On a Probabilistic Chemical Abstract Machine and the Expressiveness of Linda Languages

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

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

Abstract. The Chemical Abstract Machine (CHAM) of Berry and Boudol provides a commonly accepted, uniform framework for describing the operational semantics of various process calculi and languages, such as for example CCS, the π calculus and coordination languages like Linda. In its original form the CHAM is purely non-deterministic and thus only describes what reactions are *possible* but not how long it will take (in the average) before a certain reaction takes place or its *probability*. Such quantitative information is however often vital for "real world" applications such as systems biology or performance analysis. We propose a probabilistic version of the CHAM. We then define a linear operator semantics for the probabilistic CHAM which exploits a tensor product representation for distributions over possible solutions. Based on this we propose a novel approach towards comparing the expressive power of different calculi via their encoding in the probabilistic CHAM. We illustrate our approach by comparing the expressiveness of various Linda Languages.

1 Introduction

The chemical reaction metaphor was introduced in [1]. Gamma is a declarative programming language that supports massive parallelism. A Gamma program consists of a *shared data space* – a multiset – and a collection of *conditional rewrite rules*. The chemical metaphor is of molecules reacting in a solution under physical laws; the condition in a Gamma rule is normally referred to as the *reaction* condition, whilst the non-conditional part of the rule is referred to as the *action*. A major principle in Gamma is that of local action: rules consume a small number of elements from the multiset and produce a small number of elements into the multiset; the conditional application of rules is determined by predicates over the consumed elements – there is no global state. Rules potentially compete for elements from the multiset. The "program" terminates when no further rules are enabled. This process is described in [2] in the following way: computation is "the global result of the successive applications of local, independent, atomic reactions".

One motivation for the model was that the standard data types used in declarative languages, for example the ubiquitous list, were over-constraining

for parallel systems. For example, a Gamma rule for computing primes is:

$$x, y \rightarrow x \Leftarrow \mathit{multiple}(x, y)$$

where *multiple* is a predicate which is true whenever y is a multiple of x (the reader should compare this with the usual sieve solution). When this rule is combined with the solution $\{\!|2, 3, 4, 5, 6, 7, 8|\!\}$, the program will terminate with $\{\!|2, 3, 5, 7|\!\}$. There are a number of different reduction sequences that lead to this result; for example, one sequence might include the parallel execution of:

$$2, 4 \rightarrow 2 \text{ and } 3, 6 \rightarrow 3$$

There have been a number of later developments of Gamma; for example to introduce more structure to the multiset but without constraining the execution [3] and to add higher-order features [4]. The shared data space concept has proved to be a powerful principle in coordination programming [5]. More importantly from the perspective of this paper, Gamma provided inspiration for Berry and Boudol's Chemical Abstract Machine [6].

The CHAM was introduced to provide an abstract machine-based operational semantics for process calculi. It has been used for a variety of calculi including CCS, π-calculus and Linda-based coordination languages. The solution in a CHAM is again a multiset which may be structured using the notion of membranes – to encapsulate a sub-multiset – and airlocks – to expose part of a sub-multiset for interaction with the rest of the solution. A CHAM specification is then a collection of rules. Rules are either *specific* or *general*. Specific rules are similar to our primes example from above; however, the CHAM rules generally rely on sophisticated pattern matching rather than reaction conditions – this is sufficient to match input channels and output channels in the typical synchronous communication of CCS. The general rules provide for the "compatible closure" of the specific rules – allowing computation inside membranes and also in the presence of other elements in the solution.

The main contributions of this paper are twofold:

- to adapt the CHAM model to allow probabilistic computation; we believe this to be important not only to provide a formal semantics for the burgeoning number of probabilistic process calculi but also because probabilities will be essential for the more advanced modelling of biological and chemical systems.
- to use the structure of particular CHAMs to compare the expressiveness of different calculi.

The rest of this paper is structured as follows: in Section 2 we introduce the probabilistic CHAM; we then present a linear operator semantics for the pCHAM in Section 3; Section 4 presents a number of properties of the pCHAM; we describe the process of encoding various calculi in the pCHAM in Section 5 — this is relatively straightforward and follows the classical CHAM encodings; Section 6 concerns expressiveness; and we conclude in Section 7.

2 A Probabilistic CHAM

The idea of the *probabilistic CHAM* (pCHAM) is to "quantify" the likelihood or probability of executing an applicable rule. This allows us to resolve any non-deterministic choice in a reaction (sequence) probabilistically. We define the semantics of the pCHAM in terms of a Probabilistic Transition System (PTS) where the state space is represented by multisets of molecules. We recall the general definition of a (labelled) PTS as given in [7, Def 2].

Definition 1. A *probabilistic transition system* is a tuple $(S, A, \longrightarrow, \pi_0)$, where:

- S is a non-empty, countable set of *states*,
- A is a non-empty, finite set of *actions*,
- $\longrightarrow \subseteq S \times A \times Dist(S)$ is a *transition relation*, and
- $\pi_0 \in Dist(S)$ is an *initial distribution* on S.

A PTS $(S, A, \longrightarrow, \pi_0)$ is called *generative* if the transition relation is a partial function $\longrightarrow\colon S \hookrightarrow Dist(S \times A)$.

The semantics of a pCHAM is defined by a generative unlabelled PTS, i.e. a PTS with a single, anonymous label $\tau \in A$ (which we simply omit). We will concentrate on finite state spaces S, although we will occasionally also remark on the general countable case.

2.1 State Space

The probabilistic CHAM has the same basic state space as the classical CHAM, namely *solutions*, i.e. multisets of molecules. We denote by $\mathcal{T}$ the set of possible molecules, i.e. terms in some formal algebra or language, and by $\mathcal{M} = \mathcal{M}(\mathcal{T})$ the set of multisets of molecules in $\mathcal{T}$, i.e. functions of the form $S : \mathcal{T} \to \mathbb{N}$ for which we will usually use the common notation $\{\!\{ \dots \}\!\}$.

We will assume a finite set of possible molecules in $\mathcal{T} = \{m_1, \dots, m_t\}$, i.e. $t = |\mathcal{T}| < \infty$ and a finite (strict) upper bound s for the multiplicity of any molecule, i.e. $\max_{m_i \in \mathcal{T}} S(m_i) < s < \infty$. We will denote the set of possible multiplicities by $\mathcal{N} = \{1, \dots, s-1\} \subseteq \mathbb{N}$. These finiteness conditions can be relaxed relatively easily. However, they allow us for the time being a clearer presentation of the basic elements of the pCHAM; we can, for example, work with distributions in place of general measures, etc.

We will refer to distributions over solutions, i.e. over $\mathcal{M} = \mathcal{M}(\mathcal{T})$, as *ensembles* of molecules in $\mathcal{T}$ and denote them by:

$$\mu = \{\langle \{\!\{ m_{11}, \dots, m_{1i_1} \}\!\}, p_1 \rangle, \dots, \langle \{\!\{ m_{j1}, \dots, m_{ji_j} \}\!\}, p_j \rangle\}$$

where i_k is the cardinality and p_k the probability of the multiset $\{\!\{ m_{k1}, \dots, m_{ki_k} \}\!\}$. For the sake of simplicity of notation we use m_{ji} instead of the the more correct notation m_{j_i}. Moreover, in order to make the representation of ensembles more compact, we do not list multisets with zero probability.

A pCHAM with molecules in $\mathcal{T}$ and with an initial solution $S_0 \in \mathcal{M}(\mathcal{T})$ defines a probabilistic transition system $(\mathcal{M}, \Longrightarrow_p, \mu_0)$ with the point distribution $\mu_0 = \{\langle S_0, 1 \rangle\}$.

2.2 Specific Rules

The transition relation $\Longrightarrow_p$ for the PTS $(\mathcal{M}, \Longrightarrow_p, \mu_0)$ representing a pCHAM is specified via a certain set $\mathcal{R}$ of *specific rules* or — as with the non-deterministic CHAM in order to avoid "multiset matching" — by *rule schemata*. These rules are expressions of the form:

$$m_{i1}, \ldots, m_{ik} \longrightarrow_p m_{j1}, \ldots, m_{jl}$$

where $m_{i'j'}$ are molecules (or variables, cf [6]). These rules specify an individual pCHAM by describing possible rewriting steps together with a *probability* p.

It is important (in the context of the pCHAM) to distinguish between rules, for which we use the notation $\longrightarrow_p$, and the probabilistic transition relation on $\mathcal{M}$ which defines the multiset rewriting and for which we use the notation $\Longrightarrow_p$.

Probabilities are associated with rules not molecules. However, the rules of a specific pCHAM can exploit information contained in the molecules in order to obtain the intended probability p.

Example 2. We can also introduce probabilistic information as part of a molecule, e.g. $m'_i = p_i : m_i$, i.e. we can annotate standard molecules by providing information about their "reactiveness". Rules like $m_1, m_2 \longrightarrow m_3$ then would become, for example, something like:

$$p_1 : m_1, p_2 : m_2 \longrightarrow_{p_1 \cdot p_2} \max(p_1, p_2) : m_3$$

Example 3. Another possibility could be to provide "position information" and make the reaction probability of two molecules proportional to their "spatial" closeness, e.g.:

$$m_1@(x_1, y_1), m_2@(x_2, y_2) \longrightarrow_{d(m_1, m_2)} m_3@((x_1 + x_2)/2, (y_1 + y_2)/2)$$

with $d(m_1, m_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ the "Euclidean distance" between m_1 and m_2.

Example 4. The "spatial" information can also be generalised to "allocation environments" — similar to the notion in KLAIM [8], i.e. we can specify a sub-(multi)set $\{\!|m_1, \ldots, m_k|\!\}$ of molecules a given molecule m_0 can react with, i.e. the (multi)set of its "neighbours".

A further generalisation could introduce "probabilistic allocation environments" which specifies not just the possibility of reactions between certain molecules, but also their probability. Allocation environments can be used to simulate membranes of sub-solutions and probabilistic allocation environments allow the introduction of "soft membranes".

Example 5. Any non-deterministic CHAM can be lifted to a pCHAM by replacing the specific rule $m_{i1}, \ldots, m_{ik} \longrightarrow m_{j1}, \ldots, m_{jl}$ by $m_{i1}, \ldots, m_{ik} \longrightarrow_1 m_{j1}, \ldots, m_{jl}$. The 'standard probability' $p = 1$ will be renormalised (see below)

to obtain a uniform distribution over all possible transformations of a multiset. Note that this is only one of the many possible ways a nondeterministic specific rule can be *implemented* as a probabilistic rule in order to specify an individual pCHAM. We will make the relationship between non-deterministic and probabilistic CHAM more precise in Section 4.2.

2.3 General Laws

A concrete pCHAM is defined by specifying its set of possible molecules $\mathcal{T}$, specific rules $\mathcal{R}$ and (optionally) an initial solution $S_0 \in \mathcal{M}$. The rules of the pCHAM can be extended and translated into the transition relation $\Longrightarrow_p$ of the corresponding probabilistic transition system $(\mathcal{M}, \Longrightarrow_p, \mu_0)$, with $\mu_0 = \{\langle S_0, 1\rangle\}$, via four *general laws* which are straightforward generalisations of the laws of the non-deterministic CHAM, cf [6, Sect 3].

Reaction Law. This provides the essential mechanism which translates or lifts specific pCHAM rules to multiset rewritings. It also renormalises the (intended) transition probabilities p such that the probabilities associated to the transitions from any given multiset add up to one.

$$\frac{m_{i1}, \ldots, m_{ik} \longrightarrow_p m_{j1}, \ldots, m_{jl}}{\{\!|m_{i1}, \ldots, m_{ik}|\!\} \Longrightarrow_{\tilde{p}} \{\!|m_{j1}, \ldots, m_{jl}|\!\}}$$

where $\tilde{p}$ is the normalised probability $\tilde{p} = \frac{p}{P}$ with P the sum over all possible rewritings of $\{\!|m_{i1}, \ldots, m_{ik}|\!\}$.

This law is *non-local* as the normalisation has to take into account all the other transitions which could be applied to a given multiset/solution. However, as we will see later, we can actually avoid normalisation until we reach, for example, a terminal solution.

Chemical Law. The remaining general laws of the pCHAM extend the specific rules. The Chemical Law allows us to apply a rule in any context:

$$\frac{m_{i1}, \ldots, m_{ik} \longrightarrow_p m_{j1}, \ldots, m_{jl}}{m_{i1}, \ldots, m_{ik}, m'_1, \ldots, m'_n \longrightarrow_p m_{j1}, \ldots, m_{jl}, m'_1, \ldots, m'_n}$$

where $m'_1, \ldots, m'_n$ is any (maybe empty) collection of molecules.

This means that if we state a specific rule like $m_{i1}, \ldots, m_{ik} \longrightarrow_p m_{j1}, \ldots, m_{jl}$ for a pCHAM we also can use rules like $m_{i1}, \ldots, m_{ik}, m_0 \longrightarrow_p m_{j1}, \ldots, m_{jl}, m_0$, and $m_{i1}, \ldots, m_{ik}, m_0, m_3 \longrightarrow_p m_{j1}, \ldots, m_{jl}, m_0, m_3$, etc.

Membrane Law. For solutions with solutions as molecules we extend the set of specific rules such that a sub-solution can develop on its own.

$$\frac{m_{i1}, \ldots, m_{ik} \longrightarrow_p m_{j1}, \ldots, m_{jl}}{m'_1, \ldots, m'_n, \{\!|m_{i1}, \ldots, m_{ik}|\!\} \longrightarrow_p m'_1, \ldots, m'_n, \{\!|m_{j1}, \ldots, m_{jl}|\!\}}$$

If we define a *context* $C[]$ as usual as a multiset with a "hole", i.e. as a multiset $\{\!|m_1, \ldots, m_n, []|\!\}$ with a distinguished "hole" molecule [] which can be replaced by any multiset S then we obtain the corresponding law of the general CHAM:

$$\frac{\{\!|m_{i1}, \ldots, m_{ik}|\!\} \Longrightarrow_p \{\!|m_{j1}, \ldots, m_{jl}|\!\}}{\{\!|m'_1, \ldots, m'_n, \{\!|m_{i1}, \ldots, m_{ik}|\!\}|\!\} \Longrightarrow_{\tilde{p}} \{\!|m'_1, \ldots, m'_n, \{\!|m_{j1}, \ldots, m_{jl}|\!\}|\!\}}$$

or more concisely:

$$\frac{S \Longrightarrow_p S'}{\{\!|C[S]|\!\} \Longrightarrow_{\tilde{p}} \{\!|C[S']|\!\}}$$

where $\tilde{p}$ is obtained by (re)normalising p.

Airlock Law. The "partial activation" of a sub-solution providing an 'airlock' to the surrounding solution is realised via:

$$\begin{aligned} m, m_1, \ldots, m_n &\longrightarrow_{h(T)} m \triangleleft \{\!|m_1, \ldots, m_n|\!\} \\ m \triangleleft \{\!|m_1, \ldots, m_n|\!\} &\longrightarrow_{f(T)} m, m_1, \ldots, m_n \end{aligned}$$

The probabilities $h(T)$ and $f(T)$ specify the chances of "heating" up or "freezing" down airlocks. It could be the case that $h(T) = f(T)$, i.e. that the two rewritings happen with the same probability, but it could also be that one happens more frequently. Furthermore, it is also possible to change the probabilities depending on a control parameter T (*temperature*).

Considering the multiset rewrites which this universal rule justifies we get the following "probabilistic version" of the corresponding classical law:

$$\{\!|m|\!\} \uplus S \Longrightarrow_{\tilde{h}(T)} \{\!|m \triangleleft S|\!\} \quad \text{and} \quad \{\!|m \triangleleft S|\!\} \Longrightarrow_{\tilde{f}(T)} \{\!|m|\!\} \uplus S.$$

2.4 An Example

Consider (a finite version) of a CHAM which implements the well known sieve method for finding prime numbers. The initial solution is in this case the set of natural numbers $\{2, 3, \ldots, n\}$ or in general a multiset of numbers. The specific reaction rules are as follows:

> If i and j are in solution such that there exists a $k \neq 1$ such that $j = i \cdot k$ then eliminate j from solution.

or as a simple rule schemata:

$$i, j \longrightarrow j \text{ iff } \exists k \neq 1. ik = j$$

An execution of this CHAM leads, for example, to the following reductions:

$$\begin{aligned} \{\!|2, 3, 4, 5, 6, 7, 8, 9|\!\} &\longrightarrow \{\!|2, 3, 5, 6, 7, 8, 9|\!\} \\ &\longrightarrow \{\!|2, 3, 5, 7, 8, 9|\!\} \\ &\longrightarrow \{\!|2, 3, 5, 7, 8|\!\} \\ &\longrightarrow \{\!|2, 3, 5, 7|\!\} \end{aligned}$$

It is easy to see that we will always end up with the (multi)set of primes up to n. However, it is left open how fast we will reach this state, or how long it will take in the average until a certain non-prime number is eliminated.

A probabilistic version of the CHAM in this example needs to specify the probabilities for rule applications. A rather simple, maybe somewhat uninspired, way to this is to assume each rule will fire with the same probability, i.e.

$$i, j \longrightarrow_1 j \text{ iff } \exists k \neq 1. ik = j$$

One possible sequence of reductions for this pCHAM is then:

$$\begin{aligned} \{\!\{2,3,4,5,6,7,8,9\}\!\} &\Longrightarrow_{\frac{1}{6}} \{\!\{2,3,5,6,7,8,9\}\!\} \\ &\Longrightarrow_{\frac{1}{4}} \{\!\{2,3,5,7,8,9\}\!\} \\ &\Longrightarrow_{\frac{1}{2}} \{\!\{2,3,5,7,8\}\!\} \\ &\Longrightarrow_{1} \{\!\{2,3,5,7\}\!\} \end{aligned}$$

Again, it is easy to see that this pCHAM will always end up with a multiset containing only primes. However, (despite the uniform distribution of probabilities to rules) we observe certain execution paths or traces with different probabilities, e.g. the one above with probability $\frac{1}{6}\frac{1}{4}\frac{1}{2} = \frac{1}{48}$.

3 Linear Operator Semantics of the pCHAM

As in the theory of stochastic processes, in particular of Markov Chains, we encode the probabilistic transition relation $\Longrightarrow_p$ as a linear operator $\mathbf{T}$ on the vector spaces $\mathcal{V}(\mathcal{M})$. Suppose we have an enumeration of the solutions in $\mathcal{M}$. The fact that $S_i \Longrightarrow_{p_{ij}} S_j$ will then be reflected by the fact that the entry $\mathbf{T}_{ij} = p_{ij}$. The advantage of the linear operator semantics is that it not only encodes the probability of transitions between solutions but also canonically extends to a relation between ensembles, i.e. distributions over solutions.

3.1 State Space

The first question we have to address is how many possible solutions are there in $\mathcal{M}$; this means that we have to determine the reachable set, i.e. the state space of possible configurations of the pCHAM. Unfortunately, the general situation requires an exponentially growing space to represent it: Assume that we have t different types of molecules and that their multiplicity is restricted by s, i.e. a given molecule/term type can appear with multiplicities 0 (not at all), 1, 2, etc. up to $s-1$.

A finite multiset representing such a solution can be defined as a map from the set of all possible molecules $\mathcal{T} = \{m_1, m_2, \ldots, m_t\}$ to the set of multiplicities $\mathcal{N} = \{0, 1, \ldots, s-1\}$. The cardinality of the set of all maps $\mathcal{T} \to \mathcal{N}$ is $|\mathcal{N}|^{|\mathcal{T}|} = s^t$.

Ensembles correspond to particular vectors in the vector space of (formal) linear combinations of multisets:

$$\mathcal{V}(\mathcal{M}) = \left\{ \sum_i x_i S_i \mid x_i \in \mathbb{R} \text{ and } S_i \in \mathcal{M} \right\}$$

which, concentrating on the *coordinates* x_i, we can also identify with the space of tuples in $\mathbb{R}^{|\mathcal{M}|}$. A distribution over solutions of the pCHAM is a positive vector with 1-norm one, i.e. $x_i \geq 0$ for all i and $\sum_i |x_i| = 1$. The vector space containing all ensembles thus is unfortunately extremely large: $\mathcal{V}(\mathcal{M}(\mathcal{T})) = \mathbb{R}^{s^t}$.

3.2 Tensor Product Representations

The state space of size s^t is prohibitively large but at the same time unavoidable; a priori we cannot exclude any of the s^t possible molecular solutions and in principle it is possible that rules governing the dynamics of the pCHAM specify transitions from any of the s^t configurations to any other.

However, we can exploit the "structure" of the state space. If we consider for the moment only a single type of molecules, i.e. $t = 1$, then we have only to consider the bounding multiplicity s. The state space of this type of pCHAM has $s^1 = s$ possible states; ensembles thus are vectors in $\mathbb{R}^s = \mathcal{V}(\{0, \ldots, s-1\}) = \mathcal{V}(\mathcal{N})$. Considering two types of molecules, i.e. $t = 2$, requires that we keep track of the multiplicity of each of the two types of molecules. We thus get the state space of possible solutions as the Cartesian product $\mathcal{N} \times \mathcal{N}$. The possible ensembles, i.e. distributions of this space, are then elements of the tensor product: $\mathcal{V}(\mathcal{N} \times \mathcal{N}) = \mathcal{V}(\mathcal{N}) \otimes \mathcal{V}(\mathcal{N})$.

As an example, let us consider an ensemble on $\mathcal{T} = \{m_1, m_2, m_3\}$ which is given by $\{\langle S_1, \frac{1}{3}\rangle, \langle S_2, \frac{2}{3}\rangle\}$ with $S_1 = \{\!\{m_1, m_1, m_3\}\!\}$ and $S_2 = \{\!\{m_1, m_2\}\!\}$. The two multisets are represented by the vectors (taking as bound $s = 2$ for the multiplicities) in $\mathbb{R}^{3^3} = (\mathbb{R}^3)^{\otimes 3} = \mathbb{R}^{27}$: $(0,0,1) \otimes (1,0,0) \otimes (0,1,0)$ — which specifies the multiplicity of m_1 to be 2, the one for m_2 as 0 and of m_3 to be 1 — and $(0,1,0) \otimes (0,1,0) \otimes (1,0,0)$ — which expresses the fact that molecules m_1 and m_2 have a multiplicity one, while m_3 does not appear — and which we denote by μ_1 and μ_2. For both solution the three factors in the tensor product describe the multiplicity of each of the three molecules m_1, m_2 and m_3; the entries in these factors (p_0, p_1, p_2) specify that the molecule is missing with probability p_0, that there is one copy with probability p_1, and that its multiplicity is two with probability p_2. The original ensemble is represented by the weighted vector sum

$$\frac{1}{3}\mu_1 + \frac{2}{3}\mu_2 = \frac{1}{3}(0,0,1) \otimes (1,0,0) \otimes (0,1,0) + \frac{2}{3}(0,1,0) \otimes (0,1,0) \otimes (1,0,0).$$

Generalising this construction gives us an alternative description of the state-space of a pCHAM with t types of molecules and bounding multiplicity s:

$$\mathcal{V}(\mathcal{N}^t) = \mathcal{V}(\mathcal{N})^{\otimes t} = (\mathbb{R}^s)^{\otimes t} = \mathbb{R}^{s^t}$$

where $\mathcal{V}^{\otimes t}$ denotes the t-fold tensor product of $\mathcal{V}$, i.e. $\mathcal{V} \otimes \mathcal{V} \otimes \ldots \otimes \mathcal{V}$.

Although this representation (obviously) does not reduce the dimension of the state space of the pCHAM it "partitions" it in a certain way which will allow us to describe "local" rules in a more efficient way.

3.3 Representation of Rules

The encoding of a specific rule of a concrete pCHAM is straightforward and purely syntax-directed. Given a rule of the form:

$$m_{i_1}, \ldots, m_{i_k} \longrightarrow_p m_{j_1}, \ldots, m_{j_l}$$

we can translate it into a linear operator on the ensemble space $\mathcal{V}(\mathcal{M}) = (\mathbb{R}^s)^{\otimes t}$.

We first need to consider a *creation operator* $\mathbf{C} = \mathbf{C}_s$ and a *destruction operator* $\mathbf{D} = \mathbf{D}_s$ on each tensor component of $\mathcal{V}(\mathcal{M})$, i.e. on $\mathcal{V}(\mathcal{N}) = \mathbb{R}^s$ represented by the following matrices which increase or decrease the multiplicity of certain molecules, i.e.

$$(\mathbf{C})_{ij} = \begin{cases} 1 \text{ for } j = i+1 \\ 0 \text{ otherwise} \end{cases} \quad \text{and} \quad (\mathbf{D})_{ij} = \begin{cases} 1 \text{ for } j = i-1 \\ 0 \text{ otherwise} \end{cases}$$

These creation and destruction operators link the semantics of the pCHAM closely to so-called "Birth-and-Death processes" in probability theory, cf. e.g. [10].

Using these operators we can increase and decrease the multiplicity of every molecule m_k in a solution. To increase, for example, the multiplicity of the molecule m_2 by two in a solution made up of (at most) four different molecules we have to apply $(\mathbf{I} \otimes \mathbf{C} \otimes \mathbf{I} \otimes \mathbf{I}) \cdot (\mathbf{I} \otimes \mathbf{C} \otimes \mathbf{I} \otimes \mathbf{I}) = (\mathbf{I} \otimes \mathbf{CC} \otimes \mathbf{I} \otimes \mathbf{I})$ with $\mathbf{I}$ the identity operator/matrix on $\mathcal{V}(\mathcal{N})$, to the vector representing a given solution. In general, we can define the following two operators on $\mathcal{M}$:

$$\mathbf{G}_k = \bigotimes_{i=1}^{k-1} \mathbf{I} \otimes \mathbf{C} \otimes \bigotimes_{i=k+1}^{t} \mathbf{I} \quad \text{and} \quad \mathbf{K}_k = \bigotimes_{i=1}^{k-1} \mathbf{I} \otimes \mathbf{D} \otimes \bigotimes_{i=k+1}^{t} \mathbf{I}$$

which increase (generate) or decrease (kill) the multiplicity of the molecule m_k.

The last thing we need in order to define the encoding of rules is a test operator which checks whether there exists a certain molecule in the current solution. The local version of this existence operator $\mathbf{E}$ and its obvious extension $\mathbf{E}^{\geq \min}$ are given by:

$$(\mathbf{E})_{ij} = \begin{cases} 1 \text{ for } i = j \geq 1 \\ 0 \text{ otherwise} \end{cases} \quad \text{and} \quad (\mathbf{E}^{\geq \min})_{ij} = \begin{cases} 1 \text{ for } i = j \geq \min \\ 0 \text{ otherwise} \end{cases}$$

Note that $\mathbf{E}^n \neq \mathbf{E}^{\geq n}$. Using these local tests we can construct a global one:

$$\mathbf{E}_k = \bigotimes_{i=1}^{k-1} \mathbf{I} \otimes \mathbf{E} \otimes \bigotimes_{i=k+1}^{t} \mathbf{I} \quad \text{and} \quad \mathbf{E}_k^{\geq \min} = \bigotimes_{i=1}^{k-1} \mathbf{I} \otimes \mathbf{E}^{\geq \min} \otimes \bigotimes_{i=k+1}^{t} \mathbf{I}$$

A specific rule of a pCHAM $m_{i_1}, \ldots, m_{i_k} \longrightarrow_p m_{j_1}, \ldots, m_{j_l}$ is now represented by the operator product:

$$\mathbf{E}_{i_1}^{\geq S(m_{i_1})} \cdots \mathbf{E}_{i_k}^{\geq S(m_{i_k})} \cdot \mathbf{K}_{i_1}^{S(m_{i_1})} \cdots \mathbf{K}_{i_k}^{S(m_{i_k})} \cdot \mathbf{G}_{j_1}^{S(m_{j_1})} \cdots \mathbf{G}_{j_l}^{S(m_{j_l})}$$

where $S(m_k)$ is the multiplicity of each molecule on the left and right-hand side of the rule. The encoding simply tests if enough molecules are present such that the rule can be applied, then it destroys the molecules mentioned on the left-hand side of the rule, and finally it generates all molecules on the right-hand side. Of course, one can think of optimising the encoding by only destroying molecules which do not re-appear on the right-hand side, etc.

3.4 Representation of pCHAMs

After we have represented all specific rules we finally can present the encoding of a concrete pCHAM $(\mathcal{M}, \mathcal{R}, \mu_0)$. We first have to construct the operators $\mathbf{R}_i$ of all specific rules in $\mathcal{R}$ together with all the extended rules we obtain by the Chemical, Membrane and Airlock Laws. We denote the extended set of rules by $\mathcal{R}'$. Each of these rules comes with a probability p_i (either inherited from a more specific version if the Chemical, Membrane or Airlock Law was involved or directly from the specification of the specific rules).

The execution of the pCHAM now simply corresponds to choosing one of the applicable rule operators with the corresponding probability and applying it to the vector representing the current solution or more generally ensemble. This is achieved by considering the operator:

$$\mathbf{T} = \mathcal{N}\left(\sum_{R_i \in \mathcal{R}'} p_i \mathbf{R}_i\right)$$

where the *normalisation* operation $\mathcal{N}$ is defined by:

$$\mathcal{N}(\mathbf{T})_{ij} = \begin{cases} \frac{\mathbf{T}_{ij}}{T_j} & \text{if } T_j = \sum_i \mathbf{T}_{ij} \neq 0 \\ 1 & \text{if } T_j = \sum_i \mathbf{T}_{ij} = 0 \text{ and } i = j \\ 0 & \text{otherwise.} \end{cases}$$

The test operators $\mathbf{E}$ in the definitions of the $\mathbf{R}_i$s "filter" out all those rules which are not applicable, the p_i's weight the chances of each reaction according to the specific rules, and the normalisation $\mathcal{N}$ computes the correct probabilities $\tilde{p}_i$. If a solution is inactive, normalisation adds a one on the diagonal which preserves the current solution without changing anything.

The linear operator $\mathbf{T}$ encodes the probabilistic transition relation $\Longrightarrow_p$ of the probabilistic transition system $(\mathcal{M}, \Longrightarrow_p, \mu_0)$ defining the operational semantics of the pCHAM $(\mathcal{M}, \mathcal{R}, \mu_0)$ as stated by the following proposition.

Proposition 6. *Given a pCHAM $(\mathcal{M}, \mathcal{R}, \mu_0)$, let $\Longrightarrow_p$ be the probabilistic transition relation on $\mathcal{M}$ of the associated probabilistic transition system, and let $\mathbf{T}$ be the linear operator on $\mathcal{V}(\mathcal{M})$ associated to $\mathcal{R}$. Then, for all $S_i, S_j \in \mathcal{M}$*

$$S_i \Longrightarrow_p S_j \text{ iff } \mathbf{T}_{ij} = p.$$

In order to implement the execution of a pCHAM $(\mathcal{M}, \mathcal{R}, \mu_0)$ we have only to compute the iterated applications of the operator to the vector representing the initial ensemble, i.e. $\mathbf{T}^n(\mu_0)$, which realises a discrete time Markov Chain. Depending on the questions we are interested in we can investigate, for example, the long run average of this Markov Chain, or other features commonly studied in the theory of stochastic processes.

4 Properties of the pCHAM

Let us next discuss some of the properties and aspects of the pCHAM and its linear operator semantics. This is not an exhaustive study but merely attempts to address some of the more interesting features.

4.1 Completeness of the Linear Operator Semantics

In the previous section we have shown that the linear operator semantics allows us to encode any transformation of ensembles of a set of molecules $\mathbf{T}$, that is any pCHAM. The following proposition shows that the reverse also holds.

Proposition 7. *For every linear operator $\mathbf{T}$ on a finite-dimensional vector space $\mathcal{V}$ there exists a set of molecules $\mathcal{T}$ with $|\mathcal{T}| = t$ and multiplicity bounded by s and a pCHAM $(\mathcal{M}, \mathcal{R}, \mu_0)$ over $\mathcal{T}$, such that $\mathcal{V} = \mathcal{V}(\mathcal{M}(\mathcal{T}))$, i.e. the tensor product $(\mathbb{R}^s)^{\otimes t}$, and $\mathbf{T}$ is represented by a linear combination of the rule operators $\mathbf{R}_i$.*

Proof. It is sufficient to show that all *matrix units* $\mathbf{B}_{ij}$, i.e. matrices with a single 1 at row i and column j and all other entries 0, for each tensor factor $\mathcal{V}(\mathcal{N})$ can be represented as $\mathbf{E}^{\geq \max} \cdot \mathbf{D}^n \cdot \mathbf{C}^m$.

One can show that $\mathbf{B}_{ij} = \mathbf{E}^{\geq i}\mathbf{D}^{i-1}\mathbf{C}^j$. Operationally this corresponds to filtering out all situations where the multiplicity of a molecule is too small, then by destroying all additional copies and then creating the needed j copies.

Any element in $\mathcal{V}(\mathcal{N})^{\otimes t}$ can then be represented by a linear combination of matrix units in $\mathcal{V}(\mathcal{N})^{\otimes t}$ which in turn are represented as the tensor product of certain matrix units in each $\mathcal{V}(\mathcal{N})$. □

This result proves that any Markov Chain on $\mathcal{V}(\mathcal{M}(\mathcal{T}))$ can be represented by a pCHAM, i.e. whatever (memoryless, discrete time) random process one chooses, it is always possible to define the rules of a particular pCHAM which implements this behaviour.

It is interesting to note that the proof of this fact is not by giving an explicit construction of this pCHAM but by arguing that the dimension of the space generated by the "rule operators" is as large, i.e. has the same dimension, as $\mathcal{L}(\mathcal{V}(\mathcal{M}(\mathcal{T})))$ — where $\mathcal{L}(\mathcal{V})$ denotes the set of all linear operators on $\mathcal{V}$ — and that we therefore can construct a representation of any possible random "behaviour" as a linear combination of the basic operators $\mathbf{C}$, $\mathbf{D}$ and $\mathbf{E}$. We will utilise a similar reasoning when we compare the expressiveness of calculi in Section 6.

4.2 Non-Deterministic vs Probabilistic CHAMs

An obvious issue concerns the relation between the classical CHAM and its probabilistic version. In order to clarify this relation it may help to discuss the role non-determinism plays in a specification as opposed to the role of probability. The following example illustrates the difference.

Example 8. Consider two types of molecules: a and d. We think of the first one as "active" elements which can spontaneously produce a d molecule or turn itself into a d molecule; the other one is "dead", i.e. it decays immediately.

The specification of such a behaviour via a non-deterministic CHAM can be given using the following specific rules.

$$a \longrightarrow a, d \qquad a \longrightarrow d \qquad d \longrightarrow$$

A concrete pCHAM with the same behaviour might have the following rules:

$$a \longrightarrow_{\frac{1}{2}} a, d \qquad a \longrightarrow_{\frac{1}{2}} d \qquad d \longrightarrow_1$$

or a general specification of pCHAMs with this behaviour could utilise 'unspecified' probabilities, i.e. with $p \in [0, 1]$:

$$a \longrightarrow_p a, d \qquad a \longrightarrow_{1-p} d \qquad d \longrightarrow_1$$

If we assume *fairness* in the CHAM case and $p \in (0, 1)$, i.e. $p \neq 0$ and $p \neq 1$ then we could expect that we get the same long time behaviour if we start with $\{\!|a|\!\}$ in the case of the CHAM and pCHAM formulation.

It is easy to see that in the case of the CHAM it is possible to obtain after n steps the solution $\{\!|a|\!\}$ as well as $\{\!|a, d, \ldots, d|\!\} = \{\!|a, d^k|\!\}$ for $k \leq n$, as well as the empty solution $\{\!|\,|\!\}$. However, in the case of the pCHAM, we will — after n steps — quite likely have the empty solution. More precisely, the probability of obtaining $\{\!|a, d^k|\!\}$ tends to vanish for increasing n.

In some sense the probabilistic specification reflects more closely than the non-deterministic specification the real situation as it discards the most unlikely solution. As the classical example of the Gambler's Ruin (cf. e.g. [9]) shows: In real life it is not enough to state that it is *possible* that one will win the lottery; we need to say how *probable* this is.

Essentially we can simulate a non-deterministic CHAM by a pCHAM by "forgetting" about the concrete probabilities which describe the ensembles $\mathbf{T}^n(\mu)$.

Given a non-deterministic CHAM $(\mathcal{M}, \mathcal{R}, \mu_0)$ we can construct a corresponding pCHAM $(\mathcal{M}, \mathcal{R}', \mu_0)$ by attaching to each classical rule in $\mathcal{R}$ some non-zero probability in order to obtain the set of probabilistic rules $\mathcal{R}'$. Vice versa we can also construct a non-deterministic CHAM for every given pCHAM by dropping the probabilities related to the rules. A pCHAM gives rise to a unique CHAM, but there are several pCHAM which correspond to any given CHAM.

Proposition 9. *Given a CHAM $(\mathcal{M}, \mathcal{R}, S_0)$ and a corresponding pCHAM represented by the operator $\mathbf{T}$ on $\mathcal{V}(\mathcal{M}(\mathcal{T})) = \mathcal{V}(\mathcal{M})^{\otimes t}$, a solution $S \in \mathcal{M}$ is reachable in n reaction steps in $(\mathcal{M}, \mathcal{R}, S_0)$ if and only if $\mathbf{T}^n(\{\langle S_0, 1\rangle\})$ has a non-zero component corresponding to S.*

4.3 Locality of Rules of the pCHAM

Several people have argued that the need for normalisation imposes a severe non-locality constraint on probabilistic models of computations. This is only partly true as one can postpone the normalisation under certain circumstances. More precisely, if we start with a point distribution μ, i.e. if we know exactly with which chemical solution we start, then instead of normalising $\mathbf{T}$ we can normalise the $\mathbf{T}(\mu)$ — extending the normalisation to positive vectors in the obvious way by dividing them by their 1-norm $\|.\|_1$, i.e. the sum of all their coordinates.

Proposition 10. *Given a pCHAM* $(\mathcal{M}, \mathcal{R}, \mu_0)$ *and its operator* $\mathbf{T} = \mathcal{N}(\mathbf{T}')$ *with* $\mathbf{T}' = \sum_i p_i \mathbf{R}_i$ *on* $\mathcal{V}(\mathcal{M}(\mathcal{T})) = \mathcal{V}(\mathcal{N})^{\otimes t}$ *and an ensemble* $\mu = \{\langle \mu, 1\rangle\}$ *then we have:*

$$\mathbf{T}(\mu) = \begin{cases} \mathcal{N}(\mathbf{T}'(\mu)) & \text{if } \mathcal{N}(\mathbf{T}'(\mu)) \neq o \\ \mu & \text{otherwise} \end{cases} \quad \text{with } \mathcal{N}(\mu) = \begin{cases} \frac{\mu}{\|\mu\|_1} & \text{if } \|\mu\|_1 \neq 0 \\ \mu & \text{otherwise.} \end{cases}$$

We have for a single step and a point-ensemble essentially that $\mathbf{T}'$ and $\mathcal{N}$ commute, i.e. $\mathcal{N}(\mathbf{T}')(\mu) = \mathcal{N}(\mathbf{T}'(\mu))$ — only in the case of "blocked" solutions, which result in the zero vector o, we have to complicate things by 're-producing' the original vector.

In other words, if we consider the probabilities along a certain execution path we do not need to normalise the complete operator $\mathbf{T}'$. We can start with a point-ensemble, compute the probabilities of its successors in the above way by $\mathcal{N}(\mathbf{T}'(\mu))$, pick a *single* successor ensemble and repeat the application of the non-normalised $\mathbf{T}'$. Non-locality problems only arise when we consider $\mathbf{T}^n(\mu)$. The reason for this is that in this case we do not compute (the probabilities of) a *single* computational path but of *all* possible paths at the same time. This obviously implies the need to "distribute" the available probability non-locally between them by normalising $\mathbf{T}'$.

The operator $\mathbf{T}'$ encodes the rules $\longrightarrow_p$ while $\mathbf{T} = \mathcal{N}(\mathbf{T}')$ represents the rewrite steps $\Longrightarrow_p$. The possibility to postpone normalisation has also important consequences for the Linear Operator Semantics and its tensor product representation which can be treated in a "lazy" fashion. The only time we have to compute the tensor product effectively is when we normalise $\mathbf{T}'$, as long as we work with the original $\mathbf{T}'$ we can apply it component-wise to a given vector as for a distribution $\mu = \sum_i x_i \mu_i$ with $\mu_i = \bigotimes_j \mu_{ij}$ the application of $\mathbf{T} = \sum_k p_k \mathbf{R}_k$ with $\mathbf{R}_k = \bigotimes_j \mathbf{R}_{kj}$ is obtained as:

$$\mathbf{T}(\mu) = \sum_k p_k \sum_i (\bigotimes_j \mathbf{R}_{kj})(\bigotimes_j \mu_{ij}) = \sum_k p_k \sum_i \bigotimes_j \mathbf{R}_{kj}(\mu_{ij})$$

4.4 Finite vs Infinite pCHAMs

In the foregoing sections we have assumed that the multiplicity of molecules is bounded. However we can drop this finiteness condition and work instead with

infinite matrices **E**, **C** and **D**. These correspond to so called *projections* (**E**), and *shift operators* (**C** and **D**).

We can consider instead of $\mathcal{V}(\mathcal{N})$ the (Banach) space of infinite sequences with bounded p-norm, i.e.

$$\ell^p(\mathcal{M}) = \left\{ \sum_{i=0}^{\infty} x_i S_i \mid x_i \in \mathbb{R}, S_i \in \mathcal{M}, (\sum_{i=0}^{\infty} |x_i|^p)^{\frac{1}{p}} < \infty \right\} \subseteq \mathcal{V}(\mathbb{N})$$

The infinite matrices **E**, **C** and **D** represent bounded (and therefore continuous) operators. In particular, we can take the Hilbert space $\ell^2(\mathcal{M})$ and directly recast our finite-dimensional framework in this setting. To a certain degree this framework is even more convenient than the finite-dimensional one; for example we have $\mathbf{CD} = \mathbf{I}$, which is not the case in finite dimensions.

The construction of **T** follows the same recipe as before. We also observe that if we start with any initial solution μ_0 which can be represented by a vector in $\ell^2(\mathcal{M})$ — which is obviously the case for a point distribution — we can guarantee that the iterations $\mathbf{T}^n(\mu)$ will stay in $\ell^2(\mathcal{M})$.

5 Encoding Probabilistic Linda Languages

The CHAM model can be used to describe the operational semantics of various calculi like CCS, the π-calculus, and the Linda calculus which is at the base of several coordination languages, see e.g. [2]. Probabilistic versions of such calculi can be modelled via the pCHAM. We will concentrate here on the encoding of probabilistic Linda-like languages which we will then use as a base for demonstrating our approach to define and measure language expressiveness.

We consider a family of languages $\mathcal{L}(X)$ which differ from one another for the set X of communication primitives used. These primitives correspond to the basic Linda primitives for adding a token to a shared data-space, getting it from the data-space, and checking for its presence or absence in the data-space. The languages $\mathcal{L}(X)$ also include standard prefix and a probabilistic choice operator.

The syntax of $\mathcal{L}(X)$ is formally defined by the following grammar:

$$P ::= \mathbf{stop} \mid C.P \mid P \mid P \mid P +_p P$$
$$C ::= \mathbf{ask}(t) \mid \mathbf{tell}(t) \mid \mathbf{get}(t)$$

where t is a generic element called *token* in a denumerable set $\mathcal{D}$, P is a process and C a communication action (or prefix); we denote by $\mathcal{P}$ the set of all processes. The parameter X defining a Linda-like language $\mathcal{L}(X)$ is a subset of the primitives defined by C.

A program in $\mathcal{L}(X)$ is therefore either an inactive, trivial program **stop**, or a sequential composition $C.P$ or a probabilistic choice $P +_p P$. As usual we omit a trailing **stop** if it is prefixed by a non-empty sequence of basic actions C.

A pCHAM encoding for $\mathcal{L}(X)$ is defined by specifying the set of molecules as $\mathcal{D} \cup \mathcal{P}$ and the following molecule transformation, i.e. specific rules (cf also [11]):

(i) $P_1 \mid P_2 \longrightarrow_1 P_1, P_2$
(ii) $P_1, P_2 \longrightarrow_1 P_1 \mid P_2$
(iii) $P_1 +_p P_2 \longrightarrow_p P_1$
(iv) $P_1 +_p P_2 \longrightarrow_{1-p} P_2$
(v) $\mathbf{stop} \longrightarrow_1 \mathbf{stop}$
(vi) $\mathbf{tell}(t).P \longrightarrow_1 P, t$
(vii) $\mathbf{ask}(t).P, s \longrightarrow_1 P, s$ if $t = s$
(viii) $\mathbf{get}(t).P, s \longrightarrow_1 P$ if $t = s$

As with any pCHAM these rules give rise to a set of *rule operators* $\mathbf{R}_i$. All possible executions of $\mathcal{L}(X)$ programs are choices between or sequential application of these rules. In other words, all possible "behaviours" of $\mathcal{L}(X)$ programs are linear combinations and products of the rule operators $\mathbf{R}_i$. The possibilities of programs in a language $\mathcal{L}(X)$ are thus reflected in the structure of the algebra $\mathcal{A}(X)$ which is generated by (linear combinations and products of) the rule operators $\mathbf{R}_i$ of the pCHAM for $\mathcal{L}(X)$.

Example 11. To illustrate this let us construct the algebra $\mathcal{A}(\mathbf{tell}, \mathbf{ask}, \mathbf{get})$ for a "bounded" version version of the language $\mathcal{L}(\mathbf{tell}, \mathbf{ask}, \mathbf{get})$. We will allow only one type of token t, i.e. $\mathcal{D} = \{t\}$, which appears only with multiplicity $0, 1$ or 2.

The linear operator semantics of this language is given by operators on $\mathcal{V}(\mathcal{P}) \otimes \mathcal{V}(\{\{\!|\}, \{\!|t|\!\}, \{\!|t, t|\!\}\})$. To keep things as simple as possible we will concentrate only on the behaviour of the store, i.e. the possible transformations on $\mathcal{V}(\{\{\!|\}, \{\!|t|\!\}, \{\!|t, t|\!\}\})$ and ignore how the processes themselves change.

The operators corresponding to the rules for **tell** and **get** and the guard in the **ask** rule are given by:

$$\mathbf{T} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{G} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The **stop** rule is implemented by the identity matrix $\mathbf{I}$. The other four rules do not directly influence the store component. However, they allow the combination of basic operators via linear combination (the choice rules) and product (via the parallel rules). The possible "behaviours" are therefore linear combination and products of these basic operators, i.e. we have to look at the algebras generated by some subset of $\{\mathbf{A}, \mathbf{G}, \mathbf{T}, \mathbf{I}\}$.

We can show that $\{\mathbf{A}, \mathbf{G}, \mathbf{T}, \mathbf{I}\}$ generate the full 3×3 matrix algebra $M(3)$. To do this it is sufficient to show that we can construct all matrix units (cf. proof of Proposition 7) in $M(3)$ using the basic matrices $\mathbf{A}$, $\mathbf{G}$, $\mathbf{T}$ and $\mathbf{I}$. Any other matrix in $M(3)$ is then a linear combination of the matrices $\mathbf{E}_{nm}$. For example one can easily check that the following holds:

$$\begin{array}{lll} \mathbf{B}_{11} = \mathbf{I} - \mathbf{A}, & \mathbf{B}_{12} = (\mathbf{I} - \mathbf{A})\mathbf{T}, & \mathbf{B}_{13} = (\mathbf{I} - \mathbf{A})\mathbf{TT} \\ \mathbf{B}_{21} = (\mathbf{I} - \mathbf{G})\mathbf{ATGG}, & \mathbf{B}_{22} = (\mathbf{I} - \mathbf{G})\mathbf{ATG}, & \mathbf{B}_{23} = (\mathbf{I} - \mathbf{G})\mathbf{AT} \\ \mathbf{B}_{31} = \mathbf{GAG}, & \mathbf{B}_{32} = \mathbf{GA}, & \mathbf{B}_{33} = \mathbf{GAT} \end{array}$$

It might be worth noting that we extensively used the negative of $\mathbf{A}$, i.e. the testing for the absence of the token. One might therefore argue that the algebra $\mathcal{A}(X)$ contains not only the effective behaviours of programs in $\mathcal{L}(X)$ but also artificial ones. However, for the method for comparison of the expressiveness of languages we introduce in the next section, $\mathcal{A}(X)$ is a useful approximation.

6 Expressiveness

The pCHAM provides a uniform encoding for several probabilistic languages and calculi, in that it is a common abstract semantics on the base of which the observable behaviour of programs/processes in different languages/calculi can be specified. It is therefore reasonable to utilise such an encoding to compare the expressive power of one language relative to another. The idea is similar to the notion of embedding introduced in [12] and later refined in [13]: given two probabilistic languages Γ_1 and Γ_2 we first define their associated pCHAMs and then we compare them by checking whether or not one can "simulate" the other. The difference with the original notion of embedding is that we do not need to specify any particular observation criteria nor to compile a program in a language into a program in the other language; rather we encode both languages into the same kind of abstract semantics and compare "how many" computations can be performed by the first abstract machine which cannot be performed by the second. In fact, the linear operator semantics of the pCHAM also allows us to determine the "size" of the space of "behaviours" generated by a certain set of specific rules and thus provide obstructions which prevents the embedding or simulation of one pCHAM into another without constructing a counter-example.

This approach can also be adopted to compare the expressive power of non-probabilistic languages: we just need to consider the restriction of our linear operators to point distributions.

Example 12. Assume a finite set of molecules $\{m_1, \ldots, m_t\}$ without a bound on the multiplicity s. If the specific rules of a (p)CHAM are of the form:

$$m_{i1}, \ldots, m_{ik} \longrightarrow m_{j1}, \ldots, m_{jl} \text{ or } m_{i1}, \ldots, m_{ik} \longrightarrow_p m_{j1}, \ldots, m_{jl}$$

such that $l > k$ it is immediately clear that in each reaction the size of the solution is monotonically increasing. Such a (p)CHAM therefore is unable to, for example, purge a solution from all multiples of a certain molecule $\{\!|m_1, m_1, m_1|\!\} \Longrightarrow \{\!|m_1|\!\}$. It is also unable to embed a (p)CHAM whose specific rules include a rule like $m_2, m_4 \longrightarrow m_2$.

Let Γ be a probabilistic calculus and consider an operational semantics for Γ defined via a set of states $\mathcal{C}(\Gamma)$ and a transition relation $\rightarrow_p$ on $\mathcal{C}(\Gamma)$. By an *encoding* e of Γ into a pCHAM $pCHAM(\Gamma) = (\mathcal{M}, \mathcal{R}, \mu_0)$ we mean a (total) function:

$$e : \mathcal{C}(\Gamma) \rightarrow \mathcal{M}$$

which associates to every state of Γ a solution of $pCHAM(\Gamma)$ such that transitions in Γ correspond to transitions in $pCHAM(\Gamma)$, i.e. $C_1 \rightarrow_p C_2$ implies $e(C_1) \Longrightarrow_p e(C_2)$. We omit a more formal definition (which might impose additional constraints and correctness conditions on e) as we will consider here only the encoding of the Linda-like languages we presented in Section 5.

In order to compare the expressiveness of languages we could, on one hand, introduce a notion of an *embedding of languages* following directly Shapiro's approach [12, 13]. Our idea, on the other hand, is to compare languages on the base

of the possible behaviours of their associated pCHAMs. In order to do this one could develop a general notion of *pCHAM embeddings*, or a bit more concretely, address the question when pCHAMs, which are the result of the encoding of calculi or programming languages, can simulate each others behaviour. However, we will go one step further and base our notion of embedding not on the pCHAMs themselves but instead on the linear operator semantics. Our notion of embedding is illustrated by the following diagram:

$$\begin{array}{ccccc} \Gamma_1 & \xrightarrow{e_1} & pCHAM(\Gamma_1) & \approx & \mathcal{A}(\Gamma_1) \\ & & & & \uparrow \mathbf{N} \\ \Gamma_2 & \xrightarrow[e_2]{} & pCHAM(\Gamma_2) & \approx & \mathcal{A}(\Gamma_2) \end{array}$$

where e_1 and e_2 are the pCHAM *encodings* of Γ_2 and Γ_1 respectively, $\mathcal{A}(\Gamma_1)$ and $\mathcal{A}(\Gamma_2)$ are the linear algebras generated from the rules in $pCHAM(\Gamma_1)$ and $pCHAM(\Gamma_2)$ respectively, and $\mathbf{N}$ is a map $\mathbf{N} : \mathcal{A}(\Gamma_2) \to \mathcal{A}(\Gamma_1)$ which implements the embedding.

In principle this embedding could be as complicated as one wants. If the two pCHAMs in question encode Turing complete calculi then it is always possible to encode one in the other in some way. However, we will consider only "reasonable" encodings which respect the structure of the calculi and their pCHAMs, i.e. encodings which are somehow compositional (on the molecular level). We will therefore concentrate our attention only on particular linear maps $\mathbf{N}$ between $\mathcal{A}(\Gamma_2)$ and $\mathcal{A}(\Gamma_1)$.

Definition 13. A *linear embedding* of a linear algebra $\mathcal{A}_1$ into a linear algebra $\mathcal{A}_2$ is an injective algebra homomorphism, i.e. a linear and product preserving map $\mathbf{N} : \mathcal{A}_1 \to \mathcal{A}_2$.

A linear embedding of a calculus Γ_1 into another one Γ_2 is given by a linear embedding of the corresponding algebras $\mathcal{A}(\Gamma_1)$ into $\mathcal{A}(\Gamma_2)$.

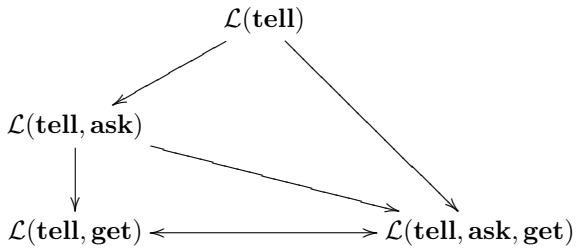
There are further restrictions one could impose, e.g. that the pCHAMs and their algebras reflect the structure of the calculi in a particular way, etc. A particular situation is the comparison of sub-calculi of a given calculus; in this case e_1 and e_2 are the same function.

Example 14. A language $\mathcal{L}(X)$ can be embedded into a language $\mathcal{L}(Y)$ iff $\mathcal{A}(X)$ can be embedded in $\mathcal{A}(Y)$, cf also [14]. A simple property of the algebras $\mathcal{A}(X)$ and $\mathcal{A}(Y)$ like their dimension can give us a criterion to decide whether $\mathcal{A}(X)$ can be embedded into $\mathcal{A}(Y)$: if $\dim(\mathcal{A}(X)) > \dim(\mathcal{A}(Y))$ then it is impossible to embed $\mathcal{A}(X)$ into $\mathcal{A}(Y)$. However, it is not correct to conclude the opposite, i.e. $\dim(\mathcal{A}(X)) \leq \dim(\mathcal{A}(Y))$ does not necessarily imply that there is a "reasonable" embedding of $\mathcal{L}(X)$ into $\mathcal{L}(Y)$.

Considering the Linda-like languages from the previous section we can show, for example, that $\dim(\mathcal{A}(\mathbf{tell}, \mathbf{ask})) > \dim(\mathcal{A}(\mathbf{tell}))$ and $\dim(\mathcal{A}(\mathbf{tell}, \mathbf{get})) > \dim(\mathcal{A}(\mathbf{tell}))$ which means that it is impossible to embed either $\mathcal{L}(\mathbf{tell}, \mathbf{get})$

nor $\mathcal{L}(\mathbf{tell}, \mathbf{ask})$ in $\mathcal{L}(\mathbf{tell})$. More concretely, one can show that $\mathcal{L}(\mathbf{tell}, \mathbf{ask})$ generates only the sub-algebra of upper triangular 3×3 matrices $U(3)$. We can embed this sub-algebra into the full matrix algebra $M(3)$ by taking $\mathbf{N}$ the identity restricted to the sub-algebra $U(3)$, but not vice versa. An obstruction against the embeddability of $M(3)$ into $U(3)$ is the fact that the dimensions are incompatible, i.e. $\dim(M(3)) = 9$ while $\dim(U(3)) = 6$.

This is consistent with the hierarchy of languages in, e.g., [14]:



7 Conclusions

We presented a probabilistic version of the Chemical Abstract Machine. The CHAM and its probabilistic version pCHAM provide a basic and simple framework for comparing various probabilistic (or quantitative) calculi. The pCHAM model is based on a particular, discrete time Markov Chain (DTMC) model in which a scheduler decides at each time step on the probability that any of the applicable rules or reactions gets executed.

It will be interesting to investigate different, perhaps more general execution models for a pCHAM: For example, one could allow the scheduler to execute at each time step not only a single rule but any number of rules as long as they are not in "conflict"; this makes it necessary to develop a probabilistic mechanism for resolving such conflicts. Another line of further work will be devoted to the formulation of a continuous time Markov Chain (CTMC) model which we can define via so-called $\mathbf{Q}$ matrices which themselves generate transition matrices as $\mathbf{T}_t = \exp(t\mathbf{Q})$; in this model the chance that any two rules "fire" simultaneously is zero and conflicts between rules are therefore not a problem.

Finally, we plan a closer investigation of the relation between (general) discrete and continuous time models and of expressiveness issues regarding the pCHAM encodings of synchronous versus asynchronous calculi, see e.g. [15].

References

1. Banâtre, J.P., Le Métayer, D.: The gamma model and its discipline of programming. Science of Computer Programming **15** (1990) 55–77
2. Banâtre, J.P., Fradet, P., Le Métayer, D.: Gamma and the chemical reaction model: Fifteen years after. In Calude, C., ed.: Multiset Processing. Volume 2235 of Lecture Notes in Computer Science., Springer Verlag (2001) 17–44

3. Fradet, P., Le Métayer, D.: Structured gamma. Science of Computer Programming **31** (1998) 263–289
4. Le Métayer, D.: Higher-order multiset programming. In: DIMACS workshop on specifications of parallel algorithms. Volume 18 of Dimacs series in Discrete Mathematics., American Mathematical Society (1994)
5. Andreoli, J.M., Hankin, C., Le Métayer, D.: Coordination Programming. Imperial College Press, London (1996)
6. Berry, G., Boudol, G.: The chemical abstract machine. Theoretical Computer Science **96** (1992) 217–248
7. Jonsson, B., Yi, W., Larsen, K.: 11. In: Probabilistic Extensions of Process Algebras. Elsevier Science, Amsterdam (2001) 685–710 see [16].
8. 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
9. Grimmett, G., Stirzaker, D.: Probability and Random Processes. second edn. Clarendon Press, Oxford (1992)
10. Parzen, E.: Stochastic Processes. second edn. Classics in Applied Mathematics. SIAM (1999)
11. Ciancarini, P., Jensen, K., Yankelevich, D.: On the operational semantics of a coordination language. In Ciancarini, P., Nierstrasz, O., Yonezawa, A., eds.: ECOOP Workshop. Volume 924 of Lecture Notes in Computer Science., Springer Verlag (1995) 77–106
12. Shapiro, E.: Embeddings among concurrent programming languages. In Cleaveland, W.R., ed.: Proceedings CONCUR 92, Stony Brook, NY, USA. Volume 630 of Lecture Notes in Computer Science., Springer-Verlag (1992) 486–503
13. de Boer, F.S., Palamidessi, C.: Embedding as a tool for language comparison. Information and Computation **108** (1994) 128–157
14. Brogi, A., Di Pierro, A., Wiklicky, H.: Linear embedding for a quantitative comparison of language expressiveness. In: QAPL'01 — ACM Workshop on Quantitative Aspects of Programming Languages. Volume 59:3 of ENTCS., Elsevier (2002)
15. Palamidessi, C.: Comparing the expressive power of the synchronous and the asynchronous pi-calculus. Mathematical Structures in Computer Science **13** (2003) 685–719
16. Bergstra, J., Ponse, A., Smolka, S., eds.: Handbook of Process Algebra. Elsevier Science, Amsterdam (2001)
17. Palmer, T.: Banach Algebras and The General Theory of $*$-Algebras – Volume I: Algebras and Banach Algebras. Volume 49 of Encyclopedia of Mathematics and Its Applications. Cambridge University Press, Cambridge – New York (1994)
18. Kadison, R., Ringrose, J.: Fundamentals of the Theory of Operator Algebras: Volume I — Elementary Theory. Volume 15 of Graduate Studies in Mathematics. American Mathematical Society, Providence, Rhode Island (1997) reprint from Academic Press edition 1983.
19. Fillmore, P.A.: A User's Guide to Operator Algebras. John Wiley & Sons, New York — Chicester (1996)
20. Wegge-Olsen, N.: K-Theory and C*-Algebras — A Friendly Approach. Oxford University Press, Oxford (1993)

A Tensor Products

The tensor product plays a central role in our discussion. For the convenience of the reader we therefore recall some of the important facts about the tensor product of vectors, (Hilbert) spaces, operators, etc.

Let $\mathcal{V}_1$, $\mathcal{V}_2$, ..., $\mathcal{V}_n$ and $\mathcal{W}$ be linear spaces. A map $f : \mathcal{V}_1 \times \mathcal{V}_2 \times \ldots \times \mathcal{V}_n \to \mathcal{W}$ is called *multi-linear* if f is linear in each of its arguments. We denote by $\mathcal{L}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n; \mathcal{W})$ the set of multi-linear maps. The algebraic tensor product of vector spaces is defined via a universal property as follows (see e.g. Definition 1.10.1 in [17]).

Definition 15. The algebraic tensor product of vector spaces $\mathcal{V}_1$, $\mathcal{V}_2$, ..., $\mathcal{V}_n$ is given by a vector space $\bigotimes_{i=1}^n \mathcal{V}_i$ and a map $p = \otimes_{i=1}^n \in \mathcal{L}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n; \bigotimes_{i=1}^n \mathcal{V}_i)$ such that if $\mathcal{W}$ is any vector space and $f \in \mathcal{L}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n; \mathcal{W})$ then there exists a unique map $h : \bigotimes_{i=1}^n \mathcal{V}_i \to \mathcal{W}$ satisfying $f = h \circ p$.

This algebraic construction is sufficient for finite dimensional vector spaces. It is easy to show that in the finite dimensional case we have: $\mathcal{V}(X \times X) \cong \mathcal{V}(X) \otimes \mathcal{V}(X)$. In the infinite dimensional case one has to consider also topological aspects; for example, the algebraic tensor product of Hilbert spaces does not form in general a Hilbert space. Without going into the details — see for example [18], [19] or Appendix T in [20] — it is however possible to construct from the algebraic tensor product of Hilbert spaces $\mathcal{H}_1$, $\mathcal{H}_2$, ..., $\mathcal{H}_n$, a Hilbert space which is the tensor product $\bigotimes_{i=1}^n \mathcal{H}_i$.

The following results summarise important properties of the tensor product:

Proposition 16. *If $\mathcal{H}_1, \ldots, \mathcal{H}_n$ are Hilbert spaces and $\mathbf{A}_i \in \mathcal{B}(\mathcal{H}_i)$ with $i = 1, \ldots, n$ bounded linear operators, then there exists a unique bounded linear operator $\mathbf{A} \in \mathcal{B}(\mathcal{H}_1 \otimes \ldots \otimes \mathcal{H}_n)$ such that:*

$$\mathbf{A}(x_1 \otimes \ldots \otimes x_n) = \mathbf{A}_1(x_1) \otimes \ldots \otimes \mathbf{A}_n(x_n).$$

for all $x_i \in \mathcal{H}_i$ and we write $\mathbf{A} = \mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n$.

Proposition 17. *The tensor product of (bounded) linear operators $\mathbf{A}_1$, $\mathbf{A}_2$, ..., $\mathbf{A}_n$ (on Hilbert spaces) is associative and has the following properties:*

(i) $(\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n)(\mathbf{B}_1 \otimes \ldots \otimes \mathbf{B}_n) = (\mathbf{A}_1\mathbf{B}_1 \otimes \ldots \otimes \mathbf{A}_n\mathbf{B}_n)$
(ii) $\mathbf{A}_1 \otimes \ldots \otimes (\alpha\mathbf{A}_i) \otimes \ldots \otimes \mathbf{A}_n = \alpha(\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_i \otimes \ldots \otimes \mathbf{A}_n)$
(iii) $\mathbf{A}_1 \otimes \ldots \otimes (\mathbf{A}_i + \mathbf{B}_i) \otimes \ldots \otimes \mathbf{A}_n = \mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_i \otimes \ldots \otimes \mathbf{A}_n + \mathbf{A}_1 \otimes \ldots \otimes \mathbf{B}_i \otimes \ldots \otimes \mathbf{A}_n$
(iv) $(\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n)^* = \mathbf{A}_1^* \otimes \ldots \otimes \mathbf{A}_n^*$
(v) $\|\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n\| = \|\mathbf{A}_1\| \ldots \|\mathbf{A}_n\|$

For a proof of these properties see e.g. discussions and remarks following Proposition 2.6.12 in [18].