Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 203 (2008) 79–92
www.elsevier.com/locate/entcs

Interaction Nets With Nested Pattern Matching

Abubakar Hassan[a,1] and Shinya Sato[b,2]

[a] *Department of Computer Science*
King's College London
Strand, London WC2R 2LS, UK

[b] *Faculty of Econoinformatics*
Himeji Dokkyo University
7-2-1 Kamiohno, Himeji-shi, Hyogo 670-8524, JAPAN

Abstract

Reduction rules in Interaction Nets are constrained to pattern match exactly one argument at a time. Consequently, a programmer has to introduce auxiliary rules to perform more sophisticated matches. We propose an extension of Interaction Nets which facilitates nested pattern matching on interaction rules. We then define a practical compilation scheme from extended rules to pure interaction rules. We achieve a system that provides convenient ways to express Interaction Net programs without defining auxiliary rules.

Keywords: Interaction nets, pattern matching, programming language design.

1 Introduction

Interaction Nets [5] can be considered as a graphical–or visual–programming language. Programs are expressed as graphs, and computation is graph reduction. From another perspective, Interaction Nets are also a low level implementation language: we can define systems of Interaction Nets that are instructions for the target of compilation schemes of other programming languages. For instance, Interaction Nets have been used for the implementation of optimal reduction [4,6] and other efficient implementations of the λ-calculus [8]. In addition, there has been various implementations of Interaction Nets [7,9]. Despite that we can already program in Interaction Nets (they are Turing complete), they still remain far from being used as a programming language. Drawing an analogy with functional programming, we

[1] abubakar.hassan@kcl.ac.uk
[2] shinya@himeji-du.ac.jp

doi:10.1016/j.entcs.2008.03.035

only have the pure λ-calculus without syntactic sugar, constants, data-structures, etc.

In this paper we take a step towards developing a richer language based on Interaction Nets. Interaction Nets have a very primitive notion of pattern matching since only two agents can interact at a time. Consequently, many auxiliary agents and rules are needed to implement more sophisticated matches. These auxiliaries are implementation details and should be generated automatically other than by the programmer. To achieve this, we extend Interaction Nets to allow rules with nested patterns to be defined. We then give a compilation scheme from extended to ordinary interaction rules.

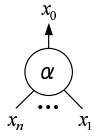
There has been several works that extend Interaction Nets in some way (see Section 6.2). Sinot and Mackie's Macros for Interaction Nets [10] are quite close to what we present in this paper. They allow pattern matching on more than one argument by relaxing the restriction of one principal port per agent. The main difference with our work is that their system does not allow nested pattern matching. Our system facilitates nested/deep pattern matching of agents.

The rest of this paper is organised as follows: In the next section we give a brief introduction to Interaction Nets. In Section 3 we motivate our work through an example. We give the proposed extensions in Section 4, followed by the compilation schemes in Section 5. In Section 6 we discuss some implementation issues. Finally, we conclude the paper in Section 7

2 Interaction Nets

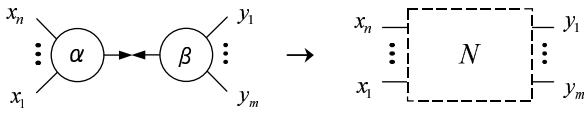
We review the basic notions of Interaction Nets. See [5] for a more detailed presentation. Interaction Nets are specified by the following data:

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



  We use the textual notation $x_0 - \alpha(x_1, ..., x_n)$ to represent an agent α where x_0 is the principal port and $x_1, ..., x_n$ are its auxiliary ports.
- A *net* built on Σ is an undirected graph with agents at the vertices. The edges of the net connect agents together at the ports such that there is only one edge at every port. A port which is not connected is called a *free port*.
- Two agents $(\alpha, \beta) \in \Sigma \times \Sigma$ connected via their principal ports form an *active pair* (analogous to a redex). An interaction rule $((\alpha, \beta) \rightarrow N) \in \mathcal{R}$ replaces the pair (α, β) by the net N. All the free ports are preserved during reduction, and there is at most one rule for each pair of agents. The following diagram illustrates the

idea, where N is any net built from Σ.



We represent this rule textually as $\alpha(x_1, ..., x_n) \bowtie \beta(y_1, ..., y_m) \rightarrow N$ or $\beta(y_1, ..., y_m) \bowtie \alpha(x_1, ..., x_n) \rightarrow N$. It does not matter which active agent we choose to write first. We use the notation $N_1 \Rightarrow N_2$ for the one step reduction and $\Rightarrow^*$ for its transitive and reflexive closure.

Interaction Nets have the following property [5]:

- **Strong Confluence**: Let N be a net. If $N \Rightarrow N_1$ and $N \Rightarrow N_2$ with $N_1 \neq N_2$, then there is a net N_3 such that $N_1 \Rightarrow N_3$ and $N_2 \Rightarrow N_3$.

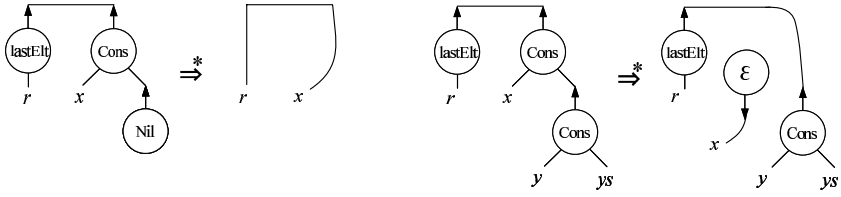
3 Motivations

In this section, we motivate our work by investigating how we can translate a function with pattern matching into Interaction Nets.

Example 3.1 Our example is the following definition of a function that returns the last element of a list:

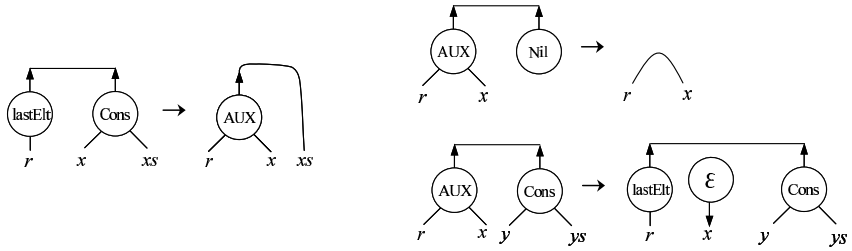
```
fun lastElt [x] = x
  | lastElt (x::y::ys) = lastElt (y::ys);
```

If we consider a functional programming language as an orthogonal term rewriting system, we can translate programs into Interaction Nets [3]. In this way, if we take both the name of the function and the first argument as agents, we can represent the above function as interaction rules:



However, these rules are not valid in Interaction Nets as the left hand side (LHS) of a rule must be a net with exactly two agents (active pair).

Therefore, to encode this example in interaction nets, we have to introduce auxiliary agents and rules:



This set of rules will compute and return the last element of a list. We argue that the introduction of the auxiliary agents to the system is not satisfactory from a programmer's perspective. Programmers' want to write simpler programs rather than more complicated ones. To solve this problem, we extend the definition of rules to facilitate nested pattern matching.

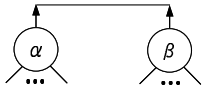
4 Interaction rules for nested patterns (INP)

4.1 An extension of the definition of interaction rules

In this section we present our framework INP that extends ordinary interaction rules (ORN) so that we can perform rewritings between nested agents. The main difference from ORN is that we allow the left hand side of a rule to contain more than two agents. The definition of agents and nets remain the same as for ORN.

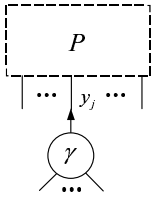
Definition 4.1 A *nested active pair* P is inductively defined as follows:

Base: Every active pair in ORN is a nested active pair



represented textually as: $\langle \alpha(x_1, ..., x_n) \bowtie \beta(y_1, ..., y_m) \rangle$.

Step: A net obtained as a result of connecting the principal port of some agent to a free port in a nested active pair P is also a nested active pair.



We represent this nested active pair textually as $\langle P,\ y_j - \gamma(z_1, ..., z_l) \rangle$.

Definition 4.2 An interaction rule in INP is given by $P \to N$ where P is a nested active pair. All the free ports are preserved during reduction, and there is at most one rule with P as a LHS in any given system.

Proposition 4.3 *ORN* $\subset$ *INP.*

Proof. All rules $P \to N$ where P contains just two agents (active pair) are valid

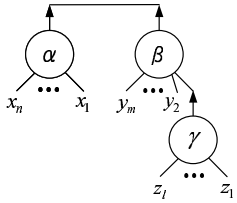
ORN rules. These active pairs fall into the base definition of nested active pairs. □

We aim to extend ORN in a conservative way and retain the property of strong confluence. For this purpose, we introduce a condition that restricts the formation of the set of interaction rules in INP.

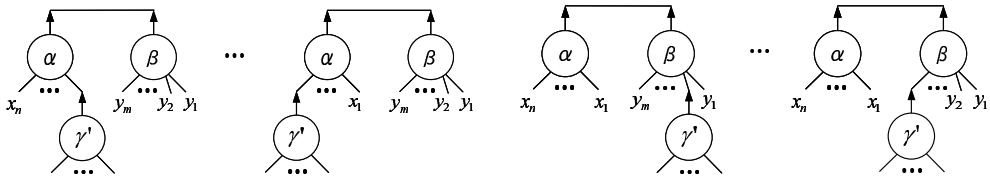
Definition 4.4 A set of nested active pairs $\mathcal{P}$ is *sequential* if and only if, when $\langle P,\ y_j - \gamma(z_1, ..., z_l)\rangle \in \mathcal{P}$, then

- for the nested pair P, $P \in \mathcal{P}$ and,
- for all free ports y in P except the y_j and for all agents α, $\langle P,\ y - \alpha(w_1, ..., w_n)\rangle \notin \mathcal{P}$.

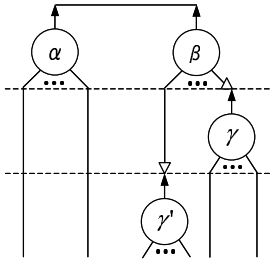
As an example, consider the following nested active pair P in a sequential set $\mathcal{P}$:



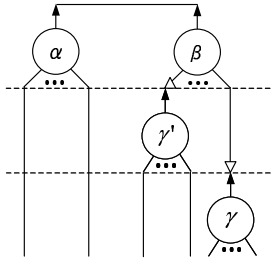
represented textually as $\langle \alpha(x_1, ..., x_n) \bowtie \beta(y_1, ..., y_m),\ y_1 - \gamma(z_1, ..., z_l)\rangle$. Then we can not have any other nested active pair (α, β) such that the port y_1 is free. Thus, the following definitions violate the condition of the set $\mathcal{P}$:



For clarity, we draw lines and triangles on auxiliary ports that connect to nested agents. As an example, we represent a nested active pair $\langle P,\ y_m - \gamma'(w_1, ..., w_k)\rangle$ graphically as follows:



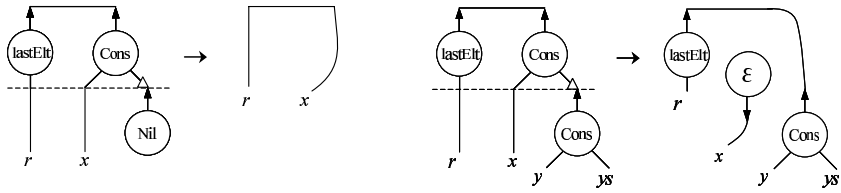
Note that this nested active pair can belong to the set $\mathcal{P}$ because $P \in \mathcal{P}$. On the other hand, the nested active pair below can not belong to the same set $\mathcal{P}$ since P is already in $\mathcal{P}$ and the condition of Definition 4.4 requires that
$\langle \alpha(x_1, ..., x_n) \bowtie \beta(y_1, ..., y_m),\ y_m - \gamma'(w_1, ..., w_k)\rangle \notin \mathcal{P}$.



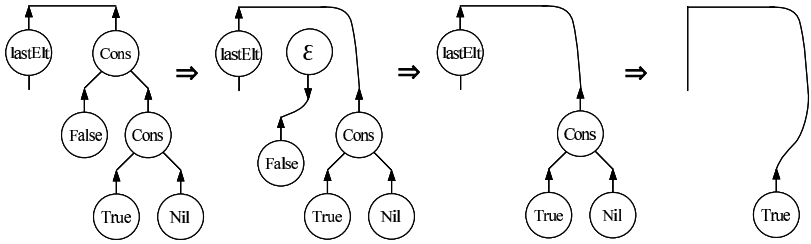
Definition 4.5 A set of rules $\mathcal{R}$ in INP is *well-formed* if and only if,

- there is a sequential set which contains every LHS of rules in $\mathcal{R}$,
- for every rule $P \to N$ in $\mathcal{R}$, there is no interaction rule $P' \to N'$ in $\mathcal{R}$ such that P' is a subnet of P.

Example 4.6 The rule set in Example 3.1 is well-formed



and the following computation can be performed:

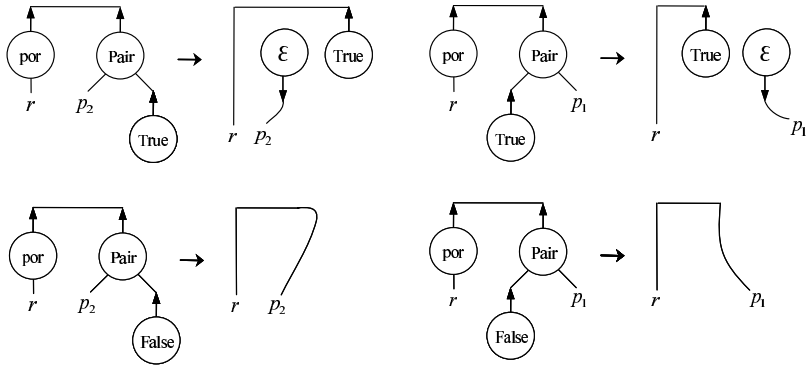


In the above example, the rewriting is strongly confluent because there is no *critical pair*. We loose this property if there are more than two rules that can be applied to the same pattern.

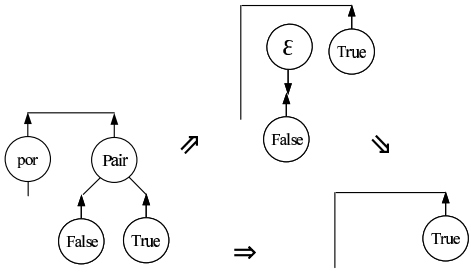
Example 4.7 We can encode the following definition of the parallel-or function `por`:

$$\begin{aligned}
\texttt{por}(\texttt{True}, y) &= \texttt{True} \\
\texttt{por}(y, \texttt{True}) &= \texttt{True} \\
\texttt{por}(\texttt{False}, y) &= y \\
\texttt{por}(y, \texttt{False}) &= y
\end{aligned}$$

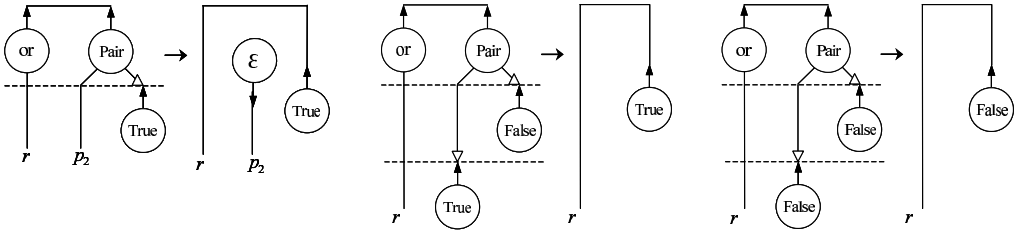
as a set of INP rules:



However, this is not a well-formed set of rules because there is no sequential set which contains both $\langle \texttt{por}(x) \bowtie \texttt{Pair}(y_1, y_2),\ y_1 - \texttt{True} \rangle$ and $\langle \texttt{por}(x) \bowtie \texttt{Pair}(y_1, y_2),\ y_2 - \texttt{True} \rangle$. Therefore, the reduction is not strongly confluent (but still confluent in this example).



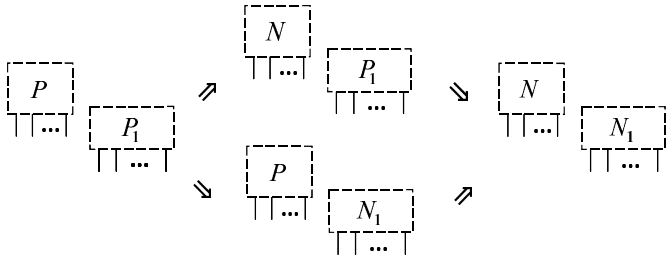
On the other hand, the following rule set of the *or* function is well-defined:



Proposition 4.8 (Strong Confluence) *If a given rule set $\mathcal{R}$ in INP is well-formed, then the reduction in $\mathcal{R}$ is strongly confluent.*

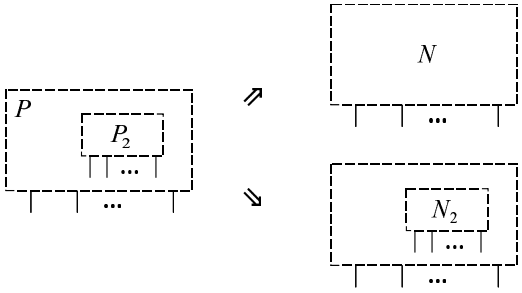
Proof. Assume that $P \to N \in \mathcal{R}$. There are two cases where critical pairs can arise for a net which contains P:

case 1: there is no overlap between rules. We assume that there is a rule $P_1 \to N_1 \in \mathcal{R}$ where P_1 does not overlap with P. In this case, the reduction is strongly confluent:



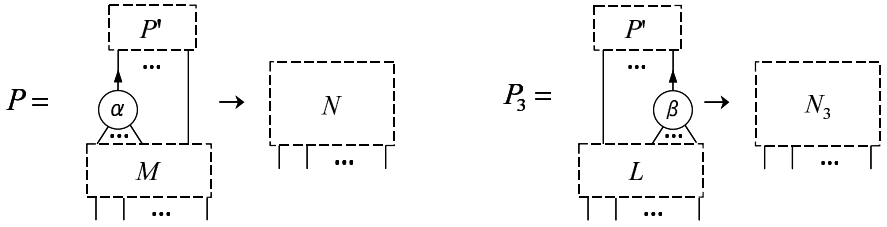
case 2: there are overlaps between rules.

case 2.1: We assume that there is a rule $P_2 \to N_2 \in \mathcal{R}$ where P_2 is a subnet of P.



This case can not arise if $\mathcal{R}$ is well formed. Therefore $P_2 \to N_2 \notin \mathcal{R}$

case 2.2: We assume that there is a rule $P_3 \to N_3 \in \mathcal{R}$ where P_3 contains the subnet of P.



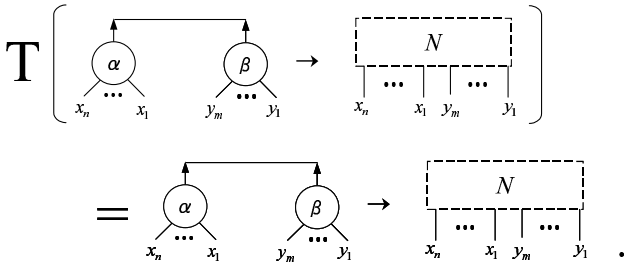
There is no sequential set which contains both P and P_3, therefore $P_3 \to N_3 \notin \mathcal{R}$.

□

5 Translation

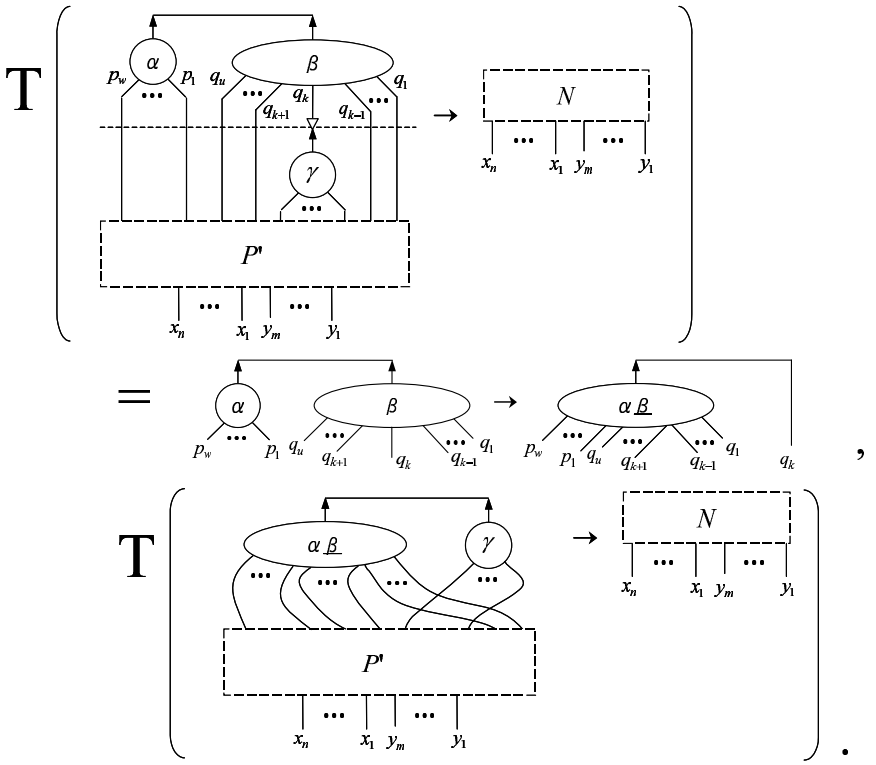
In this section, we define the translation function **T** from interaction rules with nested active pairs to interaction rules with only active pairs:

- If a nested active agent contains an active pair of just two agents, then the translation is the identity:

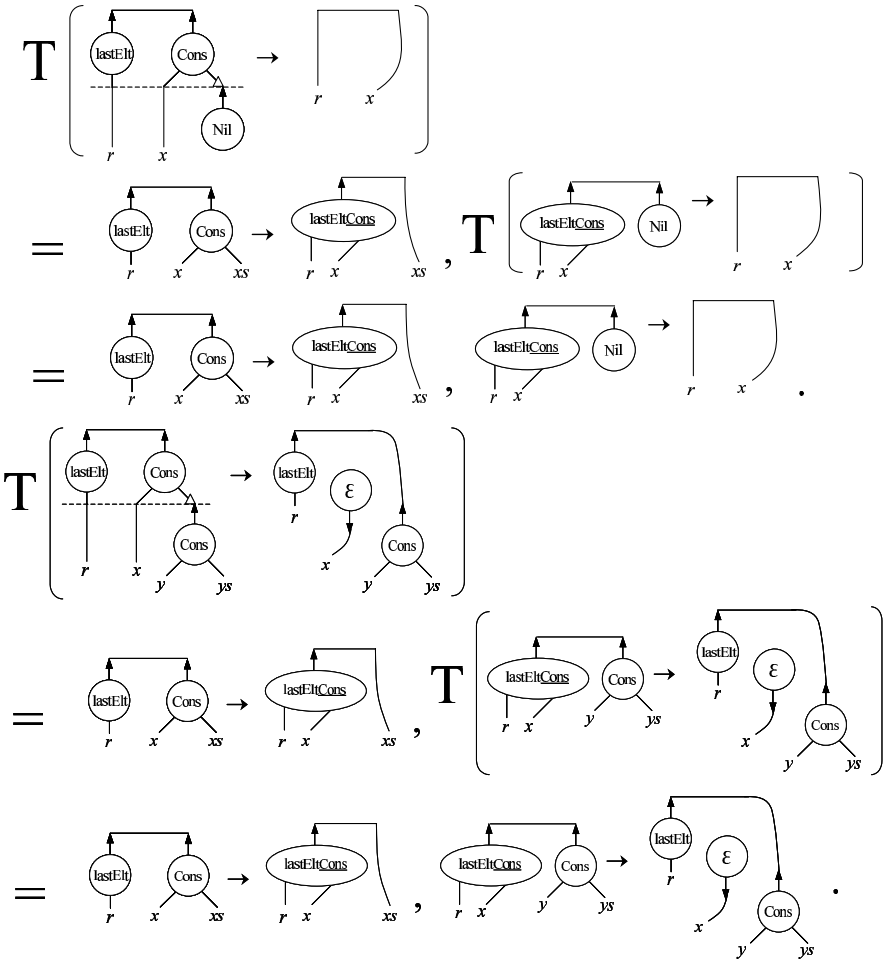


- The translation of a rule $P \to N$ where $P = \langle \alpha(p_1, ..., p_w) \bowtie \beta(q_1, .., q_k, .., q_u),\ q_k - \gamma(z_1, ..., z_l),\ \mathbf{a} \rangle$ where $\mathbf{a}$ is a sequence of agents, generates the following rules:
  - $\alpha(p_1, ..., p_w) \bowtie \beta(q_1, .., q_k, .., q_u) \to q_k - \alpha\underline{\beta}(q_1, .., q_{k-1}, q_{k+1}, .., q_u, p_1, ..., p_w)$ where $\alpha\underline{\beta}$ is a new agent named from a concatenation of the LHS nested active pair agents. Since q_k is connected to the principal port of γ, an active pair $(\alpha\underline{\beta}, \gamma)$ will be formed.
  - $\langle \alpha\underline{\beta}(q_1, .., q_{k-1}, q_{k+1}, .., q_u, p_1, ..., p_w) \bowtie \gamma(z_1, ..., z_l),\ \mathbf{a} \rangle \to N$. This rule is recursively translated to obtain a rule with just an active pair.

  Graphically, this translation is given by:



Example 5.1 We give the translation of the function in Example 3.1 that computes and returns the last element of a list.



Lemma 5.2 *Let $\mathcal{R}$ be a well-formed rule set in* INP *and $R_1, R_2 \in \mathcal{R}$. Then, a rule set $\mathbf{T}[R_1] \cup \mathbf{T}[R_2]$ contains no rule such that $P \to N_1$ and $P \to N_2$ where $N_1 \neq N_2$.*

Proof. Let $R_1 = P_1 \to M_1$ and $R_2 = P_2 \to M_2$.

case 1: the active pairs in P_1 and P_2 are different. Let $P_1 = \langle \alpha_1(\mathbf{x}_1) \bowtie \beta_1(\mathbf{y}_1), \mathbf{a_1} \rangle$ and $P_2 = \langle \alpha_2(\mathbf{x}_2) \bowtie \beta_2(\mathbf{y}_2), \mathbf{a_2} \rangle$ where $\mathbf{x}_1, \mathbf{y}_1, \mathbf{x}_2, \mathbf{y}_2$ are sequences of auxiliary ports, $\mathbf{a_1}, \mathbf{a_2}$ are sequences of agents and $\alpha_1 \neq \alpha_2$ or $\beta_1 \neq \beta_2$.

In this case, distinct names $\alpha_1\underline{\beta_1}, \alpha_2\underline{\beta_2}$ are introduced by $\mathbf{T}$ for those active pairs respectively. Therefore, every LHS of the rules generated by recursively applying $\mathbf{T}$ also have distinct active pairs. Thus, there is no rule such that it's LHS occurs as a LHS of another rule.

case 2: the active pairs in P_1 and P_2 are the same. Because both P_1 and P_2 belong to the same sequential set, then P_1 and P_2 have a same sequence of agents succeeding from the active pair. Let $P_1 = \langle \alpha(\mathbf{x}) \bowtie \beta(\mathbf{y}), \mathbf{a}, \mathbf{a_1} \rangle$ and $P_2 = \langle \alpha(\mathbf{x}) \bowtie \beta(\mathbf{y}), \mathbf{a}, \mathbf{a_2} \rangle$ where $\mathbf{x}, \mathbf{y}$ are sequences of auxiliary ports, $\mathbf{a}, \mathbf{a_1}, \mathbf{a_2}$ are sequences of agents and $\mathbf{a_1} \neq \mathbf{a_2}$.

For the same parts $\alpha(\mathbf{x}) \bowtie \beta(\mathbf{y})$ and $\mathbf{a}$, same rules are obtained by using $\mathbf{T}$.

For the remaining agents, it turns out that there is no rule such that $P \to M_1$ and $P \to M_2$ by applying case 1.

□

Proposition 5.3 *Let $\mathcal{R}$ be a well-formed rule set in* INP. *The set* $\bigcup \mathbf{T}[R]$ *where $R \in \mathcal{R}$ is a correct rule set in* ORN.

Proof. From the definition of $\mathbf{T}$, it is clear that every LHS of rules obtained by using $\mathbf{T}$ contains only an active pair. Moreover, by Lemma 5.2, there is no rule $P \to N_1$ and $P \to N_2$ in the resulting rule set. □

Proposition 5.4 (Conservativity) *Let $\mathcal{R}$ be a well-formed set of rules in* INP. *If $P \to N \in \mathcal{R}$, then $P \Rightarrow^* N$ by using the rules obtained by the translation $\mathbf{T}[P \to N]$.*

Proof. If P is just an active pair, then we can perform $P \Rightarrow N$ because $\mathbf{T}[P \to N] = P \to N$.

If $P = \langle \alpha(\mathbf{x}) \bowtie \beta(\mathbf{y}, y),\ y - \gamma(\mathbf{z}),\ \mathbf{a} \rangle$ where $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are sequences of auxiliary ports and $\mathbf{a}$ is a sequence of agents, then

$$\mathbf{T}[P \to N] = \alpha(\mathbf{x}) \bowtie \beta(\mathbf{y}, y) \to \alpha\underline{\beta}(\mathbf{x}, \mathbf{y}) - y,\ \mathbf{T}[\,\langle \alpha\underline{\beta}(\mathbf{x}, \mathbf{y}) \bowtie \gamma(\mathbf{z}),\ \mathbf{a} \rangle \to N].$$

By using the first rule,

$$\alpha(\mathbf{x}) \bowtie \beta(\mathbf{y}, y),\ y - \gamma(\mathbf{z}),\ \mathbf{a} \ \Rightarrow\ \alpha\underline{\beta}(\mathbf{x}, \mathbf{y}) - \gamma(\mathbf{z}),\ \mathbf{a}.$$

Applying recursively this operation to the rule $\langle \alpha\underline{\beta}(\mathbf{x}, \mathbf{y}) \bowtie \gamma(\mathbf{z}),\ \mathbf{a} \rangle \to N$ and the nested agent pair $\alpha\underline{\beta}(\mathbf{x}, \mathbf{y}) \bowtie \gamma(\mathbf{z})$, we will perform $P \Rightarrow^* N$. □

6 Discussion

6.1 Implementation

In this section we briefly discuss implementation issues of INP. There are two approaches to implement INP. One is to translate into ORN rules then use existing evaluators of Interaction Nets. The other is to implement them directly. Here we look at this second option, and show how the main tasks of performing computation in this framwork can be achieved. Our aim here is to show that a direct implemention of INP can be done quite easily. We describe a simple method of achieving this.

The main tasks of an Interaction Net evaluator are to locate the next active pair to reduce, find the matching rule, and apply it to the active pair.

Locating the next active pair can be done locally during rewrite; while rewiring the ports, we check if an active pair is formed then push it into a stack. Reduction will then pop the active pairs from the stack and find the matching rule to apply.

We can store rules in a hash table with a key formed from an ordered concatenation of the (LHS) active pair names. Since INP rules can have more than one active pair of the same agents, we maintain a list such that each key maps onto a

list of rules that share the same active pair names. We iterate through the list to find a rule that matches the structure of the active pair to be reduced.

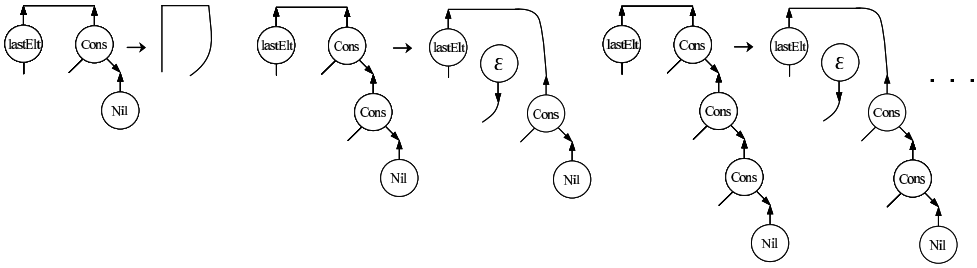
Although ORN will find the matching rule in constant time (each key will only map to one rule) the total number of interactions I performed in ORN: $I(\text{ORN}) > I(\text{INP})$ for a system with nested agents, and $I(\text{ORN}) = I(\text{INP})$ if there is no nested agents. This comes from the fact that ORN introduces extra auxiliary rules for pattern matching.

If we define the cost of computation to be the number of interactions performed, then INP provides an efficient model. However, without empirical studies we are not able to say which system is efficient in terms of execution speed.

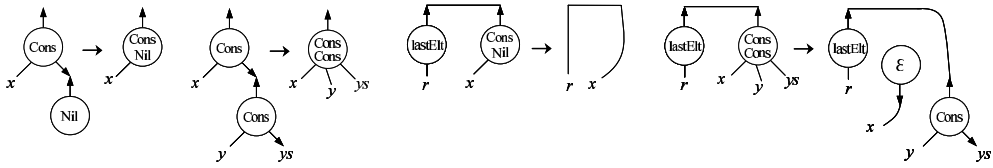
6.2 Related Works

In this section, we discuss other approaches to nested pattern matching by using methods that have been proposed as extensions of Interaction Nets.

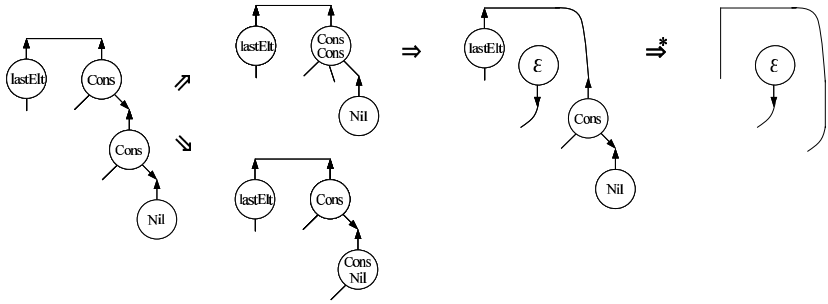
Pattern matching on more than one argument: Sinot and Mackie [10] introduced *Macros* for Interaction Nets and they allow pattern matching on more than one argument by relaxing the restriction of one principal port per agent. Their system requires all principal ports of an agent in the LHS net of a rule to be connected to principal ports of other agents for the purpose of holding the property of strong confluence. Therefore, this system is useful as a conservative extension. However, we can hardly encode the function `lastElt` as it requires nested pattern matching. This is because in the case that the `Cons` agent has two principal ports, we have to write all cases as follows:



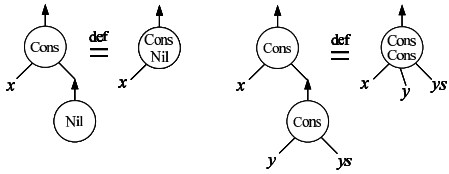
Alexiev's *interaction nets with multiple principal ports* (IMNPP) [1] is also useful for this purpose because this system also allows more than one principal port per agent. However, interactions are still performed only on an active pair. Therefore, in the case of nested pattern matching, we have to introduce auxiliary agents and rules as in Section 3. As another solution, we can introduce rules between `Cons` and `Nil`:



These cause computation between the list structures even if it is not needed.



Computation for nets: Bechet [2] proposed computation for nets on interaction rules as *abbreviations*, where nets are captured as an agent and reductions of the agent are realized by the rules corresponding to the computation of the net. As an example of applying this method to nested pattern matching, we consider our example function `lastElt`. One solution is to define the agent `lastElt` by using other agents that have already been defined. It is not simple to find a good combination with those agents. As another solution, we introduce abbreviations for list structures:



However, we have to define rules between `lastElt` and `Cons` for the case that those abbreviations are unfolded, therefore we have to introduce auxiliary agents in the end.

7 Conclusion

We have shown how to extend Interaction Nets to facilitate nested pattern matching without introducing auxiliary rules. This provides a convenient and a more natural way of expressing Interaction Net programs. We see this extension as a positive step towards using Interaction Nets as a practical programming language.

References

[1] Alexiev, V., "Non-deterministic interaction nets," Ph.D. thesis (1999), adviser-Jia You.

[2] Bechet, D., *Partial evaluation of interaction nets*, in: M. Billaud, P. Castéran, M. M. Corsini, K. Musumbu and A. Rauzyand, editors, *Proceedings of the Second Workshop on Static Analysis WSA'92*, Bigre Journal **81-82**, 1992, pp. 331–338.

[3] Fernández, M. and I. Mackie, *From term rewriting to generalised interaction nets*, in: H. Kuchen and S. D. Swierstra, editors, *Proceedings of the 8th International Symposium on Programming Languages, Implementations, Logics and Programs (PLILP'96)*, Lecture Notes in Computer Science **1140** (1996), pp. 319–333.

[4] Gonthier, G., 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)* (1992), pp. 15–26.

[5] Lafont, Y., *Interaction nets*, in: *Seventeenth Annual Symposium on Principles of Programming Languages* (1990), pp. 95–108.

[6] Lamping, J., *An algorithm for optimal lambda calculus reduction*, in: *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)* (1990), pp. 16–30.

[7] Lippi, S., *in*^{2} *: A graphical interpreter for interaction nets*, in: S. Tison, editor, *Rewriting Techniques and Applications (RTA'02)*, Lecture Notes in Computer Science **2378** (2002), pp. 380–386.

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

[9] Pinto, J. S., *Parallel evaluation of interaction nets with mpine.*, in: A. Middeldorp, editor, *RTA*, Lecture Notes in Computer Science **2051** (2001), pp. 353–356.

[10] Sinot, F.-R. and I. Mackie, *Macros for interaction nets: A conservative extension of interaction nets.*, Electr. Notes Theor. Comput. Sci. **127** (2005), pp. 153–169.