u = y, y = v, \Delta \,\rangle \xrightarrow{com} \langle\, \bar{t}[t/x] \mid u = v, \Delta \,\rangle = C_2$. Then, $C_1 \xrightarrow{com} \langle\, \bar{t} \mid x = t, u = v, \Delta \,\rangle \xrightarrow{col} C_2$. □

Lemma 2 *If* $C_1 \xrightarrow{col} \cdot \xrightarrow{sub} C_2$ *then* $C_1 \xrightarrow{sub} \cdot \xrightarrow{col} C_2$. □

Lemma 3 *If* $C_1 \xrightarrow{col} \cdot \xrightarrow{int} C_2$ *then* $C_1 \xrightarrow{int} \cdot \xrightarrow{col} C_2$. □

By Lemma 1, 2, 3, the following holds.

Lemma 4 *If* $C_1 \Downarrow C_2$ *then there is a configuration* C *such that* $C_1 \longrightarrow^* C \xrightarrow{col}{}^* C_2$ *and* C_1 *is reduced to* C *without the application of any* Collect *rule.* □

Next, we examine whether or not we can postpone the application of Substitution rules. Note that applying the Substitution rule to an equation does not generate any other equations which require the application of an Interaction rule. Therefore the following properties hold.

Lemma 5 *If* $C_1 \xrightarrow{sub} \cdot \xrightarrow{com} C_2$ *then* $C_1 \xrightarrow{com} \cdot \xrightarrow{sub} C_2$. □

Lemma 6 *If* $C_1 \xrightarrow{sub} \cdot \xrightarrow{int} C_2$ *then* $C_1 \xrightarrow{int} \cdot \xrightarrow{sub} C_2$ *or* $C_1 \xrightarrow{int} \cdot \xrightarrow{com} C_2$. □

By Lemma 4, 5 and 6 the following theorem holds.

Theorem 1 *If* $C_1 \Downarrow C_2$ *then there is a configuration* C *such that* $C_1 \longrightarrow^* C \xrightarrow{sub}{}^* \cdot \xrightarrow{col}{}^* C_2$ *and* C_1 *is reduced to* C *by applying only* Communication *and* Interaction *rules.* □

This theorem shows that all Interaction rules can be performed without applying Substitution rules. We define $C_1 \Downarrow_{\mathrm{ic}} C_2$ by $C_1 \longrightarrow^* C_2$ where C_2 is a $\{\xrightarrow{int}, \xrightarrow{com}\}-$normal form.

4 Lightweight abstract machine

In this section we define the Lightweight abstract machine which is based on the lightweight rewriting rules.

Definition 4 (Machine configuration) A configuration of our abstract machine state is given by a 5-tuple $(\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid \Delta)$ where

- Γ is an environment which maps a variable to a term. We use $[]$ as an empty map and the following notation:

$$\Gamma[x \mapsto t](z) = \begin{cases} t & (z \text{ is } x) \\ \Gamma(z) & (\text{otherwise}) \end{cases}$$

- ϕ is a *connection map*. When $\phi(x)$ is undefined, we use the following notation:

$$\phi[x \leftrightarrow \bot](z) = \begin{cases} \text{undefined} & (z = x) \\ \phi(z) & (\text{otherwise}) \end{cases}$$

- $\bar{t}$ is a sequence of terms
- Θ is a sequence of error codes that are not executable

Δ is a sequence of equations which we also regard as codes. We write "$-$" for an empty sequence of codes.

In Figure 2 we give the semantics of the machine as a set of transitional rules of the form: $(\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid \Delta) \Longrightarrow (\Gamma' \mid \phi' \mid \bar{t} \mid \Theta' \mid \Delta')$. The functions **interaction**$(S = T)$ and **error**$(S = T)$ are defined as follows:

$$\textbf{interaction}(S=T) = \begin{cases} \Delta_1 & (\text{when } \langle \mid S=T \mid \rangle \xrightarrow{int} \langle \mid \Delta_1 \mid \rangle), \\ - & (\text{otherwise}) \end{cases}$$

$$\textbf{error}(S=T) = \begin{cases} - & (\text{when } \langle \mid S=T \mid \rangle \xrightarrow{int} \langle \mid \Delta_1 \mid \rangle), \\ S=T & (\text{otherwise}) \end{cases}$$

For readability purposes we present the transitions in a table format. For example, the entry:

		Before	After
II.0	Connections	$\phi\ [x \leftrightarrow \bot]$	$\phi\ [x \leftrightarrow \bot]$
	Env.	$\Gamma\ [x \mapsto \bot]$	$\Gamma\ [x \mapsto U]$
	Code	$x = U,\ \Delta$	Δ

corresponds to:

$$(\Gamma\ [x \mapsto \bot] \mid \phi\ [x \leftrightarrow \bot] \mid \bar{t} \mid - \mid x = U,\ \Delta) \Longrightarrow (\Gamma\ [x \mapsto U] \mid \phi\ [x \leftrightarrow \bot] \mid \bar{t} \mid - \mid \Delta)$$

4.1 Correctness

In order to show the correctness of our abstract machine, we first define a decompilation function from configurations to terms. Several lemmas follow before the correctness theorem.

Definition 5 (Decompilation) We define a translation $\lfloor . \rfloor_{env}$ from an environment Γ into a multiset of equations as follows:

$$\begin{aligned} \lfloor [] \rfloor_{env} &\stackrel{\text{def}}{=} empty, \\ \lfloor \Gamma[x \mapsto t] \rfloor_{env} &\stackrel{\text{def}}{=} x = t, \lfloor \Gamma \rfloor_{env}. \end{aligned}$$

The function $\lfloor . \rfloor_{con}$ translates a connection map ϕ into a multiset of equations as follows:

$$\begin{aligned} \lfloor [] \rfloor_{con} &\stackrel{\text{def}}{=} empty, \\ \lfloor \phi[x \leftrightarrow y] \rfloor_{con} &\stackrel{\text{def}}{=} x = y, \lfloor \phi \rfloor_{con}. \end{aligned}$$

We write just $\lfloor . \rfloor$ instead of $\lfloor . \rfloor_{env}$, $\lfloor . \rfloor_{con}$ when there is no ambiguity.

The machine will stop when there is no executable code. These cases arise not only when the code sequence is empty, but also when names are included in both the domains of Γ and ϕ. We define the latter case as inconsistent:

		Before	After
I	Error Code	Θ $U = T,\ \Delta$	$\mathbf{error}(U = T), \Theta$ $\mathbf{interaction}(U = T),\ \Delta$
II.0	Connections Env. Code	$\phi\ [x \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot]$ $x = U,\ \Delta$	$\phi\ [x \leftrightarrow \bot]$ $\Gamma\ [x \mapsto U]$ Δ
II.c	Connections Env. Code	$\phi\ [x \leftrightarrow y]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ $x = U,\ \Delta$	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto U]$ Δ
II.e	Connections Env. Code	$\phi\ [x \leftrightarrow \bot]$ $\Gamma\ [x \mapsto T]$ $x = U,\ \Delta$	$\phi\ [x \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot]$ $T = U,\ \Delta$
II.–	Code	$U = x, \Delta$	$x = U,\ \Delta$
III.0_0	Connections Env. Code	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ $x = y, \Delta$	$\phi [x \leftrightarrow y]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ Δ
III.0_c	Connections Env. Code	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow w]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ $x = y,\ \Delta$	$\phi\ [x \leftrightarrow w][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ Δ
III.0_e	Connections Env. Code	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto U]$ $x = y,\ \Delta$	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto U][y \mapsto \bot]$ Δ
III.c_0	Connections Env. Code	$\phi\ [x \leftrightarrow z][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ $x = y,\ \Delta$	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow z]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ Δ
III.c_c	Connections Env. Code	$\phi\ [x \leftrightarrow z][y \leftrightarrow w]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ $x = y, \Delta$	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot][z \leftrightarrow w]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ Δ
III.c_e	Connections Env. Code	$\phi\ [x \leftrightarrow z][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto U]$ $x = y, \Delta$	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot][z \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot][z \mapsto U]$ Δ
III.e_0	Connections Env. Code	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto T][y \mapsto \bot]$ $x = y,\ \Delta$	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto T]$ Δ
III.e_c	Connections Env. Code	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow w]$ $\Gamma\ [x \mapsto T][y \mapsto \bot]$ $x = y,\ \Delta$	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot][w \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot][w \mapsto T]$ Δ
III.e_e	Connections Env. Code	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto T][y \mapsto U]$ $x = y,\ \Delta$	$\phi\ [x \leftrightarrow \bot][y \leftrightarrow \bot]$ $\Gamma\ [x \mapsto \bot][y \mapsto \bot]$ $T = U,\ \Delta$

Figure 2: Transitions $(\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid \Delta) \Longrightarrow (\Gamma' \mid \phi' \mid \bar{t} \mid \Theta' \mid \Delta')$

Definition 6 (Consistency of a machine state) A state $(\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid \Delta)$ is consistent iff

- $\langle\, \bar{t} \mid \lfloor\Gamma\rfloor, \lfloor\phi\rfloor, \Theta, \Delta \,\rangle$ is a configuration, thus every name occurs at most twice,
- for every $x \in \mathcal{N}$, x is not included in both domains of Γ and ϕ.

The following lemma shows that consistency is preserved during transitions:

Lemma 7 *Let M_1 be a consistent state. If $M_1 \Longrightarrow M_2$, then M_2 is also consistent.* □

Let M_1 and M_2 be two abstract machine states. We define $M_1 \Downarrow M_2$ by $M_1 \Longrightarrow^* M_2$ where M_2 is a $\Longrightarrow$ −normal form.

Lemma 8 *Let M_1 be a consistent state, If $M_1 \Downarrow (\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid \Delta)$, then Δ is empty.*

Proof. There exists a transition which can be applied to an equation $t = s$ whenever $(\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid t = s, \Delta)$ is consistent. □

Lemma 9 *Let M_1 be a consistent state $(\Gamma_1 \mid \phi_1 \mid \bar{t} \mid \Theta_1 \mid \Delta_1)$. If $M_1 \Longrightarrow (\Gamma_2 \mid \phi_2 \mid \bar{t} \mid \Theta_2 \mid \Delta_2)$, then one of the following holds:*

- $\langle\, \bar{t} \mid \lfloor\Gamma_1\rfloor, \lfloor\phi_1\rfloor, \Theta_1, \Delta_1 \,\rangle = \langle\, \bar{t} \mid \lfloor\Gamma_2\rfloor, \lfloor\phi_2\rfloor, \Theta_2, \Delta_2 \,\rangle$,
- $\langle\, \bar{t} \mid \lfloor\Gamma_1\rfloor, \lfloor\phi_1\rfloor, \Theta_1, \Delta_1 \,\rangle \xrightarrow{int} \langle\, \bar{t} \mid \lfloor\Gamma_2\rfloor, \lfloor\phi_2\rfloor, \Theta_2, \Delta_2 \,\rangle$,
- $\langle\, \bar{t} \mid \lfloor\Gamma_1\rfloor, \lfloor\phi_1\rfloor, \Theta_1, \Delta_1 \,\rangle \xrightarrow{com} \langle\, \bar{t} \mid \lfloor\Gamma_2\rfloor, \lfloor\phi_2\rfloor, \Theta_2, \Delta_2 \,\rangle$,
- $\langle\, \bar{t} \mid \lfloor\Gamma_1\rfloor, \lfloor\phi_1\rfloor, \Theta_1, \Delta_1 \,\rangle \xrightarrow{com} \cdot \xrightarrow{com} \langle\, \bar{t} \mid \lfloor\Gamma_2\rfloor, \lfloor\phi_2\rfloor, \Theta_2, \Delta_2 \,\rangle$. □

Theorem 2 *Let $\langle\, \bar{t} \mid \Delta \,\rangle$ be a configuration. If $(\,[\,] \mid [\,] \mid \bar{t} \mid - \mid \Delta)$ terminates at $(\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid \Delta')$, then Δ' is empty and $\langle\, \bar{t} \mid \Delta \,\rangle \Downarrow_{ic} \langle\, \bar{t} \mid \lfloor\Gamma\rfloor, \lfloor\phi\rfloor, \Theta \,\rangle$.*

Proof. By Lemma 8, Δ' is empty. Since $(\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid -)$ is consistent by Lemma 7, $\lfloor\Gamma\rfloor$ and $\lfloor\phi\rfloor$ cannot contain equations that are reducible using the Communication rule. Therefore, by Lemma 9, $\langle\, \bar{t} \mid \Delta \,\rangle \Downarrow_{ic} \langle\, \bar{t} \mid \lfloor\Gamma\rfloor, \lfloor\phi\rfloor, \Theta \,\rangle$. □

Definition 7 We define the operation `update` as follows:

- $\texttt{update}(\Gamma \mid \phi[x \leftrightarrow y] \mid \bar{t} \mid \Theta \mid -) = \texttt{update}(\Gamma[x/y] \mid \phi \mid \bar{t}[x/y] \mid \Theta \mid -)$,
- $\texttt{update}(\Gamma[x \mapsto s] \mid [\,] \mid \bar{t} \mid \Theta \mid -) = \texttt{update}(\Gamma[s/x] \mid [\,] \mid \bar{t}[s/x] \mid \Theta \mid -)$,
- $\texttt{update}([\,] \mid [\,] \mid \bar{t} \mid \Theta \mid -) = \bar{t}$.

Each execution of `update` corresponds to an application of either Substitution or Collect rules. Therefore, we can show the following property:

Theorem 3 (Correctness) *Let $\langle\, \bar{t} \mid \Delta \,\rangle$ be a configuration. If $(\,[\,] \mid [\,] \mid \bar{t} \mid - \mid \Delta) \Downarrow (\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid \Delta')$, then Δ' is empty and there is a reduction path such that $\langle\, \bar{t} \mid \Delta \,\rangle \Downarrow \langle\, \bar{u} \mid \Theta' \,\rangle$ where $\texttt{update}(\Gamma \mid \phi \mid \bar{t} \mid \Theta \mid -) = \bar{u}$.*

	AMINE	Light	AMINE/Light
255II	14.07	0.09	156.33
264II	50.02	0.14	357.29
256II	119.93	0.23	521.43
A 3 6	4.14	0.18	23.00
A 3 7	40.15	0.71	57.04
A 3 8	612.19	1.70	360.11

Table 1: The execution times in seconds on Linux PC (2.6GHz, Pentium 4, 512MByte)

Example 3 *The computation of* $\langle\, r \mid \mathtt{Add}(r, \mathtt{Z}) = \mathtt{S}(\mathtt{Z}) \,\rangle$ *is given below:*

$$
\begin{aligned}
&(\,[]\mid[]\mid r\mid -\mid \mathtt{Add}(r,\mathtt{Z}) = \mathtt{S}(\mathtt{Z})\,)\\
&\quad\Longrightarrow (\,[]\mid[]\mid r\mid -\mid r = \mathtt{S}(x), \mathtt{Z} = \mathtt{Add}(x,\mathtt{Z})\,) && \text{(I)}\\
&\quad\Longrightarrow (\,[r\mapsto \mathtt{S}(x)]\mid[]\mid r\mid -\mid \mathtt{Z} = \mathtt{Add}(x,\mathtt{Z})\,) && \text{(II.0)}\\
&\quad\Longrightarrow (\,[r\mapsto \mathtt{S}(x)]\mid[]\mid r\mid -\mid x = \mathtt{Z}\,) && \text{(I)}\\
&\quad\Longrightarrow (\,[r\mapsto \mathtt{S}(x)][x\mapsto \mathtt{Z}]\mid[]\mid r\mid -\mid -\,) && \text{(II.0).}
\end{aligned}
$$

$$
\begin{aligned}
&\mathtt{update}(\,[r\mapsto \mathtt{S}(x)][x\mapsto \mathtt{Z}]\mid[]\mid r\mid -\mid -\,)\\
&\quad = \mathtt{update}(\,[r\mapsto \mathtt{S}(\mathtt{Z})]\mid[]\mid r\mid -\mid -\,) = \mathtt{S}(\mathtt{Z}).
\end{aligned}
$$

4.2 Benchmark results

We compare the lightweight version with Pinto's implementation (AMINE). Both are written in C language. Table 1 shows execution times in seconds of our implementation and AMINE. The final column gives the ratio between the two. The first three input programs are applications of church numerals where $n = \lambda f.\lambda x.f^n x$ and $\mathtt{I} = \lambda x.x$. The encodings of these terms into interaction nets are given in [Mac98]. The next programs compute the Ackermann function. The following rules are the interaction net encoding of the Ackermann function:

$$
\begin{array}{ll}
\mathtt{Pred}(\mathtt{Z}) \bowtie \mathtt{Z}, & \mathtt{Dup}(\mathtt{Z},\mathtt{Z}) \bowtie \mathtt{Z},\\
\mathtt{Pred}(x) \bowtie \mathtt{S}(x), & \mathtt{Dup}(\mathtt{S}(a),\mathtt{S}(b)) \bowtie \mathtt{S}(\mathtt{Dup}(a,b)),\\
\mathtt{A}(r,\mathtt{S}(r)) \bowtie \mathtt{Z}, & \mathtt{A1}(\mathtt{Pred}(\mathtt{A}(\mathtt{S}(\mathtt{Z}),r)),r) \bowtie \mathtt{Z},\\
\mathtt{A}(\mathtt{A1}(\mathtt{S}(x),r),r) \bowtie \mathtt{S}(x), & \mathtt{A1}(\mathtt{Dup}(\mathtt{Pred}(\mathtt{A}(r1,r)),\mathtt{A}(y,r1)),r) \bowtie \mathtt{S}(y),
\end{array}
$$

and A 3 6 means computation of $\langle\, r \mid \mathtt{A}(\mathtt{S}(\mathtt{S}(\mathtt{S}(\mathtt{S}(\mathtt{S}(\mathtt{S}(\mathtt{Z})))))),r) = \mathtt{S}(\mathtt{S}(\mathtt{S}(\mathtt{Z})))\,\rangle$.

The results that we have obtained are better than previous implementation results, and allow substantially larger classes of functions to be executed very efficiently. Depending on the architecture used, these results will vary slightly. We however invite the reader to try some of these examples by downloading our implementation: http://www.interaction-nets.org/.

5 Conclusion

The aim of this paper is to report on current work on the foundations of the implementations of interaction nets. Specifically, we have presented a new implementation that is the most efficient to date. In the work where interaction nets are considered as an intermediate language for compilation, this work gives a speedup by a factor of ten or more.

Implementation work for interaction nets is currently being investigated very actively, and although this step is a considerable one, we believe that there is still much more to do. Our implementations are still very much prototype in nature, and no program optimisations have been included here. Future work will be directed towards developing stable and efficient implementations for both sequential and parallel architectures.

Bibliography

[Abr93] S. Abramsky. Computational Interpretations of Linear Logic. *Theoretical Computer Science* 111:3–57, 1993.

[AG98] A. Asperti, S. Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge Tracts in Theoretical Computer Science 45. Cambridge University Press, 1998.

[FM99] M. Fernández, I. Mackie. A Calculus for Interaction Nets. In Nadathur (ed.), *Proceedings of the International Conference on Principles and Practice of Declarative Programming (PPDP'99)*. LNCS 1702, pp. 170–187. Springer-Verlag, 1999. ftp://lix.polytechnique.fr/pub/mackie/papers/calin.ps.gz

[GAL92] G. Gonthier, M. Abadi, J.-J. Lévy. The Geometry of Optimal Lambda Reduction. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages (POPL'92)*. Pp. 15–26. ACM Press, Jan. 1992.

[HMS08] A. Hassan, I. Mackie, S. Sato. Interaction nets: programming language design and implementation. *ECEASST* 10, 2008.

[HMS09] A. Hassan, I. Mackie, S. Sato. Compilation of Interaction Nets. *Electron. Notes Theor. Comput. Sci.* 253(4):73–90, 2009. doi:http://dx.doi.org/10.1016/j.entcs.2009.10.018

[Laf90] Y. Lafont. Interaction Nets. In *Seventeenth Annual Symposium on Principles of Programming Languages*. Pp. 95–108. ACM Press, San Francisco, California, 1990.

[Mac98] 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)*. Pp. 117–128. ACM Press, September 1998. ftp://lix.polytechnique.fr/pub/mackie/papers/yalyal.ps.gz

[Mac05] I. Mackie. Towards a Programming Language for Interaction Nets. *Electronic Notes in Theoretical Computer Science* 127(5):133–151, May 2005.

[Pin00] J. S. Pinto. Sequential and Concurrent Abstract Machines for Interaction Nets. In Tiuryn (ed.), *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*. Lecture Notes in Computer Science 1784, pp. 267–282. Springer-Verlag, 2000.