Electronic Notes in Theoretical Computer Science 68 No. 2 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume68.html 18 pages

Interaction Nets with McCarthy's amb

Maribel Fernández and Lionel Khalil [1]

LIENS (CNRS UMR 8548),
École Normale Supérieure
45 Rue d'Ulm, 75005 Paris, France

Abstract

Interaction nets are graphical rewrite systems which have been successfully used to implement various efficient evaluation strategies in the λ-calculus (including optimal reduction). However, they are intrinsically deterministic and this prevents from applying these techniques to concurrent languages where non-determinism plays a key rôle. In this paper we show that a minimal extension — the addition of one agent in the spirit of McCarthy's amb operator — allows us to define non-deterministic processes such as angelic and infinity merge, and more generally, to encode process calculi and wide classes of term rewriting systems (including systems defining parallel functions). We also show that Alexiev's INMPP (interaction nets with multiple principal ports) can be encoded, for which we give a textual calculus and a type system that ensures the absence of deadlock.

1 Introduction

Interaction nets [9] are a graphical model of computation derived from the multiplicative proof nets of linear logic. An interaction net program consists of a graph with agents at the nodes, and a set of graph rewriting rules which specify the interaction between two agents connected through their principal ports (each agent has a unique principal port, and there is a unique rule for each pair of agents). Interaction nets have been used to implement the optimal evaluator for the λ-calculus [7,3], on which the programming language BOHM [2] is based. They enjoy nice theoretical and pragmatic properties, such as strong confluence and locality of rewriting. However, they are not suitable for modelling parallel functions and non-deterministic systems, such as process calculi or term rewriting systems.

In the past, several extensions to interaction nets have been proposed with the aim of implementing non-deterministic features of programming languages [1,5]. These extensions are roughly of two kinds: either they use the

[1] Email: maribel@di.ens.fr, khalil@di.ens.fr

same nets but break the confluence property by relaxing some conditions in the definition of interaction rule (for instance, in [5] a state is added, and agents are allowed to interact through the state), or they extend the formalism by adding specific agents with rewriting mechanisms which are not allowed in conventional interaction nets (for instance, in the language INMPP [1] agents are allowed to have more than one principal port).

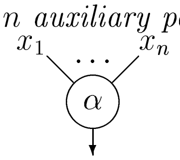
In this paper we consider the latter alternative, and define a *minimal* extension of interaction nets, called INAMB, which allows us to encode non-determinism but remains as close to conventional interaction nets as possible. The extension consists of adding just one agent, amb, with two interaction rules (similar to McCarthy's amb operator [11]). To demonstrate the expressive power of the extended system, we first show that the non-deterministic angelic and infinity merge processes can be defined in INAMB, but fair merge cannot. We then show that INAMB is actually powerful enough to encode the whole language INMPP (nets with multiple principal ports), which Alexiev used to encode a process calculus. Finally we show that INAMB is a parallel model of computation which can implement all functions defined by constructor term rewriting systems (including non-sequential functions, that cannot be implemented in conventional interaction nets). This makes this formalism a good candidate for the implementation of concurrent programming languages based on term rewriting.

We present the extension first in an intuitive graphical way, and then give a textual calculus for INMPP (which applies also to INAMB as a particular case) with a formal operational semantics, and a type system that ensures the absence of deadlock.

Overview. In the next section we recall some basic preliminaries on interaction nets. Section 3 gives examples of the kind of system we would like to model, and defines the extension of interaction nets with the agent amb and the encoding of the parallel merge primitives. In Section 4 we formalize the system, giving the operational semantics and a type system that ensures deadlock-freeness. We use this calculus to prove the equivalence between INAMB and INMPP. Section 5 describes the encoding of term rewriting systems in INAMB. We conclude the paper in Section 6.

2 Interaction Nets

An interaction net system is specified by a set Σ of *agents*, and a set IR of *interaction rules*. Each $\alpha \in \Sigma$ has an associated (fixed) *arity*. If the arity of α is n, then the agent has $n+1$ *ports*: a distinguished one called the *principal port* depicted by an arrow, and n *auxiliary ports*:



A *net* N built on Σ is a graph (not necessarily connected) 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 (edges may connect two ports of the same agent). A net may also have edges with free extremes, called *wires*, and their extremes are called ports by analogy. The *interface* of a net is its set of free ports.

A pair of agents $(\alpha, \beta) \in \Sigma^2$ connected together on their principal ports is called an *active pair*; the interaction net analogue of a redex. An *interaction rule* $((\alpha, \beta) \Longrightarrow N) \in \mathsf{IR}$ replaces an occurrence of the active pair (α, β) by the net N. Rules have to satisfy two very strong conditions: the interface of the active pair must be equal to the interface of the right-hand side, and at most one rule can be defined for each active pair. These conditions imply that interactions are always binary, local, and strongly confluent. For this reason, interactions can take place in any order in a net, even in parallel. We refer to [9] for a detailed presentation and examples of interaction nets.

We recall the interaction calculus developed in [6], which provides a textual notation for nets and rules, as well as a formal account of the rewriting process. Let Σ be a set of agents and $\mathcal{N}$ a set of *names* (or variables) disjoint with Σ. *Terms* are defined by the grammar

$$t ::= x \mid \alpha(t_1, \ldots, t_n)$$

where $x \in \mathcal{N}, \alpha \in \Sigma$, n is the arity of α and $t_1, \ldots, t_n$ are terms with the restriction that each name can at most appear twice (*linearity constraint*). $\mathcal{N}(t)$ denotes the set of names occurring in t, and we write $(t_1, \ldots, t_n)$ as $\vec{t}$. An *equation* $\alpha(\vec{t}) = \beta(\vec{u})$ indicates a connection between the principal ports of the agents α and β. We denote a list $t_1 = u_1, \ldots, t_n = u_n$ of equations by $\overline{t = u}$. To represent nets we use *configurations*, which are pairs $\langle \vec{t} \mid \Delta \rangle$ where $\vec{t}$ is the *interface* and Δ is a multiset of equations describing the connections between the agents in the net. Configurations satisfy the linearity constraint (each variable occurs at most twice).

An *interaction rule* between α and β is written $\alpha(\vec{t}) \bowtie \beta(\vec{u})$, where $\vec{t}$ and $\vec{u}$ represent the net in the right-hand side of the graphical rule (intuitively, since the rule preserves the interface, it is sufficient to indicate the subnets to be connected to each port in the interface of the active pair, see [6] for details).

Let $\mathcal{R}$ be a set of rules, the rewriting process is defined by four computation rules that apply to configurations:

Interaction: $\alpha(\vec{s'}) \bowtie \beta(\vec{u'}) \in \mathcal{R} \;\Rightarrow\; \langle \vec{t} | \alpha(\vec{s}) = \beta(\vec{u}), \Gamma \rangle \to \langle \vec{t} | \overline{s = s'}, \overline{u = u'}, \Gamma \rangle$

Indirection: $x \in \mathcal{N}(u) \quad \Rightarrow \quad \langle \vec{t} \mid x = t, u = v, \Gamma \rangle \to \langle \vec{t} \mid u[t/x] = v, \Gamma \rangle$

Collect: $x \in \mathcal{N}(\vec{t}) \quad \Rightarrow \quad \langle \vec{t} \mid x = u, \Gamma \rangle \to \langle \vec{t}[u/x] \mid \Gamma \rangle$

Multiset: $\Theta \rightleftharpoons^* \Theta', \langle \vec{t_1} \mid \Theta' \rangle \to \langle \vec{t_2} \mid \Delta' \rangle, \Delta' \rightleftharpoons^* \Delta \quad \Rightarrow \quad \langle \vec{t_1} \mid \Theta \rangle \to \langle \vec{t_2} \mid \Delta \rangle$

where $\rightleftharpoons$ is an equivalence that states the irrelevance of the order of equations in the multiset, as well as the order of the members in an equation.

Two configurations which are the same up to renaming of variables are called α-convertible, and in the first rule above we always use α-conversion to get a copy of the interaction rule with all variables fresh.

Example 2.1 *Addition of natural numbers.* We use agents Z, S, Add with $arity(Z) = 0$, $arity(S) = 1$, $arity(\mathsf{Add}) = 2$, and two interaction rules:

$$\mathsf{Add}(S(x), y) \bowtie S(\mathsf{Add}(x, y)) \qquad \mathsf{Add}(x, x) \bowtie Z$$

The addition $1 + 0$ is represented by the configuration $\langle a \mid \mathsf{Add}(a, Z) = S(Z)\rangle$, which rewrites to the configuration representing 1 as expected:

$$\begin{aligned} &\langle a \mid \mathsf{Add}(a, Z) = S(Z)\rangle \quad \longrightarrow \langle a \mid a = S(x'), y' = Z, Z = \mathsf{Add}(x', y')\rangle \longrightarrow^* \\ &\langle S(x') \mid Z = \mathsf{Add}(x', Z)\rangle \longrightarrow \langle S(x') \mid x'' = x', x'' = Z\rangle \longrightarrow^* \langle S(Z) \mid \rangle \end{aligned}$$

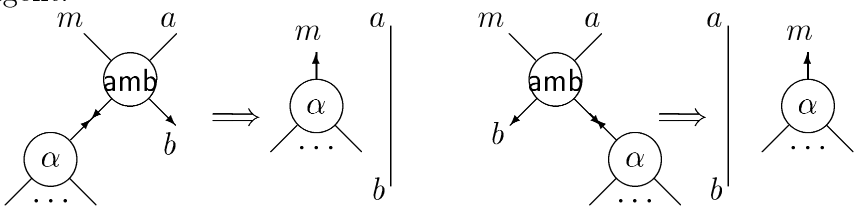
3 Adding a non-deterministic agent

Interaction nets have been used as a tool to model and implement functional programming languages [3,10]. They provide efficient evaluation strategies, and are well suited for parallel implementations since the order of the interactions does not matter [14]. However, although interactions can occur in parallel, the constraints in the definition of rules make them unsuitable for the implementation of concurrent languages. More precisely, in the interaction net framework it is not possible to define non-deterministic processes, or non-sequential functions (see for instance [4] for a definition of sequential function). A well-known example of such a function is parallel-or, defined by the rewrite system:

$$\begin{aligned} \mathsf{por}(\mathsf{True}, x) &\to \mathsf{True} \\ \mathsf{por}(x, \mathsf{True}) &\to \mathsf{True} \\ \mathsf{por}(\mathsf{False}, \mathsf{False}) &\to \mathsf{False} \end{aligned}$$

As an example of a non-deterministic process, we consider a parallel merge: it can be specified in three ways, called *angelic merge*, *infinity merge*, and *fair merge* [13]. All the merge primitives have a pair of input sequences and one output sequence. The elements of the input sequences appear unaltered in the output sequence, and their relative order in the input sequence is preserved (but elements from different input sequences can appear in any order in the output). The difference between these primitives is that, in a fair merge, every element of an input sequence will eventually appear in the output, whereas for an angelic merge all that is guaranteed is that the output sequence is infinite if at least one of the input sequences is infinite. The infinity merge has the dual property: it guarantees that if one of the input sequences is infinite then all the elements of the other one will appear in the output. It is well-known that angelic merge can be implemented using fair merge, and that infinity merge can be implemented with angelic merge. Moreover, these three levels of expressivity are fundamentally different: fair merge cannot be implemented by angelic merge, which in turn cannot be implemented by infinity merge [12].

Our aim is to increase the expressive power of the interaction net framework, but remaining as close as possible to the original definition. A first idea would be to allow two rules with the same left member. In this way we obtain a demonic non-determinism. Instead, we will define a minimal extension of interaction nets which consists of adding one agent with two principal ports (used as inputs) and two auxiliary ports. The agent, which we call amb inspired by McCarthy's work [11], is defined by rules as shown below, where α is any agent.



When an agent α has its principal port connected to a principal port of amb, an interaction can take place, and the agent α arrives at the main output port of amb, which we called m in the diagram above. If in a net there are agents with principal ports connected to both principal ports of amb, the choice of the interaction rule to be applied is non-deterministic.

To illustrate the expressive power gained by this extension we give the definition of angelic and infinity merge.

Example 3.1 *Parallel Merge.* We implement angelic merge using amb, an agent AM of arity 2, and unary agents α representing the elements of the input sequences L_1 and L_2, as shown in Fig. 1 (the angelic merge process is represented by the net at the left and the corresponding reduction rules for each $\alpha \in L_1, L_2$ are given at the right).

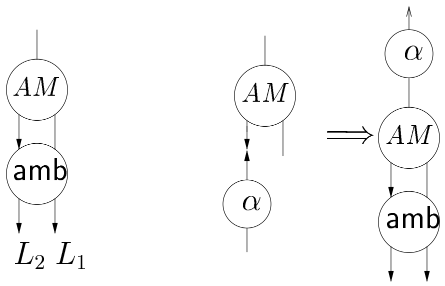


Fig. 1. Angelic Merge

Stark [15] gives an implementation of infinity merge using an oracle, which is a process that generates an infinite sequence of arbitrary numbers. This oracle is used by a process that repeats forever:

- read a value n from the oracle and output n values from L_1;
- read another value n' from the oracle and output n' values from L_2.

If at any point we have less than n tokens left in the input sequence then the process is blocked. But we can guarantee that if one input sequence is infinite then all the tokens in the other one will eventually be in the output (which is the specification of infinity merge). Therefore, to show that infinity merge can be defined in interaction nets with amb it is sufficient to show the implementation of a random number generator, which can then be used to build the oracle. Fig. 2 shows a random number generator: the net at the left generates an arbitrary number in the output port x using the interaction rules given at the right, where ϵ and δ (the standard eraser and duplicator agents) are used to preserve the interface. Notice that there are active pairs in the right hand sides of these rules.

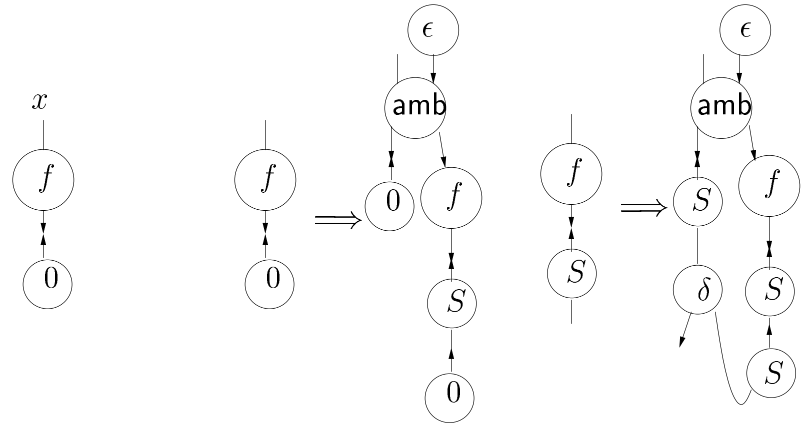


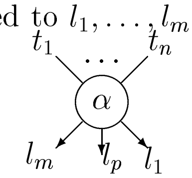
Fig. 2. Random Number Generator

We remark that the fair merge primitive cannot be implemented in interaction nets with amb: this is a consequence of the results of [12]. Actually, the addition of amb turns out to give a computational model which is as powerful as Alexiev's INMPP (interaction nets with multiple principal ports). To show this, in the next section we formalize the operational semantics of the system.

4 A Calculus for INMPP

In order to give a formal operational semantics to the extension of interaction nets with amb, which we call INAMB hereafter, we will introduce a textual interaction calculus which is an extension of the one defined in [6] (see Section 2). INAMB is clearly a particular case of INMPP, and to facilitate the comparison in the other direction, we give the textual calculus for the whole of INMPP. The main feature of INMPP is that agents can have any finite number of principal ports, but interaction rules still specify binary interactions and preserve the interface of the active pair.

Intuitively, an agent α with n auxiliary ports connected to nets $t_1, \ldots, t_n$ and m principal ports connected to $l_1, \ldots, l_m$, depicted:



will be represented by a generalized term $(l_1, \ldots, l_m)\alpha(t_1, \ldots, t_n)$. If the pth principal port is ready to interact we will write an equation of the form $(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(t_1, \ldots, t_n) = l_p$.

Definition 4.1 *Terms and Equations for INMPP.* Let Σ be a set of agents and $\mathcal{N}$ a set of names disjoint with Σ. *Terms* are defined by the grammar:

$$t ::= x \mid (l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(t_1, \ldots, t_n)$$

where $x \in \mathcal{N}$, $\alpha \in \Sigma$, $arity(\alpha) = n$, m is the number of principal ports of α, $t_1, \ldots, t_n$, $l_1, \ldots, l_{p-1}, l_{p+1}, \ldots, l_m$ are terms, and $-$ indicates the selected principal port. Each variable occurs at most twice in a term: variables that occur once represent free ports and are called *free variables* and variables that occur twice represent links. If α has a single principal port we simply write $\alpha(\vec{t})$ instead of $(-)\alpha(\vec{t})$. The *root* of x is x, and the root of $(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(t_1, \ldots, t_n)$ is the agent α.

Equations are defined by the grammar:

$$eq ::= (l_1, \ldots, l_m)\alpha(k_1, \ldots, k_n) \mid t = u$$

where $\alpha \in \Sigma$, $m > 0$ is the number of principal ports of α, $arity(\alpha) = n$, and $l_1, \ldots, l_m, k_1, \ldots, k_n, t, u$ are terms.

Equations of the form $t = u$ are *explicit*, they indicate either a renaming (if t or u is a variable), or an active pair (a connection between two principal ports) that is ready to be reduced. Equations of the form $(\vec{l})\alpha(\vec{k})$, which we also call *multiequations*, indicate potential interactions between α and the agents at the root of the terms in $\vec{l}$.

Definition 4.2 *Interaction rules in INMPP.* An *interaction rule* between the pth principal port of α and the qth principal port of β is written

$$(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{t}) \bowtie (k_1, \ldots, k_{q-1}, -, k_{q+1}, \ldots, k_n)\beta(\vec{u}), \vec{eq}$$

where $\vec{t}, \vec{u}, \vec{l}, \vec{k}, \vec{eq}$ represent the right-hand side of the graphical rule.

Note that the definition of interaction rule differs from the one given for conventional interaction nets (see Section 2) in that it not only indicates the subnets of the right-hand side to be connected to each port in the interface of the left-hand side (terms $\vec{t}, \vec{u}, \vec{l}, \vec{k}$), but it also allows extra equations $\vec{eq}$. These equations are used to represent the active pairs in the right-hand side of the graphical rule. Although for conventional interaction nets it is sensible to assume that there are no active pairs in the right hand side of rules [9], to model non-deterministic primitives involving potentially infinite data structures it is crucial to allow active pairs in right-hand sides (see Fig. 2).

We represent nets by configurations, for which we first define normal terms (i.e. terms without implicit active pairs) to be used in the interface.

Definition 4.3 *Normal Terms.* A term t is *normal* if it is a variable or it has the form $(x_1, \ldots, x_{p-1}, -, x_{p+1}, \ldots, x_m)\alpha(u_1, \ldots, u_n)$ where

- $\vec{x} \in \mathcal{N}$ are the *principal variables* of t (i.e. names of principal ports of α),

- $\vec{u}$ are normal terms, and
- if x is a principal variable of t or of any subterm of t, then x occurs at most once in t.

Note that since any principal variable occurs at most once in t, there are no connections between principal ports in the net represented by t.

Definition 4.4 *Configurations.* A configuration is a pair $\langle \vec{t} | \Delta \rangle$ where Δ is a multiset of equations, $\vec{t}$ is a list of normal terms that do not share principal variables, and no principal variable occurring in $\vec{t}$ occurs in Δ. Each variable occurs at most twice in a configuration; variables occurring once are *free.*

The conditions on $\vec{t}$ guarantee that there are no active pairs in the interface of the configuration. The definition given in Section 2 for conventional interaction nets is a particular case, since conventional terms have no principal variable (every term is normal).

Before giving an example, we present the computation rules that define the dynamics of the system. These are a generalization of the computation rules for the interaction calculus in Section 2. There are three kinds of Indirection rules. The first is the standard rule. The second takes into account the fact that $(l_1, \ldots, l_{p-1}, x, l_{p+1}, \ldots, l_m)\alpha(\vec{s})$ represents the same net as $x = (l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{s})$ since intuitively in both cases we are saying that x is the name of the pth principal port of α. The third Indirection rule transforms a multiequation into an explicit equation (where an active pair is represented explicitly); interaction then applies in the usual way. Choosing which equation is made explicit corresponds to choosing which principal port is used for interaction. We also need two Collect rules, since we have two kinds of equations. Intuitively, the Collect rules move to the interface of the configuration the subnets where the computation has already finished; we need some conditions on these rules to ensure that the result is a well-formed configuration.

Definition 4.5 The Multiset rule does not change (see Section 2), only the Interaction, Indirection and Collect rules are generalised:

Indirection (i) $x \in \mathcal{N}(eq) \quad \Rightarrow \quad \langle \vec{t} \,|\, x = u, eq, \Gamma \rangle \rightarrow \langle \vec{t} \,|\, eq[u/x], \Gamma \rangle$

Indirection (ii) $x \in \mathcal{N}(eq) \quad \Rightarrow \quad \langle \vec{t} | (l_1, \ldots, l_{p-1}, x, l_{p+1}, \ldots, l_m)\alpha(\vec{s}), eq, \Gamma \rangle \rightarrow$
$\langle \vec{t} \,|\, eq[(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{s})/x], \Gamma \rangle$

Indirection (iii)

$$\langle \vec{t} \,|\, (l_1, \ldots, l_{p-1}, (k_1, \ldots, k_{q-1}, -, k_{q+1}, \ldots, k_n)\beta(\vec{u}), l_{p+1}, \ldots, l_m)\alpha(\vec{s}), \Gamma \rangle \rightarrow$$
$$\langle \vec{t} \,|\, (l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{s}) = (k_1, \ldots, k_{q-1}, -, k_{q+1}, \ldots, k_n)\beta(\vec{u}), \Gamma \rangle$$

Interaction

$$(l'_1, \ldots, l'_{p-1}, -, l'_{p+1}, \ldots, l'_m)\alpha(\vec{s'}) \bowtie (k'_1, \ldots, k'_{q-1}, -, k'_{q+1}, \ldots, k'_n)\beta(\vec{u'}), \vec{eq} \in \mathcal{R}$$
$$\Rightarrow$$
$$\langle \vec{t} \,|\, (l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{s}) = (k_1, \ldots, k_{q-1}, -, k_{q+1}, \ldots, k_n)\beta(\vec{u}), \Gamma \rangle \rightarrow$$
$$\langle \vec{t} \,|\, \overline{s = s'}, \overline{u = u'}, l_i = l'_i \ (1 \leq i \leq m, i \neq p), k_j = k'_j \ (1 \leq j \leq n, j \neq q), \vec{eq}, \Gamma \rangle$$

Collect (i)

$x \in \mathcal{N}(\vec{t})$, u normal, no principal variable in u occurs in $\Gamma \Rightarrow$
$$\langle \vec{t} \mid x = u, \Gamma\rangle \to \langle \vec{t}[u/x] \mid \Gamma\rangle$$

Collect (ii)

$x \in \mathcal{N}(\vec{t})$, $(l_1, \ldots, l_{p-1}, x, l_{p+1}, \ldots, l_m)\alpha(\vec{s})$ normal and no principal variable occurs in $\Gamma \Rightarrow \langle \vec{t} \mid (l_1, \ldots, l_{p-1}, x, l_{p+1}, \ldots, l_m)\alpha(\vec{s}), \Gamma\rangle \to$
$$\langle \vec{t}[(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{s})/x] \mid \Gamma\rangle$$

In the Interaction rule we use α-conversion to get a fresh copy of the rule in $\mathcal{R}$, as usual. Note that the Indirection rules do not change the underlying net, it is the Interaction rule that performs the actual computation.

It is easy to translate a diagram into the textual notation (but the translation is not unique): briefly, it suffices to give a name to each port, put in the interface of the configuration the names of the free ports of the net, and write an equation of the form $(x_1, \ldots, x_n)\alpha(y_1, \ldots, y_m)$ for each agent. We can then simplify the configuration using the Indirection rules. For the reverse translation, we use the Indirection rules in both directions to expand all the equations into terms such as $(x_1, \ldots, x_n)\alpha(y_1, \ldots, y_m)$, draw the agents, and draw the edges corresponding to two occurrences of the same name.

The Indirection and Interaction rules allow us to simulate the graphical reduction: when an interaction can take place in a net (i.e. there is a connection between principal ports), the corresponding configuration can be reduced using the Interaction rule, modulo Indirection.

Example 4.6 We show the interaction rules for the agents amb and α:
$$(-, l)\mathsf{amb}(\alpha(\vec{x}), l) \bowtie \alpha(\vec{x}) \qquad (l, -)\mathsf{amb}(\alpha(\vec{x}), l) \bowtie \alpha(\vec{x})$$
and the interaction rule for angelic merge given in Fig. 1 (right):

$$AM(\alpha(x), z) \bowtie \alpha(z'), (z, z')\mathsf{amb}(AM(x, y), y)$$

The following configuration c represents the angelic merge process shown in Fig. 1 (left), where the input sequences contain agents α_i and β_i respectively:

$$\langle x \mid (\beta_1(\beta_2(z')), \alpha_1(\alpha_2(\alpha_3(z))))\mathsf{amb}(AM(x, y), y)\rangle$$

Using the computation rules we can reduce it as follows:

$$\begin{aligned}
c \to_{\textbf{Indirection}}\ & \langle x \mid (\beta_1(\beta_2(z')), -)\mathsf{amb}(AM(x, y), y) = \alpha_1(\alpha_2(\alpha_3(z)))\rangle \\
\to_{\textbf{Interaction}}\ & \langle x \mid l = \beta_1(\beta_2(z')), x' = \alpha_2(\alpha_3(z)), \alpha_1(x') = AM(x, y), l = y\rangle \\
\to_{\textbf{Interaction}}\ & \langle x \mid l = \beta_1(\beta_2(z')), x' = \alpha_2(\alpha_3(z)), x' = z_1, x = \alpha_1(x_1), y = z_0, \\
& \quad (z_0, z_1)\mathsf{amb}(AM(x_1, y_1), y_1), l = y\rangle \\
\to^{*}_{\textbf{Indirection}}\ & \langle x \mid x = \alpha_1(x_1), (\beta_1(\beta_2(z')), \alpha_2(\alpha_3(z)))\mathsf{amb}(AM(x_1, y_1), y_1)\rangle \\
\to_{\textbf{Collect}}\ & \langle \alpha_1(x_1) \mid (\beta_1(\beta_2(z')), \alpha_2(\alpha_3(z)))\mathsf{amb}(AM(x_1, y_1), y_1)\rangle
\end{aligned}$$

The reduction sequence continues until both input sequences are empty.

4.1 A Type System for INMPP

Inspired by [9,6,8], we will develop a polymorphic type system for INMPP. We will use the type system to ensure the absence of deadlock (a deadlock is a cycle of principal ports).

Definition 4.7 We consider a user-defined set of types, built out of a set of type variables $(\varphi, \varphi', \ldots)$ and a set of type constructors (such as **nat**, **bool**, **list**, ...). The type of a port is written σ^s with two components: σ is the type of the information and s is the direction of the information passing through the port ($-/+$ for input/ output), modulo the equivalences:
$(\sigma^-)^- = \sigma^+$, $(\sigma^-)^+ = \sigma^-$, $(\sigma^+)^- = \sigma^-$, $(\sigma^+)^+ = \sigma^+$, $(\sigma^s)^- = (\sigma^-)^s = \sigma^{-s}$, $(\sigma^s)^+ = (\sigma^+)^s = \sigma^{+s} = \sigma^s$.

To type equations we use the symbol $\diamond$ and to assign types to nets we use one-sided sequents and the following inference rules:

Definition 4.8 For each agent α with p principal ports there is a user-defined *Graft* rule:

$$\frac{\Gamma_1, t_1{:}\,\sigma_1^{s_1}, \ldots, t_{i_1}{:}\,\sigma_{i_1}^{s_{i_1}} \quad \ldots \quad \Gamma_k, t_k{:}\,\sigma_k^{s_k}, \ldots, t_{i_k}{:}\,\sigma_{i_k}^{s_{i_k}}}{\Gamma_1, \ldots, \Gamma_k, \alpha(t_1, \ldots, t_n){:}\,(\tau_1^{s_1}, \ldots, \tau_p^{s_p})}\,(\mathit{Graft}\ \alpha)$$

which indicates that the types of the p principal ports of α are $\tau_1^{s_1}, \ldots, \tau_p^{s_p}$, and specifies how the auxiliary ports are typed. Although when $p > 1$ $\alpha(t_1, \ldots, t_n)$ is not a term according to Definition 4.1, this notation allows us to give the types of all the principal ports at the same time.

$$\frac{\Gamma, t{:}\,\sigma, u{:}\,\tau, \Delta}{\Gamma, u{:}\,\tau, t{:}\,\sigma, \Delta}\,(\mathit{Exchange}) \qquad \frac{\Gamma \quad \Delta}{\Gamma, \Delta}\,(\mathit{Mix})$$

To type edges, which are represented by variables and equations, we use the rules:

$$\frac{}{x{:}\,\sigma^s, x{:}\,\sigma^{-s}}\,(\mathit{Axiom}) \qquad \frac{\Gamma, t{:}\,\sigma^s \quad \Delta, u{:}\,\sigma^{-s}}{\Gamma, \Delta, t = u{:}\,\diamond}\,(\mathit{Cut})$$

$$\frac{\forall 1 \le i \le m, l_i{:}\,\sigma_i^{-s_i}, \Gamma_i \quad \alpha(\vec{t}){:}\,(\sigma_1^{s_1}, \ldots, \sigma_m^{s_m}), \Gamma}{\Gamma_1, \ldots, \Gamma_m, \Gamma, (l_1, \ldots, l_m)\alpha(\vec{t}){:}\,\diamond}\,(\mathit{MultiCut})$$

$$\frac{\forall 1 \le i \le m, i \neq j, l_i{:}\,\sigma_i^{-s_i}, \Gamma_i \quad \alpha(\vec{t}){:}\,(\sigma_1^{s_1}, \ldots, \sigma_m^{s_m}), \Gamma}{\Gamma_1, \ldots, \Gamma_m, \Gamma, (l_1, \ldots, l_{j-1}, -, l_{j+1}, \ldots, l_m)\alpha(\vec{t}){:}\,\sigma_j^{s_j}}\,(\mathit{Select})$$

Definition 4.9 *Typeable Configurations.* Let $\{x_1, \ldots, x_m\}$ be the set of free names of t, then t is a term of type σ^s if there exist types $\tau_1, \ldots, \tau_m$ such that $x_1{:}\,\tau_1, \ldots, x_m{:}\,\tau_m, t{:}\,\sigma^s$ is derivable with the rules above.

Equations are typed in a similar way: an equation eq (which is either of the form $t = u$ or $(l_1, \ldots, l_p)\alpha(t_1, \ldots, t_n)$) with free names $\{x_1, \ldots, x_m\}$ is typeable if $x_1{:}\, \tau_1 \ldots, x_m{:}\, \tau_m, eq{:}\, \diamond$ is derivable.

A configuration $\langle t_1, \ldots, t_n \mid eq_1, \ldots, eq_m \rangle$ with free names $x_1, \ldots, x_p$ is typeable by $\sigma_1^{s_1}, \ldots, \sigma_n^{s_n}$ if there are types $\rho_1, \ldots, \rho_p$ such that

$$x_1{:}\, \rho_1, \ldots, x_p{:}\, \rho_p, t_1{:}\, \sigma_1^{s_1}, \ldots, t_n{:}\, \sigma_n^{s_n}, eq_1{:}\, \diamond, \ldots, eq_m{:}\, \diamond$$

is derivable.

Lafont [9] defines a class of (standard) interaction nets without deadlocks (cycles of principal ports), called *semi-simple nets*, which are built by induction using a set of operations on nets. This property can be checked (for the general nets in INMPP) using the type system above.

Proposition 4.10 (Absence of Deadlock) *Typeable configurations are deadlock-free.*

More generally, we can guarantee that reduction does not create deadlocks if typeable rules are used.

Definition 4.11 *Typeable Rules.* Let Σ be a set of agents with their associated *Graft* rules. An interaction rule
$(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{t}) \bowtie (k_1, \ldots, k_{q-1}, -, k_{q+1}, \ldots, k_n)\beta(\vec{u}), eq_1, \ldots, eq_i$
is typeable if:

(i) There is a type derivation D with conclusion

$$(z_1, \ldots, z_{p-1}, -, z_{p+1}, \ldots, z_m)\alpha(\vec{x}) = (z'_1, \ldots, z'_{q-1}, -, z'_{q+1}, \ldots, z'_n)\beta(\vec{y}){:}\, \diamond$$

and leaves containing assumptions for the variables $\vec{z}, \vec{z'}, \vec{x}$ and $\vec{y}$.

(ii) There is a type derivation with the same assumptions leading to the conclusion:
$z_1 = l_1{:}\, \diamond, \ldots, z_m = l_m{:}\, \diamond,\ z'_1 = k_1{:}\, \diamond, \ldots, z'_n = k_n{:}\, \diamond,$
$x_1 = t_1{:}\, \diamond, \ldots, x_h = t_h{:}\, \diamond,\ y_1 = u_1{:}\, \diamond, \ldots, y_j = u_j{:}\, \diamond,\ \ eq_1{:}\, \diamond, \ldots, eq_i{:}\, \diamond$

(iii) And whenever an equation
$(l'_1, \ldots, l'_{p-1}, -, l'_{p+1}, \ldots, l'_m)\alpha(\vec{t'}) = (k'_1, \ldots, k'_{q-1}, -, k'_{q+1}, \ldots, k'_n)\beta(\vec{s'})$ is typeable, its type derivation is obtained by using instances (replacing type-variables by types) of the *Graft* rules for α and β applied in D.

Example 4.12 The interaction rules for amb (see Example 4.6) are typeable using the following *Graft* rule:

$$\frac{\Gamma, t_1{:}\, \varphi^s, t_2{:}\, \varphi^s}{\Gamma, \mathsf{amb}(t_1, t_2){:}\, (\varphi^s, \varphi^s)}\ (\textit{Graft}\ \mathsf{amb})$$

To check the typeability of the first rule for instance, we build a derivation for the active pair

$$(-, y)\mathsf{amb}(z, z') = \alpha(\vec{x}){:}\, \diamond$$

which requires assumptions $\alpha(\vec{x})\colon \sigma^{-s}$, $y\colon \sigma^{-s}$, $z\colon \sigma^{s}$ and $z'\colon \sigma^{s}$, where σ^{-s} is the type given to α in its *Graft* rule. Since the equations $\alpha(\vec{x}) = z, l = y, l = z'$ are trivially typeable with these assumptions, the rule is typeable.

Proposition 4.13 (Subject Reduction) *The rules Indirection, Interaction, Collect and Multiset, preserve typeability and types.*

As a consequence of Prop. 4.10 and 4.13, a typeable system remains deadlock free if we use typeable interaction rules for reduction.

4.2 Equivalence between INAMB and INMPP

Clearly, INAMB is included in INMPP. In this section we show that INMPP can be encoded in INAMB: we will simulate agents with n principal ports using the agents with a maximum of two principal ports available in INAMB. Let Σ be a set of agents and $\mathcal{R}$ a set of interaction rules in INMPP. The image in INAMB of Σ will be called Σ'. For each agent $(\vec{t})\alpha(\vec{y})$ of Σ, we will emulate separately the non-deterministic choice between principal ports and the sequential reduction rules. For the encoding of the non-deterministic choice, let $(\vec{t})S\mathsf{amb}_n(\vec{x})$ be the following configuration (see Fig. 3, right):

$$(\vec{t})S\mathsf{amb}_n(\vec{x}) =^{def} \langle \vec{x}, \vec{t} \,|\, \forall i, 1 \leq i \leq n, t_i = \delta(\mathsf{i}(l_i), k_i), (\vec{l})\mathsf{amb}_n(select(\vec{x}, \vec{k}))\rangle$$

which selects one principal port to interact, and copies (with a duplicator agent δ) the nets connected to the other principal ports in case they are used in a rule as auxiliary information. More precisely,

- $t_i = \delta(\mathsf{i}(l_i), k_i)$ duplicates and marks each input with a label,
- $(\vec{l})\mathsf{amb}_n(s)$ chooses in a non-deterministic mode an input
- $select(\vec{x}, \vec{k})$ selects the arguments corresponding to the chosen rule.

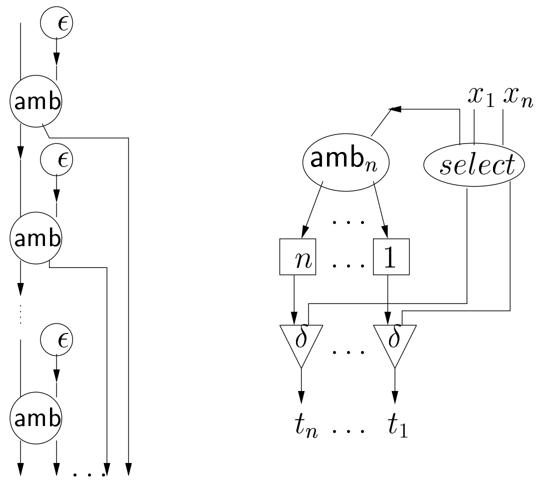


Fig. 3. Representation of amb_n and $S\mathsf{amb}_n$

Let us now define the agents introduced in the configuration $(\vec{t})S\mathsf{amb}_n(\vec{x})$. For labelling the inputs, we use unary agents $\mathsf{1}, \mathsf{2}, \ldots, \mathsf{n}$, which will mark an agent before it interacts with amb so that we can know after the interaction which principal port of amb was used. We also use a clearing agent Cl to remove the labels. The interaction rules for these labelling agents are:

$$\mathsf{i}(\mathsf{j}(x)) \bowtie \mathsf{j}(\mathsf{i}(x)) \qquad \mathsf{i}(\alpha_i(\vec{u})) \bowtie \alpha(\vec{u}) \qquad \mathsf{i}(\mathsf{Cl}(x)) \bowtie \mathsf{Cl}(\mathsf{i}(x))$$

$$\mathsf{Cl}(\mathsf{Cl}(x)) \bowtie \mathsf{Cl}(\mathsf{Cl}(x)) \qquad\qquad \mathsf{Cl}(\alpha(\vec{u})) \bowtie \alpha_i(\vec{u})$$

The net amb_n behaves like amb but with n principal ports (see Fig. 3, left). It is defined by induction with the following configuration, where ϵ is the eraser:

$$(\vec{t})\mathsf{amb}_n(s) =^{def} \langle s, \vec{t} \mid (t_1, s')\mathsf{amb}(s, \epsilon), (t_2, \ldots, t_n)\mathsf{amb}_{n-1}(s')\rangle$$

The agent *select* is defined by the following rules:

$$select(l_p, l_1, \ldots, l_{p-1}, l_{p+1}, \ldots, l_m, l_1, \ldots, l_{p-1}, \mathsf{p}(l_p), l_{p+1}, \ldots, l_m) \bowtie \alpha_p(\vec{\epsilon})$$

To emulate the sequential part of an agent $(\vec{t})\alpha(\vec{y})$ in INMPP with m principal ports and its rules $\mathcal{R}$, we use an agent α' with one principal port, which is intuitively the projection of the agent on each of its principal ports. This agent α' can have a label such as $\alpha'_1, \ldots, \alpha'_n$, which is a mark of the selected principal port of the agent interacting with it.

For each rule in $\mathcal{R}$ between the pth principal port of α and the qth principal port of β, we create a rule between α'_q and β'_p in $\mathcal{R}'$. Before giving the definition of the rules in $\mathcal{R}'$, we formalize the encoding of nets (configurations).

To simplify the encoding we will consider a class of *shallow configurations* where the interface contains only variables (this is always possible thanks to the Collect rule) and equations are of the form

- $t = u$ where t, u are either variables or terms $(l_1, \ldots, -, \ldots, l_m)\alpha(\vec{t})$ where $\vec{l}$ and $\vec{t}$ are variables,
- or $(\vec{l})\alpha(\vec{t})$ where $\vec{l}$ and $\vec{t}$ are variables.

Proposition 4.14 *For any configuration c there exists a shallow configuration c' such that $c = c'$ modulo Indirection, Collect and Multiset.*

Definition 4.15 *Encoding Configurations.* We define a function θ from configurations in INMPP over a set Σ of agents into configurations in INAMB over a set Σ' of agents containing:

- the set of labelling agents and the clearing agent: $\mathsf{1}, \ldots, \mathsf{n}, \mathsf{Cl}$,
- amb, $select$, δ, ϵ,
- for each agent $\alpha \in \Sigma$, the agents α, α_i, α', α'_i.

The translation of a configuration c in INMPP will be done as follows: Let $\langle \vec{t} \mid \Delta \rangle$ be a shallow configuration equivalent to c (modulo Indirection, Collect and Multiset). Then $\theta(c)$ will simply be the configuration $\langle \vec{t} \mid \zeta(\Delta)\rangle$, where ζ is the function translating equations in INMPP to equations in INAMB as follows:

(i) an equation $x = y$ is not changed by ζ,

(ii) an equation $x = (l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{t})$ where $\vec{l}$ and $\vec{t}$ are variables, is translated as $\zeta((l_1, \ldots, l_{p-1}, x, l_{p+1}, \ldots, l_m)\alpha(\vec{t}))$

(iii) $(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{t}) = (k_1, \ldots, k_{q-1}, -, k_{q+1}, \ldots, k_r)\beta(\vec{u})$ is translated as the union of $\zeta((l_1, \ldots, l_{p-1}, z, l_{p+1}, \ldots, l_m)\alpha(\vec{t}))$ and $\zeta((k_1, \ldots, k_{q-1}, z, k_{q+1}, \ldots, k_r)\beta(\vec{u}))$, where z is a fresh variable,

(iv) $\zeta((\vec{l})\alpha(\vec{t}))$ is simply $l = \alpha(\vec{t})$ if α has only one principal port, otherwise it is the multiset of equations in the configuration $(\vec{l})S\mathsf{amb}_m(\alpha'(\vec{t}, g_2, \ldots, g_m), \mathsf{Cl}(g_2), \ldots, \mathsf{Cl}(g_m))$.

The result of $\theta(c)$ is unique modulo Indirection and Multiset (we always work with configurations modulo α-conversion).

Definition 4.16 *Encoding Rules.* The rules $\mathcal{R}$ in INMPP are encoded in INAMB by a set $\mathcal{R}'$ of rules containing:

- the rules for the labelling agents and clearing agent,
- the rules for amb, *select*, δ, and ϵ,
- for each rule in $\mathcal{R}$, which without loss of generality (thanks to the Indirection rules) we assume to be of the form
  $(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(\vec{t}) \bowtie (k_1, \ldots, k_{q-1}, -, k_{q+1}, \ldots, k_n)\beta(\vec{u}), \vec{eq}$
  where $\vec{l}, \vec{k}, \vec{t}, \vec{u}$ are variables, we include in $\mathcal{R}'$ the rule
  $\alpha'_q(l_{p-1}, \ldots, l_1, \vec{t}, l_m, \ldots, l_{p+1}) \bowtie \beta'_p(k_{q-1}, \ldots, k_1, \vec{u}, k_n, \ldots, k_{q+1}), \zeta(\vec{eq})$

Proposition 4.17 (Completeness) *For any configuration c in INMPP such that $c \to c'$ there exists a configuration d in INAMB, the image of c in the above encoding, such that $d \to^* d'$ where d' is the encoding of c' plus eventually some trees of δ, ϵ and labelling agents, which will be erased by further interactions.*

Proposition 4.18 (Soundness) *If $d = \theta(c)$ and $d \to^* d'$, then there exists a configuration c' such that $c \to^* c'$ and $d' \to^* d''$ where d'' is the encoding of c' plus eventually some trees of δ, ϵ and labelling agents, which will be erased by further interactions.*

Alexiev [1] shows that INMPP has the same expressive power as Interaction Nets with Multiple Connections (called INMC), and also shows that the finite π-calculus (i.e. the π-calculus without the operator of choice and replication) can be encoded in INMC. The same techniques can be applied to encode the finite π-calculus in INAMB, since we have shown how to encode INMPP using

INAMB. Due to space restrictions, we cannot give the encoding here and just state the result:

Proposition 4.19 (Encoding a Process Calculus)
The finitary π-calculus can be represented in INAMB.

5 Application: Implementation of Rewrite Systems

One of the motivations for adding non-deterministic primitives to interaction nets is to be able to implement parallel functions defined by term rewriting systems. As shown in [5], interaction nets can only implement a class of constructor term rewriting systems which satisfies a strong matching restriction: sequentiality (see [4]). The system defining Parallel-or in Section 3 does not satisfy this restriction. However, we can implement it in INAMB using the configuration:

$$\langle x, y, s \mid x = \delta(or(l, b), a), y = \delta(or(u, a), b), (l, u)\mathsf{amb}(s, \epsilon)\rangle$$

which we call $(x, y)Por(s)$, agents T and F to represent the booleans True and False, and rules $F \bowtie or(x, x)$ and $T \bowtie or(T, \epsilon)$. It is easy to show that, given two boolean terms t_1 and t_2, $(t_1, t_2)Por(s)$ reduces to true if one of the terms t_1 or t_2 reduces to true, and to false if both reduce to false.

More generally, we will show that INAMB can be used to implement the whole class of constructor term rewriting systems. A similar result was shown in [5] using an extension of interaction nets with state. The encoding in INAMB is simpler in that only one extra agent is added (the agent amb).

In the constructor systems used in most functional programming languages the set of function symbols is partitioned into a set $\mathcal{C}$ of constructors and a set $\mathcal{D}$ of defined functions, and every left-hand side $f(t_1, \ldots, t_n)$ of a rule satisfies $f \in \mathcal{D}$ and $t_1, \ldots, t_n$ are built out of constructors and variables. We will restrict our attention to left-linear systems, since the encoding of non-left-linear rules can be done in the same way as in [5] by using some standard interaction rules.

By adding new function symbols to the signature, we can assume without loss of generality that all the patterns used in the left-hand sides of rules have depth less than or equal to 1. We will show how to encode the rules defining each function symbol. Let $f \in \mathcal{D}$ be defined with m rules of the form

$$\mathcal{R}_p : \quad f(x_1, \ldots, c_{i_p}(\vec{y_i}), \ldots, c_{j_p}(\vec{y_j}), \ldots, x_n) \to t_{out:p}$$

where c_{k_l} are constructors and $t_{out:p}$ is a term where the variables $\vec{x}$ and $\vec{y}$ may occur. Note that these rules might have superpositions. We will define an n-ary Parallel-or to encode rules with superpositions.

We use indexed agents T_i and F_i to represent the booleans True and False, where the indexes represent positions. The binary Parallel-or is represented by the configuration $(x, y)Por(s)$ defined above, and rules $F_i \bowtie or(x, x)$ and $T_i \bowtie or(T_i, \epsilon)$. We define by induction the configuration $(\vec{l})Por_n(s)$ representing an n-ary Parallel-or: $\langle l_1, \ldots, l_n, s \mid (l_1, r)Por(s), (l_2, \ldots, l_n)Por_{n-1}(r)\rangle$.

Let R_{p_i} be a set of agents which give True, specifically the agent T_{p_i}, or False, specifically F_{p_i}, depending on whether the ith argument of f matches or not the constructor c_{i_p} of the rule $\mathcal{R}_p$, that is, for $1 \leq p \leq m$:

$c_{i_p}(\vec{y}) \bowtie R_{p_i}(T_{p_i}, \vec{y})$ and $\alpha(\vec{y}) \bowtie R_{p_i}(F_{p_i}, \vec{y})$ for $\alpha \neq c_{i_p}$.

Using a binary agent and with rules $and(y, y) \bowtie T_i$, $and(F_i, \epsilon) \bowtie F_i$ we define by induction the n-ary agent and_n such that $and_2 \equiv and$, and $and_n(z, and_{n-1}(z, y_3, \ldots, y_n), y_3, \ldots, y_n) \bowtie T_i$, $and_n(F_i, \vec{\epsilon}) \bowtie F_i$.

The following configuration, called $(\vec{l})R_p(s, \vec{y})$, checks whether the left-hand side of the rule $\mathcal{R}_p$ is matched:

$$\langle \vec{l}, s, \vec{y} \mid l_i = R_{p_i}(r_i, \vec{y_i}), r_1 = and_q(s, r_2, \ldots, r_q)\rangle$$

In the sequel, when there is no ambiguity, we use the name of the configuration to denote its multiset of equations.

The rewrite rules for f are then encoded with the interaction rules shown in Fig. 4 (right), which in the textual calculus are written:

$$f(t_{out:p}(\vec{z}), \vec{z}) \bowtie T_p, 1 \leq p \leq m$$

$$\forall p, 1 \leq p \leq m, f(x, \vec{u}) \bowtie F_p, (\vec{u})\delta_{mn}(\vec{t}, \vec{v}), (\vec{v_i})\vec{R_i}(\vec{z_i}, \vec{y_i}), (\vec{z})Por_m(f(y_{out}, \vec{t}))$$

where $(\vec{u})\delta_{mn}(\vec{t}, \vec{v}), (\vec{v_i})\vec{R_i}(\vec{z_i}, \vec{y_i}), (\vec{z})Por_m(f(y_{out}, \vec{t}))$ is used to loop if no rule matches. δ_{mn} is a compact notation for a net that creates m copies of a vector of n inputs.

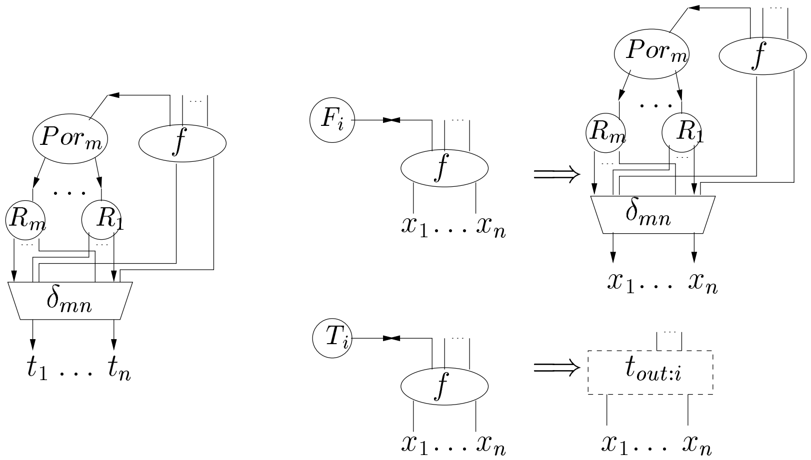


Fig. 4. Representation of $f(t_1, \ldots, t_n)$ (left), and rules for f (right).

Finally, terms are encoded using a function θ such that

- $\theta(x)$ is x,

- If c is a constructor, $\theta(c(\vec{s}))$ is the equation $x = c(\theta(\vec{s}))$, where the variable x is fresh, and denotes the root of the term,
- If f is a defined symbol of arity n with m rewrite rules, and the roots of the terms $\vec{t}$ are translated using variables $\vec{x_i}$, then $\theta(f(\vec{t}))$ is defined by (see Fig. 4, left): $\theta(t_1), \ldots, \theta(t_n), (\vec{x})\delta_{mn}(\vec{t'}, \vec{v}), (\vec{v_i})R_i(\vec{z_i, y_i}), (\vec{z})Por_m(f(x, \vec{t'})))$ where x denotes the root of the translated term and:
  · $(\vec{x})\delta_{mn}(\vec{t'}, \vec{v})$ duplicates $m+1$ times the input information $\vec{t}$;
  · $(\vec{l_j})R_j(\vec{r_j})$ analyzes the left hand side of the rule $\mathcal{R}_j$, and gives T_j if $\vec{l_j}$ matches this rule, F_j otherwise;
  · $(\vec{z})Por_m(s)$ chooses one rule for the reduction (between all the rules that matched);
  · finally, $s = f(x, \vec{t'})$ reduces $f(\vec{t})$ into the right hand side of the selected rule.

Term rewriting systems can be seen as a high-level (implicit) parallel language, but they are also a useful tool for the implementation of theorem provers based on equational logic. Therefore an encoding of term rewriting systems in INAMB can also be seen as a first step towards the development of new implementation techniques for equational theorem provers.

6 Conclusions

We have defined a simple though powerful extension of interaction nets, INAMB, and shown that several interesting languages can be encoded in this framework, specifically term rewriting systems, INMPP (and as a consequence a process calculus). We leave for future work the study of the encoding of the full π-calculus. We have also shown the limits of INAMB, which can provide encodings for angelic and infinity merge, but not for fair merge.

The advantage of remaining close to standard interaction nets is that an implementation of INAMB can be obtained by a minor modification of an interaction net implementation. We hope to use the calculus defined for INMPP to define an abstract machine for INAMB similar to the one defined in [14].

Acknowledgments: We thank C. Palamidessi and P. Panangaden for several pointers to related work, and the members of the CNRS/NSF project *Logic based specification and verification tools for concurrent languages* and of the Working Group on Interaction Nets for many enlightening discussions.

References

[1] V. Alexiev. *Non-deterministic Interaction Nets.* PhD thesis, University of Alberta, 1999.

[2] A. Asperti, C. Giovannetti and A. Naletto. The Bologna Optimal Higher-order Machine. *Journal of Functional Programming*, vol.6(6):763–810, 1996.

[3] 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.

[4] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Progress in Theoretical Computer Science. Birkhauser, 1993.

[5] M. Fernández and I. Mackie. From term rewriting to generalised interaction nets. In *Proceedings of PLILP'96. Programming Languages: Implementations, Logics, and Programs*, number 1140 in Lecture Notes in Computer Science. Springer-Verlag, 1996.

[6] M. Fernández and I. Mackie. A calculus for interaction nets. In *Proceedings of PPDP'99, Paris*, number 1702 in Lecture Notes in Computer Science. Springer, 1999.

[7] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of ACM Symposium Principles of Programming Languages*, pages 15–26, 1992.

[8] L. Khalil. Polymorphic types for interaction nets. In H. Ehrig and G. Taentzer, editors, *Proceedings of GRATRA'2000, Joint Appligraph and Getgrats Workshop on Graph Transformation Systems*, pages 257–266. Technical University of Berlin Report Number 2000–2, March 2000.

[9] Y. Lafont. Interaction nets. In *Proceedings, 17th ACM Symposium on Principles of Programming Languages*, pages 95–108, 1990.

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

[11] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–69. North Holland, 1963.

[12] P. Panangaden and V. Shanbhogue. The expressive power of indeterminate dataflow primitives. *Information and Computation*, 1992.

[13] D. Park. The fairness problem and non-deterministic computing networks. In *Proceedings of the 4th Advanced Course on Theoretical Computer Science.* Mathematisch Centrum, 1982.

[14] 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.

[15] E. W. Stark. On the relations computable by a class of concurrent automata. In *Proceedings, 17th ACM Symposium on Principles of Programming Languages*, pages 329–340, 1990.