INTERACTION NETS WITH McCARTHY’S amb:
PROPERTIES AND APPLICATIONS

MARIBEL FERNANDEZ
Computer Science
King’s College London
Strand, London WC2R 2LS,
U.K.
maribel@dcs.kcl.ac.uk

LIONEL KHALIL
LIENS (CNRS UMR 8548)
École Normale Supérieure
45 Rue d’Ulm, 75005 Paris,
France
khalil@di.ens.fr

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 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 [13] are a graphical model of computation derived from the multiplicative proof nets of linear logic [10]. 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 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 [11, 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, 7]. These extensions are roughly of two kinds: either they use the same nets but break the confluence property by relaxing some conditions in the definition of interaction rule (for instance, in [7] 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 principal ports. The definition of amb is inspired by McCarthy’s ambiguous choice operator amb(x, y), which can have a value x or y when both are defined, otherwise whichever is defined [15].

To demonstrate the expressive power of INAMB 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. Another application of this language is for the implementation of fractal equations.

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.

This is a revised and extended version of [6].

2. Interaction Nets

An interaction net system is specified by a set Σ of agents, and a set IR of interaction rules.

Each α ∈ Σ 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, and n auxiliary ports. Graphically, an agent α of arity n is represented as follows, where the principal port is depicted by an arrow:
A net \( N \) built on \( \Sigma \) 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. A net may be empty, or consist only of wires. 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 \(((\alpha, \beta) \Rightarrow N) \in \text{IR}\) replaces an occurrence of the active pair \((\alpha, \beta)\) by the net \(N\) which must have exactly the same interface as the active pair. The following diagram shows the general form of an interaction rule, using agents \(\alpha\) and \(\beta\) of arity 2 and 3 respectively. In the sequel we will not write the names of the ports in the interface when the correspondence is clear by position.

![Interaction Rule Scheme](image)

We write \(N_1 \Rightarrow^* N_k\) when there is a sequence of interactions

\[ N_1 \Rightarrow N_2 \Rightarrow \ldots \Rightarrow N_k. \]

Interaction rules must satisfy two very strong conditions: the interface of the active pair must be preserved, 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 [13] for a detailed presentation and examples of interaction nets.

### 2.1 Interaction Calculus

We recall the interaction calculus developed in [8], which provides a textual notation for nets and rules, as well as a formal account of the rewriting process.

Let \(\Sigma\) be a set of agents and \(\mathcal{N}\) a set of names (or variables) disjoint with \(\Sigma\). 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 \( \alpha \) and \( t_1, \ldots, t_n \) are terms with the restriction that each name can at most appear twice in a term \((\text{linearity constraint})\). \( \mathcal{N}(t) \) denotes the set of names occurring in \( t \), and we write \((t_1, \ldots, t_n)\) as \( \vec{t} \).

Intuitively, a term \( \alpha(t_1, \ldots, t_n) \) represents a net where the agent \( \alpha \) at the root has a free principal port and the auxiliary ports are connected to the nets represented by \( t_1, \ldots, t_n \). A variable that occurs twice in \( \alpha(t_1, \ldots, t_n) \) represents a connection between leaves of the tree. There are no active pairs in a term, since all the principal ports point in the same direction. To represent active pairs we use equations. An equation \( \alpha(\vec{t}) = \beta(\vec{u}) \) indicates a connection between the principal ports of the agents \( \alpha \) and \( \beta \). We denote a list \( t_1 = u_1, \ldots, t_n = u_n \) of equations by \( \vec{u} \). To represent nets we use configurations, which are pairs \( \langle \vec{t}, \Gamma \rangle \) where \( \vec{t} \) is the interface and \( \Delta \) is a multiset of equations describing the connections between the agents in the net. Configurations satisfy the linearity constraint \( (\text{each variable occurs at most twice}) \).

An interaction rule between \( \alpha \) and \( \beta \) 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 (see Fig. 2.1). Each variable occurs twice in the rule. Intuitively, since the rule preserves the interface, it is sufficient to indicate the subnets \( \vec{t}, \vec{u} \) to be connected to each port in the interface of the active pair (see [8] for details).

Let \( \mathcal{R} \) be a set of interaction rules. The rewriting relation \( c \to c' \) is defined by four computation rules that apply to configurations:

**Interaction:**
\[
\alpha(\vec{s}) \bowtie \beta(\vec{u}) \in \mathcal{R} \Rightarrow \langle \vec{t} \mid \alpha(\vec{s}) = \beta(\vec{u}), \Gamma \rangle \to \langle \vec{t} \mid \vec{s} = \vec{s}', \vec{u} = \vec{u}', \Gamma \rangle
\]

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

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

**Multiset:**
\[
\Theta \rightleftharpoons^* \Theta', \langle \vec{t}_1 \mid \Theta' \rangle \to \langle \vec{t}_2 \mid \Delta' \rangle, \Delta' \rightleftharpoons^* \Delta \Rightarrow \langle \vec{t}_1 \mid \Theta \rangle \to \langle \vec{t}_2 \mid \Delta \rangle
\]

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, and \( u[t/x] \) denotes the term \( u \) where the only occurrence of \( x \) is replaced by \( t \).

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

**Example 1. Addition of Natural Numbers.** We use agents \( Z, S, \text{Add} \) with \( \text{arity}(Z) = 0, \text{arity}(S) = 1, \text{arity}(\text{Add}) = 2 \). The \text{Add} agent adds the numbers connected to its principal port and its second auxiliary port, and outputs the result in the first auxiliary port, as the two interaction rules in Fig. 2.2 indicate. The textual version of the rules is:
\[
\text{Add}(S(x), y) \bowtie S(\text{Add}(x, y)) \quad \text{Add}(x, x) \bowtie Z
\]
The addition 1+0 is represented by the configuration \( \langle a | \text{Add}(a, Z) = S(Z) \rangle \), which rewrites to the configuration representing 1 as expected:

\[
\begin{align*}
\langle a | \text{Add}(a, Z) = S(Z) \rangle & \rightarrow \langle a | a = S(x'), y' = Z, Z = \text{Add}(x', y') \rangle \\
& \rightarrow^* \langle S(x') | Z = \text{Add}(x', Z) \rangle \\
& \rightarrow \langle S(x') | x'' = x', x'' = Z \rangle \\
& \rightarrow^* \langle S(Z) | \rangle
\end{align*}
\]

**Fig. 2.2:** Addition of Natural Numbers

**Example 2. Combinators \( \delta \) and \( \epsilon \).** Two standard agents \( \delta \) and \( \epsilon \) are used in interaction nets to duplicate and erase nets, respectively. Their reduction rules are shown in Fig. 2.3. Note that as a particular case, the interaction between \( \epsilon \) and a 0-ary agent produces an empty net.

The textual version of the interaction rules is:

\[
\begin{align*}
\alpha(\delta(x_1, y_1), \ldots, \delta(x_n, y_n)) & \bowtie \delta(\alpha(x_1, \ldots, x_n), \alpha(y_1, \ldots, y_n)) \\
\alpha(\epsilon, \ldots, \epsilon) & \bowtie \epsilon
\end{align*}
\]

**Fig. 2.3:** Reduction Rules for \( \delta \) and \( \epsilon \)
3. Adding a Non-Deterministic Agent

Interaction nets have been used as a tool to model and implement functional programming languages [3, 14]. They provide efficient evaluation strategies, and are well suited for parallel implementations since the order of the interactions does not matter [18]. 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:

\[
\begin{align*}
\text{por}(T, x) & \rightarrow T \\
\text{por}(x, T) & \rightarrow T \\
\text{por}(F, F) & \rightarrow F
\end{align*}
\]

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* [17]. 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 [16].

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 accept more than one interaction rule for each active pair (that is, several rules with the same left-hand side). In this way we obtain a limited form of 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 [15], is defined by rules as shown below, where \( \alpha \) is any agent.

When an agent \( \alpha \) has its principal port connected to a principal port of *amb*, an interaction can take place, and the agent \( \alpha \) 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 three examples: the definition of the parallel-or function, the definition of angelic and infinity merge, and the definition of several interaction rules for the same pair of agents using just one rule with amb.

**Example 3. Or and Parallel-or.** The agent or represents the boolean function or, it is defined (in standard interaction nets) by two interaction rules:

![Interaction Rules between amb and α](image)

The function Parallel-or must give a result True as soon as one of the arguments is True, even if the other one is undefined (for this reason it cannot be defined in conventional interaction nets [7]). Using an agent amb we can easily encode Parallel-or with the net:

![Parallel-or Net](image)

**Example 4. 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 L₁ and L₂, as shown in Fig. 3.2 (the angelic merge process is represented by the net at the left and the corresponding reduction rules for each α ∈ L₁, L₂ are given at the right).

Stark [19] 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 L₁;
Fig. 3.2: Angelic Merge

- 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 \( \text{amb} \) it is sufficient to show the implementation of a random number generator, which can then be used to build the oracle. Fig. 3.3 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 \( \epsilon \) and \( \delta \) are used to preserve the interface. Notice that there are active pairs in the right hand sides of these rules.

Fig. 3.3: Random Number Generator

We remark that the fair merge primitive cannot be implemented in interaction nets with \( \text{amb} \): this is a consequence of the results of [16].

**Example 5.** Non-deterministic Choice of Right-hand Side. One of the main
constraints in interaction nets is that we can only define one interaction rule for each pair of agents. Alexiev [1] considered an extension of interaction nets in which it is possible to write several rules for a pair of agents, and the choice of right-hand side is made in a non-deterministic way. Although in our extension with \texttt{amb} we have kept the restriction of one interaction rule for each pair of agents, it is easy to simulate the definition of several rules between $\alpha$ and $\beta$ with just one rule if we use \texttt{amb}. For instance the two rules in Fig. 3.4 are equivalent to the rule in Fig. 3.5.

The interaction rules between the agent $select$ and the agent $p$ (1 or 2) simply connect all the $x_i^p$ with $x_i$ and the $x_i^q (p \neq q)$ with an $\epsilon$ (to erase
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 [8] (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 \( \alpha \) with \( n \) auxiliary ports connected to nets \( t_1, \ldots, t_n \) and \( m \) principal ports connected to \( l_1, \ldots, l_m \), depicted:

\[
\begin{array}{c}
  \alpha \\
  t_1 \quad \ldots \quad t_n \\
  l_m \\
  l_1
\end{array}
\]

will be represented by a generalized term \((l_1, \ldots, l_m) \alpha(t_1, \ldots, t_n)\). If the \( p \)th 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 1.** Terms and Equations for INMPP. Let \( \Sigma \) be a set of agents and \( \mathcal{N} \) a set of names disjoint with \( \Sigma \). Terms are defined by the grammar:

\[
t ::= x \mid (l_1, \ldots, l_m) \alpha(t_1, \ldots, t_n)
\]

where \( x \in \mathcal{N} \), \( \alpha \in \Sigma \), \( \text{arity}(\alpha) = n \), \( m \) is the number of principal ports of \( \alpha \), \( t_1, \ldots, t_n \), \( l_1, \ldots, l_{p-1}, l_{p+1}, \ldots, l_m \) 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 \( \alpha \) has a single principal port we simply write \( \alpha(\bar{t}) \) instead of \((-)\alpha(\bar{t})\). 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 \( \alpha \).

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 \( \alpha \), \( \text{arity}(\alpha) = n \), 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. Implicit equations of the form $(\vec{l})\alpha(\vec{k})$ are also called *multiequations* since they indicate potential interactions between $\alpha$ and the agents at the root of the terms in $\vec{l}$.

**Definition 2.** Interaction rules in INMPP. An interaction rule between the $p$th principal port of $\alpha$ and the $q$th principal port of $\beta$ is written

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

where $\vec{l}, \vec{u}, \vec{l}, \vec{k}, \vec{e}q$ represent the right-hand side of the graphical rule. Each variable occurs twice in a 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 $\vec{l}, \vec{u}, \vec{l}, \vec{k}$), but it also allows extra equations $\vec{e}q$. 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 [13], to model non-deterministic primitives involving potentially infinite data structures it is natural to allow active pairs in right-hand sides (see Fig. 3.3).

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

**Definition 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 $\alpha$),
- $\vec{u}$ 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.** Configurations. A configuration is a pair $\langle \vec{t} | \Delta \rangle$ where $\Delta$ 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 $\Delta$. 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. The rewriting relation on configurations is denoted \( c \to c' \), and is generated by computation rules that generalize the rules for the interaction calculus given 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 \(p\)th principal port of \(\alpha\). 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 5.** Computation rules for INMPP. The Multiset rule does not change (see Section 2), only the Interaction, Indirection and Collect rules are generalised:

**Indirection (i)**

\[
x \in \mathcal{N}(eq) \Rightarrow \langle \vec{t} | x = u, eq, \Gamma \rangle \to \langle \vec{t} | eq[u/x], \Gamma \rangle
\]

**Indirection (ii)**

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

**Indirection (iii)**

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

**Interaction**

\[
\langle \vec{t} | (l_1, \ldots, l_{p-1}, -l_{p+1}, \ldots, l_m)\alpha(\vec{s}) = (k_1, \ldots, k_{q-1}, -k_{q+1}, \ldots, k_m)\beta(\vec{u}), \Gamma \rangle \rightarrow \langle \vec{t} | s = \vec{s}, u = \vec{u}, l_i = l_i' (1 \leq i \leq m, i \neq p), k_j = k_j' (1 \leq j \leq n, j \neq q), c\vec{q}, \Gamma \rangle
\]

**Collect (i)**

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

**Collect (ii)**

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

In the Interaction rule we use \(\alpha\)-conversion to get a fresh copy of the rule in \(\mathcal{R}\), as usual.

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.

**Example 6.** We show the textual version of the interaction rules for the agents \(\text{amb}\) and \(\alpha\) (see Fig. 3.1):

\((-l)\text{amb}(\alpha(\bar{x}), l) \Rightarrow \alpha(\bar{x}) \quad (l,-)\text{amb}(\alpha(\bar{x}), l) \Rightarrow \alpha(\bar{x})\)

and the interaction rule for angelic merge given in Fig. 3.2 (right):

\(AM(\alpha(x), z) \Leftrightarrow \alpha(z'), (z, z')\text{amb}(AM(x, y), y)\)

The following configuration \(c\) represents the angelic merge process shown in Fig. 3.2 (left), where the input sequences contain agents \(\alpha_i\) and \(\beta_i\) respectively:

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

Using the computation rules we can reduce it as follows:

\[c \rightarrow \text{Indirection} \quad \langle x \mid (\beta_1(\beta_2(z')), \alpha_1(\alpha_2(\alpha_3(z))))\text{amb}(AM(x, y), y) = \alpha_1(\alpha_2(\alpha_3(z))) \rangle \]

\[\rightarrow \text{Interaction} \quad \langle x \mid l = \beta_1(\beta_2(z')), x' = \alpha_2(\alpha_3(z)), \alpha_1(x') = AM(x, y), \quad y = y \rangle \]

\[\rightarrow \text{Interaction} \quad \langle x \mid l = \beta_1(\beta_2(z')), x' = \alpha_2(\alpha_3(z)), x' = z_1, x = \alpha_1(x_1), \quad y = z_0, (z_0, z_1)\text{amb}(AM(x_1, y_1), y_1), l = y \rangle \]

\[\rightarrow \text{Indirection} \quad \langle x \mid (\beta_1(\beta_2(z')), \alpha_2(\alpha_3(z)))\text{amb}(AM(x_1, y_1), y_1), \quad x = \alpha_1(x_1) \rangle \]

\[\rightarrow \text{Collect} \quad \langle \alpha_1(x_1) \mid (\beta_1(\beta_2(z')), \alpha_2(\alpha_3(z)))\text{amb}(AM(x_1, y_1), y_1) \rangle \]

The reduction sequence continues until both input sequences are empty.

Note that the Indirection and Collect rules do not change the underlying net, it is the Interaction rule that performs the actual computation. The Collect rules move to the head of the configuration the parts of the net that have already been evaluated, whereas 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. As a consequence we obtain the following property, indicating that there is a precise correspondence between the rewrite relations on nets (graph rewriting) and on configurations (textual rewriting).
Proposition 1. Soundness and Completeness of the calculus for INMPP.

(1) Let $N, N'$ be nets represented by the configurations $c, c'$. If $N \longrightarrow^* N'$ then $c \rightarrow^* c'$.

(2) Let $c, c'$ be configurations representing the nets $N, N'$. If $c \rightarrow^* c'$ then $N \longrightarrow^* N'$.

4.1 A Type System for INMPP

We will develop a polymorphic type system for INMPP inspired by [13, 8, 12]. We will use the type system to ensure the absence of deadlock. A deadlock is a cycle of principal ports, as depicted in Fig. 4.1.

![Fig. 4.1: A Deadlock](image)

Definition 6. 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 $\sigma^s$ with two components: $\sigma$ is the type of the information and $s$ is the direction of the information passing through the port ($-/+\,$ for input/output), modulo the equivalences:

$$(\sigma^-)^- = \sigma^+ \quad (\sigma^-)^+ = \sigma^- \quad (\sigma^+)^- = \sigma^+ \quad (\sigma^+)^+ = \sigma^-$$

$$(\sigma^s)^s = \sigma^{-s} \quad (\sigma^s)^s = \sigma^{+s} = \sigma^s$$

To define the type system we use one-sided sequents $t_1: \sigma_1^{s_1}, \ldots, t_n: \sigma_n^{s_n}$ where $t_1, \ldots, t_n$ are terms and $\sigma_1^{s_1}, \ldots, \sigma_n^{s_n}$ are types. To type equations we use the symbol $\diamondsuit$. We have the following typing rules:

Definition 7. For each agent $\alpha$ with $p$ principal ports there is a user-defined Graft rule:

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

which indicates that the types of the $p$ principal ports of $\alpha$ 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 Def. 1, this notation allows us to give the types of all the principal ports at the same time.
Γ, t: σ, u: τ, Δ (Exchange)  Γ Δ (Mix)

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

\( x: \sigma', x: \sigma^{-s} (Axiom) \)
\( \Gamma, t: \sigma^s, u: \sigma^{-s} (Cut) \)

\( \forall 1 \leq i \leq m, l_i: \sigma_i^{-s}, \Gamma_i \alpha(\vec{t}): (\sigma_1^{s_1}, \ldots, \sigma_m^{s_m}), \Gamma \) (MultiCut)

\( \forall 1 \leq i < j \leq m, l_i: \sigma_i^{-s}, \Gamma_i \alpha(\vec{t}): (\sigma_1^{s_1}, \ldots, \sigma_m^{s_m}), \Gamma \) (Select)

As an example, we give the Graft rules for the agents defined in Example 1, together with a polymorphic erasing agent \( \varepsilon \).

\( \Gamma, t: \text{nat}^+ (S) \)
\( \Gamma, Z: \text{nat}^+ (Z) \)
\( \Gamma, t_1: \text{nat}^-, t_2: \text{nat}^+ (Add) \)
\( \Gamma, \varepsilon: \sigma^s (\varepsilon) \)

**Definition 8.** Typeable Configurations. Let \( \{x_1, \ldots, x_m\} \) be the set of free names of \( t \), then \( t \) is a term of type \( \sigma^s \) 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 \( (l_1, \ldots, l_p)\alpha(\vec{t_1}, \ldots, \vec{t_n}) \), 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 \( \langle \vec{t_1}, \ldots, \vec{t_n} | eq_1, \ldots, eq_m \rangle \) 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

\( x_1: \rho_1, \ldots, x_p: \rho_p, l_1: \sigma_1^{s_1}, \ldots, l_n: \sigma_n^{s_n}, eq_1: \diamond, \ldots, eq_m: \diamond \)

is derivable.

Lafont [13] defines a class of (standard) interaction nets without deadlocks, 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 2.** (Absence of Deadlock) Typeable configurations are deadlock-free.

**Proof.** We show that for any sequent \( \Gamma \) derivable in the system given in Def. 7, the underlying net does not have a cycle of principal ports. The proof is by induction on the type derivation. We distinguish cases according to the rule applied at the root of the derivation tree.
(Graft $\alpha$): By induction, there are no cycles in the subnets associated to the premisses. Moreover, there are no connections between these subnets (since the axiom cannot be applied to variables belonging to different branches of the derivation tree), and the grafted agent has all its principal ports free. Therefore the conclusion is free of deadlock.

(Exchange), (Mix): Directly by induction.

(Axiom): Trivial, the net is just a wire.

(Cut): By induction, there are no cycles in the subnets associated to the premisses. Moreover, there are no connections between these subnets (since the axiom cannot be applied to variables belonging to different branches of the derivation tree). Since in the conclusion we only add a connection between two free principal ports, no cycle is created.

(MultiCut), (Select): Generalization of the previous case: by induction, there are no cycles in the subnets associated to the premisses, and moreover there are no connections between these subnets. In the conclusion we join together pairs of free principal ports, which cannot create a deadlock.

Remark 1. (1) The rules Cut and Select can be removed from the system if we consider configurations modulo Indirection, that is, we transform explicit equations into implicit ones before typing.

(2) For some applications, we can consider a weaker notion of typeability where only the consistent use of information is checked. For this it is sufficient to join all the premises in each typing rule.

(3) For some agents, called structural agents in [12] (such as the duplicator $\delta$), it is possible to overload the Graft rules by authorizing several distributions of premises (see [12] for details).

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

Definition 9. Typeable Rules. Let $\Sigma$ be a set of agents with their associated Graft rules. An interaction rule

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

is typeable if:

(1) There is a type derivation $D$ with conclusion

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

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

(2) There is a type derivation with the same assumptions leading to the conclusion:

$$z_1 = l_1: \diamond, \ldots, z_m = l_m: \diamond, y_1 = u_1: \diamond, \ldots, y_j = u_j: \diamond,$$

$$x_1 = t_1: \diamond, \ldots, x_h = t_h: \diamond, y_1 = u_1: \diamond, \ldots, y_j = u_j: \diamond, \overline{eq}: \overline{\diamond}$$
(3) And whenever an equation
\[(l'_1, \ldots, l'_{p-1}, -, l'_{q-1}, -, k'_q+1, \ldots, l'_{m})\alpha(\vec{v}) = (k'_1, \ldots, k'_q, -, k_{q+1}, \ldots, k'_m)\beta(\vec{s})\]
is typeable, its type derivation is obtained by using instances (replacing type-variables by types) of the Graft rules for \(\alpha\) and \(\beta\) applied in \(D\).

**Example 7.** The interaction rules for \(amb\) (see Example 6) are typeable using the following Graft rule:

\[
\frac{\Gamma, t_1: \varphi^s, t_2: \varphi^s}{\Gamma, \text{amb}(t_1, t_2): (\varphi^s, \varphi^s)} \quad (\text{Graft amb})
\]

We show how to type the rule

\[-, l)\text{amb}(\alpha(\vec{x}), l) \bowtie \alpha(\vec{x})\]

(the other rule is similar). We start by building a derivation for the active pair

\[-, y)\text{amb}(z, z') = \alpha(\vec{x}): \diamond\]

which requires assumptions \(\alpha(\vec{x}): \sigma^{-s}, y: \sigma^{-s},\) and \(z: \sigma^s, z': \sigma^s,\) where \(\sigma^{-s}\) is the type given to \(\alpha\) in its Graft rule:

\[
\frac{y: \sigma^{-s}}{\text{amb}(z, z'): (\sigma^s, \sigma^s)} \quad (\text{Graft amb})
\]

\[
\frac{z: \sigma^s, z': \sigma^s}{(\text{Select}) \quad \alpha(\vec{x}): \sigma^{-s}}
\]

\[
\frac{-, y)\text{amb}(z, z') = \alpha(\vec{x}): \diamond}{(\text{Cut})}
\]

Since the equations \(\alpha(\vec{x}) = z, l = y, l = z'\) are typeable with these assumptions as shown below, the rule is typeable.

\[
\frac{z: \sigma^s, z': \sigma^s}{(\text{Axiom}) \quad l: \sigma^{-s}, l: \sigma^s \quad y: \sigma^{-s}}
\]

\[
\frac{\alpha(\vec{x}): \sigma^{-s}}{(\text{Cut})}
\]

\[
\frac{\alpha(\vec{x}) = z: \diamond, l = y: \diamond, l = z': \diamond}{(\text{Cut})}
\]

Note that \(amb\) is a polymorphic agent, it can interact with agents of different types.

**Proposition 3. (Subject Reduction)** The rules Indirection, Interaction, Collect and Multiset, preserve typeability and types.

**Proof.** Similar to [8, 12]. \(\Box\)

As a consequence of Prop. 2 and 3, a typeable configuration 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 \( \text{amb} \) and agents with one principal port.

Let \( \Sigma \) be a set of agents and \( \mathcal{R} \) a set of interaction rules in INMPP. The image in INAMB of \( \Sigma \) will be called \( \Sigma' \). For each agent \((\vec{x})\alpha(\vec{y})\) of \( \Sigma \), 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 \( \text{amb}_n \) be an agent with \( n \) principal ports and \( n + 1 \) auxiliary ports (see Fig. 4.2) with the interaction rule described in Fig. 4.3. We introduce agents \( 1, \ldots, n \), which indicate an active principal port. In order to emulate \( \text{amb}_n \) with \( \text{amb} \), we introduce labelling agents \( \hat{i} \) with one auxiliary port and the interaction rule \( \alpha(\vec{e}) \Rightarrow \hat{i}(i) \), which outputs the label \( i \) of the active principal port captured by \( \hat{i} \) (see Fig. 4.4).

\[
\begin{align*}
s & \xrightarrow{i} \\text{out}_1 \ldots \text{out}_n \\
& \text{amb}_n \\
\downarrow & \downarrow \\
in_m & \ldots & \text{in}_1
\end{align*}
\]

**Fig. 4.2:** The agent \( \text{amb}_n \)

\[
\begin{align*}
\begin{array}{c}
s \text{amb}_n \text{out}_n \ldots \text{out}_1 \\
\downarrow \\
in_i \end{array} & \Rightarrow \\
\begin{array}{c}
s \text{amb}_n \text{out}_n \ldots \text{out}_1 \\
\downarrow \\
in_i \\
\alpha \\
\ldots \end{array}
\end{align*}
\]

**Fig. 4.3:** Interaction Rule for \( \text{amb}_n \) and \( \alpha \)

The agent \((\text{in}_1, \ldots, \text{in}_n)\text{amb}_n(\text{out}_1, \ldots, \text{out}_n)\) shown in Fig. 4.2 is defined in INAMB by the following configuration, which represents the net shown in Fig. 4.5.
Here an active principal port (arriving to $in_1, \ldots, in_n$) is selected for interaction while the nets arriving to the other principal ports are duplicated (using $\delta$) and kept to be used as auxiliary information in a subsequent interaction.
To encode an agent \((\vec{l})\alpha(\vec{t})\) in INMPP with \(m\) principal ports and \(n\) auxiliary ports and its rules \(R\), we use \texttt{amb}_m (i.e. \texttt{amb}, \delta, \epsilon, \text{ and the labelling agents which are all available in INAMB}), and an agent \(\alpha'(\vec{t}, \vec{l})\) with one principal port and \(n + m\) auxiliary ports (see Fig. 4.6). The agent \(\alpha'\) will interact with an agent \(i\) which indicates the activated principal port (if several principal ports of \(\alpha\) are ready to interact, only one is chosen by \texttt{amb}_m). The interaction rule between \(\alpha'\) and \(i\) outputs an agent \(\alpha'_i(l_{i+1}, \ldots, l_m, \vec{t}, l_1, \ldots, l_{i-1})\) with only one principal port and \(n + m - 1\) auxiliary ports (see Fig. 4.7), which represents a deterministic component of \(\alpha\). The textual version of the interaction rule is:

\[
\alpha'(\vec{t}, l_1, \ldots, l_{i-1}, \alpha'_i(l_{i+1}, \ldots, l_m, \vec{t}, l_1, \ldots, l_{i-1}), l_{i+1}, \ldots, l_m) \bowtie i
\]

For each rule in \(R\) between the \(p\)th principal port of \(\alpha\) and the \(q\)th prin-
principal port of $\beta$, we create a rule between $\alpha_p'$ and $\beta_q'$ in $R'$. Before giving the definition of the rules in $R'$, 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 $(l_1, \ldots, -l, \ldots, l_m)\alpha(\vec{t})$
  where $\vec{t}$ and $\vec{t}'$ are variables,
- or $(\vec{l})\alpha(\vec{t})$ where $\vec{l}$ and $\vec{t}$ are variables.

**Proposition 4.** For any configuration $c$ there exists a shallow configuration $c'$ such that $c = c'$ modulo Indirection, Collect and Multiset.

**Proof.** The Collect rule allows us to transform the interface and the Indirection and Multiset rules allow us to transform the equations to satisfy the requirements. $\square$

**Definition 10.** Encoding Configurations. We define a function $\theta$ from configurations in INMPP over a set $\Sigma$ of agents into configurations in IN-AMB over a set $\Sigma'$ of agents containing:

- labelling agents $i$ and $i'$,
- $\delta$ and $\epsilon$,
- $\text{amb}$,
- for each agent $\alpha \in \Sigma$ with one principal port: $\alpha$,
- for each agent $\alpha \in \Sigma$ with $m$ principal ports: $\alpha', \alpha'_1, \ldots, \alpha'_m$.

The translation of a configuration $c$ in INMPP will be done as follows:

Let $(\vec{t} | \Delta)$ be a shallow configuration equivalent to $c$ (modulo Indirection, Collect and Multiset). Then $\theta(c)$ will simply be the configuration $(\vec{t} | \zeta(\Delta))$, where $\zeta$ is the function translating equations in INMPP to equations in IN-AMB as follows:

1. An equation $x = y$ is not changed by $\zeta$,
2. 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 $\zeta((l_1, \ldots, l_{p-1}, x, l_{p+1}, \ldots, l_m)\alpha(\vec{t}))$,
3. $(l_1, \ldots, l_{p-1}, -l_{p+1}, \ldots, l_m)\alpha(\vec{t}) = (k_1, \ldots, k_{q-1}, -k_{q+1}, \ldots, k_r)\beta(\vec{u})$ 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,
4. $\zeta((\vec{l})\alpha(\vec{t}))$ is $l = \alpha(\vec{t})$ if $\alpha$ has one principal port, otherwise, if $\alpha$ has $m$ principal ports the translation is the multiset of equations in the configuration $(\vec{l})\text{amb}_m(\alpha'(\vec{t}, \vec{g}), g_m, \ldots, g_1)$. Graphically, the translation of $\alpha$ is the net shown in Fig. 4.6.

The result of $\theta(c)$ is unique modulo Indirection and Multiset (we always work with configurations modulo $\alpha$-conversion).
Definition 11. Encoding Rules. The rules $R$ in INMPP are encoded in INAMB by a set $R'$ of rules containing:

- the interaction rules for the agents $i$,
- the interaction rules for the agents used in $\text{amb}_n$, i.e., $\text{amb}$, $\dot{i}$, $\delta$, and $\epsilon$;
- for each rule in $R$, which without loss of generality (thanks to the Indirection rules) we assume to be of the form

$$(l_1, \ldots, l_{p-1}, -, l_{p+1}, \ldots, l_m)\alpha(i) \bowtie (k_1, \ldots, k_{q-1}, -, k_{q+1}, \ldots, k_n)\beta(\bar{a}), \bar{c}q$$

where $\bar{l}, \bar{k}, \bar{i}, \bar{u}$ are variables, we include in $R'$ the following rules (see Fig. 4.6 and Fig. 4.7):

$$\alpha'(\bar{l}, l_1, \ldots, l_{i-1}, \alpha'(l_{i+1}, \ldots, l_m, \bar{l}, l_1, \ldots, l_{i-1}), l_{i+1}, \ldots, l_m) \bowtie i,$$

$$\beta'(\bar{u}, k_1, \ldots, k_{j-1}, \beta'(k_{j+1}, \ldots, k_n, \bar{u}, k_1, \ldots, k_{j-1}), k_{j+1}, \ldots, k_n) \bowtie j,$$

$$\alpha'_p(l_{p+1}, \ldots, l_m, \bar{l}, l_1, \ldots, l_{p-1}) \bowtie \beta'_q(k_{q+1}, \ldots, k_n, \bar{u}, k_1, \ldots, k_{q-1}), \zeta(eq).$$

We give an example of the encoding of rules.

Example 8. Let $(in^1, in^2)\alpha(out^1, out^2)$, $(in^1, in^2)\beta(out^1, out^2)$, $(in^1, in^2)\gamma(out^1, out^2)$ and $\epsilon$ be agents in INMPP.

Assume we have the interaction rule shown in Fig. 4.8, which in the textual calculus for INMPP is written:

$$(\neg, out^1)\alpha(\epsilon, out^2) \bowtie (\epsilon, -)\beta(in^1, in^2), (in^1, in^2)\gamma(out^1, out^2)$$

For the translation, we define the set of agents:

- $\text{amb}$, $1$, $2$, $\epsilon$, $\delta$, $1$, $2$,
- $\alpha'$, $\beta'$, $\gamma'$; $\alpha_1$, $\beta_2$.

The images of $\alpha$, $\beta$, $\gamma$ and $\epsilon$ are:

- for $\alpha$, $\beta$ and $\gamma$ we use the nets $\zeta(\alpha)$, $\zeta(\beta)$, $\zeta(\gamma)$ (we show $\zeta(\alpha)$ in Fig. 4.9, the others are similar);
- the image of $\epsilon$ is $\epsilon$.

The image in INAMB of the interaction rule for $\alpha$ and $\beta$ is a rule for $\alpha'_1$ and $\beta'_2$, as shown in Fig. 4.10. The textual version of the INAMB rule is:

$$\alpha'_1(out^1, \epsilon, out^2) \bowtie \beta'_2(in^1, in^2, \epsilon), (a_1, a_2)\text{amb}(\gamma'(in^1, in^2, g_1, g_2), \epsilon),$$

$$out^1 = \delta(1(a_1), g_1), out^2 = \delta(2(a_2), g_2)$$

The encoding defined above can be seen as an implementation of INMPP in INAMB. This implementation is correct, in the sense that we can mimic all the INMPP reductions, as the two propositions below show. However, it introduces many “administrative” agents (duplicators, erasers, labelling agents) which will need more interactions to be erased.
**Proposition 5. (Completeness)** For any configuration \( c \) in \( \text{INMPP} \) such that \( c \rightarrow c' \) there exists a configuration \( d \) in \( \text{INAMB} \), the image of \( c \) in the above encoding, such that \( d \rightarrow^* d' \) where \( d' \) is the encoding of \( c' \) plus eventually some trees of \( \delta \), \( \epsilon \) and labelling agents, which will be erased by further interactions.

**Proof.** The most interesting case is when \( c \rightarrow \text{Interaction} c' \). Let

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

be the rule used in \( c \rightarrow c' \). The encoding \( \theta(c) \) of the configuration \( c \) (see Def. 10) contains the equations in the configurations

\[
(\vec{l})\text{amb}_m(\alpha' (\vec{l}', \vec{s}), \vec{s}) \text{ and } (\vec{k'})\text{amb}_n(\beta' (\vec{u'}, \vec{v}), \vec{v}), \text{ where } l'_p = k'_q.
\]

Reducing these equations we obtain the equations

\[
p = \alpha' (\vec{l}', \vec{s}), q = \beta' (\vec{u'}, \vec{v})
\]

which reduce to

\[
\alpha'_p(s_{p+1}, \ldots, s_m, \vec{l}', s_1, \ldots, s_{p-1}) = \beta'_q(v_{p+1}, \ldots, v_m, \vec{u'}, v_1, \ldots, v_{p-1})
\]
which in turn reduces to $d'$ (the image of $c'$ in the above encoding plus eventually some trees of labelling agents, $\delta$, and $\epsilon$, which will be erased by further interactions). □

**Proposition 6. (Soundness)** If $d = \theta(c)$ and $d \rightarrow^* d'$, then there exists a configuration $c'$ such that $c \rightarrow^* c'$ and $d' \rightarrow^* d''$ where $d''$ is the encoding of $c'$ plus eventually some trees of $\delta$, $\epsilon$ and labelling agents, which will be erased by further interactions.

**Proof.** We give a sketch of the proof. To mimic the reduction $d \rightarrow^* d'$ we proceed as follows:

- label all the multiport agents in $c$ and mark all the terms corresponding to those agents in $d$;
- identify the interaction rules used in $d \rightarrow^* d'$;
- for all the reduction rules involving the image of multiport agents, finish the multiple reduction of $\text{amb}$ and the reduction of labelling agents in $d'$, obtaining in this way the configuration $d''$;
- trace the rules used in $d \rightarrow^* d''$ and group together the reduction rules of $\text{amb}_m$ and labelling agents, which are the image by $\theta$ of multiport-agent reduction rules,
apply the corresponding sequence of reduction rules to \( c \), obtaining in this way a configuration \( c' \) such that \( c \rightarrow^* c' \). The configuration \( d'' \) is \( \theta(c') \) plus eventually some labelling, \( \delta \) and \( \epsilon \) 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 \( \pi \)-calculus (i.e. the \( \pi \)-calculus without the operator of choice and replication) can be encoded in INMC. The same techniques can be applied to encode the finite \( \pi \)-calculus in INAMB, since we have shown how to encode INMPP using INAMB. We have therefore the following result:

**Proposition 7. (Encoding a Process Calculus)**

The finitary \( \pi \)-calculus can be represented in INAMB.

### 5. Application: Implementation of Term Rewrite Systems

One of the motivations for adding non-deterministic primitives to interaction nets is to be able to encode parallel functions defined by term rewriting systems. As shown in [7], 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 (as shown in Example 3) using the configuration:

\[
(x, y) \text{or } (s, \text{true}) = \text{def} \langle x, y, s \mid (x, y) \text{amb}(a_1, a_2), \text{or}(s, a_2) = a_1 \rangle
\]

with agents \( T \) and \( F \) to represent the booleans True and False, and rules \( F \Join \text{or}(x, x) \) and \( T \Join \text{or}(T, \epsilon) \). It is easy to show that, given two boolean terms \( t_1 \) and \( t_2 \), \((t_1, t_2) \text{or}(s)\) 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 [7] using an extension of interaction nets with state. The encoding in INAMB is simpler in that only one extra agent is added (the agent \( \text{amb} \)). In the case of parallel-or, the encoding in INAMB is also more efficient. However, in what follows we will show the completeness of INAMB without concern for efficiency.

Before giving the encoding, let us recall the basic definitions and notations for term rewriting systems (we refer the reader to [5] for more details).

A term rewriting system is defined by a **signature** \( \mathcal{F} \) (which is a finite set of **function symbols** with fixed arities), a set \( \mathcal{X} \) of **variables**, the set \( T(\mathcal{F}, \mathcal{X}) \) of **terms** built up from \( \mathcal{F} \) and \( \mathcal{X} \), and a set of rewrite rules \( \mathcal{R} = \{ l_i \rightarrow r_i \} \), where \( l_i \notin \mathcal{X} \) and \( \text{Var}(r_i) \subseteq \text{Var}(l_i) \). We use \( \sigma \) to denote a substitution and write \( t \sigma \) for its application to \( t \). A term \( t \) **rewrites** to a term \( u \), written \( t \rightarrow u \), if there is a subterm \( s \) of \( t \) which is an instance of \( l \) (i.e. \( s \equiv \ell \sigma \)) and \( u \) is the term \( t \) where the subterm \( s \) is replaced by \( r \sigma \). We denote by
\( \rightarrow^+ \) (resp. \( \rightarrow^* \)) the transitive (resp. transitive and reflexive) closure of the rewrite relation \( \rightarrow \). \( \mathcal{R} \) is left-linear if all left-hand sides of rules in \( \mathcal{R} \) are linear terms (i.e. terms where each variable occurs at most once).

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 [7] 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

\[
\mathcal{R}_p : \quad f(x_1, \ldots, c_{i_p}(\vec{y}_i), \ldots, c_{j_p}(\vec{y}_j), \ldots, x_n) \rightarrow t_{\text{out}\, p}
\]

where \( c_k \) are constructors and \( t_{\text{out}\, p} \) 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 will give us information about rules. The binary Parallel-or is represented by the configuration \( (x, y) \text{Por}(s) \) defined above, and rules \( F_i \bowtie \text{or}(x, x) \) and \( T_i \bowtie \text{or}(T_i, \epsilon) \).

We define by induction the configuration \( (\bar{l}) \text{Por}_n(s) \) representing an \( n \)-ary Parallel-or (in the sequel, when there is no ambiguity, we use the name of the configuration to denote its multiset of equations):

\[
(\bar{l})\text{Por}_n(s) = \text{def} \langle l_1, \ldots, l_n, s \mid (l_1, r)\text{Por}(s), (l_2, \ldots, l_n)\text{Por}_{n-1}(r) \rangle.
\]

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

\[
c_{i_p}(\vec{y}) \bowtie R_{i_p}(T_{i_p}, \vec{y})
\]

and for \( \alpha \neq c_{i_p} \):

\[
\alpha(\vec{y}) \bowtie R_{i_p}(F_{i_p}, \vec{y})
\]

Using a binary agent and with rules \( \text{and}(y, y) \bowtie T_i, \text{and}(F_i, \epsilon) \bowtie F_i \) we define by induction the \( n \)-ary agent \( \text{and}_n \) such that \( \text{and}_0 \equiv T, \text{and}_1 \equiv \text{id}, \text{and}_2 \equiv \text{and}, \) and

\[
\text{and}_n(z, \text{and}_{n-1}(z, y_3, \ldots, y_n), y_3, \ldots, y_n) \bowtie T_i
\]

\[
\text{and}_n(F_i, \epsilon) \bowtie F_i
\]

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

\[ \langle \vec{l}, s, \vec{y} | l_i = R_{p_i}(r_i, \vec{y}_i), 1 \leq i \leq q, r_1 = \text{and}_q(s, r_2, \ldots, r_q) \rangle \]

The rewrite rules for \( f \) are then encoded with the interaction rules shown in Fig. 5.1 (right), which in the textual calculus are written:

\[ f(t_{\text{out}}(\vec{z}), \vec{z}) \triangleright T_p, 1 \leq p \leq m \]

\[ \forall p, 1 \leq p \leq m, f(x, \vec{u}) \triangleright F_p, (\vec{u})\delta_{mn}(\vec{t}, \vec{v}), (\vec{v})R_i(z_i, \vec{y}_i), (\vec{z})\text{Por}_m(f(y_{\text{out}}, \vec{t})) \]

where \((\vec{u})\delta_{mn}(\vec{t}, \vec{v}), (\vec{v})R_i(z_i, \vec{y}_i), (\vec{z})\text{Por}_m(f(y_{\text{out}}, \vec{t}))\) is used to loop if no rule matches. \( \delta_{mn} \) is a compact notation for a net that creates \( m + 1 \) copies of a vector of inputs.

Finally, terms are encoded using a function \( \theta \) such that

- \( \theta(x) \) is \( x \),
- If \( c \) is a constructor, \( \theta(c(\vec{s})) \) is the set of equations \( x = c(x), \theta(\vec{s}), \) where the variables \( x, \vec{x} \) are fresh, and denote the roots of the terms \( c(\vec{s}) \) and \( \vec{s} \) (resp.),
- 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 \( \vec{x} \), then \( \theta(f(\vec{t})) \) is defined by (see Fig. 5.1, left):

  \[ \theta(t_1), \ldots, \theta(t_n), (\vec{x})\delta_{mn}(\vec{t}, \vec{v}), (\vec{v})R_i(z_i, \vec{y}_i), (\vec{z})\text{Por}_m(f(x, \vec{t}))) \] where \( x \) denotes the root of the translated term and:

  - \( (\vec{x})\delta_{mn}(\vec{t}, \vec{v}) \) copies the input information \( \vec{t} \) in order to test whether a left-hand side of a rule is matched;
– \((l_j)R_j(r_j)\) analyzes the left hand side of the rule \(R_j\), and gives \(T_j\) if \(l_j\) matches this rule, \(F_j\) otherwise;
– \((z)P_{or_m}(s)\) chooses one rule for the reduction (between all the rules that matched);
– finally, \(s = f(x, \overrightarrow{t})\) reduces \(f(\overrightarrow{t})\) 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. Conclusion

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 \(\pi\)-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 have used the calculus defined for INMPP to define an abstract machine for INAMB similar to the one defined in [18].

INMPP is a very convenient language to implement fractal equations since the geometry of the fractals can be directly reflected in the net (more details about this application can be found in the forthcoming thesis of the second author). However, as any pure calculus, INMPP is not efficient in the manipulation of basic data (such as numbers or booleans) since these need to be encoded. This is not a new problem: in order to implement realistic programming languages, interaction nets have been generalized in [9] so that primitive data structures can be directly manipulated in the language.

Acknowledgements

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