Probabilistic Linda-based Coordination Languages

Alessandra Di Pierro[1], Chris Hankin[2], and Herbert Wiklicky[2]

[1] Dipartimento di Informatica, Universitá di Pisa, Italy
[2] Department of Computing, Imperial College London, UK

Abstract. Coordination languages are intended to simplify the development of complex software systems by separating the coordination aspects of an application from its computation aspects. Coordination refers to the ways the independent active pieces of a program (e.g. a process, a task, a thread, etc.) communicate and synchronise with each other. We review various approaches to introducing probabilistic or stochastic features in coordination languages. The main objective of such a study is to develop a semantic basis for a quantitative analysis of systems of interconnected or interacting components, which allows us to address not only the functional (qualitative) aspects of a system behaviour but also its non-functional aspects, typically considered in the realm of performance modelling and evaluation.

1 Introduction

An early example of a Coordination Language was Linda [1]; Gelernter and Carriero offer the following equation [2]:

$$\text{Concurrent Programming} = \text{Computation} + \text{Coordination}$$

The intention being that Coordination Languages are "glue" languages for controlling the various computational components of a concurrent program.

Linda is an example of a *Shared Data Space* coordination language – the glue is provided by interaction through a shared tuple space. Alternative ways of synchronising components could be Message Passing, as in the Manifold model [3], or broadcast (cf Sands use of the CBS calculus [4]).

In this paper we consider various ways in which probabilities/quantities can be added to this basic paradigm; we distinguish between a data-driven and a schedule-driven approach. We also consider ways in which mobility can be added.

Our main objective is to develop a semantic basis for a quantitative analysis of networks. A quantitative analysis allows in general for the consideration of more "realistic" situations. For example, a probabilistic analysis allows for establishing the security of a system up to a given tolerance factor expressing how much the system is actually vulnerable. This is in contrast to a qualitative analysis which typically might be used to validate the absolute security of a given

system. In a distributed environment quantitative analysis is also of a great practical use in the consideration of timing issues which involve the asynchronous communications among processes running with different clocks.

The rest of this paper is structured into three main parts. First we consider how probabilities/quantities might be added to a Linda-like language. We consider a data-driven approach as presented in [5]; we also consider a schedule driven approach where probabilities/quantities are explicitly attached to parallel operators. We then consider a slightly more complicated language which includes mobility by introducing located processes and distributing the tuple space. For this part we will survey the approaches in [6–8]. We consider two alternatives: one in which nodes are visited with a certain probability (the discrete case) and the other in which nodes are visited at a certain rate (the continuous case). Finally, we consider an analysis framework for studying the properties of networks of processes; we briefly consider two variants which correspond to the discrete and continuous case respectively.

2 Linda

Linda [1] is a coordination language that relies on an asynchronous and associative communication mechanism based on a shared global space called the Tuple Space (TS), consisting of a multiset of tuples. The Linda language provides four simple operations for manipulating tuples by introducing, removing and reading tuples from the space TS. These operations allow processes to communicate and synchronise by interacting with the Tuple Space.

In order to investigate the introduction of quantitative information in the Linda paradigm, [5] introduces a minimal language, called Linda Calculus or LinCa, which includes the Linda core calculus expressed via only three constructs: prefix, parallel composition and replication. The syntax is presented in Table 1. **nil** represents the inactive process. The $\mathbf{out}(e)$ action causes a tuple to be deposited into the tuple space. The $\mathbf{in}\ t\ (\boldsymbol{x})$ and $\mathbf{read}\ t\ (\boldsymbol{x})$ actions both input tuples from the tuple space; the tuple is required to match the pattern t and the fields of the matching tuple are bound to the components of $\boldsymbol{x}$. The **in** action removes one matching tuple from the tuple space whereas **read** is non-destructive.

The original Linda language also includes an action **eval** which is similar to $\mathbf{out}(e)$, but capable of creating processes: for each tuple element which is a function, this primitive creates a new process to evaluate the function. Once all functions have been evaluated, **eval** will place the resulting tuple into the tuple space.

The semantics of LinCa is shown in Table 2. This is a small-step operational semantics. The configurations or states consist of a process and a tuple space, TS. The tuple space is essentially a multiset; $\oplus$ represents multiset union and $-$ represents multiset subtraction. We use $\rhd$ to represent pattern matching; the exact details of how a tuple matches a pattern need not concern us in this paper. We have omitted the symmetric rule for parallel composition. Finally, we use the

$P ::=$	$\mathbf{nil}$	null process
	$\mathbf{out}(e).P$	output prefix
	$\mathbf{in}\ t\ (\boldsymbol{x}).P$	input prefix
	$\mathbf{read}\ t\ (\boldsymbol{x}).P$	non-destructive input prefix
	$P \mid P$	parallelism
	$!\mathbf{in}\ t\ (\boldsymbol{x}).P$	replication

Table 1. The LinCa syntax

$$[\mathbf{out}(e).P, TS] \longrightarrow [P, TS \oplus e]$$

$$\frac{\exists e \in TS : e \rhd t}{[\mathbf{in}\ t\ (\boldsymbol{x}).P, TS] \longrightarrow [P[e/\boldsymbol{x}], TS - e]}$$

$$\frac{\exists e \in TS : e \rhd t}{[\mathbf{read}\ t\ (\boldsymbol{x}).P, TS] \longrightarrow [P[e/\boldsymbol{x}], TS]}$$

$$\frac{[P, TS] \longrightarrow [P', TS']}{[P \mid Q, TS] \longrightarrow [P' \mid Q, TS']}$$

$$\frac{[\mathbf{in}\ t\ (\boldsymbol{x}).P, TS] \longrightarrow [P', TS']}{[!\mathbf{in}\ t\ (\boldsymbol{x}).p, TS] \longrightarrow [P' \mid !\mathbf{in}\ t\ (\boldsymbol{x}).P, TS]}$$

Table 2. LinCa semantics

notation $P[e/\boldsymbol{x}]$ to represent the process P where all instances of the components of $\boldsymbol{x}$ (i.e. $x_1, \ldots, x_n$) have been replaced by corresponding "values" from e.

2.1 Adding Probabilities/Quantities

In the recent literature three proposals have been presented aimed at extending the basic coordination model à la Linda with quantities (probabilities, priorities, rates). These proposals follow two main approaches:

- Data Driven: In this approach the quantitative information is added to the data (tuples); it is adopted in [5] to define two quantitative versions of the core Linda language LinCa called PrioLinCa and ProbLinCa respectively (cf. Section 2.2). The main objective of [5] is the investigation of the expressive power of the different quantitative extensions compared to the basic paradigm.
- Schedule Driven: In this approach quantitative information is added to the "processes"; this is the approach taken in pKLAIM [6, 7] and StocKLAIM

[8], where the motivation is more analysis-oriented. We will describe these two proposals in Section 3.2 and Section 3.3 in the context of a coordination language which extends Linda with distributed programming and mobility features.
We will also adopt this approach to define in Section 2.3 alternative versions of the prioritised and probabilistic LinCa introduced in [5].

2.2 Data Driven Approach

The starting point of the approach in [5] is the observation that nondeterminism is inherent in the definition of the Linda primitives. It occurs when a tuple becomes available on which more than one **in** t $(\boldsymbol{x})$ or **read** t $(\boldsymbol{x})$ action were suspended, or similarly when there is more than one tuple matching $\boldsymbol{x}$ in a **in** t $(\boldsymbol{x})$ or **read** t $(\boldsymbol{x})$ operation. This nondeterminism can be controlled by labelling tuples with quantities that can be interpreted respectively as priority or probability. For the resulting models, called PrioLinCa and ProbLinCa, Bravetti et al. have shown the following results:

- LinCa is not Turing complete (termination is decidable)
- PrioLinCa is Turing complete (encoding of RAM)
- ProbLinCa can solve the Leader Election problem; neither LinCa nor PrioLinCa can.

PrioLinCa. In PrioLinCa priorities (positive natural numbers) are added as attributes of tuples. The significant change in the language is in the semantics of **in** and **read**. For example the rule for **in** becomes:

$$\frac{\exists e \in TS : e \rhd t \;\; \forall e' \in TS : e' \rhd t \Rightarrow prio(e) \geq prio(e')}{[\mathbf{in}\ t\ (\boldsymbol{x}).P, TS] \longrightarrow [P[e/\boldsymbol{x}], TS - e],}$$

where $prio(e)$ denotes the quantity labelling the tuple e. A matching tuple is only removed from the tuple space if its priority is higher than any other matching tuple.

Rather than use priorities, it is possible to take a probabilistic approach which is exemplified by the alternative calculus called ProbLinCa.

ProbLinCa. In ProbLinCa weights (positive real numbers) are added as attributes of tuples. A tuple is then selected with a probability which is proportional to its weight. This is reflected in the semantics of the language by defining transition rules which probabilistically determine the next state according to a distribution which depend on the basic action of the starting state. Thus the rule for **in** becomes:

$$\frac{\exists e \in TS : e \rhd t}{[\mathbf{in}\ t\ (\boldsymbol{x}).P, TS] \longrightarrow \rho}$$

where ρ is a distribution on states which is computed as follows:

$$\rho(s) = \begin{cases} \dfrac{weight(e) \cdot TS(e)}{\sum_{e' \in TS : e' \rhd t} weight(e') \cdot TS(e')} & \text{if } s = [P[e/\boldsymbol{x}], TS - e], \\ & e \rhd t, e \in TS \\ 0 & \text{otherwise.} \end{cases}$$

where $TS(e)$ is the number of occurrences of the tuple e in TS. The probability of picking a particular tuple is thus computed in the following way:

- if the tuple, e, matches the pattern, t, the probability is the weight of the tuple times the number of occurrences of that tuple, normalised by the sum of the weights times multiplicities of all matching tuples.
- otherwise, when e doesn't match t, the probability is zero.

The rule for the **read** action determines an analogous distribution, while the rule for **out**(e) deterministically (i.e. with probability 1) leads to the state where the tuple e is added to the space independently of its weight:

$$[\mathbf{out}(e).P,\, TS] \longrightarrow \rho,$$

where $\rho([P,\, TS \oplus e]) = 1$ and $\rho(s) = 0$ for all the other states.

Finally, the rule for the parallel composition $P_1 \mid P_2$ nondeterministically chooses among the probability distributions determined by the transition rules for P_1 and P_2.

2.3 Schedule Driven Approach

In this section we propose alternative quantitative extensions of LinCa by adopting a schedule driven approach. In this approach we add priorities or probabilities to the operators – in particular, to the parallel operator.

A useful notion for the definition of a prioritised or probabilistic scheduler is the notion of "active state" which identifies those processes (essentially those prefixed by a **in** t $(\boldsymbol{x})$ or **read** t $(\boldsymbol{x})$ action) that are able to make a transition (essentially are not blocked awaiting for a tuple to become available).

Definition 1. We define the set *Active* of *active* states as

$$\begin{aligned} \mathit{Active} = {} & \{[P, TS] \mid P \equiv \mathbf{in}\ t\ (\boldsymbol{x}).P' \text{ and } \exists e \in TS : e \rhd t\} \\ & \cup \{[P, TS] \mid P \equiv \mathbf{read}\ t\ (\boldsymbol{x}).P' \text{ and } \exists e \in TS : e \rhd t\} \\ & \cup \{[P, TS] \mid P \equiv \mathbf{out}(e).P'\}. \end{aligned}$$

Prioritised Scheduling. We replace the LinCa parallel composition $P \mid P$ by the prioritised parallel operator $p_1 : P_1 \mid p_2 : P_2$ where p_1 and p_2 are numbers (e.g. positive natural numbers as in PrioLinCa) expressing some priorities. A prioritised scheduler will (non-deterministically) select the state with higher priority among the active ones. Thus the semantics of the prioritised parallel operator can be defined by the rule:

$$\frac{[P_1,\, TS] \longrightarrow [P_1',\, TS'] \text{ and } p_1 \geq p_2}{[p_1 : P_1 \mid p_2 : P_2,\, TS] \longrightarrow [p_1 : P_1' \mid p_2 : P_2,\, TS']}$$

and the symmetric rule with P_2 in the premise.

From this semantics we can retrieve the data driven semantics of PrioLinCa. To see this, define the weight $\mathit{weight}(s)$ of a state $s = [P, TS]$ as the maximal weight of a matching tuple for P if s is active; $\mathit{weight}(s) = 0$ otherwise. Then assume that in the previous rule, priorities p_1 and p_2 are defined respectively as $\mathit{weight}([P_1,\, TS])$ and $\mathit{weight}([P_2,\, TS])$.

$$[\mathbf{out}(e).P,\, TS] \longrightarrow_1 [P,\, TS \oplus e]$$

$$\frac{\exists e \in TS : e \rhd t}{[\mathbf{in}\ t\ (\boldsymbol{x}).P,\, TS] \longrightarrow_1 [P[e/\boldsymbol{x}],\, TS - e]}$$

$$\frac{\exists e \in TS : e \rhd t}{[\mathbf{read}\ t\ (\boldsymbol{x}).P,\, TS] \longrightarrow_1 [P[e/\boldsymbol{x}],\, TS]}$$

$$\frac{[P_i,\, TS] \longrightarrow_p [P_i',\, TS']}{[|_{j=1}^{n}\ p_j : P_j,\, TS] \longrightarrow_{p \cdot \tilde{p_i}} [p_i : P_i' \mid |_{j=1, j\neq i}^{n}\ p_j : P_j,\, TS']}$$

$$\frac{[P[e/\boldsymbol{x}],\, TS] \longrightarrow_p [P',\, TS']}{[A(e),\, TS] \longrightarrow_p [P',\, TS']} \qquad \text{if } A(\boldsymbol{x}) \equiv P$$

Table 3. Schedule Driven Probabilistic Semantics

Probabilistic Scheduling. A probabilistic LinCa can be defined in the schedule-driven approach by replacing the parallel operator $P \mid P$ in the syntax of LinCa by a probabilistic one $p_1 : P_1 \mid p_2 : P_2$, where p_1 and p_2 are probabilities, that is real numbers in $[0, 1]$. Alternatively, we can let p_1 and p_2 range over the interval $[0, \infty)$: the normalisation process occurring at run-time guarantees that our quantities will indeed be transformed into probabilities.

The semantics of this alternative probabilistic Linda language can be defined in the usual SOS style via a probabilistic transition system $(S, \longrightarrow_p)$, where the the parameter p in the transition relation $\longrightarrow_p$ on states specifies the probability of a single step transition from one state to another. The rules defining $\longrightarrow_p$ are given in Table 3.

For the probabilistic parallel composition, in line with our previous work we opted for a more convenient n-ary version rather than the binary version used in Linda. In this rule the probability p_i is normalised to take account of the fact that the other branches of the parallel operator might be blocked. More precisely, we define the cumulative probability, $C_{[P,\,TS]}$, of all active processes in a parallel composition $P = |_{j=1}^{n}\ p_j : P_j$ as

$$C_{[P,\,TS]} = \sum_j \{p_j \mid [P_j,\, TS] \in \mathit{Active}\}.$$

Then the normalised probability $\tilde{p}_i$ is given by $\frac{p_i}{C_{[P,\,TS]}}$ if at least one of the two processes in P is active, and zero otherwise.

Replication introduces a new parallel operator; we must add probabilities to this:

$$\frac{[\mathbf{in}\ t\ (\boldsymbol{x}).P,\, TS] \longrightarrow_q [P',\, TS']}{[!\mathbf{in}\ t\ (\boldsymbol{x}).P,\, TS] \longrightarrow_q [p : P' \mid (1-p) :!\mathbf{in}\ t\ (\boldsymbol{x}).P,\, TS]}$$

This raises an issue about the choice of a value for p. This could be avoided by adding named processes and recursion rather than replication. We therefore introduce process *constants*, ranged over by A, and recursive definitions of the

form $A(\boldsymbol{x}) \equiv P$. The transition rule for recursion simply models the execution of a call to a procedure named A.

As a comparison with the probabilistic semantics for ProbLinCa defined in [5], we observe that the probabilistic model at the base of our semantics is *generative* according to the classification introduced in [9]: at each step the scheduler can select the next state according to one single probability distribution over the states. In the data driven semantics the probabilistic transition system conforms to the *reactive* model of probability instead: the scheduler can choose among different distributions depending on the (out/in/read) action taken. As a consequence our semantics contains strictly more information than the data driven semantics.

3 Distributed Tuple Spaces: KLAIM

The original Linda primitives are not completely adequate for programming distributed systems composed of mobile components. The KLAIM language (Kernel Language for Agents Interaction and mobility) was introduced in [10] as a distributed mobile version of Linda which extends the Linda interaction model by replacing the single shared tuple space with multiple distributed tuples spaces and allowing for explicit manipulation of localities and locality names.

3.1 A Core KLAIM Calculus

As before we will identify a simple core language where we consider only the basic constructs for prefixing, parallel composition and recursion. Moreover, we restrict to the actions **out**, **in** and **read** excluding the other KLAIM primitives, namely $\mathbf{eval}(P)@\ell$ which allows for a remote evaluation of the process argument[1], and $\mathbf{newloc}(u)$ which creates a new location accessible via the locality variable u. We also omit the consideration of allocation environments, that is partial functions used in the the full KLAIM language for the linking of symbolic names to physical addresses of nodes. The syntax of this minimal language, which we call cKLAIM, is given in Table 4.

The idea is to 'localise' processes P and their tuple spaces at some sites s and to construct networks N out of such nodes. We assume that locations are unique, i.e. only one process is attached to each location. The primitive actions **out**, **in** and **read** must now specify the local tuple space they refer to; this is done by introducing in their syntax the suffix "$@\ell$".

The operational semantics of cKLAIM is a restriction of the semantics of full KLAIM as presented in [10]. This is a two levels semantics: there are rules describing local transitions $P \xrightarrow{action} P'$ which are labelled with some (possible) action, and a global network semantics $N \rightarrowtail N'$ which specifies how a whole network evolves. The transition relation $\rightarrowtail$ is defined in terms of the local semantics $\longrightarrow$.

[1] This is different from the operation $\mathbf{eval}(t)$ in Linda whose argument is a tuple.

$P ::=$	$\mathbf{nil}$	null process
	$\mathbf{out}(e)@\ell.P$	output prefix
	$\mathbf{in}\ t\ (\boldsymbol{x})@\ell.P$	input prefix
	$\mathbf{read}\ t\ (\boldsymbol{x})@\ell.P$	non-destructive input prefix
	$P \mid P$	parallelism
	$!\mathbf{in}\ t\ (\boldsymbol{x})@\ell.P$	replication
$N ::=$	$s :: P$	node
	$N_1 \parallel N_2$	composition

Table 4. cKLAIM Process and Network Syntax

$N ::= s ::^q P$	node
$N_1 \parallel N_2$	composition

$N ::= s ::^\lambda P$	node
$N_1 \parallel N_2$	composition

Table 5. Discrete and Continuous Time Network Syntax

3.2 Probabilistic KLAIM

Our main motivation for adding probabilities/quantities to coordination languages is to support quantitative analysis of distributed systems. The techniques that we have developed, based on Discrete or Continuous Time Markov Chains, provide a strong link between program analysis and recent advances in Performance Analysis [11]. A primary application of our work is in the study of quantitative aspects in Language Based Security, relative to e.g. denial of service, viruses, epidemiology, etc.

We will consider here only a probabilistic version of cKLAIM, and omit a prioritised one. Based on the two layered semantics of cKLAIM we will introduce probabilities both on the local and the global level. Locally we introduce probabilities into the parallel construct (scheduling information); globally, we introduce two different versions: in one we associate a probability with each node, while in the other we associate a rate. As a result we obtain two probabilistic extensions of KLAIM according to a discrete time and a continuous time Markov chain model respectively.

The only changes to the syntax are thus, as before, the introduction of a probabilistic parallel composition with scheduling probabilities p (in the discrete time model) or scheduling rates λ (in the continuous time model) for nodes. These changes are depicted in Table 5 with p and λ positive real numbers.

Local Semantics Local transitions are labelled with an *action* label and are of the form:

$$[P, TS] \xrightarrow{action}_p [P', TS].$$

$$[\mathbf{out}(t)@\ell.P, TS] \xrightarrow{o(t)@\ell}_1 [P, TS]$$

$$[\mathbf{in}(t)@\ell.P, TS] \xrightarrow{i(t)@\ell}_1 [P, TS]$$

$$[\mathbf{read}(t)@\ell.P, TS] \xrightarrow{r(t)@\ell}_1 [P, TS]$$

$$\frac{[P_j, TS] \xrightarrow{\mu}_p [P'_j, TS']}{[|_{i=1}^n p_i : P_i, TS] \xrightarrow{\mu}_{p \cdot p_j} [\ |_{j \neq i=1}^n P_i \mid P'_j, TS']}$$

$$\frac{[P[e/x], TS] \xrightarrow{\mu}_p [P', TS']}{[A(e), TS] \xrightarrow{\mu}_p [P', TS']} \quad \text{with } A(x) \equiv P$$

Table 6. The Local Structural Semantics

This does not correspond to an actual change of the (local) configuration of a node but indicates the *possibility* of a local transition. It will be up to the global scheduler to activate such a potential update. In the local semantics we only consider how the process P changes, while the local tuple spaces TS remains the same and again it will be the global semantics to determine or not an actual update of TS.

The local semantics is defined in Table 6. As in the original semantics for KLAIM, we use the label *action* to describe the activities performed in the evolution; thus, for example $o(t)@\ell$ refers to the action of sending the tuple t in the tuple space specified by ℓ, and $r(t)@\ell$ is the action of consuming the tuple t in the tuple space specified by ℓ.

Global Semantics The global semantics relies on the idea that state changes (transitions) do occur at certain points in time. In the discrete time case, every time step the scheduler selects one node to initiate an update of the whole network according to a (normalised) probability q. In the continuous time case jumps from one network state to another occur at rates specified by the scheduling rates λ. These rates determine an exponentially distributed time between transitions from one configuration of the network into another, according to a continuous time Markov chain model (cf. e.g. [12–14]).

According to Table 5 a probabilistic KLAIM network is either of the form $N \equiv \|_{i=1}^n s_i ::^{q_i} P_i$ or $N \equiv \|_{i=1}^n s_i ::^{\lambda_i} P_i$. We define a network configuration as a pair $[N, TS]$ with TS a global tuple space which is constructed out of the local tuple spaces TS_i, i.e. $TS = (TS_1, TS_2, \ldots, TS_n) = (TS_i)_{i=1}^n$. We will denote a

network configuration $[N, TS] = [\|_{i=1}^{n} s_i ::^{x_i} P_i, (TS_i)_{i=1}^{n}]$ also as:

$$\|_{i=1}^{n} s_i ::^{x_i} [P_i, TS_i] = s_1 ::^{x_1} [P_1, TS_1] \parallel s_2 ::^{x_1} [P_2, TS_2] \parallel \ldots s_n ::^{x_i} [P_n, TS_n],$$

where $x_i = q_i$ or $x_i = \lambda_i$.

Discrete Time Version. The discrete time semantics of KLAIM networks is defined as a Discrete Time Markov Chain (DTMC) where the states are the network configurations; we will denote by $\mathcal{N}$ the set of all such configurations. A discrete time random process is a sequence $\{X_t\}_{t=1}^{\infty}$ of random variables, i.e. of functions $X_t : \Omega \to S$ from a probability space Ω into a state space S. We will restrict our presentation to finite[2] state spaces S and identify random variables $X(t) = X_t$ with their associated probability distributions $\mathbf{P}(X_t = s)$, i.e. the probability that the random variable X_i will be in state s. For finite state spaces S we can represent this probability distribution with a (column) vector which we will also denote by X_t.

A discrete time random process with initial distribution X_0 is called a discrete time Markov chain if the distribution for X_{t+1} only depends on the previous distribution X_t

$$\mathbf{P}(X_{t+1} = s_{t+1} \mid X_0 = s_0, \ldots, X_t = s_t) = \mathbf{P}(X_{t+1} = s_{t+1} \mid X_t = s_t).$$

This allows us to determine the distribution X_{t+1} via $X_{t+1} = X_t \mathbf{P}(t)$ where $\mathbf{P}(t)$ is a stochastic matrix, i.e. a matrix with row sums equal to one. For so called homogeneous DTMCs we have $\mathbf{P}(t) = \mathbf{P}$ for all t and:

$$X_t = X_{t-1}\mathbf{P} \quad \text{or} \quad X_t = X_{t-n}\mathbf{P}^n$$

In the case of the discrete time KLAIM networks we define their semantics as a (homogeneous) DTMC as follows: The state space S is the set of all possible network configurations $\mathcal{N}$. We can restrict ourselves to the set of network configurations $\mathcal{N}(N(0))$ which are reachable from the initial configuration $N(0) = [N_0, TS_0]$ and assume that $\mathcal{N}(N(0))$ is finite. The entries in the transition matrix $\mathbf{P}$ are then defined by using the rules in Table 7 as:

$$\mathbf{P}_{N_i,N_j} = \begin{cases} \sum p_{ij} & \text{with } N_i \overset{p_{ij}}{\rightarrowtail} N_j \\ 0 & \text{otherwise.} \end{cases}$$

The rules in Table 7 describe how a global scheduler can utilise potential local transitions in order to update the global network configuration. The update is triggered by one of the nodes, at s_1, with a probability corresponding to its scheduling probability p_i. Each update involves at most two nodes at sites s_1 and s_2 in the context of the remaining nodes of the network, denoted by N. If $s_1 = s_2$ the rules in Table 7 have to be applied in the obvious way.

[2] For countable infinite state spaces we have to use probability measures instead of probability distributions.

As some nodes could be blocked — e.g. because no matching tuple is available for an **in** to proceed — we have to use the normalised probabilities $\tilde{p}_i$. For this we define the set of *active* sites in a global configuration $[P, TS] = \|_{i=1}^{n} s_i ::^{p_i} [P_i, TS_i]$ as:

$$Active([P, TS]) = \{s_i \mid [P_i, TS_i] \xrightarrow{a}_p [P_i', TS_i']\}\}$$

i.e. a site s_i is active if its process P_i can make at least one local transition with some action a and probability p. Then we define

$$\tilde{p}_i = \frac{p_i}{C_{[P,TS]}} \quad \text{with} \quad C_{[P,TS]} = \sum_j \{p_j \mid s_j \in Active([P, TS])\}.$$

In a similar way we also have to normalise the local probability p. The active actions of a local configuration (in some network context) are given by:

$$\begin{aligned} Active([P_i, TS_i]) = \{&o(t)@s_j \mid [P_i, TS_i] \xrightarrow{o(t)@s_j}_p [P_i', TS_i']\} \\ \cup \{&i(e)@s_j \mid [P_i, TS_i] \xrightarrow{i(t)@s_j}_p [P_i', TS_i'] \text{ and } \exists e \in TS_j : e \rhd t\} \\ \cup \{&r(e)@s_j \mid [P_i, TS_i] \xrightarrow{r(t)@s_j}_p [P_i', TS_i'] \text{ and } \exists e \in TS_j : e \rhd t\} \end{aligned}$$

and with this we get the normalised local transition probabilities as:

$$\tilde{p} = \frac{p_i}{C_{[P,TS]}} \quad \text{with} \quad C_{[P_i,TS_i]} = \sum_j \{p_j \mid a_j \in Active([P_i, TS_i])\}.$$

If no node is active for a network configuration N we will force a diagonal entry $\mathbf{P}_{N,N} = 1$ to guarantee that $\mathbf{P}$ is indeed a stochastic matrix. Operationally this corresponds to introducing a self-transition or loop for stuck network configurations.

Continuous Time Version. In this model each node can initiate a network update independently at any time with a certain probability which is proportional to its rate. This parameter is specified by the superscript λ in the syntax of a node. We assume that these rates are independent from the time and therefore each node "fires", i.e. initiates an update, via a so called Poisson process (see e.g. [13, Sect 2.4]).

We model the continuous time semantics of KLAIM networks as a Continuous Time Markov Chains (CTMC), i.e. as a particular continuous time random process $\{X_t\}_{t\in[0,\infty)}$. Like in the case of DTMCs the dependency between the random variables $X_t = X(t)$ in a CTMC is very restricted: it depends only on (any) single previous moment. This means that there exist stochastic matrices $\mathbf{P}(t)$, with $t \in [0, \infty)$, such that we can compute the distribution X_t as:

$$X_t = X_{t-\triangle t}\mathbf{P}(\triangle t).$$

$$\frac{[P_1, TS_1] \xrightarrow{o(t)@s_2}_p [P_1', TS_1]}{s_1 ::^{p_1} [P_1, TS_1] \| s_2 ::^{p_2} [P_2, TS_2] \| N \overset{\tilde{p}\tilde{p}_1}{\Longrightarrow} s_1 ::^{p_1} [P_1', TS_1] \| s_2 ::^{p_2} [P_2, TS_2 \oplus t] \| N}$$

$$\frac{[P_1, TS_1] \xrightarrow{i(t)@s_2}_p [P_1', TS_1] \quad \exists e \in TS_2 : e \rhd t}{s_1 ::^{p_1} [P_1, TS_1] \| s_2 ::^{p_2} [P_2, TS_2] \| N \overset{\tilde{p}\tilde{p}_1}{\Longrightarrow} s_1 ::^{p_1} [P_1'[x/e], TS_1] \| s_2 ::^{p_2} [P_2, TS_2 - e] \| N}$$

$$\frac{[P_1, TS_1] \xrightarrow{r(t)@s_2}_p [P_1', TS_1] \quad \exists e \in TS_2 : e \rhd t}{s_1 ::^{p_1} [P_1, TS_1] \| s_2 ::^{p_2} [P_2, TS_2] \| N \overset{\tilde{p}\tilde{p}_1}{\Longrightarrow} s_1 ::^{p_1} [P_1'[x/e], TS_1] \| s_2 ::^{p_2} [P_2, TS_2] \| N}$$

Table 7. Discrete Time Network Semantics

The matrices $\mathbf{P}(t)$ form a semi-group, i.e. $\mathbf{P}(0) = \mathbf{I}$ the identity matrix ($p_{ij}(0) = 1$ for $i = j$ and $p_{ij}(0) = 0$ otherwise) and for any $t, s \in [0, \infty)$ we have the so called semi-group property: $\mathbf{P}(s + t) = \mathbf{P}(s)\mathbf{P}(t)$.

It is possible to obtain the $\mathbf{P}(t)$ matrices as solutions to certain linear differential equations (cf. e.g. [13]). This allows us to specify the $\mathbf{P}(t)$s via the parameters describing these differential equations. These parameters are referred to as the rate or Q-matrix $\mathbf{Q} = (q_{ij})_{ij}$ which has the following properties:

- $0 \leq -q_{ii} < \infty$ for all i,
- $q_{ij} \geq 0$ for all $i \neq j$,
- $\sum_j q_{ij} = 0$ for all i.

From the Q-matrix of a system we can obtain the transition probabilities $\mathbf{P}(t)$ via:

$$\mathbf{P}(t) = \exp(t\mathbf{Q}) = \sum_{n=0}^{\infty} \frac{(t\mathbf{Q})^n}{n!}$$

In the case of the continuous time semantics for KLAIM networks we only need to specify the rate matrix $\mathbf{Q}$ using the rules in Table 8:

$$\mathbf{Q}_{N_i, N_j} = \begin{cases} \sum w_{ij} & \text{for } N_i \overset{w_{ij}}{\Longrightarrow} N_j \\ -\sum_{j \neq i} w_{ij} & \text{for } N_i = N_j \\ 0 & \text{otherwise.} \end{cases}$$

In Table 8 the relation $\overset{w_{ij}}{\Longrightarrow}$ between two network configurations N_i and N_j is labelled by rates w_{ij} which are obtained as a product between the firing rate λ_k of the node which initiates the update and the normalised probabilities $\tilde{p}$ of the local transitions occurring in the nodes involved in the update. The normalisation of the local transition probabilities is again needed in order to accommodate locally blocked transitions. The rates λ need no normalisation

$$\frac{[P_1, TS_1] \xrightarrow{o(t)@s_2}_p [P_1', TS_1]}{s_1 ::^{\lambda_1} [P_1, TS_1] \| s_2 ::^{\lambda_2} [P_2, TS_2] \| N \overset{\tilde{p}\lambda_1}{\Longrightarrow} s_1 ::^{\lambda_1} [P_1', TS_1] \| s_2 ::^{\lambda_2} [P_2, TS_2 \oplus t] \| N}$$

$$\frac{[P_1, TS_1] \xrightarrow{i(t)@s_2}_p [P_1', TS_1] \quad \exists e \in TS_2 : e \rhd t}{s_1 ::^{\lambda_1} [P_1, TS_1] \| s_2 ::^{\lambda_2} [P_2, TS_2] \| N \overset{\tilde{p}\lambda_1}{\Longrightarrow} s_1 ::^{\lambda_1} [P_1'[x/e], TS_1] \| s_2 ::^{\lambda_2} [P_2, TS_2 - e] \| N}$$

$$\frac{[P_1, TS_1] \xrightarrow{r(t)@s_2}_p [P_1', TS_1] \quad \exists e \in TS_2 : e \rhd t}{s_1 ::^{\lambda_1} [P_1, TS_1] \| s_2 ::^{\lambda_2} [P_2, TS_2] \| N \overset{\tilde{p}\lambda_1}{\Longrightarrow} s_1 ::^{\lambda_1} [P_1'[x/e], TS_1] \| s_2 ::^{\lambda_2} [P_2, TS_2] \| N}$$

Table 8. Continuous Time Network Semantics

and we need no special treatment of completely blocked network configurations (this is achieved via the construction of the diagonal elements in $\mathbf{Q}$ and related to the basic fact that $\exp(0) = 1$).

The continuous time model realises true concurrency as several transitions *seem* to happen in "parallel". In fact, two transitions are actually never happening at exactly the same moment, as the probability for this is zero. However, after a single time unit we can observe that two or more transitions have happened.

This allows us to avoid considering "clashes" like for example two $\mathbf{in}(t)$ actions trying to access the same token: the probability of this happening vanishes. We can however ask for the probability that either of the two **in**'s is executed first and in this way determine the chances that the token in question has been consumed by the first or the second **in** after a given time (or, as also could be the case, that neither of them has already consumed the token).

3.3 Stochastic KLAIM

An alternative stochastic version of KLAIM is the proposal in [8]. The language defined in this work, called StocKLAIM, extends a core subset of KLAIM by associating to each action some specific rates representing the time taken by the action to be executed. Similarly to the continuous time version of our pKLAIM, this time is determined by random variables which are exponentially distributed, so that the operational semantics of the language can be represented in terms of stochastic processes and in particular as a Continuous Time Markov Chain.

In StocKLAIM the only source of probabilistic information is therefore the action prefix process whose syntax $(a, r).P$ allows for the specification of a rate by instantiating the name r. All the other constructs of the language, namely the choice $P + P$ and the parallel composition $P \mid P$ as well the network parallel operator $N \parallel N$ keep their non deterministic syntax, so that for example no information (coming e.g. from statistical or other form of data which are available for a given application) can be specified at this level.

The straightforward relation of the structural operational semantics defined via a labelled transition system with the CTMC model, makes various tools and techniques which have been developed for stochastic model checking directly available for the analysis of a network specified in StochKLAIM. In fact one main motivation of this approach is towards the definition of logics and semantics based tools for the performance modelling and analysis of mobile and distributed application.

Alternatively, a probabilistic model like the one offered by probabilistic KLAIM (cf. Section 3.2) is more oriented (even when it is based on a CMTC model) towards a quantitative analysis of networks based on tools and techniques which are typical of program analysis; these are obtained by transforming the approaches that have been developed for a purely qualitative static analysis of program properties into probabilistic/quantitative ones suitable for a distributed setting (cf. Section 4).

4 Analysis

Given a program in probabilistic Linda or a network in probabilistic KLAIM we are interested in analysing properties such as the chances that a program terminates, or the chance that a token (worm) moves from one node in the network to another one in a given number of steps (or time interval). As in classical program analysis, also probabilistic properties can be expressed as solutions to a set of (in)equations. Since most properties are undecidable, an exact solution of these (in)equations is often not possible. We are thus led to construct reasonable approximations. Depending on the structure of the domain of the (in)equations there are various options regarding a formal definition of "reasonable approximation". In particular, we can consider the following two settings:

Order Theoretic: This is based on partial orders or lattice structures and aims at computing the best "safe" solution. Classical Abstract Interpretation [15, 16] utilises the notion of a Galois Connection to achieve this aim.

Linear Structures: This is based on linear spaces or operator algebras and allows for the construction of least squares solution as "closest" fit. Probabilistic Abstract Interpretation [17] uses the so-called Moore-Penrose Pseudo-inverse for this purpose.

In this paper we will assume the second setting as the base of our treatment.

4.1 Probabilistic Abstract Interpretation

Probabilistic Abstract Interpretation (PAI) is a general framework introduced in [17] for the static analysis of probabilistic programs. Similarly to the classical abstract interpretation framework, it provides general techniques for constructing approximations of the semantics of a system relatively to a given property of interest. However, the *correctness* of these approximations which is guaranteed in the classical case by the *order-theoretic* notion of *Galois connection*, is replaced

in PAI by a notion of *closeness* which includes some quantitative measurement of the loss of precision. This is obtained by moving from the order-theoretic setting of classical abstract interpretation to one based on linear spaces and linear operators, where the notion of so-called *Moore-Penrose pseudo-inverse* (see below) replaces the classical notion of a Galois connection. Moreover, the properties of the Moore Penrose inverse guarantees the *optimality* of the approximations constructed via PAI: they are the closest possible to the concrete semantics of the given system.

The definition of a probabilistic abstract interpretation is given in terms of *probabilistic domains*. A probabilistic domain is essentially a space which represents the distributions $Dist(S)$ on a state space S. In the general case including infinite dimensional vector spaces, a probabilistic domain is defined as the Hilbert space $\mathcal{H}(S) = \ell^2(S)$ on S[18, 19]. However, in the finite dimensional case this is equivalent to consider the simple vector space $\mathcal{V}(S)$, built out of all linear combinations of elements from S with coefficients in $\mathbb{R}$:

$$\mathcal{V}(S) = \left\{ \sum c_s \boldsymbol{s} \mid c_s \in \mathbb{R},\ s \in S \right\}.$$

For the purpose of this work it is sufficient for us to consider only finite dimensional vector spaces, so we will present the PAI framework in this restricted setting.

Definition 2. Let $\mathcal{C}$ an $\mathcal{D}$ be two probabilistic domains. A *probabilistic abstract interpretation* is a pair of bounded linear operators $\mathbf{A} : \mathcal{C} \to \mathcal{D}$ and $\mathbf{G} : \mathcal{D} \to \mathcal{C}$, between (the concrete domain) $\mathcal{C}$ and (the abstract domain) $\mathcal{D}$, such that $\mathbf{G}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$, and vice versa.

A particular PAI technique similar to the classical abstract interpretation technique defining a so-called *induced abstract semantics* consists in the following: Given a linear operator Φ on some Hilbert space $\mathcal{V}$ expressing the probabilistic semantics of a concrete system, and a linear abstraction function $\mathbf{A} : \mathcal{V} \mapsto \mathcal{W}$ from the concrete domain into an abstract domain $\mathcal{W}$, we compute the Moore-Penrose pseudo-inverse $\mathbf{G} = \mathbf{A}^\dagger$ of $\mathbf{A}$. The abstract semantics can then be defined as the linear operator on the abstract domain $\mathcal{W}$:

$$\Psi = \mathbf{A} \circ \Phi \circ \mathbf{G}.$$

Moore-Penrose Pseudo-Inverse We can define the notion of a Moore-Penrose pseudo-inverse of a bounded linear operator $\mathbf{A} \in \mathcal{B}(\mathcal{H})$ on a Hilbert space $\mathcal{H}$ purely algebraically (cf. Section 4.7 of [20], and Section 8.43 of [21]). This is sufficient for the finite-dimensional setting, while for dealing with the infinite-dimensional case we need some topological considerations [22].

Definition 3. Let $\mathcal{C}$ and $\mathcal{D}$ be two Hilbert spaces and $\mathbf{A} : \mathcal{C} \mapsto \mathcal{D}$ a bounded linear map between them. A bounded linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{D} \mapsto \mathcal{C}$ is the *Moore-Penrose pseudo-inverse* of $\mathbf{A}$ iff

(i) $\mathbf{A} \circ \mathbf{G} = \mathbf{P}_A$, and
(ii) $\mathbf{G} \circ \mathbf{A} = \mathbf{P}_G$,

where $\mathbf{P}_A$ and $\mathbf{P}_G$ denote orthogonal projections onto the ranges of $\mathbf{A}$ and $\mathbf{G}$.

In the finite dimensional case, the Moore-Penrose pseudo inverse of a linear operator always exists and is unique; moreover various algorithms are known for its construction [23]. A general technique for computing the Moore-Penrose pseudo-inverse of infinite operators is to approximate them by a sequence of finite dimensional operators. For infinite dimensional operators — i.e. operator algebras over infinite dimensional Hilbert spaces — various results guarantee the existence of the Moore-Penrose pseudo-inverse for every operator. For the general case we mention here the one in [20] (Theorem 4.24) which also states how one can "construct" the Moore-Penrose Pseudo-Inverse.

Probabilistic Abstract Interpretation and Classification. In many cases the abstraction is a surjective function. An alternative view of abstraction in this case is that it maps concrete values to equivalence classes. Equivalence relations can be represented by a particular kind of operators, namely classification operators. In the finite dimensional setting these can be described as follows.

We call an $n \times m$-matrix $\mathbf{K}$ a *classification* matrix, if it is a 0/1-matrix, where every row has exactly one non-zero entry and columns have at least one non-zero entry. Classification matrices are thus particular kinds of stochastic matrices. We denote by $\mathcal{K}(n, m)$ the set of all $n \times m$-classification matrices. Let $X = \{x_1, \ldots, x_n\}$ be a finite set. Then for each equivalence relation $\approx$ on X with $|X/_{\approx}| = m$, there exists a classification matrix $\mathbf{K} \in \mathcal{K}(n, m)$ and vice versa.

The Moore-Penrose pseudo-inverse of a classification matrix $\mathbf{K} \in \mathcal{K}(n, m)$ corresponds to its normalised transpose or adjoint (these coincide for real $\mathbf{K}$), i.e.

$$\mathbf{K}^\dagger = \mathcal{N}(\mathbf{K}^T) = \mathcal{N}(\mathbf{K}^*).$$

where the normalisation operation $\mathcal{N}$ is defined for a matrix $\mathbf{A}$ by:

$$\mathcal{N}(\mathbf{A})_{ij} = \begin{cases} \frac{\mathbf{A}_{ij}}{a_j} \text{ if } a_j = \sum_i \mathbf{A}_{ij} \neq 0 \\ 0 \quad \text{otherwise.} \end{cases}$$

4.2 Analysis – Discrete Case

In order to exploit the framework of Probabilistic Abstract Interpretation in the case of probabilistic KLAIM we need a semantics given in terms of linear (transition) operators. This is provided by the operators $\mathbf{P}$ and $\mathbf{P}(t)$ introduced in Section 3.2.

In the case of the discrete time model we have to consider a single step operator $\mathbf{P}$ which describes the probabilistic transitions between (network) configurations. For KLAIM we can construct this operator either as a direct encoding of the operational semantics (as in Section 3.2) or compositionally reflecting the two-layered semantics:

The local semantics defines a Probabilistic Transition System (PTS) — this is represented as a linear operator, in particular a stochastic matrix [24].

The global semantics is then constructed compositionally as the tensor product of the local semantics [25].

Based on the concrete semantics given by $\mathbf{P}$ we can construct an abstract induced semantics **GPA** using some abstraction operator $\mathbf{A}$ and its Moore-Penrose pseudo inverse $\mathbf{G} = \mathbf{A}^\dagger$. This amounts effectively to a "simplification" of the DTMC by reducing the dimension of the transition matrix $\mathbf{P}$.

Example 4. Consider the following 5×5 transition matrix:

$$\mathbf{P} = \begin{pmatrix} 0 & \frac{3}{4} & \frac{1}{4} & 0 & 0 \\ \frac{3}{4} & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & 0 & \frac{3}{4} \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} & 0 \end{pmatrix}$$

Suppose that we abstract states into one of two classes. This corresponds to partition the set of states in two equivalence classes which can suitably be represented via the classification operator $\mathbf{K}$ and its pseudo-inverse $\mathbf{K}^\dagger$:

$$\mathbf{K} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \text{ and } \mathbf{K}^\dagger = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

In this case the abstract 2×2 transition matrix is:

$$\mathbf{K}^\dagger \mathbf{P} \mathbf{K} = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{6} & \frac{5}{6} \end{pmatrix},$$

which can then be safely used to replace P to simplify our analysis. Note that in our PAI framework "safely" means that the approximation error we make is controllable, that is always quantifiable.

In fact, one advantage of the use of linear operators is that we can measure them. The standard way to measure the "size" of a linear operator is via an operator norm which in turn may have its origins in a vector norm. This allows us, for example, to quantify the error introduced by the abstraction. The close relation between least squares approximation and the Moore-Penrose pseudo-inverse (cf. e.g. [23]) guarantees that the abstract induced semantics is giving the closest (with respect to the Euclidean norm) approximation to the concrete behaviour among all the possible ones.

4.3 Analysis – Continuous Case

We can also apply this simplification technique to CTMCs. Concretely, we have to construct the generator $\mathbf{Q}$ as described in Section 3.2. The probability that the network configuration $N(t)$ at any time t is N_j starting from initial configuration N_i is then $\mathbf{P}(t)_{ij} = \mathbf{P}(N(t) = n_j \mid N(0) = N_i) = (\exp(t\mathbf{Q}))_{ij}$.

In the same way as with the discrete time model we can simplify the operator $\mathbf{P}(t)$ by subjecting it to an abstract interpretation.

In fact, the common method for effectively computing $\mathbf{P}(t)$ is closely related to a particular form of probabilistic abstract interpretation. It is based on the fact that every matrix — in particular the generator matrix $\mathbf{Q}$ — can be "abstracted" into a so called Jordan canonical form, e.g. [26, Chap 7] or [27, Sect III.12].

A *Jordan matrix* $\mathbf{J}$ is a square matrix of the form $\mathbf{J} = \mathrm{diag}(\mathbf{J}_{r_1}(\lambda_1), \ldots, \mathbf{J}_{r_m}(\lambda_m)$ with $\mathbf{J}_{r_i}(\lambda_i)$ so called *Jordan blocks*, i.e. $r_i \times r_i$ matrices of the form:

$$\mathbf{J}_{r_i}(\lambda_i) = \begin{pmatrix} \lambda_i & 1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_i & 1 \\ 0 & 0 & 0 & \cdots & 0 & \lambda_i \end{pmatrix} = \begin{pmatrix} \lambda_i & 0 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_i & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_i & 0 \\ 0 & 0 & 0 & \cdots & 0 & \lambda_i \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

Theorem 5. *[27, Thm III.12.2] Any complex square matrix* $\mathbf{A}$ *is* similar *to a Jordan matrix* $\mathbf{J}$*, i.e. there exists an invertible matrix* $\mathbf{X}$ *such that* $\mathbf{A} = \mathbf{X}^{-1}\mathbf{J}\mathbf{X} = \mathbf{X}^{\dagger}\mathbf{J}\mathbf{X}$.

As every Jordan block $\mathbf{J}_{r_i}(\lambda_i)$ is the sum of a diagonal $\mathbf{D}_{r_i}(\lambda_i) = \mathrm{diag}(\lambda_i, \ldots, \lambda_i)$ and a strict upper triangular matrix $\mathbf{N}_{r_i}$ the same holds for Jordan matrices, i.e. $\mathbf{J} = \mathbf{D} + \mathbf{N}$. Furthermore, it is easy to see that $\mathbf{N}$ is *nilpotent*, i.e. there exists a $n \in \mathbb{N}$ such that $\mathbf{N}^m$ is the null matrix, and that $\mathbf{D}$ and $\mathbf{N}$ commute: $\mathbf{D}\mathbf{N} = \mathbf{N}\mathbf{D}$, see e.g. [26, 27].

This allows for an efficient way of computing $\mathbf{P}(t) = \exp(t\mathbf{Q})$: With the Jordan canonical form $\mathbf{J}$ of $\mathbf{Q}$, i.e. $\mathbf{Q} = \mathbf{X}^{-1}\mathbf{J}\mathbf{X}$, we get:

$$\begin{aligned} \exp(t\mathbf{Q}) &= \sum_{k=0}^{\infty} \frac{1}{k!}(t\mathbf{Q})^k = \sum_{k=0}^{\infty} \frac{t^k}{k!}(\mathbf{X}^{-1}\mathbf{J}\mathbf{X})^k = \sum_{k=0}^{\infty} \frac{t^k}{k!}\mathbf{X}^{-1}\mathbf{J}^k\mathbf{X} \\ &= \mathbf{X}^{-1}\left(\sum_{k=0}^{\infty} \frac{1}{k!}(t\mathbf{J})^k\right)\mathbf{X} = \mathbf{X}^{-1}\exp(t\mathbf{J})\mathbf{X} \end{aligned}$$

Furthermore, we can compute the exponential of a Jordan matrix $\mathbf{J}$ easily (exploiting the fact that $\mathbf{D}$ and $\mathbf{J}$ commute):

$$\exp(t\mathbf{Q}) = \exp(t\mathbf{D} + t\mathbf{N})) = \exp(t\mathbf{D})\exp(t\mathbf{N}).$$

Finally, we observe that the exponential of diagonal matrices $\mathbf{D}$ is simply:

$$\exp(\mathrm{diag}(d_1, d_2, \ldots, d_n)) = \mathrm{diag}(\exp(d_1), \exp(d_2), \ldots, \exp(d_n))$$

and that for nilpotent matrices $\mathbf{N}$ the series

$$\exp(\mathbf{N}) = \sum_{k=0}^{\infty} \frac{1}{k!}\mathbf{N}^k = \sum_{k=0}^{m} \frac{1}{k!}\mathbf{N}^k$$

degenerates to a finite sum, with m the nilpotency of $\mathbf{N}$. In the case of diagonalisable matrices $\mathbf{Q}$ the Jordan canonical form is a diagonal matrix, i.e. $\mathbf{N}$ is the null matrix, which means that we obtain $\mathbf{P}(t) = \exp(t\mathbf{Q}) = \mathbf{X}^{-1}\exp(t\mathbf{D})\mathbf{X} = \mathbf{X}^{-1}\mathrm{diag}(\exp(td_1), \exp(td_2), \ldots, \exp(td_n))\mathbf{X}$.

This approach is a special instance of a general way for solving linear differential equations, see e.g. [28]. Unfortunately, the key property $\exp(\mathbf{X}^{-1}\mathbf{J}\mathbf{X}) = \mathbf{X}^{-1}\exp(\mathbf{J})\mathbf{X}$ does not hold if we consider proper abstractions, i.e. $\mathbf{X}^{\dagger}$ instead of $\mathbf{X}^{-1}$, i.e. $\exp(\mathbf{X}^{\dagger}\mathbf{J}\mathbf{X}) \neq \mathbf{X}^{\dagger}\exp(\mathbf{J})\mathbf{X}$. This is due to the fact that $(\mathbf{X}^{\dagger}\mathbf{J}\mathbf{X})^k \neq \mathbf{X}^{\dagger}\mathbf{J}^k\mathbf{X}$ as $\mathbf{X}\mathbf{X}^{\dagger} \neq \mathbf{I}$. The factor $\mathbf{X}\mathbf{X}^{\dagger}$ describes exactly the approximation error we introduce by considering the abstract in place of the concrete semantics.

5 Conclusions

We have explored the design space for adding quantitative information to coordination languages. We have used a basic Linda calculus for this. We showed that one could either add priorities or probabilities; we also demonstrated how such quantities could be added to the data in the tuple space or to the processes for scheduling parallel threads. We have also shown how to add mobility, as in the KLAIM language. We introduced probabilities both at the local (or process) level and at the network level. We also presented a continuous-time model where we use rates to determine how often a node is active. This information contributes to the probability of network updates.

The probabilistic version of KLAIM we have introduced in this paper is closely related to various *probabilistic programming languages* and *probabilistic process calculi* proposed in the recent literature. Among these we mention discrete time approaches — e.g. PCCS [29, 30], PCCP [31], etc. — as well as continuous time approaches — e.g. PEPA [11], Stochastic π calculus [32].

Work in performance analysis is often based on probabilistic process calculi, for example, on Hillston's PEPA [33], or EMPA by Bernardo and Gorrieri [34]. One of the long term aims of the work presented in this paper is the development of semantics based approaches towards *performance analysis* along similar lines as in classical program analysis. We also aim to investigate more closely the relation of our work to recent work on probabilistic verification and model checking, such as PRISM [35] and de Alfaro [36].

We have considered here a model based on Poisson processes which are some of the simplest examples of continuous-time Markov chains. More complicated continuous time behaviour could be considered, but this might require more parameters than just rate to describe the time distributions [14]. The language could also be extended so as to allow for a dynamic change of probabilities and rates, i.e. for rate and probability which depend on the time. These last two extensions require further work.

References

1. Gelernter, D.: Generative communication in linda. ACM Trans. Program. Lang. Syst. **7** (1985) 80–112
2. Gelernter, D., Carriero, N.: Coordination languages and their significance. Commun. ACM **35** (1992) 97–107
3. Arbab, F.: Manifold. Future Generation Computer Systems **10** (1994) 273–277
4. Sands, D., Weichert, M.: From gamma to cbs: Refining multiset transformations with broadcasting processes. In El-Rewini, H., ed.: Proceedings of 31st Hawaii International Conference on System Sciences. Volume VII., IEEE (1998) 265–274
5. Bravetti, M., Gorrieri, R., Lucchi, R., Zavattaro, G.: Quantitative information in the tuple space coordination model. Theoretical Computer Science (To appear)
6. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic KLAIM. In Nicola, R.D., Ferrari, G., Meredith, G., eds.: Proceedings of Coordination 2004. Number 2949 in Lecture Notes in Computer Science, Berlin — Heidelberg — New York, Springer Verlag (2004) 119–134
7. Di Pierro, A., Hankin, C., Wiklicky, H.: Continuous-time probabilistic KLAIM. In: SecCo'04 — CONCUR Workshop on Security Issues in Coordination Models, Languages, and Systems. Electronic Notes in Theoretical Computer Science, Elsevier (2004)
8. Nicola, R.D., Latella, D., Massink, M.: Formal modeling and quantitative analysis of KLAIM-based mobile systems. In: 20th Annual ACM Symposium on Applied Computing, ACM (2005)
9. van Glabbeek, R., Smolka, S., Steffen, B.: Reactive, generative and stratified models of probabilistic processes. Information and Computation **121** (1995) 59–80
10. De Nicola, R., Ferrari, G., Pugliese, R.: KLAIM: A kernel language for agents interaction and mobility. IEEE Transactions on Software Engineering **24** (1998) 315–330
11. Hillston, J.: PEPA: Performance enhanced process algebra. Technical Report CSR-24-93, University of Edinburgh, Edinburgh, Scotland (1993)
12. Tijms, H.C.: Stochastic Models – An Algorithmic Approach. John Wiley & Sons, Chichester (1994)
13. Norris, J.: Markov Chains. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge (1997)
14. Bause, F., Kritzinger, P.S.: Stochastic Petri Nets – An Introduction to the Theory. second edn. Vieweg Verlag (2002)
15. Cousot, P., Cousot, R.: Abstract Interpretation and Applications to Logic Programs. Journal of Logic Programming **13** (1992) 103–180
16. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer Verlag, Berlin – Heidelberg (1999)
17. Di Pierro, A., Wiklicky, H.: Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In: Proceedings of PPDP'00, Montréal, Canada, ACM (2000) 127–138
18. Di Pierro, A., Wiklicky, H.: Linear structures for concurrency in probabilistic programming languages. In: Proceedings of MFCSIT00 – First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology, Cork, Ireland. Volume 40 of Electronic Notes in Theoretical Computer Science., Elsevier (2001)
19. Di Pierro, A., Wiklicky, H.: Operator algebras and the operational semantics of probabilistic languages. In: Proceedings of MFCSIT04 – Third Irish Conference on

the Mathematical Foundations of Computer Science and Information Technology, Dublin, Ireland. Electronic Notes in Theoretical Computer Science, Elsevier (To appear)
20. Böttcher, A., Silbermann, B.: Introduction to Large Truncated Toeplitz Matrices. Springer Verlag, New York (1999)
21. Deutsch, F.: Best Approximation in Inner Product Spaces. Volume 7 of CMS Books in Mathematics. Springer Verlag, New York — Berlin (2001)
22. Di Pierro, A., Hankin, C., Wiklicky, H.: Measuring the confinement of probabilistic systems. Theoretical Computer Science **340** (2005) 3–56
23. Ben-Israel, A., Greville, T.: Generalised Inverses — Theory and Applications. second edn. Volume 15 of CMS Books in Mathematics. Springer Verlag, New York — Berlin (2003)
24. Di Pierro, A., Hankin, C., Wiklicky, H.: Quantitative relations and approximate process equivalences. In Lugiez, D., ed.: Proceedings of CONCUR'03 — International Conference on Concurrency Theory. Number 2761 in Lecture Notes in Computer Science, Berlin — Heidelberg — New York, Springer Verlag (2003) 508–522
25. Di Pierro, A., Hankin, C., Wiklicky, H.: Quantitative static analysis of distributed systems. Journal of Functional Programming **15** (2005) 1–47
26. Friedberg, S., Insel, A., Spence, L.: Linear Algebra. forth edn. Prentice Hall (2003)
27. Prasolov, V.: Problems and Theorems in Linear Algebra. Volume 134 of Translation of Mathematical Monographs. American Mathematical Society, Providence, Rhode Island (1994)
28. Hirsch, M., Smale, S.: Differential Equations, Dynamical Systems, and Linear Algebra. Academic Press, Orlando (1974)
29. Giacalone, A., Jou, C.C., Smolka, S.: Algebraic reasoning for probabilistic concurrent systems. In: Proceedings of the IFIP WG 2.2/2.3 Working Conference on Programming Concepts and Methods, Sea of Galilee, North-Holland (1990) 443–458
30. Jonsson, B., Yi, W., Larsen, K.: 11. In: Probabilistic Extensions of Process Algebras. Elsevier Science, Amsterdam (2001) 685–710 see [**?**].
31. Di Pierro, A., Wiklicky, H.: Quantitative observables and averages in Probabilistic Concurrent Constraint Programming. In Apt, K., Kakas, T., Monfroy, E., Rossi, F., eds.: New Trends in Constraints — Selected Papers of the ERCIM/Compulog Workshop on Constraints, October 1999, Paphos, Cyprus. Number 1865 in Lecture Notes in Computer Science, Berlin — Heidelberg — New York, Springer Verlag (2000)
32. Priami, C.: Stochastic π-calculus. Computer Journal **38** (1995) 578–589
33. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
34. Bernardo, M., Gorrieri, R.: A tutorial on empa: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. Technical Report UBLCS-96-17, Department of Computer Science, University of Bologna (1997)
35. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. In Katoen, J.P., Stevens, P., eds.: Proceedings of TACAS'02. Volume 2280 of Lecture Notes in Computer Science., Springer Verlag (2002) 52–66
36. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University, Department of Computer Science (1998)