Proving the correctness of optimising destructive and non-destructive reads over tuple spaces

Rocco De Nicola[1], Rosario Pugliese[1], and Antony Rowstron[2]

[1] Dipartimento di Sistemi ed Informatica, Università di Firenze,
Via C. Lombroso, 6/17 50135 Firenze, Italy.
{denicola, pugliese}@dsi.unifi.it
[2] Microsoft Research Ltd, St. George House,
1 Guildhall Street, Cambridge, CB2 3NH, UK.
antr@microsoft.com

Abstract. In this paper we describe the proof of an optimisation that can be applied to tuple space based run-time systems (as used in Linda). The optimisation allows, under certain circumstances, for a tuple that has been destructively removed from a shared tuple space (for example, by a Linda in) to be returned as the result for a non-destructive read (for example, a Linda rd) for a different process. The optimisation has been successfully used in a prototype run-time system.

1 Introduction

In this paper we present the proof of an optimisation that can be applied to tuple space based run-time systems, which was first presented in Rowstron [1]. Examples of tuple space based systems are JavaSpaces [2], KLAIM [3], Linda [4], PageSpace [5], TSpaces [6], TuCSoN [7] and WCL [8] to name just a few.

Throughout this paper we will just use the three standard Linda tuple space access primitives:

out(tuple) Insert a tuple into a tuple space.

in(template) If a tuple exists that matches the template then remove the tuple and return it to the process performing the `in`. If no matching tuple is available then the process blocks until a matching tuple is available.

rd(template) If a tuple exists that matches the template then return a copy of the tuple to the process that performed the `rd`. If there is no matching tuple then the process blocks until a matching tuple is available.

Moreover, we shall assume that a single global tuple space is being used by all processes.

The optimisation proved in this paper is referred to as *tuple ghosting*. The (informal) semantics of the `in` primitive leads implementers to remove the tuple that is returned to a process from the tuple space as soon as the `in` primitive is completed. Tuple ghosting allows the tuple to potentially remain as a valid result tuple for a non-destructive read performed by another process whilst a set of assumptions holds.

Studying the soundness of the optimisation has been highly valuable; it showed that the original algorithm [1] was too optimistic and allowed the result to remain visible for too long. In certain circumstances, the original rules used for the optimisation altered the semantics of the access primitives. We are confident now that the actual optimisation, modified to use the semantics given in Section 3, is sound.

1.1 Motivation for the optimisation

Optimisation of tuple removal is useful because often tuples are used to store shared state between processes. For instance, a list is usually stored in a tuple space so that the items of the list are stored in separate tuples, with each tuple containing a unique number as the first field, representing its position in the list. A single tuple is required that contains a shared counter indicating the number of the next element that can be added. In order to add an element to the list, the shared counter is removed using an `in`, the value of the counter increased and the tuple is re-inserted, and then a new tuple is inserted containing the number of the counter and the data as an element in the list. This is a common operation and there have been proposals for the addition of new primitives to help performing the update of the shared counter (see e.g. Eilean [9]), and when using compile-time analysis, to convert the counter updating into a single operation [10]. The proposals were made with the intention of increasing concurrency. Additionally, in high performance servers the cost of managing a primitive blocked waiting for a matching tuple is greater than finding a matching tuple and not blocking. One of the new challenges for tuple space implementers is to create large-scale high throughput servers, and therefore optimisations that reduce the server load are important.

1.2 Implementation

Tuple ghosting has been implemented in a Java run-time environment and it has proved to be clean and efficient. To provide tuple ghosting the implementation uses the following informal rules.

When a tuple is returned as the result of an `in`:

1. the same tuple cannot be returned as a result of another `in`;
2. the process, which performed the `in` that matched the tuple, cannot access the tuple anymore;
3. when the process which performed the `in` on the tuple performs any tuple space access or terminates, the tuple is removed.

The kernel works by marking tuples as ghosted once they have been returned by an `in` primitive. Every process using the kernel has a Globally Unique Identifier (GUID) created dynamically as it starts to execute. When the process registers with the run-time system the GUID is passed to the run-time system and it creates a primitive counter associated with the process. Each time a process performs a tuple space access, the counter associated with the process is

incremented by one (before the primitive is performed). When a process requests a tuple using an `in`, the matched tuple is marked as "ghosted" and tagged with the identity tag of the process that removed the tuple and with the current value of the primitive counter associated with the process. Any other process can then perform a `rd` and have this tuple as the result. However, whenever the tuple is matched the system compares on-the-fly the current primitive counter associated with the GUID attached to the tuple with the primitive counter attached to the tuple. If the primitive counters differ or if the process has terminated then the tuple is discarded, and not used as a result for the `rd`.

All communication between the processes must occur through the shared tuple space. Hidden communication between the processes would allow the processes to determine that one had read a tuple after it had been destructively removed by another. Process termination is an example of hidden communication (where, for example, one process is started after another process terminates). The starting process can deduce that any tuples removed by the terminated process should not exist. Therefore, accounting for termination is important.

The rules that the kernel uses are described in detail in Section 3.

1.3 Performance

Table 1 shows some experimental results which demonstrate the advantages of using tuple ghosting using the example of a list stored in a tuple space. We use the scenario that the list is accessed by two reader processes that read the counter 20 times, and a writer processes that appends 40 elements to the end of the list (update the counter and add a new element).

The experimental run-time was written in Java, with the reader and writer processes running as Java threads. The results were gathered on a Pentium II 400 MHz PC. The results shown in Table 1 are the average times of 20 executions with tuple ghosting both enabled and disabled. The execution times (with standard deviations) are shown for the three processes. For the reader processes, the number of blocked and ghosted `rd` primitives are also shown. A ghosted `rd` is one that would have blocked if tuple ghosting was not enabled.

The results show (as expected) that no `rd` primitive leads to blocking when tuple ghosting is enabled, but when ghosting is disabled we have that 70% of the `rd` primitives do lead to a block. Tuple ghosting has therefore increased the level of concurrency achieved in the system. In addition, the execution times are reduced when the tuple ghosting is enabled. This is due to the overhead associated with managing a `rd` that is blocked because no tuple is available.

The rest of the paper is structured as follows. In the next section the structural operational semantics for a traditional Linda implementation is outlined, then in Section 3 the optimisation is outlined in more detail, and the structural operational semantics for the optimised Linda implementation is presented. The proof of correctness of the optimised version is then given in Section 4.

		Ghosting disabled		Ghosting enabled	
		Value	St. Dev.	Value	St. Dev.
Reader 1	Time (ms)	**4367**	566	**185**	39
	No. of blocking `rd`	**15.75**	1.37	**0**	0
	No. of ghosted `rd`	0	0	9.75	0.64
Reader 2	Time (ms)	**4281**	743	**194**	37
	No. of blocking `rd`	**15.35**	1.81	**0**	0
	No. of ghosted `rd`	0	0	9.9	0.85
Writer	Time (ms)	**4886**	670	**590**	71

Table 1. Performance of the implementation with tuple ghosting enabled and disabled.

2 Structural Operational Semantics for a Linda Kernel

2.1 Syntax

The three standard Linda tuple space primitives are the elementary actions that processes can perform. Processes are constructed by using three composition operators: the *null* process *nil* is a process constant that denotes a terminated process, the *action prefix* operator $a._-$ is a unary operator that denotes a process that first executes action a and then behaves as its process argument, and the *parallel composition* operator $\_ \parallel \_$ is a binary operator that denotes the concurrent execution of its two arguments. Processes can also consist of *evaluated* tuples (they are a separate syntactic category), that represent tuples that have been added to the tuple space (as in [11]). An evaluated tuple is denoted by $\underline{out}(v)$ with $v \in VAL$.

We assume the existence of some predefined syntactic categories that processes can use. EXP, the category of *value expressions*, which is ranged over by e, contains a set of *variable symbols*, VAR, ranged over by x, y and z, and a non-empty countable set of value symbols, VAL, ranged over by v.

To give a simpler presentation of our formal framework, we make a few symplifying assumptions. We assume that tuples and templates consists of just one field. The only difference between tuples and templates is that the formers can only contain expressions (or values) while the latters can also contain formal parameters (i.e. variables to be assigned). A parameter x is denoted by $\underline{x}$.

By summarizing, the syntax of the language is

$$P, Q \ ::= \ nil \mid a.P \mid P \parallel Q \mid P \parallel O \ \mid \ O \parallel P$$
$$O \ ::= \ \underline{out}(v) \ \mid \ O_1 \parallel O_2$$
$$a \ ::= \ out(e) \mid rd(t) \mid in(t)$$
$$t \ ::= \ e \mid \underline{x}$$

Variables which occur in formal parameters of a template t are *bound* by $rd(t)._-$ and $in(t)._-$. If P is a process, we let $bv(P)$ denote the set of bound variables in P and $fv(P)$ denote that of free variables in P. Sets $bv(\_)$ and $fv(\_)$ can be inductively defined as follows:

$$
\begin{array}{ll}
fv(nil) \stackrel{\text{def}}{=} \emptyset & bv(nil) \stackrel{\text{def}}{=} \emptyset \\
fv(a.P) \stackrel{\text{def}}{=} fv(P) \setminus bv(a) & bv(a.P) \stackrel{\text{def}}{=} bv(P) \cup bv(a) \\
fv(P \parallel Q) \stackrel{\text{def}}{=} fv(P) \cup fv(Q) & bv(P \parallel Q) \stackrel{\text{def}}{=} bv(P) \cup bv(Q) \\
fv(P \parallel O) \stackrel{\text{def}}{=} fv(P) & bv(P \parallel O) \stackrel{\text{def}}{=} bv(P) \\
fv(O \parallel P) \stackrel{\text{def}}{=} fv(P) & bv(O \parallel P) \stackrel{\text{def}}{=} bv(P)
\end{array}
$$

$$
\begin{array}{ll}
fv(out(e)) \stackrel{\text{def}}{=} \begin{cases} \{e\} \text{ if } e \in VAR \\ \emptyset \quad \text{otherwise} \end{cases} & bv(out(e)) \stackrel{\text{def}}{=} \emptyset \\
fv(in(t)) \stackrel{\text{def}}{=} \begin{cases} \{t\} \text{ if } t \in VAR \\ \emptyset \quad \text{otherwise} \end{cases} & bv(in(t)) \stackrel{\text{def}}{=} \begin{cases} \{x\} \text{ if } t = \underline{x} \\ \emptyset \quad \text{otherwise} \end{cases} \\
fv(rd(t)) \stackrel{\text{def}}{=} \begin{cases} \{t\} \text{ if } t \in VAR \\ \emptyset \quad \text{otherwise} \end{cases} & bv(rd(t)) \stackrel{\text{def}}{=} \begin{cases} \{x\} \text{ if } t = \underline{x} \\ \emptyset \quad \text{otherwise} \end{cases}
\end{array}
$$

As usual, we write $P[v/t]$ to denote the term obtained by substituting each free occurrence of t in P with v, whenever $t \in VAR$, and to denote P, otherwise.

2.2 Operational semantics

The operational semantics assumes the existence of a function for evaluating value expressions; $[\![\cdot]\!] : EXP \longrightarrow VAL$. So, $[\![e]\!]$ will then denote the value of expression e, provided it does not contain variables.

The operational semantics of the language is defined in the SOS style [12] by means of a *Labelled Transition System* (LTS). This LTS is the triple $(\mathcal{P}_1, \mathcal{L}_1, \longrightarrow_1)$ where:

- $\mathcal{P}_1$, ranged over by P and Q, is the set of processes generated by the syntax given in Section 2.1.
- $\mathcal{L}_1 \stackrel{\text{def}}{=} \{out(v), rd(v), in(v) | v \in VAL\}$ is the set of *labels* (we shall use a to range over $\mathcal{L}_1$ and s over $\mathcal{L}_1^*$).
- $\longrightarrow_1 \subseteq \mathcal{P}_1 \times \mathcal{L}_1 \times \mathcal{P}_1$, called the *transition relation*, is the least relation induced by the operational rules in Table 2 (to give a simpler presentation of the rules, we rely on a *structural relation* defined as the least equivalence relation closed under parallel composition that satisfies the structural rules in Table 2). We shall write $P \stackrel{a}{\longrightarrow} Q$ instead of $(P, a, Q) \in \longrightarrow_1$.

For $s \in \mathcal{L}_1^*$ and $P, Q \in \mathcal{P}_1$, we shall write $P \stackrel{s}{\longrightarrow} Q$ to denote that $P = Q$, if $s = \epsilon$, and that $\exists P_1, \ldots, P_{n-1} \in \mathcal{P}_1 : P \stackrel{a_1}{\longrightarrow} P_1 \stackrel{a_2}{\longrightarrow} \ldots P_{n-1} \stackrel{a_n}{\longrightarrow} Q$, if $s = a_1 a_2 \ldots a_n$.

Let us briefly comment on the rules in Table 2. The structural laws simply say that, as expected, parallel composition is commutative, associative and has *nil* as the identity element. The operational rules S1-S5 should be self-explanatory. Rule S4 says that a read operation can be performed only if there is a tuple matching the template used by the operation. To check pattern-matching, condition "$[\![t]\!] = v \vee t \in VAR$" is used; it is satisfied when either t is an expression

Structural Rules

$$P \parallel nil \equiv P \qquad P \parallel Q \equiv Q \parallel P \qquad P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$$

Operational Rules

S1 $$a.P \xrightarrow{a} P$$

S2 $$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q}$$

S3 $$\frac{P \xrightarrow{out(e)} P' \quad \wedge \quad [\![e]\!] = v}{P \xrightarrow{out(v)} P' \parallel \underline{out}(v)}$$

S4 $$\frac{P \xrightarrow{rd(t)} P' \quad \wedge \quad ([\![t]\!] = v \vee t \in VAR)}{P \parallel \underline{out}(v) \xrightarrow{rd(v)} P'[v/t] \parallel \underline{out}(v)}$$

S5 $$\frac{P \xrightarrow{in(t)} P' \quad \wedge \quad ([\![t]\!] = v \vee t \in VAR)}{P \parallel \underline{out}(v) \xrightarrow{in(v)} P'[v/t]}$$

S6 $$\frac{P \equiv Q \quad \wedge \quad Q \xrightarrow{a} Q' \quad \wedge \quad Q' \equiv P'}{P \xrightarrow{a} P'}$$

Table 2. Linda Operational Semantics

that evaluates to v (the value stored in the tuple) or t is a variable (a variable matches any value). Rule S5 differs from S4 just for the management of the accessed tuple: indeed, in S5, the tuple is consumed, while, in S4, the tuple is left untouched. Finally, rule S6 ensures that the structural relation does not modify the behaviour of processes.

3 Structural Semantics for Optimised Linda

Having described the basic Linda structural semantics, we now consider the structural semantics for the optimised Linda implementation that uses tuple ghosting. In order to illustrate tuple ghosting in more detail, let us consider two very simple processes that interact through the tuple space. Their actions are shown in Table 3.

We shall use Petri Nets and their unfoldings as case graphs to describe the difference between the "classical" and the "optimized" semantics. In a Petri net the circles represent places, and the squares represent transitions. A transition can fire only when all the places that are preconditions for that transition contain

Process A	Process B
A_1 out(a)	B_1 in(a)
A_2 rd(a)	B_2 out(b)
A_3 rd(b)	B_3 out(a)

Table 3. Two simple example processes.

tokens. When a transition fires it consumes the tokens in its preconditions and places a token in each of the output places that are linked to it by arcs.

The Petri net and the case graph showing the parallel composition of our two processes can be seen in Figure 1. If one ignores the dotted links in the figure, then the Petri net and the case graph are those created according to the semantics for the primitives as given in the previous section.

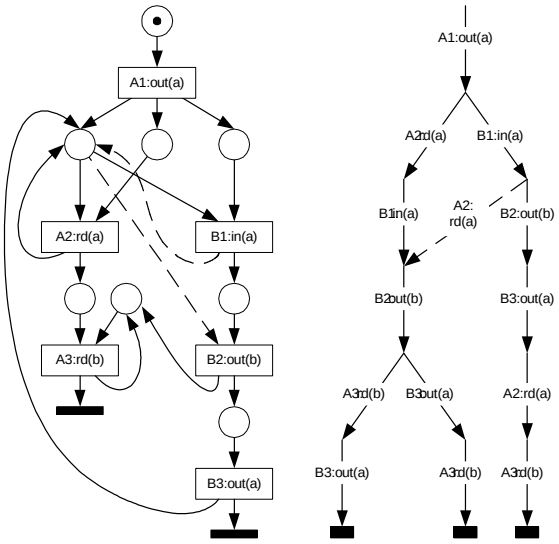

Fig. 1. A Petri Net and case graph for processes A and B.

In Figure 1 the token starts in the initial place, and the only transition that can fire is `A1:out(a)`. When this fires, a token is placed in the three output places connected to the transition. This means that either the transitions `A2:rd(a)` or `B1:in(a)` can fire. If `B1:in(a)` fires then the other cannot fire, because the token is removed from one of its preconditions. This token is replaced when the transition `B3:out(a)` is fired. If `A2:rd(a)` fires, then the precondition tokens are consumed, but the transition is linked to one of its own preconditions. So, a token is reinserted in that place. However, the same rule cannot re-fire because the other precondition does not contain a token any longer. This means that the

transition `B1:in(a)` is the only one that can fire, as it is the only transition that has all precondition places filled with a token. The case graph shown in the same figure shows the different ordering of the possible transition firings (of course, the dotted arc has to be ignored).

In Figure 1 the dotted arcs represent the tuple ghosting optimisation. We allow the transition `A2:rd(a)` to fire after the transition `B1:in(a)` fires. This means that the manipulation of a tuple has been suspended in the middle of the operation; Process B has performed the `in` operation and has received the tuple and can continue, but the tuple is not actually removed whilst Process A cannot know that process B has received the tuple. This only occurs when there is the possibility of a synchronisation between the two processes, which happens using the tuple `b`, when Process B inserts it.

From the global perspective, this appears to be incorrect; it allows the reading of a tuple that should have been removed. We will now present the formal semantics of the optimised version, and then show the proof that the two semantics are equivalent.

3.1 Optimized operational semantics

The *optimized* operational semantics of the language is defined by means of another LTS. To this aim, we assume the existence of a set of process *locations*, Loc, ranged over by ℓ, where the parallel components of processes can be allocated, and of a distinct location, τ, where evaluated tuples are placed. We denote by $\underline{Loc}$ a disjoint set of *ghost* locations (where *ghost* tuples can be placed) which is in bijection with Loc via the operation $\underline{\cdot}$. Finally, we let $LOC = Loc \cup \underline{Loc} \cup \{\tau\}$, ranged over by λ, be the set of all locations. Locations shall be used to model the GUID given to processes in the implementation.

The idea is that Linda processes are statically allocated, e.g. distributed over a net of processors, once and for all. The names of locations and the distribution of processes over locations can be arbitrarily chosen. Hence, for any given process P, its distribution is determined by the number of its parallel components, i.e. by the number of occurrences of the parallel operator which are not guarded by any action. For instance, the process $\underline{out}(1) \parallel \underline{out}(2).(\underline{out}(3) \parallel \underline{out}(4))$ has initially two parallel components (although, after the execution of the $\underline{out}(2)$ operation, it could be composed of three parallel processes) and can be allocated over, at most, two processors. This means that, as far as distribution is concerned, we have conceptually two different parallel operators and it is convenient to use different notations for them: we shall use $|$ to denote the occurrences of the parallel operator that do not cause distribution of their components, e.g. those occurrences guarded by some action, and shall still use $\parallel$ for the other occurrences, e.g. (some of) the unguarded occurrences. Obviously, the semantics of $|$ is defined by rules analogues to S2 and to the structural ones.

To manage locations we introduce two new operators: an *allocator* operator $\lambda :: P$, that says that process P is allocated at location λ, and a *location remover* operator $P \setminus \lambda$, that says that location λ (and the process located there) must be removed from P.

The optimized LTS is the triple $(\mathcal{P}_2, \mathcal{L}_2, \longrightarrow_2)$ where:

- $\mathcal{P}_2$, ranged over by P and Q, is the set of processes generated by the syntax given in Section 2.1 extended with the following productions

  $P, Q \ ::= \ \ldots \mid P \mid Q \mid \lambda :: P \mid P \setminus \lambda$

  $\mathcal{P}_2$ also contains the *distributed* versions of processes from $\mathcal{P}_1$.
- $\mathcal{L}_2 \stackrel{\text{def}}{=} \{out(v)@\lambda, rd(v)@\lambda, in(v)@\lambda, stop@\lambda | v \in VAL, \lambda \in LOC\}$ is the set of *labels* (we shall use $\alpha@\lambda$ to range over $\mathcal{L}_2$ and σ over $\mathcal{L}_2^*$).
- $\longrightarrow_2 \subseteq \mathcal{P}_2 \times \mathcal{L}_2 \times \mathcal{P}_2$, called the *transition relation*, is the least relation closed under parallel composition that satisfies the operational rules in Table 4 (again, to give a simpler presentation of the rules, we rely on a *structural relation* defined as the least equivalence relation closed under parallel composition that satisfies the structural rules in Table 4). We shall write $P \xrightarrow{\alpha@\lambda} Q$ instead of $(P, \alpha@\lambda, Q) \in \longrightarrow_2$.

For $\sigma \in \mathcal{L}_2^*$ and $P, Q \in \mathcal{P}_2$, we shall write $P \xrightarrow{\sigma} Q$ to denote that $P = Q$, if $\sigma = \epsilon$, and that $\exists P_1, \ldots, P_{n-1} \in \mathcal{P}_2 : P \xrightarrow{\alpha_1@\lambda_1} P_1 \xrightarrow{\alpha_2@\lambda_2} \ldots P_{n-1} \xrightarrow{\alpha_n@\lambda_n} Q$, if $\sigma = \alpha_1@\lambda_1 \cdot \alpha_2@\lambda_2 \cdot \ldots \cdot \alpha_n@\lambda_n$.

Let us briefly comment on the rules in Table 4. The additional structural laws say that the location remover distributes with respect to parallel composition and that the removal just concerns the location (and the process located there) explicitely named by the operator. The operational rules should be quite explicative. The general idea is the following. Tuples are initially allocated at location τ. When a tuple located at τ is accessed by an `in` action performed by a process located at ℓ, the tuple becomes a ghost tuple and is allocated at the ghost location $\underline{\ell}$. Whenever a process located at ℓ performs an action or terminates, removal of the ghost tuple that could have been allocated at $\underline{\ell}$ takes place. In particular, we let $\ell :: nil$ perform the action $stop@\ell$ (rule OS2), and, in the presence of a $stop@\ell$ action, we require the removal of ghost tuples at $\underline{\ell}$ (rule OS4). Rule OS5 deals with addition of tuples to the tuple space (located at τ). Rule OS6 says that a `rd` operation can also access ghost tuples that are not allocated at the location of the process that performs the operation. Rule OS7 says that an `in` operation can access just tuples stored in the tuple space (i.e., it cannot access ghost tuples). Location removal is actually performed into two steps: first, a location restriction is put and, then, when applying rule OS8, the removal actually takes place by means of the structural relation. The information stored in the transition labels always refer the location of the process that performs the operation, apart for rule OS6 that, whenever a ghost tuple is accessed, stores the location of such a tuple.

The structural relation enjoys the following property (that will be used later).

Proposition 1. For all $P \in \mathcal{P}_2$ and $\ell, \ell' \in Loc$, $P \setminus \underline{\ell} \setminus \underline{\ell'} \equiv P \setminus \underline{\ell'} \setminus \underline{\ell}$.

Proof. By easy induction on the structure of P and transitivity.

Structural Rules

$$P \parallel nil \equiv P \qquad P \parallel Q \equiv Q \parallel P$$

$$P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \qquad (P \parallel Q) \setminus \underline{\ell} \equiv P \setminus \underline{\ell} \parallel Q \setminus \underline{\ell}$$

$$(\underline{\ell} :: P) \setminus \underline{\ell} \equiv nil \qquad (\lambda :: P) \setminus \underline{\ell} \equiv \lambda :: P \quad \text{if } \lambda \neq \underline{\ell}$$

Operational Rules

OS1 $$\frac{P \xrightarrow{a} P'}{\ell :: P \xrightarrow{a@\ell} \ell :: P'}$$

OS2 $$\ell :: nil \xrightarrow{stop@\ell} nil$$

OS3 $$\frac{P \xrightarrow{\alpha@\ell} P'}{P \parallel Q \xrightarrow{\alpha@\ell} P' \parallel Q}$$

OS4 $$\frac{P \xrightarrow{stop@\ell} P'}{P \xrightarrow{stop@\ell} P' \setminus \underline{\ell}}$$

OS5 $$\frac{P \xrightarrow{out(e)@\ell} P' \quad \wedge \quad [\![e]\!] = v}{P \xrightarrow{out(v)@\ell} P' \setminus \underline{\ell} \parallel \tau :: \underline{out}(v)}$$

OS6 $$\frac{P \xrightarrow{rd(t)@\ell} P' \quad \wedge \quad ([\![t]\!] = v \vee t \in VAR) \quad \wedge \quad \lambda \neq \underline{\ell}}{P \parallel \lambda :: \underline{out}(v) \xrightarrow{rd(v)@\lambda'} P'[v/t] \setminus \underline{\ell} \parallel \lambda :: \underline{out}(v)} \quad \text{where } \lambda' = \begin{cases} \ell & \text{if } \lambda = \tau \\ \lambda & \text{otherwise} \end{cases}$$

OS7 $$\frac{P \xrightarrow{in(t)@\ell} P' \quad \wedge \quad ([\![t]\!] = v \vee t \in VAR)}{P \parallel \tau :: \underline{out}(v) \xrightarrow{in(v)@\ell} P'[v/t] \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)}$$

OS8 $$\frac{P \equiv Q \quad \wedge \quad Q \xrightarrow{\alpha@\lambda} Q' \quad \wedge \quad Q' \equiv P'}{P \xrightarrow{\alpha@\lambda} P'}$$

Table 4. Optimized Linda Operational Semantics

4 Proof of correctness

In this section, we prove the equivalence of the two semantics.

The two main results can be informally stated as follows:

- each computation from a distributed version of a process P allowed by the optimized semantics can be simulated by a computation from P within the original semantics (Theorem 1);
- each computation from a process P allowed by the original semantics can be simulated by a computation from a distributed version of P within the optimized semantics (Theorem 2).

First, it is convenient to fix the allocation function used to distribute the parallel components of processes. To this aim, we assume that $\{l, r\}^* \subseteq Loc$; we shall use ρ to range over $\{l, r\}^*$. Hence, strings of the form llr and $rllrl$ are valid locations. Now, we define an allocation function that, intuitively, for any process $P \in \mathcal{P}_1$ returns its "maximal" distribution: each parallel component is allocated over a different location. Locations of the form $\{l, r\}^*$ are easily "duplicated": given a ρ, ρl and ρr are two new different locations.

Definition 1. The *allocation function* $\mathcal{L}_\rho : \mathcal{P}_1 \longrightarrow \mathcal{P}_2$ is defined as follows:

$$\begin{array}{rlrl}
\mathcal{L}_\rho(nil) & \stackrel{\text{def}}{=} \rho :: nil & \mathcal{L}_\rho(a.P) & \stackrel{\text{def}}{=} \rho :: a.P \\
\mathcal{L}_\rho(P \mid Q) & \stackrel{\text{def}}{=} \rho :: (P \mid Q) & \mathcal{L}_\rho(P \parallel Q) & \stackrel{\text{def}}{=} \mathcal{L}_{\rho l}(P) \parallel \mathcal{L}_{\rho r}(Q) \\
\mathcal{L}_\rho(P \parallel O) & \stackrel{\text{def}}{=} \mathcal{L}_\rho(P) \parallel \mathcal{T}(O) & \mathcal{L}_\rho(O \parallel P) & \stackrel{\text{def}}{=} \mathcal{T}(O) \parallel \mathcal{L}_\rho(P) \\
\mathcal{T}(O_1 \parallel O_2) & \stackrel{\text{def}}{=} \mathcal{T}(O_1) \parallel \mathcal{T}(O_2) & \mathcal{T}(\underline{out}(v)) & \stackrel{\text{def}}{=} \tau :: \underline{out}(v)
\end{array}$$

where function $\mathcal{T}$ separately allocates all evaluated tuples at location τ.

Function $\mathcal{L}_\rho$ relates the states of $\mathcal{P}_1$ to those of $\mathcal{P}_2$ and satisfies the following basic property.

Proposition 2. For all $P \in \mathcal{P}_1$ and $\ell \in Loc$, $\mathcal{L}_\rho(P) \setminus \underline{\ell} \equiv \mathcal{L}_\rho(P)$.

Proof. It directly follows from Definition 1 since $\underline{\ell} \notin \Lambda(\mathcal{L}_\rho(P))$.

Correctness will be sketched in the case function $\mathcal{L}_\rho$ (hence, maximal distribution) is used for allocating processes. The proof would proceed similarly also if a different allocation function was used.

We will also use an "inverse" function $\mathcal{C}$ that relates the states of $\mathcal{P}_2$ to those of $\mathcal{P}_1$.

Definition 2. The *cleaning function* $\mathcal{C} : \mathcal{P}_2 \longrightarrow \mathcal{P}_1$ is defined as follows:

$$\begin{array}{rlrl}
\mathcal{C}(\ell :: P) & \stackrel{\text{def}}{=} P & \mathcal{C}(\tau :: \underline{out}(v)) & \stackrel{\text{def}}{=} \underline{out}(v) \\
\mathcal{C}(\underline{\ell} :: P) & \stackrel{\text{def}}{=} nil & \mathcal{C}(P \mid Q) & \stackrel{\text{def}}{=} P \mid Q \\
\mathcal{C}(P \parallel Q) & \stackrel{\text{def}}{=} \mathcal{C}(P) \parallel \mathcal{C}(Q) & &
\end{array}$$

With abuse of notation, given a label $\alpha@\lambda \in \mathcal{L}_2$ we write $\mathcal{C}(\alpha@\lambda)$ to denote the action part α whenever $\alpha \in \mathcal{L}_1$, and the empty action ϵ otherwise (i.e. whenever $\alpha = \mathit{stop}$). A similar notation shall be used for sequences of labels from $\mathcal{L}_2^*$.

The following property says that $\mathcal{C}$ is the inverse function of $\mathcal{L}_\rho$.

Proposition 3. For all $P \in \mathcal{P}_1$, $\mathcal{C}(\mathcal{L}_\rho(P)) = P$.

Proof. By easy induction on the structure of P.

Given P_o, we shall use $\Lambda(P_o)$ to denote the set of locations occurring in P_0. Formally, function $\Lambda : \mathcal{P}_2 \longrightarrow \mathit{LOC}$ is defined inductively as follows:

$$\Lambda(\mathit{nil}) = \Lambda(a.P) = \Lambda(\underline{\mathit{out}}(v)) = \Lambda(P \mid Q) = \emptyset, \quad \Lambda(P_o \parallel Q_o) = \Lambda(P_o) \cup \Lambda(Q_o),$$
$$\Lambda(\lambda :: P) = \{\lambda\}, \qquad \Lambda(P \setminus \lambda) = \Lambda(P) \setminus \{\lambda\}.$$

As a matter of notation, given P_o, we shall use $P_o[\ell'/\ell]$ to denote the term obtained by substituting each occurrence of ℓ in P_o with ℓ'. Finally, we use the notation $\Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{\mathit{out}}(v_i)$ as a shorthand for $\underline{\ell_1} :: \underline{\mathit{out}}(v_1) \parallel \ldots \parallel \underline{\ell_n} :: \underline{\mathit{out}}(v_n)$ (the order in which the operands $\underline{\ell_i} :: \underline{\mathit{out}}(v_i)$ are arranged is unimportant, as $\parallel$ is associative and commutative in both the two operational semantics considered in the paper); and when $n = 0$, this term will by convention indicate nil.

Let us start proving that the original semantics can simulate the optimized one. To this aim, let us introduce the following preorder, $\prec$, over $\mathcal{L}_2^*$.

Definition 3. Let $\prec$ the least preorder relation over $\mathcal{L}_2^*$ induced by the two following laws:

TP1 $\sigma' \cdot rd(v)@\tau \cdot in(v)@\ell \cdot \sigma \prec \sigma' \cdot in(v)@\ell \cdot rd(v)@\underline{\ell} \cdot \sigma$

TP2 $\sigma' \cdot \alpha@\lambda \cdot in(v)@\ell \cdot \sigma \prec \sigma' \cdot in(v)@\ell \cdot \alpha@\lambda \cdot \sigma$ if $\lambda \neq \ell, \underline{\ell}$

The intuition behind the preorder $\prec$ is that if $P \xrightarrow{\sigma} Q$ and $\sigma' \prec \sigma$ then it also holds that $P \xrightarrow{\sigma'} Q$. Law TP1 permits exchanging the execution order of two operations accessing the same evaluated tuple in order to avoid accessing ghost tuples. Law TP2 permits exchanging the execution order of operations that are not causally related (its simple presentation relies on the observation that there cannot be two ghost tuples at the same location, hence if $\mathcal{L}_\rho(P) \xrightarrow{\sigma} \xrightarrow{in(v)@\ell} \xrightarrow{\alpha@\underline{\ell}} Q$ then it should be $\alpha = rd(v)$ and we would fall in the case dealt with by law TP1).

Let us now introduce some useful notations. We shall write $a \notin s$ to denote that there are not s_1, s_2 such that $s = s_1 a s_2$ ($\alpha@\lambda \notin \sigma$ has a similar meaning). Moreover, we write $g(\sigma)$ to denote the number of occurrences in σ of locations of $\underline{\mathit{Loc}}$ ('g' stands for 'ghost'). Intuitively, sequences of labels $\sigma \in \mathcal{L}_2^*$ such that $g(\sigma) > 0$ are singled out from sequences of operations that make use of ghost tuples and, hence, cannot be mimicked in the original semantics. However, we will show that for each σ with $g(\sigma) > 0$ it is possible to find a σ' such that $g(\sigma') = 0$, hence σ' is singled out from a sequence of operations that can also be performed according to the original semantics, and $\sigma' \prec \sigma$, hence σ' simulates σ according to the optimized semantics.

The following basic property relates $\underline{a@\ell}$-labelled transitions to $\underline{a}$-labelled transitions and can be considered the inverse of Proposition 13.

Proposition 4. For all $P, Q \in \mathcal{P}_2$, t template and $\ell \in Loc$, $P \xrightarrow{\underline{a@\ell}} Q$ where $a \in \{in(t), rd(t), out(e)\}$ implies that there are $P', Q' \in \mathcal{P}_1$ such that $P' \xrightarrow{\underline{a}} Q'$. Moreover, either $P = \ell :: P'$ and $Q = \ell :: Q'$, or there is $R \in \mathcal{P}_2$ such that $P = (\ell :: P') \parallel R$, $Q = (\ell :: Q') \parallel R$ and $t \notin fv(R)$.

Proof. By induction on the length of the proof of transition $P \xrightarrow{\underline{a@\ell}} Q$.

Basic step. The proof has length 1. Then rule OS1 has been used and there are $P', Q' \in \mathcal{P}_1$ such that $P = \ell :: P'$, $Q = \ell :: Q'$ and $P' \xrightarrow{\underline{a}} Q'$.

Inductive step. The proof has length $n > 1$, hence the last applied rule is OS3. This means that there are $P_1, Q_1, R_1 \in \mathcal{P}_2$ such that $P = P_1 \parallel R_1$, $Q = Q_1 \parallel R_1$ and $P_1 \xrightarrow{\underline{a@\ell}} Q_1$. Since P is closed then $t \notin fv(R_1)$. By inductive hypothesis, we have that there are $P', Q' \in \mathcal{P}_1$ such that $P' \xrightarrow{\underline{a}} Q'$ and either $P_1 \equiv \ell :: P'$ and $Q_1 \equiv \ell :: Q'$, or there is $R' \in \mathcal{P}_2$ such that $P_1 \equiv (\ell :: P') \parallel R'$, $Q_1 \equiv (\ell :: Q') \parallel R'$ and $t \notin fv(R')$. The thesis follows by taking $R = R_1$ in the former case and $R = R' \parallel R_1$ in the latter.

The following property is similar to the previous one but takes into account $\underline{stop@\ell}$-labelled transitions.

Proposition 5. For all $P, Q \in \mathcal{P}_2$ and $\ell \in Loc$, $P \xrightarrow{\underline{stop@\ell}} Q$ implies that either $P = \ell :: nil$ and $Q = nil$, or there is $R \in \mathcal{P}_2$ such that $P = (\ell :: nil) \parallel R$ and $Q = nil \parallel R$.

Proof. By induction on the length of the proof of transition $P \xrightarrow{\underline{stop@\ell}} Q$.

Basic step. The proof has length 1. Then rule OS2 has been applied and the thesis obviously follows.

Inductive step. The proof has length $n > 1$, hence the last applied rule is OS3. This means that there are $P_1, Q_1, R_1 \in \mathcal{P}_2$ such that $P = P_1 \parallel R_1$, $Q = Q_1 \parallel R_1$ and $P_1 \xrightarrow{\underline{stop@\ell}} Q_1$. By inductive hypothesis, we have that either $P_1 \equiv \ell :: nil$ and $Q_1 = nil$, or there is $R_2 \in \mathcal{P}_2$ such that $P_1 \equiv (\ell :: nil) \parallel R_2$, $Q_1 \equiv nil \parallel R_2$. Then, the thesis follows by taking $R = R_1$ in the former case and $R = R_1 \parallel R_2$ in the latter.

Now, we can prove soundness of laws TP1 and TP2.

Proposition 6. For all $P \in \mathcal{P}_1$, $Q_1, Q_2, R \in \mathcal{P}_2$, $\rho \in \{l, r\}^*$, $\sigma \in \mathcal{L}_2^*$, $v \in VAL$ and $\ell \in Loc$, $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q_1 \xrightarrow{\underline{in(v)@\ell}} Q_2 \xrightarrow{\underline{rd(v)@\ell}} R$ implies that there are $Q_3 \in \mathcal{P}_2$ and $\ell' \in Loc$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q_1 \xrightarrow{\underline{rd(v)@\ell'}} Q_3 \xrightarrow{\underline{in(v)@\ell}} R$.

Proof. Transition $Q_1 \xrightarrow{in(v)@\ell} Q_2$ must have been deduced by applying rule OS7 (and, possibly, OS8). Hence, there are $P_1, P_2 \in \mathcal{P}_2$ and a template t such that $Q_1 \equiv P_1 \parallel \tau :: \underline{out}(v)$, $P_1 \xrightarrow{in(t)@\ell} P_2$, $Q_2 \equiv P_2[v/t] \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$, and $[\![t]\!] = v$ or $t \in VAR$. By Proposition 4, $P_1 \xrightarrow{in(t)@\ell} P_2$ implies that there are $P', P'_1 \in \mathcal{P}_1$ and $P''_1 \in \mathcal{P}_2$ such that $P' \xrightarrow{in(t)} P'_1$, $P_1 \equiv (\ell :: P') \parallel P''_1$, $P_2 \equiv (\ell :: P'_1) \parallel P''_1$ and $t \notin fv(P''_1)$. In the same way, transition $Q_2 \xrightarrow{rd(v)@\underline{\ell}} R$ must have been deduced by applying OS6 with $\lambda = \underline{\ell}$ (and, possibly, OS8). Hence, there are $R', R'' \in \mathcal{P}_2$, a template t' and $\ell' \in Loc$ with $\ell' \neq \ell$ (because $\lambda = \underline{\ell}$ and the premisis of rule OS6 requires that $\lambda \neq \underline{\ell'}$) such that $P_2[v/t]\backslash\underline{\ell} \equiv R'$, $R' \xrightarrow{rd(t')@\ell'} R''$, $R \equiv R''[v/t'] \setminus \underline{\ell'} \parallel \underline{l} :: \underline{out}(v)$, and $[\![t']\!] = v$ or $t' \in VAR$. By Proposition 4, $R' \xrightarrow{rd(t')@\ell'} R''$ implies that there are $P'', P'_2 \in \mathcal{P}_1$ and $P''_2 \in \mathcal{P}_2$ such that $P'' \xrightarrow{rd(t')} P'_2$, $P_2[v/t] \setminus \underline{\ell} \equiv R' \equiv (\ell' :: P'') \parallel P''_2$, $R'' \equiv (\ell' :: P'_2) \parallel P''_2$ and $t' \notin fv(P''_2)$. Now, by transitivity, from $P_2[v/t] \setminus \underline{\ell} \equiv ((\ell :: P'_1) \parallel P''_1)[v/t] \setminus \underline{\ell} \equiv \ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell}$ and $P_2[v/t] \setminus \underline{\ell} \equiv (\ell' :: P'') \parallel P''_2$ it follows that $\ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell} \equiv (\ell' :: P'') \parallel P''_2$ from which we get that there is $P_3 \in \mathcal{P}_2$ such that $P''_1 \equiv (\ell' :: P'') \parallel P_3$ and $P''_2 \equiv \ell :: P'_1[v/t] \parallel (P_3 \setminus \underline{\ell})$. Therefore, we have that $Q_1 \equiv P_1 \parallel \tau :: \underline{out}(v) \equiv \ell :: P' \parallel P''_1 \parallel \tau :: \underline{out}(v) \equiv \ell :: P' \parallel \ell' :: P'' \parallel P_3 \parallel \tau :: \underline{out}(v) \equiv \ell' :: P'' \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v)$ and that $R \equiv R'[v/t'] \setminus \underline{\ell'} \parallel \underline{\ell} :: \underline{out}(v) \equiv (\ell' :: P'_2 \parallel P''_2)[v/t'] \setminus \underline{\ell'} \parallel \underline{\ell} :: \underline{out}(v) \equiv \ell' :: P'_2[v/t'] \parallel P''_2 \setminus \underline{\ell'} \parallel \underline{\ell} :: \underline{out}(v) \equiv \ell' :: P'_2[v/t'] \parallel (\ell :: P'_1[v/t] \parallel P_3 \setminus \underline{\ell}) \setminus \underline{\ell'} \parallel \underline{\ell} :: \underline{out}(v) \equiv \ell' :: P'_2[v/t'] \parallel \ell :: P'_1[v/t] \parallel P_3 \setminus \underline{\ell} \setminus \underline{\ell'} \parallel \underline{\ell} :: \underline{out}(v) \equiv \ell :: P'_1[v/t] \parallel \ell' :: P'_2[v/t'] \parallel P_3 \setminus \underline{\ell'} \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$ (where we have also used Proposition 1). Now, take $Q_3 = \ell :: P' \parallel \ell' :: P'_2[v/t'] \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v)$. Since $P'' \xrightarrow{rd(t')} P'_2$, by applying rules OS1 and OS3, we get $\ell' :: P'' \parallel \ell :: P' \parallel P_3 \xrightarrow{rd(t')@\ell'} \ell' :: P'_2 \parallel \ell :: P' \parallel P_3$, and, by applying rules OS6 and OS8, we get $Q_1 \xrightarrow{rd(v)@\ell'} Q_3$. Moreover, since $P' \xrightarrow{in(t)} P'_1$, by applying rules OS1 and OS3, we get $\ell :: P' \parallel \ell' :: P'_2[v/t'] \parallel P_3 \setminus \underline{\ell'} \xrightarrow{in(t)@\ell} \ell :: P'_1 \parallel \ell' :: P'_2[v/t'] \parallel P_3 \setminus \underline{\ell'}$, and, by applying rules OS7 and OS8, we get $Q_3 \xrightarrow{in(v)@\ell} R$, that concludes the proof.

Proposition 7. For all $P \in \mathcal{P}_1$, $Q_1, Q_2, R \in \mathcal{P}_2$, $\rho \in \{l, r\}^*$, $\sigma \in \mathcal{L}_2^*$, $v \in VAL$, $\ell \in Loc$, $\lambda \in LOC$ and $\alpha@\lambda \in \mathcal{L}_2$, $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q_1 \xrightarrow{in(v)@\ell} Q_2 \xrightarrow{\alpha@\lambda} R$ and $\lambda \neq \ell, \underline{\ell}$ imply that there is $Q_3 \in \mathcal{P}_2$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q_1 \xrightarrow{\alpha@\lambda} Q_3 \xrightarrow{in(v)@\ell} R$.

Proof. Transition $Q_1 \xrightarrow{in(v)@\ell} Q_2$ must have been deduced by applying rule OS7 (and, possibly, OS8). Hence, there are $P_1, P_2 \in \mathcal{P}_2$ and a template t such that $Q_1 \equiv P_1 \parallel \tau :: \underline{out}(v)$, $P_1 \xrightarrow{in(t)@\ell} P_2$, $Q_2 \equiv P_2[v/t] \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$,

and $[\![t]\!] = v$ or $t \in VAR$. By Proposition 4, $P_1 \xrightarrow{in(t)@\ell} P_2$ implies that there are $P', P'_1 \in \mathcal{P}_1$ and $P''_1 \in \mathcal{P}_2$ such that $P' \xrightarrow{in(t)} P'_1$, $P_1 \equiv (\ell :: P') \parallel P''_1$, $P_2 \equiv (\ell :: P'_1) \parallel P''_1$ and $t \notin fv(P''_1)$. Now, we have that $Q_1 \equiv P_1 \parallel \tau :: \underline{out}(v) \equiv \ell :: P' \parallel P''_1 \parallel \tau :: \underline{out}(v)$ and, since $t \notin fv(P''_1)$, that $Q_2 \equiv P_2[v/t] \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \equiv ((\ell :: P'_1) \parallel P''_1)[v/t] \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \equiv \ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$ and we can proceed by case analysis on action $\alpha@\lambda$.

$\alpha@\lambda = stop@\ell'$, for some $\ell' \in Loc$. Then rule OS4 (and, possibly, OS8) has been used to derive transition $Q_2 \xrightarrow{\alpha@\lambda} R$. Hence, there are $R', R'' \in \mathcal{P}_2$ such that $R' \xrightarrow{stop@\ell'} R''$, $Q_2 \equiv R'$ and $R \equiv R'' \setminus \underline{\ell}$. By Proposition 5, $R' \xrightarrow{stop@\ell'} R''$ implies that there is $R_1 \in \mathcal{P}_2$ such that $R' \equiv (\ell' :: nil) \parallel R_1$ and $R'' \equiv nil \parallel R_1$. Now, by transitivity, from $Q_2 \equiv \ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$ and $Q_2 \equiv R' \equiv (\ell' :: nil) \parallel R_1$ it follows that $\ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \equiv (\ell' :: nil) \parallel R_1$ from which we get that there is $P_3 \in \mathcal{P}_2$ such that $P''_1 \equiv (\ell' :: nil) \parallel P_3$ and $R_1 \equiv \ell :: P'_1[v/t] \parallel (P_3 \setminus \underline{\ell}) \parallel \underline{\ell} :: \underline{out}(v)$. Therefore, we have that $Q_1 \equiv \ell :: P' \parallel P''_1 \parallel \tau :: \underline{out}(v) \equiv \ell :: P' \parallel \ell' :: nil \parallel P_3 \parallel \tau :: \underline{out}(v) \equiv \ell' :: nil \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v)$ and that $R \equiv R'' \setminus \underline{\ell'} \equiv (nil \parallel R_1) \setminus \underline{\ell'} \equiv (\ell :: P'_1[v/t] \parallel P_3 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)) \setminus \underline{\ell'} \equiv \ell :: P'_1[v/t] \parallel P_3 \setminus \underline{\ell} \setminus \underline{\ell'} \parallel \underline{\ell} :: \underline{out}(v) \equiv \ell :: P'_1[v/t] \parallel P_3 \setminus \underline{\ell'} \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$ (where we have also used Proposition 1). Now, take $Q_3 = \ell :: P' \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v)$. Since, by rule OS2, $\ell' :: nil \xrightarrow{stop@\ell'} nil$, then, by applying rule OS3, we get $\ell' :: nil \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v) \xrightarrow{stop@\ell'} nil \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v)$ and, by applying rules OS4 and OS8, we get $Q_1 \xrightarrow{stop@\ell'} Q_3$. Moreover, since $P' \xrightarrow{in(t)} P'_1$, by applying rules OS1 and OS3, we get $\ell :: P' \parallel P_3 \setminus \underline{\ell'} \xrightarrow{in(t)@\ell} \ell :: P'_1 \parallel P_3 \setminus \underline{\ell'}$ and, by applying rules OS7 and OS8, we get $Q_3 \xrightarrow{in(v)@\ell} R$ (recall that $\ell \neq \ell'$ and that $t \notin fv(P_3)$ because $t \notin fv(P''_1)$), that concludes the proof for this case.

$\alpha@\lambda = out(v')@\ell'$, for some $\ell' \in Loc$ and $v' \in VAL$. Then rule OS5 (and, possibly, OS8) has been used to derive transition $Q_2 \xrightarrow{\alpha@\lambda} R$. Hence, there are $R', R'' \in \mathcal{P}_2$ and $e' \in EXP$ such that $R' \xrightarrow{out(e')@\ell'} R''$, $Q_2 \equiv R'$, $R \equiv R'' \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v')$ and $[\![e']\!] = v'$. By Proposition 4, $R' \xrightarrow{out(e')@\ell'} R''$ implies that there are $P'', P'_2 \in \mathcal{P}_1$ and $P''_2 \in \mathcal{P}_2$ such that $P'' \xrightarrow{out(e')} P'_2$, $R' \equiv (\ell' :: P'') \parallel P''_2$, $R'' \equiv (\ell' :: P'_2) \parallel P''_2$ and $e' \notin fv(P''_2)$. Now, by transitivity, from $Q_2 \equiv \ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$ and $Q_2 \equiv R' \equiv (\ell' :: P'') \parallel P''_2$ it follows that $\ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \equiv (\ell' :: P'') \parallel P''_2$ from which we get that there is $P_3 \in \mathcal{P}_2$ such that $P''_1 \equiv (\ell' :: P'') \parallel P_3$ and $P''_2 \equiv \ell :: P'_1[v/t] \parallel (P_3 \setminus \underline{\ell}) \parallel \underline{\ell} :: \underline{out}(v)$. Therefore, we have that $Q_1 \equiv \ell :: P' \parallel P''_1 \parallel \tau :: \underline{out}(v) \equiv \ell :: P' \parallel \ell' :: P'' \parallel P_3 \parallel \tau :: \underline{out}(v) \equiv \ell' :: P'' \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v)$ and that $R \equiv R' \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \equiv (\ell' :: P'_2 \parallel P''_2) \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \equiv \ell' :: P'_2 \parallel (\ell :: P'_1[v/t] \parallel P_3 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)) \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \equiv \ell' :: P'_2 \parallel \ell :: P'_1[v/t] \parallel$

$P_3 \setminus \underline{\ell'} \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \parallel \tau :: \underline{out}(v')$ (where we have also used Proposition 1). Now, take $Q_3 = \ell :: P' \parallel \ell' :: P'_2 \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \parallel \tau :: \underline{out}(v)$. Since $P'' \xrightarrow{out(e')} P'_2$, by applying rules OS1 and OS3, we get $\ell' :: P'' \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v) \xrightarrow{out(e')@\ell'} \ell' :: P'_2 \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v)$ and, by applying rules OS5 and OS8, we get $Q_1 \xrightarrow{out(v')@\ell'} Q_3$. Moreover, since $P' \xrightarrow{in(t)} P'_1$, by applying rules OS1 and OS3, we get $\ell :: P' \parallel \ell' :: P'_2 \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \xrightarrow{in(t)@\ell} \ell :: P'_1 \parallel \ell' :: P'_2 \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v')$ and, by applying rules OS7 and OS8, we get $Q_3 \xrightarrow{in(v)@\ell} R$ (recall that $\ell \neq \ell'$ and that $t \notin fv(P_3)$ because $t \notin fv(P''_2)$), that concludes the proof for this case.

$\alpha@\lambda = rd(v')@\ell'$, for some $\ell' \in Loc$ and $v' \in VAL$. Then rule OS6 (and, possibly, OS8) has been used to derive transition $Q_2 \xrightarrow{\alpha@\lambda} R$. Hence, there are $R', R'' \in \mathcal{P}_2$ and a template t' such that $R' \xrightarrow{rd(t')@\ell'} R''$, $Q_2 \equiv R' \parallel \tau :: \underline{out}(v')$, $R \equiv R''[v'/t'] \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v')$ and $[\![t']\!] = v'$ or $t' \in VAR$. By Proposition 4, $R' \xrightarrow{rd(t')@\ell'} R''$ implies that there are $P'', P'_2 \in \mathcal{P}_1$ and $P''_2 \in \mathcal{P}_2$ such that $P'' \xrightarrow{rd(t')} P'_2$, $R' \equiv (\ell' :: P'') \parallel P''_2$, $R'' \equiv (\ell' :: P'_2) \parallel P''_2$ and $t' \notin fv(P''_2)$. Now, by transitivity, from $Q_2 \equiv \ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$ and $Q_2 \equiv R' \parallel \tau :: \underline{out}(v') \equiv (\ell' :: P'') \parallel P''_2 \parallel \tau :: \underline{out}(v')$ it follows that $\ell :: P'_1[v/t] \parallel P''_1 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \equiv (\ell' :: P'') \parallel P''_2 \parallel \tau :: \underline{out}(v')$ from which we get that there is $P_3 \in \mathcal{P}_2$ such that $P''_1 \equiv (\ell' :: P'') \parallel P_3 \parallel \tau :: \underline{out}(v')$ and $P''_2 \equiv \ell :: P'_1[v/t] \parallel (P_3 \setminus \underline{\ell}) \parallel \underline{\ell} :: \underline{out}(v)$. Therefore, we have that $Q_1 \equiv \ell :: P' \parallel P''_1 \parallel \tau :: \underline{out}(v) \equiv \ell :: P' \parallel \ell' :: P'' \parallel P_3 \parallel \tau :: \underline{out}(v') \parallel \tau :: \underline{out}(v)$ and, since $t' \notin fv(P''_2)$, that $R \equiv R''[v'/t'] \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \equiv ((\ell' :: P'_2) \parallel P''_2)[v'/t'] \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \equiv \ell' :: P'_2[v'/t'] \parallel (P''_2 \setminus \underline{\ell'}) \parallel \tau :: \underline{out}(v') \equiv \ell' :: P'_2[v'/t'] \parallel (\ell :: P'_1[v/t] \parallel P_3 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)) \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \equiv \ell' :: P'_2[v'/t'] \parallel \ell :: P'_1[v/t] \parallel P_3 \setminus \underline{\ell'} \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \parallel \tau :: \underline{out}(v')$ (where we have also used Proposition 1). Now, take $Q_3 = \ell' :: P'_2[v'/t'] \parallel \ell :: P' \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v) \parallel \tau :: \underline{out}(v')$. Since $P'' \xrightarrow{rd(t')} P'_2$, by applying rules OS1 and OS3, we get $\ell' :: P'' \parallel \ell :: P' \parallel P_3 \parallel \underline{out}(v) \xrightarrow{rd(t')@\ell'} \ell' :: P'_2 \parallel \ell :: P' \parallel P_3 \parallel \underline{out}(v)$ and, by applying rules OS6 and OS8, we get $Q_1 \xrightarrow{rd(v')@\ell'} Q_3$. Moreover, since $P' \xrightarrow{in(t)} P'_1$, by applying rules OS1 and OS3, we get $\ell :: P' \parallel \ell' :: P'_2[v'/t'] \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v') \xrightarrow{in(t)@\ell} \ell :: P'_1 \parallel \ell' :: P'_2[v'/t'] \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v')$ and, by applying rules OS7 and OS8, we get $Q_3 \xrightarrow{in(v)@\ell} R$ (recall that $\ell \neq \ell'$ and that $t \notin fv(P_3)$ because $t \notin fv(P''_2)$), that concludes the proof for this case.

$\alpha@\lambda = in(v')@\ell'$, for some $\ell' \in Loc$ and $v' \in VAL$. Then rule OS7 (and, possibly, OS8) has been used to derive transition $Q_2 \xrightarrow{\alpha@\lambda} R$. Hence, there are $R', R'' \in \mathcal{P}_2$ and $t' \in EXP$ such that $R' \xrightarrow{in(t')@\ell'} R''$, $Q_2 \equiv R' \parallel \tau :: \underline{out}(v')$, $R \equiv$

$R''[v'/t'] \setminus \underline{\ell'} \parallel \underline{\ell'} :: \underline{out}(v')$ and $[\![t']\!] = v'$ or $t' \in VAR$. By Proposition 4, $R' \xrightarrow{in(t')@\ell'} R''$ implies that there are $P'', P_2' \in \mathcal{P}_1$ and $P_2'' \in \mathcal{P}_2$ such that $P'' \xrightarrow{in(t')} P_2'$, $R' \equiv (\ell' :: P'') \parallel P_2''$, $R'' \equiv (\ell' :: P_2') \parallel P_2''$ and $t' \notin fv(P_2'')$. Now, by transitivity, from $Q_2 \equiv \ell :: P_1'[v/t] \parallel P_1'' \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$ and $Q_2 \equiv R' \parallel \tau :: \underline{out}(v') \equiv (\ell' :: P'') \parallel P_2'' \parallel \tau :: \underline{out}(v')$ it follows that $\ell :: P_1'[v/t] \parallel P_1'' \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \equiv (\ell' :: P'') \parallel P_2'' \parallel \tau :: \underline{out}(v')$ from which we get that there is $P_3 \in \mathcal{P}_2$ such that $P_1'' \equiv (\ell' :: P'') \parallel P_3 \parallel \tau :: \underline{out}(v')$ and $P_2'' \equiv \ell :: P_1'[v/t] \parallel (P_3 \setminus \underline{\ell}) \parallel \underline{\ell} :: \underline{out}(v)$. Therefore, we have that $Q_1 \equiv \ell :: P' \parallel P_1'' \parallel \tau :: \underline{out}(v) \equiv \ell :: P' \parallel \ell' :: P'' \parallel P_3 \parallel \tau :: \underline{out}(v') \parallel \tau :: \underline{out}(v)$ and, since $t' \notin fv(P_2'')$, that $R \equiv R''[v'/t'] \setminus \underline{\ell'} \parallel \underline{\ell'} :: \underline{out}(v') \equiv ((\ell' :: P_2') \parallel P_2'')[v'/t'] \setminus \underline{\ell'} \parallel \underline{\ell'} :: \underline{out}(v') \equiv \ell' :: P_2'[v'/t'] \parallel (P_2'' \setminus \underline{\ell'}) \parallel \underline{\ell'} :: \underline{out}(v') \equiv \ell' :: P_2'[v'/t'] \parallel (\ell :: P_1'[v/t] \parallel P_3 \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)) \setminus \underline{\ell'} \parallel \underline{\ell'} :: \underline{out}(v') \equiv \ell' :: P_2'[v'/t'] \parallel \ell :: P_1'[v/t] \parallel P_3 \setminus \underline{\ell'} \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \parallel \underline{\ell'} :: \underline{out}(v')$ (where we have also used Proposition 1). Now, take $Q_3 = \ell' :: P_2'[v'/t'] \parallel \ell :: P' \parallel P_3 \setminus \underline{\ell'} \parallel \tau :: \underline{out}(v) \parallel \underline{\ell'} :: \underline{out}(v')$. Since $P'' \xrightarrow{in(t')} P_2'$, by applying rules OS1 and OS3, we get $\ell' :: P'' \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v) \xrightarrow{in(t')@\ell'} \ell' :: P_2' \parallel \ell :: P' \parallel P_3 \parallel \tau :: \underline{out}(v)$ and, by applying rules OS7 and OS8, we get $Q_1 \xrightarrow{in(v')@\ell'} Q_3$. Moreover, since $P' \xrightarrow{in(t)} P_1'$, by applying rules OS1 and OS3, we get $\ell :: P' \parallel \ell' :: P_2'[v'/t'] \parallel P_3 \setminus \underline{\ell'} \parallel \underline{\ell'} :: \underline{out}(v') \xrightarrow{in(t)@\ell} \ell :: P_1' \parallel \ell' :: P_2'[v'/t'] \parallel P_3 \setminus \underline{\ell'} \parallel \underline{\ell'} :: \underline{out}(v')$ and, by applying rules OS7 and OS8, we get $Q_3 \xrightarrow{in(v)@\ell} R$ (recall that $\ell \neq \ell'$ and that $t \notin fv(P_3)$ because $t \notin fv(P_2'')$), that concludes the proof for this case.

The following proposition shows how the laws of the trace preoder can be used to reduce the number of ghost tuples accessed during a computation.

Proposition 8. For all $P \in \mathcal{P}_1$, $Q, R \in \mathcal{P}_2$, $\rho \in \{l, r\}^*$, $\sigma \in \mathcal{L}_2^+$, $v \in VAL$ and $\ell \in Loc$, $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q \xrightarrow{rd(v)@\underline{\ell}} R$ implies that there are $\sigma_1, \sigma_2 \in \mathcal{L}_2^*$ such that $\sigma = \sigma_1 \cdot in(v)@\ell \cdot \sigma_2$. Moreover, if $in(v)@\ell \notin \sigma_2$ and for all $\alpha@\lambda \in \sigma_2$ we have that $\lambda \neq \ell, \underline{\ell}$, then there is $\ell' \in Loc$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma_1} \xrightarrow{\sigma_2} \xrightarrow{rd(v)@\ell'} \xrightarrow{in(v)@\ell} R$.

Proof. Transition $Q \xrightarrow{rd(v)@\underline{\ell}} R$ must have been deduced by applying OS6 (and, possibly, OS8) with $\lambda = \underline{\ell}$. In particular, this means that there is $Q' \in \mathcal{P}_2$ such that $Q \equiv Q' \parallel \underline{\ell} :: \underline{out}(v)$. Since, by definition, $\underline{\ell} \notin \Lambda(\mathcal{L}_\rho(P))$ then rule OS7 must have been applied to give rise to the ghost tuple $\underline{\ell} :: \underline{out}(v)$. Therefore, label $in(v)@\ell$ occurs in the sequence σ, i.e. $in(v)@\ell \in \sigma$. Now, let $\sigma_1, \sigma_2 \in \mathcal{L}_2^*$ be such that $\sigma = \sigma_1 \cdot in(v)@\ell \cdot \sigma_2$ and $in(v)@\ell \notin \sigma_2$. By hypothesis, $\mathcal{L}_\rho(P) \xrightarrow{\sigma_1} \xrightarrow{in(v)@\ell} \xrightarrow{\sigma_2} \xrightarrow{rd(v)@\underline{\ell}} R$. Hence, if $\sigma_2 = \epsilon$ then the thesis directly follows from Proposition 6. Otherwise, by repeatedly applying Proposition 7, we get that $\mathcal{L}_\rho(P) \xrightarrow{\sigma_1} \xrightarrow{\sigma_2} \xrightarrow{in(v)@\ell} \xrightarrow{rd(v)@\underline{\ell}} R$ and, again, the thesis follows from Proposition 6.

We can now give a method for transforming a generic computation in an equivalent computation (i.e. with the same final state) that corresponds to a sequence of operations that never access ghost tuples.

Proposition 9. For all $P \in \mathcal{P}_1$, $Q \in \mathcal{P}_2$, $\rho \in \{l,r\}^*$ and $\sigma \in \mathcal{L}_2^*$, $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q$ implies that there is $\sigma' \prec \sigma$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma'} Q$ and $g(\sigma') = 0$.

Proof. By induction on $g(\sigma)$. If $g(\sigma) = 0$ then we can take $\sigma' = \sigma$. Otherwise, let $\sigma_1, \sigma_2, \sigma_3, \sigma_4 \in \mathcal{L}_2^*$ be such that $\sigma = \sigma_1 \cdot rd(v)@\underline{\ell} \cdot \sigma_2$, $\sigma_1 = \sigma_3 \cdot in(v)@\ell \cdot \sigma_4$, $in(v)@\ell \notin \sigma_4$ and $\lambda \neq \ell, \underline{\ell}$, for all $\alpha@\lambda \in \sigma_4$. The sequence σ_4 does exist because $g(\sigma) > 0$ and $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q$ imply the following facts: for some $\ell \in Loc$ and $v \in VAL$, $rd(v)@\underline{\ell} \in \sigma$ (because $g(\sigma) > 0$); hence, $in(v)@\ell \in \sigma$ and there is some occurrence of label $in(v)@\ell$ that is on the left of $rd(v)@\underline{\ell}$ within σ (because label $in(v)@\ell$ is singled out when it is produced the ghost tuple $\underline{\ell} :: \underline{out}(v)$ that is accessed when label $rd(v)@\underline{\ell}$ is singled out); finally, the occurrences of labels $in(v)@\ell$ and $rd(v)@\underline{\ell}$ can be chosen in such a way that $rd(v)@\underline{\ell}$ is the first label after $in(v)@\ell$ that accesses the ghost tuple at ℓ and no label with ℓ as location occurs in between (because, after $in(v)@\ell$, $\underline{out}(v)$ is the only ghost tuple at ℓ untill it will be destroyed by the execution of any operation at ℓ). Now, by Proposition 8, there is $\ell' \in Loc$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma_3} \xrightarrow{\sigma_4} \xrightarrow{rd(v)@\ell'} \xrightarrow{in(v)@\ell} R$, where R is such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma_1} \xrightarrow{rd(v)@\underline{\ell}} R$. Hence, by letting $\sigma_5 = \sigma_3 \cdot \sigma_4 \cdot rd(v)@\ell' \cdot in(v)@\ell \cdot \sigma_2$, we have that $\mathcal{L}_\rho(P) \xrightarrow{\sigma_5} Q$. The thesis follows by induction since $\sigma_5 \prec \sigma$ and $g(\sigma_5) = g(\sigma) - 1$.

The following two propositions relate the transitions of the optimized semantics to the transitions of the original one.

Proposition 10. For all $P \in \mathcal{P}_1$, $Q \in \mathcal{P}_2$, $\rho \in \{l,r\}^*$, $a \in \mathcal{L}_1$, $\ell \in \Lambda(\mathcal{L}_\rho(P)) \cap Loc$, $L \subseteq \Lambda(\mathcal{L}_\rho(P))$ and $v_i \in VAL$ $(\forall i : \ell_i \in L)$, $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{a@\ell} Q$ implies that there are $R \in \mathcal{P}_1$, $\rho' \in \{l,r\}^*$ and $L' \subseteq \Lambda(\mathcal{L}_{\rho'}(R))$ such that $P \xrightarrow{a} R$ and $Q \equiv \mathcal{L}_{\rho'}(R) \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i)$.

Proof. We proceed by case analysis on a.

$a = out(v)$, for some $v \in VAL$. Then, for deriving transition $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{a@\ell} Q$, rule OS5 (and, possibly, rule OS8) has been used, hence there are a template t and $P', Q' \in \mathcal{P}_2$ such that $[\![t]\!] = v$, $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \equiv P'$, $P' \xrightarrow{out(e)@\ell} Q'$ and $Q \equiv Q' \setminus \underline{\ell} \parallel \tau :: \underline{out}(v)$. Moreover, by Proposition 4, $P' \xrightarrow{out(e)@\ell} Q'$ implies that there are $P_1, P_1' \in \mathcal{P}_1$ and $P_2 \in \mathcal{P}_2$ such that $P_1 \xrightarrow{out(e)} P_1'$, $P' \equiv (\ell :: P_1) \parallel P_2$, $Q' \equiv (\ell :: P_1') \parallel P_2$ and $e \notin fv(P_2)$. Now, $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \equiv P' \equiv (\ell :: P_1) \parallel P_2$ implies that there are $P_3, P_4 \in \mathcal{P}_1$ such that $P \equiv P_1 \parallel P_3$, $P_4 \equiv P_1' \parallel P_3$ and $Q' \equiv (\ell :: P_1') \parallel P_2 \equiv \mathcal{L}_\rho(P_4) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. Hence, we have that $Q \equiv Q' \setminus \underline{\ell} \parallel \tau :: \underline{out}(v) \equiv \mathcal{L}_\rho(P_4) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i) \parallel \tau :: \underline{out}(v)$.

Let ρ' be such that $\rho = \rho' \cdot u$ where $u \in \{l, r\}$. We let $L' = L \setminus \{\ell\}$ and $R = P_4 \parallel \underline{out}(v)$, if $u = l$, and $R = \underline{out}(v) \parallel P_4$, if $u = r$. In both cases we have $L' \subseteq \Lambda(\mathcal{L}_{\rho'}(R))$ (namely, $\Lambda(\mathcal{L}_{\rho'}(R)) = \Lambda(\mathcal{L}_\rho(P')) \cup \{\tau\}$) and, since $P_1 \xrightarrow{out(e)} P_1'$, by applying rules S2, S3 and S6, we get $P \xrightarrow{out(v)} R$, and the thesis is proven.

$a = rd(v)$, for some $v \in VAL$. Then, for deriving transition $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{a@\ell} Q$, rule OS6 (and, possibly, rule OS8) has been used, hence there are a template t and $P', Q' \in \mathcal{P}_2$ such that $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \equiv P' \parallel \tau :: \underline{out}(v)$, $P' \xrightarrow{rd(t)@\ell} Q'$, $Q \equiv Q'[v/t] \setminus \underline{\ell} \parallel \tau :: \underline{out}(v)$ and $[\![t]\!] = v$ or $t \in VAR$. Moreover, by Proposition 4, $P' \xrightarrow{rd(t)@\ell} Q'$ implies that there are $P_1, P_1' \in \mathcal{P}_1$ and $P_2 \in \mathcal{P}_2$ such that $P_1 \xrightarrow{rd(t)} P_1'$, $P' \equiv (\ell :: P_1) \parallel P_2$, $Q' \equiv (\ell :: P_1') \parallel P_2$ and $t \notin fv(P_2)$. Now, $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \equiv P' \parallel \tau :: \underline{out}(v) \equiv (\ell :: P_1) \parallel P_2 \parallel \tau :: \underline{out}(v)$ implies that there are $P_3, P_4 \in \mathcal{P}_1$ such that $P \equiv P_1 \parallel P_3 \parallel \underline{out}(v)$, $P_4 \equiv P_1'[v/t] \parallel P_3 \parallel \underline{out}(v)$ and $Q'[v/t] \parallel \tau :: \underline{out}(v) \equiv (\ell :: P_1'[v/t]) \parallel P_2 \parallel \tau :: \underline{out}(v) \equiv \mathcal{L}_\rho(P_4) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. Hence, we have that $Q \equiv Q'[v/t] \setminus \underline{\ell} \parallel \tau :: \underline{out}(v) \equiv \mathcal{L}_\rho(P_4) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$. Now, let $R = P_4$ and $L' = L \setminus \{\ell\}$. Since $P_1 \xrightarrow{rd(t)} P_1'$, by applying rule S2, we get $P_1 \parallel P_3 \xrightarrow{rd(t)} P_1' \parallel P_3$ and, by applying rules S4 (recall that $t \notin fv(P_2)$) and OS8, we get that $P \xrightarrow{rd(v)} R$ and the thesis is proven.

$a = in(v)$, for some $v \in VAL$. Then, for deriving transition $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{a@\ell} Q$, rule OS7 (and, possibly, rule OS8) has been used, hence there are a template t and $P', Q' \in \mathcal{P}_2$ such that $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \equiv P' \parallel \tau :: \underline{out}(v)$, $P' \xrightarrow{in(t)@\ell} Q'$, $Q \equiv Q'[v/t] \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$ and $[\![t]\!] = v$ or $t \in VAR$. Moreover, by Proposition 4, $P' \xrightarrow{in(t)@\ell} Q'$ implies that there are $P_1, P_1' \in \mathcal{P}_1$ and $P_2 \in \mathcal{P}_2$ such that $P_1 \xrightarrow{in(t)} P_1'$, $P' \equiv (\ell :: P_1) \parallel P_2$, $Q' \equiv (\ell :: P_1') \parallel P_2$ and $t \notin fv(P_2)$. Now, $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \equiv P' \parallel \tau :: \underline{out}(v) \equiv (\ell :: P_1) \parallel P_2 \parallel \tau :: \underline{out}(v)$ implies that there are $P_3, P_4 \in \mathcal{P}_1$ such that $P \equiv P_1 \parallel P_3 \parallel \underline{out}(v)$, $P_4 \equiv P_1'[v/t] \parallel P_3$ and $Q'[v/t] \equiv (\ell :: P_1'[v/t]) \parallel P_2 \equiv \mathcal{L}_{\rho l}(P_4) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. Hence, we have that $Q \equiv Q'[v/t] \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v) \equiv \mathcal{L}_{\rho l}(P_4) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i) \parallel \underline{\ell} :: \underline{out}(v)$. Now, let $R = P_4$, $\rho' = \rho l$ and $L' = L$ (with $v_i = v$ for $\ell_i = \ell$). We have that $L' \subseteq \Lambda(\mathcal{L}_{\rho l}(P_4))$ (because $\Lambda(\mathcal{L}_\rho(P)) = \Lambda(\mathcal{L}_{\rho l}(P_4)) \cup \{\tau\}$. Since $P_1 \xrightarrow{in(t)} P_1'$, by applying rule S2, we get $P_1 \parallel P_3 \xrightarrow{in(t)} P_1' \parallel P_3$ and, by applying rules S4 (recall that $t \notin fv(P_2)$) and OS8, we get that $P \xrightarrow{in(v)} R$ and the thesis is proven.

Proposition 11. For all $P \in \mathcal{P}_1$, $Q \in \mathcal{P}_2$, $\rho \in \{l, r\}^*$, $\ell \in \Lambda(\mathcal{L}_\rho(P))$, $L \subseteq \Lambda(\mathcal{L}_\rho(P))$ and $v_i \in VAL$ $(\forall i : \ell_i \in L)$, $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{stop@\ell} Q$

implies that there is $R \in \mathcal{P}_1$ such that $P \equiv R$ and $Q \equiv \mathcal{L}_\rho(R) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$.

Proof. Let us take $R = P$. Obviously, we have $P \equiv R$. We are left to show that $Q \equiv \mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$. This can be easily proved by induction on the length of derivation of transition $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{stop@\ell} Q$. Indeed, either rule OS4 has been the last rule applied, in which case we have $Q = \mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$, or rule OS8 has been the last rule applied, in which case, by induction, we can assume that there is $Q' \in \mathcal{P}_2$ such that $Q' \equiv \mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$ and $Q \equiv Q'$, and the thesis follows by transitivity.

We can now generalize the previous properties to sequences of transitions.

Proposition 12. For all $P \in \mathcal{P}_1$, $Q \in \mathcal{P}_2$, $\rho \in \{l, r\}^*$ and $\sigma \in \mathcal{L}_2^*$, $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q$ and $g(\sigma) = 0$ imply that there are $P' \in \mathcal{P}_1$, $\rho' \in \{l, r\}^*$, $L \subseteq \Lambda(\mathcal{L}_{\rho'}(P'))$ and $v_i \in VAL$ $(\forall i : \ell_i \in L)$ such that $P \xrightarrow{\mathcal{C}(\sigma)} P'$ and $Q \equiv \mathcal{L}_{\rho'}(P') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$.

Proof. By induction on the length of σ.

Basic step. If $\sigma = \epsilon$ then $Q = \mathcal{L}_\rho(P)$ and the thesis easily follows by taking $P' = P$, $\rho' = \rho$ and $L = \emptyset$.

Inductive step. Let $\sigma = \sigma' \cdot \alpha@\ell$ for some $\sigma' \in \mathcal{L}_2^*$, $\alpha \in \mathcal{L}_1 \cup \{stop\}$ and $\ell \in Loc$. Hence, $g(\sigma') = 0$, $g(\alpha@\ell) = 0$ (i.e. $\ell \in Loc$) and there is some $R \in \mathcal{P}_2$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma'} R \xrightarrow{\alpha@\ell} Q$. By induction, there are $R' \in \mathcal{P}_2$, $\rho_1 \in \{l, r\}^*$, $L' \subseteq \Lambda(\mathcal{L}_{\rho_1}(R'))$ and $v_i \in VAL$ $(\forall i : \ell_i \in L)$ such that $P \xrightarrow{\mathcal{C}(\sigma')} R'$ and $R \equiv \mathcal{L}_{\rho_1}(R') \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i)$. By using rule OS8, from $\mathcal{L}_{\rho_1}(P) \xrightarrow{\sigma'} R$, $R \xrightarrow{\alpha@\ell} Q$ and $R \equiv \mathcal{L}_{\rho_1}(R') \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i)$ we get that $\mathcal{L}_{\rho_1}(P) \xrightarrow{\sigma'} \mathcal{L}_\rho(R') \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i)$ and $\mathcal{L}_{\rho_1}(R') \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{\alpha@\ell} Q$. Now the thesis follows from Proposition 10, if $\alpha \in \mathcal{L}_1$, and from Proposition 11 otherwise (i.e. $\alpha = stop$).

Finally, we have that the original semantics can simulate the optimized one.

Theorem 1. For all $P \in \mathcal{P}_1$, $Q \in \mathcal{P}_2$, $\rho \in \{l, r\}^*$ and $\sigma \in \mathcal{L}_2^*$, $\mathcal{L}_\rho(P) \xrightarrow{\sigma} Q$ implies that there are $\sigma' \prec \sigma$ and $P' \in \mathcal{P}_1$ such that $P \xrightarrow{\mathcal{C}(\sigma')} P'$ and $\mathcal{C}(Q) \equiv P'$.

Proof. Directly follows from Propositions 9 and 12.

We now prove that the optimized semantics can simulate the original one (Theorem 2). The following basic property relates $\underline{a}$-labelled transitions to $\underline{a@\ell}$-labelled transitions.

Proposition 13. For all $P, Q \in \mathcal{P}_1$ and $a \in \mathcal{L}_1$, $P \xrightarrow{a} Q$ implies that there is $\ell \in \Lambda(\mathcal{L}_\rho(P))$ such that $\mathcal{L}_\rho(P) \xrightarrow{a@\ell} \mathcal{L}_\rho(Q)$. Moreover, either $\mathcal{L}_\rho(P) = \ell :: P$ and $\mathcal{L}_\rho(Q) = \ell :: Q$, or there are $P', Q' \in \mathcal{P}_1$ and $R \in \mathcal{P}_2$ such that $P' \xrightarrow{a} Q'$, $\mathcal{L}_\rho(P) = (\ell :: P') \parallel R$ and $\mathcal{L}_\rho(Q) = (\ell :: Q') \parallel R$.

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

Basic step. The proof has length 1. Then rule S1 has been used and there is P_1 such that: $P = a.P_1$ and $Q = P_1$. By definition, we have $\mathcal{L}_\rho(P) = \rho :: a.P_1$ and $\mathcal{L}_\rho(Q) = \rho :: P_1$ (because P_1 cannot be of the form $P_1' \parallel P_1''$). The thesis follows by applying rule OS1 and taking $\ell = \rho$, $P' = a.P_1$ and $Q' = P_1$.

Inductive step. The proof has length $n > 1$, hence the last applied rule is S2 or its analogous for the operator $|$. In the case S2 is the last applied rule, there are $P_1, P_2, Q_1 \in \mathcal{P}_1$ such that $P = P_1 \parallel P_2$, $P_1 \xrightarrow{a} Q_1$ and $Q = Q_1 \parallel P_2$. By definition, we have that $\mathcal{L}_\rho(P) = \mathcal{L}_{\rho l}(P_1) \parallel \mathcal{L}_{\rho r}(P_2)$ and $\mathcal{L}_\rho(Q) = \mathcal{L}_{\rho l}(Q_1) \parallel \mathcal{L}_{\rho r}(P_2)$. By inductive hypothesis, we have that there is $\ell' \in \Lambda(\mathcal{L}_{\rho l}(P_1))$, such that $\mathcal{L}_{\rho l}(P_1) \xrightarrow{a@\ell'} \mathcal{L}_{\rho l}(Q_1)$ and either $\mathcal{L}_{\rho l}(P_1) = \ell' :: P_1$ and $\mathcal{L}_{\rho l}(Q_1) = \ell :: Q_1$ or there are $P_1', Q_1' \in \mathcal{P}_1$ and $R' \in \mathcal{P}_2$ such that $P_1' \xrightarrow{a} Q_1'$, $\mathcal{L}_{\rho l}(P_1) = (\ell' :: P_1') \parallel R'$ and $\mathcal{L}_{\rho l}(Q_1) = (\ell' :: Q_1') \parallel R'$. Hence, by applying OS3, we get the transition $\mathcal{L}_\rho(P) = \mathcal{L}_{\rho l}(P_1) \parallel \mathcal{L}_{\rho r}(P_2) \xrightarrow{a@\ell'} \mathcal{L}_{\rho l}(Q_1) \parallel \mathcal{L}_{\rho r}(P_2) = \mathcal{L}_\rho(Q)$. Now, $\ell' \in \Lambda(\mathcal{L}_\rho(P))$ because $\Lambda(\mathcal{L}_\rho(P)) = \Lambda(\mathcal{L}_{\rho l}(P_1)) \cup \Lambda(\mathcal{L}_{\rho r}(P_2))$. Then, the thesis follows by taking $\ell = \ell'$ and either $R = \mathcal{L}_{\rho r}(P_2)$ or $R = R' \parallel \mathcal{L}_{\rho r}(P_2)$. In the case the analogous of S2 for the operator $|$ is the last applied rule, then there are $P_1, P_2, Q_1 \in \mathcal{P}_1$ such that $P = P_1 \mid P_2$, $P_1 \xrightarrow{a} Q_1$ and $Q = Q_1 \mid P_2$. Now, by definition, we have $\mathcal{L}_\rho(P) = \rho :: P_1 \mid P_2$ and $\mathcal{L}_\rho(Q) = \rho :: Q_1 \mid P_2$. Hence, from $P_1 \xrightarrow{a} Q_1$ we get $P_1 \mid P_2 \xrightarrow{a} Q_1 \mid P_2$ (by the analogous of S2), from which we get $\rho :: P_1 \mid P_2 \xrightarrow{a@\rho} \rho :: Q_1 \mid P_2$ (by applying OS1). The thesis then follows by taking $\ell = \rho$, $P' = P_1 \mid P_2$ and $Q' = Q_1 \mid P_2$.

The following four propositions formalize the idea that locations can be arbitrarily chosen and processes that only differ for the names of their locations behave similarly. The main point is that the allocation function does not preserve structural equivalence. Indeed, when allocating two structurally equivalent processes, two new processes are obtained that are not structurally equivalent. However, by appropriately renaming the locations of one of the two processes by means of a one-to-one function it is possible to obtain a process which is structurally equivalent to the other one.

We will use the following notation: if $P \in \mathcal{P}_2$, $L \subseteq LOC$ and $\phi : \Lambda(P) \longrightarrow L$ then $\phi(P)$ denotes the process obatined by replacing in P each occurrence of any $\ell \in \Lambda(P)$ with $\phi(\ell)$. A similar notation is used for renaming locations within transition labels. We use $\phi' \circ \phi''$ to denote the composition of functions ϕ' and ϕ'' so that $\phi' \circ \phi''(\ell) = \phi''(\phi'(\ell))$.

Proposition 14. For all $P \in \mathcal{P}_1$ and $\rho, \rho' \in \{l, r\}^*$, there is a one-to-one function $\phi : \Lambda(\mathcal{L}_{\rho'}(P)) \longrightarrow \Lambda(\mathcal{L}_{\rho}(P))$ such that $\mathcal{L}_{\rho}(P) \equiv \phi(\mathcal{L}_{\rho'}(P))$.

Proof. The proof proceeds by induction on the syntax of P.

Basic step. The basic step is when P has one of the following forms: nil, $a.P$ and $Q \mid R$. In any case, by definition of allocation function we have $\mathcal{L}_{\rho}(P) = \rho :: P$ and $\mathcal{L}_{\rho'}(P) = \rho' :: P$, and the thesis follows by taking ϕ such that $\phi : \{\rho'\} \longrightarrow \{\rho\}$ and $\phi(\rho') = \rho$.

Inductive step. We reason by case analisys on the top-level operator in P.

- Suppose that $P = Q \parallel R$. Then, by definition, means that $\mathcal{L}_{\rho}(P) = \mathcal{L}_{\rho l}(Q) \parallel \mathcal{L}_{\rho r}(R)$ and $\mathcal{L}_{\rho'}(P) = \mathcal{L}_{\rho' l}(Q) \parallel (\mathcal{L}_{\rho' r}(R)$. By induction, we can assume that there are two one-to-one functions $\phi_1 : \Lambda(\mathcal{L}_{\rho' l}(Q)) \longrightarrow \Lambda(\mathcal{L}_{\rho l}(Q))$ and $\phi_2 : \Lambda(\mathcal{L}_{\rho' r}(R)) \longrightarrow \Lambda(\mathcal{L}_{\rho r}(R))$ such that $\phi_1(\mathcal{L}_{\rho' l}(Q)) = \mathcal{L}_{\rho l}(Q)$ and $\phi_2(\mathcal{L}_{\rho' r}(R)) = \mathcal{L}_{\rho r}(R)$. Since $dom(\phi_1) \cap dom(\phi_2) = range(\phi_1) \cap range(\phi_2) = \emptyset$, function composition $\phi_1 \circ \phi_2$ gets a one-to-one function and, by taking $\phi = \phi_1 \circ \phi_2$, the thesis follows.
- Suppose that $P = Q \parallel O$ (the symmetric case $P = O \parallel Q$ is dealt with similarly). Then, by definition, means that $\mathcal{L}_{\rho}(P) = \mathcal{L}_{\rho}(Q) \parallel \mathcal{T}(O)$ and $\mathcal{L}_{\rho'}(P) = \mathcal{L}_{\rho'}(Q) \parallel \mathcal{T}(O)$. By induction, we can assume that there are is a one-to-one functions $\phi' : \Lambda(\mathcal{L}_{\rho'}(Q)) \longrightarrow \Lambda(\mathcal{L}_{\rho}(Q))$ such that $\phi_1(\mathcal{L}_{\rho'}(Q)) = \mathcal{L}_{\rho}(Q)$. Now we have two cases to consider. Either $\tau \in \Lambda(\mathcal{L}_{\rho'}(Q))$, then we take $\rho = \rho'$, or $\tau \notin \Lambda(\mathcal{L}_{\rho'}(Q))$, then we take $\rho = \rho' \circ \rho''$ where $\rho'' : \{\tau\} \longrightarrow \{\tau\}$ and $\rho''(\tau) = \tau$. In both cases, the thesis follows.

Proposition 15. For all $P, P' \in \mathcal{P}_1$ and $\rho, \rho' \in \{l, r\}^*$, if $P \equiv P'$ can be proved without using the first structural law, then there is a one-to-one function $\phi : \Lambda(\mathcal{L}_{\rho'}(P')) \longrightarrow \Lambda(\mathcal{L}_{\rho}(P))$ such that $\mathcal{L}_{\rho}(P) \equiv \phi(\mathcal{L}_{\rho'}(P'))$.

Proof. The proof proceeds by induction on the length of the proof of $P \equiv P'$.

Basic step. Only one structural law has been applied to deduce that $P \equiv P'$. The proof for this case proceeds by case analisys on the structural law used.

- Suppose that the structural law applied is $Q_1 \parallel Q_2 \equiv Q_2 \parallel Q_1$, where $P = Q_1 \parallel Q_2$ and $P' = Q_2 \parallel Q_1$. This, by definition, means that $\mathcal{L}_{\rho}(P) = \mathcal{L}_{\rho l}(Q_1) \parallel \mathcal{L}_{\rho r}(Q_2)$ and that $\mathcal{L}_{\rho'}(P') = \mathcal{L}_{\rho' l}(Q_2) \parallel \mathcal{L}_{\rho' r}(Q_1)$. Now, by Proposition 14, we have that there are two one-to-one functions $\phi_1 : \Lambda(\mathcal{L}_{\rho' r}(Q_1)) \longrightarrow \Lambda(\mathcal{L}_{\rho l}(Q_1))$ and $\phi_2 : \Lambda(\mathcal{L}_{\rho' l}(Q_2)) \longrightarrow \Lambda(\mathcal{L}_{\rho l}(Q_2))$ such that $\phi_1(\mathcal{L}_{\rho' r}(Q_1)) = \mathcal{L}_{\rho l}(Q_1)$ and $\phi_2(\mathcal{L}_{\rho' l}(Q_2)) = \mathcal{L}_{\rho r}(Q_2)$. Since $dom(\phi_1) \cap dom(\phi_2) = range(\phi_1) \cap range(\phi_2) = \emptyset$, function composition $\phi_1 \circ \phi_2$ gets a one-to-one function and, by taking $\phi = \phi_1 \circ \phi_2$, the thesis follows.

  If the structural law applied is $Q_1 \mid Q_2 \equiv Q_2 \mid Q_1$, where $P = Q_1 \mid Q_2$ and $P' = Q_2 \mid Q_1$, then, by definition, $\mathcal{L}_{\rho}(P) = \rho :: (Q_1 \mid Q_2)$ and $\mathcal{L}_{\rho'}(P') = \rho' :: (Q_2 \mid Q_1)$, and the thesis follows by taking ϕ such that $\phi : \{\rho'\} \longrightarrow \{\rho\}$ and $\phi(\rho') = \rho$.

– Suppose that the structural law applied is $Q_1 \parallel (Q_2 \parallel Q_3) \equiv (Q_1 \parallel Q_2) \parallel Q_3$, where $P = Q_1 \parallel (Q_2 \parallel Q_3)$ and $P' = (Q_1 \parallel Q_2) \parallel Q_3$. This, by definition, means that $\mathcal{L}_\rho(P) = \mathcal{L}_{\rho l}(Q_1) \parallel (\mathcal{L}_{\rho rl}(Q_2) \parallel \mathcal{L}_{\rho rr}(Q_3))$ and that $\mathcal{L}_{\rho'}(P') = (\mathcal{L}_{\rho' ll}(Q_1) \parallel \mathcal{L}_{\rho' lr}(Q_2)) \parallel \mathcal{L}_{\rho' r}(Q_3)$. Now, by Proposition 14, we have that there are three one-to-one functions $\phi_1 : \Lambda(\mathcal{L}_{\rho' ll}(Q_1)) \longrightarrow \Lambda(\mathcal{L}_{\rho l}(Q_1))$, $\phi_2 : \Lambda(\mathcal{L}_{\rho' lr}(Q_2)) \longrightarrow \Lambda(\mathcal{L}_{\rho rl}(Q_2))$, and $\phi_3 : \Lambda(\mathcal{L}_{\rho' r}(Q_3)) \longrightarrow \Lambda(\mathcal{L}_{\rho rr}(Q_3))$ such that $\phi_1(\mathcal{L}_{\rho' ll}(Q_1)) = \mathcal{L}_{\rho l}(Q_1)$, $\phi_2(\mathcal{L}_{\rho' lr}(Q_2)) = \mathcal{L}_{\rho rl}(Q_2)$ and $\phi_3(\mathcal{L}_{\rho' r}(Q_3)) = \mathcal{L}_{\rho rr}(Q_3)$. Since definition domains and ranges of these functions are pairwise disjoint, their composition is a one-to-one function too and, by taking $\phi = \phi_1 \circ \phi_2 \circ \phi_3$, the thesis follows.
If the structural law applied is $Q_1 \mid (Q_2 \mid Q_3) \equiv (Q_1 \mid Q_2) \mid Q_3$, where $P = Q_1 \mid (Q_2 \mid Q_3)$ and $P' = (Q_1 \mid Q_2) \mid Q_3$, then, by definition, $\mathcal{L}_\rho(P) = \rho :: Q_1 \mid (Q_2 \mid Q_3)$ and $\mathcal{L}_{\rho'}(P') = \rho' :: (Q_1 \mid Q_2) \mid Q_3$, and the thesis follows by taking ϕ such that $\phi : \{\rho'\} \longrightarrow \{\rho\}$ and $\phi(\rho') = \rho$.

Inductive step. In this case the proof proceeds by case analisys on the last structural law applied when proving that $P \equiv P'$. Suppose that $P'' \in \mathcal{P}_1$ is such that $P \equiv P''$ and $P'' \equiv P'$ can be proved without using the first structural law and the last equivalence is proved by a single application of one of the remaining structural laws. By induction, for any $\rho'' \in \{l, r\}^*$, there is a one-to-one function $\phi'' : \Lambda(\mathcal{L}_{\rho''}(P'')) \longrightarrow \Lambda(\mathcal{L}_\rho(P))$ such that $\mathcal{L}_\rho(P) \equiv \phi''(\mathcal{L}_{\rho''}(P''))$. By reasoning as in the *basic step*, there is a one-to-one function $\phi' : \Lambda(\mathcal{L}_{\rho'}(P')) \longrightarrow \Lambda(\mathcal{L}_{\rho''}(P''))$ such that $\mathcal{L}_{\rho''}(P'') \equiv \phi'(\mathcal{L}_{\rho'}(P'))$. Then, the thesis follows by taking $\phi = \phi' \circ \phi''$.

Proposition 16. For all $P, P' \in \mathcal{P}_2$, $a \in \mathcal{L}_1$, $\ell \in \Lambda(P)$, $L \subseteq LOC$ and $\phi : \Lambda(P) \longrightarrow L$ one-to-one function, if $P \xrightarrow{a@\ell} P'$ then $\phi(P) \xrightarrow{a@\phi(\ell)} \phi(P')$.

Proof. By easy induction on the length of the proof of $P \xrightarrow{a@\ell} P'$.

Proposition 17. For all $P, P', Q' \in \mathcal{P}_1$, $\rho, \rho' \in \{l, r\}^*$ and $\phi : \Lambda(\mathcal{L}_{\rho'}(P')) \longrightarrow \Lambda(\mathcal{L}_\rho(P))$ one-to-one function, if $P \equiv P'$ can be proved without using the first structural law, $\phi(\mathcal{L}_{\rho'}(P')) \equiv \mathcal{L}_\rho(P)$ and $\Lambda(\mathcal{L}_{\rho'}(P')) = \Lambda(\mathcal{L}_{\rho'}(Q'))$ then there is $Q \in \mathcal{P}_1$ such that $Q \equiv Q'$ and $\phi(\mathcal{L}_{\rho'}(Q')) \equiv \mathcal{L}_\rho(Q)$.

Proof. The proof relies on the following fact:

> if $\Lambda(\mathcal{L}_{\rho'}(P')) = \Lambda(\mathcal{L}_{\rho'}(Q'))$ then processes P' and Q' have the same number of sequential components and the same structure with respect to parallel composition

that can be proved by induction on the syntax of P'. It is then possibile to define a one-to-one function π from the sequential components of P' to those of Q' so that $\pi(P') = Q'$. Since $P \equiv P'$ can be proved without using the first structural law, P and P' have the same sequential componets and only differ for their composition. This implies that $\pi(P) \in \mathcal{P}_1$, and that processes $\pi(P)$ and Q' have the same sequential components and only differ for their

composition, hence, by using the structural rules it is possible to prove that $\pi(P) \equiv Q'$. Now, take $Q = \pi(P)$. Since function π commutes both with $\mathcal{L}$ and with ϕ, from the hipothesis that $\phi(\mathcal{L}_{\rho'}(P')) \equiv \mathcal{L}_\rho(P)$ that also implies that $\phi(\mathcal{L}_{\rho'}(P'))$ and $\mathcal{L}_\rho(P)$ have the same sequential components, it follows that $\phi(\mathcal{L}_{\rho'}(Q')) = \phi(\mathcal{L}_{\rho'}(\pi(P'))) = \pi(\phi(\mathcal{L}_{\rho'}(P'))) \equiv \pi(\mathcal{L}_\rho(P)) = \mathcal{L}_\rho(\pi(P)) = \mathcal{L}_\rho(Q)$, and the thesis is proved.

Now, by exploiting the previous properties, we are able to prove the relationship between the transitions of the original semantics and those of the optimized one. Notice that the states of the optimized semantics can also consist of ghost tuples.

Proposition 18. For all $P, Q \in \mathcal{P}_1$, $a \in \mathcal{L}_1$, $\rho \in \{l, r\}^*$, $L \subseteq \Lambda(\mathcal{L}_\rho(P))$ and $v_i \in VAL$ $(\forall i : \ell_i \in L)$, $P \xrightarrow{a} Q$ implies that there are $R \in \mathcal{P}_1$, $\ell \in \Lambda(\mathcal{L}_\rho(P))$ and $L' \subseteq \Lambda(\mathcal{L}_\rho(R))$ such that $R \equiv Q$ and $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{a@\ell} \mathcal{L}_\rho(R) \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i)$.

Proof. By induction on the length of the derivation of transition $P \xrightarrow{a} Q$.

Basic step. The transition has length 1 and one of rules S3, S4 or S5 is the only rule used to infer the transition. We proceed by case analysis on the applied rule.

Rule S3*:* Then $a = out(v)$ for some $v \in VAL$ and there are $e \in EXP$ and $P' \in \mathcal{P}_1$ such that $[\![e]\!] = v$, $P \xrightarrow{out(e)} P'$ and $Q = P' \parallel \underline{out}(v)$. By Proposition 13, we have that $\mathcal{L}_\rho(P) \xrightarrow{out(e)@\ell} \mathcal{L}_\rho(P')$ for some $\ell \in \Lambda(\mathcal{L}_\rho(P))$. Hence, by applying rule OS3, we get that $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{out(e)@\ell} \mathcal{L}_\rho(P') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. Now, by applying rule OS5, we get the transition $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{out(v)@\ell} (\mathcal{L}_\rho(P') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)) \setminus \underline{\ell} \parallel \tau :: \underline{out}(v)$. By Proposition 2, we have that $(\mathcal{L}_\rho(P') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)) \setminus \underline{\ell} \parallel \tau :: \underline{out}(v) \equiv \mathcal{L}_\rho(P') \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i) \parallel \tau :: \underline{out}(v)$. By definition of allocation function, we have that $\mathcal{L}_\rho(P') \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i) \parallel \tau :: \underline{out}(v) \equiv \mathcal{L}_\rho(P' \parallel out(v)) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$ and the thesis follows by taking $R = Q$ and $L' = L \setminus \{\ell\}$, and by applying rule OS8.

Rule S4*:* Then $a = rd(v)$ for some $v \in VAL$ and there are a template t and $P', P'' \in \mathcal{P}_1$ such that $[\![\, t \,]\!] = v$ or $t \in VAR$, $P = P' \parallel \underline{out}(v)$, $P' \xrightarrow{rd(t)} P''$ and $Q = P''[v/t] \parallel \underline{out}(v)$. By definition, $\mathcal{L}_\rho(P) = \mathcal{L}_\rho(P') \parallel \tau :: \underline{out}(v)$, and, by Proposition 13, $\mathcal{L}_\rho(P') \xrightarrow{rd(t)@\ell} \mathcal{L}_\rho(P'')$ for some $\ell \in \Lambda(\mathcal{L}_\rho(P'))$. Hence, by applying rule OS3, we get that $\mathcal{L}_\rho(P') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{rd(t)@\ell} \mathcal{L}_\rho(P'') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. Then, by applying rules OS6 and OS8, we get the transition $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{rd(v)@\ell} (\mathcal{L}_\rho(P'') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i))[v/t] \setminus \underline{\ell} \parallel \tau :: \underline{out}(v)$. Now, function $\mathcal{L}_\rho$ and substitution $[v/t]$ commute, hence $(\mathcal{L}_\rho(P'') \parallel$

$\Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i))[v/t] = \mathcal{L}_\rho(P''[v/t]) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. Moreover, since $\underline{\ell} \notin \Lambda(\mathcal{L}_\rho(P''[v/t]))$, by Proposition 2, we have that $(\mathcal{L}_\rho(P''[v/t]) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)) \setminus \underline{\ell} \equiv \mathcal{L}_\rho(P''[v/t]) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$. Finally, by definition of allocation function, $\mathcal{L}_\rho(P''[v/t]) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i) \parallel \tau :: \underline{out}(v) \equiv \mathcal{L}_\rho(P''[v/t] \parallel \underline{out}(v)) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$. The thesis follows by taking $L' = L \setminus \{\ell\}$ and by applying rule OS8.

Rule S5*:* Then $a = in(v)$ for some $v \in VAL$ and there are a template t and $P', P'' \in \mathcal{P}_1$ such that $[\![t]\!] = v$ or $t \in VAR$, $P = P' \parallel \underline{out}(v)$, $P' \xrightarrow{in(t)} P''$ and $Q = P''[v/t]$. By definition, $\mathcal{L}_\rho(P) = \mathcal{L}_\rho(P') \parallel \tau :: \underline{out}(v)$, and, by Proposition 13, $\mathcal{L}_\rho(P') \xrightarrow{in(t)@\ell} \mathcal{L}_\rho(P'')$ for some $\ell \in \Lambda(\mathcal{L}_\rho(P))$. Hence, by applying rule OS3, we get that $\mathcal{L}_\rho(P') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{in(t)@\ell} \mathcal{L}_\rho(P'') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. Then, by applying rules OS7 and OS8, we get the transition $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{in(v)@\ell} (\mathcal{L}_\rho(P'') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i))[v/t] \setminus \underline{\ell} \parallel \underline{\ell} :: \underline{out}(v)$. Now, function $\mathcal{L}_\rho$ and substitution $[v/t]$ commute, hence $(\mathcal{L}_\rho(P'') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i))[v/t] = \mathcal{L}_\rho(P''[v/t]) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. Moreover, since $\underline{\ell} \notin \Lambda(\mathcal{L}_\rho(P''[v/t]))$, by Proposition 2, we have that $(\mathcal{L}_\rho(P''[v/t]) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)) \setminus \underline{\ell} \equiv \mathcal{L}_\rho(P''[v/t]) \parallel \Pi_{\ell_i \in L \setminus \{\ell\}} \underline{\ell_i} :: \underline{out}(v_i)$. The thesis follows by taking $L' = L$ (with $v_i = v$ for $\ell_i = \ell$) and by applying rule OS8.

Inductive step. The last applied rule is S6, hence there are $P', Q' \in \mathcal{P}_1$ such that $P \equiv P'$, $P' \xrightarrow{a} Q'$ and $Q \equiv Q'$. Without loss of generality, we can assume that $P \equiv P'$ (and $Q \equiv Q'$) is proved without using the first structural law, hence P and P' (resp. Q and Q') have the same number of parallel components. Indeed, it can be easily seen that, for $R_1, R_1', R_2 \in \mathcal{P}_1$ and $a' \in \mathcal{L}_1$, $R_1 \parallel R_2 \xrightarrow{a'} R_1' \parallel R_2$ if and only if $R_1 \parallel nil \parallel R_2 \xrightarrow{a'} R_1' \parallel nil \parallel R_2$. By induction we have that for any given $\rho' \in \{l, r\}^*$ and $L_1 \subseteq \Lambda(\mathcal{L}_{\rho'}(P'))$, there are $R' \in \mathcal{P}_2$, $\ell' \in \Lambda(\mathcal{L}_{\rho'}(P'))$ and $L_2 \subseteq \Lambda(\mathcal{L}_{\rho'}(R'))$ such that $R' \equiv Q'$ and $\mathcal{L}_{\rho'}(P') \parallel \Pi_{\ell_i \in L_1} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{a@\ell'} \mathcal{L}_{\rho'}(R') \parallel \Pi_{\ell_i \in L_2} \underline{\ell_i} :: \underline{out}(v_i)$. Since P and P' have the same number of parallel components, by Proposition 15, there is a one-to-one function $\phi : \Lambda(\mathcal{L}_{\rho'}(P')) \longrightarrow \Lambda(\mathcal{L}_\rho(P))$ such that $\mathcal{L}_\rho(P) \equiv \phi(\mathcal{L}_{\rho'}(P'))$ and, then, that $\mathcal{L}_\rho(P) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i) \equiv \phi(\mathcal{L}_{\rho'}(P') \parallel \Pi_{\ell_i \in L_1} \underline{\ell_i} :: \underline{out}(v_i))$. By Proposition 16, we have that $\phi(\mathcal{L}_{\rho'}(P') \parallel \Pi_{\ell_i \in L_1} \underline{\ell_i} :: \underline{out}(v_i) \xrightarrow{a@\phi(\ell')} \phi(\mathcal{L}_{\rho'}(R') \parallel \Pi_{\ell_i \in L_2} \underline{\ell_i} :: \underline{out}(v_i))$. Finally, by Proposition 17, we have that there is $R \in \mathcal{P}_1$ such that $R \equiv R'$ and $\mathcal{L}_\rho(R) \equiv \phi(\mathcal{L}_{\rho'}(R'))$. Then, by taking $L' = \phi(L_2)$, we have that $\mathcal{L}_\rho(R) \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i) \equiv \phi(\mathcal{L}_{\rho'}(R') \parallel \Pi_{\ell_i \in L_2} \underline{\ell_i} :: \underline{out}(v_i))$, from which, by applying rule OS8, the thesis follows.

The previous property is now generalized to nonempty sequences of transitions.

Proposition 19. For all $P, Q \in \mathcal{P}_1$, $\rho \in \{l, r\}^*$ and $s \in \mathcal{L}_1^+$, $P \xrightarrow{s} Q$ implies that there are $R \in \mathcal{P}_1$, $\sigma \in \mathcal{L}_2^+$, $L \subseteq \Lambda(\mathcal{L}_\rho(R))$ and $v_i \in VAL$ $(\forall i : \ell_i \in L)$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma} \mathcal{L}_\rho(R) \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$, $s = \mathcal{C}(\sigma)$ and $R \equiv Q$.

Proof. By induction on the length of the sequence s.

Basic step. In this case $s = a$ for some $a \in \mathcal{L}_1$ and the thesis is an immediate consequence of Proposition 18.

Inductive step. Suppose now that $s = s'a$ for some $s' \in \mathcal{L}_1^+$ and $a \in \mathcal{L}_1$. Then there is $P' \in \mathcal{P}_1$ such that $P \xrightarrow{s'} P' \xrightarrow{a} Q$. By induction, there are $R' \in \mathcal{P}_1$, $\sigma' \in \mathcal{L}_2^+$, $L' \subseteq \Lambda(\mathcal{L}_\rho(R'))$ and $v_i' \in VAL$ $(\forall i : \ell_i \in L')$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma'} \mathcal{L}_\rho(R') \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i')$, $s' = \mathcal{C}(\sigma')$ and $R' \equiv P'$. Now, from $R' \equiv P'$ and from the hypothesis $P' \xrightarrow{a} Q$, by applying S6, we get that $R' \xrightarrow{a} Q$. Hence, from Proposition 18, there are $R'' \in \mathcal{P}_1$, $\rho \in \{l, r\}^*$ and $L'' \subseteq \Lambda(\mathcal{L}_\rho(R''))$ such that $R'' \equiv Q$ and $\mathcal{L}_\rho(R') \parallel \Pi_{\ell_i \in L'} \underline{\ell_i} :: \underline{out}(v_i') \xrightarrow{a@\ell} \mathcal{L}_\rho(R'') \parallel \Pi_{\ell_i \in L''} \underline{\ell_i} :: \underline{out}(v_i')$. This concludes the proof (just take $\sigma = \sigma' \cdot a@\ell$, $R = R''$ and $L = L''$).

Finally, we can prove that the optimized semantics can simulate the original one.

Theorem 2. For all $P, Q \in \mathcal{P}_1$, $\rho \in \{l, r\}^*$ and $s \in \mathcal{L}_1^*$, $P \xrightarrow{s} Q$ implies that there are $R \in \mathcal{P}_2$ and $\sigma \in \mathcal{L}_2^*$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma} R$, $s = \mathcal{C}(\sigma)$ and $\mathcal{C}(R) \equiv Q$.

Proof. If $s = \epsilon$ then $Q = P$; hence, taken $\sigma = \epsilon$ and $R = \mathcal{L}_\rho(P)$, the thesis follows from Proposition 3. Otherwise, from Proposition 19 it follows that there are $R' \in \mathcal{P}_1$, $\sigma \in \mathcal{L}_2^+$, $L \subseteq \Lambda(\mathcal{L}_\rho(R'))$ and $v_i \in VAL$ $(\forall i : \ell_i \in L)$ such that $\mathcal{L}_\rho(P) \xrightarrow{\sigma} \mathcal{L}_\rho(R') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$, $s = \mathcal{C}(\sigma)$ and $R' \equiv Q$. Now, take $R = \mathcal{L}_\rho(R') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)$. By definition of $\mathcal{C}$, we have that $\mathcal{C}(\mathcal{L}_\rho(R') \parallel \Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)) = \mathcal{C}(\mathcal{L}_\rho(R')) \parallel \mathcal{C}(\Pi_{\ell_i \in L} \underline{\ell_i} :: \underline{out}(v_i)) \equiv \mathcal{C}(\mathcal{L}_\rho(R'))$. Moreover, by Proposition 3, we have that $\mathcal{C}(\mathcal{L}_\rho(R')) = R'$ which implies the thesis.

5 Conclusion

We have described a tuple ghosting optimisation that allows tuples to be still used as the results of non-destructive tuple space accesses once they have been destructively removed. The motivation for tuple ghosting has been briefly outlined, as have some practical results from a prototype system demonstrating the advantage of the approach.

The operational semantics of the original Linda and of the version with the optimisation are illustrated. Using these operational semantics, we have presented a the formal proof of the tuple ghosting optimisation, and shown that the optimisation does not alter the semantics of the primitives from a programmers' perspective. This has been achieved by proving that the optimised semantics can simulate the original semantics, and that a sequence of transitions from the optimised semantics can be mimicked by a sequence of transitions from the original semantics.

References

1. A. Rowstron. Optimising the Linda `in` primitive: Understanding tuple-space runtimes. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *Proceedings of the 2000 ACM Symposium on Applied Computing*, volume 1, pages 227–232. ACM Press, March 2000.
2. Sun Microsystems. Javaspace specification. available at: `http://java.sun.com/`, 1999.
3. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.
4. N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.
5. P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche. Coordinating multiagent applications on the WWW: A reference architecture. *IEEE Transactions on Software Engineering*, 24(5):362–366, 1998.
6. P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. TSpaces. *IBM Systems Journal*, 37(3):454–474, 1998.
7. A. Omicini and F. Zambonelli. Coordination for internet application development. *Autonomous Agents and Multi-agent Systems*, 2(3):251–269, 1999. Special Issue on Coordination Mechanisms and Patterns for Web Agents.
8. A. Rowstron. WCL: A web co-ordination language. *World Wide Web Journal*, 1(3):167–179, 1998.
9. J. Carreria, L. Silva, and J. Silva. On the design of Eilean: A Linda-like library for MPI. Technical report, Universidade de Coimbra, 1994.
10. N. Carriero and D. Gelernter. Tuple analysis and partial evaluation strategies in the Linda precompiler. In D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and Compilers for Parallel Computing*, Research Monographs in Parallel and Distributed Computing, pages 114–125. MIT Press, 1990.
11. R. De Nicola and R. Pugliese. A process algebra based on linda. In P. Ciancarini and C. Hankin, editors, *Proceedings of the First International Conference on Coordination Models and Languages (COORDINATION'96)*, volume 1061 of *Lecture Notes in Computer Science*, pages 160–178. Springer, 1996.
12. G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Dep. of Computer Science, Aarhus University, Denmark, 1981.