ELSEVIER Theoretical Computer Science 297 (2003) 157–181
Theoretical Computer Science
www.elsevier.com/locate/tcs

Operational equivalence for interaction nets

Maribel Fernández[a,*], Ian Mackie[b]

[a]*DI-LIENS (CNRS UMR 8548), École Normale Supérieure, 45 Rue d'Ulm, 75005 Paris, France*
[b]*CNRS-LIX (UMR 7650), École Polytechnique, 91128 Palaiseau Cedex, France*

Abstract

The notion of contextual (or operational) equivalence is fundamental in the theory of programming languages. By setting up a notion of bisimilarity, and showing that it coincides with contextual equivalence, one obtains a simple coinductive proof technique for showing that two programs are equivalent in all contexts. In this paper we apply these (now standard) techniques to interactions nets, a graphical programming language characterized by local reduction. This work generalizes previous studies of operational equivalence in typed interaction nets since it can be applied to untyped systems, thus all systems of interaction nets are captured.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Interaction nets; Bisimulation equivalence; Operational semantics

1. Introduction

Interaction nets, introduced by Lafont [11], are graph rewriting systems which generalize multiplicative proof nets of linear logic [6]. One interesting aspect of interaction nets is that they can be regarded both as a high-level programming language, or as a low-level implementation (machine) language. An interaction net program consists of a net (a graph built from a set of agents and wires) and a set of interaction rules that describe the way in which the net will be reduced. We are interested in the problem of defining an equivalence relation between programs that compute the same results, or in other words, that behave in the same way, in all contexts. In that case, one program can be replaced by the other, for example for efficiency reasons, without altering the operational semantics of the system. To define this equivalence relation we first need to develop an operational theory of interaction nets specifying in a precise way how programs are executed (i.e. a strategy of evaluation of nets and a notion of value).

* Corresponding author.
E-mail addresses: maribel.fernandez@ens.fr (M. Fernández), mackie@lix.polytechnique.fr (I. Mackie).

In [3] we proposed a way of adapting the coinductive techniques, used successfully for the functional and object-oriented programming paradigms, to give a notion of operational equivalence for the interaction paradigm. The language of interaction nets that was studied focussed on the notion of type, which is natural if interaction nets are seen as a programming paradigm. In particular, types allow us to distinguish values from programs. However, some applications of interaction nets do not fit into the typed framework in a natural way. For instance, systems based on the interaction combinators [12], or the systems of interaction used for the encoding of the λ-calculus [14], are untyped. Although it is possible to develop a type system for them [10], a natural approach would be to develop an operational theory of interaction nets that does not rely on the notion of types. The same remark can be made in the case of functional languages based on the λ-calculus, where two different approaches can be found in the literature, depending on whether the calculus is typed (see for instance [16]) or untyped [1].

In this paper we present an operational theory for *untyped* interaction nets, including a notion of contextual equivalence (also called operational or observational equivalence) and an associated bisimilarity relation which permits the use of coinductive techniques in the proofs of operational equivalence. To express these notions we use the textual calculus of interaction nets presented in [4] instead of the graphical language. This allows us to give a concise and formal presentation, leaving the use of diagrams for the examples and intuitive explanations.

A system of interaction nets is a user-defined language, in the same spirit as systems based on term-rewriting. Our results are applicable to *any* system of interaction nets; we are not restricted to one specific set of rules. If the system is typed, the information provided by types can be used to obtain a more refined equivalence relation between nets, recovering the results of [3]. We remark that interaction nets are also used as an object language for the coding of other rewriting systems. The λ-calculus is perhaps the most studied example of this (see e.g. [7,14]). Our results are also applicable here, so we have a proof technique for optimizations of such systems.

Related work. Surprisingly, there is very little in the literature about equivalence of interaction nets. Only ad hoc techniques, or very restrictive notions like having the same normal form, seem to have been used. We mention two works where equivalence arises: Lafont [12] studies various permutations of agents which are equivalent in a given path semantics for the interaction combinators; however, these are only valid for the system of combinators and do not cover the contextual equivalence. Bechet [2], in the study of partial evaluation of interaction nets, has studied simple cases of behavioural equivalence, in particular, when collections of agents behave like the identity (a connecting edge). This notion of equivalence is used mostly for optimizing specific rules.

The methods that we use here are inspired by those of Gordon [8], Howe [9], and Pitts [16] for bisimulation. In particular, we follow the abstract approach of Howe in this paper, generalizing the ideas to interaction nets.

Overview. The rest of this paper is structured as follows. In the next section we set up the definition of interaction nets. In Section 3 we define our evaluation strategy.

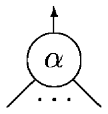
Section 4 sets up the notion of bisimilarity. In Section 5 we give some examples of use of this relation. In Section 6 we formalize the notion of contextual equivalence, and Section 7 shows that this coincides with bisimilarity. Finally, we conclude the paper in Section 8. This paper is a revised and extended version of a paper presented at LATIN'00 [5].

2. Background

Interaction nets were presented by Lafont [11] as a graphical programming paradigm where computation is local and strongly confluent, and as a consequence reduction can be easily parallelized (taking place at several places at the same time) [15]. We begin by recalling a textual calculus of interaction nets that we will use for the rest of the paper; we refer the reader to [4] for a more detailed description and examples. We remark that all that we have to say can be recast in the graphical framework of interaction nets, but the calculus allows us to write definitions and proofs in a more concise and formal way. Moreover, the calculus brings out some of the finer details of interaction nets which are implicit in the graphical framework.

A program consists of a set of rules and a net, which in the calculus is represented by a configuration. The main components of configurations are terms and equations, which are defined from agents and variables (also called names).

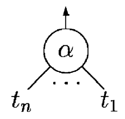
Agents. Let Σ be a set of symbols, ranged over by $\alpha, \beta, \dots,$ each with a given *arity* $ar : \Sigma \to \mathbb{N}$. An occurrence of a symbol will be called an *agent*. Every agent has one principal port and a number of auxiliary ports equal to its arity. Graphically, the principal port of an agent is indicated by an arrow.



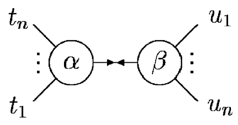
The set of symbols can be partitioned into a set of *constructors* and a set of *destructors*: $\Sigma = \mathscr{C} \cup \mathscr{D}$. For instance, if we are programming arithmetic operations on natural numbers encoded by the agents Z (zero) and S (successor), it is standard to consider Z and S as constructors, and the agents encoding the operations of addition, multiplication, etc. as destructors.

Names. Let N be a set of names, ranged over by x, y, z, etc. Names denote ports of agents in the graphical representation. We assume that N and Σ are disjoint.

Terms. A term is built on Σ and N by the grammar: $t ::= x \mid \alpha(t_1, \dots, t_n)$, where $x \in N$, $\alpha \in \Sigma$, $ar(\alpha) = n$ and $t_1, \dots, t_n$ are terms, with the restriction that each name may appear at most twice. If $n = 0$, then we omit the parentheses. $\mathscr{N}(t)$ denotes the set of names occurring in t. If a name occurs twice in a term, we say that it is *bound*, otherwise it is *free*. Since free names occur exactly once, we say that terms are *linear*. We write $\vec{t}$ for a list of terms $t_1, \dots, t_n$. Graphically, a term of the form $\alpha(\vec{t})$ can be seen as a tree with the principal port of α at the root, and where the terms $t_1,\dots,t_n$ are the subtrees connected to the auxiliary ports of α.



A free variable represents a free port, and a bound variable represents a wire connecting two auxiliary ports (this is why a variable can occur at most twice in a term).
Terms cannot represent all nets since in a tree there are no connections between two principal ports. For this we introduce equations.
Equations. If t and u are terms, then the (unordered) pair $t = u$ is an *equation*. Δ, $\Theta,\dots$ will be used to range over multisets of equations. Examples of equations include: $x = \alpha(\vec{t})$, $x = y$, $\alpha(\vec{t}) = \beta(\vec{u})$. The graphical representation of an equation $\alpha(\vec{t}) = \beta(\vec{u})$ is a pair of trees connected by their roots (principal ports), as in the diagram below:



Interaction rules. Rules are pairs of terms written as $\alpha(\vec{t}) \bowtie \beta(\vec{u})$, where $(\alpha,\beta) \in \Sigma^2$ is the redex, called the *active pair*, of the rule. All names occur exactly twice in a rule, and there is one rule for each pair of agents. In the graphical framework, interaction rules are oriented pairs of graphs, at the left we have the active pair and at the right the net that will replace it in a reduction step. In the calculus, the terms $\vec{t}$, $\vec{u}$ represent the right-hand side of the graph rewriting rule.

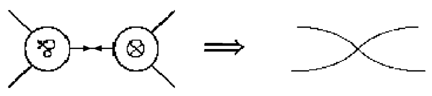
We now have all the machinery that we need to define interaction nets.

Definition 1 (Configurations). A *configuration* is a pair: $c = (\mathcal{R}, \langle \vec{t} \mid \Delta \rangle)$, where $\mathcal{R}$ is a set of rules, $\vec{t}$ a list $t_1,\dots,t_n$ of terms, and Δ a multiset of equations. Each variable occurs at most twice in c. If a name occurs once in c then it is *free*, otherwise it is *bound*. For simplicity we sometimes omit $\mathcal{R}$ when there is no ambiguity. We use c, c' to range over configurations. We call $\vec{t}$ the *head* and Δ the *body* of a configuration.

Intuitively, $(\mathcal{R}, \langle \vec{t} \mid \Delta \rangle)$ represents an interaction net that we evaluate using $\mathcal{R}$; Δ gives the set of active pairs and the renamings of the net. To draw the net represented by a configuration $c = \langle \vec{t} \mid \Delta \rangle$, we simply draw the trees for the terms in c, connect the common variables together, and connect the trees corresponding to the members of an equation together on their roots. The roots of the terms in the head of the configuration and the free names correspond to ports in the *interface* (i.e. the free ports) of the net. Note that the head of the configuration may contain all or just some of the ports in the interface of the net, called *observable*. For this reason, the head is sometimes called the *observable interface* of the configuration.

We work modulo α-equivalence for bound names as usual, but also for free names. Configurations that differ only on the names of the free variables are equivalent, since they represent the same net.

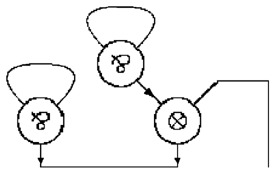
Example 2 (Configurations and rules). The system of interaction representing proof nets of multiplicative linear logic consists of two agents, $\parr$ and $\otimes$ of arity 2, and one interaction rule, represented graphically as



In the calculus this rule is written as: $\parr(x, y) \bowtie \otimes(x, y)$. The configuration

$$\langle c \mid \parr(a,a) = \otimes(\parr(b,b), c)\rangle$$

represents a net with one active pair and only one free port:



The configuration $\langle c \mid \parr(a,a) = \otimes(b,b)\rangle$ represents a net without an interface, containing an active pair. The empty net is represented by $\langle \mid \rangle$, and the configuration $\langle x, x \mid \rangle$ represents a net containing just a wire.

Computation is performed by rewriting equations in configurations using the appropriate rule, as described by the following rewrite system on configurations, where if r is a rule, $\hat{r}$ denotes a fresh generic *instance* of r, that is, a copy of r where we introduce a new set of names:

Indirection. If $x \in \mathscr{N}(u)$, then $x = t, u = v \longrightarrow u[t/x] = v$.
Interaction. If $r \in \mathscr{R}$ and $\hat{r} = \alpha(t'_1, \ldots, t'_n) \bowtie \beta(u'_1, \ldots, u'_m)$, then

$$\begin{aligned}&\alpha(t_1, \ldots, t_n) = \beta(u_1, \ldots, u_m) \rightarrow \\ &\quad t_1 = t'_1, \ldots, t_n = t'_n, u_1 = u'_1, \ldots, u_m = u'_m\end{aligned}$$

Context. If $\Delta \longrightarrow \Delta'$, then $\langle \vec{t} \mid \Gamma, \Delta, \Gamma'\rangle \longrightarrow \langle \vec{t} \mid \Gamma, \Delta', \Gamma'\rangle$.
Collect. If $x \in \mathscr{N}(\vec{t})$, then $\langle \vec{t} \mid x = u, \Delta\rangle \longrightarrow \langle \vec{t}[u/x] \mid \Delta\rangle$.

This rewrite system generates an equational theory, the corresponding equivalence relation is denoted by $c \leftrightarrow^* c'$. The reduction relation $\rightarrow$ is strongly confluent [4].

3. Evaluation

Various strategies of evaluation and notions of value can be defined in the calculus (see [4]). The *values* (or canonical forms) that we use in this paper are called *interface normal forms*. Intuitively, a configuration is in interface normal form if the terms in its head are rooted by agents, or, if a term is a variable, then it will never be changed by reduction. Graphically, an interaction net is in interface normal form when there are principal ports on all of the observable interface, or, if a port x is not principal, then it will never become principal by reduction. In the latter case two situations are possible, either there is a path starting from x and following principal ports leading to a free port y (this is called an *open path*), or there is a *cycle of principal ports*, as depicted in Fig. 1.

Definition 3 (Open path, cycle). A term t_i in the head of a configuration $\langle \vec{t} \mid \Delta \rangle$ is in an open path if $t_i = x$ such that $x \in \mathcal{N}(t_j)$ for some $j \neq i$, or $x \in \mathcal{N}(u)$ for some $y == u \in \Delta$, where $y \in N$ is free. It is in a cycle if $t_i = x$ and $x \in \mathcal{N}(u)$ for some $y == u \in \Delta$ where $y \in \mathcal{N}(u)$.

Definition 4 (Interface normal form). A configuration $(\mathcal{R}, \langle \vec{t} \mid \Delta \rangle)$ is in *interface normal form* (*INF*) if each t_i in $\vec{t}$ is of one of the following *canonical forms*:

- $\alpha(\vec{s})$. e.g.

$$\langle S(x) \mid x == Z, \Delta \rangle$$

- x, where x is in an open path. e.g.

$$\langle x, x \mid \Delta \rangle$$

- x, where x occurs in a cycle of principal ports. e.g.

$$\langle x \mid y == \alpha(\beta(y), x), \Delta \rangle$$

We denote by INF_i the set of configurations where the ith port in the head is canonical, $INF = \bigcap_i INF_i$.

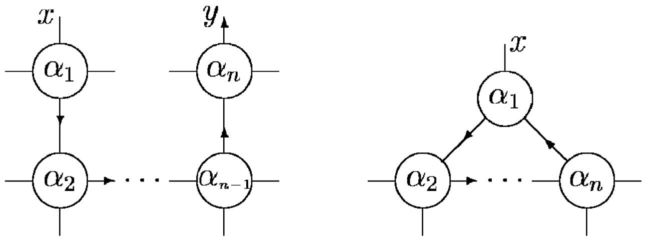


Fig. 1. Open path between x and y (left), and cycle (right).

Example 5. The configuration $\langle x \mid \delta(x,y) = y \rangle$ is in *INF*, since the principal port of δ is in a cycle. The configuration $c = \langle x \mid \delta(x,y) = I(y) \rangle$ is not in *INF*, but reduces to $\langle I(x) \mid \delta(x,y) = I(y) \rangle$ in *INF* using the rule:

$$\delta(I(x), I(y)) \bowtie I(\delta(x,y))$$

Computing interface normal forms suggests that we do the minimum work required to bring principal ports to the interface, applying the rules only when they are needed. This strategy is defined by the set of inference rules:

Axiom:

$$\frac{c \in INF}{c \Downarrow c}\ (\mathsf{Axiom}).$$

Collect:

$$\frac{\langle t_1, \ldots, t, \ldots, t_n \mid \Delta \rangle \Downarrow c}{\langle t_1, \ldots, x, \ldots, t_n \mid x = t, \Delta \rangle \Downarrow c}\ (\mathsf{Collect}).$$

Indirection: if $x \in \mathcal{N}(u)$ and $y \in \mathcal{N}(t, u = v)$

$$\frac{\langle t_1, \ldots, y, \ldots, t_n \mid u[t/x] = v, \Delta \rangle \Downarrow c}{\langle t_1, \ldots, y, \ldots, t_n \mid x = t, u = v, \Delta \rangle \Downarrow c}\ (\mathsf{Indirection}).$$

Interaction: if $x \in \mathcal{N}(\alpha(\vec{t}) = \beta(\vec{u}))$, $r \in \mathcal{R}$, $\hat{r} = \alpha(\vec{t'}) \bowtie \beta(\vec{u'})$

$$\frac{\langle s_1, \ldots, x, \ldots, s_n \mid \overrightarrow{t = t'}, \overrightarrow{u = u'}, \Delta \rangle \Downarrow c}{\langle s_1, \ldots, x, \ldots, s_n \mid \alpha(\vec{t}) = \beta(\vec{u}), \Delta \rangle \Downarrow c}\ (\mathsf{Interaction}).$$

We write $c \Downarrow v$ if v is the interface normal form of c, that is, if the judgement $c \Downarrow v$ can be derived in the previous system. We write $c \Downarrow_i v$ if the rules Indirection and Interaction are only applied when the variable is at position i in the head of the configuration and the axiom is replaced by

$$\frac{c \in INF_i}{c \Downarrow_i c}$$

In other words, if $c \Downarrow_i v$ then the position i in the head of v is canonical.

Example 6 (Natural numbers). Let $\Sigma = \{\mathsf{Z}, \mathsf{S}, \mathsf{A}\}$ with $ar(\mathsf{Z}) = 0$, $ar(\mathsf{S}) = 1$, $ar(\mathsf{A}) = 2$, and $\mathcal{R}$:

$$\begin{array}{lcl} \mathsf{A}(\mathsf{S}(x), y) & \bowtie & \mathsf{S}(\mathsf{A}(x,y)), \\ \mathsf{A}(x,x) & \bowtie & \mathsf{Z}. \end{array}$$

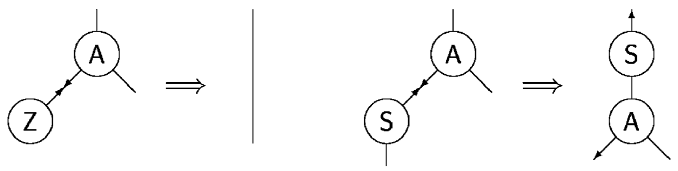
In this system, A represents addition, and Z, S are respectively zero and successor. The net for 1+1 is given by the configuration

$$c = (\mathcal{R}, \langle a \mid \mathsf{A}(a, \mathsf{S}(\mathsf{Z})) = \mathsf{S}(\mathsf{Z}) \rangle).$$

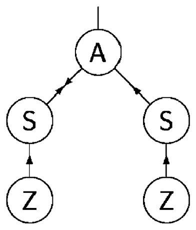
The interface normal form is $v = \langle \mathsf{S}(x') \mid y' = \mathsf{S}(\mathsf{Z}), \mathsf{A}(x', y') = \mathsf{Z} \rangle$, obtained as follows:

$$\dfrac{\dfrac{\dfrac{}{\langle \mathsf{S}(x') \mid y' = \mathsf{S}(\mathsf{Z}), \mathsf{A}(x', y') = \mathsf{Z} \rangle \Downarrow v}\,(\mathsf{Axiom})}{\langle a \mid \mathsf{S}(x') = a, y' = \mathsf{S}(\mathsf{Z}), \mathsf{A}(x', y') = \mathsf{Z} \rangle \Downarrow v}\,(\mathsf{Collect})}{\langle a \mid \mathsf{A}(a, \mathsf{S}(\mathsf{Z})) = \mathsf{S}(\mathsf{Z}) \rangle \Downarrow v}\,(\mathsf{Interaction})$$

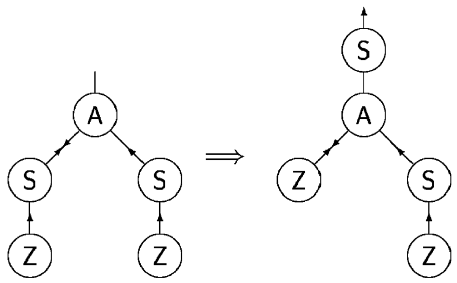
Graphically, the rules for this system are represented as



and the configuration c is depicted as



which reduces to v in INF as follows:



Example 7 (Combinators). The interaction combinators [12] are a system of interaction built from the agents: ε of arity 0, δ and γ of arity 2, together with the following

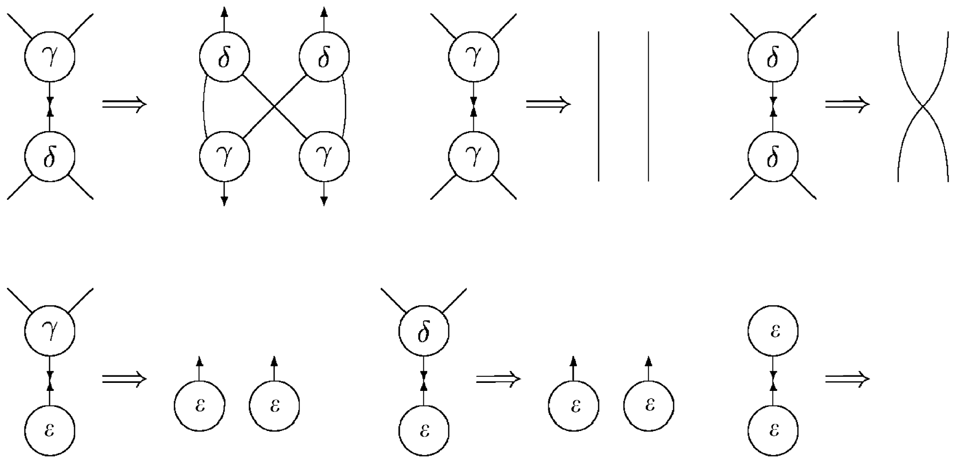


Fig. 2. Interaction combinators: rules.

six rules:

$$\begin{array}{lcl}
\delta(\gamma(a,b),\gamma(c,d)) & \bowtie & \gamma(\delta(a,c),\delta(b,d)),\\
\gamma(x,y) & \bowtie & \gamma(y,x),\\
\delta(x,y) & \bowtie & \delta(x,y),\\
\gamma(\varepsilon,\varepsilon) & \bowtie & \varepsilon,\\
\delta(\varepsilon,\varepsilon) & \bowtie & \varepsilon,\\
\varepsilon & \bowtie & \varepsilon.
\end{array}$$

In Fig. 2 we show the graphical representation of these rules. Although there is an infinite reduction sequence starting from the configuration

$$\langle x \mid \gamma(y,x) = \delta(\varepsilon,y)\rangle$$

it has an interface normal form

$$\langle \delta(a,b) \mid \varepsilon = \gamma(c,a), \delta(c,d) = \gamma(d,b)\rangle.$$

The following properties are proved in [4].

Proposition 8 (Determinacy). *$c \Downarrow c'$ and $c \Downarrow c''$ implies $c' = c''$.*

Proposition 9. *If we consider only the rules Indirection and Collect and the Axiom (excluding Interaction), the system of evaluation is terminating.*

4. Bisimilarity

In functional languages, or simply the λ-calculus, we consider two functions equivalent when we can apply them to the same arguments, and obtain the same result in each case. In other words, we perform some form of experiment on the objects under test, and compare the results: the way that we can interact with a function is to apply it to an argument. For interaction nets, we take this general idea as inspiration. The only way that we can make experiments with a net is to interact with it on a free principal port. Hence, connecting nets on free principal ports is our analogue of applying a function to an argument. We are then able to evaluate the nets, and compare the results. After evaluation, all that we can observe about a net is the fact that some principal ports are at the interface, which is analogous to observing that a λ-term has evaluated to an abstraction.

Depending on the application, we may be interested in comparing only part of the interface of the nets: the *observable ports*. Only the observable ports are available for the experiments, the other free ports are hidden. It only makes sense to compare nets that have the same number of observable ports. Configurations with the same number of terms in the head will be said *comparable*.

To compare two configurations, we evaluate them to interface normal form and compare the heads term by term. It might be that some observable ports do not have a canonical form (the relation $\Downarrow_i$ is undefined at that position: $c\not\Downarrow_i$), or that there is a cycle of principal ports, in which case nothing will ever change at this port, no matter what we connect. The interesting cases in the comparison appear when, by evaluating the configurations at position i, we obtain terms rooted by agents or open paths. In the first case, we can compare the agents, and in the second case, by connecting agents to the other end of the open path (say at position j in the observable interface) we might obtain different (or the same) results at position i. In summary, given two configurations to be compared, our experiments will apply to each position i in the observable interface and will consist in:

(1) finding the canonical form of the configurations at position i (INF_i);
(2) comparing the roots of the terms for each position i in the observable interface. If they are equal, we continue our experiments in the rest of the configuration. If they are different, there are two cases: if they are constructors, we can already decide that the configurations are not equivalent, otherwise, we need to provide an agent to interact with at this position, and see if the behaviour is the same (i.e. we have to continue our experiments).

Two configurations that cannot be distinguished by any experiment will be called *bisimilar*. We will show that the bisimilarity relation can be defined as the greatest post-fixpoint of an operator (which allows us to use coinductive techniques to prove that two configurations are bisimilar), and more important, it coincides with the contextual equivalence, that is, two bisimilar configurations cannot be distinguished by any context and can therefore be exchanged without altering the semantics of the system.

We begin with some basic definitions to formalize these ideas.

Definition 10 (Visible interface). We say that a configuration

$$c = \langle t_1, \ldots, t_n \mid \Delta \rangle \in INF_i$$

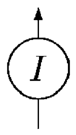
has a visible interface at position i if either t_i is not a variable or there is an open path starting at t_i and finishing at some $t_j = \alpha'(\vec{u}) \in \vec{t}$, that is, $t_i = \alpha(\vec{u})$ or $t_i = x$ and there is an open path to $t_j = \alpha'(\vec{u})$. The *visible agent* at position i is α in the first case, α' in the second case. The rest of the net is called the *kernel*: $\mathscr{K}_i(c) = \langle t_1, \dots, t_{k-1}, \vec{u}, t_{k+1}, \dots, t_n \mid \Delta \rangle$ where $k = i$ or $k = j$ depending on whether we are in the first or second case. The set of *new observable positions* in $\mathscr{K}_i(c)$, denoted $NP_{\mathscr{K}}(c, i)$, is the set of positions of the terms $\vec{u}$ if $k = i$, otherwise it contains just the new position of t_i.

We denote by $\mathscr{V}_i$ the set of all the configurations with a visible interface at position i.

If v and v' are comparable and have the same visible agent at position i, we write $SVA_i(v, v')$. If the visible agents are different, but they are not both constructors, we write $\neg\, Constr_i(v, v')$.

Graphically, a net has a visible interface at position i if the ith position of the observable interface is a principal port, or an open path finishing at a principal port. The following example illustrates this definition.

Example 11. Let $c = \langle I(x), x \mid \rangle$. Since $t_1 = I(x)$ is not a variable, $c \in \mathscr{V}_1$. Since $t_2 = x$ and there is an open path to $t_1 = I(x)$, $c \in \mathscr{V}_2$. The visible agent is I for both positions, and $\mathscr{K}_1(c) = \mathscr{K}_2(c) = \langle x, x \mid \rangle$. The net corresponding to the configuration c contains just one occurrence of agent I:



When we have different agents in the visible interfaces of the nets under test, and they are not constructors, we need to see if these agents behave in the same way for each possible agent interacting with them. This is the intuition behind the following definition.

Definition 12 (Closing). A closing at position i of a configuration

$$c = \langle \vec{t} \mid \Delta \rangle \in \mathscr{V}_i$$

denoted by $cl_i(c)$, is obtained from c by one of the following operations, where $k = i$, or $k = j$ if there is an open path starting at position i and finishing at position j in c:

(1) Replace $t_k = \alpha(\vec{s})$ in $\vec{t}$, by a list of variables $z_1, \dots, z_p \in \mathscr{N}(\vec{u})$, and add to Δ the equation $t_k == \alpha'(\vec{u})$, where
- α' is an agent such that there is an interaction rule for the active pair (α, α'), and
- the terms in $\vec{u}$ are either new variables (in which case they can appear twice in $\alpha'(\vec{u})$ or once in $\alpha'(\vec{u})$ and once in $\vec{z}$) or elements of $\vec{t}$, in which case they are erased from $\vec{t}$.

The set $NP_{cl}(c, i)$ of *new observable positions* in $cl_i(c)$ contains the positions of the variables $z_1, \dots, z_p$ in the new head if $i = k$, otherwise it contains just the new position of t_i.

(2) Erase t_k and another term t_p in $\vec{t}$ and add $t_k == t_p$ to Δ. In this case $NP_{cl}(c,i)=\emptyset$ if $i=k$, otherwise it contains just the new position of t_i.

By abuse of notation, we will denote by $cl_i(c')$ the result of applying to a configuration c' comparable with c the operations that define a closing at position i for c.

Graphically, the first operation corresponds to connecting the principal port of an agent α' to the kth observable port in the interface of the net, and connecting some auxiliary ports of α' between them (if a variable appears twice in $\vec{z}$), or to other observable ports in the net (if $\vec{u}$ contains terms in $\vec{t}$). The second operation corresponds to simply adding a wire connecting the observable ports k and p.

Example 13. *Let* $c=\langle I(x),x \mid \rangle$, $c'=\langle x,x \mid \rangle$. The following are possible closings of c at position 1. In each case we also show the configuration resulting from applying them to c'.

- $cl_1(c)=\langle z_1,z_2,x \mid I(x)=\alpha(z_1,z_2,z_3,z_3)\rangle$ which results from connecting the principal port of the agent α to the principal port of I, leaving two auxiliary ports of α free, and connecting the other two together. The new observable positions are $\{1,2\}$. Applying the same operation to c' we obtain $cl_1(c')=\langle z_1,z_2,x \mid x=\alpha(z_1,z_2,z_3,z_3)\rangle$.
- $cl_1(c)=\langle x \mid I(x)=\varepsilon\rangle$ results from applying the same operation with a 0-ary agent ε. In this case the set of new observable positions is empty. Applying the same operation to c' we obtain $cl_1(c')=\langle x \mid x=\varepsilon\rangle$.
- $cl_1(c)=\langle \mid I(x)=x\rangle$ is a closing where we apply the second operation adding a link between the ports of I. Again the set of new observable positions is empty (the net does not have an interface). Applying this operation to c' we obtain $cl_1(c')=\langle \mid x=x\rangle$.

We consider a complete lattice $(Rel, \subseteq)$ where Rel is the set of binary relations between pairs $(c,i),(c',i)$ such that c,c' are comparable configurations whose heads have at least i elements (i.e. we can talk of the ith observable port). Recall that a complete lattice is a partially ordered set $(X, \leqslant)$ such that every subset $S \subseteq X$ has a least upper bound with respect to $\leqslant$. The monotone operators $\langle \mathscr{R}\rangle$ and $[\mathscr{R}]$ for $\mathscr{R} \in Rel$ will be used to define *similarity* and *bisimilarity*, respectively. A monotone operator on $(Rel, \subseteq)$ is a function

$$\Phi: Rel \to Rel$$

such that $\forall \mathscr{R},\mathscr{R}' \in Rel : \mathscr{R} \subseteq \mathscr{R}' \Rightarrow \Phi(\mathscr{R}) \subseteq \Phi(\mathscr{R}')$.

Definition 14 (Operators). Let c,c' be comparable configurations with at least i terms in the head.

$$\begin{aligned}
(c,i)\langle\mathscr{R}\rangle(c',i) \overset{\text{def}}{\Leftrightarrow}\ & c \Downarrow_i v \in \mathscr{V}_i \Rightarrow \exists v',\ (c' \Downarrow_i v' \text{ and} \\
& \text{either } SVA_i(v,v') \text{ and} \\
& \qquad \forall p \in NP_{\mathscr{K}}(v,i), (\mathscr{K}_i(v),p)\,\mathscr{R}\,(\mathscr{K}_i(v'),p) \\
& \text{or} \neg\, Constr_i(v,v') \text{ and} \\
& \qquad \forall cl_i(v), \forall p \in NP_{cl}(v,i), (cl_i(v),p)\,\mathscr{R}\,(cl_i(v'),p)) \\
(c,i)[\mathscr{R}](c',i) \overset{\text{def}}{\Leftrightarrow}\ & (c,i)\langle\mathscr{R}\rangle(c',i) \text{ and } (c',i)\langle\mathscr{R}\rangle(c,i)
\end{aligned}$$

Proposition 15. $\langle\cdot\rangle, [\cdot]$ *are monotone operators.*

Proof. Let $\mathscr{R}, \mathscr{R}' \in Rel$ such that $\mathscr{R} \subseteq \mathscr{R}'$. We will prove that $\langle\mathscr{R}\rangle \subseteq \langle\mathscr{R}'\rangle$.

If $(c,i)\langle\mathscr{R}\rangle(c',i)$ then $c \Downarrow_i v \in \mathscr{V}_i$ implies $c' \Downarrow_i v'$ and either

- $SVA_i(v,v')$ and $\forall p \in NP_{\mathscr{K}}(v,i), (\mathscr{K}_i(v), p)\mathscr{R}(\mathscr{K}_i(v'), p)$, in which case, since $\mathscr{R} \subseteq \mathscr{R}'$, also $(\mathscr{K}_i(v), p)\mathscr{R}'(\mathscr{K}_i(v'), p)$; or
- $\neg\, Constr_i(v,v')$ and $\forall cl_i(v)$, $\forall p \in NP_{cl}(v,i)$, $(cl_i(v), p)\mathscr{R}(cl_i(v'), p)$, in which case, since $\mathscr{R} \subseteq \mathscr{R}'$, $(cl_i(v), p)\mathscr{R}'(cl_i(v'), p)$.

Therefore $(c,i)\langle\mathscr{R}'\rangle(c',i)$, and also $(c,i)[\mathscr{R}'](c',i)$. □

Definition 16 (Similarity, bisimilarity).

- A relation $\mathscr{S} \in Rel$ such that $\mathscr{S} \subseteq \langle\mathscr{S}\rangle$ (i.e. $\mathscr{S}$ is a post-fixpoint of $\langle\cdot\rangle$) is a *simulation*. The greatest such $\mathscr{S}$ is called a *similarity*, and written as $\lesssim$. If c, c' are comparable configurations with n elements in the head, then $c \vec{\lesssim} c'$ if $(c,i) \lesssim (c',i)$ for all $1 \leqslant i \leqslant n$.
- A relation $\mathscr{B} \in Rel$ such that $\mathscr{B} \subseteq [\mathscr{B}]$ (i.e. $\mathscr{B}$ is a post-fixpoint of $[\cdot]$) is a *bisimulation*. The greatest such $\mathscr{B}$ is called a *bisimilarity*, and written as $\simeq$. If c, c' are comparable configurations with n elements in the head, then $c \vec{\simeq} c'$ if $(c,i) \simeq (c',i)$ for all $1 \leqslant i \leqslant n$.

Note that $\langle\cdot\rangle$ and $[\cdot]$ posses a greatest post-fixpoint by the Tarski–Knaster fixed point theorem because $(Rel, \subseteq)$ is a complete lattice, and they are monotone operators. Moreover, $\lesssim$ and $\simeq$ are fixed points, i.e. $\lesssim = \langle\lesssim\rangle$ and $\simeq = [\simeq]$, and hence satisfy

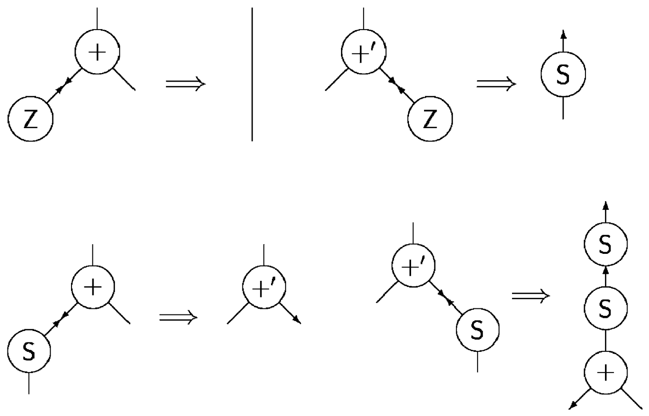
$$\begin{aligned}
(c,i) \lesssim (c',i) \overset{\text{def}}{\Leftrightarrow}\ & c \Downarrow_i v \in \mathscr{V}_i \Rightarrow \exists v', (c' \Downarrow_i v' \text{ and} \\
& \quad \text{either } SVA_i(v,v') \text{ and} \\
& \qquad \forall p \in NP_{\mathscr{K}}(v,i), (\mathscr{K}_i(v), p) \lesssim (\mathscr{K}_i(v'), p) \\
& \quad \text{or} \neg\, Constr_i(v,v') \text{ and} \\
& \qquad \forall p \in NP_{cl}(v,i), (cl_i(v), p) \lesssim (cl_i(v'), p)) \\
(c,i) \simeq (c',i) \overset{\text{def}}{\Leftrightarrow}\ & (c,i) \lesssim (c',i) \text{ and } (c',i) \lesssim (c,i).
\end{aligned}$$

There are other alternatives for the definition of the operators $\langle\cdot\rangle$ and $[\cdot]$, resulting in different relations $\lesssim$ and $\simeq$. The choice is of course guided by the equivalence relation that we want to capture.

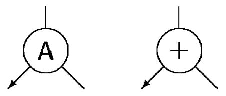
Remark 17. The main difference with the typed approach to the definition of bisimilarity resides in the definition of closings and the way they are used in the definition of the operators $\langle\cdot\rangle$ and $[\cdot]$. Here closings are applied "on demand" whereas they are a static notion in the typed framework, since the type information tells us how many arguments are needed for each agent. More precisely, in a typed net ports are classified as input or output, and a closing is built just by connecting agents to all the free *input* ports. The subject reduction property ensures that reduction will not create new free input ports. Two nets with free input ports are in the relation $\langle R\rangle$ if their closings are (for every closing); for nets without free input ports, the definition is as given above but without the second case (where we apply a closing to v, v') since all the closings were done before hand. Instead, here we close one principal port at a time, and since reduction might create a new free principal port, closings are applied in a dynamic way.

The following example illustrates this point.

Example 18. Let A be the addition agent defined in Example 6, and $+$ be a new version of addition where if one of the arguments is zero, we directly give the other argument as result.



If we compare the nets in *INF*



we see that the visible interface is different, and we have to apply a closing. This provides an argument for the addition. But after evaluation, the visible interfaces might still be different. However, after applying a second argument the same agent is visible in both nets. If we had type information saying that two of the ports of A and $+$ are inputs and the other is an output, we could close the input ports by providing two arguments at once.

Proposition 19 (Coinduction principle). *Let c, c' be comparable configurations with n observable ports. To prove $c \overset{\rightarrow}{\simeq} c'$ it suffices to find a bisimulation $\mathscr{B}$ such that $(c,i)\mathscr{B}(c',i)$ for $1 \leqslant i \leqslant n$.*

Proof. Since $\mathscr{B} \subseteq [\mathscr{B}]$, then $\mathscr{B} \subseteq \simeq$. Hence if $(c,i)\mathscr{B}(c',i)$ for $1 \leqslant i \leqslant n$, then $c \overset{\rightarrow}{\simeq} c'$. $\square$

Remark 20. The relations $\lesssim$, $\simeq$ can be defined by levels, in the same way as Abramsky's (bi)simulation for the untyped λ-calculus [1]. Let c, c' be comparable configurations with n observable ports, we define:

- level 0: $(c,i) \precsim^0 (c',i)$ holds for $1 \leqslant i \leqslant n$;
- level $k+1$: for $1 \leqslant i \leqslant n$

$$\begin{aligned}(c,i) \precsim^{k+1} (c',i) \overset{\text{def}}{\Leftrightarrow}\ & c \Downarrow_i v \in \mathscr{V}_i \Rightarrow \exists v', (c' \Downarrow_i v' \text{ and} \\ & \text{either } SVA_i(v,v') \text{ and} \\ & \quad \forall p \in NP_{\mathscr{K}}(v,i), (\mathscr{K}_i(v), p) \precsim^k (\mathscr{K}_i(v'), p) \\ & \text{or} \neg\, Constr_i(v,v') \text{ and} \\ & \quad \forall p \in NP_{cl}(v,i), (cl_i(v), p) \precsim^k (cl_i(v'), p))\end{aligned}$$

- $(c,i) \precsim (c',i) \overset{\text{def}}{\Leftrightarrow} \forall k, (c,i) \precsim^k (c',i)$.

In Section 5 we give some examples of application of the coinduction principle to prove bisimilarity. In particular, we will show that two nets may be bisimilar even if they are *not* equivalent in the equational theory defined by the interaction rules. The bisimilarity relation strictly includes the equational theory $\leftrightarrow^*$, as the following theorem shows.

Theorem 21 (Bisimilarity includes the equational theory). *If $c \leftrightarrow^* c'$ then $c \eqsim c'$.*

Proof. By coinduction. We show that the relation containing the pairs $((c,i),(c',i))$ such that $c \leftrightarrow^* c'$ is a bisimulation. If $c \Downarrow_i v \in \mathscr{V}_i$ then by strong confluence $c' \Downarrow_i v'$ and $SVA_i(v,v')$. Moreover, $\mathscr{K}_i(v) \leftrightarrow^* \mathscr{K}_i(v')$ since $v \leftrightarrow^* v'$. $\square$

5. Examples

In this section we shall show several examples of the use of the coinduction principle.

The identity agent and a wire. Let I be the identity agent defined by rules

$$I(\alpha(x_1,\ldots,x_n)) \bowtie \alpha(I(x_1),\ldots,I(x_n))$$

for any $\alpha \in \Sigma$. Then $\langle I(x), x \mid \rangle \eqsim \langle x, x \mid \rangle$.

First we show that these configurations are in the relation $\precsim$. By coinduction, it is sufficient to prove that there is a simulation R containing these pairs for $i = 1,2$. We will take R containing pairs $((c,i),(c',i))$ such that c' is obtained from c by erasing the I agents at the root of a term in the head, or at the root of a member of an equation. The following configurations c, c' are related by R:

$$\begin{aligned} c &= \langle I(x_1),\ldots,I(x_n),\vec{x} \mid \rangle, \\ c' &= \langle \vec{x},\vec{x} \mid \rangle, \\ c &= \langle I(x_1),\ldots,I(x_{k-1}),\vec{z},I(x_{k+1}),\ldots,I(x_n),\vec{x} \mid I(x_k) = \alpha(\vec{z}) \rangle, \\ c' &= \langle x_1,\ldots,x_{k-1},\vec{z},x_{k+1},\ldots,x_n,\vec{x} \mid x_k = \alpha(\vec{z}) \rangle, \end{aligned}$$

$$\begin{aligned} c &= \langle I(x_1),\ldots,I(x_{k-1}),\vec{z},I(x_{k+1}),\ldots,I(x_n),\alpha(t[\vec{x}])\,|\,I(x_k)=\alpha'(\vec{z})\rangle,\\ c' &= \langle x_1,\ldots,x_{k-1},\vec{z},x_{k+1},\ldots,x_n,\alpha(t[\vec{x}])\,|\,x_k=\alpha'(\vec{z})\rangle. \end{aligned}$$

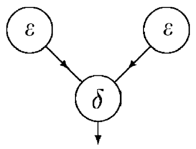
When $c \Downarrow_i v \in \mathscr{V}_i$, then $c' \Downarrow_i v'$, and either they have the same visible agent α at position i, in which case the kernels are in the relation for all the new observable positions, or if they differ, then one is rooted by I and the other is just a variable. In that case the closings are in the relation, which is sufficient since I is not a constructor.

To show $\langle x,x\,|\,\rangle \overset{\rightarrow}{\lesssim} \langle I(x),x\,|\,\rangle$ we take the symmetric of R.

In the same way we can prove that $\langle x\,|\,\delta(x,y)=y\rangle \overset{\rightarrow}{\simeq} \langle x\,|\,\delta(x,y)=I(y)\rangle$. Although the interface normal forms have different visible agents (see Example 5), we can take the same R as before to prove that they are bisimilar.

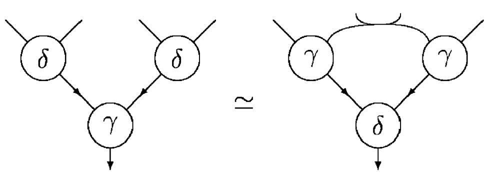
Two versions of addition. We consider now the two agents A and $+$ defined in Example 18. We can prove $\langle +(x,y),x,y\,|\,\rangle \overset{\rightarrow}{\simeq} \langle \mathsf{A}(x,y),x,y\,|\,\rangle$ by coinduction. The relation R containing $\lesssim$, this pair, its closings, and the closings of the configurations $\langle x,y,+'(x,y)\,|\,\rangle$ and $\langle \mathsf{A}(x,y),S(x),y\,|\,\rangle$, is a simulation, and so is its symmetric.

Copying before erasing or just erasing. In the system of the interaction combinators, replacing a net of the form



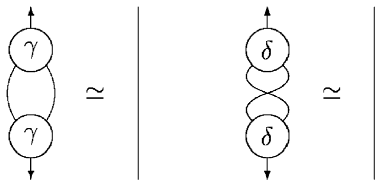
by the agent ε seems an intuitive optimization. We can prove that they are bisimilar by coinduction, showing that there is a bisimulation B containing the pair $((c,1),(c',1))$ where $c=\langle \delta(\varepsilon,\varepsilon)\,|\,\rangle$ and $c'=\langle \varepsilon\,|\,\rangle$. The main Theorem 26 tells us then that these configurations are contextually equivalent and, therefore, we can replace the first one by the second one in any context, without altering the behaviour of the program—the optimization is correct.

Agents γ and δ. The following nets are bisimilar:



To show this using the coinduction principle it is sufficient to consider a relation containing $\simeq$ and these pairs, for any closing of the free principal port. The interesting closings are built by adding an agent ε, γ, or δ (the closings using just wires do not reduce to a value with a visible interface). The case of ε is trivial. For the other cases, by reducing to interface normal form we obtain configurations that have the same visible agents and whose kernels are easily shown to be bisimilar, hence contained in our relation.

η-rules for γ and δ. The following nets are bisimilar:

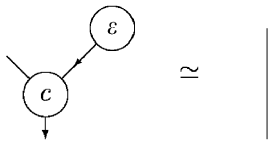


We prove this by coinduction, using a relation containing $\simeq$ and these pairs, together with their closings, as in the previous example. To simplify the proof, one may use the main Theorem 26.

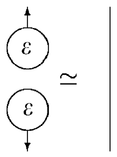
Note that these last two equivalences are neither included in the equational theory (since the nets are different normal forms) nor provable using the path semantics developed by Lafont [12].

Encoding of the λ-calculus. There are a wealth of optimizations that can be applied in various systems of interaction nets for the λ-calculus. We just mention here two examples that can be applied in the systems [13,14].

The first example is a commonly occurring configuration that arises during computation in these systems. The agent c is a copying agent, and ε is an erasing agent. The following equivalence states that to copy a net, then erase one of the copies is the same as doing nothing at all; clearly a substantial optimization.



A second example is an η-rule for erasing agents:



(One has to take care to distinguish between ε (erase) and W (weakening) agents for this example, which were represented as a single agent in the systems mentioned: the rule does not apply to W agents.)

Both of these examples can be proved equivalent using the techniques presented in this paper.

Non-termination. Finally, we give an example where one of the nets does not have an interface normal form. Consider the net represented by $c = \langle y \mid \alpha(\varepsilon) = \beta(\alpha(y)) \rangle$ and the rule $\alpha(a) \bowtie \beta(\beta(\alpha(a)))$. We have the following non-terminating reduction sequence:

$$\begin{aligned} \langle y \mid \alpha(\varepsilon) = \beta(\alpha(y)) \rangle &\rightarrow \langle y \mid \varepsilon = a, \beta(\alpha(a)) = \alpha(y) \rangle \\ &\rightarrow \langle y \mid \beta(\alpha(\varepsilon)) = \alpha(y) \rangle \\ &\rightarrow \cdots . \end{aligned}$$

Since c does not have an interface normal form, then $c \vec{\lesssim} c'$ for any c'. Let v be a configuration in interface normal form but without a visible interface, for example $v = \langle x \mid y = \delta(x, y) \rangle$. Also in this case $v \vec{\lesssim} c'$ for any c'. In particular, $c \vec{\simeq} v$.

6. Contextual equivalence

We begin with the definition of context. We will define a set of operations that build a context for a configuration, in the same way that closings were defined by operations on configurations. But there are more operations in the case of contexts, and we can have a sequence of operations instead of just one operation.

Definition 22 (Context). A context at position i for a configuration

$$c = \langle \vec{t} \mid \Delta \rangle$$

is defined by a (possibly empty) sequence of operations, that is, a context may be empty, or inductively defined as follows (we distinguish three cases according to the first operation used), where $k = i$, or $k = j$ if there is an open path starting at position i and finishing at position j in c.

(1) Addition of agent by principal port: This operation replaces t_k in $\vec{t}$ by a list of variables $z_1, \ldots, z_p \in \mathcal{N}(\vec{u})$, and adds to Δ the equation $t_k = \alpha(\vec{u})$, where
- α is any agent and
- the terms in $\vec{u}$ are either new variables (in which case they can occur twice in $\alpha(\vec{u})$ or once in $\alpha(\vec{u})$ and once in $\vec{z}$) or elements of $\vec{t}$, in which case they are erased from $\vec{t}$.

In this case the rest of the sequence is the concatenation of contexts at the positions of the variables $z_1, \ldots, z_p$ in the new head and at the new position of t_i if $i \neq k$.

(2) Addition of agent by auxiliary port: This operation replaces t_k in $\vec{t}$ by a list of variables $z_1, \ldots, z_p$ occurring free in $y = \alpha(\vec{u})$ and adds this equation to Δ, where
- α is any agent and
- the terms in $\vec{u}$ are either new variables (in which case they can occur twice in $\alpha(\vec{u})$ or once in $\alpha(\vec{u})$ and once in $\vec{z}$) or elements of $\vec{t}$, in which case they are erased from $\vec{t}$. The term t_k must occur in $\vec{u}$.

Also in this case the rest of the sequence is composed of contexts at the positions of the variables $z_1,\dots,z_p$ in the new head and at the new position of t_i if $i \neq k$.

(3) Addition of a wire: erase t_k and another term t_p in $\vec{t}$ and add $t_k == t_p$ to Δ. In this case the rest of the sequence is empty if $i = k$, otherwise it is a context at the new position of t_i.

We denote by $op^1_{i,\vec{j}}(c)$, (resp. $op^2_{i,\vec{j}}(c)$, $op^3_{i,\vec{j}}(c)$) the result of applying an operation *op* of the first (resp. second, third) class to the configuration c at position i, using the positions $\vec{j}$ in $\vec{t}$. We write $op_{i,\vec{j}}(c)$ when we do not need to distinguish the kind of operation applied. We denote by $C_i[c]$ the configuration resulting of applying the context C, defined by a sequence of operations as above, to the configuration c at position i, and by $C(c,i)$ a generic context for c at position i.

We compute the set of *new observable positions* $NP_C(c,i)$ of $C_i[c]$ as follows: we start with the set $\{i\}$, and compute a new set each time we perform an operation. The first and second operations add the positions of the variables $z_1,\dots,z_p$ in the new head, and if $i = k$ they erase i, otherwise they replace i by the new position of t_i in the head. The third operation simply erases the position i from the set if $i = k$, otherwise replaces i by the new position of t_i.

We will also denote by $C_i[c']$ the result of applying to a configuration c' comparable with c the operations that define a context at position i for c.

Graphically, the first two operations correspond to connecting an agent α to an observable port of the net (using the principal port of α in the first one, and an auxiliary port in the second one). The third operation corresponds to adding a wire connecting the observable ports k and p. Closings are particular cases of contexts defined by one operation of the first or third class.

Definition 23 (Contextual preorder and contextual equivalence). Let c,c' be comparable configurations with n elements in the head.

$$\begin{aligned}
c \quad \vec{\leqslant} \quad c' \quad &\overset{\text{def}}{\Leftrightarrow} \forall i \in [1\dots n], (c,i) \leqslant (c',i)\\
(c,i) \leqslant (c',i) \quad &\overset{\text{def}}{\Leftrightarrow} \forall C(c,i), \forall p \in NP_C(c,i), C_i[c] \Downarrow_p v \in \mathscr{V}_p \Rightarrow\\
&\qquad \exists v', (C_i[c'] \Downarrow_p v' \text{ and}\\
&\qquad\quad \text{either } SVA_p(v,v')\\
&\qquad\quad \text{or } \neg\, Constr_p(v,v')),\\
(c,i) = (c',i) \quad &\overset{\text{def}}{\Leftrightarrow} (c,i) \leqslant (c',i) \text{ and } (c',i) \leqslant (c,i),\\
c \quad \vec{=} \quad c' \quad &\overset{\text{def}}{\Leftrightarrow} \forall i \in [1\dots n], (c,i) = (c',i).
\end{aligned}$$

Remark 24. Abramsky's definition for untyped λ-calculus is based on convergence: Let $t, t' \in \Lambda^0$ (closed λ-terms),

$$t \leqslant t' \overset{\text{def}}{\Leftrightarrow} \forall C[] \in \Lambda^0, \quad C[t] \Downarrow \Rightarrow C[t'] \Downarrow .$$

This is natural since there is only one constructor (λ), therefore convergence of closed terms is equivalent to convergence to the same constructor. Our definition subsumes this one, but since we can have more constructors (or none), apart from requiring that equivalent programs behave in the same way with respect to convergence, we also require that they produce the same agent, or at least not different constructors.

As for the notion of bisimilarity, there are several alternative definitions of contextual equivalence (depending on the notion of context that we use and the kind of observations that we make). The definition that we have given corresponds to the notion of bisimilarity of Definition 16, as we will show in the following section. This result justifies the use of coinductive techniques to prove contextual equivalence of nets.

7. Main result

In this section we show that the notions of contextual equivalence and bisimilarity coincide, if the interaction net system has "enough contexts". We need to ensure that there are enough contexts in Σ to extract the kernels of all the values.

Definition 25. A system of interaction is *complete* if for any $v \in \mathscr{V}_i$ with visible agent α at position i, there exists a context C^α such that

$$\forall p \in NP_{\mathscr{K}}(v,i), (\mathscr{K}_i(v), p) \simeq (C_i^\alpha[v], p) \text{ and } p \in NP_{C^\alpha}(v,i).$$

For example, if we have lists defined by the agents *cons* and *nil* (constructors), and *append* (destructor) implements the concatenation of lists, the system is not complete: no context can extract the elements of the lists. To complete the system, we add for instance the agents *head* and *tail* with the corresponding interaction rules allowing us to extract the first element of the list (head) and the rest (tail).

Theorem 26. *The contextual preorder $\vec{\leqslant}$ (resp. equivalence $\vec{=}$) coincides with similarity $\vec{\lesssim}$ (resp. bisimilarity $\vec{\simeq}$) when the interaction net system is complete.*

If the system is not complete then similarity (resp. bisimilarity) is included in the contextual preorder (resp. equivalence) but not necessarily the converse.

Proof. We will prove the inclusions:
(1) $\lesssim \subseteq \leqslant$, and
(2) $\leqslant \subseteq \lesssim$ assuming completeness.
The first one is the most difficult to prove. It is sufficient to show that $\lesssim$ is preserved by context, more precisely, $(c,i) \lesssim (c',i) \Rightarrow \forall C(c,i), \forall p \in NP_C(c,i), (C_i[c],p) \lesssim (C_i[c'],p)$. We will prove that each of the operations used to build a context preserves $\lesssim$. A relation that is a preorder (i.e. reflexive and transitive) preserved by these operations will be called a *precongruence*. We will then prove that $\lesssim$ is a precongruence.

Proposition 27 (Preorder). *$\lesssim$ is reflexive and transitive.*

Proof. *Reflexivity*: The set of pairs $((c,i),(c,i))$ is a simulation since $\Downarrow$ is deterministic (Proposition 8). Therefore $\lesssim$ is reflexive by coinduction.
Transitivity: $\lesssim \circ \lesssim = \langle \lesssim \rangle \circ \langle \lesssim \rangle \subseteq \langle \lesssim \circ \lesssim \rangle$. $\square$

To prove that $\lesssim$ is preserved by the operations that build a context, we will follow Howe's method [9] and use an auxiliary relation $\lesssim^*$, called the *Precongruence candidate*. The relation $\lesssim^*$ will be easily shown to be preserved by the operations; the main task will be to show that it coincides with $\lesssim$.

Definition 28 (Precongruence candidate). Let c, c' be comparable configurations with n elements in their heads.

$$\begin{array}{llll} c \vec{\lesssim}^* c' & \overset{\text{def}}{\Leftrightarrow} & \forall i \in [1 \ldots n], & (c,i) \lesssim^* (c',i) \\ (c,i) \lesssim^* (c',i) & \overset{\text{def}}{\Leftrightarrow} & \text{either} & (c,i) \lesssim (c',i), \\ & & \text{or} & c = op_{p,\vec{j}}\,(d),\ i \in NP_{op}(d,p), \\ & & & (d,q) \lesssim^* (d',q), \forall q \in \vec{j} \text{ and} \\ & & & (op_{p,\vec{j}}\,(d'),i) \lesssim (c',i) \end{array}$$

The relation $\vec{\lesssim}^*$ enjoys the following properties.

Proposition 29. (1) $\vec{\lesssim} \subseteq \vec{\lesssim}^*$.
(2) $\vec{\lesssim}^*$ *is reflexive.*
(3) $c \vec{\lesssim}^* c', c' \vec{\lesssim} c'' \Rightarrow c \vec{\lesssim}^* c'$
(4) $\vec{\lesssim}^*$ *is preserved by context*: $c \vec{\lesssim}^* c' \Rightarrow \forall i,\ \forall C(c,i),\ C_i[c] \vec{\lesssim}^* C_i[c']$.

Proof.
(1) By definition of $\lesssim^*$.
(2) Using the previous part and reflexivity of $\lesssim$.
(3) By definition of $\lesssim^*$, using transitivity of $\lesssim$.
(4) For the positions not affected by the context (i.e. $\forall p \notin NP_C(c,j)$), we obtain $(C_j[c],p) \lesssim^* (C_j[c'],p)$ by induction on the definition of $\lesssim^*$, using the fact that $(c,i) \lesssim (c',i)$ implies $(C_j[c],i') \lesssim (C_j[c],i')$ for $j \neq i$ and where i' is the new position of i in $C_j[c]$. For the positions $p \in NP_C(c,j)$, we obtain $(C_j[c],p) \lesssim^* (C_j[c'],p)$ by induction on the number of operations in C. For one operation the property holds by definition of $\lesssim^*$, using reflexivity of $\lesssim$. $\square$

To prove that $\lesssim$ is a precongruence it is sufficient to prove that it coincides with $\lesssim^*$, as the following theorem says.

Theorem 30. $\lesssim$ *is a precongruence iff* $\lesssim^* = \lesssim$.

Proof. Assume $\lesssim$ is a precongruence. We have already proved $\lesssim \subseteq \lesssim^*$ (Proposition 29), and we can prove $\lesssim^* \subseteq \lesssim$ by induction on the definition of $\lesssim^*$ using the fact that $\lesssim$ is transitive and a precongruence.

Assume $\precsim^* = \precsim$. Since $\precsim^*$ is preserved by context, and we have already proved that $\precsim$ is a preorder, we deduce that it is a precongruence. □

It remains to prove $\vec{\precsim}^* \subseteq \vec{\precsim}$. By coinduction, we will prove that if c, c' are comparable configurations with n terms in their heads, then $c \vec{\precsim}^* c'$ implies $(c,i)\langle\precsim^*\rangle(c',i)$ for $1 \leqslant i \leqslant n$. We do this in two steps:

Proposition 31. (1) $v \in \mathscr{V}_i,\ (v,i) \precsim^* (c',i) \Rightarrow (v,i)\langle\precsim^*\rangle(c',i)$;
(2) $c \vec{\precsim}^* c',\ c \Downarrow_i v \in \mathscr{V}_i \Rightarrow v \vec{\precsim}^* c'$

Proof.
(1) Assume $(v,i) \precsim^* (c',i)$. Since $v \in \mathscr{V}_i$, only two cases of the definition of $\precsim^*$ are possible:
 (a) $(v,i) \precsim (c',i)$, in which case $c' \Downarrow_i v'$ and
 either $SVA_i(v,v')$ and

$$\forall p \in NP_{\mathscr{K}}(v,i), (\mathscr{K}_i(v), p) \precsim (\mathscr{K}_i(v'), p),$$

 which implies that $(\mathscr{K}_i(v), p) \precsim^* (\mathscr{K}_i(v'), p)$ by definition of $\precsim^*$, otherwise

$$\neg\, Constr_i(v,v') \text{ and } \forall p \in NP_{cl}(v,i),\ (cl_i(v), p) \precsim (cl_i(v'), p),$$

 which implies $(cl_i(v), p) \precsim^* (cl_i(v'), p)$ by definition of $\precsim^*$. Therefore $(v,i)\langle\precsim^*\rangle(c',i)$.
 (b) $v = op^2_{p,\vec{j}}(d)$, $\forall q \in \vec{j}$, $(d,q) \precsim^* (d',q)$, and $(op^2_{p,\vec{j}}(d'), i) \precsim (c',i)$. That is, v is obtained by adding an agent at position p to d, leaving its principal port free (position i). Since $op^2_{p,\vec{j}}(d') \in \mathscr{V}_i$ (by induction on the definition of $\precsim^*$), the latter implies that $c' \Downarrow_i v'$ and:
 - either $SVA_i(op^2_{p,\vec{j}}(d'), v')$ and $\forall r \in NP_{\mathscr{K}}(v,i), (\mathscr{K}_i(op^2_{p,\vec{j}}(d')), r) \precsim (\mathscr{K}_i(v'), r)$, in which case, since $\mathscr{K}_i(op^2_{p,\vec{j}}(d')) = d'$, $\mathscr{K}_i(v) = d$ and the positions r coincide with the positions q, we obtain

$$\forall r \in NP_{\mathscr{K}}(v,i), (\mathscr{K}_i(v), r) \precsim^* (\mathscr{K}_i(v'), r)$$

 by Proposition 29, part 3.
 - or $\neg\, Constr_i(op^2_{p,\vec{j}}(d'), v')$ and

$$\forall r \in NP_{cl}(op^2_{p,\vec{j}}(d'), i), (cl_i(op^2_{p,\vec{j}}(d')), r) \precsim (cl_i(v'), r),$$

 in which case, we deduce $\neg\, Constr_i(v,v')$, and since $\precsim^*$ is preserved by the operations that build a context (which include the operations that define a closing), we obtain $(cl_i(v), r) \precsim^* (cl_i(v'), r)$ by Proposition 29, part 3.

(2) It is sufficient to prove that $\forall i, \forall j,\ (c,i) \precsim^* (c',i)$ and $c \Downarrow_j v \in \mathscr{V}_j \Rightarrow (v,i) \precsim^* (c',i)$. We prove this by induction, using the well-founded lexicographic ordering associated to $\Downarrow_j$ and the size of c. If $c = v$ then we are done. If $c \Downarrow_j v \in \mathscr{V}_j$ and $c \neq v$, we analyse each possible case for $(c,i) \precsim^* (c',i)$.
 (a) $(c,i) \precsim (c',i)$. Since $\precsim$ is preserved by reduction (Theorem 21), $(v,i) \precsim (c',i)$, hence $(v,i) \precsim^* (c',i)$.

(b) $c = op^1_{p,\vec{j}}(d), i \in NP_{op^1}(d, p), \forall q \in \vec{j}(p \in \vec{j}), (d, q) \lesssim^* (d', q)$, and $(op^1_{p,\vec{j}}(d'), i) \lesssim (c', i)$.

If j is not related to i, then $d \Downarrow_j u \in \mathscr{V}_j$ with an equivalent derivation, and since the size of the configuration decreased, by induction we get $(u, q) \lesssim^* (d', q)$. If j is related to i then $d \Downarrow_p u \in \mathscr{V}_p$ with a shorter derivation, and again by induction, $(u, q) \lesssim^* (d', q)$.

Since $\lesssim^*$ is preserved by the operations, $(op^1_{p,\vec{j}}(u), i) \lesssim^* (op^1_{p,\vec{j}}(d'), i)$. Now, if $op^1_{p,\vec{j}}(u) \Downarrow_j v$ with a shorter derivation than $c \Downarrow_j v$, then we get $(v, i) \lesssim^* (c', i)$ by Proposition 29, part 3. Otherwise, $d = u$ and $u \in \mathscr{V}_p$. Note that the operation created an active pair between the agent with principal port free at position p and the new agent added, let R be the right-hand side of the associated rule. By the previous part, we deduce $(u, p) \lesssim (d', p)$. Then $d' \Downarrow_p v'$ such that

(i) either $SVA_p(u, v')$ and $\forall r \in NP_{\mathscr{K}}(u, p), (\mathscr{K}_p(u), r) \lesssim (\mathscr{K}_p(v'), r)$, in which case $(\mathscr{K}_p(u), r) \lesssim^* (\mathscr{K}_p(v'), r)$. Let e be the configuration obtained by adding to $\mathscr{K}_p(u)$ the right-hand side R of the interaction rule for the active pair created (using the operations), and e' the configuration obtained by applying the same operations to $\mathscr{K}_p(v')$. The relation $\lesssim^*$ is preserved by these operations (since $\forall q \in \vec{j}(p \in \vec{j})$, $(d, q) \lesssim^* (d', q)$, and we can show by induction on the definition of $\lesssim^*$ that they are still in the relation after erasing the agent at position p), therefore $(e, i) \lesssim^* (e', i)$. Since $c' \leftrightarrow e'$, $(e', i) \lesssim (c', i)$. Therefore by Proposition 29, $(e, i) \lesssim^* (c', i)$. Now we can apply the induction hypothesis to e, which reduces to v with a shorter derivation, hence $(v, i) \lesssim^* (c', i)$.

(ii) or $\neg Constr_p(u, v')$ and $\forall r \in NP_{cl}(u, p), (cl_p(u), r) \lesssim (cl_p(v'), r)$. In this case, since c is a particular closing of u at position p, and $\lesssim$ includes the equational theory, we obtain $(v, i) \lesssim^* (c', i)$.

(c) $c = op^2_{p,\vec{j}}(d)$, $i \in NP_{op^2}(d, p)$, $(d, q) \lesssim^* (d', q)$, for $q \in \vec{j}$, and $(op^2_{p,\vec{j}}(d'), i) \lesssim (c', i)$.

Since $c \neq v$, $j \neq i$. Therefore $d \Downarrow_j v'$ and by induction $(v', q) \lesssim^* (d', q)$, $\forall q \in \vec{j}$. Since $\lesssim^*$ is preserved by the operations we obtain $(v, i) \lesssim^* (op^2_{p,\vec{j}}(d'), i)$, and by Proposition 29, $(v, i) \lesssim^* (c', i)$.

(d) $c = op^3_{p,j_1,j_2}(d), i \in NP_{op^3}(d, p)$ (i.e. there is an open path from i to j_1), $(d, q) \lesssim^* (d', q)$, for $q \in \vec{j}$, and $(op^3_{p,\vec{j}}(d'), i) \lesssim (c', i)$.

This case is similar to the second one. Again the difficulty appears when $d = u \in \mathscr{V}_{j_1}, \mathscr{V}_{j_2}$. Using the previous part, we deduce that $(d, q) \lesssim (d', q)$ for $q = j_1, j_2$. Therefore $d' \Downarrow_{j_1} v'$ such that either:

(i) $SVA_{j_1}(u, v')$ and $\forall r \in NP_K(u, j_1), (\mathscr{K}_{j_1}(u), r) \lesssim (\mathscr{K}_{j_1}(v'), r)$; or

(ii) $\neg Constr_{j_1}(u, v')$ and $\forall r \in NP_{cl}(u, j_1), (cl_{j_1}(u), r) \lesssim (cl_{j_1}(v'), r)$. In this case, since c is a particular closing of u at position j_1, and $\lesssim$ includes the equational theory, we obtain $(v, i) \lesssim^* (c', i)$.

In the first case, we also know that $(d, j_2) \lesssim (v', j_2)$ and therefore $v' \Downarrow_{j_2} v''$ and again we have two cases. The second one is solved as before. In the first one we know that v'' has the same visible agents at positions j_1 and

j_2 as u, and by coinduction we can show that $(\mathscr{K}_{j_1,j_2}(d),r)\precsim(\mathscr{K}_{j_1,j_2}(v''),r)$ for all $r\in NP_{\mathscr{K}}(d,j_1,j_2)$. And since $\precsim\subseteq\precsim^*$, and $\precsim^*$ is preserved by the operations, we can add the right hand side of the rule for the active pair obtaining $(e,i)\precsim^*(e',i)$. Since $c'\leftrightarrow e'$, $(e',i)\precsim(c',i)$. Therefore by Proposition 29, $(e,i)\precsim^*(c',i)$. We can now apply the induction hypothesis to e, which reduces to v with a shorter derivation, hence $(v,i)\precsim^*(c',i)$. □

This concludes the proof of the first inclusion: $\precsim\subseteq\leqslant$.

To prove the second inclusion we use coinduction: it is sufficient to show $\leqslant\subseteq\langle\leqslant\rangle$. Assume $(c,i)\leqslant(c',i)$. By definition of $\leqslant$, using an empty context, $c,\Downarrow_i v\in\mathscr{V}_i\Rightarrow\exists v', c'\Downarrow_i v'$ and either $SVA_i(v,v')$ or $\neg Constr_i(v,v')$. In the latter case we are done, since closings are particular cases of contexts. We only need to prove that in the first case $(\mathscr{K}_i(v),p)\leqslant(\mathscr{K}_i(v'),p)$, $\forall p\in NP_{\mathscr{K}}(v,i)$. But we know by completeness that $(\mathscr{K}_i(v),p)\simeq(C_i^{\alpha}[v],p),\forall p\in NP_{\mathscr{K}}(v,i)$. Moreover, since bisimilarity includes the equational theory (Theorem 21), and $(c,i)\leqslant(c',i)$: $(C_i^{\alpha}[v],p)\simeq(C_i^{\alpha}[c],p)\leqslant(C_i^{\alpha}[c'],p)\simeq(C_i^{\alpha}[v'],p)$. Again by completeness (since $SVA_i(v,v')$), $(C_i^{\alpha}[v'],p)\simeq(\mathscr{K}_i(v'),p)$.

Since we have already proved $\simeq\subseteq=$, we get

$$(\mathscr{K}_i(v),p)\leqslant(\mathscr{K}_i(v'),p),\ \forall p\in NP_{\mathscr{K}}(v,i)$$

as required. □

8. Conclusion

In this paper we have presented a notion of bisimilarity for (untyped) interaction nets, and shown that it coincides with a notion of contextual equivalence. Therefore we have a simple proof technique (coinduction) for showing when two nets are operationally equivalent. We also remark that systems of typed interaction nets are included in this framework, and thus the results of this paper extend our earlier work [3].

One of the main applications that we see for this work are general correctness proofs for optimizations in interaction net implementations of various systems, such as the λ-calculus or term rewriting systems.

References

[1] S. Abramsky, The lazy λ-calculus, in: D.A. Turner (Ed.), Research Topics in Functional Programming, Addison-Wesley, Reading, MA, 1990, pp. 65–117 (Chapter 4).

[2] D. Bechet, Partial evaluation of interaction nets, in: M. Billaud, P. Castéran, M.M. Corsini, K. Musumbu, A. Rauzyand (Eds.), Proc. Second Workshop on Static Analysis WSA'92; Bigre J. 81–82 (1992) 331–338.

[3] M. Fernández, I. Mackie, Coinductive techniques for operational equivalence of interaction nets, in: Proc. 13th Annu. IEEE Symp. on Logic in Computer Science (LICS'98), IEEE Computer Society Press, Silver Spring, MD, June 1998, pp. 321–332.

[4] M. Fernández, I. Mackie, A calculus for interaction nets in: G. Nadathur (Ed.), Proc. Internat. Conf. on Principles and Practice of Declarative Programming (PPDP'99), Lecture Notes in Computer Science, vol. 1702, Springer, Berlin, September 1999, pp. 170–187.

[5] M. Fernández, I. Mackie, A theory of operational equivalence for interaction nets, in: G. Gonnet, D. Panario, A. Viola (Eds.), LATIN 2000. Theoretical Informatics. Proc. Fourth Latin American Symp., Punta del Este, Uruguay, Lecture Notes in Computer Science, vol. 1776, Springer, Berlin, April 2000, pp. 447–456.
[6] J.-Y. Girard, Linear logic, Theoret. Comput. Sci. 50 (1) (1987) 1–102.
[7] G. Gonthier, M. Abadi, J.-J. Lévy, The geometry of optimal lambda reduction, in: Proc. 19th ACM Symp. on Principles of Programming Languages (POPL'92), ACM Press, New York, January 1992, pp. 15–26.
[8] A.D. Gordon, G.D. Rees, Bisimilarity for a first-order calculus of objects with subtyping, in: Proc. 23rd ACM Symp. on Principles of Programming Languages, ACM Press, New York, January 1996.
[9] D.J. Howe, Proving congruence of bisimulation in functional programming languages, Inform. Comput. 124 (2) (1996) 103–112.
[10] L. Khalil, Polymorphic types for interaction nets, in: H. Ehrig, G. Taentzer (Eds.), Proc. GRATRA'2000, Joint Appligraph and Getgrats Workshop on Graph Transformation Systems, Technical University of Berlin Report Number 2000–2, March 2000, pp. 257–266.
[11] Y. Lafont, Interaction nets, in: Proc. 17th ACM Symp. on Principles of Programming Languages (POPL'90), ACM Press, New York, January 1990, pp. 95–108.
[12] Y. Lafont, Interaction combinators, Inform. Comput. 137 (1) (1997) 69–101.
[13] I. Mackie, The geometry of implementation, Ph.D. Thesis, Department of Computing, Imperial College of Science, Technology and Medicine, September 1994.
[14] I. Mackie, YALE: Yet another lambda evaluator based on interaction nets, in: Proc. Third ACM SIGPLAN Internat. Conf. on Functional Programming (ICFP'98), ACM Press, New York, September 1998, pp. 117–128.
[15] J.S. Pinto, Sequential and concurrent abstract machines for interaction nets, in: J. Tiuryn (Ed.), Proc. Foundations of Software Science and Computation Structures (FOSSACS), Lecture Notes in Computer Science, vol. 1784, Springer, Berlin, 2000, pp. 267–282.
[16] A.M. Pitts, Operationally-based theories of program equivalence, in: P. Dybjer, A.M. Pitts (Eds.), Semantics and Logics of Computation, Publications of the Newton Institute, Cambridge University Press, Cambridge, 1997, pp. 241–298.