On the Expressiveness of Linda Coordination Primitives*

Nadia Busi   Roberto Gorrieri   Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura Anteo Zamboni 7, I-40127 Bologna, Italy

e-mail: {busi,gorrieri,zavattar}@cs.unibo.it

Abstract

We introduce a process algebra containing the coordination primitives of Linda (asynchronous communication via a shared data space, read operation, non-blocking test operators on the shared space). We compare two possible semantics for the output operation: the former, that we call *ordered*, defines the output as an operation that returns when the message has reached the shared data space; the latter, that we call *unordered*, returns just after sending the message to the tuple space. The process algebra under the ordered semantics is Turing powerful, as we are able to program any Random Access Machine. The main result of the paper is that the process algebra under the unordered semantics is not Turing powerful. This result is achieved by resorting to a net semantics in terms of contextual nets (P/T nets with inhibitor and read arcs), and showing that there exists a deadlock-preserving simulation of such nets by finite P/T nets, a formalism where termination is decidable.

1 Introduction

Linda [CG90] is the most prominent representative of the family of coordination languages based on the principle of *generative communication* [Gel85, GC92]: a sender communicates with a receiver through a shared data space (called *tuple space*, TS for short), where emitted messages are collected; the receiver can read the message or even remove it from the TS; a message generated by a process has an independent existence in the tuple space until it is explicitly withdrawn by a receiver; in fact, after its insertion in the TS, a message becomes equally accessible to all processes, but it is bound to none.

Besides the non-blocking output operation $out(a)$ (that sends the message a to the tuple space), the blocking read operation $rd(a)$ (that succeeds only if a is in the TS) and the blocking input operation $in(a)$ (that removes message a from the TS), Linda offers two further conditional input and read predicates, called $inp(a)$ and $rdp(a)$ [CG89, CG90, Car87, SCA95]. These predicates check the current status of the TS; if the required message a is absent, the value $false$ is returned; on the contrary, if the message is found, their behavior is the same as the in/rd operation and the value $true$ is returned.

We present a process algebra **L** containing the coordination primitives of Linda[1], together with parallel composition and a limited form of recursion (the replication operator [Mil91],

*Full version of [BGZ97d].

[1] We do not consider the *eval* operator of Linda because in our process algebraic approach it would simply correspond to a spawn operation.

guarded on inputs). The primitives $in(a)$, $out(a)$ and $rd(a)$ are represented by prefix operators, whereas the predicates are modeled by means of the *if-then-else* constructs, namely $inp(a)?P\_Q$ and $rdp(a)?P\_Q$. They are conditional instructions that direct the flow of control to P or to Q depending on the presence or absence of the message a in the TS; if the message a is present, $inp(a)?P\_Q$ removes it before becoming P, while $rdp(a)?P\_Q$ does not. The test on the tuple and the removal of the message are performed in one atomic step.

The problem of defining a formal semantics for Linda has recently attracted some researchers (see, e.g., [BGZ97c] and the references therein for an overview). One interesting aspect is related to the semantics of the output operation $out(a)$, for which there exist at least two natural interpretations, depending on the fact that the execution of the output primitive $out(a)$ can be seen as composed of two phases: the *emission* of the message a (sending a to the TS) and the *rendering* of a (actual presence of the message a in the TS, that we denote by $\langle a \rangle$). The following two different semantics can be suggested:

- **Ordered**: The emission and the rendering of one message form together one single autonomous atomic action: $out(a).P$ becomes in one (internal) step the agent $\langle a \rangle | P$, where $|$ is the parallel composition operator. In this way, the order of emission is respected by the rendering order. Under this semantics, the processes are synchronous with respect to the TS when they emit messages, hence a hand–shake protocol could be used in order to implement $out(a)$: the sender sends the message a and then waits for an acknowledgement from the TS.

- **Unordered**: The emission and the rendering of one message are distinct autonomous actions. Hence, $out(a).P$ emits the message a, becoming the agent $\langle\langle a \rangle\rangle | P$ in one (internal) step, where P is free to proceed, but a is not yet present in the TS; indeed, $\langle\langle a \rangle\rangle$ takes one further internal step to become $\langle a \rangle$. In this case the processes behave asynchronously with respect to the TS: the implementation of $out(a)$ could consist of a send operation of message a to the TS, without waiting for any acknowledgement. In this way the order of emission may not be respected by the rendering order: for instance, if a process executes the sequence $out(a).out(b)$, then a may be rendered before or after the emission of b, or even after the rendering of b.

As an illustrative example, consider $(out(b).out(a) \mid in(a).inp(b)?P\_Q)$. If we assume the ordered semantics, then the input of a is possible only if $\langle b \rangle$ is already in the TS; hence, the execution of inp will always enable P. Differently, if we assume the unordered semantics, then no guarantee is given that $\langle b \rangle$ is in the TS; hence, it is sometimes possible that Q is executed instead.

While the SCA Linda User's Guide [SCA95] supports the unordered view (see the comment to the program reported on page 2-25), other papers describing implementations [Car87, DWR95] or formal semantics of Linda [Jag90, CJY94] adopt the ordered one. Thus, it seems worthwhile to investigate the impact of the two different interpretations for the *out* operator on the expressive power of our process algebra. To this aim, we provide $\mathbf{L}$ with both the ordered and unordered semantics, and we compare the expressive power of the two variations, called $\mathbf{L}_o$ and $\mathbf{L}_u$ respectively, using Turing equivalence as a yardstick. Quite amazingly, $\mathbf{L}_o$ and $\mathbf{L}_u$ possess a radically different expressive power; indeed, under the ordered semantics we get a Turing powerful language, while this is not the case under the unordered semantics.

Turing completeness of $\mathbf{L}_o$ is proved by showing that it is possible to encode any Random Access Machine (RAM) [SS63]. On the contrary, the proof that $\mathbf{L}_u$ is not Turing complete is rather elaborate and constitutes the main achievement of this paper. The proof consists in showing that the problem of termination is decidable in $\mathbf{L}_u$. We proceed as follows:

- *Net semantics*: we define a net semantics for $\mathbf{L}_u$ in terms of contextual P/T nets, i.e., P/T nets with inhibitor as well as read arcs (see, e.g., [MR95, BP95, Bus98]). This semantics, defined following the style of [BG95, BGZ97a], preserves the interleaving behaviour, hence also the possibility of deadlock.

- *Deadlock-preserving simulation*: given the contextual P/T net semantics, we present a mapping on finite (standard) P/T nets that preserves deadlock. As deadlock is decidable on finite P/T nets, we conclude that the termination problem is decidable under the unordered semantics; hence $\mathbf{L}_u$ is not Turing powerful.

The paper is organized as follows. In Section 2 we introduce the syntax of language $\mathbf{L}$ and we report on the two operational semantics for the *out* primitive. Section 3 presents the RAM implementation in $\mathbf{L}_o$. Section 4 presents the proof of the non Turing equivalence of $\mathbf{L}_u$: basic definitions on P/T nets and contextual nets are recalled, the distributed semantics for $\mathbf{L}_u$ is presented, and finally a mapping on finite P/T nets that preserves deadlock is shown.

2 The Language and its Operational Semantics

Let *Messages* be a denumerable set of message names, ranged over by $a, b, \ldots$. The syntax of the language $\mathbf{L}$ is defined by the following grammar:

$$P \;::=\; \langle a \rangle \;\mid\; C \;\mid\; P|P$$
$$C \;::=\; 0 \;\mid\; \eta.C \;\mid\; \mu?C\_C \;\mid\; C|C$$

where:

$$\eta \;::=\; in(a) \;\mid\; rd(a) \;\mid\; out(a) \;\mid\; !in(a)$$
$$\mu \;::=\; inp(a) \;\mid\; rdp(a)$$

Agents, ranged over by P, Q, ..., consist of the parallel composition of the messages already in the TS (each one denoted by one agent $\langle a \rangle$) and the concurrent programs denoted by C, D, ..., that share the tuples. A program can be a terminated program 0 (which is usually omitted for the sake of simplicity), a prefix form $\eta.P$, an *if-then-else* form $\mu?P\_Q$, or the parallel composition of programs. In the following, prefix and if-then-else forms are also called *sequential programs* (or *sequential subprograms* when they appear as subterms). A prefix η can be one of the Linda primitives $in(a)$, $rd(a)$, or $out(a)$ indicating the withdrawing, the reading (without consumption) or the emission of message a, respectively. We also consider the bang operator $!in(a)$ which is a form of replication guarded on input operations: the term $!in(a).P$ is always ready to consume a message a in the TS and then activate a copy of the program P. The if-then-else forms are used to model the $inp(a)$ and $rdp(a)$ primitives: $inp(a)?P\_Q$ ($rdp(a)?P\_Q$) is a program which requires the message a to be consumed (or simply read); if a is present, the program P is executed, otherwise Q is chosen. In the following, *Agent* denotes the set containing all possible agents.

We present two different operational semantics (in terms of labeled transition systems) for our language, one according to the ordered interpretation of the *out* operator and the other following the unordered interpretation, as presented in the previous section. The two variations are called $\mathbf{L}_o$ and $\mathbf{L}_u$, respectively.

The labeled transition systems are of the kind $(Agent, Label, \longrightarrow)$ where $Label = \{\tau\} \cup \{a, \underline{a}, \overline{a}, \neg a \mid a \in Messages\}$ (ranged over by α, β, ...) is the set of the possible labels. The

(1)	$\langle a \rangle \xrightarrow{\overline{a}} 0$		(2)	$in(a).P \xrightarrow{a} P$	
(3)	$rd(a).P \xrightarrow{\underline{a}} P$		(4)	$!in(a).P \xrightarrow{a} P \vert !in(a).P$	
(5)	$inp(a)?P\_Q \xrightarrow{a} P$		(6)	$rdp(a)?P\_Q \xrightarrow{\underline{a}} P$	
(7)	$inp(a)?P\_Q \xrightarrow{\neg a} Q$		(8)	$rdp(a)?P\_Q \xrightarrow{\neg a} Q$	
(9)	$\dfrac{P \xrightarrow{\alpha} P'}{P \vert Q \xrightarrow{\alpha} P' \vert Q}$	$\alpha \neq \neg a$	(10)	$\dfrac{P \xrightarrow{\neg a} P' \quad Q \not\xrightarrow{\overline{a}}}{P \vert Q \xrightarrow{\neg a} P' \vert Q}$	
(11)	$\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \vert Q \xrightarrow{\tau} P' \vert Q'}$		(12)	$\dfrac{P \xrightarrow{\underline{a}} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \vert Q \xrightarrow{\tau} P' \vert Q}$	

Table 1: Operational semantics (symmetric rules omitted).

labeled transition relation $\longrightarrow_o$ for the *ordered* semantics is the smallest one satisfying the axioms and rules (1) – (12) in Table 1 and the axiom (13) of Table 2; while $\longrightarrow_u$ for the unordered semantics considers the axioms and rules (1) – (12), (14) and (15). For the sake of simplicity we omit the symmetric rules of (9) – (12). The indexes o and u (distinguishing the two variations of the language and the two transition relations) are omitted when no confusion arises.

Axiom (1) shows that the tuple $\langle a \rangle$ is able to give its contents to the environment by performing an action labeled with $\overline{a}$. Axioms (2) and (3) describe the action performed by the prefixes $in(a)$ and $rd(a)$, whose labels are a and $\underline{a}$, respectively. The term $!in(a).P$ is able to activate a new copy of the program P by performing a derivation labeled with a that requires a tuple $\langle a \rangle$ to be consumed (axiom (4)). Axioms (5) – (8) describe the semantics of the two if-then-else forms $inp(a)?P\_Q$ and $rdp(a)?P\_Q$: if the required message a is present, it can be consumed by the former (by means of a derivation labeled with a) or only read by the latter (by means of a derivation labeled with $\underline{a}$). In both cases, if the required tuple $\langle a \rangle$ is not found, its absence is guessed by performing an action labeled with $\neg a$ (axioms (7) and (8)). The usual rule (9) for the parallel operator can be applied only to the labels different from $\neg a$. In fact, when an agent P willing to perform a derivation labeled with $\neg a$ is composed in parallel with another agent Q, the executability of $\neg a$ by $P|Q$ depends on the inability of Q to offer the tuple $\langle a \rangle$. Otherwise, the guess of P is wrong and the derivation labeled with $\neg a$ cannot be executed (rule (10)). Rule (11) is the usual one for synchronization between the complementary actions a and $\overline{a}$, while the rule (12) represents a new form of synchronization between the labels $\underline{a}$ and $\overline{a}$ in which the agent performing the derivation labeled with $\overline{a}$ is left unchanged. In fact, a reading operation does not change the state of the TS (i.e., it does not remove the read tuple).

The rules which differentiate the *ordered* from the *unordered* semantics are presented in Table 2. In the *ordered* approach the output operation consists of one internal move which creates the tuple $\langle a \rangle$. In this way, when a sequence of outputs is executed, the tuples are rendered in the same order they are emitted.

In the *unordered* approach, the execution of an output operation emitting the tuple with

Ordered:	(13) $out(a).P \xrightarrow{\tau} \langle a\rangle \vert P$	**Unordered:**	(14) $out(a).P \xrightarrow{\tau} \langle\langle a\rangle\rangle \vert P$ (15) $\langle\langle a\rangle\rangle \xrightarrow{\tau} \langle a\rangle$

Table 2: Two semantics for the *out* primitive.

contents a does not directly generate the corresponding term $\langle a\rangle$, but it creates an agent which will make the tuple $\langle a\rangle$ available only after a non predictable delay. This agent, denoted by $\langle\langle a\rangle\rangle$ (term added to the syntax of the language in the case of unordered semantics) represents the fact that the message a has been sent to the TS but it has not been received yet. This behaviour is obtained by allowing $\langle\langle a\rangle\rangle$ to perform an internal step, becoming $\langle a\rangle$.

Note that rule (10) uses a negative premise; however, our operational semantics is well defined, because (as described by the following proposition) our transition system specification is strictly stratifiable [Gro93], condition that ensures (as proved in [Gro93]) the existence of a unique transition system agreeing with it.

Proposition 2.1 The transition system specification of Table 1 is strictly stratifiable.
Proof Consider function S defined in the following way:

$$S(P \xrightarrow{\alpha} P') \;=\; n \quad \text{if } n \text{ is the number of parallel composition operators appearing in } P$$

Function S is a strict stratification of the transition system specification of Table 1: if $P \xrightarrow{\alpha} P'$ is the conclusion of a rule in Table 1, then the number of parallel composition operators in P is strictly greater than the occurrences of the parallel operators in Q for any $Q \xrightarrow{\alpha} Q'$ or $Q \stackrel{\beta}{\not\longrightarrow}$ appearing in the premises of the rule. ■

In the following we only consider computations consisting of reduction steps, i.e. the internal derivations that a *stand-alone* agent is able to perform independently of the context. In our language, we consider as reductions not only the usual derivations labeled with τ, but also those labeled with $\neg a$. In fact, a derivation $P \xrightarrow{\neg a} P'$ indicates that P can become P' if no tuples $\langle a\rangle$ are available in the external environment. Hence, if P is stand-alone (i.e. without external environment), it can be considered able to become P'. Indeed, the label $\neg a$ has been only introduced for helping an SOS formulation of the semantics, but it is conceptually an internal step. Formally:

$$P \longrightarrow P' \quad \text{iff } P \xrightarrow{\tau} P' \text{ or } P \xrightarrow{\neg a} P' \text{ for some } a$$

We use $P \longrightarrow^i P'$ to state that P' can be reached from P by means of i reduction steps, while $P \longrightarrow^* P'$ is the reflexive and transitive closure of $\longrightarrow$.

An agent P is *dead* if $P \not\longrightarrow$. As in process algebra there is no explicit notion of state, the only reasonable form of termination of an agent is the absence of outgoing transitions. Because of the presence of the bang operator, it is senseless to distinguish between properly terminated (i.e. consisting of the parallel composition of agents 0) and deadlocked agents. Indeed, each nontrivial (i.e., containing a bang) agent with no outgoing transitions cannot be properly terminated.

In order to deal with the encoding of recursive functions in a nondeterministic setting, as a process algebra, we have to specify what we mean for an agent to terminate. Following the approach of other formalisms, such as λ-calculus, (nondeterministic) Turing machines, phrase structure grammars, we say that an agent terminates if there exists a computation leading to a dead agent. Formally, the agent P terminates if it *has a deadlock*, i.e., there exists a dead agent Q such that $P \longrightarrow {}^*Q$.

3 $\mathbf{L}_o$ is Turing Powerful

We show that under the *ordered* semantics the language is Turing powerful because it is expressive enough to model any Random Access Machine (RAM) [SS63].

A RAM is a computational model composed of a finite set of registers, that can hold arbitrary large natural numbers, and by a program, that is a sequence of simple numbered instructions, like arithmetical operations (on the contents of registers) or conditional jumps.

To perform a computation, the inputs are provided in registers $r_1, \ldots, r_m$; if other registers $r_{m+1}, \ldots, r_n$ are used in the program, they are supposed to contain the value 0 at the beginning of the computation. The execution of the program begins with the first instruction and continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached; this happens if the program was executing the last instruction of the program and this instruction does not require a jump, or if the current instruction requires a jump to an instruction number not appearing in the program. If the program terminates, the result of the computation is the contents of the registers specified as outputs.

In [Min67] it is shown that the following two instructions are sufficient to model every recursive function:

- $Succ(r_j)$: adds 1 to the contents of register r_j;
- $DecJump(r_j, s)$: if the contents of register r_j is not zero, then decreases it by 1 and go to the next instruction, otherwise jumps to instruction s.

For example, the following program computes the sum of the registers r_1 and r_2, putting the result in the register r_1 (note that the third instruction corresponds to an unconditional jump, because the register r_3 contains the value 0 at the beginning of the computation and its contents is never modified by the program):

$1 \ : \ DecJump(r_2, 4)$
$2 \ : \ Succ(r_1)$
$3 \ : \ DecJump(r_3, 1)$

The state of the computation is represented by means of configurations $(i, c_1, c_2, \ldots, c_n)$ where i indicates that the next instruction to execute is the i^{th} and c_l is the contents of the register r_l for l in $1 \ldots n$. Given a program R and a configuration $(i, c_1, c_2, \ldots, c_n)$ we use the notation:

$$(i, c_1, c_2, \ldots, c_n) \longrightarrow_R (j, c'_1, c'_2, \ldots, c'_n)$$

to state that after the execution of the i^{th} instruction of the program R with contents of the registers $c_1, \ldots, c_n$, the program counter points to the j^{th} instruction, and the registers will contain $c'_1, \ldots, c'_n$.

To model a RAM in our language we present one encoding for the configurations and one for the programs. The program counter in the configuration $(i, c_1, c_2, \ldots, c_n)$ is represented by the *program counter tuple* $\langle p_i \rangle$ while the contents of register r_l is modeled by c_l occurrences of the tuple $\langle r_l \rangle$ (for $l = 1 \ldots n$):

$$[\![(i, c_1, c_2, \ldots, c_n)]\!] \stackrel{def}{=} \langle p_i\rangle | \underbrace{\langle r_1\rangle|\ldots|\langle r_1\rangle}_{c_1 \text{ times}} | \ldots | \underbrace{\langle r_n\rangle|\ldots|\langle r_n\rangle}_{c_n \text{ times}}$$

To model the program R composed of the sequence of instructions $I_1 \ldots I_k$, we consider the parallel composition of k agents, each one modeling an instruction. A $Succ$ instruction on register r_j at position i is represented by an agent that consumes the program counter tuple, adds a tuple r_j and updates the program counter by adding a tuple p_{i+1}. An instruction $DecJump(r_j, s)$ at position i is modeled by an agent that, after consuming the program counter tuple, performs an inp on message r_j; if the operation succeeds, then a tuple $\langle r_j\rangle$ has been withdrawn from the tuple space and the agent updates the program counter by adding $\langle p_{i+1}\rangle$, otherwise a jump to the s^{th} instruction is performed by adding $\langle p_s\rangle$.

$$\begin{array}{lcl} [\![R]\!] & \stackrel{def}{=} & [\![I_1]\!]|\ldots|[\![I_k]\!] \\ [\![i : Succ(r_j)]\!] & \stackrel{def}{=} & !in(p_i).out(r_j).out(p_{i+1}) \\ [\![i : DecJump(r_j, s)]\!] & \stackrel{def}{=} & !in(p_i).inp(r_j)?out(p_{i+1})\_out(p_s) \end{array}$$

The use of the bang operator in the representation of the instructions permits to reuse them. It is interesting to note that each RAM instruction is realized in our implementation by means of three internal steps: the first consumes the program counter, the second updates/tests the contents of the registers, and the third introduces the new program counter tuple.

The agent modeling the program R that starts its computation from the configuration $(1, c_1, c_2, \ldots, c_n)$ is $[\![R]\!]|[\![(1, c_1, c_2, \ldots, c_n)]\!]$.

Theorem 3.1 Let R be a RAM program, then:

- *Soundness:* if $[\![R]\!]|[\![(i, c_1, c_2, \ldots, c_n)]\!] \longrightarrow^3 Q$ then there exists a unique configuration $(j, c'_1, c'_2, \ldots, c'_n)$ such that:
  $Q = [\![R]\!]|[\![(j, c'_1, c'_2, \ldots, c'_n)]\!]$ and $(i, c_1, c_2, \ldots, c_n) \longrightarrow_R (j, c'_1, c'_2, \ldots, c'_n)$
- *Completeness:* if $(i, c_1, c_2, \ldots, c_n) \longrightarrow_R (j, c'_1, c'_2, \ldots, c'_n)$ then also:
  $[\![R]\!]|[\![(i, c_1, c_2, \ldots, c_n)]\!] \longrightarrow^3 [\![R]\!]|[\![(j, c'_1, c'_2, \ldots, c'_n)]\!]$

Proof By cases on the possible instructions that can be activated. In the proof of soundness we use the fact that the program counter tuple $\langle p_i\rangle$ in the term $[\![R]\!]|[\![(i, c_1, c_2, \ldots, c_n)]\!]$ ensures that only the agent corresponding to the i^{th} instruction can move. ■

4 $\mathbf{L}_u$ is not Turing Powerful

The implementation of the RAM we have presented does not satisfy the soundness property of Theorem 3.1 under the unordered semantics. Consider an execution of the program:

$$\begin{array}{lcl} 1 & : & Succ(r_1) \\ 2 & : & DecJump(r_1, 1) \end{array}$$

starting with the register r_1 empty. The implementation of the $Succ(r_1)$ instruction creates two tuples: $\langle r_1\rangle$ and the new program counter tuple $\langle p_2\rangle$. If the tuple $\langle p_2\rangle$ becomes available before $\langle r_1\rangle$, the following $DecJump(r_1, 1)$ instruction could execute the jump because no tuple $\langle r_1\rangle$ is available. Then, the computation gives rise to a loop that is exited only when the first tuple $\langle r_1\rangle$ becomes available in the TS. Instead, the expected behaviour of the RAM is that the program terminates with the register r_1 empty, after it has been incremented and then decremented.

We assert not only that the implementation we have presented for $\mathbf{L}_o$ is not sound under the *unordered* semantics, but we will also prove that no implementation of the RAM exists. In fact, in the remainder of this section we show that $\mathbf{L}_u$ is not Turing powerful. In order to prove this, we give a contextual P/T net semantics to $\mathbf{L}_u$ for which there exists a deadlock preserving mapping on finite (standard) P/T nets. As deadlock is decidable on these nets, we can conclude that the termination is decidable for $\mathbf{L}_u$. In the following sections we recall some definitions about contextual P/T nets, we present the net semantics for $\mathbf{L}_u$, and finally we provide the obtained class of nets with a deadlock preserving simulation on finite P/T nets.

4.1 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, MR95]). Here we provide a characterization of this model which is convenient for our aims.

Definition 4.1 Given a set S, a *finite multiset* over S is a function $m : S \to \mathbb{N}$ 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 $\wp_{fin}(S)$.

Given the multiset m and m', we write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$ while $\oplus$ denotes their *multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes *multiset difference*: $(m \setminus m')(s) =$ if $m(s) \geq m'(s)$ then $m(s) - m'(s)$ else 0. The *scalar product*, $j \cdot m$, of a number j with m is $(j \cdot m)(s) = j \cdot (m(s))$. ■

To lighten the notation, we sometimes use the following abbreviation. If m is a multiset containing only one occurrence of an element s (i.e., $dom(m) = \{s\}$ and $m(s) = 1$) we denote m by only s. Multiset union is represented also by comma, i.e., $m, m' = m \oplus m'$. Let m be a multiset over S and m' a multiset over $S' \supseteq S$, such that $(m'(s') = 0)$ for each $s' \in S' \setminus S$; with abuse of notation, we sometimes use m in place of m', and vice versa.

Definition 4.2 A P/T system is a triple $N = (S, T, m_0)$ where S is the set of *places*, $T \subseteq \mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(S)$ is the set of *transitions*, and m_0 is a finite multiset over the set S of places. Finite multisets over the set S of places are called *markings*; m_0 is called *initial marking*. Given a marking m and a place s, we say that the place s contains $m(s)$ *tokens*. A P/T system is finite if both S and T are finite.

A transition $t = (c, p)$ is usually written in the form $c \longrightarrow p$. The marking c, usually denoted by ${}^\bullet t$, is called the *preset* of t and represents the tokens to be *consumed*; the marking p, usually denoted by $t^\bullet$, is called the *postset* of t and represents the tokens to be *produced*. 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 $m \xrightarrow{t} m'$ or simply $m \longrightarrow m'$ when the transition t is not relevant. We use σ, τ to range over sequences of transitions; the empty sequence is denoted by ε; let $\sigma = t_1, \ldots, t_n$, we write $m \xrightarrow{\sigma} m'$ to mean the *firing sequence* $m \xrightarrow{t_1} \cdots \xrightarrow{t_n} m'$. We say that m' is *reachable from* m if there exists σ such that $m \xrightarrow{\sigma} m'$.

Given a place s and a transition t, if $s \in dom({}^\bullet t) \cap dom(t^\bullet)$ then we say that there is a *self-loop* between t and s.

A marking m is *dead* if $m \not\longrightarrow$. The net $N = (S, T, m_0)$ has a *deadlock* if there exists a dead marking reachable from m_0. The deadlock problem for a net consists in deciding if it has a deadlock.

A transition t is *live* if for each marking m reachable from m_0 there exists a marking m' reachable from m such that t is enabled at m'. A net is *live* if each of its transitions is live. The liveness problem for a net consists in deciding if it is live. ■

The deadlock problem for a finite P/T system is decidable; this can be proved by reducing it to the liveness problem, which is known to be decidable (see, e.g., [Reu88]). The theorem below is a slight generalization of the result proved in [CEP95] for nets where the preset and postset of transitions are sets.

Theorem 4.3 Deadlock is reducible to liveness.
Proof Let $N = (S, T, m_0)$ be a P/T system. We construct a net $N' = (S', T', m'_0)$, where $S' = S \cup \{ok\}$. For each $t \in T$, let $t' = {}^\bullet t \longrightarrow t^\bullet$ and $t'' = {}^\bullet t \longrightarrow ok$; let $live = ok \longrightarrow \bigoplus_{t \in T} {}^\bullet t \oplus ok$. $T' = \{t', t'' \mid t \in T\} \cup \{live\}$ and $m'_0 = m_0$.

We show that N has no reachable dead marking iff N' is live.

Suppose that N has a reachable dead marking m; also N' can reach the marking m by firing the corresponding t' transitions. We have $m(ok) = 0$, then the transition $live$ is not enabled at m; as transitions t' and t'' have the same preset of the corresponding transition t, also these transitions are not enabled at m in N'; then m is a dead marking for N', hence N' is not live.

Suppose N has no reachable dead markings. Let m be a reachable marking in N'; two cases can happen:

- $m(ok) > 0$; the transition $live$ is enabled at m and, after the firing of $live$, each transition in T' is enabled;

- $m(ok) = 0$; note that if we produce a token in the place ok it will remain always marked; then only transitions t' have been fired to reach m, hence m is reachable also in N; as N has no reachable dead markings, there exists $t \in T$ such that $m \xrightarrow{t}$ in N; then $m \xrightarrow{t''} m'$ in N', and $m'(ok) = 1$, leading to the previous case. ■

Definition 4.4 A *contextual* P/T system is a triple $N = (S, T, m_0)$ where S is the set of places, $T \subseteq \mathcal{M}_{fin}(S) \times \wp_{fin}(S) \times \wp_{fin}(S) \times \mathcal{M}_{fin}(S)$ is the set of transitions, and m_0 is the *initial marking.*

A transition $t = (c, r, i, p)$ is usually written in the form $r, \not{i} : c \longrightarrow p$ where r and $\not{i}$ are omitted when empty. The set r, denoted by $\hat{t}$, is called the *contextual set* of t and represents the tokens to be *tested for presence*; the set i, denoted by $^\circ t$, 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 $dom(m) \cap {}^\circ t = \emptyset$.

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

4.2 Net Semantics

The basic idea underlying the definition of an operational net semantics for a process algebra is to decompose a process P into a multiset of sequential components, which can be thought

$dec(0)$	$=$	$\emptyset$	$dec(\langle a\rangle)$	$=$	$\{\langle a\rangle\}$
$dec(\langle\langle a\rangle\rangle)$	$=$	$\{\langle\langle a\rangle\rangle\}$	$dec(\eta.P)$	$=$	$\{\eta.P\}$
$dec(\mu?P\_Q)$	$=$	$\{\mu?P\_Q\}$	$dec(P\|Q)$	$=$	$dec(P), dec(Q)$

Table 3: Decomposition function.

`in(a,Q)`		$in(a).Q, \langle a\rangle \longrightarrow dec(Q)$
`rd(a,Q)`	$\langle a\rangle :$	$rd(a).Q \longrightarrow dec(Q)$
`out(a,Q)`		$out(a).Q \longrightarrow \langle\langle a\rangle\rangle, dec(Q)$
`rend(a)`		$\langle\langle a\rangle\rangle \longrightarrow \langle a\rangle$
`inp+(a,Q,R)`		$inp(a)?Q\_R, \langle a\rangle \longrightarrow dec(Q)$
`rdp+(a,Q,R)`	$\langle a\rangle :$	$rdp(a)?Q\_R \longrightarrow dec(Q)$
`inp-(a,Q,R)`	$\langle \not{a}\rangle :$	$inp(a)?Q\_R \longrightarrow dec(R)$
`rdp-(a,Q,R)`	$\langle \not{a}\rangle :$	$rdp(a)?Q\_R \longrightarrow dec(R)$
`!in(a,Q)`	$!in(a).Q :$	$\langle a\rangle \longrightarrow dec(Q)$

Table 4: Net semantics.

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. Reductions are represented by transitions which consume and produce multisets of sequential components if the contextual conditions are satisfied.

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

Given the agent P, we define the corresponding contextual P/T system $Net(P)$. In order to do this, we need the following notations.

- Let $\mathcal{S}$ be the set $\{P \mid P \text{ is a sequential program}\} \cup \{\langle a\rangle, \langle\langle a\rangle\rangle \mid a \text{ is a message name}\}$.
- Let the function $dec : Agent \to \mathcal{M}_{fin}(\mathcal{S})$ be the decomposition of agents into markings, reported in Table 3.
- Let $\mathcal{T}$ contain the transitions obtained as instances of the axiom schemata presented in Table 4.

The axioms in Table 3, describing the decomposition of agents, state that the agent 0 generates no tokens; the decomposition of the terms $\langle a\rangle$, $\langle\langle a\rangle\rangle$, and of the sequential programs produces one token in the corresponding place; while the parallel composition is interpreted as multiset union, i.e, the decomposition of $P|Q$ is $dec(P), dec(Q)$.

The axioms in Table 4 deal with the possible transitions. Axioms `in(a,Q)` and `rd(a,Q)` deal with the execution of the primitives $in(a)$ and $rd(a)$: in the first case a token from place $\langle a\rangle$ is consumed, while in the second case it is only tested for presence. Axiom `out(a,Q)` describes how a new token in place $\langle\langle a\rangle\rangle$ can be generated by the output prefix $out(a)$; this new token can be consumed (according to axiom `rend(a)`) to generate a new one in place $\langle a\rangle$. The axioms `inp+(a,Q,R)` and `rdp+(a,Q,R)` are similar to the corresponding axioms `in(a,Q)` and `rd(a,Q)`. In the axioms `inp-(a,Q,R)` and `rdp-(a,Q,R)`, tuple $\langle a\rangle$ is tested for absence instead: if it is not present, the agent R is selected. Finally, axiom `!in(a,Q)` deals with the bang operator: if a token is present in place $!in(a).Q$ and a token can be consumed from place $\langle a\rangle$, then a new copy of $dec(Q)$ is produced.

With the name of a transition without parameters we denote any instance of that transition, i.e. we use `in` to mean any instance of `in(a,Q)`.

Definition 4.5 Let P be an agent. We define the triple $Net(P) = (S, T, m_0)$ where:

$$\begin{array}{lcl}
S & = & \{Q \mid Q \text{ is a sequential subprogram of } P\} \cup \\
 & & \{\langle a\rangle, \langle\langle a\rangle\rangle \mid a \text{ is a message name in } P\} \\
T & = & \{r, \not{i} : c \longrightarrow p \in \mathcal{T} \mid \exists\ Q \text{ sequential subprogram of } P \text{ s.t. } Q \in dom(c) \cup r\} \cup \\
 & & \{\texttt{rend(a)} \mid a \text{ is a message name in } P\} \\
m_0 & = & dec(P)
\end{array}$$

■

The following proposition states that $Net(P)$ is well defined, in the sense it is a contextual P/T system; moreover, it is finite.

Proposition 4.6 Given the agent P, $Net(P)$ is a finite contextual P/T system.
Proof We first show that $Net(P) = (S, T, m_0)$ is a contextual P/T system, then we observe that both S and T are finite.

The domain of the initial marking is a multiset on the set of places, i.e., $dom(dec(P)) \subseteq S$, and it is finite.

Moreover, each transition in T is legal for $Net(P)$; indeed, given $r, \not{i} : c \longrightarrow p \in T$, it is easy to see that $r \cup i \cup dom(c) \cup dom(p) \subseteq S$.

The finiteness of S and T is ensured by the following facts:

- the sets of message names in P and of sequential subprograms of P are finite due to the finite syntax of P;
- each place in S corresponds to a sequential subterm Q of P or to a message name a in P;
- each transition `rend(a)` in T corresponds to a message name a in P. On the other hand, each transition $t \in T$, that is not a `rend()`, corresponds to a sequential subprogram Q of P s.t. $Q \in dom({}^\bullet t) \cup \hat{t}$. But, given a sequential subprogram Q, the set of transitions t such that $Q \in dom({}^\bullet t) \cup \hat{t}$ is finite. ■

4.2.1 Soundness and Completeness of the Net semantics

The proof of the appropriateness of the contextual P/T net semantics w.r.t. the SOS consists of showing that each sequence of SOS reductions of an agent is matched by a sequence of transitions of the corresponding net, and vice versa. The kernel of the proof is formed by two theorems: first, we show that each reduction derivable from the SOS specification is matched by a transition in the net; then, we show that every transition in the net corresponds to a reduction step in the transition system. In order to prove this, we need some auxiliary results, relating labelled SOS transitions with tokens in corresponding places of the net.

In the following two propositions we show that the ability of an agent to perform an output transition labelled with $\overline{a}$ corresponds to the presence of a token in place $\langle a \rangle$ in the marking corresponding to the agent.

Proposition 4.7 Let P be an agent. If $P \xrightarrow{\overline{a}} P'$ then there exists a marking m such that $dec(P) = \langle a \rangle \oplus m$ and $dec(P') = m$.

Proof We proceed by induction on the proof of the transition $P \xrightarrow{\overline{a}} P'$. The base case is trivial: $P = \langle a \rangle$ and $m = \emptyset$. In the induction case, $P = P_1|P_2$ and (it is not restrictive to suppose) $P' = P_1'|P_2$ with $P_1 \xrightarrow{\overline{a}} P_1'$. By induction hypothesis, there exists the marking m' such that $dec(P_1) = \langle a \rangle \oplus m'$ and $dec(P_1') = m'$. It is sufficient to take $m = m' \oplus dec(P_2)$, as $dec(P_1|P_2) = \langle a \rangle \oplus m' \oplus dec(P_2)$ and $dec(P_1'|P_2) = m' \oplus dec(P_2)$. ■

Proposition 4.8 Let P be an agent. If $dec(P) = \langle a \rangle \oplus m$ then there exists an agent P' such that $P \xrightarrow{\overline{a}} P'$ and $dec(P') = m$.

Proof By structural induction on P. ■

If an agent can perform an input (resp. read) transition labelled with a (resp. $\underline{a}$), then there is a token in a place corresponding to a sequential subprocess starting with an input (resp. read) of message a.

Proposition 4.9 Let P be an agent. If $P \xrightarrow{a} P'$ then one of the following holds:

1. $dec(P) = \{in(a).Q\} \oplus m$ and $dec(P') = dec(Q) \oplus m$
2. $dec(P) = \{!in(a).Q\} \oplus m$ and $dec(P') = dec(P) \oplus dec(Q)$
3. $dec(P) = \{inp(a)?Q_1\_Q_2\} \oplus m$ and $dec(P') = dec(Q_1) \oplus m$

Proof By induction on the proof of the transition $P \xrightarrow{a} P'$. ■

Proposition 4.10 If $P \xrightarrow{\underline{a}} P'$ then one of the following holds:

1. $dec(P) = \{rd(a).Q\} \oplus m$ and $dec(P') = dec(Q) \oplus m$
2. $dec(P) = \{rdp(a)?Q_1\_Q_2\} \oplus m$ and $dec(P') = dec(Q_1) \oplus m$

Proof By induction on the proof of the transition $P \xrightarrow{\underline{a}} P'$. ■

We need to show that each reduction, performable by an agent reachable from P after a sequence of reductions, is matched by the firing of a corresponding transition in $Net(P)$. Actually, we prove a more general result: instead of considering only agents reachable from P, we examine all the agents whose decomposition turns out to be a legal marking of $Net(P)$.

Theorem 4.11 Let $Net(P) = (S, T, m_0)$. Let R be an agent s.t. $dom(dec(R)) \subseteq S$ then:

1. if $R \xrightarrow{\tau} R'$ then there exists $t \in T$ such that $dec(R) \xrightarrow{t} dec(R')$ and $^\circ t = \emptyset$;

2. if $R \xrightarrow{\neg a} R'$ then there exists $t \in T$ such that $dec(R) \xrightarrow{t} dec(R')$ and $^\circ t = \{\langle a \rangle\}$.

Proof In order to prove the first item we proceed by induction on the proof of transitions $R \xrightarrow{\tau} R'$.

The base case consists of two possible axioms of Table 2:
$\langle\langle a \rangle\rangle \xrightarrow{\tau} \langle a \rangle$ and $out(a).Q \xrightarrow{\tau} \langle\langle a \rangle\rangle | Q$.

In the first case $dec(\langle\langle a \rangle\rangle) = \{\langle\langle a \rangle\rangle\}$. The thesis follows from $\{\langle\langle a \rangle\rangle\} \stackrel{\texttt{rend(a)}}{\longrightarrow} \{\langle a \rangle\}$ where $\{\langle a \rangle\} = dec(\langle a \rangle)$. In the second case $dec(out(a).Q) = \{out(a).Q\}$. The thesis in ensured by the derivation $\{out(a).Q\} \stackrel{\texttt{out(a,Q)}}{\longrightarrow} \{\langle\langle a \rangle\rangle\} \oplus dec(Q)$ where $\{\langle\langle a \rangle\rangle\} \oplus dec(Q) = dec(\langle\langle a \rangle\rangle | Q)$.

In the induction case three possible rules can be used to derive the transition (the treatment of symmetric rules is similar):

1. Rule (9). In this case $R = R_1 | R_2$ and $R' = R'_1 | R_2$ where $R_1 \xrightarrow{\tau} R'_1$. The inductive hypothesis can be applied as $dom(dec(R_1)) \subseteq dom(dec(R)) \subseteq S$. Then, $dec(R_1) \xrightarrow{t} dec(R'_1)$ with $^\circ t = \emptyset$. Hence also $dec(R_1 | R_2) = dec(R_1) \oplus dec(R_2) \xrightarrow{t} dec(R'_1) \oplus dec(R_2) = dec(R')$ because $^\circ t = \emptyset$.

2. Rule (11). In this case $R = R_1 | R_2$ and $R' = R'_1 | R'_2$ where $R_1 \xrightarrow{a} R'_1$ and $R_2 \xrightarrow{\overline{a}} R'_2$. The inductive hypothesis can be applied because $dom(dec(R_i)) \subseteq dom(dec(R)) \subseteq S$ for $i = \{1, 2\}$. By Propositions 4.9 there are three possible cases to consider. We present only the analysis of the first one (the others can be treated similarly). By 4.9(1) $dec(R_1) = \{in(a).Q\} \oplus m_1$ and $dec(R'_1) = dec(Q) \oplus m_1$. By Proposition 4.7 $dec(R_2) = \{\langle a \rangle\} \oplus m_2$ and $dec(R'_2) = m_2$. Hence $dec(R_1 | R_2) = dec(R_1) \oplus dec(R_2) = \{in(a).Q\} \oplus m_1 \oplus \{\langle a \rangle\} \oplus m_2$; we have that $\{in(a).Q\} \oplus m_1 \oplus \{\langle a \rangle\} \oplus m_2 \stackrel{\texttt{in(a,Q)}}{\longrightarrow} dec(Q) \oplus m_1 \oplus m_2 = dec(R'_1) \oplus dec(R'_2) = dec(R'_1 | R'_2)$, and $^\circ\texttt{in(a,Q)} = \emptyset$.

3. Rule (12). Similar to the case above.

In order to prove the second item of the theorem we proceed by induction on the proof of transitions $R \xrightarrow{\neg a} R'$. The base cases consist of Axioms (7) and (8) of Table 1. We consider Axiom (7) only: we have $R = inp(a)?R_1\_R_2$ and $R' = R_2$. Thus $dec(inp(a)?R_1\_R_2) = \{inp(a)?R_1\_R_2\}$ where $\{inp(a)?R_1\_R_2\} \stackrel{\texttt{inp-(a,R}_1\texttt{,R}_2\texttt{)}}{\longrightarrow} dec(R_2)$ and $^\circ\texttt{inp-(a,R}_1\texttt{,R}_2\texttt{)} = \{\langle a \rangle\}$.

In the induction case, rule (10) of Table 1 has been used in order to derive the transition; hence, $R = R_1 | R_2$, $R' = R'_1 | R_2$ where $R_1 \xrightarrow{\neg a} R'_1$ and $R_2 \stackrel{\overline{a}}{\not\longrightarrow}$. By inductive hypothesis $dec(R_1) \xrightarrow{t} dec(R'_1)$ with $^\circ t = \{\langle a \rangle\}$. By Proposition 4.8, $R_2 \stackrel{\overline{a}}{\not\longrightarrow}$ ensures $\langle a \rangle \not\in dec(R_2)$. Hence also, $dec(R_1 | R_2) \xrightarrow{t} dec(R'_1 | R_2)$. ■

If the decomposition of an agent contains a sequential subprocess starting with an input (resp. read) of message a, then the agent can perform an input (resp. read) transition labelled with a (resp. $\underline{a}$).

Proposition 4.12 Let P be an agent.

1. if $dec(P) = \{in(a).Q\} \oplus m$ then there exists an agent P' such that $P \xrightarrow{a} P'$ and $dec(P') = dec(Q) \oplus m$
2. if $dec(P) = \{!in(a).Q\} \oplus m$ then there exists an agent P' such that $P \xrightarrow{a} P'$ and $dec(P') = dec(P) \oplus dec(Q)$
3. if $dec(P) = \{inp(a)?Q_1\_Q_2\} \oplus m$ then there exists an agent P' such that $P \xrightarrow{a} P'$ and $dec(P') = dec(Q_1) \oplus m$

Proof By structural induction on P. ■

Proposition 4.13 Let P be an agent.

1. if $dec(P) = \{rd(a).Q\} \oplus m$ then there exists an agent P' such that $P \xrightarrow{\underline{a}} P'$ and $dec(P') = dec(Q) \oplus m$
2. if $dec(P) = \{rdp(a)?Q_1\_Q_2\} \oplus m$ then there exists an agent P' such that $P \xrightarrow{\underline{a}} P'$ and $dec(P') = dec(Q_1) \oplus m$

Proof By structural induction on P. ■

We show that each transition, firable in the net corresponding to an agent P, is matched by a corresponding reduction performable by an agent reachable from P after a sequence of reductions. As for the theorem above, we prove the result in a more general setting.

Let t be a transition of $Net(P)$; note that, if ${}^\circ t \neq \emptyset$, then ${}^\circ t = \{\langle a \rangle\}$ for some message a.

Theorem 4.14 Let $Net(P) = (S, T, m_0)$. Let R be an agent such that $dom(dec(R)) \subseteq S$. If $dec(R) \xrightarrow{t} m'$ then there exists R' such that $dec(R') = m'$ and

1. if ${}^\circ t = \emptyset$ then $R \xrightarrow{\tau} R'$;
2. if ${}^\circ t = \{\langle a \rangle\}$ then $R \xrightarrow{\neg a} R'$.

Proof By case analysis on the transitions, hence by structural induction on R. We show only two significative cases.

- The transition is `rd(a,Q)`. Hence ${}^\circ$`rd(a,Q)` $= \emptyset$, $dec(R) = \{rd(a).Q, \langle a \rangle\} \oplus m$ and $m' = dec(Q) \oplus \{\langle a \rangle\} \oplus m$.

  As $dec(R)$ contains at least two elements, we have $R = R_1|R_2$.

  The following cases can happen:

  - $dec(R_1) = \{rd(a).Q\} \oplus m_1$, $dec(R_2) = \{\langle a \rangle\} \oplus m_2$, and $m = m_1 \oplus m_2$.

    By Proposition 4.13 there exists R_1' such that $R_1 \xrightarrow{\underline{a}} R_1'$ and $dec(R_1') = dec(Q) \oplus m_1$. By Proposition 4.8 there exists R_2' such that $R_2 \xrightarrow{\bar{a}} R_2'$ and $dec(R_2') = m_2$; by rule (12) of Table 1 also $R_1|R_2 \xrightarrow{\tau} R_1'|R_2$ holds. Hence we have that: $dec(R_1'|R_2) = dec(R_1') \oplus dec(R_2) = dec(Q) \oplus m_1 \oplus \{\langle a \rangle\} \oplus m_2 = dec(Q) \oplus \{\langle a \rangle\} \oplus m = m'$.
  - $dec(R_1) = \{\langle a \rangle\} \oplus m_1$, $dec(R_2) = \{rd(a).Q\} \oplus m_2$, and $m = m_1 \oplus m_2$.

    As above, using the symmetric rule of (12).

- $dec(R_1) = \{rd(a).Q, \langle a\rangle\} \oplus m_1$ and $m = m_1 \oplus dec(R_2)$.
  Transition `rd(a,Q)` is enabled at $dec(R_1)$, and $dec(R_1) \xrightarrow{\texttt{rd(a,Q)}} \{\langle a\rangle\} \oplus m_1$.
  We have $dom(dec(R_1)) \subseteq dom(dec(R)) \subseteq S$, hence by inductive hypothesis $R_1 \xrightarrow{\tau} R_1'$ and $dec(R_1') = \{\langle a\rangle\} \oplus m_1$.
  By rule (9) of Table 1 we get $R_1|R_2 \xrightarrow{\tau} R_1'|R_2$, and $dec(R_1'|R_2) = dec(R_1') \oplus dec(R_2) = dec(Q) \oplus \{\langle a\rangle\} \oplus m_1 \oplus dec(R_2) = m'$.

- The transition is `inp-(a,`Q_1`,`Q_2`)`. Hence $^\circ$`inp-(a,`Q_1`,`Q_2`)`$= \{\langle a\rangle\}$, $dec(R) = \{inp(a)?Q_1\_Q_2\} \oplus m$, $m(\langle a\rangle) = 0$ and $m' = dec(Q_2) \oplus m$.

  The following cases can happen:

  - $R = inp(a)?Q_1\_Q_2$.
    In this case $m = \emptyset$; by rule (7) $R \xrightarrow{\neg a} Q_2$.
  - $R = R_1|R_2$, $dec(R_1) = \{inp(a)?Q_1\_Q_2\} \oplus m_1$ and $m = m_1 \oplus dec(R_2)$.
    As $m_1 \subseteq m$, $m(\langle a\rangle) = 0$ implies $m_1(\langle a\rangle) = 0$, hence `inp-(a,`$Q_1$`,`$Q_2$`)` is enabled at $dec(R_1)$ and $dec(R_1) \xrightarrow{\texttt{inp-(a,}Q_1\texttt{,}Q_2\texttt{)}} dec(Q_2) \oplus m_1$.
    We have $dom(dec(R_1)) \subseteq dom(dec(R)) \subseteq S$, hence by inductive hypothesis $R_1 \xrightarrow{\neg a} R_1'$ and $dec(R_1') = dec(Q_2) \oplus m_1$.
    As $dec(R_2) \subseteq m$ and $m(\langle a\rangle) = 0$, we have $dec(R_2)(\langle a\rangle) = 0$. By Proposition 4.7 we have $R_2 \stackrel{\bar{a}}{\not\longrightarrow}$; by rule (10) of Table 1 we obtain $R_1|R_2 \xrightarrow{\neg a} R_1'|R_2$. Hence the following holds:
    $dec(R_1'|R_2) = dec(Q_2) \oplus m_1 \oplus dec(R_2) = dec(Q_2) \oplus m$.
  - $R = R_1|R_2$, $dec(R_2) = \{inp(a)?Q_1\_Q_2\} \oplus m_2$ and $m = m_2 \oplus dec(R_1)$.
    As above using the symmetric rules. ■

We are now able to state that the contextual P/T net translation is *sound* and *complete* with respect to the reduction semantics introduced in the previous section.

Corollary 4.15 Let P be an agent.

- *Soundness:* if $dec(P) \longrightarrow m_1 \ldots \longrightarrow m_n$ in $Net(P)$ then there exist $P_1 \ldots P_n$ such that $P \longrightarrow P_1 \ldots \longrightarrow P_n$ and $dec(P_i) = m_i$ for $i = 1 \ldots n$;

- *Completeness:* if $P \longrightarrow P_1 \ldots \longrightarrow P_n$ then $dec(P) \longrightarrow dec(P_1) \ldots \longrightarrow dec(P_n)$ in $Net(P)$.

Proof By induction on n. Soundness directly follows from Theorem 4.14 while completeness is a corollary of Theorem 4.11. ■

With respect to deadlock, the above theorem ensures that the net semantics preserves it, in the sense that an $\mathbf{L}_u$ process P has a deadlock iff $Net(P)$ has a deadlock.

4.3 Deadlock is decidable

We show that for each $\mathbf{L}_u$ process P there exists a finite P/T system $DNet(P)$, which is equivalent to $Net(P)$ w.r.t. deadlock, by providing a net transformation that removes contextual arcs at the price of adding new places and (normal) transitions.

We describe this transformation informally before presenting the formal definition of $DNet(P)$.

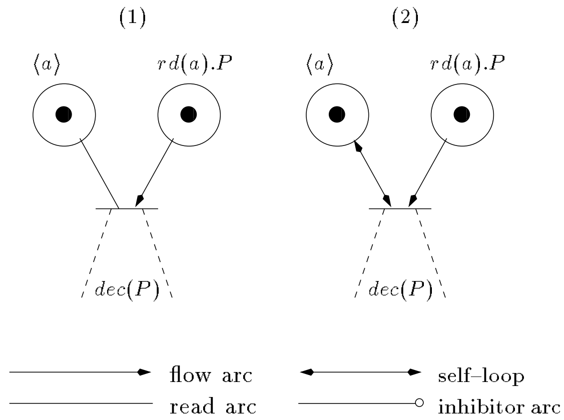

Figure 1: Eliminating read arcs in $Net(\langle a\rangle | rd(a).P)$.

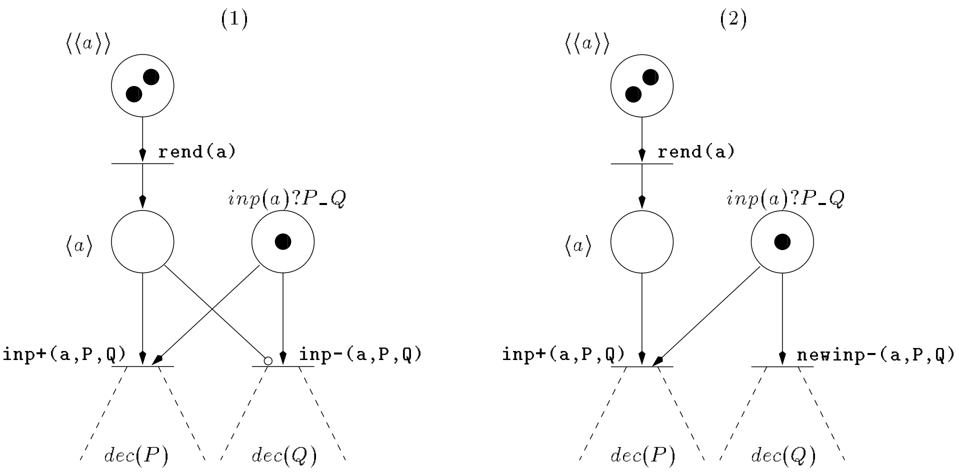

Figure 2: Eliminating inhibitor arcs in $Net(\langle\langle a\rangle\rangle | \langle\langle a\rangle\rangle | inp(a)?P\_Q)$.

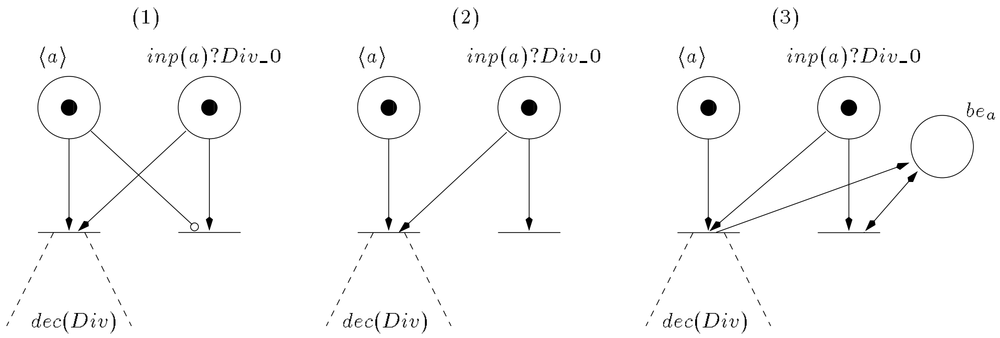

Figure 3: Transforming $Net(\langle a\rangle|inp(a)?Div\_0)$ in a deadlock-equivalent P/T system.

Read arcs

The read arcs can be replaced by self-loops: in this way, a test for presence of a token is changed to a consumption followed by an emission of one token in the same place. This transformation preserves the firing sequences, hence also the deadlock behaviour. An instance of this transformation is presented in Figure 1. In that figure, also the graphical representations of the different kinds of arc are shown.

Inhibitor arcs

Regarding inhibitor arcs, the first attempt is to remove all of them, replacing, e.g. the transition `inp-(a,Q,R)` with:

`newinp-(a,Q,R)` $\quad inp(a)?Q\_R \longrightarrow dec(R)$.

This approach seems to work, because of the structure of the net we are studying: the only places tested for absence are $\langle a\rangle$, and the only transition producing tokens in $\langle a\rangle$ is `rend(a)`. Moreover, transition `rend(a)` performs very smooth transformations on the marking, leaving unchanged the places corresponding to sequential subprograms. For this reason, it seems possible to change the position of some `rend` transitions in a firing sequence, without compromising its firability, and reaching the same marking.

As an example, consider the P/T system of Figure 2(2) obtained by applying the above transformation to the contextual P/T system $Net(\langle\langle a\rangle\rangle|\langle\langle a\rangle\rangle|inp(a)?P\_Q)$ represented in Figure 2(1). It permits the firability of the firing sequence $\sigma =$ `rend(a) rend(a) newinp-(a,P,Q)`. The corresponding firing sequence $\sigma' =$ `rend(a) rend(a) inp-(a,P,Q)` is not firable in Figure 2(1) due to the inhibitor arc connecting the place $\langle a\rangle$ to the transition `inp-(a,P,Q)`. Nevertheless, we can obtain the legal firing sequence $\sigma'' =$ `inp-(a,P,Q) rend(a) rend(a)`, leading to the same marking, only by delaying all the `rend` transitions that produce the tokens in the inhibiting place mentioned above.

However, the following two problems arise.

- **Initially present messages:**
  A first problem is due to the presence of messages in the initial state. Let Div be a divergent agent, e.g. $Div = out(c).!in(c).out(c)$. Consider the agent $Q = \langle a\rangle|inp(a)?Div\_0$.

In $Net(Q)$, reported in Figure 3(1), the only firable transition is `inp+(a,`Div`,0)`, leading to the marking $dec(Div)$; it is easy to see that no dead marking is reachable from it. By substituting a self-loop for each read arc, and by eliminating inhibitor arcs, we obtain the net in Figure 3(2): `newinp-(a,`Div`,0)` can fire, leading to the dead marking containing only one token in $\langle a\rangle$; hence the behaviour w.r.t. deadlock is not preserved.

This is due to the fact that place $\langle a\rangle$ contains some tokens at the initial marking; as these tokens are not produced by the occurrence of a transition `rend(a)`, we cannot apply the trick described above to obtain a legal firing sequence, because all previously reached markings already contain tokens in $\langle a\rangle$. To apply the trick, we have to wait that all tokens in $\langle a\rangle$ at the initial marking are consumed; at this point, if the current marking contains k tokens in place $\langle a\rangle$, then at least k transitions `rend(a)` have occurred.

We solve the problem in this way (see Figure 3(3)): we add an auxiliary place be_a, whose contents equals the number of tokens consumed from place $\langle a\rangle$. This place is used by transitions inhibited by $\langle a\rangle$ to be sure that all the tokens in the initial marking have been consumed. Place be_a is empty in the initial marking and is filled with a token by each transition consuming a message $\langle a\rangle$. Denoting with $\#_P(a)$ the number of tuples $\langle a\rangle$ present in the initial TS of P, we add a self-loop on be_a, with arc weights $\#_P(a)$, to all transitions inhibited by $\langle a\rangle$ in the P/T net representation of the agent P.

- **Messages tested for presence:**
  Another problem arises with transitions testing for presence of messages. Consider the agent $R = out(a)|rd(a).out(b)|in(b).inp(a)?Div\_0$; Figure 4(1) represents $Net(R)$ while Figure 4(2) is the P/T system obtained after the application of all the transformations described above. $Net(R)$ has no dead markings: indeed, it can only start with the firing sequence $\sigma =$ `out(a,0) rend(a) rd(a,`$out(b)$`) out(b,0) rend(b) in(b,`$inp(a)?Div\_0$`)`. After its firing, a token is contained in $\langle a\rangle$, hence the only transition that can be fired is `inp+(a,`Div`,0)`, leading to the marking $dec(Div)$, from which no dead marking is reachable. In the net in Figure 4(2), after the firing of σ also `newinp-(a,`Div`,0)` can fire, leading to the dead marking $\{\langle a\rangle, be_a\}$.

The above example indicates that the trick of delaying the `rend(a)` transitions is not applicable in general. In particular, a `rend(a)` cannot be delayed after a `rd(a,P)` if the token tested for presence by the latter transition is the same token produced by the former one.

More in detail, the problem of the messages tested for presence is due to the fact that if a transition, testing without consuming a token in $\langle a\rangle$, happens, we are sure that at least one token is contained in $\langle a\rangle$ in the reached marking; thus we should not fire transition `newinp-(a,P,Q)` until that token is consumed by some transition.

Consider a firing sequence leading to a marking where place $\langle a\rangle$ contains a single token; suppose also that the transition $t =$ `rend(a)` producing it occurs before $u =$ `rd(a,P)`, which in turn is not followed by any transition removing tokens from $\langle a\rangle$. Then the token tested for presence by u is exactly the one produced by t, hence the sequence obtained by removing t is not firable.

We solve the problem by adding two auxiliary places, ab_a and pr_a, for each message name a. A token in pr_a denotes the fact that we are sure that at least one token is contained in $\langle a\rangle$. Place ab_a is the complementary of pr_a, i.e. it contains a token iff pr_a is

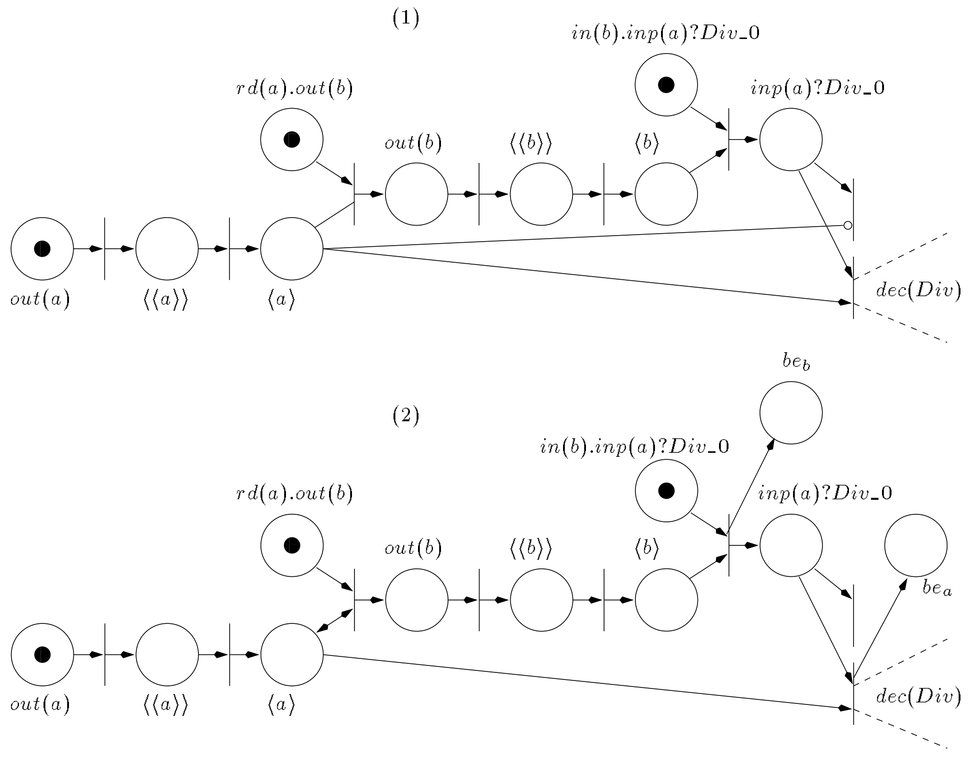

Figure 4: Transforming $Net(out(a)|rd(a).out(b)|in(b).inp(a)?Div\_0)$.

`In`$_{pr}$`(a,Q)`	$in(a).Q,\ \langle a\rangle,\ pr_a \longrightarrow dec(Q),\ be_a,\ ab_a$
`In`$_{ab}$`(a,Q)`	$in(a).Q,\ \langle a\rangle,\ ab_a \longrightarrow dec(Q),\ be_a,\ ab_a$
`Rd`$_{pr}$`(a,Q)`	$rd(a).Q,\ \langle a\rangle,\ pr_a \longrightarrow dec(Q),\ \langle a\rangle,\ pr_a$
`Rd`$_{ab}$`(a,Q)`	$rd(a).Q,\ \langle a\rangle,\ ab_a \longrightarrow dec(Q),\ \langle a\rangle,\ pr_a$
`Out(a,Q)`	$out(a).Q \longrightarrow \langle\langle a\rangle\rangle,\ dec(Q)$
`Rend(a)`	$\langle\langle a\rangle\rangle \longrightarrow \langle a\rangle$
`Inp+`$_{pr}$`(a,Q,R)`	$inp(a)?Q\_R,\ \langle a\rangle,\ pr_a \longrightarrow dec(Q),\ be_a,\ ab_a$
`Inp+`$_{ab}$`(a,Q,R)`	$inp(a)?Q\_R,\ \langle a\rangle,\ ab_a \longrightarrow dec(Q),\ be_a,\ ab_a$
`Rdp+`$_{pr}$`(a,Q,R)`	$rdp(a)?Q\_R,\ \langle a\rangle,\ pr_a \longrightarrow dec(Q),\ \langle a\rangle,\ pr_a$
`Rdp+`$_{ab}$`(a,Q,R)`	$rdp(a)?Q\_R,\ \langle a\rangle,\ ab_a \longrightarrow dec(Q),\ \langle a\rangle,\ pr_a$
`Inp-(a,Q,R)`	$inp(a)?Q\_R,\ \#_P(a)\cdot be_a,\ ab_a \longrightarrow dec(R),\ \#_P(a)\cdot be_a,\ ab_a$
`Rdp-(a,Q,R)`	$rdp(a)?Q\_R,\ \#_P(a)\cdot be_a,\ ab_a \longrightarrow dec(R),\ \#_P(a)\cdot be_a,\ ab_a$
`!In`$_{pr}$`(a,Q)`	$!in(a).Q,\ \langle a\rangle,\ pr_a \longrightarrow dec(Q),\ !in(a).Q,\ be_a,\ ab_a$
`!In`$_{ab}$`(a,Q)`	$!in(a).Q,\ \langle a\rangle,\ ab_a \longrightarrow dec(Q),\ !in(a).Q,\ be_a,\ ab_a$

Table 5: P/T net semantics that preserves deadlock (parametric on agent P).

empty; it is used to enable each transition in the P/T net corresponding to a transition inhibited by place $\langle a\rangle$ in $Net(P)$. The sum of tokens in pr_a and ab_a is equal to 1 in each reached marking.

The correct information in these places is maintained by transitions testing or consuming tokens from $\langle a\rangle$. After a testing without consuming transition, we are sure that at least one token is present in $\langle a\rangle$; after a consuming transition, we are no longer sure of this fact, i.e. $\langle a\rangle$ could be empty. We split each of these transitions into two transitions, according to the two possible contents of the auxiliary places. The pr_a-version of the consuming transition removes the token from pr_a and puts a token in ab_a, whereas the ab_a-version has a self-loop on place ab_a; the pr_a-version of the testing transition has a self-loop on place pr_a, whereas the ab_a-version removes the token from ab_a and puts a token in pr_a.

Putting together the mechanisms listed above, we obtain the transformation from $Net(P)$ to the P/T system $DNet(P)$, which turns out to be equivalent to $Net(P)$ w.r.t. deadlock. Figure 5 illustrates the complete transformation for the net in Figure 4(1). Observe that in the marking in which the place $inp(a)?Div\_0$ contains a token, the place ab_a is empty; this ensures that no dead marking can be reached.

Before formally defining $DNet(P)$, we need some auxiliary notation.

- With $\#_P(a)$ we denote the number of tuples $\langle a\rangle$ available in the term P:

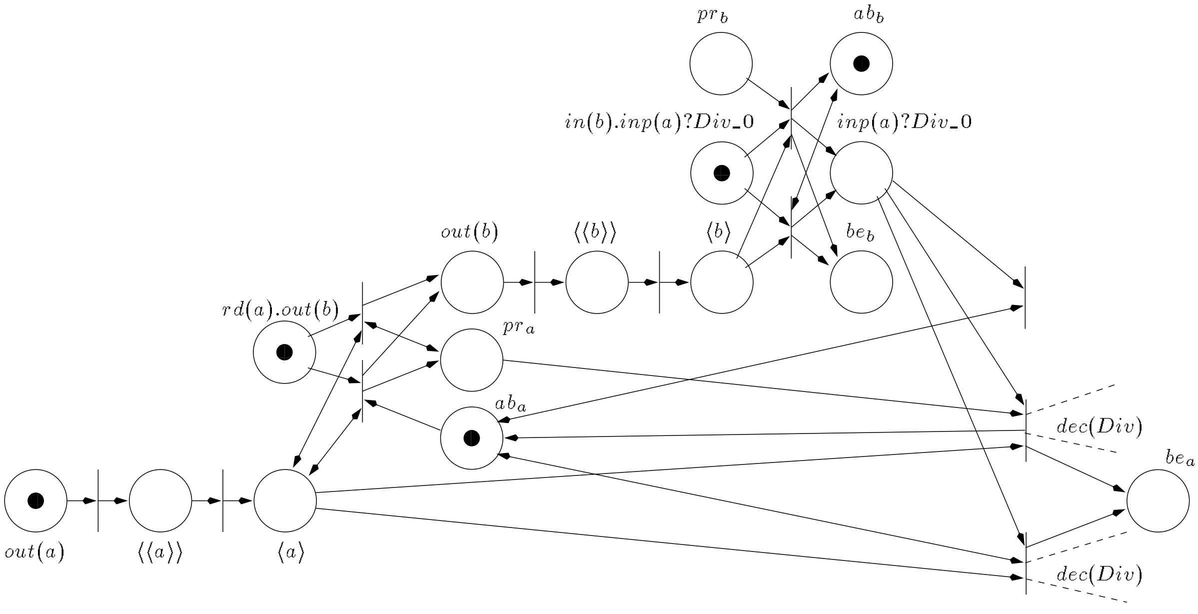

Figure 5: $DNet(out(a)|rd(a).out(b)|in(b).inp(a)?Div\_0)$.

$$\begin{array}{lcll} \#_P(a) & = & 0 & \text{if } P \text{ is a sequential program} \\ \#_{\langle a\rangle}(a) & = & 1 & \\ \#_{\langle\langle a\rangle\rangle}(a) & = & 0 & \\ \#_{P_1|P_2}(a) & = & \#_{P_1}(a) + \#_{P_2}(a) & \end{array}$$

- $\mathcal{T}'_P$ contains all the transitions obtained as instances of the axiom schemata parametric in the agent P presented in Table 5. The definition is parametric because the transitions `Inp-(a,Q,R)` and `Rdp-(a,Q,R)` use $\#_P(a)$, i.e. the number of occurrences of $\langle a\rangle$ initially available in the agent P.

Definition 4.16 Let P be an agent. We define the triple $DNet(P) = (\tilde{S}, \tilde{T}, \tilde{m}_0)$, where:

$$\begin{array}{lcl} \tilde{S} & = & \{Q \mid Q \text{ is a sequential subprogram of } P\} \cup \\ & & \{\langle a\rangle, \langle\langle a\rangle\rangle, be_a, pr_a, ab_a \mid a \text{ is a message name in } P\} \\ \tilde{T} & = & \{c \longrightarrow p \in \mathcal{T}'_P \mid \exists\, Q \text{ sequential subprogram of } P \text{ s.t. } Q \in dom(c)\} \cup \\ & & \{\texttt{Rend(a)} \mid a \text{ is a message name in } P\} \\ \tilde{m}_0 & = & dec(P) \oplus \{ab_a \mid a \text{ is a message name in } P\} \end{array}$$

■

Also in this case $DNet(P)$ is well defined, as it is a P/T system; moreover it is finite.

Proposition 4.17 Given the agent P, $DNet(P)$ is a finite P/T system.
Proof Similar to the proof of Proposition 4.6. ■

Now we are ready to show that $Net(P)$ and $DNet(P)$ have the same behaviour w.r.t. deadlock. The proof proceeds as follows: first we show that, given a dead marking in $Net(P)$, there exists a dead marking in $DNet(P)$ (and the two markings coincide on common places); then we prove that also the vice versa holds. Both proofs make use of results relating firing sequences of $Net(P)$ with firing sequences in $DNet(P)$.

The following function, parametric w.r.t. $\{pr, ab\}$, maps transitions in $Net(P)$ to transitions in $DNet(P)$:

$F_{pr}(\texttt{rend(a)}) = F_{ab}(\texttt{rend(a)}) = \texttt{Rend(a)}$
$F_{pr}(\texttt{out(a,Q)}) = F_{ab}(\texttt{out(a,Q)}) = \texttt{Out(a,Q)}$
$F_{pr}(\texttt{inp-(a,Q,R)}) = F_{ab}(\texttt{inp-(a,Q,R)}) = \texttt{Inp-(a,Q,R)}$
$F_{pr}(\texttt{rdp-(a,Q,R)}) = F_{ab}(\texttt{rdp-(a,Q,R)}) = \texttt{Rdp-(a,Q,R)}$

$F_{pr}(\texttt{in(a,Q)}) = \texttt{In}_{pr}\texttt{(a,Q)}$	$F_{ab}(\texttt{in(a,Q)}) = \texttt{In}_{ab}\texttt{(a,Q)}$
$F_{pr}(\texttt{rd(a,Q)}) = \texttt{Rd}_{pr}\texttt{(a,Q)}$	$F_{ab}(\texttt{rd(a,Q)}) = \texttt{Rd}_{ab}\texttt{(a,Q)}$
$F_{pr}(\texttt{inp+(a,Q,R)}) = \texttt{Inp+}_{pr}\texttt{(a,Q,R)}$	$F_{ab}(\texttt{inp+(a,Q,R)}) = \texttt{Inp+}_{ab}\texttt{(a,Q,R)}$
$F_{pr}(\texttt{rdp+(a,Q,R)}) = \texttt{Rdp+}_{pr}\texttt{(a,Q,R)}$	$F_{ab}(\texttt{rdp+(a,Q,R)}) = \texttt{Rdp+}_{ab}\texttt{(a,Q,R)}$
$F_{pr}(\texttt{!in(a,Q)}) = \texttt{!In}_{pr}\texttt{(a,Q)}$	$F_{ab}(\texttt{!in(a,Q)}) = \texttt{!In}_{ab}\texttt{(a,Q)}$

The reverse mapping is defined as $G(\tilde{t}) = t$ iff $(F_{pr}(t) = \tilde{t}) \vee (F_{ab}(t) = \tilde{t})$. Observe that G is well defined as whenever there exists $\tilde{t}$, t, and t' such that $F_{pr}(t) = \tilde{t}$ and $F_{ab}(t') = \tilde{t}$ then $t = t'$. Function G is extended to sequences of transitions by elementwise application.

We define some sets of transitions that will be useful in the following: $IN(a)$ and $RD(a)$ are the sets of transitions (in any net $Net(P)$) consuming (resp. testing for presence of) tokens from place $\langle a\rangle$; $\tilde{IN}(a)$ and $\tilde{RD}(a)$ are the corresponding sets for net $DNet(P)$.

$$
\begin{aligned}
IN(a) &= \{\texttt{in(a,Q)}, \texttt{inp+(a,Q,R)}, \texttt{!in(a,Q)} \mid Q, R \text{ are sequential programs}\} \\
RD(a) &= \{\texttt{rd(a,Q)}, \texttt{rdp+(a,Q,R)} \mid Q, R \text{ are sequential programs}\} \\
\tilde{IN}(a) &= \{F_{pr}(t), F_{ab}(t) \mid t \in IN(a)\} \\
\tilde{RD}(a) &= \{F_{pr}(t), F_{ab}(t) \mid t \in RD(a)\}
\end{aligned}
$$

The transitions in $IN(a)$ and $\tilde{IN}(a)$ are called in-transitions; the ones in $RD(a)$ and $\tilde{RD}(a)$ rd-transitions.

Let P be an agent; the proposition below provides some relation between two transitions in $\mathcal{T}$ and $\mathcal{T}'_P$ related by G:

1. the changes produced by the firing of two related transitions on respective markings are the same on common (i.e. not auxiliary) places;

2. the second item will be used to relate the firability of two related transitions in markings agreeing on common places.

Proposition 4.18 Let P be an agent, $t \in \mathcal{T}$, $\tilde{t} \in \mathcal{T}'_P$, $t = G(\tilde{t})$ (or, equivalently, $\tilde{t} = F_{ab}(t)$ or $\tilde{t} = F_{pr}(t)$) and $s \in \mathcal{S}$.

1. $(\tilde{t}^\bullet \setminus {}^\bullet\tilde{t})(s) = (t^\bullet \setminus {}^\bullet t)(s)$

2. ${}^\bullet\tilde{t}(s) = ({}^\bullet t \oplus \hat{t})(s)$
   if t is not an instance of `inp-` or `rdp-` then ${}^\circ t = \emptyset$

Proof By case analysis on the axioms of Table 4. ■

The following proposition enunciates some general properties about reachable markings in $DNet(P)$, telling that places be_a, ab_a and pr_a behave as expected:

1. the number of tokens in place be_a corresponds to the number of occurred in-transitions;
2. the sum of tokens in places ab_a and pr_a is always equal to 1;
3. place ab_a contains a token iff each rd-transition is followed by an in-transition.

Proposition 4.19 Let P be an agent and $DNet(P) = (\tilde{S}, \tilde{T}, \tilde{m}_0)$.
If $\tilde{m}_0 \xrightarrow{\tilde{t}_1} \tilde{m}_1 \ldots \xrightarrow{\tilde{t}_n} \tilde{m}_n$ then

1. $\tilde{m}_n(be_a) = |\{k \mid 1 \leq k \leq n \wedge \tilde{t}_k \in \tilde{IN}(a)\}|$
2. $\tilde{m}_n(pr_a) + \tilde{m}_n(ab_a) = 1$
3. $\tilde{m}_n(ab_a) = 1$ iff $\forall k((1 \leq k \leq n \wedge \tilde{t}_k \in \tilde{RD}(a)) \Rightarrow \exists h(k < h \leq n \wedge \tilde{t}_h \in \tilde{IN}(a)))$

Proof By induction on n. ■

The lemma below relates the firing sequences of $Net(P)$ with the ones in $DNet(P)$: for each firing sequence in $Net(P)$ there exists a firing sequence in $DNet(P)$, obtained by replacing each transition with its image via F_{ab} or F_{pr}, such that the reached markings coincide on common places.

Lemma 4.20 Let P be an agent, $Net(P) = (S, T, m_0)$ and $DNet(P) = (\tilde{S}, \tilde{T}, \tilde{m}_0)$.
If $m_0 \xrightarrow{t_1} m_1 \ldots \xrightarrow{t_n} m_n$ then $\tilde{m}_0 \xrightarrow{\tilde{t}_1} \tilde{m}_1 \ldots \xrightarrow{\tilde{t}_n} \tilde{m}_n$, where

1. $\tilde{m}_n(s) = m_n(s)$ for all $s \in S$
2. $\tilde{t}_i = \begin{cases} F_{pr}(t_i) & \text{if } t_i \in IN(a) \cup RD(a) \wedge \tilde{m}_{i-1}(pr_a) = 1 \\ F_{ab}(t_i) & \text{otherwise} \end{cases} \quad \text{for } 1 \leq i \leq n$

Proof By induction on n.
For $n = 0$ the only nonempty property is 1, which easily follows from the definitions of $Net(P)$ and $DNet(P)$.
Suppose $m_0 \xrightarrow{t_1} m_1 \ldots \xrightarrow{t_{n-1}} m_{n-1} \xrightarrow{t_n} m_n$;
by inductive hypothesis, $\tilde{m}_0 \xrightarrow{\tilde{t}_1} \tilde{m}_1 \ldots \xrightarrow{\tilde{t}_{n-1}} \tilde{m}_{n-1}$, $\tilde{m}_{n-1}(s) = m_{n-1}(s)$ for all $s \in S$ and condition 2 of the lemma is satisfied for $1 \leq i \leq n-1$.

We show that there exists $\tilde{t}_n$, satisfying condition 2 of the lemma, that is enabled at $\tilde{m}_{n-1}$. Three cases can happen:

- t_n is not an `inp-` or a `rdp-` transition.

  The following cases can happen:

  - t_n is not a in- or a rd-transition.
    Hence, t_n = `rend(a)` or t_n = `out(a)` for some message name a. As $\tilde{m}_{n-1}(s) = m_{n-1}(s)$ for all $s \in S$, by item 2 of Proposition 4.18 both $F_{pr}(t_n)$ and $F_{ab}(t_n)$ are enabled at $\tilde{m}_{n-1}$.

- Exists a such that $t_n \in IN(a) \cup RD(a)$ and $\tilde{m}_n(pr_a) \geq 1$.

  As $\tilde{m}_{n-1}(s) = m_{n-1}(s)$ for all $s \in S$, by item 2 of Proposition 4.18 we have that $F_{pr}(t_n)$ is enabled at $\tilde{m}_{n-1}$.

- Exists a such that $t_n \in IN(a) \cup RD(a)$ and $\tilde{m}_n(pr_a) = 0$.

  By item 2 of Proposition 4.19, $\tilde{m}_n(pr_a) = 0$ implies $\tilde{m}_n(ab_a) = 1$. Proceeding as above we obtain that $F_{ab}(t_n)$ is enabled at $\tilde{m}_{n-1}$.

- $t_n =$ `inp-(a,Q,R)`.

  Take $\tilde{t}_n =$ `Inp-(a,Q,R)`; as `Inp-(a,Q,R)` $= F_{ab}($`inp-(a,Q,R)`$)$, condition 2 of the lemma is satisfied.

  We show that $\tilde{t}_n$ is enabled at $\tilde{m}_{n-1}$.

  - We show that $\tilde{m}_{n-1}(inp(a)?Q\_R) > 0$.

    As t_n is enabled at m_{n-1}, we have that $m_{n-1}(inp(a)?Q\_R) > 0$. As m_{n-1} and $\tilde{m}_{n-1}$ coincide on common places, then also $\tilde{m}_{n-1}(inp(a)?Q\_R) > 0$.

  - We show that $\tilde{m}_{n-1}(be_a) \geq \#_P(a)$.

    As t_n is enabled at m_{n-1}, we have that $m_{n-1}(\langle a \rangle) = 0$. By definition of $Net(P)$, $m_0(\langle a \rangle) = \#_P(a)$. Observing that the only transitions able to remove tokens from place $\langle a \rangle$ are those in $IN(a)$ (and they remove exactly one token), it follows that at least $\#_P(a)$ transitions in $IN(a)$ occurred, i.e. $|\{i \mid 1 \leq i \leq n-1 \wedge t_i \in IN(a)\}| \geq \#_P(a)$. By definition of F_{ab} (resp. F_{pr}) we have that $t_i \in IN(a)$ iff $F_{ab}(t_i) \in \tilde{IN}(a)$ (resp. $F_{pr}(t_i) \in \tilde{IN}(a)$). Condition 2 of the lemma, holding by inductive hypothesis for $1 \leq i \leq n-1$, ensures that $t_i \in IN(a)$ iff $\tilde{t}_i \in \tilde{IN}(a)$ for $1 \leq i \leq n-1$; hence $|\{i \mid 1 \leq i \leq n-1 \wedge \tilde{t}_i \in \tilde{IN}(a)\}| = |\{i \mid 1 \leq i \leq n-1 \wedge t_i \in IN(a)\}|$. By item 1 of proposition 4.19 we have that $\tilde{m}_{n-1}(be_a) = |\{i \mid 1 \leq i \leq n-1 \wedge \tilde{t}_i \in \tilde{IN}(a)\}|$, hence $\tilde{m}_{n-1}(be_a) \geq \#_P(a)$.

  - We show that $\tilde{m}_{n-1}(ab_a) = 1$.

    We start showing that each transition in $RD(a)$ is followed by at least one transition in $IN(a)$. Suppose there exists $i \in \{1, \ldots, n-1\}$ such that $t_i \in RD(a)$ and $t_j \notin IN(a)$ for $i < j \leq n-1$. As $t_i \in RD(a)$, we have that $m_{i-1}(\langle a \rangle) > 0$ and also $m_i(\langle a \rangle) > 0$. As there is no subsequent transition consuming tokens from $\langle a \rangle$, we have that $m_{n-1}(\langle a \rangle) > 0$, contradicting the fact that t_n is enabled at m_{n-1}.

    As observed above, by inductive hypothesis and by definition of F_{ab} and F_{pr}, we have that $t_i \in IN(a)$ iff $\tilde{t}_i \in \tilde{IN}(a)$ for $1 \leq i \leq n-1$. It is easy to see that the same holds for sets $RD(a)$ and $\tilde{RD}(a)$. Hence, also in the firing sequence $\tilde{t}_1 \ldots \tilde{t}_{n-1}$ each transition in $\tilde{RD}(a)$ is followed by a transition in $\tilde{IN}(a)$. Thus, by item 3 of Proposition 4.19 we have that $\tilde{m}_{n-1}(ab_a) = 1$.

- $t_n =$ `rdp-(a,Q,R)`.

  Similar to the case above.

Let $\tilde{m}_n$ be the marking produced by the firing of $\tilde{t}_n$. As by inductive hypothesis we have that $m_{n-1}(s) = \tilde{m}_{n-1}(s)$ for all $s \in S$, by item 1 of Proposition 4.18 also the markings m_n and $\tilde{m}_n$ coincide on common places. ■

The following theorem states that to a dead marking m in $Net(P)$ corresponds a dead marking in $DNet(P)$, reached by a firing sequence obtained from the firing sequence leading to m by replacing each transition with its image via either F_{ab} or F_{pr}. To show that the reached marking in $DNet(P)$ is dead, the only difficult case is if an instance of `Inp-` or `Rdp-` can fire, solved by showing that places $inp(a)?Q\_R$ and $rdp(a)?Q\_R$ are always empty in dead markings of $Net(P)$.

Theorem 4.21 Let P be an agent, $Net(P) = (S, T, m_0)$ and $DNet(P) = (\tilde{S}, \tilde{T}, \tilde{m}_0)$. If $Net(P)$ has a deadlock then $DNet(P)$ has a deadlock.
Proof Suppose $m_0 \xrightarrow{\sigma} m \not\longrightarrow$. By Lemma 4.20, $\tilde{m}_0 \xrightarrow{\tilde{\sigma}} \tilde{m}$ with $\tilde{m}(s) = m(s)$ for all $s \in S$. We show that $\tilde{m} \not\longrightarrow$.
Suppose $\tilde{m} \xrightarrow{\tilde{t}}$. Three cases can happen:

- $\tilde{t}$ is not an instance of `Inp-` or `Rdp-`.

  As $\tilde{m}(s) = m(s)$ for all $s \in S$, by item 2 of Proposition 4.18 we have that also the transition $G(\tilde{t})$ in $Net(P)$ can fire at m, contradiction.

- $\tilde{t} =$ `Inp-(a,Q,R)`.

  Thus, $\tilde{m}(inp(a)?Q\_R) > 0$; as markings m and $\tilde{m}$ coincide on common places, also $m(inp(a)?Q\_R) > 0$. If $m(\langle a \rangle) > 0$, then the transition `inp+(a,Q,R)` can fire at m, else the transition `inp-(a,Q,R)` can fire, contradiction.

- $\tilde{t} =$ `Rdp-(a,Q,R)`.

  Managed as the previous case. ■

We need the following auxiliary (partial) functions on sequences of transitions in $Net(P)$: $rmlast^a_k$ removes the last k occurrences of transition `rend(a)`, whereas $rmrend$ removes all transitions `rend`.

Definition 4.22

$$rmlast^a_{i+1}(\sigma t) = \begin{cases} rmlast^a_i(\sigma) & \text{if } t = \texttt{rend(a)} \\ rmlast^a_{i+1}(\sigma)t & \text{otherwise} \end{cases}$$

$$rmlast^a_0(\sigma) = \sigma$$

$$rmrend(\sigma t) = \begin{cases} rmrend(\sigma) & \text{if } \exists a(t = \texttt{rend(a)}) \\ rmrend(\sigma)t & \text{otherwise} \end{cases}$$

$$rmrend(\varepsilon) = \varepsilon.$$ ■

Note that $rmlast^a_k(\sigma)$ is defined only if σ contains at least k occurrences of transition `rend(a)`.

The following lemma enunciates an important property of firing sequences in $Net(P)$: if

1. the reached marking contains at least k tokens in place $\langle a \rangle$,
2. all the tokens in $\langle a \rangle$ at the initial marking have been consumed by in-transitions (i.e. the number of occurrences of in-transitions is greater or equal to the number of tokens in $\langle a \rangle$ at the initial marking) and
3. each rd-transition is followed by an in-transition

then the firing sequence obtained by dropping the last k occurrences of `rend(a)` is still firable at the initial marking.

Lemma 4.23 Let P be an agent, $Net(P) = (S, T, m_0)$ and a a message name. If

1. $m_0 \xrightarrow{t_1} \ldots \xrightarrow{t_n} m \oplus k \cdot \langle a \rangle$,
2. $|\{i \mid 1 \leq i \leq n \wedge t_i \in IN(a)\}| \geq m_0(\langle a \rangle)$ and
3. $\forall i(1 \leq i \leq n \wedge t_i \in RD(a) \Rightarrow \exists j(i < j \leq n \wedge t_j \in IN(a)))$

then $m_0 \xrightarrow{\sigma} m \oplus k \cdot \langle\langle a \rangle\rangle$, where $\sigma = rmlast^a_k(t_1 \ldots t_n)$.
Proof By induction on k.
The case $k = 0$ is trivial.
If $k > 0$, we have that $m_0 \xrightarrow{t_1} \cdots \xrightarrow{t_n} (m \oplus \langle a \rangle) \oplus (k-1) \cdot \langle a \rangle$.
By inductive hypothesis, we have that $m_0 \xrightarrow{u_1} m'_1 \cdots \xrightarrow{u_{n-k+1}} m'_{n-k+1}$, where $u_1 \ldots u_{n-k+1} = rmlast^a_{k-1}(t_1 \ldots t_n)$ and $m'_{n-k+1} = (m \oplus \langle a \rangle) \oplus (k-1) \cdot \langle\langle a \rangle\rangle$.
We start showing that the transition sequence $u_1 \ldots u_{n-k+1}$ contains at least a `rend(a)` transition.
As $rmlast^a_{k-1}$ removes `rend(a)` transitions only, it is easy to see that properties 2 and 3 of the lemma imply the following two conditions, dealing with the transition sequence $u_1 \ldots u_{n-k+1}$ instead of $t_1 \ldots t_n$:

2′ $|\{i \mid 1 \leq i \leq n-k+1 \wedge u_i \in IN(a)\}| \geq m_0(\langle a \rangle)$ and

3′ $\forall i(1 \leq i \leq n-k+1 \wedge u_i \in RD(a) \Rightarrow \exists j(i < j \leq n \wedge u_j \in IN(a)))$

By property 2′ (saying that the number of transitions in $IN(a)$ is greater or equal to the initial marking of place $\langle a \rangle$) and observing that $m'_{n-k+1}(\langle a \rangle) > 0$, we have that at least one transition `rend(a)` occurs in $u_1 \ldots u_{n-k+1}$.
Let u_l be the last occurrence of `rend(a)`: we show that the sequence $u_1 \ldots u_{l-1} u_{l+1} \ldots u_{n-k+1}$ is firable from m_0.
Take $m''_i = m'_i \setminus \langle a \rangle \oplus \langle\langle a \rangle\rangle$.
We want to show that $m_0 \xrightarrow{u_1} m'_1 \ldots \xrightarrow{u_{l-1}} m'_{l-1} \xrightarrow{u_{l+1}} m''_{l+1} \ldots \xrightarrow{u_{n-k+1}} m''_{n-k+1}$.
The part $m_0 \xrightarrow{u_1} m'_1 \ldots \xrightarrow{u_{l-1}} m'_{l-1}$ is valid by inductive hypothesis. As $u_l =$ `rend(a)`, from $m'_{l-1} \xrightarrow{u_l} m'_l$ we have that $m'_l = m'_{l-1} \setminus \langle\langle a \rangle\rangle \oplus \langle a \rangle$; hence, by definition of m''_l, $m''_l = m'_{l-1}$.
Thus, it remains to show that $m''_l \xrightarrow{u_{l+1}} \ldots \xrightarrow{u_{n-k+1}} m''_{n-k+1}$.
We prove that u_{i+1} is enabled at m''_i for $l \leq i < n-k+1$. Let $\#^i_{IN(a)} = |\{j \mid i < j \leq n-k+1 \wedge u_j \in IN(a)\}|$ be the number of transitions in $IN(a)$ occurring after u_i. In the following we will use the following fact: as u_l is the last transition producing tokens in $\langle a \rangle$ and $m'_{n-k+1}(\langle a \rangle) > 0$, we have that $m'_i(\langle a \rangle) > \#^i_{IN(a)}$. We now consider u_{i+1} with $l \leq i < n-k+1$; three cases can happen:

- $u_{i+1} \in IN(a)$.
  As $u_{i+1} \in IN(a)$, we have $\#^i_{IN(a)} \geq 1$, hence $m'_i(\langle a \rangle) > 1$. By definition of m''_i we have $m''_i(\langle a \rangle) > 0$. As m'_i and m''_i coincide on places different from $\langle a \rangle$, $\langle\langle a \rangle\rangle$, and because a transition in $IN(a)$ needs exactly one token in $\langle a \rangle$ (and no tokens in $\langle\langle a \rangle\rangle$), we have that u_{i+1} is enabled at m''_i.

- $u_{i+1} \in RD(a)$.
  By $3'$ there exists h such that $i+1 < h \leq n-k+1$ and $u_h \in IN(a)$. Hence $\#^i_{IN(a)} \geq 1$; proceeding as above we have $m''_i(\langle a \rangle) > 0$. Following the reasoning used above, we can show that u_{i+1} is enabled at m''_i.

- $u_{i+1} \notin IN(a) \cup RD(a)$.
  As m''_i is obtained from m'_i by removing a token in $\langle a \rangle$ and adding a token in $\langle\langle a \rangle\rangle$, and seeing that u_{i+1} does not have $\langle a \rangle$ in its pre or test set and does not have $\langle\langle a \rangle\rangle$ in its inhibitor set, we have that u_{i+1} is enabled at m''_i.

Now we show that by firing u_{i+1} at m''_i we reach m''_{i+1}, i.e. that $m''_{i+1} = m''_i \setminus {}^\bullet u_{i+1} \oplus u^\bullet_{i+1}$.
We need the following auxiliary result: $\langle a \rangle \subseteq m'_j$ for $l \leq j \leq n-k+1$. By definition of u_l the last transition producing tokens in $\langle a \rangle$ is u_l; this implies $m'_l(\langle a \rangle) \geq m'_{l+1}(\langle a \rangle) \geq \ldots \geq m'_{n-k+1}(\langle a \rangle)$. As $m'_{n-k+1}(\langle a \rangle) > 0$ then $m'_j(\langle a \rangle) > 0$ (i.e. $\langle a \rangle \subseteq m'_j$) for $l \leq j \leq n-k+1$.
As $m''_i = m'_i \setminus \langle a \rangle \oplus \langle\langle a \rangle\rangle$, showing that $m''_{i+1} = m''_i \setminus {}^\bullet u_{i+1} \oplus u^\bullet_{i+1}$ is the same as proving that $m''_{i+1} = (m'_i \setminus \langle a \rangle \oplus \langle\langle a \rangle\rangle) \setminus {}^\bullet u_{i+1} \oplus u^\bullet_{i+1}$.
By definition, $m''_{i+1} = m'_{i+1} \setminus \langle a \rangle \oplus \langle\langle a \rangle\rangle$; as $m'_i \xrightarrow{u_{i+1}} m'_{i+1}$ we have $m'_{i+1} = m'_i \setminus {}^\bullet u_{i+1} \oplus u^\bullet_{i+1}$, hence $m''_{i+1} = (m'_i \setminus {}^\bullet u_{i+1} \oplus u^\bullet_{i+1}) \setminus \langle a \rangle \oplus \langle\langle a \rangle\rangle$.
It remains to show that $(m'_i \setminus \langle a \rangle \oplus \langle\langle a \rangle\rangle) \setminus {}^\bullet u_{i+1} \oplus u^\bullet_{i+1} = (m'_i \setminus {}^\bullet u_{i+1} \oplus u^\bullet_{i+1}) \setminus \langle a \rangle \oplus \langle\langle a \rangle\rangle$.
The above equality can be easily derived from the following facts: $\langle a \rangle \subseteq m'_i$, $\langle a \rangle \subseteq m'_{i+1}$, ${}^\bullet u_{i+1} \subseteq m'_i$ and ${}^\bullet u_{i+1} \subseteq m''_i$ (the last two statements are true because u_{i+1} is enabled at both m'_i and m''_i).
As $u_1 \ldots u_{n-k+1} = rmlast^a_{k-1}(t_1 \ldots t_n)$ and u_l is the last `rend(a)` transition occurring in $u_1 \ldots u_{n-k+1}$, we have that $u_1 \ldots u_{l-1} u_{l+1} \ldots u_{n-k+1} = rmlast^a_k(t_1 \ldots t_n)$. Moreover, as $m''_{n-k+1} = m'_{n-k+1} \setminus \langle a \rangle \oplus \langle\langle a \rangle\rangle$ and $m'_{n-k+1} = m \oplus \langle a \rangle \oplus (k-1) \cdot \langle\langle a \rangle\rangle$, we get $m''_{n-k+1} = m \oplus k \cdot \langle\langle a \rangle\rangle$. ■

The following lemma relates firing sequences of $DNet(P)$ with firing sequences of $Net(P)$; though the correspondence on markings is preserved, we do not have the same close correspondence on transitions as for Lemma 4.20: it may happen that the transition `Inp-(a,Q,R)` fires in $DNet(P)$ even if place $\langle a \rangle$ is not empty, thus forbidding the firing of the corresponding transition `inp-(a,Q,R)` in $Net(P)$. However, the enabledness of `Inp-(a,Q,R)` ensures that the hypothesis of Lemma 4.23 are satisfied; defining k as the number of tokens in $\langle a \rangle$, we can obtain a firing sequence in $Net(P)$ by dropping the last k `rend(a)` transitions, then fire `inp-(a,Q,R)` (that has become enabled) and finally fire the dropped k `rend(a)` transitions.

Lemma 4.24 Let P be an agent, $Net(P) = (S, T, m_0)$ and $DNet(P) = (\tilde{S}, \tilde{T}, \tilde{m}_0)$.
If $\tilde{m}_0 \xrightarrow{\tilde{\sigma}} \tilde{m}$ then there exist a marking m and a permutation τ of $G(\tilde{\sigma})$ such that

1. $rmrend(G(\tilde{\sigma})) = rmrend(\tau)$,
2. $m_0 \xrightarrow{\tau} m$ and
3. $m(s) = \tilde{m}(s)$ for all $s \in S$.

Proof By induction on $|\tilde{\sigma}|$.
If $|\tilde{\sigma}| = 0$ the theorem trivially follows from definitions of $Net(P)$ and $DNet(P)$.
Take $\tilde{\sigma} = \tilde{\sigma}'\tilde{t}$; hence there exists $\tilde{m}'$ such that $\tilde{m}_0 \xrightarrow{\tilde{\sigma}'} \tilde{m}' \xrightarrow{\tilde{t}} \tilde{m}$. By inductive hypothesis, there exists a permutation τ' of $G(\tilde{\sigma}')$ such that

1. $rmrend(G(\tilde{\sigma}')) = rmrend(\tau')$,
2. $m_0 \xrightarrow{\tau'} m'$ and
3. $m'(s) = \tilde{m}'(s)$ for all $s \in S$.

We show that there exists a permutation of $G(\tilde{\sigma})$ satisfying the conditions of the lemma. We distinguish the following cases:

- $\tilde{t}$ is not an instance of `Inp-` or `Rdp-`.
  Take the transition sequence $\tau' G(\tilde{t})$; as τ' is a permutation of $G(\tilde{\sigma}')$, also $\tau' G(\tilde{t})$ is a permutation of $G(\tilde{\sigma}'\tilde{t})$.

  We show that the requirements of the lemma are satisfied by the sequence $\tau' G(\tilde{t})$:

  1. By item 1 of inductive hypothesis and by definition of $rmrend$ we have that $rmrend(G(\tilde{\sigma}'\tilde{t})) = rmrend(\tau' G(\tilde{t}))$.
  2. As $\tilde{m}' \xrightarrow{\tilde{t}} \tilde{m}$, by item 3 of inductive hypothesis and item 2 of Proposition 4.18 we have that $G(\tilde{t})$ is enabled at m'. Denoting with m the marking resulting from the firing of $G(\tilde{t})$ at m', we have $m' \xrightarrow{G(\tilde{t})} m$. By item 2 of inductive hypothesis we get $m_0 \xrightarrow{\tau' G(\tilde{t})} m$.
  3. By item 3 of inductive hypothesis and item 1 of Proposition 4.18 it follows that $m(s) = \tilde{m}(s)$ for all $s \in S$.

- $\tilde{t} =$ `Inp-(a,Q,R)`.
  The enabledness of $\tilde{t}$ at $\tilde{m}'$ ensures that $\tilde{m}'(inp(a)?Q\_R) > 0$; by item 3 of inductive hypothesis also $m'(inp(a)?Q\_R) > 0$.

  Two cases can happen:

  - if $m'(\langle a \rangle) = 0$, then also $G(\tilde{t}) =$ `inp-(a,Q,R)` is enabled at m'. Following the above reasoning we can show that also in this case the three properties of the lemma are satisfied.
  - if $m'(\langle a \rangle) > 0$, take k such that $m' = m'' \oplus k \cdot \langle a \rangle$ and $m''(\langle a \rangle) = 0$.
    We show that the firing sequence $m_0 \xrightarrow{\tau'} m'$ satisfies the conditions of Lemma 4.23:

    1. We have $m_0 \xrightarrow{\tau'} m'' \oplus k \cdot \langle a \rangle$.
    2. As $\tilde{t}$ is enabled at $\tilde{m}'$, $\tilde{m}'(be_a) \geq \#_P(a)$; by item 1 of Proposition 4.19, at least $\#_P(a)$ occurrences of transitions in the sequence $\tilde{\sigma}'$ must belong to the set of transitions $\tilde{IN}(a)$. Seeing that (by definition of G) $\tilde{t}' \in \tilde{IN}(a)$ iff $G(\tilde{t}') \in IN(a)$ for all transitions $\tilde{t}'$ of $DNet(P)$, also the sequence $G(\tilde{\sigma}')$ contains at least $\#_P(a)$ occurrences of transitions in $IN(a)$. Since τ' is a permutation of $G(\tilde{\sigma}')$, the same holds for τ'. The second condition of Lemma 4.23 is now satisfied, because $m_0(\langle a \rangle) = \#_P(a)$ by definition of $Net(P)$.
    3. As $\tilde{t}$ is enabled at $\tilde{m}'$, $\tilde{m}'(ab_a) = 1$; by item 3 of Proposition 4.19, each transition in $\tilde{\sigma}'$ belonging to $\tilde{RD}(a)$ is followed by a transition in $\tilde{IN}(a)$. As $\tilde{t}' \in \tilde{IN}(a)$ iff $G(\tilde{t}') \in IN(a)$, and the same holds also replacing $\tilde{IN}(a)$ (and $IN(a)$) with $\tilde{RD}(a)$ (and $RD(a)$), also in $G(\tilde{\sigma}')$ each transition belonging to

$RD(a)$ is followed by a transition in $IN(a)$. As $rmrend(G(\tilde{\sigma}')) = rmrend(\tau')$, the property holds also in τ'. Then the condition 3 of Lemma 4.23 is satisfied.

By applying Lemma 4.23, we have $m_0 \xrightarrow{\rho'} m'' \oplus k \cdot \langle\langle a \rangle\rangle$, where $\rho' = rmlast^a_k(\tau')$. Take $\tau = \rho' G(\tilde{t}) u_1 \dots u_k$, where $u_j =$ `rend(a)` for $1 \leq j \leq k$. Remembering that $rmlast^a_k$ removes k occurrences of transition `rend(a)` from τ', and that τ' is a permutation of $G(\tilde{\sigma}')$, it is easy to see that τ is a permutation of $G(\tilde{\sigma}'\tilde{t})$. Now we show that the requirements of the lemma are satisfied by τ.

1. By inductive hypothesis $rmrend(G(\tilde{\sigma}')) = rmrend(\tau')$; by definition of τ we have $rmrend(\tau) = rmrend(\tau' G(\tilde{t}))$, thus we obtain $rmrend(G(\tilde{\sigma}'\tilde{t})) = rmrend(\tau)$.
2. We show that τ is firable at m_0. We have already observed that the prefix ρ' of τ can be fired at m_0. It remains to show that $G(\tilde{t}) u_1 \dots u_k$, where $u_j =$ `rend(a)`, is firable at the reached marking $m'' \oplus k \cdot \langle\langle a \rangle\rangle$. As $m'(inp(a)?Q\_R) > 0$, and observing that m' and m'' differ only on place $\langle a \rangle$, we have that also $m''(inp(a)?Q\_R) > 0$; as $m''(\langle a \rangle) = 0$ (by definition of m''), the transition `inp-(a,Q,R)` $= G(\tilde{t})$ can be fired at $m'' \oplus k \cdot \langle\langle a \rangle\rangle$. As the firing of $G(\tilde{t})$ does not change the marking of the place $\langle\langle a \rangle\rangle$, at least k tokens are present in $\langle\langle a \rangle\rangle$ even after its firing; hence, the firing sequence $u_1 \dots u_k$ can be fired at that place.
3. We have already observed that τ is a permutation of $G(\tilde{\sigma}'\tilde{t})$. As a permutation of the ordering of transitions does not influence the reached marking, following the lines for the previous case we can show that also property 3 of the lemma is satisfied by τ.

- $\tilde{t} =$ `Rdp-(a,Q,R)`.

  Similar to the previous case. ■

Finally, we have that a dead marking of $Net(P)$ corresponds to a dead marking of $DNet(P)$.

Theorem 4.25 Let P be an agent, $Net(P) = (S, T, m_0)$ and $DNet(P) = (\tilde{S}, \tilde{T}, \tilde{m}_0)$. If $DNet(P)$ has a deadlock then also $Net(P)$ has a deadlock.

Proof Suppose $\tilde{m}_0 \xrightarrow{\tilde{\sigma}} \tilde{m} \not\longrightarrow$. By Lemma 4.24 there exists a permutation τ of $G(\tilde{\sigma})$ such that $rmrend(G(\tilde{\sigma})) = rmrend(\tau)$, $m_0 \xrightarrow{\tau} m$, and $m(s) = \tilde{m}(s)$ for all $s \in S$.
We show that $m \not\longrightarrow$.
Suppose that $m \xrightarrow{t}$. Three cases can happen:

- t is not an instance of `inp-` or `rdp-`.

  It is easy to see that, by item 2 of Proposition 4.18 and item 2 of Proposition 4.19, at least one of the transitions $F_{ab}(t)$, $F_{pr}(t)$ can fire at $\tilde{m}$.

- $t =$ `inp-(a,Q,R)`.

  We show that `Inp-(a,Q,R)` is enabled at $\tilde{m}$.

  - As t is enabled at m, we have $m(inp(a)?Q\_R) > 0$. As m and $\tilde{m}$ coincide on common places, also $\tilde{m}(inp(a)?Q\_R) > 0$.

  - As t is enabled at m, we have $m(\langle a \rangle) = 0$; hence, all the tokens in $\langle a \rangle$ in the initial marking have been consumed, i.e. there are at least $\#_P(a)$ occurrences of transitions in $IN(a)$ in τ. As $G(\tilde{\sigma})$ is a permutation of τ, the same property holds in $G(\tilde{\sigma})$. Noting that $\tilde{t} \in \tilde{IN}(a)$ iff $G(\tilde{t}) \in IN(a)$ for all transitions $\tilde{t}$ of $DNet(P)$, there are at least $\#_P(a)$ occurrences of transitions in $\tilde{IN}(a)$ also in $\tilde{\sigma}$. By item 1 of Proposition 4.19 we get $\tilde{m}(be_a) \geq \#_P(a)$.
  - Suppose that a transition in $RD(a)$ occurs in τ: it has to be followed by a transition in $IN(a)$, otherwise $m(\langle a \rangle) > 0$, contradicting the fact that t is enabled at m. As $rmrend(\tau) = rmrend(G(\tilde{\sigma}))$, the same property holds for $G(\tilde{\sigma})$. Noting that $\tilde{t} \in \tilde{IN}(a)$ iff $G(\tilde{t}) \in IN(a)$ and the same holds after replacing $\tilde{IN}(a)$ (and $IN(a)$) with $\tilde{RD}(a)$ (and $RD(a)$), the property continues to hold for $\tilde{\sigma}$; by item 3 of Proposition 4.19 then $\tilde{m}(ab_a) = 1$.

  As $\tilde{m}(inp(a)?Q\_R) > 0$, $\tilde{m}(be_a) \geq \#_P(a)$, and $\tilde{m}(ab_a) = 1$, the transition `Inp-(a,Q,R)` is enabled at $\tilde{m}$, contradiction.

- $t =$ `rdp-(a,Q,R)`.

  Analogous to the previous one. ■

The above theorems allow us to conclude that the termination problem is decidable in $\mathbf{L}_u$, as we have shown that also the presence of deadlock in finite P/T systems is decidable.

Corollary 4.26 $\mathbf{L}_u$ is not Turing powerful.
Proof By contradiction, if $\mathbf{L}_u$ is Turing powerful then it allows to program all the computable functions. But, for every possible encoding P of the function f with input i, the problem of termination of P is decidable. ■

5 Conclusion

This paper comes in pair with [BGZ97b], where we studied the operational and observational semantics of three different interpretations for the output operation for the subcalculus not containing *rd* and *rdp*. Besides the ordered and unordered semantics presented here, we also studied the so-called *instantaneous* semantics, according to which $out(a).P$ is structural equivalent to $\langle a \rangle | P$ (adopted in the asynchronous π-calculus [HT91, Bou92]). The behavioural semantics for the three operational semantics was defined following the widely accepted approach of [MS92]. The main results of [BGZ97b] are that a variant of the asynchronous bisimulation [ACS98] is the correct semantics for the instantaneous interpretation, while the correct semantics for the other two cases is a variant of the classic (synchronous) bisimulation [Mil89], where inputs and outputs are treated symmetrically. In [BGZ97b] we also claimed the non Turing completeness of the unordered semantics; this result is strengthened (it holds also in presence of *rd* and *rdp*) and proved in full detail here.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments.

References

[ACS98] R. Amadio, I. Castellani, and D. Sangiorgi. On Bisimulations for the Asynchronous π-Calculus. *Theoretical Computer Science*, 195(2):291-324, 1998.

[Bou92] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA, Sophia–Antipolis,1992.

[BC91] A. Brogi and P. Ciancarini. The concurrent language Shared Prolog. *ACM Transactions on Programming Languages and Systems*, 13(1):99–123, 1991.

[Bus98] N. Busi. Petri Nets with Inhibitor and Read Arcs: Semantics, Analysis and Application to Process Calculi, Ph.D. Thesis, Department of Mathematics, University of Siena, Italy, 1998.

[BG95] N. Busi, R. Gorrieri. A Petri Net Semantics for π-calculus, In *Concur'95*, volume 962 of *LNCS*, 145–159, 1995.

[BGZ97a] N. Busi, R. Gorrieri, and G. Zavattaro. A Truly Concurrent view of Linda Interprocess Communication. Technical Report UBLCS 97–2, Department of Computer Science, University of Bologna, February 1997.

[BGZ97b] N. Busi, R. Gorrieri, and G. Zavattaro. Three Semantics of the Output Operation for Generative Communication. In *Coordination'97*, volume 1282 of *LNCS*, 205–219, 1997.

[BGZ97c] N. Busi, R. Gorrieri, and G. Zavattaro. A Process Algebraic View of Linda Coordination Primitives. *Theoretical Computer Science*, 192(2): 167–199, 1998.

[BGZ97d] N. Busi, R. Gorrieri, and G. Zavattaro. On the Turing Equivalence of Linda Coordination Primitives. In *Express'97*, volume 7 of *Electronic Notes in Theoretical Computer Science*.

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

[Car87] N. Carriero. Implementation of Tuple Space Machines. Research Report YALEU/DCS/RR-567, 1987.

[CG89] N. Carriero and D. Gelernter. How to Write Parallel Programs: A Guide to the Perplexed. *ACM Computing Surveys*, volume 21(3):323–357, 1989.

[CG90] N. Carriero and D. Gelernter. *How to Write Parallel Programs: A First Course*. MIT Press, 1990.

[CGZ95] N. Carriero, D. Gelernter, and L. Zuck. Bauhaus Linda. In *Object-Based Models and Languages for Concurrent Systems*, volume 924 of *LNCS*, pages 66–76. Springer Verlag, 1995.

[CEP95] A. Cheng, J. Esparza, J. Palsberg. Complexity results for 1-safe nets. In *Theoretical Computer Science*, 147:117–136, 1995.

[CJY94] 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*, volume 924 of *LNCS*, pages 77–106, Springer, 1994.

[DWR95] A. Douglas, A. Wood, and A. Rowstron. Linda implementation revisited. Transputer and Occam Developments, pages 125–138, IOS Press, 1995.

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

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

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

[Gro93] J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118:263-299, 1993.

[HT91] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communication. In *ECOOP '91*, volume 512 of *LNCS*, pages 133–147. Springer Verlag, 1991.

[Jag90] S. Jagannathan. Semantics and Analysis of First-Class Tuple Spaces. Technical Report YALEU/DCS/TR-783, Department of Computer Science, Yale University, April 1990.

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

[Mil91] R. Milner. The Polyadic π-Calculus: A Tutorial Technical Report, Department of Computer Science, University of Edinburgh, ECS–LFCS–91–180, October 1991.

[MS92] R. Milner and D. Sangiorgi. Barbed Bisimulation. In *ICALP'92*, volume 623 of *LNCS*, pages 685–695, Springer Verlag, 1992.

[Min67] M. L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Englewood Cliffs, 1967.

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

[Nar89] J. E. Narem. An Informal Operational Semantics of C-Linda V2.3.5. Technical Report, Department of Computer Science, Yale University, YALEU/DCS/TR-839, December 1989.

[Reu88] C. Reutenauer. *The Mathematics of Petri Nets.* Masson, 19I88.

[SCA95] Scientific Computing Associates. *Linda: User's guide and reference manual.* Scientific Computing Associates, 1995.

[SS63] J. C. Shepherdson and J. E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.