Interaction Nets vs. the ρ-calculus: Introducing Bigraphical Nets

Maribel Fernández, Ian Mackie [2]

Department of Computer Science, King's College London
Strand, London WC2R 2LS, UK

François-Régis Sinot [1,3]

LIX, École Polytechnique, 91128 Palaiseau, France

Abstract

The ρ-calculus generalises both term rewriting and the λ-calculus in a uniform framework. Interaction nets are a form of graph rewriting which proved most successful in understanding the dynamics of the λ-calculus, the prime example being the implementation of optimal β-reduction. It is thus natural to study interaction net encodings of the ρ-calculus as a first step towards the definition of efficient reduction strategies. We give two interaction net encodings which bring a new understanding to the operational semantics of the ρ-calculus; however, these encodings have some drawbacks and to overcome them we introduce *bigraphical nets*—a new paradigm of computation inspired by Lafont's interactions nets and Milner's bigraphs.

Key words: Rewriting Calculus, Interaction Nets, Bigraphs.

1 Introduction

Pattern calculi [18,17,3,5,6,13] combine the expressiveness of pure functional calculi and algebraic term rewriting. The rewriting calculus, also called ρ-calculus [5,6], provides a simple framework generalising both term rewriting and the λ-calculus. It is an extension of the λ-calculus where we can abstract on patterns, not just on variables, hence providing a suitable foundational theory for modern programming languages with pattern-matching features.

Interaction nets [15] are graph rewrite systems which have been used for the implementation of efficient reduction strategies for the λ-calculus [12,1,16].

[1] Projet Logical: PCRI, CNRS, École Polytechnique, INRIA, Université Paris-Sud
[2] Email: maribel,ian@dcs.kcl.ac.uk
[3] Email: frs@lix.polytechnique.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

One of the main features of interaction nets is that *all* the computation steps are explicit and expressed in the same formalism; there is no external machinery. Also, since reduction in interaction nets is local and strongly confluent, reductions can take place in any order, even in parallel (see [19]), which makes interaction nets well-suited for the implementation of programming languages and rewriting systems [9,8]. Since pattern calculi encompass term rewriting and the λ-calculus, clearly there are natural questions about the existence of efficient implementations using interaction nets. In this paper we focus on the ρ-calculus, but our results can be applied to other pattern calculi.

We adapt interaction net implementations of the λ-calculus to deal with the specific features of the ρ-calculus. We concentrate on the problem of pattern-matching and take for granted that binders can be dealt with using one of the many available encodings of the λ-calculus mentioned previously. We give two alternative encodings of matching in interaction nets, both with certain advantages and drawbacks, and claim that a fully satisfactory solution cannot be obtained in the interaction net framework. This indeed comes as a surprise and highlights a deep difference between the λ-calculus and the ρ-calculus, namely that the ρ-calculus has more potential (implicit) parallelism. We then propose a third encoding using a particular class of bigraphs [14] which we call *bigraphical nets*. Bigraphs incorporate a notion of locality which is missing in interaction nets. We exploit this feature to overcome the drawbacks of the previous solutions, while still remaining close to a machine implementation.

Our work is modular with respect to the encoding of binders that is used, i.e. the same method can be applied to transform any interaction net implementation of the λ-calculus into an implementation of the ρ-calculus (even the bigraphical net encoding is modular, since interaction nets are also bigraphical nets). We can also easily export the solution to other calculi with patterns. Besides practical advantages (e.g. sharing of computations), graph representations are often useful to abstract away from syntactical details (e.g. α-conversion comes for free) and bring more understanding to the theory. In particular, the encodings presented in this paper highlight the differences between the various operational semantics defined for the ρ-calculus.

Section 2 recalls the ρ-calculus, interaction nets and bigraphs. Sections 3 and 4 give two interaction net encodings of the ρ-calculus, and their properties. Section 5 gives a third encoding using bigraphical nets, a new paradigm of graph rewriting incorporating locality. We conclude in Section 6.

2 Background

2.1 The Rewriting Calculus

We start with a short presentation of the ρ-calculus; for more details see [5,6,2]. We write $x, y, \ldots$ for variables and $f, g, \ldots$ for constants. The set of *ρ-terms*

(or just terms) $\mathcal{T}$ is defined by:

$$t, u ::= x \mid f \mid p \twoheadrightarrow t \mid [p \ll u].t \mid (t\ u)$$

where $p \in \mathcal{P}$ is a *pattern*; $p \twoheadrightarrow t$ is a *generalised abstraction* (it can be seen either as a λ-abstraction on a pattern p instead of a single variable, or as a standard term rewriting rule); $[p \ll u].t$ is a *delayed matching constraint* denoting a matching problem $p \ll u$ whose solutions (if any) will be applied to t; $(t\ u)$ denotes an *application* (we omit brackets whenever possible, and associate to the left). Terms are always considered modulo α-conversion (this will be realised for free in interaction nets later).

The ρ-calculus can be parametrised by the set $\mathcal{P}$ of patterns. Here we use *linear algebraic patterns*:

$$p ::= x \mid f\ p_1 \dots p_n$$

where each variable occurs at most once in p.

The reduction rules are the following:

$$\begin{array}{lrcl} (\rho) & (p \twoheadrightarrow t)\ u & \to & [p \ll u].t \\ (\sigma) & [p \ll u].t & \to & \sigma_{p \ll u}(t) \end{array}$$

The rule σ asks for an *external* matching algorithm to find a solution of the matching of p with u, and applies the corresponding substitution to t. Here we assume syntactic matching; under this assumption the calculus is confluent [6].

We will focus on the implementation of the σ rule, isolating as much as possible the problem of matching from the problems of implementing binders. This methodology is justified by the fact that the term $p \twoheadrightarrow t$ is extensionally equivalent to $x \twoheadrightarrow [p \ll x].t$, so that we can safely precompile terms in order to abstract only on variables and have explicit matchings from the beginning. To give a full implementation of the calculus, including the matching algorithm, we first define a version of the ρ-calculus with explicit matching inspired by [4]. Substitution is still implicit since it will be realised for free in the graphical representation.

$$\begin{array}{lrcl} (\rho) & (p \twoheadrightarrow t)\ u & \to & [p \ll u].t \\ (\sigma_v) & [x \ll u].t & \to & t\{x = u\} \\ (\sigma_{a_n}) & [f\ p_1 \dots p_n \ll f\ u_1 \dots u_n].t & \to & [p_1 \ll u_1] \dots [p_n \ll u_n].t \end{array}$$

Note that (σ_{a_n}) represents an infinite family of rules; we thus replace it by a

finite set of local rules, more suitable for an encoding into interaction nets:

$$\begin{array}{lrcl}
(a_c) & f\ t & \to & f \bullet t \\
(a_a) & (t \bullet u)\ v & \to & (t \bullet u) \bullet v \\
(\sigma_c) & [f \ll f].t & \to & t \\
(\sigma_a) & [(p \bullet r) \ll (u \bullet v)].t & \to & [p \ll u].[r \ll v].t
\end{array}$$

A matching $[p \ll u]$ may have no solution; this is called a *blocked matching*. We can add a rule to detect failure for constants:

$$(\bot) \qquad [f \ll g].t \quad \to \quad \bot \qquad \text{if } f \neq g$$

Now if there is a $\bot$ in a term, in which cases should the whole term evaluate to $\bot$? There are mainly two options:

$$(\textit{strict}) \qquad C[\bot] \quad \to \quad \bot \qquad \text{for any context } C[\cdot]$$

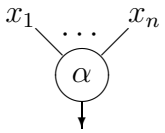
The *strict* rule corresponds to an exception-like semantics of matching failure (as in ML) e.g. even if the argument of an application is not used by the function, the result is $\bot$. In this context, a higher priority is given to this rule than to any other applicable rule. If we desire a non-strict semantics, this rule should be weakened to a particular class $\mathcal{C}$ of strict contexts:

$$(\textit{non-strict}) \qquad C[\bot] \quad \to \quad \bot \qquad \text{for any } C[\cdot] \in \mathcal{C}$$

We take $\mathcal{C} = \{([\,]\ t), t \in \mathcal{T}\}$, although a larger class of contexts is acceptable. Rules of the form $[p \ll u].t \to \bot$ if p and u have different head symbols, are unsafe and lead to non-confluence [5]. Our encodings will not implement unsafe rules (automatically, due to the strong confluence of interaction nets).

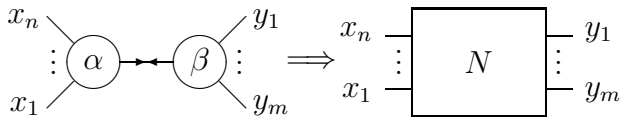
2.2 Interaction Nets

A system of interaction nets is specified by a set Σ of symbols with fixed arities, and a set $\mathcal{R}$ of interaction rules. An occurrence of a symbol $\alpha \in \Sigma$ is called an *agent*. If the arity of α is n, then the agent has $n + 1$ *ports*: a *principal port* depicted by an arrow, and n *auxiliary ports*. Such an agent will be drawn in the following way:



Intuitively, a net N is a graph (not necessarily connected) with agents at the vertices and each edge connecting at most 2 ports. The ports that are not connected to another agent are *free*. There are two special instances of a net: a wiring (no agents) and the empty net; the extremes of wirings are also called

free ports. The *interface* of a net is its set of free ports. An interaction rule $((\alpha, \beta) \Longrightarrow N) \in \mathcal{R}$ replaces a pair of agents $(\alpha, \beta) \in \Sigma \times \Sigma$ connected together on their principal ports (an *active pair* or *redex*) by a net N with the same interface. The following diagram shows the format of interaction rules (N can be any net built from Σ).



Reduction is local, and there can be at most one rule for each pair of agents. We use the notation $\Longrightarrow$ for the one-step reduction relation and $\Longrightarrow^*$ for its transitive and reflexive closure. If a net does not contain any active pairs then it is in normal form. One-step reduction satisfies the diamond property.

2.3 Bigraphs

In [14] a notion of graph transformation system is defined, using nested (or hierarchical) graphs called *bigraphs*. Bigraphs represent two kinds of structure: locality (nodes may occur inside other nodes) and connectivity (nodes have ports that may be connected by links). We recall the basic terminology of bigraphs and refer the reader to [14] for details and examples.

Nodes are labelled by *controls* with fixed arities; the arity of a control corresponds to the number of ports of the node. Links are attached to nodes from the inside or the outside, so bigraphs have both an inner and an outer interface. A control is *atomic* if it cannot contain a nested graph, otherwise it is non-atomic. The reduction relation is defined by a set of reaction rules, which are pairs of bigraphs (called redex and reactum). The redex has a *width*, corresponding to the number of sites it occupies in the outer bigraph (see [14]). A non-atomic control K can be specified as active, in which case reactions can occur inside, or passive, in which case reactions in the internal bigraph can only occur after the control K has been destroyed.

Interaction nets are a particular kind of bigraphs without nesting: all controls (called agents in interaction nets) are atomic, and have a distinguished port. Interaction rules can be seen as reactions in which both redex and reactum have width 1, and redexes are restricted to just two controls connected by one link through the distinguished ports.

3 A Simple Interaction Net Encoding of the ρ-calculus

We can obtain an implementation of the ρ-calculus starting from any off-the-shelf interaction net encoding of the λ-calculus (that we will not describe) and adding a matching algorithm as specified in the explicit ρ-calculus (see Section 2.1). We first describe an encoding that aims at simplicity; however,

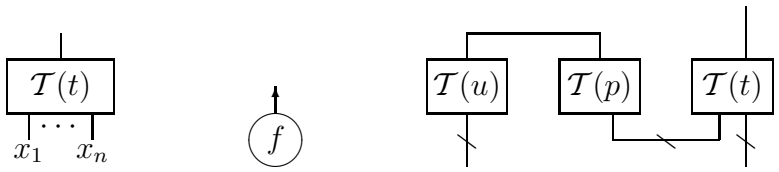


Fig. 1. Translation of Terms

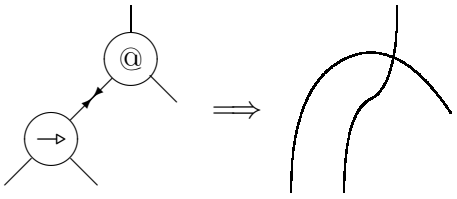


Fig. 2. Interaction Rule for Abstraction and Application

we will see that this encoding is not able to represent a non-strict ρ-calculus semantics, which will motivate the following sections.

We give a translation $\mathcal{T}(\cdot)$ of ρ-terms, and the interaction rules that will be used for solving matching constraints. A ρ-term t with free variables $\mathsf{fv}(t) = \{x_1, \ldots, x_n\}$ will be translated to a net $\mathcal{T}(t)$ with the root edge at the top, and n free edges corresponding to the free variables, as shown in Figure 1 (left). If t is a variable then $\mathcal{T}(t)$ is just a wire. For each constant f we introduce an agent as shown in Figure 1 (middle). A term of the form $[p \ll u].t$ is encoded as shown in Figure 1 (right) [4] which can be interpreted as the substitution in t of the (possible) solution of the matching (the left subnet corresponds to the matching problem $p \ll u$). We assume that terms have been precompiled to abstract only on variables, as described in the previous section; hence we can reuse the abstraction of the λ-calculus: We introduce an agent $\multimap$ with its principal port oriented upwards. Similarly for application and interaction between abstraction and application (β/ρ-reduction): we introduce an agent @ with its principal port oriented towards the left subterm, so that interaction with an abstraction is possible. We have the usual interaction rule between abstraction and application, given in Figure 2. We insist that the light-weight encoding of this section allows to immediately reuse the corresponding interaction rule of any encoding of the λ-calculus, without modification.

In this simple encoding, the matching algorithm is initiated by connecting the root of a pattern with the term to match. Thus, the rule (σ_v) (matching against a variable) is realised for free, as in the λ-calculus. To simulate (σ_a) and $(\bot)$, constants will interact. When two identical constants interact, they cancel each other to give the empty net, as indicated in Figure 3 (left). If the agents are not the same, then we introduce an agent fail, which represents a failure in the matching algorithm, as indicated in Figure 3 (right). Note that

[4] A dashed edge represents a bunch of edges (a *bus*).

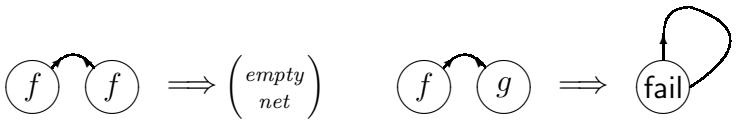


Fig. 3. Matching of constants (success and failure)

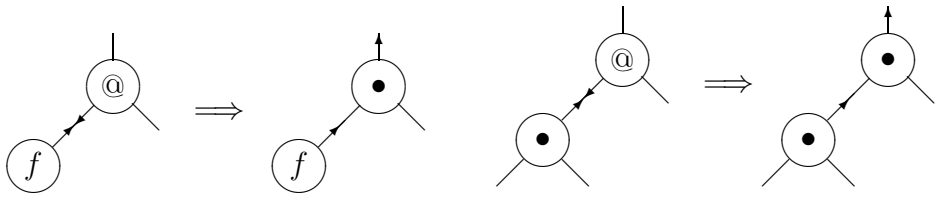


Fig. 4. Rules to transform patterns

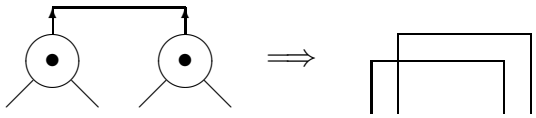
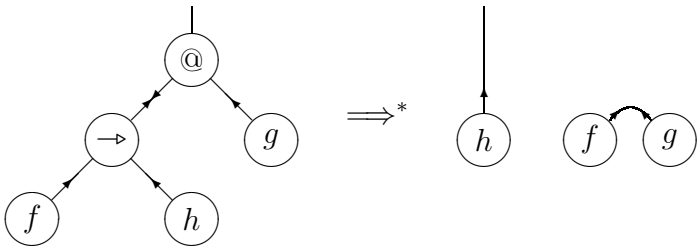


Fig. 5. Matching applications

the right-hand side of the second rule in Figure 3 is a deadlocked net (because the matching is disconnected). Interaction is not possible between the agent fail and the rest of the term; consequently, we interpret a net containing an agent fail anywhere as an overall failure. Unfortunately, this implements the rule (*strict*) as a side effect.

In the ρ-calculus application is used in two very different ways: on one hand as the application of an abstraction to a term, and on the other hand as a term constructor in patterns, as shown in Section 2.1. We have to convert a usual application (@) into a pattern application ($\bullet$) when it is part of an algebraic pattern (or term), using the rules in Figure 4. We also need a rule to match applications, which is given in Figure 5.

Example 3.1 The term $(f \twoheadrightarrow h)\ g$ evaluates to a blocked matching (failure) $[f \ll g].h$. This reduction is represented in interaction nets by:

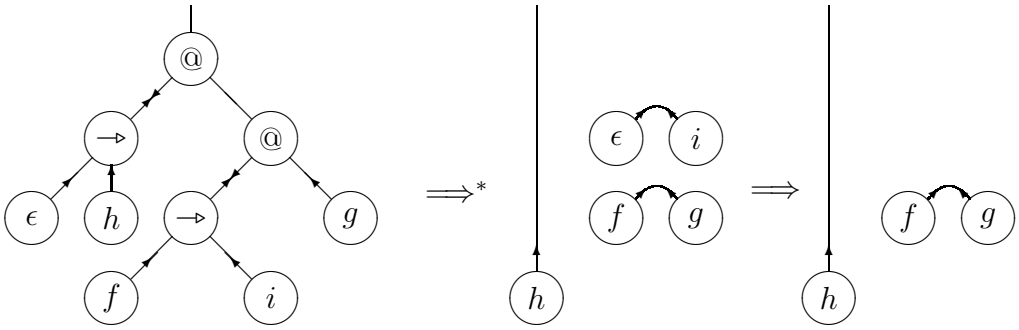


In this case there are two disconnected parts in the resulting net, one representing h and the other one representing the matching failure.

Properties of the encoding

In most applications of interaction nets, nets which become disconnected are no longer of interest, i.e. they are considered garbage and thus ignored. This is especially done because, in general, garbage collection requires a full traversal and even full evaluation, which motivated some work on strategies in interaction net [10,20]. However, the previous example shows that disconnected nets, even without any free ports, do matter in this interaction net encoding, which implies that we cannot stop reduction at the so-called interface-normal-forms [10]. On the other hand, this is very good news for a parallel implementation (which is also a motivation of interaction nets): disconnected components can easily be dispatched on different processors. Moreover, in our case, the only interesting information about these matching components is whether they finally evaluate to the empty net (successful matching) or not (matching failure, but this is undecidable).

Although good for parallelism, the nasty consequence of this observation is that we cannot implement a non-strict semantics. For instance, reduction of $(x \rightarrowtriangle h)((f \rightarrowtriangle i)\ g)$ gives (where the ϵ agent is used for erasure):



We cannot distinguish the result of this computation (for which just h could be a desirable answer) from the previous failing one. This corresponds to adding the strict rule (cf. Section 2.1), treating pattern-matching failures as exceptions (cf. [6] for a corresponding big-step semantics, although we allow some terms to be reduced before matching). If we want pattern-matching failure to be treated as a matching failure in a lazy language rather than as an exception, then we cannot disconnect matching constraints. We present an alternative encoding which maintains connectivity in Section 4.

The simulation of the rule (σ_v) is under the responsibility of the encoding of the λ-calculus used. Rules corresponding to (a_c), (a_a), (σ_c), (σ_a) and $(\bot)$ have been given. The (*strict*) rule is realised by the interpretation of agent fail, as we explain below. Assuming correctness of the encoding of the λ-calculus we can show (using $\rightarrow$ to denote reduction in the explicit ρ-calculus with the strict rule):

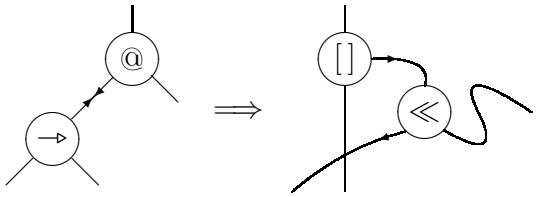
Correctness: If $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$ connected, then $t \rightarrow^* u$. If $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$ disconnected but without failure, then there is a v such that $\mathcal{T}(u) = \mathcal{T}(v)$ and $t \rightarrow^* v$. If $\mathcal{T}(t) \Longrightarrow^* N$ where N is a net with a failure, then $t \rightarrow^* \bot$.

Completeness: If t is closed and $t \to^* u \neq \bot$ in normal form, then $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$. If $t \to^* \bot$ then $\mathcal{T}(t) \Longrightarrow^* N$ containing a fail agent.

The provisos of the correctness properties are justified by situations of the form $(f \twoheadrightarrow t)\ \Omega \to [f \ll \Omega].t$ where Ω is a non-terminating term, thus the matching constraint cannot be eliminated, but it is represented by a disconnected interaction net (i.e. the readback is not unique). The restriction to t closed and u in normal form for completeness are standard [16]. Since we disconnect the matchings from the rest of the net, we can only interpret a fail agent remaining at the end of the reduction as failure; hence we are modelling a strict calculus (i.e. the explicit ρ-calculus plus the $(\bot)$ and (strict) rules).

4 Introducing a Matching Agent

Disconnecting matching constraints increases the parallelism of the implementation but does not allow us to implement a non-strict semantics. The obvious solution to this problem is to maintain connectivity by using an explicit agent for matching. Then we can keep track of the point where a matching failure may have occurred, thus staying closer to the syntax and behaviour of standard ρ-calculus. We make the minimum amount of changes to the previous encoding. The counterpart of the ρ-rule now creates an explicit matching agent $\ll$ linked to the rest of the net by an agent $[\,]$ (the right-hand side of the rule is thus the new representation of a matching constraint):



We have introduced two new agents: $\ll$ will take care of the matching and $[\,]$ will simply attach the matching to the right place, exactly mimicking the structure of terms: $[p \ll u].t$. Note that the agent $[\,]$'s principal port is pointing towards the agent $\ll$, i.e. it will wait until the matching is done and look at the result, which will be either success or failure. If the matching cannot be solved in either way, this agent will stay there forever.

Success will be represented by an agent $\top$, failure by an agent $\bot$, both 0-ary, with the expected interactions with agent $[\,]$. The ideas are settled, now the development of the matching algorithm is straightforward.

It is clear that each rewrite step in the explicit ρ-calculus corresponds to a sequence of interaction rules in this encoding, hence properties of simulation are stronger than with the previous encoding. Here $\to$ denotes the explicit non-strict ρ-calculus reduction relation, and t, u belong to $\mathcal{T} \cup \{\bot\}$.

Proposition 4.1 (Correctness and completeness) *If $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$, then $t \to^* u$. If t is closed and $t \to^* u$ in normal form, then $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$.*

The problem with this encoding is that the agent [ ] blocks any further computation between the root and what was the body of the abstraction. For instance, in the term $(a \twoheadrightarrow b)\ ((p \twoheadrightarrow c)\ t)$ where a, b, c are constants and p, t are terms, we will have to complete the matching of t against p (which may be costly) before noticing that a and c do not match. This was not the case in the first encoding. The moral interpretation of this result is that the drawback identified in the first encoding is not about connectivity but about locality. Hence we naturally turn our attention towards an extended framework inspired by bigraphs: bigraphical nets.

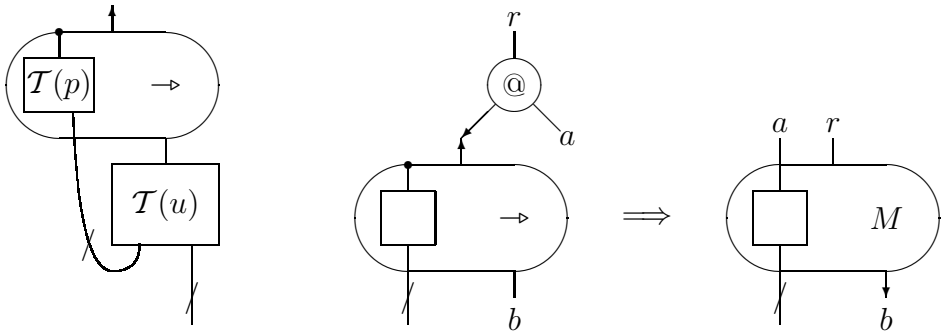
5 Using Bigraphical Nets

Bigraphs [14] introduce a notion of locality (using nesting to indicate that a graph is local to a certain node) which is missing in interaction nets, and which is a key to solving the problems of the previous encodings: if we can specify that a pattern (and later a matching constraint) is local to a certain abstraction, we can keep track of occurrences of failure and implement a non-strict ρ-calculus without introducing an explicit matching agent. Note that bigraphs permit links between nested nets and external subgraphs (unlike hierarchical graphs [7]) and rewriting can take place across control boundaries, both features which will be of use in our encoding of the ρ-calculus. To encode the ρ-calculus we will use a subclass of bigraphs that contains interaction nets, and that we call *bigraphical nets*.

Definition 5.1 Bigraphical nets are bigraphs in which each control has a distinguished *principal* port (the remaining ports are called auxiliary), and links connect at most 2 ports. Reaction rules define interactions between two controls connected by their principal ports (and their sites), or interactions of a control with its local sites, preserving the interfaces.

For examples of a bigraphical net and a reaction rule, see Figure 6. Note that, in contrast with interaction nets, a left-hand side can specify the location in which the reacting controls are, or the locations contained in these controls, and reactions can take place across boundaries. A full study of bigraphical nets as a computational framework is beyond the scope of this paper, however, we remark that all the examples of bigraphs for the π-calculus and ambient calculus given in [14] (part I) can be redefined as bigraphical nets by adding principal ports and copy/erase controls to preserve the interface of the reactions. Comparing with the properties of interaction nets, we remark that confluence does not hold in general for bigraphical nets, because of the possibility of interactions across boundaries. However reduction is still local. The latter point is crucial for implementation.

Bigraphical nets can be seen as a particular class of *higher-order nets* [11]. Following the usual terminology of interaction nets, controls in bigraphical nets are also called agents, and reaction rules are called interaction rules.



(a) Abstraction (b) Rule (ρ)

Fig. 6. Bigraphical encodings of abstraction and rule (ρ)

We will use the following agents for the encoding:

- $\multimap$ of arity 2, which is a non-atomic agent representing abstraction;
- @ of arity 2, which is an atomic agent representing application;
- $f, g, \ldots$ of arity 0, for constants;
- a non-atomic agent M of arity 1 to represent matching problems;
- a family of non-atomic agents α_M, where α is any of the agents above except M;
- $\perp$ of arity 0 (atomic), to represent matching failure.

All non-atomic agents permit interactions to take place inside nested nets, and also across the agent boundary (i.e. they are active). The translation of an application, a variable, or a constant, are the same as in the first interaction net encoding. We give the translation of an abstraction $p \multimap u$ in Figure 6(a), where we omit the encoding of the box, as before. The reaction implementing the ρ-rule is given in Figure 6(b). The rules to implement the matching algorithm are given in Figure 7. The first rule allows agents from the body of the abstraction (except for copy and erase agents) to interact at the root, keeping track of the matching constraint. We assume that the non-atomic agents α_M have the same behaviour as α (i.e. same interaction rules), and permit the same interactions as M between the net inside and the outside (whence the name α_M). We omit the rule for M and ϵ which should erase M and its content, sending ϵ agents along the interface. If α is itself an α_M then the interaction produces again α_M with an additional matching constraint. The next rule eliminates the M agents after the matching constraint has been solved. The last three rules decompose matching problems with application, and detect success or failure when constants interact (we assume $\alpha \neq f$). Similar rules for α_M are omitted (with an empty net it reduces to α).

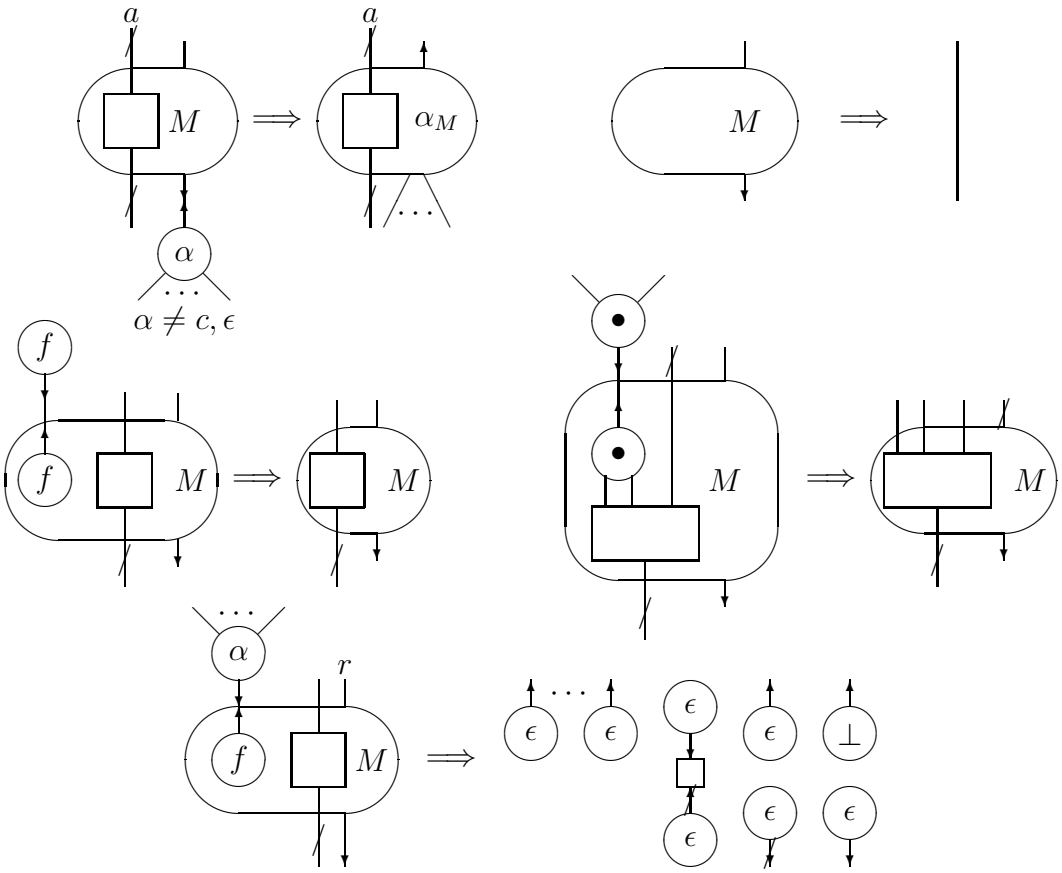


Fig. 7. Matching using bigraphs

Simulation of ρ-calculus reductions
The interaction rules above simulate the rules ρ and σ of the explicit ρ-calculus (the rule σ_v is obtained for free). This encoding allows us to implement a non-strict semantics (and also a strict one, see below), similarly to the encoding that uses an explicit matching agent. Unlike the latter, the matching agent does not block interactions between the net representing the body of the abstraction and the context of the term. We have the following results, which are similar to those stated in Section 4.

Proposition 5.2 (Correctness and completeness) *If $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$, then $t \to^* u$. If t is closed and $t \to^* u$ in normal form, then $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$.*

Example 5.3 We now reconsider the examples of Section 4 with this encoding. Evaluation of the term $(f \twoheadrightarrow h)\ g$ produces a matching failure (both under the strict and non-strict semantics) as shown in Figure 8. On the other hand, the term $(x \twoheadrightarrow h)((f \twoheadrightarrow i)\ g)$ reduces to h, since the ϵ agent erases the failing matching (therefore implementing the non-strict semantics), as shown in Figure 9. To implement a strict semantics it is sufficient to take out the rule between ϵ and a fail agent, and interpret any net containing $\perp$ as failure.

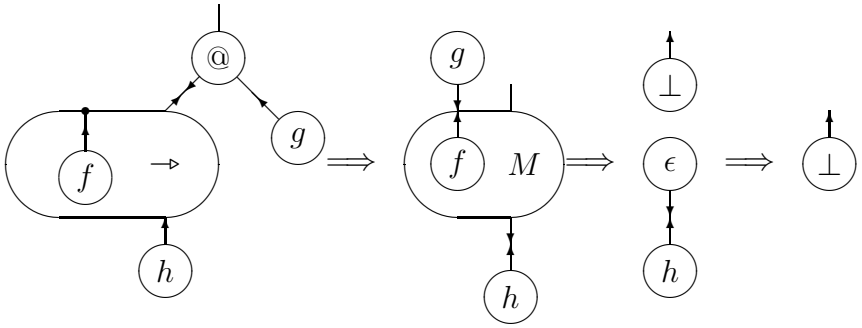


Fig. 8. Evaluation of $(f \multimap h)\ g$ with bigraphical nets

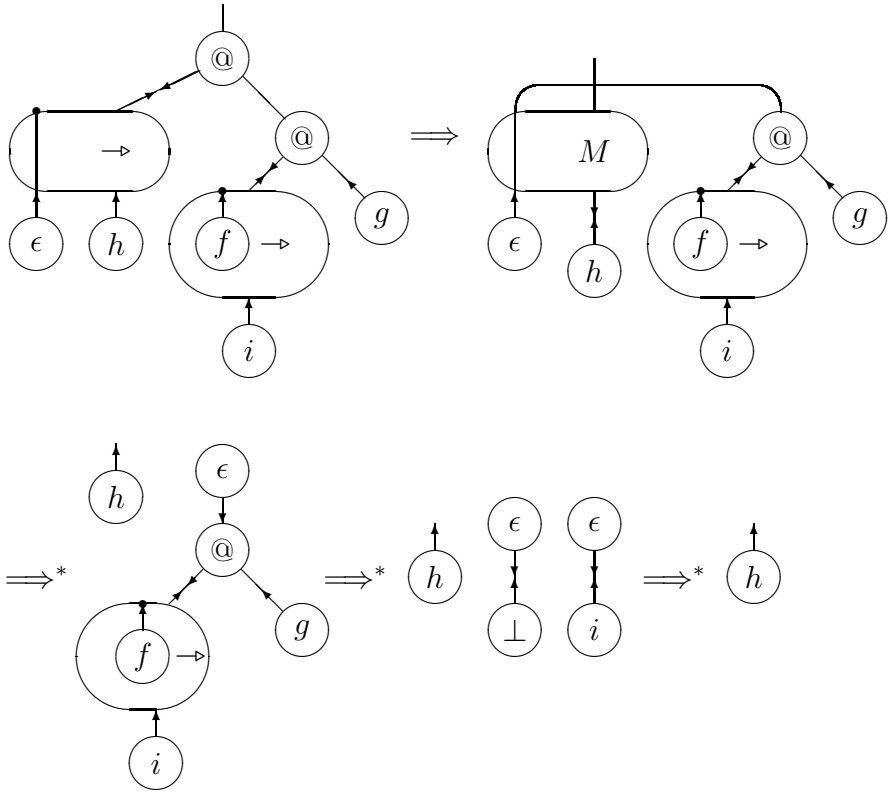


Fig. 9. Evaluation of $(x \multimap h)((f \multimap i)\ g)$ with bigraphical nets

6 Conclusion

Although there are several good encodings of the λ-calculus in interaction nets, the problem of designing a satisfactory encoding of the ρ-calculus is non-trivial, because the problem of matching introduces more potential parallelism than in the λ-calculus, which is difficult to handle satisfactorily with interaction nets because they lack a notion of locality. We finally proposed a framework which is expressive enough for this.

For the sake of clarity, we omitted some details (like free variables in patterns, and non-linear patterns) in order to isolate the real problem, that is the

problem of matching. We nevertheless assure the reader that the issue would stay unchanged in a more realistic implementation.

The original motivations for this work were to provide grounds for implementing in a distributed setting the ρ-calculus, which can be seen as a foundational model for functional languages featuring pattern-matching, or for rewriting. This actually led us far beyond, and we have also proposed bigraphical nets as a model of distributed computation with local synchronisations which is suitable for our particular problem (and for a larger class of problems). This will be the subject of future work.

References

[1] A. Asperti, C. Giovannetti, and A. Naletto. The Bologna optimal higher-order machine. *Journal of Functional Programming*, 6(6):763–810, Nov. 1996.

[2] G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure patterns type systems. In *Principles of Programming Languages - POPL2003, New Orleans, USA*. ACM, Jan. 2003.

[3] V. Breazu-Tannen, D. Kesner, and L. Puel. A typed pattern calculus. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science (LICS'93), Montreal, Canada*, 1993.

[4] H. Cirstea, G. Faure, and C. Kirchner. A rho-calculus of explicit constraint application. In *Proceedings of the 5th workshop on rewriting logic and applications*. Electronic Notes in Theoretical Computer Science, 2004.

[5] H. Cirstea and C. Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001.

[6] H. Cirstea, C. Kirchner, and L. Liquori. Rewriting calculus with(out) types. In F. Gadducci and U. Montanari, editors, *Proceedings of the fourth workshop on rewriting logic and applications*, Pisa (Italy), Sept. 2002. Electronic Notes in Theoretical Computer Science.

[7] F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. In J. Tiuryn, editor, *Proc. Conf. on Foundations of Software Science and Computation Structures (FOSSACS 2000)*, volume 1784 of *Lecture Notes in Computer Science*, pages 98–113, 2000.

[8] M. Fernández and L. Khalil. Interaction nets with McCarthy's amb: Properties and applications. *Nordic Journal of Computing*, 10(2):134–162, 2003.

[9] M. Fernández and I. Mackie. Interaction nets and term rewriting systems. *Theoretical Computer Science*, 190(1):3–39, January 1998.

[10] M. Fernández and I. Mackie. A calculus for interaction nets. In G. Nadathur, editor, *Proceedings of the International Conference on Principles and Practice*

of Declarative Programming (PPDP'99), volume 1702 of *Lecture Notes in Computer Science*, pages 170–187. Springer-Verlag, September 1999.

[11] M. Fernández, I. Mackie, and J. S. Pinto. A higher-order calculus for graph transformation. In D. Plump, editor, *Proc. of the Int. Workshop on Term Graph Rewriting (TERMGRAPH 2002)*, ENTCS Vol. 72 (1), Barcelona, 2002.

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

[13] C. B. Jay. The pattern calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(6):911–937, November 2004.

[14] O. Jensen and R. Milner. Bigraphs and mobile processes (revised). Technical Report 580, Computer Laboratory, University of Cambridge, 2004.

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

[16] I. Mackie. Efficient λ-evaluation with interaction nets. In V. van Oostrom, editor, *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA'04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 155–169. Springer-Verlag, June 2004.

[17] V. Oostrom. Lambda calculus with patterns. Technical Report IR 228, Vrije Universiteit, Amsterdam, November 1990.

[18] S. L. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice Hall International, 1987.

[19] 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)*, volume 1784 of *Lecture Notes in Computer Science*, pages 267–282. Springer-Verlag, 2000.

[20] J. S. Pinto. Weak reduction and garbage collection in interaction nets. In *Proceedings of the 3rd International Workshop on Reduction Strategies in Rewriting and Programming*, Valencia, Spain, 2003.