On the expressiveness of probabilistic and prioritized data-retrieval in Linda ⋆

Mario Bravetti, Roberto Gorrieri, Roberto Lucchi and Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy.
E-mail: {bravetti, gorrieri, lucchi, zavattar}@cs.unibo.it

Abstract

Linda tuple-spaces coordination model does not allow to express a preference of tuples. In many applications we could be interested in indicating tuples that should be returned more frequently w.r.t. other ones, or even tuples with a low relevance that should be taken under consideration only if there is no tuple with a higher importance. We present an extension of the tuple-space model with quantitative information that permit to express such forms of preference. More precisely, we consider tuples decorated with a quantitative label. Such labels will be considered with two different semantics, one modeling probabilistic distribution of data retrieval and the other modeling priorities of tuples. Finally, we report all the results concerning the expressiveness gap between the standard model and the proposed extensions. We show that by adding probabilities the leader election problem can be solved. More surprisingly, the addition of priorities makes the model Turing complete, while we prove that this is not the case for the other two calculi.

Key words: Tuple-space coordination model, probability, priority.

1 Introduction

The tuple-space coordination model introduced by Linda is based on a shared tuple-space that processes can use to coordinate their activities. Processes can send data by inserting tuples (ordered sequences of data) with the *out* primitive; to retrieve data from the space (TS for short) processes can use the *rd* (nonconsming read) and the *in* (consuming input) primitives that use a template to denote the kind of tuples they are interested in. According to

⋆ Extended abstract of the full paper 'Adding Quantitative Information to Tuple Space Coordination Languages'. M. Bravetti, R. Gorrieri, R. Lucchi and G. Zavattaro. Draft. Available at http://www.cs.unibo.it/%7Elucchi/pubbl.html

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the semantics of the model, if more than one tuple could be retrieved, it is selected non deterministically.

In some applications, we may be interested in expressing more sophisticated policies for selecting the tuple to be returned, for example, according to some priority based access (one tuple should be returned only if no other tuples of higher priority are currently available) or a probabilistic selection (one tuple should be returned with a higher probability w.r.t. another one). Such policies are based on *global* properties, that is all the tuple matching the template should be taken into account to select the one to be returned. More precisely, the data-retrieval operation should select the tuple to be returned according to some function (either probabilistic or priority-based) that has all the matching tuples in its domain. On the other hand, the semantics of the tuple-space model is based on the matching rule which is a *local* property. Consequently, in order to program such policies we need to deal with *global* operations. Some proposals are available in literature such as the *collect* primitive of [13] or the non blocking *inp* operation supported in some Linda system [8]. We motivate, by using some examples, that these primitives can be used to program such policies, but in a rather unsatisfactory manner.

We present an extension of the tuple space model where *quantitative information* can be associated to a tuple with the aim of indicating its relevance or importance. Such labes will be considered in the data-retrieval operation with two possible meaning: they can represent the *priority* or to denote the *weight* of tuples.

According to the priority approach, the choice of the tuple to be returned follows the following rule: *a tuple can be returned only if no matching tuple with a highest priority is actually available.* In other terms, the tuple is non-deterministically selected among those with the higher priority. In the second approach, the selection occurs according to a probabilistic distribution that follows this principle: *the greater is the weight of a tuple, the higher is the probability for that tuple to be returned.*

Finally, we investigate whether the addition of the quantitative information strictly increases the expressiveness of the tuple space model or, on the contrary, the quantitative labels can be simulated in the native Linda-like model. In order to answer to this question, we proceed as follows. We define a process calculus, called `LinCa` (Linda Calculus), that models the standard tuple space coordination model. After, we extend the syntax adding the possibility to associate quantitative labels to tuples; we equip the new syntax with two different semantics thus obtaining two calculi, `PrioLinCa` in which the quantitative labels denote priorities and `ProbLinCa` in which they represent weights.

These calculi permit to formally prove gap of expressiveness between the considered extensions and the native tuple space model. The gap of expressiveness between `LinCa` and `PrioLinCa` is rather surprising; the addition of a minimum structure of priority, comprising only two priority levels *high* and *low*, is sufficient to increase the expressive power of the calculus from Turing incompleteness to Turing completeness.

On the other hand, the gap of expressiveness we prove between `LinCa` and `ProbLinCa` follows a more traditional discriminating technique between nondetermistic and probabilistic behaviour in distributed systems with asynchronous communication. Namely, we consider a typical problem in the area of distributed algorithms, the *leader election problem* (see, e.g., [1]), and we show that it can be solved for symmetric networks only under the extended probabilistic model while this is not the case for the standard tuple space model.

The paper is structured as follows. In Section 2 we present a formal description of the standard tuple space model. In Section 3 we extend the syntax introducing the quantitative labels, we provide the prioritized and the probabilistic semantics. In Section 4 we formalize the expressiveness gap between the standard model and the proposed extensions. Finally, Section 5 reports some conclusive remarks.

2 `LinCa`

Here we introduce `LinCa`, the Linda Calculus. We consider the three standard primitives that any Linda-like language provides: `out(e)`, `in(t)` and `rd(t)`. The `out(e)` primitive inserts a tuple `e` in the tuple space. Primitive `in(t)` is the blocking input operation: when an occurrence of a tuple `e` matching with `t` (denoting a template) is found in the TS, it is removed from the TS and the primitive returns the tuple. The read primitive `rd(t)` is the blocking read operation that, differently from `in(t)`, returns the matching tuple `e` without removing it from the TS.

The tuples are ordered and finite sequences of typed fields, while templates are ordered and finite sequences of fields that can be either *actual* or *formal* (see [8]): a field is actual if it specifies a type and a value, while it is formal if the type only is given. For the sake of simplicity, in the formalization of Linda we are going to present, fields are not typed.

Formally, let $Mess$, ranged over by m, m', $\ldots$, be a denumerable set of messages and Var, ranged over by x, y, $\ldots$, be the set of data variables. In the following, we use $\vec{x}$, $\vec{y}$, $\ldots$, to denote finite sequences $x_1; x_2; \ldots; x_n$ of variables.

Tuples, denoted by e, e', $\ldots$, are finite and ordered sequences of data fields (we use $arity(e)$ to denote the number of fields of e), while templates, denoted by t, t', $\ldots$, are finite and ordered sequences of fields that can be either data or wildcards (used to match with any message).

Formally, tuples are defined as follows:

$$e = \langle \vec{d} \rangle$$

where $\vec{d}$ is a finite and ordered sequence of data fields, separated by ‘;’, that

can be either messages or variables.
The definition of template follows:

$$t = \langle \vec{dt} \rangle$$

where $\vec{dt}$ is a finite and ordered sequence of terms that can be messages, variables or *null*. The additional value *null* denotes the wildcard, whose meaning is the same of formal fields of Linda, i.e. it matches with any field value. In the following, the set $Tuple$ (resp. $Template$) denotes the set of tuples (resp. templates) containing no variable.

The matching rule between tuples and templates we consider is the classical one of Linda, whose definition is as follows.

Definition 2.1 Matching rule - Let $e = \langle d_1; d_2; \ldots; d_n \rangle \in Tuple$ be a tuple, $t = \langle dt_1; dt_2; \ldots; dt_m \rangle \in Template$ be a template; we say that *e matches t* (denoted by $e \triangleright t$) if the following conditions hold:

(i) $m = n$.

(ii) $dt_i = d_i$ or $dt_i = null$, $1 \leq i \leq n$.

Condition 1. checks if e and t have the same arity, whilst 2. tests if each non-wildcard field of t is equal to the corresponding field of e.

Processes, denoted by P, Q, ..., are defined as follows:

$$P ::= \mathbf{0} \mid out(e).P \mid rd\ t(\vec{x}).P \mid in\ t(\vec{x}).P \mid P|P\ |!in\ t(\vec{x}).P$$

A process can be a terminated program $\mathbf{0}$ (that we usually omit), a prefix form $\mu.P$, the parallel composition of two programs, or the replication of a program. The prefix μ can be one of the following coordination primitives: i) *out* (e), that writes the tuple e in the TS; ii) $rd\ t(\vec{x})$, that given a template t reads a matching tuple e in the TS and stores the return value in $\vec{x}$; iii) $in\ t(\vec{x})$, that given a template t consumes a matching tuple e in the TS and stores the return value in $\vec{x}$. In both the $rd\ t(\vec{x})$ and $in\ t(\vec{x})$ operations $(\vec{x})$ is a binder for the variables in $\vec{x}$. The parallel composition $P \mid Q$ of two processes P and Q behaves as two processes running in parallel. Infinite behaviours can be expressed using the replication operator $!in\ t(\vec{x}).P$. Replication is a typical operator used in process calculi to denote the parallel composition of an unbounded amount of instances of the same process. In our calculus we restrict the application of replication to input guarded processes. This is justified by the fact that replicated output operation (resp. read operations), may give rise to an undesired behaviour by producing (reading) a tuple an unbounded amount of time.

In the following, $P[d/x]$ denotes the process that behaves as P in which all occurrences of x are replaced by d. We also use $P[\vec{d}/\vec{x}]$ to denote the process obtained by replacing in P all occurrences of variables in $\vec{x}$ with the corresponding value in $\vec{d}$, i.e. $P[d_1; d_2; \ldots; d_n/x_1; x_2; \ldots; x_n] = P[d_1/x_1][d_2/x_2] \ldots [d_n/x_n]$.

We say that a process is *well formed* if each prefix operation of kind $rd/in\ \langle\vec{dt}\rangle(\vec{x})$ is such that the variables $\vec{x}$ and the data $\vec{dt}$ have the same arity. Notice that in the $rd\ t(\vec{x})$ and $in\ t(\vec{x})$ operations we use a notation which is different from the standard Linda notation: we explicitly indicate in $(\vec{x})$ the variables that will be bound to the actual fields of the matching tuple, while in the standard Linda notation these variables are part of the template. Observe that the two notations are equivalent up to the fact that our notation introduces variables also in association to the formal fields of the template. In the following, we consider only processes that are well formed; $Process$ denotes the set of such processes.

Let $DSpace$, ranged over by DS, DS', ..., be the set of possible configurations of the TS, that is $DSpace = \mathcal{M}_{fin}(Tuple)$, where $\mathcal{M}_{fin}(S)$ denotes the set of all the possible finite multisets on S. In the following, we use $DS(e)$ to denote the number of occurrences of e within $DS \in DSpace$.The set $System = \{[P, DS] \mid P \in Process, DS \in DSpace\}$, ranged over by s, s', ..., denotes the possible configurations of systems.

The semantics we use to describe processes interacting via coordination primitives is defined in terms of a transition system $(System, \longrightarrow)$, where $\rightarrow \subseteq System \times System$. More precisely, $\longrightarrow$ is the minimal relation satisfying the axioms and rules of Table 1 (symmetric rule of (4) is omitted). $(s, s') \in \longrightarrow$ (also denoted by $s \longrightarrow s'$) means that a system s can evolve (performing a single action) in the system s'. Finally, we use $s \longrightarrow^+ s'$ (resp. $s \longrightarrow^* s'$) for the transitive (resp. reflexive and transitive) closure of $\longrightarrow$. A computation $s_1 \longrightarrow s_2 \longrightarrow \ldots \longrightarrow s_n$ is maximal if there exists no s' such that $s_n \longrightarrow s'$. We say that a process *terminates* if it has a finite maximal computation.

Axiom (1) describes the output operation that produces a new occurrence of the tuple e in the shared space DS ($DS \oplus e$ denotes the multiset obtained by DS increasing by 1 the number of occurrences of e). Rules (2) and (3) describe the in and the rd operations, respectively: if a matching tuple e is currently available in the space, it is returned at the process invoking the operation and, in the case of in, it is also removed from the space ($DS - e$ denotes the removal of an occurrence of e from the multiset DS). Rule (4) represents a local computation of processes, whilst (5) the replication operator that produces a new instance of the process and copies itself.

2.1 Examples

Here, we report some simple examples with the aim of introducing the reader to the notation of `LinCa`. We present a master-worker application and a service registry. In the following sections where probabilistic and prioritized data-retrieval are modeled, we show how to add features to such examples.

The master-worker application is composed of masters and workers which coordinate via a tuple space: the masters produce job requests and store them inside the tuple space, and the workers access the tuple space to retrieve the

$$(1) \quad [out\ (e).P, DS] \longrightarrow [P, DS \oplus e]$$

$$(2) \quad \frac{\exists e \in DS : e \rhd t}{[in\ t(\vec{x}).P, DS] \longrightarrow [P[e/\vec{x}], DS - e]}$$

$$(3) \quad \frac{\exists e \in DS : e \rhd t}{[rd\ t(\vec{x}).P, DS] \longrightarrow [P[e/\vec{x}], DS]}$$

$$(4) \quad \frac{[P, DS] \longrightarrow [P', DS']}{[P \mid Q, DS] \longrightarrow [P' \mid Q, DS']}$$

$$(5) \quad \frac{[in\ t(\vec{x}).P, DS] \longrightarrow [P', DS']}{[!in\ t(\vec{x}).P, DS] \longrightarrow [P' \mid !in\ t(\vec{x}).P, DS']}$$

Table 1
Semantics of `LinCa`. Symmetric rule of (4) omitted.

description of the jobs to execute.

The second example is concerned with the problem of coordinating the collaboration among Web-Services; in particular, consider the problem of discovering a Web-Service willing to offer a particular service. A tuple space could be exploited in this scenario as a registry where the available Web-Services register the kind of services they intend to offer, while the clients access the tuple space in order to discover the actual Web-Services availability.

Example 2.2 Master-worker - The idea is that masters request a job supplied by workers by inserting a tuple containing the job and workers select jobs by retrieving such tuples from the space. Let $job \in Mess$ be a job; the procedures $submit(job)$ and $supply\_job$, which submits and supplies a job, respectively, are defined as follows:

$$submit(job) \triangleq out(\langle job \rangle)$$

$$supply\_job \triangleq in\ \langle null \rangle(x).Supply(x)$$

where $Supply(x)$ is the process performing the job x. It is clear that the workers select submitted jobs in a non-deterministic way, thus preventing to manage different urgency levels of jobs.

We could assume that jobs have different urgency levels, and that the workers must select a job for execution only if no jobs are currently registered with a higher priority. In Section 3.1 we discuss how this new feature can be easily programmed by exploiting priorities on tuples.

Example 2.3 Service registry - In this case the tuple space is used as a services registry. The registration of a new service consists in inserting a new tuple containing the service information. To discover services, processes can perform read operations. Let $s \in Mess$ and $pl \in Mess$ be a task and a link to a service, respectively. The procedure $register(s, pl)$, which registers a service supplying task s that is available at link pl, is defined as follows:

$$register(s, pl) \stackrel{\Delta}{=} out(\langle s; pl \rangle)$$

while the procedure $discover(s)$, which discovers a service supplying task s, is defined as follows:

$$discover(s) \stackrel{\Delta}{=} rd\ \langle s; null \rangle (x_1; x_2)$$

where at the end of the computation x_2 will contain the link to a service supplying task s.

It is rather clear that when more than one service supplying task$s$ is available, the link obtained in the discovery phase is non-deterministically chosen. This is not satisfactory if we intend, e.g., to distribute in a balanced way the workload, thus to avoid the overwhelming of requests towards one Web-Service while leaving other Services under-utilized. In Section 3.2 we discuss how to manage such distribution by exploiting probabilistic data-retrieval.

3 Quantitative labels

As mentioned in the Introduction, we extend the syntax of `LinCa` by adding quantitative labels that may have two different meanings, thus obtaining two calculi with the same syntax but different semantics.

Quantitative labels are used to decorate tuples. Formally, let $QLab$, ranged over by l, l', …, be the set of the possible quantitative labels. Tuples are now defined as follows:

$$e = \langle \vec{d} \rangle\,[l]$$

where $l \in QLab$ and $\vec{d}$ is defined as above. We also define $\tilde{\ }$ as the function that, given a tuple e, returns its sequence of data fields (e.g. if $e = \langle \vec{d} \rangle\,[l]$ then $\tilde{e} = \vec{d}$). In the following, we denote with QL the function that, given a tuple, returns its quantitative label (e.g., if $e = \langle \vec{d} \rangle\,[l]$ then $QL(e) = l$). Quantitative labels are not considered in the matching rule whose definition is unchanged.

In the first calculus we present, called `PrioLinCa`, quantitative labels are interpreted as priorities, while in the second, called `ProbLinCa`, they are interpreted as weights defining the probabilistic distribution of data retrieval (the greater is the weight of a tuple, the higher is the probability for that tuple to be retrieved).

3.1 `PrioLinCa`

In this section we present `PrioLinCa`, the extension of `LinCa` supporting prioritized data-retrieval. The quantitative labels on tuples will be considered as their priority level. This approach is similar to the one adopted in [3,2]. We assume that there exists a total order relation on $QLab$. For the sake of simplicity, we usually consider that $QLab$ coincides with $\mathbb{N} \setminus \{0\}$.

The `PrioLinCa` semantics differs from the one of `LinCa` in the rd and in primitives where a matching tuple is returned only if no other matching one with higher priority is available. The two rules are reported in Table 2.

$$(2') \quad \frac{\exists e \in DS : e \triangleright t \quad \forall e' \in DS : e' \triangleright t \quad QL(e) \geq QL(e')}{[in\ t(\vec{x}).P, DS] \longrightarrow [P[\tilde{e}/\vec{x}], DS - e]}$$

$$(3') \quad \frac{\exists e \in DS : e \triangleright t \quad \forall e' \in DS : e' \triangleright t \quad QL(e) \geq QL(e')}{[rd\ t(\vec{x}).P, DS] \longrightarrow [P[\tilde{e}/\vec{x}], DS]}$$

Table 2
Semantics of `PrioLinCa`. Rules (1), (4) and (5) of Table 1 omitted.

Example 3.1 Master-worker with job priorities - We extend the Example 2.2 by allowing a classification between *critical* and *standard* jobs. Critical jobs must be executed as soon as a free worker is available. We can program such a system using '*critical*' and '*standard*' as symbolic names representing quantitative labels, and interpreting *critical* as a priority higher than *standard*. Thus, the two procedures for job submissions become:
$submitCritical(job) \triangleq out(\langle job\rangle[critical])$
$submitStandard(job) \triangleq out(\langle job\rangle[standard])$.
In this way, when a worker performs the *supply_job* procedure it is ensured that no standard jobs are served in case at least one critical job has been submitted.

The priority-based access policy could be programmed by using the non blocking *inp* operation supported in some Linda system [8], that returns one tuple matching the template, if available, or terminates indicating the absence of matching tuples. The level of priority could be associated to the tuples as an extra field. In order to perform a prioritized data-retrieval primitive, processes could initially perform an *inp* taking into account the first level of priority, and passing to the subsequent levels only if no tuples are retrieved. This solution is satisfactory only if few levels of priority are considered, because it is necessary to explicitly access one level at a time.

3.2 `ProbLinCa`

In this section we introduce `ProbLinCa`, which extends `LinCa` with the probabilistic retrieval of data in the repository. Quantitative labels in this case represent the *weight* of tuples; here we let $QLab$ range over positive (non-zero) real numbers. Informally, the weight of a tuple in the TS represents its *appealing degree*: among the entries in the DS that can be read/removed by an agent, the tuple with greatest weight has the highest probability to be read/removed by the agent. It is worth noting that the Linda model accepts multiple instances of the same tuple, therefore the probability to access a specific tuple depends also on the weights associated with the several instances of each matching tuple. We consider the approach we use to express probabilistic accesses on tuples, used also in [2,3], as the best way in such a calculus. Indeed, due to the matching rule supporting wildcards, it is not convenient to define a single probability distribution over all tuples (generative approach of [15]). For more details see the full version of this paper [4].

The `ProbLinCa` semantics is defined as a probabilistic transition system, whose formalization is available in the full version of the paper. Here we informally report the semantics of primitives in `ProbLinCa`:

- $out(e)$, where $e \in Tuple$ is unchanged w.r.t. standard Linda. The tuple e is inserted in the DS.
- $in\ \langle t\rangle(\vec{x})$, where $t \in Template$; if some tuple e matching the template t is available in the DS, the execution of in causes the removal of one of such tuples e from the space and returns $\tilde{e}$. The probability of removing a particular tuple $e = \langle\vec{d}\rangle[w] \in DS$ with e that matches t is the ratio of w to the sum of the weights w' in the tuples $e' = \langle\vec{d'}\rangle[w']$ in the DS such that e' matches with t (taking into account multiple occurrences of tuples).
- $rd\ t(\vec{x})$, where $t \in Template$; if some tuple e matching the template t is available in the DS, one of such tuples is read and the returned value is $\tilde{e}$. The probability of reading a particular tuple e with e that matches t is evaluated as in the input case.
  Note that this means that the probability of reading a particular matching sequence of data fields $\langle\vec{d}\rangle$ contained in the DS is the ratio of the sum of weights w associated with the several instances of $\langle\vec{d}\rangle$ contained in the DS, to the sum of the weights of the tuples e' in the DS matching with t.

As mentioned in the Introduction, we could exploit global operation to program probabilistic accesses to tuples. Consider, e.g., the *collect* primitive of [13], that permits to withdraw all the tuples satisfying the template. If we want to force a specific probabilistic distribution of the returned tuple, we could decorate each tuple adding (as an extra field) a value that quantifies the level of relevance of the tuple. When a data-retrieval operation is executed from an agent, this agent could *collect* all the tuples satisfying the template, select the tuple according to the distribution of these values, and re-introduce the tuples in the space. Clearly, this pattern is not satisfactory because it requires to move from the tuple space to the agent (and back) possibly huge quantities of tuples, and moreover this complex operation should be executed in a transactional manner, thus requiring consistent locks.

Using the probabilistic approach, we can satisfactorily solve the problem of a balanced distribution of the workload of the Web-Services: each Web-Service indicates with a weight its current workload. When a client performs its discovery operation, a link to a Web-Service currently unloaded is more probably returned w.r.t. an overwhelmed one.

Example 3.2 Service registry with workload distribution - We extend the Example 2.3 by programming a discovery service which supports a balanced workload distribution. Weights are used to express the current workload of services: the higher is the workload, the lower is the weight. Thus, the registration procedure must now take into account the workload of the service, represented by a weight w:

$$register(s, pl, w) \triangleq out(\langle s; pl\rangle[w])$$

In this way services with the lowest workload have the highest probability to be discovered, thus obtaining a workload distribution accordingly with the probability distribution on the accesses to the tuples.

4 Expressiveness hierarchy

The master-worker application with job priorities and the discovery service supporting balanced workload have informally shown the expressiveness lacks of `LinCa` in programming such coordination patterns. In this section we mention the main results obtained in [4] about the formalization of the expressiveness gap between `LinCa` and the proposed extensions `PrioLinCa` and `ProbLinCa`. We have shown that:

- `LinCa` **is not a Turing powerful formalism -** We show that in `LinCa` termination is decidable by defining an encoding of `LinCa` systems into finite Place/Transition nets that preserves the existence of a finite computation. Then, to prove such assertion, we exploit the fact that the deadlock problem is decidable in finite P/T nets [12].
- `PrioLinCa` **is a Turing powerful formalism -** We show that `PrioLinCa` is expressive enough for encoding any Random Access Machine (RAM) [14], that is a Turing complete formalism. In particular, the encoding we propose exploits only two different priority levels; this means that simply by partitioning the space in two classes, one containing the tuples with 'low' priority and the other one the tuples with 'high' priority, we can cover that expressiveness gap with Turing formalisms. The technique we use follows some idea proposed in [7].

- **The Leader Election Problem cannot be solved neither in `LinCa` and `PrioLinCa` -** We prove that the Leader Election Problem cannot be solved in `LinCa` when we have symmetric systems. The same proof is suitable to prove the same result for `PrioLinCa`. In the context of process calculi, in particular referring to the π-calculus, this technique has been already used in [9] to prove a similar gap of expressiveness.
- **The Leader Election Problem can be solved in `ProbLinCa` -** The probabilistic extension, as happens also in other languages, makes it possible to solve the leader election protocol. We have presented a protocol for symmetric systems, following some idea of [10], and proved that the leader can be elected with a probability equal to 1.
- **Reachability is decidable in `ProbLinCa` -** The finite P/T net used to model `LinCa` systems can be easily adapted to map the probabilistic data-retrieval. The reachability problem, which consists in verifying whether there exists a computation from a given state to another one, is decidable in such P/T nets.

Leader election has been exploited also in [10] to prove the impossibility to provide a *uniform encoding* from the synchronous to the asynchronous π-calculus that preserves any *reasonable semantics*. Informally, an encoding is uniform when it is an homomorphism w.r.t. parallel composition and preserves name substitution; a semantics is reasonable if it distinguishes two processes, say P and Q, whenever in some computation of P the relevant observable actions are different from those of any computation of Q. This impossibility result holds also between `ProbLinCa` (the calculus with probabilities) and `LinCa` (the calculus representing the standard tuple space model).

From the expressiveness gap between `LinCa` and `PrioLinCa` follows that there exists no computable encoding from `PrioLinCa` to `LinCa` that preserves any reasonable semantics. This follows from the fact that a reasonable semantics is able to discriminate between a faithful encoding of RAMs and a nonfaithful encoding. It is enough to consider RAMs that make observable their termination and by formalizing faithfulness as follows: all the computations of the encoding of a RAM R terminate if and only if R terminates. Such an encoding cannot be provided in `LinCa` as we prove that termination is decidable.

Since in `PrioLinCa` it is not possible to solve the leader election problem, that can be solved in `ProbLinCa`, it is not possible to provide a uniform encodings from the model with priority to the model with probability. Moreover, since reachability cannot be decided in `PrioLinCa`, differently from `ProbLinCa` where it is decidable, there exist no computable encoding from `PrioLinCa` to `ProbLinCa`. This is a direct consequence of the fact that a reasonable semantics distinguishes between two processes that have different sets of reachable states.

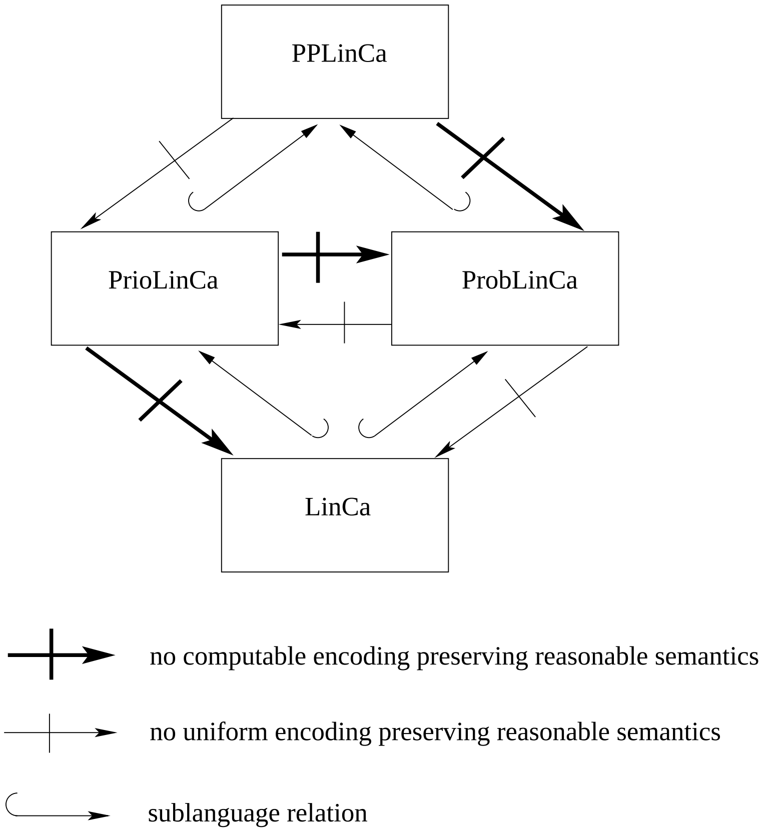

Fig. 1. The lattice of the languages.

From the latter consideration, concerning the incomparability result between `PrioLinCa` and `ProbLinCa`, we can conclude that the language supporting both probabilistic and prioritized data-retrieval has an expressiveness strictly higher than the single proposals. We call such model `PPLinCa`. Such model can be easily defined by extending the syntax: tuples now must contain two quantitative labels, one representing the priority and the other one the weight. The data-retrieval semantics is defined in such a way that selects, according with the probability distribution, one matching tuple among those with the highest priority.

Figure 1 summarizes the formalization of the expressiveness gap between the model and the proposed extensions.

5 Conclusion

In this work we have presented the extension of the standard tuple-space coordination model supporting quantitative labels that are used to express both priorities and probabilistic accesses on tuples. Some of such calculi, `ProbLinCa` and `PPLinCa`, have been originally presented in [5]. Finally, we have formalized the expressiveness gap between such models.

To the best of out knowledge, the only related work is probabilistic Klaim [11], that proposes two forms of probabilities: one at the *local* level (processes scheduling) and the other one at the *network* level (probability on the allocation functions which map logical localities into physical locations). As future work, it could be interesting investigate whether schedule-driven and data-driven probabilities are equivalent, that is they have the same expressive power.

Finally, in [6] the probabilistic extensions has been exploited to program a registry for Web-Services supporting: registration, deregistration and update of Web-Service information, and the discovery of Web-Services guaranteeing a balanced workload distribution.

References

[1] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* John Wiley and Sons, Inc., 2004. Second Edition.

[2] M. Bravetti. *Specification and Analysis of Stochastic Real-Time Systems.* PhD thesis, Dottorato di Ricerca in Informatica. Università di Bologna, Padova, Venezia, February 2002. Available at `http://www.cs.unibo.it/~bravetti/`.

[3] M. Bravetti and M. Bernardo. Compositional asymmetric cooperations for process algebras with probabilities, priorities, and time. In *Proc. of the 1st Int. Workshop on Models for Time-Critical Systems, MTCS 2000,* State College (PA), volume 39(3) of *Electronic Notes in Theoretical Computer Science.* Elsevier, 2000.

[4] Mario Bravetti, Roberto Gorrieri, Roberto Lucchi, and Gianluigi Zavattaro. Adding Quantitative Information to Tuple Space Coordination Languages. Draft. Available at `http://www.cs.unibo.it/~lucchi/pubbl.html`.

[5] Mario Bravetti, Roberto Gorrieri, Roberto Lucchi, and Gianluigi Zavattaro. Probabilistic and Prioritized Data Retrieval in the Linda Coordination Model. In *Proc. of 7th International Conference on Coordination Models and Languages (Coordination 04)*, volume 2949 of *LNCS*, pages 55–70. Springer Verlag, 2004.

[6] Mario Bravetti, Roberto Gorrieri, Roberto Lucchi, and Gianluigi Zavattaro. Web Services for E-commerce: guaranteeing security access and quality of service. In *Proc. of ACM Symposium on Applied Computing (SAC'04)*, pages 800–806. ACM Press, 2004.

[7] Nadia Busi, Roberto Gorrieri, and Gianluigi Zavattaro. On the Expressiveness of Linda Coordination Primitives. *Information and Computation*, 156(1-2):90–121, January 2000.

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

[9] Oltea Mihaela Herescu and Catuscia Palamidessi. Probabilistic asynchronous pi-calculus. In *Proc. of FoSSaCS*, pages 146–160, 2000.

[10] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In *Proc. of POPL*, pages 256–265, 1997.

[11] A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic Klaim. In *Proc. of 7th International Conference on Coordination Models and Languages (Coordination 04)*, volume 2949 of *LNCS*, pages 119–134. Springer Verlag, 2004.

[12] C. Reutenauer. *The Mathematics of Petri Nets.* Masson, 1988.

[13] A. Rowstron and A. Wood. Solving the Linda multiple `rd` problem using the `copy-collect` primitive. *Science of Computer Programming*, 31(2-3):335–358, 1998.

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

[15] R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, Generative and Stratified Models of Probabilistic Processes. *Information and Computation*, 121:59–80, 1995.