

A Timed Linda Language and its Denotational Semantics

F.S. de Boer

CWI

Kruislaan 413

NL-1098 SJ Amsterdam, The Netherlands

F.S.de.Boer@cwi.nl

M. Gabbrielli

Università di Bologna

Dipartimento di Scienze dell'Informazione

Mura A. Zamboni 7, 40127 Bologna, Italy

gabbri@cs.unibo.it

M.C. Meo

Università di Chieti-Pescara

Dipartimento di Scienze, Viale Pindaro 42

65127 Pescara, Italy

cmeo@unich.it

We introduce a Timed Linda language (T-Linda) which is obtained by a natural timed interpretation of the usual constructs of the Linda model and by adding a simple primitive which allows one to specify time-outs. Parallel execution of processes follows the scheduling policy of interleaving, however maximal parallelism is assumed for actions depending on time. We first define the operational semantics of T-Linda by means of a transition system, then we define a denotational model which is based on *timed reactive sequences*. The correctness of this model is proved w.r.t. a notion of observables which includes finite traces of actions and input/output pairs.

1. Introduction

Recently there has been some interest in the investigation of temporal properties of Linda-like coordination languages [13, 21, 5]. Time critical features are in fact important in the context of coordination of complex applications for open, distributed systems, where often one has the need to express such timing constraints as upper limits on the wait for an event, that is time-outs, and fixed (bounded) duration for granting a service, so called leasing. Consider for example the case of a bank teller machine. Once a card is accepted and its identification number has been checked, the machine asks the authorization of the bank to release the requested money. If the authorization does not arrive within a reasonable amount of time then the card should be given back to the customer. In order to model such a situation the language should allow us to specify that, in case a given time bound is exceeded (i.e. a time-out occurs), the wait is interrupted and an alternative action is taken. Indeed, time-outs can be expressed in JavaSpaces [31] and TSpaces [33], two coordination middlewares for distributed Java programming based on the Linda model [18, 14] and produced by Sun and IBM, respectively.

Temporal aspects of concurrent computations have been extensively studied in many different formal settings, including timed process algebras, temporal logic (and its executable versions) and the concurrent synchronous languages ESTEREL, LUSTRE, SIGNAL and Statecharts. In particular, timed process algebras (e.g. see [1, 2, 9, 20]) can be used to specify and verify large, non-deterministic reactive and real-time systems while deterministic concurrent synchronous languages such as ESTEREL [3] have specifically been designed for programming “kernels” of reactive systems.

The underlying assumption of the ESTEREL computational model is the *instantaneous reaction* (or perfect synchrony) hypothesis: A program is activated by some *input signals* and reacts *instantly* by producing, deterministically, the required output. So computation is performed in no time, unless a statement which explicitly consumes time is present. Communication is done by instantaneous broadcasting to all the processes of the system and the presence or absence of a signal can be detected at any instant.

A different approach to specify and program reactive systems has recently been defined [26, 27, 4] in the context of *concurrent constraint programming (ccp)* [28, 29, 30]. Ccp is a programming paradigm quite similar to Linda: In both cases the computation proceeds via accumulation of information in a global shared store, and information is produced by the concurrent and asynchronous activity of several processes. These can also check for the presence of information in the store, thus allowing one to express synchronization and coordination of different processes. However, while tuples are used in Linda, in ccp the information is represented in terms of constraints. Furthermore, differently from the case of Linda, in ccp once the information is produced it cannot be removed from the store, so the latter grows monotonically. In [26, 27, 4] timed extensions of ccp were defined around the hypothesis of *bounded asynchrony* [26]: Computation takes a bounded period of time (rather than being instantaneous as in ESTEREL) and the whole system evolves in cycles corresponding to time-units. While the language defined in [26, 27] is a deterministic one, inspired to synchronous languages and useful mainly for programming small real-time kernels, the timed ccp defined in [4] includes non-determinism and is more appropriate for specifying large systems involving several processes, possibly running on different processors, communicating via asynchronous links.

In this paper we investigate from a semantic perspective a timed extension of Linda that we call T-Linda.

We introduce directly a timed interpretation of the usual programming constructs of Linda by identifying a time-unit with the time needed for the execution of a basic Linda action (out, in and rd), and by interpreting action prefixing as the next-time operator. An explicit timing primitive is also introduced in order to allow for the specification of time-outs. The parallel operator of T-Linda is interpreted in terms of interleaving, as usual, however we assume that there exists a global clock which synchronizes all the parallel time-outs involved in a computation step. In other words, while only one process can perform a basic Linda action in one time unit, time passes for all the parallel processes involved in a computation. This approach is different from that one of [4], where maximal parallelism was assumed, and it is also different from those considered in [13, 21] (see Section 5 for further details).

We describe denotationally T-Linda by defining a fix-point compositional semantics which is correct w.r.t. a notion of observables which considers sequences of computation steps (input/output pairs can be obtained as a suitable abstraction of these observables). This denotational semantics is based on sequences of triples consisting of two multisets of tuples and a label. As we discuss later in the paper, these sequences are similar to, so called, reactive sequences which were used in the semantics of several other languages. However reactive sequences are here provided with a different interpretation which accounts for the timing aspects: Intuitively, each triple $\langle m_i, m'_i, x \rangle$ represents a “reaction”, that is computation step performed by the process P which, at time i , assuming m_i as input tuple-space (i.e. store) produces the space m'_i by performing an action of “type” x . This latter information is needed to distinguish τ actions which represent passing of time only, from the basic, so called σ , actions on the global store. As previously mentioned, all the parallel processes performing τ actions can proceed in the same time unit, while processes performing σ actions follow an interleaving scheduling policy.

The denotational semantics and its relation with the operational semantics is the very contribution of the paper. We show correctness which in general is not a trivial matter because it compares two semantics of a completely different nature. Furthermore, it is not the case that we can obtain the denotational semantics easily by a “generous interpretation” which adds more sequences, because in general this may give rise to a courser semantics which is not compositional anymore. On the other hand, clearly it would be interesting to obtain a semantics containing the minimum amount of information needed to achieve compositionality and correctness, that is, it would be desirable to obtain a full abstraction result. We do not consider here this issue and we leave it for further work.

The remaining of this paper is organized as follows. Next Section introduces the language T-Linda and its operational semantics. In Section 3 we introduce the denotational model and in Section 4 we prove its correctness. Section 5 concludes by discussing related work and by indicating directions for future research. A short version of this work appeared in [6].

2. The Timed Linda language

In this section we introduce the *Timed Linda* (*T-Linda*) language and provide its operational semantics.

The basic idea underlying Linda [18, 14] is that computation progresses via accumulation of information, represented in terms of tuples, in a global shared multiset called tuple space or store. Here we abstract from the specific nature of tuples and assume that these are elementary objects ranged over by a, b, \dots . Information is produced and removed by the concurrent and asynchronous activity of several processes

which can add a tuple a to the store by using the basic operation $\text{out}(a)$. Dually, processes can also remove a tuple from the store by performing an $\text{in}(a)$ operation and read a tuple from the store by means of a $\text{rd}(a)$ operation. Differently from the case of $\text{out}(a)$, both $\text{in}(a)$ and $\text{rd}(a)$ are blocking operations, that is, if a is not present in the store then the evaluation of $\text{in}(a)$ and $\text{rd}(a)$ is suspended, thus allowing one to express coordination among different processes. A kind of *if_then_else* is also present in the form of a construct $\text{rdp}(a)?P_Q$: If a is present in the store then the process P is evaluated, otherwise the computation proceeds with the process Q (an analogous construct $\text{inp}(a)?P_Q$ differs from the previous one only in that the tuple a , whenever present, is removed from the store). The \parallel operator allows one to express parallel composition of two processes and it is usually described in terms of interleaving.

When querying the store for some information which is not present (yet) a process will either suspend until the required information has arrived (in case of $\text{in}(a)$ and $\text{rd}(a)$ processes) or will take an alternative continuation (in case of $\text{rdp}(a)?P_Q$ and $\text{inp}(a)?P_Q$). However, as previously mentioned, in many practical cases often one has the need to express time-outs. Consider for example the case of a bank teller machine. Once a card is accepted and its identification number has been checked, the machine asks the authorization of the bank to release the requested money. If the authorization does not arrive within a reasonable amount of time, then the card should be given back to the customer.

In order to enrich *Linda* with such timing mechanisms, we assume the existence of a *discrete global clock* and we assume that the basic actions $\text{out}(a)$, $\text{in}(a)$ and $\text{rd}(a)$ take one time-unit. T-Linda computations evolve in steps of one time-unit, so called clock-cycles, and action prefixing is the syntactic marker which distinguishes a time instant from the next one. So, for example, the process $\text{out}(a).P$ has to be regarded as the process which updates the current store by adding a and then, at the *next* time instant, behaves like P . Analogously, if a is contained in the current store then the process $\text{in}(a).P$ behaves like P at the next time instant, after having removed a from the store. If a is not present in the store at time t then the process $\text{in}(a).P$ is suspended, i.e. at time $t + 1$ it is checked again whether the store contains a . The process $\text{rd}(a).P$ behaves like $\text{in}(a).P$ without removing information from the store.

The parallel construct is interpreted in terms of interleaving, as usual in many (timed) process algebras and in all the main Linda dialects. Alternatively one could adopt maximal parallelism, which means that at each moment every enabled process of the system is activated. However this implies that parallel processes are executed on different processors and that, in principle, an unbounded number of processors is required, since dynamic process creation is allowed. Furthermore, competing requests for removing the same tuple should be scheduled: For example, if the store contains only one occurrence of the tuple a , then only one of the parallel requests $\text{in}(a) \parallel \text{in}(a)$ should be satisfied. For these reasons, interleaving of accesses to the tuple space seems a more appropriate choice than maximal parallelism for T-Linda. In the case of the timed extension of concurrent constraint programming (ccp) that we defined in [4] we adopted maximal parallelism because, differently from the case of Linda, ccp allows only a monotonic accumulation of information, hence there is no need to schedule competing requests for accessing the store.

Time-outs are modelled in T-Linda by the construct $\text{rdp}(a)_t?P_Q$ whose meaning is analogous to that one of the un-timed version, with the difference that here one is allowed to wait t time units for the presence of the tuple a in the store and the subsequent evaluation of the process P ; If this time limit is exceeded then the process Q is evaluated. Thus, in case the tuple a is not present in the tuple space the temporal behavior of $\text{rdp}(a)_t?P_Q$ is in between the infinite wait of $\text{rd}(a)$ and the no-wait of $\text{rdp}(a)?P_Q$. A similar

construct $\text{inp}(a)_t?P_Q$ which removes also the tuple a from the store could be defined analogously, we omit it since its semantic treatment would be similar to that one of $\text{rdp}(a)_t?P_Q$.

Non-determinism arises in T-Linda by allowing a *choice* operator $P + Q$, as in the case of classical timed process algebras. It is worth recalling that non-determinism arises naturally when considering large reactive systems running on different processors and communicating via asynchronous links. Furthermore, absence of non-determinism rules out the possibility of expressing interesting temporal behaviors like bounded response properties [Koy90] which express that “something good will happen within a certain but not fixed amount of time”. For example, a property like “whenever signal a is present, the system must output b within three time units” cannot be directly expressed without restricting the exact time when b can happen. As argued in [32], the ability to non-deterministically choose among several alternatives can be useful in some areas involving reactive behaviors, where a certain degree of unpredictable behavior is required. For example, non-determinism is needed to model the zig-zagging behavior of a robot which can go left, forward or right at each time unit, with the constraint that the move in the current time unit is different from that one of the previous time unit.

A notion of locality is also present and it is obtained by introducing the process $P \setminus a$ which behaves like P , with a considered *local* to P .

Summarizing, we obtain the following syntax.

Definition 2.1. (Timed Linda Language)

Assuming a given set of tuples T , with typical elements a, b, \dots and a set of process variables ranged over by X, \dots . The syntax of the *Timed Linda terms* is given by the following grammar:

$$P ::= \text{stop} \mid \text{out}(a).P \mid \text{in}(a).P \mid \text{rd}(a).P \mid \text{rdp}(a)_t?P_P \mid \\ P \parallel P \mid P + P \mid P \setminus a \mid X \mid \text{rec } X.P$$

where t is a natural number greater or equal to 0 and the process variable X is bound in the recursion $\text{rec } X.P$. *Timed Linda processes* are defined as Timed Linda terms which have no free process variables.

In the previous definition, as usual, we assume that only guarded forms of recursion are used, that is, a recursive call is always prefixed by an *in*, *out*, *rd* or *rdp* action.

2.1. Operational semantics

The operational model of *T-Linda* can be formally described by a labeled transition system $T = (\text{Conf}, \text{Label}, \longrightarrow)$ where we assert that each transition step takes exactly one time-unit. Configurations *Conf* are pairs consisting of a process and a multi-set of tuples (the tuple space, also called store). In the following \uplus denotes the multisets union, while $\text{Label} = \{\tau, \sigma\}$ is the set of labels. We use labels to distinguish “real” computational steps performed by processes which have the control (label σ) from the transitions which model only the passing of time (label τ). So σ -actions are those performed by processes which modify the store (*out*, *in*), which perform a check on the store (*rd*, *rdp* _{t}) or that correspond to exceeding a time-out (*rdp*₀) and which perform a choice ($P + Q$). On the other hand τ -actions are those performed by time-out processes (*rdp* _{t}) in case they have not the control.

The transition relation $\longrightarrow_{\subseteq} Conf \times Labels \times Conf$ is the least relation satisfying the (axioms and the) rules **R1-R14** in Table 1 and characterizes the (temporal) evolution of the system. So, $\langle P, m \rangle \xrightarrow{x} \langle Q, m' \rangle$ with $x \in \{\sigma, \tau\}$ means that if at time t we have the process P and the store m , then at time $t + 1$ we have the process Q and the store m' . Let us now briefly discuss the rules in Table 1.

The process `stop` represents successful termination, so it cannot make any transition. Axiom **R1** shows that we are considering a non-blocking output operation: The process `out(a).P` adds a to the tuple space m and then it behaves as P at the next time instant. Note that the evaluation of an output action takes one time-unit and the updated space $m \uplus \{a\}$ will be visible only starting from the next time instant.

Both `in(a)` and `rd(a)` are blocking operations, as shown by the axioms **R2** and **R3**: In case the store does not contain a the computation is suspended, otherwise the computation proceeds (removing a only in case of the `in` operation). As it results from the transition rules, also the evaluation of the `in(a)` and `rd(a)` actions takes one time-unit.

The axioms **R4-R7** show that the time-out process `rdp(a)t?P_Q` behaves either as P or as Q depending on the presence of a in the tuple space in the next t time units: if $t > 0$ and a is present in the tuple space then P is evaluated (rule **R4**). If $t > 0$ and a is not present then the check for a is repeated at the next time instant and the value of the counter t is decreased (axiom **R5**); Note that in this case we use the label σ , since a check on the store has been performed. As shown by axiom **R6**, the counter can be decreased also by performing a τ -action: Intuitively this rule is used to model the situation in which, even though the evaluation of the time-out started already, another (parallel) process has the access to the tuple space. In this case, differently from the approach in [13], time continues to elapse (via τ -actions) also for the time-out process, and for this reason we do not have the side condition $a \notin m$, since the elapsing of time does not depend on the check performed (see also the rules for the parallel operator). Axiom **R7** shows that if the time-out is exceeded, i.e. the counter t has reached the value of 0, then the process `rdp(a)t?P_Q` behaves as Q .

Rules **R8-R9** model the parallel composition operator in terms of *interleaving*, since only one basic σ -action is allowed for each transition (i.e. for each unit of time). This means that the access to the shared resource consisting of the global tuple space is granted to one process a time. However, time is processed by all the active parallel time-outs, as shown by rule **R8**, since τ -actions are allowed together with σ -actions. On the other hand, a parallel component is allowed to proceed in isolation if (and only if) the other parallel component cannot perform a τ -action (rule **R9**). To summarize, we adopt maximal parallelism for time elapsing (i.e. τ -actions) and an interleaving model for basic computation steps (i.e. σ -actions).

We have adopted this approach, different from that one in [13], because it seems more adequate to the nature of time-out operators not to interrupt the elapsing of time once the evaluation of a time-out has started. Clearly one could start the elapsing of time when the time-out process is scheduled, rather than when it appears in the top-level current parallel context. This modification could easily be obtained by adding a syntactic construct to differentiate active time-outs from inactive ones and by changing accordingly the transition system. One could also easily modify the semantics (both operational and denotational) to consider a more liberal assumption which allows multiple read actions in parallel. On the contrary, considering full maximal parallelism, i.e. allowing each enabled process to proceed, would cause several substantial differences.

R1	$\langle \text{out}(a).P, m \rangle \xrightarrow{\sigma} \langle P, m \uplus \{a\} \rangle$
R2	$\langle \text{in}(a).P, m \uplus \{a\} \rangle \xrightarrow{\sigma} \langle P, m \rangle$
R3	$\langle \text{rd}(a).P, m \uplus \{a\} \rangle \xrightarrow{\sigma} \langle P, m \uplus \{a\} \rangle$
R4	$\langle \text{rdp}(a)_t?P.Q, m \uplus \{a\} \rangle \xrightarrow{\sigma} \langle P, m \uplus \{a\} \rangle \quad t > 0$
R5	$\langle \text{rdp}(a)_t?P.Q, m \rangle \xrightarrow{\sigma} \langle \text{rdp}(a)_{t-1}?P.Q, m \rangle \quad t > 0 \text{ and } a \notin m$
R6	$\langle \text{rdp}(a)_t?P.Q, m \rangle \xrightarrow{\tau} \langle \text{rdp}(a)_{t-1}?P.Q, m \rangle \quad t > 0$
R7	$\langle \text{rdp}(a)_0?P.Q, m \rangle \xrightarrow{\sigma} \langle Q, m \rangle$
R8	$\frac{\langle P, m \rangle \xrightarrow{x} \langle P', m' \rangle \quad \langle Q, m \rangle \xrightarrow{\tau} \langle Q', m \rangle \quad x \in \{\sigma, \tau\}}{\langle P \parallel Q, m \rangle \xrightarrow{x} \langle P' \parallel Q', m' \rangle}$
R9	$\frac{\langle P, m \rangle \xrightarrow{x} \langle P', m' \rangle \quad \langle Q, m \rangle \not\xrightarrow{\tau} \quad x \in \{\sigma, \tau\}}{\langle P \parallel Q, m \rangle \xrightarrow{x} \langle P' \parallel Q, m' \rangle}$
R10	$\langle P + Q, m \rangle \xrightarrow{\sigma} \langle P, m \rangle$
R11	$\frac{\langle P, (m \downarrow a) \uplus d \rangle \xrightarrow{x} \langle Q, m' \rangle \quad x \in \{\sigma, \tau\}}{\langle P^d \setminus a, m \rangle \xrightarrow{x} \langle Q^{m' \uparrow a} \setminus a, (m' \downarrow a) \uplus (m \uparrow a) \rangle}$
R12	$\frac{\langle P[\text{rec}X.P/X], m \rangle \xrightarrow{x} \langle P', m' \rangle \quad x \in \{\sigma, \tau\}}{\langle \text{rec}X.P, m \rangle \xrightarrow{x} \langle P', m' \rangle}$

Table 1. The transition system for *T-Linda* (symmetric rules of rules R8,R9,R10 are omitted).

Rule **R10** defines the behavior of the choice operator. Note that we consider here a local choice operator and that, as previously mentioned, a choice is considered a σ -action.

As specified by rule **R11**, the process $P \setminus a$ behaves like P , with a considered *local* to P , i.e. the information on a provided by the external tuple space is hidden from P and, conversely, the information on a produced locally by P is hidden from external world. To describe locality in rule **R13** the syntax has been extended by a process $P^d \setminus a$ where d is a local tuple space of P containing information on a which is hidden in the external store. When the computation starts the local store is empty, i.e. $P \setminus a = P^\emptyset \setminus a$. In this rule we use also the following notation: Given a multiset m , we denote by $m \downarrow a$ the multiset obtained from m by deleting all the occurrences of the tuple a and we denote by $m \uparrow a$ the multiset consisting only of the occurrences in m of the tuple a .

Rule **R12** treats the case of a recursive definition in the usual way (recall that guarded recursion is assumed, i.e. a recursive call is always prefixed by an in, out, rd or rdp action). Even though we use a rule with a negative premise, the relation \longrightarrow described by the rules in Table 1 is well defined since the transition system it is strictly stratifiable (see [19]).

Using such a transition system we can then define the following notion of observables which considers the behavior of terminating computations, including the deadlocked ones. Other kind of observables (e.g. input/output pairs) can be obtained as simple abstractions of the ones considered here. Note that we consider only sequences of transition steps which do not involve τ -steps, as these do not correspond to actions on the store (also time-outs can perform σ -actions, see rules **R4** and **R5**).

Definition 2.2. (Observables)

Let P be a T-Linda process. We define

$$\mathcal{O}(P) = \{m_1 m_2 \dots m_n \mid \langle P, m_1 \rangle \xrightarrow{\sigma} \langle P_2, m_2 \rangle \xrightarrow{\sigma} \dots \xrightarrow{\sigma} \langle P_n, m_n \rangle \not\xrightarrow{\sigma}\}.$$

3. The denotational model

It is easy to see that the operational semantics which associates to a process P its observables $\mathcal{O}(P)$ is not compositional, since the observables of $P \parallel Q$ cannot be obtained by composing the observables of P and those of Q (by using a suitable composition operator). As a simple counter-example, consider the processes

$$P = \text{rdp}(a)_1 ? \text{out}(b) \cdot \text{loop}$$

and

$$Q = \text{rdp}(a)_2 ? \text{out}(b) \cdot \text{loop}$$

where loop is any non terminating process. Then

$$\mathcal{O}(P) = \mathcal{O}(Q)$$

holds, however we have that

$$\{c\} \{a, c\} \{a, c\} \{a, b, c\} \in \mathcal{O}(Q \parallel \text{out}(a))$$

while

$$\{c\}\{a, c\}\{a, c\}\{a, b, c\} \notin \mathcal{O}(P \parallel \text{out}(a))$$

which clearly means that there exists no operator \mathcal{C} such that $\mathcal{O}(P \parallel Q) = \mathcal{C}(\mathcal{O}(P), \mathcal{O}(Q))$.

In order to obtain a compositional characterization of $\mathcal{O}(P)$, which ensures that previous equation can be solved, in this section we define a denotational semantics which is obtained by using *timed reactive sequences* to represent *Linda* computations. These sequences are similar to those used in the semantics of dataflow languages [22], imperative languages [11], (timed) ccp [8, 4] and Linda [10]. However, differently from the previous cases, here consider triples rather than pairs, as σ -actions have to be distinguished from τ -actions. This difference, which accounts for the temporal features of T-Linda, leads to a different technical development. Our denotational model associates to a process a set of (timed) reactive sequences of the form

$$\langle m_1, m'_1, x_1 \rangle \cdots \langle m_n, m'_n, x_n \rangle \langle m, m, \sigma \rangle$$

where for any i , $1 \leq i \leq n$, m, m_i, m'_i are multisets and $x_i \in \{\sigma, \tau\}$. Any triple $\langle m_i, m'_i, x_i \rangle$ represents a computation step performed by a generic process at time i : Intuitively, the process transforms the tuple space from m_i to m'_i by performing a transition step labelled by x_i or, in other words, m_i is the assumption on the tuple space, x_i is the label of the performed step and m'_i is the contribution of the process itself. The last triple indicates that no further information can be produced by the process, thus pointing out that a “resting point” has been reached.

Actually this intuitive interpretation of sequences is not completely adequate. Indeed, the basic idea underlying the denotational model then is that, differently from the case of the operational semantics, inactive processes can always make a τ -step, where an inactive process is either a suspended one (due to the absence of the required tuple in the store) or a non scheduled component of a parallel construct. These additional τ -steps, which represent passing of time and are needed to obtain a compositional model in a simple way, are then added to denotations as triples of the form $\langle m, m, \tau \rangle$. For example, the denotation of the process $\text{out}(a)$ (we omit the continuation for simplicity) contains all the triples of the form $\langle m, m \uplus \{a\}, \sigma \rangle$ for any possible initial store m , as these represent the action of adding the tuple a to the current store. However, such a denotation contains also sequences where $\langle m, m \uplus \{a\}, \sigma \rangle$ is preceded by a finite sequence of triples of the form $\langle m, m, \tau \rangle$, where such a sequence represents the passing of time while the process is inactive (because some other parallel process is scheduled).

Before defining formally the denotational semantics we need to define the operators $\tilde{\text{out}}, \tilde{\text{in}}, \tilde{\text{rd}}, \tilde{\text{rdp}}, \tilde{\parallel}, \tilde{+}$ and $\tilde{\setminus}$ which act on sets of sequences and are the semantic counterparts of the syntactic constructs appearing in the language. Here and in the following the set of all reactive sequences is denoted by \mathcal{S} , with typical elements $s, s_1 \dots$, while sets of reactive sequences are denoted by $S, S_1 \dots$ and $\wp(\mathcal{S})$ denotes the set of subsets of \mathcal{S} . Furthermore, \cdot indicates the operator which concatenates sequences and given a sequence $s = \langle m_1, m'_1, x_1 \rangle \langle m_2, m'_2, x_2 \rangle \cdots \langle m_n, m'_n, \sigma \rangle$ we define $\text{first}(s) = m_1$ and $\text{result}(s) = m_n$. We also assume that functions (with range $\wp(\mathcal{S})$) are ordered by the point-wise extension of the ordering \subseteq .

Definition 3.1. Let S, S_1 and S_2 be sets of reactive sequences. Then we define the operators $\tilde{\text{out}}, \tilde{\text{in}}, \tilde{\text{rd}}, \tilde{\text{rdp}}, \tilde{\parallel}, \tilde{+}$ and $\tilde{\setminus}$ as follows:

The Out-Operator $\tilde{out} : \mathbb{T} \times \wp(\mathcal{S}) \rightarrow \wp(\mathcal{S})$ is the least function (w.r.t. the ordering induced by \subseteq) which satisfies the following equation

$$\tilde{out}(a, S) = \{s \in \mathcal{S} \mid s = \langle m, m \uplus \{a\}, \sigma \rangle \cdot s' \text{ and } s' \in S\} \cup \{s \in \mathcal{S} \mid s = \langle m, m, \tau \rangle \cdot s' \text{ and } s' \in \tilde{out}(a, S)\}.$$

The In-Operator $\tilde{in} : \mathbb{T} \times \wp(\mathcal{S}) \rightarrow \wp(\mathcal{S})$ is the least function which satisfies the following equation

$$\tilde{in}(a, S) = \{s \in \mathcal{S} \mid \text{either } s = \langle m \uplus \{a\}, m, \sigma \rangle \cdot s' \text{ and } s' \in S \text{ or } s = \langle m, m, \sigma \rangle \text{ and } a \notin m\} \cup \{s \in \mathcal{S} \mid s = \langle m, m, \tau \rangle \cdot s' \text{ and } s' \in \tilde{in}(a, S)\}.$$

The Rd-Operator $\tilde{rd} : \mathbb{T} \times \wp(\mathcal{S}) \rightarrow \wp(\mathcal{S})$ is the least function which satisfies the following equation

$$\tilde{rd}(a, S) = \{s \in \mathcal{S} \mid \text{either } s = \langle m \uplus \{a\}, m \uplus \{a\}, \sigma \rangle \cdot s' \text{ and } s' \in S \text{ or } s = \langle m, m, \sigma \rangle \text{ and } a \notin m\} \cup \{s \in \mathcal{S} \mid s = \langle m, m, \tau \rangle \cdot s' \text{ and } s' \in \tilde{rd}(a, S)\}.$$

The Rdp_t-Operator $\tilde{rdp}_t : \mathbb{T} \times \wp(\mathcal{S}) \times \wp(\mathcal{S}) \rightarrow \wp(\mathcal{S})$, with $t > 0$, is defined as:

$$\tilde{rdp}_t(a, S_1, S_2) = \{s \in \mathcal{S} \mid \text{either } s = \langle m \uplus \{a\}, m \uplus \{a\}, \sigma \rangle \cdot s' \text{ and } s' \in S_1 \text{ or } s = \langle m, m, \sigma \rangle \cdot s', a \notin m \text{ and } s' \in \tilde{rdp}_{t-1}(a, S_1, S_2)\} \cup \{s \in \mathcal{S} \mid s = \langle m, m, \tau \rangle \cdot s' \text{ and } s' \in \tilde{rdp}_{t-1}(a, S_1, S_2)\}.$$

The Rdp₀-Operator $\tilde{rdp}_0 : \mathbb{T} \times \wp(\mathcal{S}) \times \wp(\mathcal{S}) \rightarrow \wp(\mathcal{S})$ is the least function which satisfies the following equation

$$\tilde{rdp}_0(a, S_1, S_2) = \{s \in \mathcal{S} \mid \text{either } s = \langle m, m, \sigma \rangle \cdot s' \text{ and } s' \in S_2 \text{ or } s = \langle m, m, \tau \rangle \cdot s' \text{ and } s' \in \tilde{rdp}_0(a, S_1, S_2)\}.$$

Parallel Composition. Let $\tilde{\parallel} \in \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$ be the (commutative and associative) partial operator defined by induction on the length of the sequences as follows:

$$\begin{aligned} \langle m, m, \sigma \rangle \tilde{\parallel} \langle m, m, \sigma \rangle &= \langle m, m, \sigma \rangle \\ \langle m, m', x \rangle \cdot s \tilde{\parallel} \langle m, m, \tau \rangle \cdot s' &= \langle m, m, \tau \rangle \cdot s' \tilde{\parallel} \langle m, m', x \rangle \cdot s = \langle m, m', x \rangle \cdot (s \tilde{\parallel} s'), \end{aligned}$$

where $x \in \{\sigma, \tau\}$.

We define the operator $S_1 \tilde{\parallel} S_2$ on sets as the image of $\mathcal{S} \times \mathcal{S}$ under the above operator.

Choice. The semantic choice operator $\tilde{+} : \wp(\mathcal{S}) \times \wp(\mathcal{S}) \rightarrow \wp(\mathcal{S})$ is the least function which satisfies the following equation:

$$S_1 \tilde{+} S_2 = \{s \in \mathcal{S} \mid s = \langle m, m, \sigma \rangle \cdot s' \text{ and } s' \in S_1 \cup S_2\} \cup \{s \in \mathcal{S} \mid s = \langle m, m, \tau \rangle \cdot s' \text{ and } s' \in S_1 \tilde{+} S_2\}$$

Hiding Operator. We first need the notions of a -connected and a -invariant sequence. Given a sequence $s = \langle m_1, m'_1, x_1 \rangle \cdots \langle m_n, m_n, \sigma \rangle$, we denote by $s \downarrow a$ the sequence obtained from s by deleting each occurrence of the tuple a in the multisets m_1, m'_1, \dots, m_n . Namely

$$s \downarrow a = \langle m_1 \downarrow a, m'_1 \downarrow a, x_1 \rangle \cdots \langle m_n \downarrow a, m_n \downarrow a, \sigma \rangle.$$

Then we say that s is a -connected if

- $m_1 \downarrow a = m_1$ (that is, the input multiset of s does not contain any occurrence of the tuple a) and
- $m_i \uparrow a = m'_{i-1} \uparrow a$ for each $i \in [2, n]$ (that is, each input multiset m_i does not contain any information on a which has not been produced previously in the sequence by some m_j).

Moreover, the sequence s is a -invariant if

- for all computation steps $\langle m, m', x \rangle$ of s , $m' \uparrow a = m \uparrow a$ holds.

The semantic hiding operator then can be defined as follows:

$$\tilde{\downarrow}(a, S) = \{s \in \mathcal{S} \mid \text{there exists } s' \in S \text{ such that } s \downarrow a = s' \downarrow a, \\ s' \text{ is } a\text{-connected and } s \text{ is } a\text{-invariant}\}.$$

It is immediate to see that the previous semantic operators are well defined, that is, the least functions which satisfy the equations actually exist and can be obtained by a standard fix-point construction.

The \tilde{out} , \tilde{in} , \tilde{rd} and \tilde{rdp}_t operators have the expected definition, including the mentioned addition of τ -steps. In the definition of \tilde{in} and \tilde{rd} we also include (as postfixes) triples of the form $\langle m, m, \sigma \rangle$ which denote suspension. More generally, as previously mentioned, these indicate that no further information can be generated, i.e. that a final result of computation has been reached (notice that each process is terminated by stop, whose denotation contains sequences ending with a $\langle m, m, \sigma \rangle$ triple).

In the semantic parallel operator (acting on sequences) we require that at each point of time at most one σ -action is present and the two arguments of the operator agree with respect to the contribution of the environment (the first component of the triple). We also require that the two arguments have the same length (in all other cases the parallel composition is assumed being undefined): this is necessary to reflect the passage of time since the i -th element of any sequence corresponds to the given process action on the i -th time step. Even though we merge point-wise sequences of the same length, this models an interleaving approach for σ -actions, because of the previously mentioned addition of τ -steps to denotations.

Concerning the semantic choice operator, we include in the result all the sequences which belong to one of the two arguments, as this models local choice. We also add the stuttering steps, as usual.

In the hiding operator we say that a sequence is a -connected if no information on a is present in the input store which has not been already accumulated by the computation of the process itself. A sequence is a -invariant if its computation steps do not provide more information on a . These definitions are essentially the same as those used to model the notion of locality in [7].

E1	$\mathcal{D}[\text{stop}] = \{\langle m_1, m_1, \tau \rangle \langle m_2, m_2, \tau \rangle \cdots \langle m_n, m_n, \sigma \rangle \in \mathcal{S} \mid n \geq 1\}$
E2	$\mathcal{D}[\text{out}(a).P] = \tilde{o}ut(a, \mathcal{D}[P]).$
E3	$\mathcal{D}[\text{in}(a).P] = \tilde{i}n(a, \mathcal{D}[P]).$
E4	$\mathcal{D}[\text{rd}(a).P] = \tilde{r}d(a, \mathcal{D}[P]).$
E5	$\mathcal{D}[\text{rdp}(a)_t?P.Q] = \tilde{r}dpt(a, \mathcal{D}[P], \mathcal{D}[Q]).$
E6	$\mathcal{D}[A \parallel B] = \mathcal{D}[A] \tilde{\parallel} \mathcal{D}[B]$
E7	$\mathcal{D}[P_1 + P_2] = \mathcal{D}[P_1] \tilde{+} \mathcal{D}[P_2].$
E8	$\mathcal{D}[P \setminus a] = \tilde{\setminus}(a, \mathcal{D}[P]).$
E9	$\mathcal{D}[\text{rec}X.P] = \mathcal{D}[P[\text{rec}X.P/X]]$

Table 2. Denotational semantics of *T-Linda*

We can then define the denotational semantics \mathcal{D} as follows. Here *Processes* denotes the set of T-Linda processes.

Definition 3.2. The denotational semantics $\mathcal{D} : \text{Processes} \rightarrow \wp(\mathcal{S})$ is the least function, w.r.t. the ordering induced by \subseteq , which satisfies the equations in Table 2.

Also \mathcal{D} is well defined and can be obtained by a fix-point construction. To see this, let us define an interpretation as a mapping $I : \text{Processes} \rightarrow \wp(\mathcal{S})$. Then let us denote by \mathcal{I} the cpo of all the interpretations (with the ordering induced by \subseteq). We can then associate to the equations in Table 2 a monotonic (and continuous) mapping $\mathcal{F} : \mathcal{I} \rightarrow \mathcal{I}$ defined by the equations of Table 2, provided that we replace the symbol \mathcal{D} for $\mathcal{F}(I)$ and that we replace equation **E9** for the following one: $\mathcal{F}(I)(\text{recX.P}) = I(\text{P}[\text{recX.P}/\text{X}])$.

Then, one can easily prove that a function satisfies the equations in Table 2 iff it is a fix-point of the function \mathcal{F} . Because this function is continuous (on a cpo) well known results (e.g. see [15]) ensure us that its least fix-point exists and it equals \mathcal{F}^ω , where the powers are defined as follows: $\mathcal{F}^0 = I_0$ (this is the least interpretation which maps any process to the empty set); $\mathcal{F}^n = \mathcal{F}(\mathcal{F}^{n-1})$ and $\mathcal{F}^\omega = \text{lub}\{\mathcal{F}^n \mid n \geq 0\}$ (where lub is the least upper bound on the cpo \mathcal{I}).

4. Correctness

As for the correctness of the denotational semantics, we notice that some reactive sequences do not correspond to real computations: Clearly, when considering a real computation no further contribution from the environment is possible. This means that, at each step, the assumption on the current store must be equal to the store produced by the previous step. In other words, for any two consecutive steps $\langle m_i, m'_i, x_1 \rangle \langle m_{i+1}, m'_{i+1}, x_2 \rangle$ we must have $m'_i = m_{i+1}$. Furthermore, triples containing τ -actions do not correspond to observable computational steps, as these involve σ -actions only. So we are led to the following.

Definition 4.1. Let $s = \langle m_1, m'_1, x_1 \rangle \langle m_2, m'_2, x_2 \rangle \cdots \langle m_{n-1}, m'_{n-1}, x_{n-1} \rangle \langle m_n, m_n, \sigma \rangle$ be a reactive sequence. We say that s is connected if $m_i = m'_{i-1}$ for each i , $2 \leq i \leq n$, and $x_j = \sigma$ for each j , $1 \leq j \leq n-1$.

According to the previous definition, a sequence is connected if all the information assumed on the tuple space is produced by the process itself, apart from the initial input, and only σ -actions are involved. A connected sequence represents a *T-Linda* computation, as it will be proved in the remaining of this section.

In order to simplify the proof of the correctness for the denotational semantics, we introduce a new transition system T' , where inactive (either suspended or not scheduled) processes can perform τ -actions. When considering our notion of observables, we can prove that such a modified transition system is equivalent to the previous one and agrees with the denotational model. Of course, we could have defined the operational semantics directly in terms of the new transition systems T' . We have preferred not to do so because we believe that the transition system defined in Table 1 has a more direct and natural intuition than the one presented here.

R0'	$\langle \text{stop}, m \rangle \xrightarrow{\tau} \langle \text{stop}, m \rangle$
R1'	$\langle \text{out}(a).P, m \rangle \xrightarrow{\tau} \langle \text{out}(a).P, m \rangle$
R2'	$\langle \text{in}(a).P, m \rangle \xrightarrow{\tau} \langle \text{in}(a).P, m \rangle$
R3'	$\langle \text{rd}(a).P, m \rangle \xrightarrow{\tau} \langle \text{rd}(a).P, m \rangle$
R7'	$\langle \text{rdp}(a)_0?P_Q, m \rangle \xrightarrow{\tau} \langle \text{rdp}(a)_0?P_Q, m \rangle$
R10'	$\langle P + Q, m \rangle \xrightarrow{\tau} \langle P + Q, m \rangle$

Table 3. The τ -rules for the transition rule T' .

The new transition system T' is then obtained from the one in Table 1 by deleting rule **R9** and by adding the rules **R0'**, **R1'**, **R2'**, **R3'**, **R7'** and **R10'**, contained in Table 3. We denote by $\Rightarrow_{\subseteq} \text{Conf} \times \text{Labels} \times \text{Conf}$ the relation defined by T' (we assume that in the rules in Table 1 \rightarrow is replaced by \Rightarrow).

The observables induced by the transition system T' are formally defined as follows.

Definition 4.2. Let P be a T-Linda process. We define

$$\mathcal{O}'(P) = \{m_1 m_2 \dots m_n \mid \langle P, m_1 \rangle \xrightarrow{\sigma} \langle P_2, m_2 \rangle \xrightarrow{\sigma} \dots \xrightarrow{\sigma} \langle P_n, m_n \rangle \not\xrightarrow{\sigma}\}.$$

Lemma 4.3 shows that the modified transition system agrees with the original one when considering our notion of observables.

We first need some definitions and technical lemmata.

In the following we say that $P <_u Q$ if and only if Q is obtained from P by unfolding an occurrence of a recursive agent, which is not prefixed by an in, out, rd, rdp or rec action.

\ll_u denotes the reflexive and transitive closure of $<_u$.

The following lemmata hold.

Lemma 4.1. Let P, Q be T-Linda processes such that $P \ll_u Q$. Then for each store m and for $x \in \{\sigma, \tau\}$

$$\langle P, m \rangle \xrightarrow{x} \langle P', m' \rangle \text{ if and only if } \langle Q, m \rangle \xrightarrow{x} \langle P', m' \rangle.$$

Proof The proof is by induction on the number n of unfolding steps used to obtained Q from P .

$n = 0$) In this case $P = Q$ and then the thesis.

$n > 0$) In this case there exists a T-Linda process R such that R is obtained from P by using $n - 1$ unfolding steps and $R <_u Q$. By inductive hypothesis the following holds

$$\langle P, m \rangle \xrightarrow{x} \langle P', m' \rangle \text{ if and only if } \langle R, m \rangle \xrightarrow{x} \langle P', m' \rangle.$$

Now, to prove the thesis, we prove by structural induction on R that for each store m ,

$$\langle R, m \rangle \xrightarrow{x} \langle P', m' \rangle \text{ if and only if } \langle Q, m \rangle \xrightarrow{x} \langle P', m' \rangle.$$

Base Case. In this case $R = \text{rec}X.R_1$ and $Q = R_1[\text{rec}X.R_1/X]$. Then the proof is straightforward by observing that by Rule **R12** of Table 1, $\langle R, m \rangle \xrightarrow{x} \langle P', m' \rangle$ if and only if $\langle Q, m \rangle \xrightarrow{x} \langle P', m' \rangle$.

Inductive Cases.

Assume that $R = R_1 \parallel R_2$. Without loss of generality, we can assume that $Q = Q_1 \parallel R_2$ and $R_1 <_u Q_1$. By inductive hypothesis

$$\langle R_1, m \rangle \xrightarrow{x} \langle R'_1, m_1 \rangle \text{ if and only if } \langle Q_1, m \rangle \xrightarrow{x} \langle R'_1, m_1 \rangle.$$

Now the proof is straightforward by using either Rule **R8** or Rule **R9**.

If either $R = R_1 + R_2$ or $R = R_1^d \setminus a$ then the proof is analogous to the previous one and hence it is omitted. \square

From the above Lemma we derive the following corollary:

Corollary 4.1. Let P, Q be T-Linda processes such that $P \ll_u Q$. Then $\mathcal{O}(P) = \mathcal{O}(Q)$.

Lemma 4.2. Let P be a T-Linda process. Then for each store m ,

1. $\langle P, m \rangle \xrightarrow{\tau} \langle Q, m' \rangle$ if and only if $m = m'$ and
 - either $\langle P, m \rangle \xrightarrow{\tau} \langle R, m \rangle$ and $R \ll_u Q$
 - or $\langle P, m \rangle \not\xrightarrow{\tau}$ and $P \ll_u Q$.
2. $\langle P, m \rangle \xrightarrow{\sigma} \langle Q, m' \rangle$ if and only if $\langle P, m \rangle \xrightarrow{\sigma} \langle R, m' \rangle$ and $R \ll_u Q$.

Proof

1. The proof is by induction on the number n of the not-guarded occurrences of a recursive agent (of the form $\text{rec } X. P$) in P (note that only recursive calls need to be guarded according to our assumptions).

$n = 0$) The proof is by induction on the complexity of P .

- P is of the form $\text{stop}, \text{out}(a).P', \text{in}(a).P', \text{rd}(a).P', \text{rdp}(a)_0?P'_1Q'$ or $P' + Q'$.

The proof is immediate by observing that by the rules in Table 1, $\langle P, m \rangle \not\xrightarrow{\tau}$ and by the rules in Table 3, $\langle P, m \rangle \xrightarrow{\tau} \langle Q, m' \rangle$ if and only if $\langle P, m \rangle = \langle Q, m' \rangle$.

- P is of the form $\text{rdp}(a)_t.P'_1.Q'$, with $t > 0$.

The proof is immediate since both the transition systems use the rule **R6** of Table 1.

- If P is of the form $P_1 \parallel P_2$.

In this case, by definition of the transition system T' and by using rule **R8** of Table 1, for each store m ,

$$\begin{aligned} \langle P, m \rangle &\xrightarrow{\tau} \langle Q_1 \parallel Q_2, m' \rangle \text{ if and only if} \\ \langle P_1, m \rangle &\xrightarrow{\tau} \langle Q_1, m' \rangle \text{ and } \langle P_2, m \rangle \xrightarrow{\tau} \langle Q_2, m \rangle \end{aligned}$$

(the symmetric case is analogous and hence it is omitted).

By inductive hypothesis this holds if and only if $m' = m$ and for $i = 1, 2$

– either $\langle P_i, m \rangle \xrightarrow{\tau} \langle R_i, m \rangle$ and $R_i \ll_u Q_i$

– or $\langle P_i, m \rangle \not\xrightarrow{\tau}$ and $P_i \ll_u Q_i$.

If there exists $i \in [1, 2]$ such that $\langle P_i, m \rangle \xrightarrow{\tau} \langle R_i, m \rangle$ then the thesis follows by using either rule **R8** or rule **R9**.

Otherwise $\langle P_1 \parallel P_2, m \rangle \not\xrightarrow{\tau}$. Then the thesis follows since by the previous results $P_1 \parallel P_2 \ll_u Q_1 \parallel Q_2$.

- P is of the form $P'^d \setminus a$.

By rule **R11** of Table 1 for each store m ,

$$\begin{aligned} \langle P'^d \setminus a, m \rangle &\xrightarrow{\tau} \langle Q'^{m' \uparrow a} \setminus a, (m' \downarrow a) \uplus (m \uparrow a) \rangle \text{ if and only if} \\ \langle P', (m \downarrow a) \uplus d \rangle &\xrightarrow{\tau} \langle Q', m' \rangle. \end{aligned}$$

By inductive hypothesis this holds if and only if

$$m' = (m \downarrow a) \uplus d \tag{1}$$

and

– either $\langle P', (m \downarrow a) \uplus d \rangle \xrightarrow{\tau} \langle R', (m \downarrow a) \uplus d \rangle$ and $R' \ll_u Q'$

– or $\langle P', (m \downarrow a) \uplus d \rangle \not\xrightarrow{\tau}$ and $P' \ll_u Q'$.

By (1), we have that

$$(m' \downarrow a) \uplus (m \uparrow a) = m \text{ and } m' \uparrow a = d.$$

Therefore, by using rule **R11** of Table 1 and by definition of \ll_u ,

– either $\langle P'^d \setminus a, m \rangle \xrightarrow{\tau} \langle R'^{m' \uparrow a} \setminus a, m \rangle$ and $R'^{m' \uparrow a} \setminus a \ll_u Q'^{m' \uparrow a} \setminus a$

– or $\langle P'^d \setminus a, m \rangle \not\xrightarrow{\tau}$ and $P'^d \setminus a \ll_u Q'^{m' \uparrow a} \setminus a$

and then the thesis.

$n > 0$) The proof is by induction on the complexity of P.

If P is of the form $P' + Q'$, $P_1 \parallel P_2$ or $P'^d \setminus a$ then the proof is analogous to the previous one and hence it is omitted. Assume now that P is of the form $\text{recX}.P'$.

By rule **R12** of Table 1 for each store m ,

$$\langle \text{recX}.P', m \rangle \xrightarrow{\tau} \langle Q, m' \rangle \text{ if and only if } \langle P'[\text{recX}.P'/X], m \rangle \xrightarrow{\tau} \langle Q, m' \rangle.$$

Now observe that the number of the not-guarded occurrences of a recursive agent in $P'[\text{recX}.P'/X]$ is $n - 1$. Then by inductive hypothesis

$$\langle P'[\text{recX}.P'/X], m \rangle \xrightarrow{\tau} \langle Q, m' \rangle.$$

if and only if $m' = m$ and

- either $\langle P'[\text{recX}.P'/X], m \rangle \xrightarrow{\tau} \langle R, m \rangle$ and $R \ll_u Q$
- or $\langle P'[\text{recX}.P'/X], m \rangle \not\xrightarrow{\tau}$ and $P'[\text{recX}.P'/X] \ll_u Q$.

Therefore, by using rule **R12** of Table 1 and since, by definition of \ll_u , $P = \text{recX}.P' \ll_u P'[\text{recX}.P'/X]$,

- either $\langle P, m \rangle \xrightarrow{\tau} \langle R, m \rangle$ and $R \ll_u Q$
- or $\langle P, m \rangle \not\xrightarrow{\tau}$ and $P \ll_u Q$

and then the thesis.

2. The proof is analogous to the previous one and hence it is omitted. \square

Lemma 4.3. Let P be a T-Linda process. Then $\mathcal{O}(P) = \mathcal{O}'(P)$.

Proof Let $s = m_1 m_2 \dots m_n = m_1 \cdot s'$. We prove by induction on n that $s \in \mathcal{O}(P)$ if and only if $s \in \mathcal{O}'(P)$.

$n = 1$)

$$\begin{aligned} m_1 \in \mathcal{O}(P) & \text{ iff (by definition of } \mathcal{O}(P)) \\ \langle P, m_1 \rangle \not\xrightarrow{\sigma} & \text{ iff (by Point 2 of Lemma 4.2)} \\ \langle P, m_1 \rangle \not\xrightarrow{\sigma} & \text{ iff (by definition of } \mathcal{O}'(P)) \\ m_1 \in \mathcal{O}'(P). & \end{aligned}$$

$n > 1$)

$$\begin{aligned} s \in \mathcal{O}(P) & \text{ iff (by definition of } \mathcal{O}(P)) \\ \langle P, m_1 \rangle \xrightarrow{\sigma} \langle P_2, m_2 \rangle \text{ and } s' \in \mathcal{O}(P_2) & \text{ iff (by Point 2 of Lemma 4.2)} \\ \langle P, m_1 \rangle \xrightarrow{\sigma} \langle P'_2, m_2 \rangle, P_2 \ll_u P'_2 \text{ and } s' \in \mathcal{O}(P_2) & \text{ iff (by Corollary 4.1)} \\ \langle P, m_1 \rangle \xrightarrow{\sigma} \langle P'_2, m_2 \rangle, P_2 \ll_u P'_2 \text{ and } s' \in \mathcal{O}(P'_2) & \text{ iff (by inductive hypothesis)} \\ \langle P, m_1 \rangle \xrightarrow{\sigma} \langle P'_2, m_2 \rangle \text{ and } s' \in \mathcal{O}'(P'_2) & \text{ iff (by definition of } \mathcal{O}'(P)) \\ s \in \mathcal{O}'(P). & \end{aligned}$$

\square

We can now easily prove that, given our definition of $\mathcal{D}[\]$, the modified transition system T' agrees with the denotational model.

Theorem 4.1. For any process P we have

$$\mathcal{O}'(P) = \{s \mid \text{there exists a connected sequence } s \in \mathcal{D}[[P]] \text{ such that} \\ s = \langle m_1, m_2, \sigma \rangle \langle m_2, m_3, \sigma \rangle \cdots \langle m_n, m_n, \sigma \rangle\}.$$

Proof We prove by induction on the complexity of P that

$$\mathcal{D}[[P]] = \{s \mid s = \langle m_1, m'_1, x_1 \rangle \langle m_2, m'_2, x_2 \rangle \cdots \langle m_n, m_n, \sigma \rangle, P_1 = P, \\ \text{for } i \in [1, n-1], \langle P_i, m_i \rangle \xrightarrow{x_i} \langle P_{i+1}, m'_i \rangle \text{ and } \langle P_n, m_n \rangle \not\xrightarrow{\sigma}\}.$$

Then the proof follows by definition of $\mathcal{O}'(P)$.

When the process P is neither of the form $P_1 \setminus a$ nor of the form $P_1 \parallel Q_1$ the thesis follows immediately from the close correspondence between the rules of the transition system and the definition of the denotational semantics.

Assume that P is of the form $P_1 \setminus a$. By definition of the denotational semantics, $s \in \mathcal{D}[[P]]$ if and only if $s = \langle m_1, m'_1, x_1 \rangle \langle m_2, m'_2, x_2 \rangle \cdots \langle m_n, m_n, \sigma \rangle$ and there exists $s' \in \mathcal{D}[[P_1]]$,

$$s' = \langle k_1, k'_1, x_1 \rangle \langle k_2, k'_2, x_2 \rangle \cdots \langle k_n, k_n, \sigma \rangle$$

such that

$$m_i \downarrow a = k_i \downarrow a \text{ and } m'_i \downarrow a = k'_i \downarrow a, \text{ for each } i \in [1, n] \quad (2)$$

$$m_i \uparrow a = m'_i \uparrow a, \text{ for each } i \in [1, n] \quad (3)$$

$$k_1 \downarrow a = k_1 \text{ and } k_i \uparrow a = k'_{i-1} \uparrow a, \text{ for each } i \in [2, n]. \quad (4)$$

By inductive hypothesis $s' \in \mathcal{D}[[P_1]]$ if and only if

$$\langle P_i, k_i \rangle \xrightarrow{x_i} \langle P_{i+1}, k'_i \rangle \text{ for } i \in [1, n-1] \text{ and } \langle P_n, k_n \rangle \not\xrightarrow{\sigma}. \quad (5)$$

By Rule **R11** and by (2), we have that (5) holds if and only if

$$\langle P_i^{k_i \uparrow a}, m_i \rangle \xrightarrow{x_i} \langle P_{i+1}^{k'_i \uparrow a}, (m'_i \downarrow a) \uplus (m_i \uparrow a) \rangle \text{ for } i \in [1, n-1] \text{ and } \langle P_n^{k_n \uparrow a}, m_n \rangle \not\xrightarrow{\sigma}. \quad (6)$$

By (3), we have that (6) holds if and only if

$$\langle P_i^{k_i \uparrow a} \setminus a, m_i \rangle \xrightarrow{x_i} \langle P_{i+1}^{k'_i \uparrow a}, m'_i \rangle \text{ for } i \in [1, n-1] \text{ and } \langle P_n^{k_n \uparrow a} \setminus a, m_n \rangle \not\xrightarrow{\sigma}. \quad (7)$$

Now, observe that by (4), since $k_1 \uparrow a = \emptyset$, $P_1^{k_1 \uparrow a} \setminus a = P_1 \setminus a$ and $P_{i+1}^{k'_i \uparrow a} = P_{i+1}^{k_{i+1} \uparrow a}$. Then (7) holds if and only if

$$\langle P_1 \setminus a, m_1 \rangle \xrightarrow{x_1} \langle P_2^{k_2 \uparrow a}, m'_1 \rangle, \langle P_i^{k_i \uparrow a} \setminus a, m_i \rangle \xrightarrow{x_i} \langle P_{i+1}^{k_{i+1} \uparrow a}, m'_i \rangle \text{ for } i \in [2, n-1] \text{ and} \\ \langle P_n^{k_n \uparrow a} \setminus a, m_n \rangle \not\xrightarrow{\sigma}$$

and then the thesis.

Assume now that P is of the form $P_1 \parallel Q_1$. By definition of the denotational semantics, $s \in \mathcal{D}[[P]]$ if and only if $s = \langle m_1, m'_1, x_1 \rangle \langle m_2, m'_2, x_2 \rangle \cdots \langle m_n, m_n, \sigma \rangle$ and there exist $s' \in \mathcal{D}[[P_1]]$ and $s'' \in \mathcal{D}[[Q_1]]$, $s' = \langle m_1, k'_1, x'_1 \rangle \langle m_2, k'_2, x'_2 \rangle \cdots \langle m_n, m_n, \sigma \rangle$, $s'' = \langle m_1, k''_1, x''_1 \rangle \langle m_2, k''_2, x''_2 \rangle \cdots \langle m_n, m_n, \sigma \rangle$, such that for each $i \in [1, n - 1]$,

$$\begin{aligned} x_i = \tau \text{ if and only if } & x'_i = x''_i = \tau \text{ and in this case } m'_i = k'_i = k''_i = m_i, \\ x_i = \sigma \text{ if and only if } & \text{either } x'_i = \sigma, x''_i = \tau, k'_i = m'_i \text{ and } k''_i = m_i \\ & \text{or } x'_i = \tau, x''_i = \sigma, k''_i = m'_i \text{ and } k'_i = m_i. \end{aligned} \quad (8)$$

By inductive hypothesis $s' \in \mathcal{D}[[P_1]]$ and $s'' \in \mathcal{D}[[Q_1]]$ if and only if

$$\begin{aligned} \langle P_i, m_i \rangle & \xrightarrow{x'_i} \langle P_{i+1}, k'_i \rangle \text{ for } i \in [1, n - 1] \text{ and } \langle P_n, m_n \rangle \not\xrightarrow{\sigma}, \\ \langle Q_i, m_i \rangle & \xrightarrow{x''_i} \langle Q_{i+1}, k''_i \rangle \text{ for } i \in [1, n - 1] \text{ and } \langle Q_n, m_n \rangle \not\xrightarrow{\sigma}. \end{aligned} \quad (9)$$

Now, observe that by definition of \Rightarrow , $\langle P \parallel Q, m \rangle \not\xrightarrow{\sigma}$ if and only if $\langle P, m \rangle \not\xrightarrow{\sigma}$ and $\langle Q, m \rangle \not\xrightarrow{\sigma}$, for each T-Linda processes P and Q and store m .

Therefore, by Rule **R8** and by (8), we have that (9) holds if and only if

$$\langle P_i \parallel Q_i, m_i \rangle \xrightarrow{x_i} \langle P_{i+1} \parallel Q_{i+1}, m'_i \rangle \text{ for } i \in [1, n - 1] \text{ and } \langle P_n \parallel Q_n, m_n \rangle \not\xrightarrow{\sigma}$$

and then the thesis. \square

Thus we obtain the following correctness result, whose proof is immediate from the previous theorems.

Corollary 4.2. (Correctness)

For any process P we have

$$\mathcal{O}(P) = \{m_1 m_2 \dots m_n \mid \text{there exists a connected sequence } s \in \mathcal{D}[[P]] \text{ such that } s = \langle m_1, m_2, \sigma \rangle \langle m_2, m_3, \sigma \rangle \cdots \langle m_n, m_n, \sigma \rangle\}.$$

5. Related and future work

Even though the original proposal of the Linda coordination languages dates back to the eighties [18, 14], the definition of specific process calculi to reason formally about these languages is quite recent [12, 17]. In particular, concerning temporal aspects, the only other formal treatments we are aware of are in [13, 21]. In [13] the authors investigate the semantics of several coordination primitives (including time-outs) which appeared in some recent extensions of the Linda model, namely JavaSpaces [31] and TSpaces [33]. Time is also considered in [21] where several variants of a language inspired to the ESTEREL model are introduced and compared.

The main differences between our approach and those in [13, 21] is that we investigate a timed Linda language mainly from the denotational semantics perspective while in [13, 21] the authors focus on the

operational semantics. It is also worth mentioning that our interpretation of time and of the time-out construct is different from those appearing in the mentioned papers. We assume that once the elapsing of time for a time-out has started it cannot be interrupted, while this is not the case for the time-out considered in [13]. More precisely, similarly to what happens in some timed process algebras (e.g. [9]) actions in [13] are partitioned into instantaneous and time consuming ones and a time consuming action, such as the time-out, can be interleaved with an arbitrary number of out and in actions, which are considered instantaneous. On the other hand, we do not consider the out and in actions instantaneous, since these are the basic computational steps on the store, and if these action are interleaved with a time-out then elapsing of time is not interrupted. Also the approach followed in [21] partitions the basic actions into instantaneous and time consuming ones. Following the ESTEREL model, computation in [21] proceeds in “bursts of activity”, that is, it uses a two phases schema: In the first phase elementary actions are executed instantaneously. Then, when no process can be further reduced, time progresses by one unit. This approach, which is substantially different from ours, is similar to that one pursued by the timed cc language [26] and its non-deterministic extension (ntcc) developed in [25, 24].

Related to the present paper is also [4], where we investigated a timed ccp language. Clearly, there are relevant differences with [4], since the ccp paradigm does not allow removal of information from the global store and this simplifies considerably the semantic issues. Furthermore, maximal parallelism was assumed in [4] while here we consider an interleaving model. This is a relevant difference, which causes also a different definition of the time-out construct: In fact, under the maximal parallelism hypothesis one could define inductively the time-out construct in terms of the usual Linda operators as follows: $\text{rdp}(a)_1?P_Q = \text{rdp}(a)?P_Q$ and $\text{rdp}(a)_t?P_Q = \text{rdp}(a)?P_(\text{rdp}(a)_{t-1}?P_Q)$ for $t > 1$. On the other hand, using an interleaving model and assuming, as we do, that once the evaluation of the time-out has started the elapsing of time can not be interrupted, the previous definition does not work, as one can easily check by considering a process consisting of two parallel time-outs¹.

We already mentioned some other papers which use reactive sequences for defining the semantics of several languages [22, 11, 8, 4, 10]. The main difference with them is in the specific model of computation that we use, where interleaving and maximal parallelism coexist, which leads us to use σ and τ annotations and to a different technical development.

In the definition of T-Linda we made the simplifying assumption that basic actions take one time unit. We can easily relax this assumption by allowing actions of different (discrete) duration without major changes in the semantic treatment. Also, our denotational semantics (and temporal logic) could easily be adapted to the other timed Linda languages mentioned before.

We are currently studying the full abstraction problem for the semantics we have defined w.r.t. the input/output notion of observables: using a rather easy abstraction on denotations we can prove full abstraction for a modified language which allows the simultaneous addition, deletion and check for presence of several tuples. We believe that a modified abstraction should suffice also for the present language, even though the proof in this case is not simple (if one considers as observables “histories”, as in [10], rather than input/output pairs, then of course the full abstraction result is substantially simplified).

Future work includes also the development of a sound and complete axiomatization of the temporal logic

¹The point here is that one would need a rule of the form $\langle \text{rdp}(a)?P_Q, m \rangle \xrightarrow{\tau} \langle Q, m \rangle$ with $a \notin m$, however this contradicts our interpretation of τ -actions, since in this case a check on the store (to test whether $a \notin m$) is performed.

introduced in this paper and the study of temporal aspects in the context of languages for mobile computing based on the Linda paradigm, such as Klaim [16], since the implementations of these languages in some cases already include a notion of time-out.

Acknowledgements

We thank the reviewers of the paper for their precise comments.

References

- [1] L. Aceto and D. Murphy. Timing and causality in process algebra. *Acta Informatica*, 33(4):317–350, 1996.
- [2] J. Baeten and J. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.
- [3] G. Berry and G. Gonthier. The estereel programming language: Design, semantics and implementation. *Science of Computer Programming*, 19(2):87–152, 1992.
- [4] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Timed CCP Language. *Information and Computation*, 161(1):45–83, 2000.
- [5] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Timed Linda Language. In A. Porto and G. Roman, editors, *Proc. of 4th International Conference on Coordination Languages and Models, COORDINATION 2000*, volume 1906 of *Lecture Notes in Computer Science*, pages 299–304. Springer-Verlag, 2000.
- [6] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Denotational Semantics for a Timed Linda Language. In *Proc. of 3rd international ACM SIGPLAN conference on Principles and practice of declarative programming, PPDP 01*, pages 28–36. ACM Press, 2001.
- [7] F. S. de Boer, J. N. Kok, C. Palamidessi, and J. J. M. M. Rutten. On Blocks: locality and asynchronous communication. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Proc. of REX workshop on Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*, pages 73–90. Springer-Verlag, 1992.
- [8] F. S. de Boer and C. Palamidessi. A Fully Abstract Model for Concurrent Constraint Programming. In S. Abramsky and T. S. E. Maibaum, editors, *Proc. of TAPSOFT/CAAP*, volume 493 of *Lecture Notes in Computer Science*, pages 296–319. Springer-Verlag, 1991.
- [9] P. Bremond-Gregoire and I. Lee. A Process Algebra of Communicating Shared Resources with Dense Time and Priorities. *Theoretical Computer Science*, 189(1-2):169–219, 1997.
- [10] A. Brogi and J. M. Jacquet. Modeling Coordination via Asynchronous Communication. In D. Garlan and D. Le Métayer, editors, *Proc. of 2nd International Conference on Coordination Languages and Models, COORDINATION '97*, volume 1282 of *Lecture Notes in Computer Science*, pages 238–255. Springer-Verlag, 1997.
- [11] S. Brookes. A fully abstract semantics of a shared variable parallel language. In M. Y. Vardi, editor, *Proc. of 8th Annual IEEE Symposium on Logic In Computer Science*, pages 98–109. IEEE Computer Society Press, 1993.
- [12] N. Busi, R. Gorrieri, and G. Zavattaro. A Process Algebraic View of Linda Coordination Primitives. *Theoretical Computer Science*, 192(2):167–199, 1998.
- [13] N. Busi, R. Gorrieri, and G. Zavattaro. Process Calculi for Coordination: from Linda to JavaSpaces. In T. Rus, editor, *Proc. of 8th International Conference on Algebraic Methodology and Software Technology, AMAST 2000*, volume 1816 of *Lecture Notes in Computer Science*, pages 198–212. Springer-Verlag, 2000.

- [14] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, 1989.
- [15] B.A. Davey and H.A. Priestley. Introduction to Lattices and Order. Cambridge University Press, 1990.
- [16] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. In *IEEE Transactions on Software Engineering*, volume 24, pages 315–330, 1998.
- [17] R. De Nicola and R. Pugliese. Linda-based Applicative and Imperative Process Algebras. *Theoretical Computer Science*, 238(1-2):389–437, 2000.
- [18] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems, TOPLAS*, 7(1):80–112, 1985.
- [19] J. F. Groote. Transition System Specifications with Negative Premises. *Theoretical Computer Science*, 118(2):263–299, 1993.
- [20] M. Hennessy and T. Regan. A temporal process algebra. *Information and Computation*, 117:221–239, 1995.
- [21] J. M. Jacquet, K. De Bosschere, and A. Brogi. On Timed Coordination Languages. In A. Porto and G. Roman, editors, *Proc. of 4th International Conference on Coordination Languages and Models*, volume 1906 of *Lecture Notes in Computer Science*, pages 81–98. Springer-Verlag, 2000.
- [22] B. Jonsson. A model and a proof system for asynchronous processes. In *Proc. of the 4th ACM Symp. on Principles of Distributed Computing*, pages 49–58. ACM Press, 1985.
- [23] R. Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.
- [24] M. Nielsen, C. Palamidessi, F. D. Valencia. Temporal Concurrent Constraint Programming: Denotation, Logic and Applications. *Nordic Journal of Computing*, 9(2):145–188, 2002.
- [25] C. Palamidessi, F. D. Valencia. A Temporal Concurrent Constraint Programming Calculus. In *Proc. of CP 01*. Springer-Verlag, LNCS 2239, 2001.
- [26] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of Timed Concurrent Constraint Programming. In S. Abramsky, editor, *Proc. of the Ninth IEEE Symposium on Logic in Computer Science, LICS 1994*, pages 71–80. IEEE Computer Press, 1994.
- [27] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Timed Default Concurrent Constraint Programming. *Journal of Symbolic Computation*, 22(5-6):475–520, 1996.
- [28] V. A. Saraswat and M. Rinard. Concurrent constraint programming. In *Proc. of 17th Annual ACM Symposium on Principles of Programming Languages, POPL 1990*, pages 232–245. ACM Press, 1990.
- [29] V. A. Saraswat, M. Rinard, and P. Panangaden. Semantics foundations of Concurrent Constraint Programming. In *Proc. of 18th Annual ACM Symposium on Principles of Programming Languages, POPL 1991*, pages 333–352. ACM Press, 1991.
- [30] G. Smolka. The Definition of Kernel Oz. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*, pages 251–292. Springer-Verlag, 1995.
- [31] Sun Microsystem, Inc. *JavaSpaces Specifications*. 1998.
- [32] F. D. Valencia. Reactive Constraint Programming. Technical report, Brics Progress Report, June 2000.
- [33] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. TSpaces. *IBM Systems Journal*, 37(3):454–474, 1998.