A Truly Concurrent View of Linda Interprocess Communication

N. Busi R. Gorrieri G. Zavattaro

Technical Report UBLCS-97-2

February 1997

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in gzipped PostScript format via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`. All local authors can be reached via e-mail at the address *last-name*`@cs.unibo.it`. Questions and comments should be addressed to `tr-admin@cs.unibo.it`.

Recent Titles from the UBLCS Technical Report Series

95-8 *Building Hypermedia for Learning: a Framework Based on the Design of User Interface*, S. Mangiaracina, C. Maioli, February 1995.

95-9 *The Bologna Optimal Higher-Order Machine*, A. Asperti, C. Giovannetti, A. Naletto, March 1995.

95-10 *Synchronization Support and Group-Membership Services for Reliable Distributed Multimedia Applications*, F. Panzieri, M. Roccetti, March 1995 (Revised July 1995).

95-11 *The Inherent Cost of Strong-Partial View-Synchronous Communication*, Ö. Babaoğlu, R. Davoli, L.-A. Giachini, P. Sabattini, April 1995.

95-12 *On the Complexity of Beta-Reduction*, A. Asperti, July 1995.

95-13 *Optimal Multi-Block Read Schedules for Partitioned Signature Files*, P. Ciaccia, August 1995.

95-14 *Integrating Performance and Functional Analysis of Concurrent Systems with EMPA*, M. Bernardo, L. Donatiello, R. Gorrieri, September 1995 (Revised March 1996).

95-15 *On Programming with View Synchrony*, Ö. Babaoğlu, A. Bartoli, G. Dini, September 1995.

95-16 *Generative Communication in Process Algebra*, P. Ciancarini, R. Gorrieri, G. Zavattaro, October 1995.

95-17 *Dynamic Declustering Methods for Parallel Grid Files*, P. Ciaccia, A. Veronesi, November 1995.

95-18 *Group Membership and View Synchrony in Partitionable Asynchronous Distributed Systems: Specifications*, Ö. Babaoğlu, R. Davoli, A. Montresor, November 1995 (Revised September 1996).

96-1 *An Investigation on the Optimal Implementation of Processes*, C. Laneve, January 1996.

96-2 *Expansivity, Permutivity, and Chaos for Cellular Automata*, F. Fagnani, L. Margara, January 1996.

96-3 *Enriched View Synchrony: A Paradigm for Programming Dependable Applications in Partitionable Asynchronous Distributed Systems*, Ö. Babaoğlu, A. Bartoli, G. Dini, February 1996 (Revised November 1996).

96-4 *May and Must Testing in the Join-Calculus*, C. Laneve, March 1996.

96-5 *The Shape of Shade: a Coordination System*, S. Castellani, P. Ciancarini, D. Rossi, March 1996.

96-6 *Engineering Formal Requirements: an Analysis and Testing Method for Z Documents*, P. Ciancarini, S. Cimato, C. Mascolo, March 1996.

96-7 *Using Bayesian Belief Networks for the Automated Assessment of Students' Knowledge of Geometry Problem Solving Procedures*, M. Roccetti, P. Salomoni, March 1996.

96-8 *Virtual Interactions: An Investigation of the Dynamics of Sharing Graphs*, A. Asperti, C. Laneve, April 1996.

96-9 *Towards an Algebra of Actors*, M. Gaspari, April 1996.

96-10 *Mobile Petri Nets*, A. Asperti, N. Busi, May 1996.

96-11 *Communication Support for Critical Distributed Multimedia Applications: an Experimental Study*, F. Panzieri, M. Roccetti, May 1996.

96-12 *A Logic Coordination Language Based on the Chemical Metaphor*, P. Ciancarini, D. Fogli, M. Gaspari, July 1996.

96-13 *Towards Parallelization of Concurrent Systems*, F. Corradini, R. Gorrieri, D. Marchignoli, August 1996 (Revised December 1996).

96-14 *The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties*, R. Focardi, R. Gorrieri, August 1996.

96-15 *Jada: a Coordination Toolkit for Java*, P. Ciancarini, D. Rossi, October 1996.

96-16 *Fault Tolerance through View Synchrony in Partitionable Asynchronous Distributed Systems*, A. Montresor, December 1996.

96-17 *A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time*, M. Bernardo, R. Gorrieri, December 1996.

97-1 *Partitionable Group Membership: Specification and Algorithms*, Ö. Babaoğlu, R. Davoli, A. Montresor, January 1997.

A Truly Concurrent View of Linda Interprocess Communication

N. Busi **R. Gorrieri** **G. Zavattaro**

Technical Report UBLCS-97-2

February 1997

Abstract

Linda is a coordination language which provides communication by means of primitives allowing the insertion, reading and withdrawal of elements to and from a shared data space. This communication mechanism is referred to as generative communication and it has been used in many other recent concurrent languages (e.g. Shared-Prolog, LO, Gamma). We first introduce a process calculus which embeds all the Linda coordination primitives, even the `inp` *and* `readp` *operation which have not been considered by other papers on the semantics of Linda. Moreover, we introduce a truly concurrent semantics for Linda by providing our calculus with a net semantics based on contextual P/T nets. Such a kind of nets allows to model a high level of true concurrency because they embed the notion of "reading without consuming" on which the* `read` *operation of Linda is based. For example, in our approach the* parallel-read *(i.e. the simultaneous reading of the same message) is modeled.*

1 Introduction

Asynchronous communication realized by means of the insertion, reading, and withdrawal of elements to and from a shared multiset, is the peculiar feature of a family of coordination languages [GC92]. The most representative language of this family is the coordination language Linda [Gel85]. Linda provides interprocess communication via a medium called *Tuple Space* (TS for short), that is a shared memory which contains a set of messages that are produced by a set of processes. TS is accessible by every parallel process by means of three coordination primitives:

`out(Message)`: produces a message;
`read(Message)`: reads a message without consuming it;
`in(Message)`: reads and consumes a message;

and two predicates:

`readp(Message)`: looks for a message in TS; if the message is present, the boolean value $true$ is returned; $false$ otherwise;
`inp(Message)`: looks for a message in TS; if the message is present, it is removed and the boolean value $true$ is returned; $false$ otherwise.

The peculiar features of Linda interprocess communication can be listed as follows:

- A process can always insert a message in TS performing an `out` operation.
- A process can perform an `in` or a `read` operation only if the required message is in TS; if not, it blocks. A side effect of the execution of an `in` operation is the withdrawal of the read message.
- A process can always evaluate an `inp` or a `readp` predicate which returns a boolean value: they are the predicate forms of `in` and `read` respectively, in the sense that they look for the specified message in TS in the same way as `in` and `read` do, but if the message is not found they return the boolean value $false$ instead of blocking. If the message is found, their behaviour is the same as `in` and `read` and the value $true$ is returned.
- The insertion order of messages in TS has no influence on their reading order.
- Multiple occurrences of the same message can be in TS at the same time (TS is a multiset of messages).

This communication mechanism is said to be *generative* because a message generated by a process has an independent existence in TS until it is explicitly withdrawn. In fact, after its insertion in TS, a message becomes equally accessible to all processes, and it is bound to none.

Recently, several frameworks to reason formally about generative communication have been introduced [CJY95, DP96, CGZ96] but no one of them takes into account the `inp` and `readp` operations. Our aim is to analyze these operations too, and the idea is to adapt the process algebra introduced in [CGZ96] in order to deal with these new features. In [CGZ96] generative communication is embedded in a process algebra by introducing the following additions to CCS [Mil89]:

- The autonomous agent $\langle a \rangle$ is introduced to represent the sent message a.
- The prefix $\hat{a}$ denotes a message which can be sent. The execution of $\hat{a}$ consists of the addition of the agent $\langle a \rangle$ to the environment:
  $\hat{a}.P \xrightarrow{\tau} \langle a \rangle | P$
  The label is τ because this step, representing an `out` operation, is a local autonomous step of computation which does not depend on the environment.
- An extra prefix $\underline{a}$ is introduced to represent the request of reading message a without consuming it.
- The agent $\langle a \rangle$ can be consumed by an agent which performs an `in` operation:
  $\langle a \rangle \xrightarrow{\overline{a}} \mathbf{0}$
  and it can be read by an agent which performs a `read` operation:
  $\langle a \rangle \xrightarrow{\overline{\underline{a}}} \langle a \rangle$
  The labels $\overline{a}$ and $\overline{\underline{a}}$ represents the "complementary" actions for a and $\underline{a}$ respectively.

In order to deal with the `inp` and `readp` predicates we introduce a new constructor which consists in a sort of *if-then-else*. In fact, we use $a?P\_Q$ or $\underline{a}?P\_Q$ to denote a process which requires the

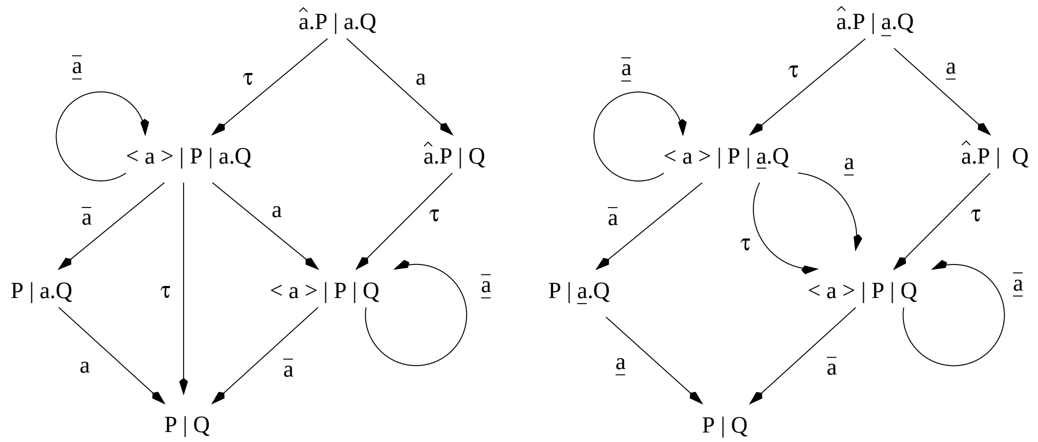

Figure 1. Generatively communicating processes.

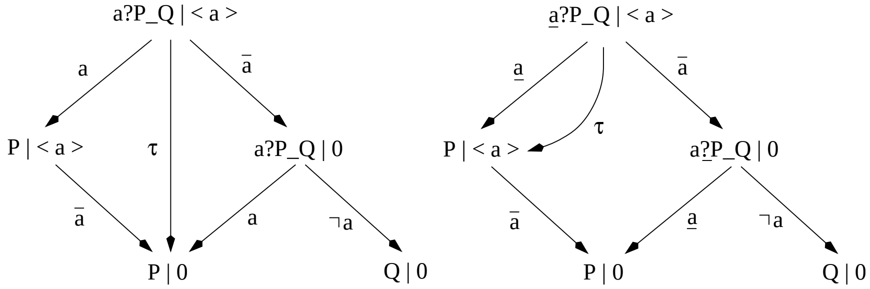

Figure 2. Processes performing `inp` or `readp` (self-loops labeled with $\underline{\overline{a}}$ are omitted).

message a to remove or simply read, respectively. If the message is found the agent behaves like P; if not, it becomes Q. The behaviour of the agent $\mu?P\_Q$ (where $\mu = a$ or $\mu = \underline{a}$) is described by the following items:

- The agent reads the required message by performing the following action:
  $\mu?P\_Q \xrightarrow{\mu} P$
- If the required message is not found the agent becomes Q by performing the action $\neg a$ which signals the absence of message a in the environment:
  $\mu?P\_Q \xrightarrow{\neg a} Q$
- An agent which performs a $\neg a$ operation can be composed in parallel only with agents which do not contain the message a, i.e. agents which are not able to perform action $\overline{a}$:
  $$\frac{P \xrightarrow{\neg a} P' \quad Q \stackrel{\overline{a}}{\not\longrightarrow}}{P|Q \xrightarrow{\neg a} P'|Q}$$
- If an agent which performs a $\neg a$ operation is restricted on message a then the operation $\neg a$ becomes a local step of computation (i.e. labeled with τ) because the search has been finished. In fact it is no more necessary to test the absence of message a because it becomes a local message name:
  $$\frac{P \xrightarrow{\neg a} P'}{P \setminus a \xrightarrow{\tau} P' \setminus a}$$

Figure 1 compares two different cases of generative communication between processes: in the first graph a reader which performs an `in` operation is considered, while the second graph describes the case of a `read` operation. The most important difference between the two cases is due to the behavior of the synchronization (i.e. the simultaneous execution of the complementary actions

P	$::=$	$\mathbf{0}$		null agent
	$\mid$	$\langle a \rangle$		message agent
	$\mid$	$\mu.P$		*prefix* operator
	$\mid$	$\mu?P\_Q$	$(\mu = a \vee \mu = \underline{a})$	*non-blocking input* operator
	$\mid$	$P + Q$		*choice* operator
	$\mid$	$P \mid Q$		*parallel* operator
	$\mid$	$P \setminus a$		*restriction* operator
	$\mid$	X		agent variable
	$\mid$	$rec\ X.P$		*recursion* operator

Table 1. Syntax.

a, $\overline{a}$ or $\underline{a}$, $\overline{\underline{a}}$). In fact, if the reader performs an `in` operation, the message is withdrawn, while if it executes a `read`, the agent $\langle a \rangle$ is not removed.

Figure 2 shows the behaviors of the agents $a?P\_Q|\langle a \rangle$ and $\underline{a}?P\_Q|\langle a \rangle$. It is interesting to see that agent $a?P\_Q|\langle a \rangle$ is not able to perform action $\neg a$ because of the presence of message a in the environment; only when the agent $\langle a \rangle$ is removed, executing $\overline{a}$, the action $\neg a$ can be performed. As for the `in` and `read` operations the only difference between the agents $a?P\_Q|\langle a \rangle$ and $\underline{a}?P\_Q|\langle a \rangle$ is due to the behaviour of the synchronization: in the case of the `inp` predicate the message is removed, otherwise it is kept.

We also introduce a truly concurrent semantics for generative communication by providing our calculus with a net semantics based on contextual P/T nets [MR95, BP95]. Such a kind of nets allows to model a high level of true concurrency because they embed the notion of "reading without consuming" on which the `read` operation is based. For example, in the framework that we introduce the *parallel-read* (i.e. the simultaneous reading of the same message) is allowed.

2 The Language and its Semantics

Let:

- $Message$, ranged over by a, b, etc., be a denumerable set of messages;
- $Prefix = \{a, \underline{a}, \hat{a} \mid a \in Message\} \cup \{\tau\}$, ranged over by μ, η, etc., be the set of prefixes;
- $Label = \{a, \underline{a}, \overline{a}, \overline{\underline{a}}, \neg a \mid a \in Message\} \cup \{\tau\}$, ranged over by α, β, etc., be the set of labels;
- $\overline{\cdot} : Label \longmapsto Label$ be a partial function such that $\overline{(a)} = \overline{a}$, $\overline{(\overline{a})} = a$, $\overline{(\underline{a})} = \overline{\underline{a}}$, $\overline{(\overline{\underline{a}})} = \underline{a}$;
- $n : Prefix \cup Label \longrightarrow Message \cup \{\tau\}$ be a total function such that $n(a) = n(\underline{a}) = n(\hat{a}) = n(\overline{a}) = n(\overline{\underline{a}}) = a$ and $n(\tau) = \tau$;
- $\mathcal{X}$, ranged over by X, Y, etc., be the set of agent variables.

The agent expressions, ranged over by P, Q, etc., are defined in Table 1. The null agent $\mathbf{0}$ and the class of terms $\langle a \rangle$ represent the possible elementary agents. Term $\mathbf{0}$ is the deadlocked agent (i.e. it is not able to perform any kind of action) whereas $\langle a \rangle$ represents a message a which has been inserted in TS so it is ready to be read or withdrawn. The *prefix operator* is used to define that an agent must perform an action before going on. There are four actions depending on the kind of prefixes: a, $\underline{a}$, $\hat{a}$ and τ corresponding to the `in`, `read`, `out` operation and local autonomous steps of computation respectively. The *non-blocking input* operator is used to represent the behavior of the evaluation of the `inp` and `readp` predicates. In fact, the agent $a?P\_Q$ and $\underline{a}?P\_Q$ requires

(in)	$\langle a\rangle \xrightarrow{\overline{a}} \mathbf{0}$	
(read)	$\langle a\rangle \xrightarrow{\underline{\overline{a}}} \langle a\rangle$	
(pref)	$\mu.P \xrightarrow{\mu} P$	$\mu \neq \hat{a}$
(out)	$\hat{a}.P \xrightarrow{\tau} \langle a\rangle \vert P$	
(found)	$\mu?P\_Q \xrightarrow{\mu} P$	
(not-found)	$\mu?P\_Q \xrightarrow{\neg n(\mu)} Q$	
(choice)	$\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	
(par)	$\dfrac{P \xrightarrow{\alpha} P'}{P \vert Q \xrightarrow{\alpha} P' \vert Q}$	$\alpha \neq \neg a$
(envCheck)	$\dfrac{P \xrightarrow{\neg a} P' \quad Q \not\xrightarrow{\overline{a}}}{P \vert Q \xrightarrow{\neg a} P' \vert Q}$	
(synch)	$\dfrac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \vert Q \xrightarrow{\tau} P' \vert Q'}$	
(restriction)	$\dfrac{P \xrightarrow{\alpha} P'}{P \setminus a \xrightarrow{\alpha} P' \setminus a}$	$n(\alpha) \neq a$
(localCheck)	$\dfrac{P \xrightarrow{\neg a} P'}{P \setminus a \xrightarrow{\tau} P' \setminus a}$	
(recursion)	$\dfrac{P[rec\, X.P/X] \xrightarrow{\alpha} P'}{rec\, X.P \xrightarrow{\alpha} P'}$	

Table 2. Operational semantics (symmetric rules omitted).

the message a to withdrawn or read respectively, and if the message is found it behaves like P, otherwise it becomes Q.

The meaning of the other operators is the usual one. The *choice* operator is used to represent a non-deterministic alternative choice between two combined agents. The *parallel* operator is used to combine agents which are able to perform actions in parallel and to synchronize on complementary actions. The *restriction* operator is used to define local actions. The *recursion* operator is used for the definition of recursive agents.

We say that X is bound in P if each occurrence of X is within some subexpression $rec\, X.Q$; X is also guarded if each occurrence in Q is within some subexpression $\mu.Q'$. We say that P is closed and guarded if only bound and guarded variables occur in it. Let $Agent$ be the set of closed and guarded terms. In the following only closed and guarded terms will be considered (i.e. P, Q, etc. will range only over $Agent$).

The operational semantics of our language is described by a *labeled transition system* ($Agent$, $Label$, $\longrightarrow$). The labeled relation $\longrightarrow \subseteq (Agent \times Label \times Agent)$ is the smallest one which satisfies the axioms and rules of Table 2 in which the symmetric rules w.r.t. (choice), (par), and (envCheck) are omitted for the sake of simplicity. The rules that we use are in the *tyft/tyxt* format [GV92] so a canonical system of transition relations agreeing with the rules exists and the strong bisimulation (only bisimulation in the following) is a congruence for all the operators.

Definition 2.1 A binary, symmetric relation $\mathcal{R}$ on $Agent$ is a *bisimulation* if $(P, Q) \in \mathcal{R}$ and $P \xrightarrow{\alpha} P'$ imply that there exists Q' s.t. $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{R}$.

Two agents P and Q are *bisimilar*, written $P \sim Q$, if there exists a bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$. ■

3 Truly Concurrent Semantics

In this section we introduce a truly concurrent semantics for generative communication by providing our calculus with a net semantics based on contextual P/T nets [MR95, BP95]. Such a kind of nets allows to show all the intended parallelism because they embed the notion of "reading without consuming" on which the `read` operation is based. For example, in the framework that we introduce the *parallel-read* (i.e. the simultaneous reading of the same message) is allowed.

The basic idea underlying the definition of an operational net semantics for a process algebra ([DDM88]) is to decompose a process P into a multiset of sequential components, which can be thought of as running in parallel. Each sequential component has a corresponding place in the net, and will be represented by a token in that place. Actions are represented by labeled transitions which consume and produce multisets of sequential components if the contextual conditions are satisfied.

We extend the approach to our language by representing tuples as tokens in the corresponding places in the net; in this way we can faithfully model the `read` operation on a tuple by a transition with a contextual arc that tests for presence of a token in the corresponding place. In the representation of the `inp` and `readp` predicates, we use inhibitor arcs that test a tuple place for absence (of tokens) in the transitions representing the selection of the else branch.

As nets are a very concrete distributed model, they offer more discriminating observability criterions. We study the so called step semantics, which permits to observe all the potential parallelism in the system; moreover, we describe the causal bisimulation, that gives a complete account of the dependencies among transitions in the net. We think that these more concrete semantics can be useful, because they define descriptions of systems closer to implementations, and can offer a better understanding of the system.

In the following subsection we report the definition of contextual P/T nets and of its step and causal semantics. In subsection 4.2 we formally define the net semantics of our language and a proof of the retrievability of the SOS semantics from the sequential execution of the net is provided. Finally, some examples illustrating the use of contextual arcs are reported.

3.1 Contextual P/T Nets

We recall simple Place/Transition nets without capacity constraints on places (see, e.g., [Gol90]). Then, we extend them with the contextual arcs (see, e.g., [BP95, Ris94]). Here we provide a characterization of this model which is convenient for our aims.

Definition 3.1 Given a set S, a *finite multiset* over S is a function $m : S \to \omega$ such that the set $dom(m) = \{s \in S \mid m(s) \neq 0\}$ is finite. The *multiplicity* of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S, denoted by $\mathcal{M}_{fin}(S)$, is ranged over by m. A multiset m such that $dom(m) = \emptyset$ is called *empty* . The set of all finite sets over S is denoted by $\wp_{fin}(S)$. If A is a finite subset of S, sometimes with abuse of notation we use A to denote the multiset m_A defined as follows: $m_A(s) =$ if $s \in A$ then 1 else 0.

We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$. The operator $\oplus$ denotes *multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes *multiset difference*: $m \setminus m'(s) = m(s) \dot{-} m'(s)$ where $i \dot{-} j = i - j$ if $i > j$, $i \dot{-} j = 0$ otherwise. The *scalar product*, $j \cdot m$, of a number j with m is $(j \cdot m)(s) = j \cdot (m(s))$. ■

Let f be a function on S. The function is extended to multisets over S in the following way: if $m \in \mathcal{M}_{fin}(S)$ then $(f(m))(s) = \sum_{s':f(s')=s} m(s')$.

Definition 3.2 A P/T net is a tuple $N = (S, Act, T)$ where S is the set of *places*, Act is the set of *labels* and $T \subseteq \mathcal{M}_{fin}(S) \times Act \times \mathcal{M}_{fin}(S)$ is the set of *transitions*. A finite multiset over the set S of places is called a *marking*. Given a marking m and a place s, we say that the place s contains $m(s)$ *tokens*.

A transition $t = (c, a, p)$ is usually written in the form $c \xrightarrow{a} p$. The marking c, usually denoted with ${}^\bullet t$, is called the *preset* of t and represents the tokens to be "consumed"; the marking p, usually denoted with $t^\bullet$, is called the *postset* of t and represents the tokens to be "produced"; a is called the *label* of t, sometimes denoted by $l(t)$. A transition t is *enabled* at m if ${}^\bullet t \subseteq m$. The execution of a transition t enabled at m produces the marking $m' = (m \setminus {}^\bullet t) \oplus t^\bullet$. This is written as $m[t\rangle m'$.

A *marked* P/T net is a tuple $N(m_0) = (S, Act, T, m_0)$, where (S, Act, T) is a P/T net and m_0 is a nonempty finite multiset over S, called the *initial marking*. ■

Definition 3.3 A *contextual* P/T net (CN net for short) is a tuple $N = (S, Act, T)$ where S is the set of places, Act the set of labels and $T \subseteq \mathcal{M}_{fin}(S) \times \wp_{fin}(S) \times \wp_{fin}(S) \times Act \times \mathcal{M}_{fin}(S)$ is the set of transitions.

A transition $t = (c, r, i, a, p)$ is usually written in the form $(c, r, i) \xrightarrow{a} p$. The set r, denoted with $\hat{t}$, is called the *contextual set* of t and represented the tokens to be "tested for presence"; the set i, denoted with ${}^\circ t$, is called the *inhibitor-set* of t and represents the tokens to be "tested for absence"; markings c and p are as above. This changes the definition of enabling: a transition t is enabled at m if ${}^\bullet t \oplus \hat{t} \subseteq m$ and $dom(m) \cap {}^\circ t = \emptyset$. Thus, any transition t for which ${}^\circ t \cap (dom({}^\bullet t) \cup \hat{t}) \neq \emptyset$ is called *blocked*, *unblocked* otherwise.

The execution of a transition t enabled at m producing the marking m', written $m[t\rangle m'$, is defined as above. ■

Definition 3.4 Let $N = (S, Act, T)$ be a (contextual) P/T net. The *interleaving marking graph* of N is $IMG_N = (\mathcal{M}_{fin}(S), \rightarrow, Act)$, where $\rightarrow \subseteq \mathcal{M}_{fin}(S) \times Act \times \mathcal{M}_{fin}(S)$ is defined by $m \xrightarrow{l(t)} m'$ iff there exists a transition $t \in T$ such that $m[t\rangle m'$. With $IMG_N(m)$ we denote the interleaving marking graph reachable from the initial marking m. ■

As a IMG is a labeled transition system, bisimulation equivalence can be defined on markings too; with $m \sim m'$ we denote that m and m' are bisimilar.

3.1.1 Step Semantics

A finite, non empty multiset over the set T is called a *step*. According to [MR95], if two transitions can happen in the same step then they can happen in either order. We have to check that not all tokens in a place tested for presence by (an occurrence of) a transition are consumed by the others and that (an occurrence of) a transition does not produce tokens in a place tested for absence by another.

A step G is enabled at m iff

- $m_1 \oplus m_3 \subseteq m$, where $m_1 = \bigoplus_t G(t) \cdot {}^\bullet t$ and $m_3 = \bigcup_{t \in dom(G)} \hat{t}$
- for all $t \in dom(G)$ ${}^\circ(t) \cap dom(m) = \emptyset$
- for all $t_1, t_2 \in dom(G)$ such that $t_1 = t_2 \Rightarrow G(t_1) \geq 2$ we have that $dom(t_1^\bullet) \cap {}^\circ t_2 = \emptyset$

The execution of a step G enabled at m produces the marking $m' = (m \setminus m_1) \oplus m_2$, where $m_2 = \bigoplus_t G(t) \cdot t^\bullet$. This is written as $m[G\rangle m'$.

The *step marking graph* of N is $SMG(N) = (\mathcal{M}_{fin}(S), \rightarrow, \mathcal{M}_{fin}(Act))$, where $\rightarrow \subseteq \mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(Act) \times \mathcal{M}_{fin}(S)$ is defined by $m \xrightarrow{A} m'$ iff there exists a step G such that $m[G\rangle m'$ and $A = l(G)$.

As a SMG is a labeled transition system, we can define a notion of (step) bisimulation equivalence on markings.

3.1.2 Causal Semantics

In [BP95] a causal semantics for contextual P/T nets is proposed, based on causal trees [DD89]. The current state of a net is represented by its marking, whereas the events that occur during the net execution are occurrences of its transitions. The standard immediate causes of an event consist of the set of events that produced the tokens it consumes; to record this information, we enrich the current state by decorating each token with its history, that is the event which produced it; to distinguish between different occurrences of the same transition, we record also the number of occurrences of each transition that have already happened. For a transition to fire, we need to check that at least one token is present in each positive contextual place and that each inhibiting place is empty (i.e. does not contain any token). So, besides the standard causes given by the decorations of the tokens it consumes, the event has other two types of causes: the (positive) contextual ones, given for each contextual place by the decoration of one of the tokens it contains, and the inhibiting ones, given by the set of events that have removed tokens from the inhibiting places. The information on the inhibiting causes is obtained by decorating each place with the set of events that have consumed tokens from it. The enriched state, containing information about the history of tokens, the number of transitions occurrences already happened and the set of events that consumed tokens from a place, is called configuration. The initial configuration of a net is obtained by setting to zero the numbers of occurrences of each transition and by decorating the tokens in the initial marking with the special symbol $\star$, meaning that it has not been produced by any event. When an event occurs, by looking at the histories of the tokens it consumes (or tests for presence) and to the set of events associated to each of its inhibiting places, we obtain information on its (immediate) causal dependencies. Obviously, the consumption of tokens decorated with $\star$ does not add causal dependencies. The (immediate causal) execution of an event is a transition from a configuration to another one, labeled with the event and the set of its immediate causal dependencies.

A causal tree is a tree labeled with pairs composed by an action and a set of relative pointers to all the predecessors that caused the present action. From the set of the (immediate) causal firing sequences of the net, we can easily construct the causal tree representing the causal execution of the net: first we perform a transitive closure of the immediate causal relation on events, then we abstract from events by replacing each event with the label of the transition of which it is an occurrence, and the set of events (representing its causes) with relative pointers. Causal bisimulation of two nets is reduced to bisimulation on the corresponding causal trees.

A formal definition of the causal semantics for contextual nets is reported in Appendix A.

3.2 Net Semantics

We need to introduce a new set of symbols, the *conflict* names, that are used to model the choice operator: an agent of the form $P + Q$ is interpreted as the parallel composition of kP and $\bar{k}Q$, where k is called a conflict name and $\bar{k}$ is its contrasting conflict name. Hence the two alternative agents are considered as parallel agents "decorated" with contrasting conflict names. The net semantics contains a place for any conflict name k, which, when holding at least one token, inhibits all the transitions starting from places/agents decorated by $\bar{k}$ (and symmetrically for $\bar{k}$, with the assumption that $\bar{\bar{k}} = k$). Formally, let $\mathcal{C}$ be a denumerable set of symbols disjoint from $Message \cup \{\tau\}$ and from $\mathcal{X}$. Let $\bar{\mathcal{C}} = \{\bar{k} \mid k \in \mathcal{C}\}$ and $Con = \mathcal{C} \cup \bar{\mathcal{C}}$ the set of conflict names. Con is ranged over by $k, h, \ldots$, with the assumption that $\bar{\bar{k}} = k$. A *conflict set* is a finite subset of Con, ranged over by $I, J, \ldots$; $\bar{I} = \{\bar{k} \mid k \in I\}$. When I is a singleton, we drop the set brackets for notational convenience.

To model the restriction operator, we partition the set $Message$ into two sets, visible messages (denoted by $VMessage$) and invisible ones ($IMessage$). Only visible messages can occur in the label of an observable (not τ) transition, whereas the invisible ones cannot. We assume that an agent initially uses only names from $VMessage$. The denotation of an agent $P\backslash a$ is obtained from the denotation of P by renaming the free occurrences (i.e. not within the scope of an inner

$dec(\mathbf{0}) = \emptyset$	
$dec(\langle a \rangle) = \{\langle a \rangle\}$	
$dec(\mu.P) = \{\emptyset\mu.P\}$	
$dec(\mu?P\_Q) = \{\emptyset\mu?P\_Q\}$	
$dec(P + Q) = k\, dec(P) \oplus \bar{k}\, dec(Q)$	k new
$dec(P \vert Q) = dec(P) \oplus dec(Q)$	
$dec(P\backslash a) = dec(P)\{b/a\}$	b new $\wedge\ b \in IMessage$
$dec(rec\ X.P) = dec(P\{rec\ X.P/X\})$	

Table 3. Decomposition function

restriction $\backslash a$) of a with an invisible name b; an action whose name is invisible can be performed only in a synchronization. As, unlike [DDM88], we forget the structure of parallel composition of agents, we always need fresh invisible names to rename the restriction binders.

The set S of places is defined as follows:

$$\begin{array}{rcl} S & = & \{\langle a \rangle \mid a \in Message\} \\ & \cup & \{I\mu.P \mid I \subseteq Con \wedge \mu.P \in Agent\} \\ & \cup & \{I\mu?P\_Q \mid I \subseteq Con \wedge \mu?P\_Q \in Agent\} \\ & \cup & Con \end{array}$$

$\langle a \rangle$ is a message agent, whereas $I\mu.P$ and $I\mu?P\_Q$ represent sequential processes with conflict set I (the conflict set associated to a sequential process is sometimes omitted when empty).

Every conflict name k is a place (with the same name). k is used to prevent the execution of any transition from any sequential process $I\mu.P$ (or $I\mu?P\_Q$) such that $\bar{k} \in I$. To better describe the implementation of the distributed choice, assume that P and Q are sequential processes; $P + Q$ is interpreted as the parallel composition of $\{k\}P$ and $\{\bar{k}\}Q$; with the execution of an action from $\{k\}P$ we produce a token in place k, hence preventing a later execution of an action from $\{\bar{k}\}Q$.

In order to define the decomposition function dec which associates a finite multiset on S to each process, we need some auxiliary definitions.

The operator $I(\_)$ is defined on places as follows: $I(\langle a \rangle) = \langle a \rangle$, $I(J\mu.P) = (I \cup J)\mu.P$, $I(J\mu?P\_Q) = (I \cup J)\mu?P\_Q$ and $I(k) = k$.

The substitution of a message a with an invisible message b is defined on agents in the usual way, with the restriction operator acting as a binder, i.e. $(P\backslash a)\{b/a\} = P\backslash a$ and $(P\backslash c)\{b/a\} = (P\{b/a\})\backslash c$ for $c \neq a$. Note that restricted names are always visible ones, so we do not need alpha conversion to avoid name clashes.

The substitution is defined on places as follows: $(\langle c \rangle)\{b/a\} = \langle c\{b/a\} \rangle$, $(I\mu.P)\{b/a\} = I((\mu.P)\{b/a\})$, $(I\mu?P\_Q)\{b/a\} = I((\mu?P\_Q)\{b/a\})$ and $(k)\{b/a\} = k$.

The set $c(s)$ of *conflicts of place* s is defined as:

$$c(I\mu.P) = c(I\mu?P\_Q) = \{k \in \mathcal{C} \mid k \in I \cup \bar{I}\} \qquad c(k) = \begin{cases} \{k\} & \text{if } k \in \mathcal{C} \\ \{\bar{k}\} & \text{otherwise} \end{cases} \qquad c(\langle a \rangle) = \emptyset$$

and extended to markings as:

$$c(m) = \bigcup_{s \in dom(m)} c(s)$$

(1)	$(\{\langle a\rangle\},\emptyset,\emptyset) \xrightarrow{\overline{a}} \emptyset$	$a \in VMessage$
(2)	$(\emptyset,\{\langle a\rangle\},\emptyset) \xrightarrow{\overline{\underline{a}}} \emptyset$	$a \in VMessage$
(3)	$(\{I\mu.P\},\emptyset,\bar{I}) \xrightarrow{\mu} dec(P) \oplus I$	$\mu \neq \hat{a} \wedge n(\mu) \in VMessage \cup \{\tau\}$
(4)	$(\{I\hat{a}.P\},\emptyset,\bar{I}) \xrightarrow{\tau} \{\langle a\rangle\} \oplus dec(P) \oplus I$	
(5)	$(\{I\mu?P\_Q\},\emptyset,\bar{I}) \xrightarrow{\mu} dec(P) \oplus I$	$n(\mu) \in VMessage$
(6)	$(\{I\mu?P\_Q\},\emptyset,\bar{I} \cup \{\langle a\rangle\}) \xrightarrow{\neg a} dec(Q) \oplus I$	$a = n(\mu) \in VMessage$
(7)	$(\{I\mu?P\_Q\},\emptyset,\bar{I} \cup \{\langle a\rangle\}) \xrightarrow{\tau} dec(Q) \oplus I$	$a = n(\mu) \in IMessage$
(8)	$(\{Ia.P,\langle a\rangle\},\emptyset,\bar{I}) \xrightarrow{\tau} dec(P) \oplus I$	
(9)	$(\{Ia?P\_Q,\langle a\rangle\},\emptyset,\bar{I}) \xrightarrow{\tau} dec(P) \oplus I$	
(10)	$(\{I\underline{a}.P\},\{\langle a\rangle\},\bar{I}) \xrightarrow{\tau} dec(P) \oplus I$	
(11)	$(\{I\underline{a}?P\_Q\},\{\langle a\rangle\},\bar{I}) \xrightarrow{\tau} dec(P) \oplus I$	

Table 4. Axiom schemata for transitions

The set $n(s)$ of *message names of place* s is defined as:

$$\begin{aligned} &n(I\mu.P) = n(\mu.p)\\ &n(I\mu?P\_Q) = n(\mu?P\_Q)\\ &n(k) = \emptyset\\ &n(\langle a\rangle) = \{a\}\end{aligned}$$

and extended to markings as:

$$n(m) = \bigcup_{s \in dom(m)} n(s)$$

The set $i(m)$ of *invisible* message names in marking m is defined as follows:

$$i(m) = n(m) \cap IMessage$$

For the sake of simplicity, we often write the functions above with multiple arguments, meaning the union of the applications to each argument. E.g., $i(m,m')$ means $i(m) \cup i(m')$.

Function $dec : Agent \to \mathcal{M}_{fin}(S)$, which defines the decomposition of agents into markings, is reported in Table 3. Agent $\mathbf{0}$ generates no tokens. The decomposition of $\langle a\rangle$ produces one token in the place $\langle a\rangle$. The decomposition of the sequential processes $\mu.P$ and $\mu?P\_Q$ produces one token in the corresponding place with the empty set of conflicts. For alternative composition, the agent $P+Q$ is turned into the multiset union of the markings for P and Q, where each place of $dec(P)$ is decorated by the singleton $\{k\}$ and, symmetrically, each place in $dec(Q)$ is decorated by $\{\bar{k}\}$. Parallel composition is interpreted as multiset union. The decomposition of a restricted agent $P\backslash a$ is the multiset obtained from the decomposition of P where a new name $b \in IMessage$ is substituted for the bound name a. Finally, a recursive process is first unwound once and then decomposed. Note that function dec is defined up to renaming of conflicts and invisible names, because the actual choice of the *new* conflict k and of the *new* invisible name b is inessential.

The net for our language is the triple $N = (S, Label, T)$, where the set T of net transitions is the least set generated by the axiom schemata reported in Table 4.

The first two axioms deal with observable actions performed by a message agent: to perform an output action, we have to consume a token from the place of the message agent, whereas for a non-consuming output we only test that place for presence.

Axiom (3) is for non-output prefixes: if no token is present in the places of the contrasting conflict names, then the transition produces, besides the tokens in $dec(P)$, also one token in each place of the conflicts in I. In this way, any other sequential process decorated with a conflict in $\bar{I}$ is blocked forever.

Axiom (4) deals with output prefix: an invisible action is performed and a token in the message agent corresponding to the prefix name is produced. Note that this axiom can be applied independently of the status (visible or invisible) of the prefix name.

In axiom (5) an observable action is performed by a non–blocking operation: its definition is similar to axiom (3).

In axioms (6) and (7) the message agent asked by a non-blocking operation is tested for absence: if the test is successful, the else branch of the operation is selected. Note that each transition in T labeled with an observable action contains a visible name.

Axioms (8) and (9) describe the withdrawal of a message by a sequential process: if a token is present in the place corresponding to the required message agent, then the sequential process consumes it and evolves.

Finally, axioms (10) and (11) deal with the reading of a message.

Note that the conflict set is not necessary for tuples, because of the condition on the occurrence of tuples inside the operands of a choice imposed on processes. In fact, the following proposition holds for processes:

Proposition 3.5 Let $P + Q$ be a process. If $\langle a \rangle \in dom(dec(P) \oplus dec(Q))$ then $a \in IMessage$. ■

If a tuple $\langle a \rangle$ occurs in the decomposition of $P + Q$, then its name is invisible; this means that the tuple occurs inside a subterm of the form $R \backslash a$. The only transitions in which the tuple can be tested or consumed are those generated by axioms (8)–(11); these axioms consume also a sequential agent $I\mu.P_1$ (or $I\mu?P_1\_Q_1$) such that $n(\mu) = a$; this means that this agent is produced by the decomposition of a subterm that lies inside the same restriction of the tuple; this implies that the sequential agent is in conflict with the same agents that are in conflict with the tuple, so the conflict set of the sequential agent is sufficient to disable all elements in conflict with the tuple.

Given a process P, the net associated to P is the subnet of N reachable from the initial marking $dec(P)$.

3.3 Retrievability

The following theorems compare the interleaving semantics for an agent P with the interleaving marking graph of the net N marked with the initial marking $dec(P)$. Then, the final corollary states that our net translation is fully abstract w.r.t. interleaving bisimulation equivalence. A sketch of the retrievability proof can be found in Appendix B.

Theorem 3.6 Let P be a process.
If $P \xrightarrow{\alpha} P'$, then there exists a marking m such that $dec(P) \xrightarrow{\alpha} m$ and $m \sim dec(P')$. ■

Theorem 3.7 Let P be a process.
If $dec(P) \xrightarrow{\alpha} m$, then there exists a process P' such that $P \xrightarrow{\alpha} P'$ and $m \sim dec(P')$. ■

Corollary 3.8 Let P and Q be processes. Then we have that $P \sim Q$ if and only if $dec(P) \sim dec(Q)$. ■

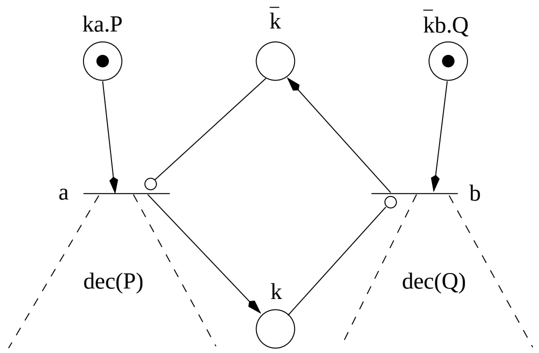

Figure 3. Net semantics of the agent $a.P + b.Q$.

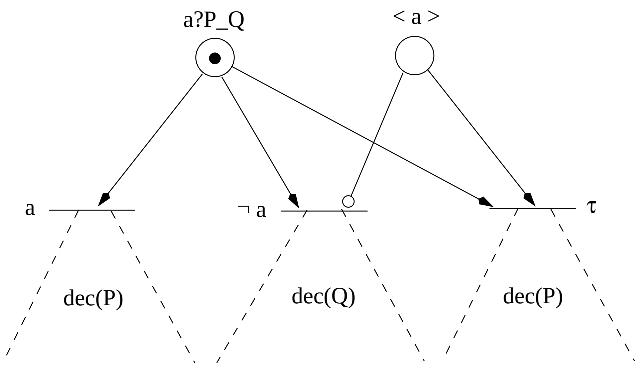

Figure 4. Net semantics of the agent $a?P\_Q|\langle a\rangle$.

4 Examples

In this section we analyze by examples the advantages of using contextual P/T nets instead of standard nets without inhibitors and contextual arcs. The inhibitor and contextual arcs can be graphically distinguished from standard arcs (denoted by lines with arrows) because they are denoted by lines with a small circle at the end and simple lines (with no circle and no arrow), respectively.

4.1 Inhibitor arcs

The inhibitor arcs are used in our net semantics in order to represent the behavior of the alternative choice operator and for testing the absence of a certain message in TS during the evaluation of the `inp` and `readp` predicates.

The net semantics of the agent $a.P + b.Q$, presented in Figure 3, shows how the inhibitor arcs are used to ensure that only one of the actions a and b is fired. In fact, the transitions a and b are both enabled but they cannot happen in the same step. When one of them is fired then a token is inserted in the conflict place which disallows the other one.

Figure 4 shows the net for the agent $a?P\_Q|\langle a\rangle$. There are three possible transitions in which the place labeled with $a?P\_Q$ is involved: a which consists in the local execution of action a (no other places are involved), $\neg a$ which can be fired only if the message a is not present (i.e. only if no tokens are in the place labeled $\langle a\rangle$) and finally τ representing the reading of the message a by the agent $a?P\_Q$. The net semantics for $\underline{a}?P\_Q|\langle a\rangle$ is very similar, except that the transition τ does not remove tokens from the place labeled with $\langle a\rangle$ (i.e. between place $\langle a\rangle$ and transition τ a contextual arc is used instead of the standard one of Figure 4).

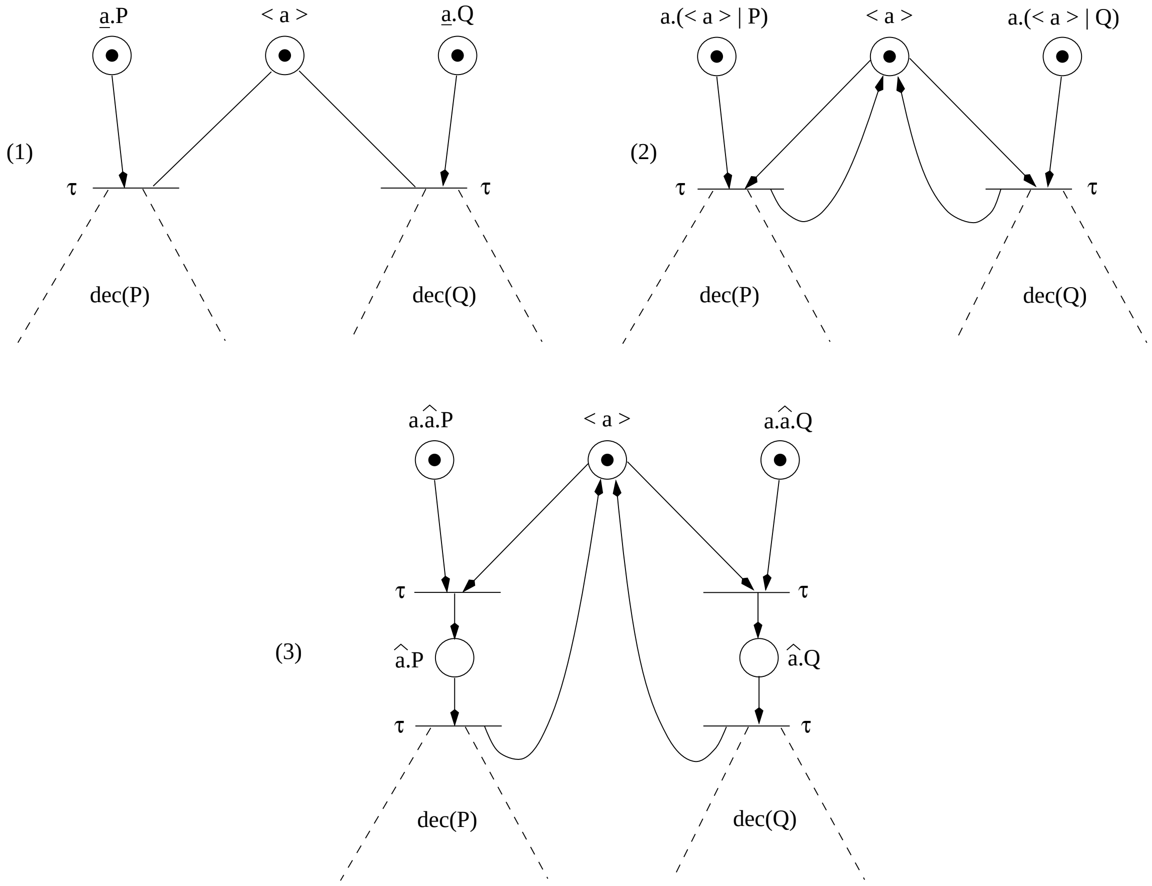

Figure 5. Net interpretations for concurrent reading operations.

4.2 Contextual arcs

Contextual arcs embed the notion of reading without consuming on which the `read` operator (and also the `readp` predicate) are based. By using such a kind of arcs it is possible to capture a high level of true concurrency. The most interesting example is due to the case of the *parallel-read*, i.e. the simultaneous execution of `read` operations on the same message. In other papers on the semantics of Linda such a kind of potential parallelism is not taken into account.

Figure 5 shows several net semantics for two concurrent reading operations executed on the message a, in particular we consider the agent $(\underline{a}.P|\underline{a}.Q|\langle a\rangle)\setminus a$. The first net is obtained by using our approach: contextual arcs allow the firing of the two transitions representing the execution of the two reading operations in the same step.

The second one is the net semantics for Linda presented in [CJY95]: the two transitions, even if both allowed, cannot be executed simultaneously. In fact, in that paper the execution of a read prefix on the message a consists of the consumption and the following insertion of one token in the place labeled with $\langle a\rangle$, i.e., $\underline{a}.P$ is considered to be the same as $a.(\langle a\rangle|P)$.

The third net is obtained by applying the standard net semantics of CCS [DDM88] to the approach for the representation of the `read` operation used in [DP96]. In that paper the `read` operation is obtained by using the `in` and `out` operations in the sense that the agent $\underline{a}.P$ has the same behaviour of $a.\hat{a}.P$. Its standard net semantics is similar to the one proposed in [CJY95] except that between the consumption of the token in the place labeled $\langle a\rangle$ and its new insertion, another transition labeled with τ must be fired.

References

[BP95] N. Busi and G.M. Pinna. A Causal Semantics for Contextual P/T nets. In Proc. *ICTCS'95*, 311–325, World Scientific, 1995.

[CGZ96] P. Ciancarini, R. Gorrieri, and G. Zavattaro. Towards a Calculus for Generative Communication. In Proc. *FMOODS'96*, 283–297, Chapman & Hall, 1996.

[CJY95] P. Ciancarini, K.K. Jensen, and D. Yankelevich. On the Operational Semantics of a Coordination Language. In *Object-Based Models and Languages for Concurrent Systems*, LNCS 924, 77-106, Springer–Verlag, 1995.

[DD89] Ph. Darondeau and P. Degano. Causal Trees. in Proc. *ICALP'89*, LNCS 372, 234–248, Springer–Verlag, 1989.

[DDM88] P. Degano, R. De Nicola, and U. Montanari. A Distributed Operational Semantics for CCS based on C/E Systems. *Acta Informatica*, 26:59–91, 1988.

[DP96] R. De Nicola and R. Pugliese. A Process Algebra based on Linda. In Proc. *Coordination'96*, LNCS 1061, 160-178, Springer–Verlag, 1996.

[GC92] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):97–107, 1992.

[Gel85] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[Gol90] U. Goltz. CCS and Petri Nets. LNCS 469, 334–357, Springer–Verlag, 1990.

[GV92] J. F. Groote and F. Vaandrager. Structured Operational Semantics and Bisimulation as a Congruence. *Information and Computation*, 100:202–260, 1992.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice–Hall, 1989.

[MR95] U. Montanari and F. Rossi. Contextual Nets. *Acta Informatica*, 32(6):545–596, 1995.

[Ris94] G. Ristori. Modelling Systems with Shared Resources via Petri Nets. Ph.D. Thesis, University of Pisa, TD-5/94, 1994.

A Causal Semantics for Contextual P/T Nets

Given a marked contextual net, we show how to generate a causal tree [DD89].

The construction proceeds by first defining types of tokens: they are essentially a decoration to a token that records some information about the way it has been generated, namely the occurrence of the transition which produced it. A place does not contain simply a set of tokens, rather a multiset of token types, because each individual token remembers its origin. Then, we introduce a notion of configuration of a marked net which essentially defines three pieces of information. The first is the present marking of the net, where the tokens are decorated by their type/history. The second is a function which associates to each place the set of transitions which have consumed tokens from that place; it will be useful because we consider that if a transition t consumes a token from the inhibitor set of a transition t', then t causes t'. The third piece of information is a counter of occurrences of transitions.

Definition A.1 Let $N = (S, Act, T, m_0)$ be a marked contextual P/T net. The set of *token types* is $\Theta = (T \times \omega^+) \cup \{*\}$, ranged over by θ, where (t, i) is the type of tokens produced by the i-th occurrence of transition t and $*$ is the type of tokens in the initial marking.

A *configuration* γ of a net is a triple (p, e, o), where

- $p : S \to \Theta \to \omega$ describes for each place the number of tokens of each type it contains.
- $e : S \to \wp(T \times \omega^+)$ describes for each place the set of (occurrences of) transitions which have consumed tokens from that place.
- $o : T \to \omega$ defines the number of occurrences of each transition.

The *initial configuration* of the net is $\gamma_0 = (p_0, e_0, o_0)$, where

$$p_0(s)(\theta) = \begin{cases} m_0(s) & \text{if } \theta = * \\ 0 & \text{otherwise} \end{cases} \qquad e_0(s) = 0 \qquad o_0(t) = 0 \qquad \blacksquare$$

With $\gamma[t^i, C > \gamma'$ we denote the firing of transition t from configuration γ to configuration γ'. Actually, it is the i-th time that transition t is fired. Set C records the immediate causes for the firing of this occurrence; its elements are occurrences of transitions.

Definition A.2 The rule for the i-causal firing rule is as follows: $(p, e, o)[t^i, C > (p', e', o')$ if and only if

- $\exists p_1 \subseteq p$ such that for all $s \in S$ ${}^\bullet t(s) = \sum_\theta p_1(s)(\theta)$
- $\exists p_2 \subseteq p$ such that for all $s \in S$ $\sum_\theta p_2(s)(\theta) = \begin{cases} 1 & \text{if } s \in \hat{t} \\ 0 & \text{otherwise} \end{cases}$
- $p_1 \oplus p_2 \subseteq p$
- $\forall s \in {}^\circ t$ $\sum_\theta p(s)(\theta) = 0$
- $o(t) = i - 1$
- $C = C_1 \cup C_2 \cup C_3$, where
  $C_1 = \{(t, i) \mid \exists s : p_1(s)(t, i) > 0\}$,
  $C_2 = \{(t, i) \mid \exists s : p_2(s)(t, i) > 0\}$ and
  $C_3 = \bigcup_{s \in {}^\circ t} e(s)$
- $p' = (p \setminus p_1) \oplus p_3$, where $p_3(s)(\theta) = \begin{cases} t^\bullet(s) & \text{if } \theta = (t, i) \\ 0 & \text{otherwise} \end{cases}$
- $e'(s) = \begin{cases} e(s) \cup \{(t, i)\} & \text{if } s \in {}^\bullet t \setminus t^\bullet \\ e(s) & \text{otherwise} \end{cases}$
- $o'(u) = \begin{cases} i & \text{if } u = t \\ o(u) & \text{otherwise} \end{cases}$ ■

An *i-causal firing sequence* (CFS) is defined inductively as follows:

- γ_0 is a CFS;
- if $\gamma_0[(\tau_1, C_1\rangle\gamma_1 \ldots [\tau_{n-1}, C_{n-1}\rangle\gamma_{n-1}$ is a CFS and $\gamma_{n-1}[\tau_n, C_n\rangle\gamma_n$ then
  $\gamma_0[\tau_1, C_1\rangle\gamma_1 \ldots [\tau_{n-1}, C_{n-1}\rangle\gamma_{n-1}$
  $[\tau_n, C_n\rangle\gamma_n$ is a CFS.

The set of i-causal firing sequences of a net N is denoted by $CFS(N)$.

The causal dependencies produce a relation between events whose transitive closure is a partial order.

Definition A.3 Let $\gamma_0[\tau_1, C_1\rangle\gamma_1 \ldots [\tau_n, C_n\rangle\gamma_n$ be an i-causal firing sequence. We define the relation $\prec$ as $\tau_i \prec \tau_j$ iff $\tau_i \in C_j$. ■

Proposition A.4 $\prec^+$ is a (strict) partial order. ■

Causal trees [DD89] are trees labeled with pairs (a, I), where a is an action and I is a set of relative pointers to all the predecessors which caused the present action a. We associate a causal tree to the set of the i-causal firing sequences $CFS(N)$.

Definition A.5 A *causal tree* over *Act* is a tree (V, A, f), where

- V is a set of nodes;
- $A \subseteq V \times V$ is a set of arcs;
- $f : A \to Act \times \wp_{fin}(\omega)$ is a labeling function. ■

Definition A.6 The causal tree of a net N is the tree $CT(N) = (V, A, f)$ defined as follows:

- $V = CFS(N)$
- $A = \{(\sigma, \sigma[\tau, C\rangle\gamma) \mid \sigma, \sigma[\tau, C\rangle\gamma \in CFS(N)\}$
- Let $\sigma = \gamma_0[\tau_1, C_1\rangle\gamma_1 \ldots [\tau_{n-1}, C_{n-1}\rangle\gamma_{n-1}$; then we have that $f(\sigma, \sigma[\tau_n, C_n\rangle\gamma_n) = (l(\tau_n), \{n-i \mid \tau_i \prec^+ \tau_n\})$. ■

With $\sigma \xrightarrow{a,I} \sigma'$ we mean that there exists an arc labeled with (a, I) between the nodes σ and σ'.

Definition A.7 Given two (marked) nets N_1 and N_2, a binary relation $R \subseteq CFS(N_1) \times CFS(N_2)$ is a *causal bisimulation* if:

- $(\gamma_{1,0}, \gamma_{2,0}) \in R$, where $\gamma_{i,0}$ is the initial configuration of N_i for $i = 1, 2$
- If $(\sigma_1, \sigma_2) \in R$ and $\sigma_1 \xrightarrow{a,I} \sigma_1'$ there exists then σ_2' such that $\sigma_2 \xrightarrow{a,I} \sigma_2'$ and $(\sigma_1', \sigma_2') \in R$.
- If $(\sigma_1, \sigma_2) \in R$ and $\sigma_2 \xrightarrow{a,I} \sigma_2'$ there exists then σ_1' such that $\sigma_1 \xrightarrow{a,I} \sigma_1'$ and $(\sigma_1', \sigma_2') \in R$.

The nets N_1 and N_2 are *causal bisimilar* ($N_1 \sim_c N_2$) iff there exists a causal bisimulation R between them. ■

B Retrievability

We sketch the proof of the retrievability of the interleaving semantics for an agent P from the interleaving marking graph of the net N marked with the initial marking $dec(P)$.

We recall a measure on processes, that will be used in the proof by induction of theorem B.9:

Definition B.1 The complexity of a process is defined in the following way:

$$complexity(\mathbf{0}) = complexity(\langle a \rangle) = complexity(\mu.P) = complexity(\mu?P\_Q) = 1$$
$$complexity(P + Q) = complexity(P|Q) = complexity(P) + complexity(Q) + 1$$
$$complexity(P \backslash a) = complexity(rec\ X.P) = complexity(P) + 1$$

Note that a process is a closed and guarded term, thus the definition of the complexity of X is useless. ■

The following observations and propositions are preparatory for the proof of the main result. The two theorems compare the transition system with the interleaving marking graph. Then, the final corollary states that our net translation is fully abstract w.r.t. interleaving bisimulation equivalence.

Proposition B.2 Given a marking m, the following hold:

- if $m \xrightarrow{\alpha}$ then $\alpha = \tau$ or $\alpha \in VMessage$.
- $\langle a \rangle \in dom(m) \cap VMessage$ iff $m \xrightarrow{\overline{a}}$. ■

Proposition B.3 Let P be a process. $P \xrightarrow{\overline{a}}$ iff $\langle a \rangle \in dom(dec(P)) \cap VMessage$. ■

Proposition B.4 Let m, m' and m'' be markings. If $k \notin c(m, m', m'')$, $c(m, m') \cap c(m'') = \emptyset$, $i(m, m') \cap i(m'') = \emptyset$ and $\langle a \rangle \in dom(m'') \Rightarrow a \in IMessage$, then $m \oplus \{k\} \oplus km' \oplus \bar{k}m'' \sim m \oplus m'$
Proof: The relation $\mathcal{R} = \{(m \oplus j \cdot \{k\} \oplus km' \oplus \bar{k}m'', m \oplus m') \mid j > 0$ and m, m', m'' are markings$\}$ is a bisimulation on markings. ■

Proposition B.5 Let m_1, m_2, m_1' and m_2' be markings. If $c(m_j) \cap c(m_j') = \emptyset$ and $i(m_j) \cap i(m_j') = \emptyset$ for $j = 1, 2$, $m_1 \sim m_2$ and $m_1' \sim m_2'$ then $m_1 \oplus m_1' \sim m_2 \oplus m_2'$.
Proof: The relation $\mathcal{R} = \{(m_1 \oplus m_1', m_2 \oplus m_2') \mid m_1, m_1', m_2, m_2'$ are markings, $(c(m_j) \cap c(m_j') = \emptyset) \wedge (i(m_j) \cap i(m_j') = \emptyset)$ for $i = 1, 2$, $m_1 \sim m_2$ and $m_1' \sim m_2'\}$ is a bisimulation on markings. ■

Proposition B.6 Let P be a process and b a fresh name. Then $dec(P\{b/a\}) = dec(P)\{b/a\}$, up to renaming of fresh conflicts and invisible names introduced during the decomposition. ■

Proposition B.7 Let m and m' be markings and b a fresh name. If $m \sim m'$ then we have that $m\{b/a\} \sim m'\{b/a\}$. ■

Theorem B.8 Let P be a process.

If $P \xrightarrow{\alpha} P'$, then there exists a marking m such that $dec(P) \xrightarrow{\alpha} m$ and $m \sim dec(P')$.

Proof: By induction on the proof of transition $P \xrightarrow{\alpha} P'$. We show the proof for two cases only, to give an idea of how the complete proof is.

- The transition is an instance of the axiom $\langle a \rangle \xrightarrow{\overline{a}} \mathbf{0}$: then $P = \langle a \rangle$, $\alpha = \overline{a}$ and $P' = \mathbf{0}$. We have $dec(\langle a \rangle) = \{\langle a \rangle\}$; the names occurring in agents are visible, so $a \in VMessage$; by axiom (1) for net transitions we obtain
$dec(P) = dec(\langle a \rangle) = \{\langle a \rangle\} \xrightarrow{\overline{a}} \emptyset = dec(\mathbf{0}) = dec(P')$.
- The last rule in the proof is $(Q \xrightarrow{\alpha} Q') \Rightarrow (Q|R \xrightarrow{\alpha} Q'|R)$, and $\alpha \neq \neg a$: then $P = Q|R$ and $P' = Q'|R$. From $Q \xrightarrow{\alpha} Q'$, by inductive hypothesis we obtain $dec(Q) \xrightarrow{\alpha} m \sim dec(Q')$. For side condition $\alpha \neq \neg a$, we have that this transition cannot be generated by axiom (6) for net transitions. The proof proceeds by case analysis on the remaining axioms. We analyse only the case of axiom (3). If the transition $dec(Q) \xrightarrow{\alpha} m \sim dec(Q')$ is obtained from axiom (3), then $dec(Q)$ contains a sequential process $I\mu.Q_1$, i.e. $dec(Q) = \{I\mu.Q_1\} \oplus m_1$; the set of inhibitor arcs of the transition is $\bar{I}$, from which follows that $\bar{I} \cap m_1 = \emptyset$. The side condition of axiom (3) ensures that $\mu \neq \hat{a}$ and $n(\mu) \in VMessage \cup \{\tau\}$. Finally, $m = dec(Q_1) \oplus I \oplus m_1$. We have that $dec(P) = dec(Q|R) = dec(Q) \oplus dec(R) = \{I\mu.Q_1\} \oplus m_1 \oplus dec(R)$. We use a fresh conflict name each time the decomposition of the choice composition of two processes is performed, and this ensures that $\bar{I} \cap dom(dec(R)) = \emptyset$. We can then apply axiom (3) to the marking $dec(P)$, thus obtaining $dec(P) = \{I\mu.Q_1\} \oplus m_1 \oplus dec(R) \xrightarrow{\mu} dec(Q_1) \oplus I \oplus m_1 \oplus dec(R) = m \oplus dec(R)$. We have that $dec(Q_1) \oplus I \oplus m_1 = m \sim dec(Q')$. For the above remark on fresh conflict names, which also holds for fresh invisible names, we can apply Proposition B.5 to markings m, $dec(Q')$, $dec(R)$ and $dec(R)$ and obtain $m \oplus dec(R) \sim dec(Q') \oplus dec(R) = dec(Q'|R) = dec(P')$. ■

Theorem B.9 Let P be a process.

If $dec(P) \xrightarrow{\alpha} m$, then there exists a process P' such that $P \xrightarrow{\alpha} P'$ and $m \sim dec(P')$.

Proof: By induction on the complexity of P. We show the proof for two cases only.

- $P = \mu.Q$. We have that $dec(\mu.Q) = \{\emptyset\mu.Q\}$. If $\mu \neq \hat{a}$, then the transition is an instance of axiom (3): $\alpha = \mu$ and $m = dec(Q)$. The transition $\mu.Q \xrightarrow{\mu} Q$ is an instance of an axiom of the SOS interleaving semantics (whose side condition $\mu \neq \hat{a}$ is satisfied). If $\mu = \hat{a}$, then the transition is an instance of axiom (4): $\alpha = \tau$ and $m = \{\langle a \rangle\} \oplus dec(Q)$. The transition $\hat{a}.Q \xrightarrow{\tau} \langle a \rangle|Q$ is an instance of an axiom of the SOS interleaving semantics; moreover, $dec(\langle a \rangle|Q) = dec(\langle a \rangle) \oplus dec(Q) = \{\langle a \rangle\} \oplus dec(Q) = m$.
- $P = rec\ X.Q$. We have that $dec(rec\ X.Q) = dec(Q\{rec\ X.Q/X\})$. The set of agents consists of closed and guarded terms, so the complexity of $Q\{rec\ X.Q/X\}$ is strictly less than the complexity of $rec\ X.Q$. By inductive hypothesis, we have that exists Q' such that $Q\{rec\ X.Q/X\} \xrightarrow{\alpha} Q'$ and $dec(Q') \sim m$. By applying the rule for recursion of the SOS interleaving semantics, we obtain that $rec\ X.Q \xrightarrow{\alpha} Q'$. ■

Corollary B.10 Let P and Q be processes. Then $P \sim Q$ if and only if $dec(P) \sim dec(Q)$. ■