JFP **15** (5): 703–749, 2005. © 2005 Cambridge University Press
doi:10.1017/S0956796805005599 Printed in the United Kingdom

Quantitative static analysis of distributed systems

ALESSANDRA Di PIERRO
University of Pisa, Pisa, Italy
(*e-mail:* dipierro@di.unipi.it)

CHRIS HANKIN and HERBERT WIKLICKY
Imperial College London, London, UK
(*e-mail:* clh@doc.ic.ac.uk)

Abstract

We introduce a quantitative approach to the analysis of distributed systems which relies on a linear operator based network semantics. A typical problem in a distributed setting is how information propagates through a network, and a typical qualitative analysis is concerned with establishing whether some information will eventually be transmitted from one node to another node in the network. The quantitative approach we present allows us to obtain additional information such as an estimation of the probability that some data is transmitted within a given interval of time. We formalise situations like this using a probabilistic version of a process calculus which is the core of KLAIM, a language for distributed and mobile computing based on interactions through distributed tuple spaces. The analysis we present exploits techniques based on Probabilistic Abstract Interpretation and is characterised by compositional aspects which greatly simplify the inspection of the nodes interaction and the detection of the information propagation through a computer network.

1 Introduction

With increasing faults and attacks on the Internet infrastructure and with society moving increasingly towards reliance on "e", there is an urgent need to develop techniques to analyse network and service vulnerability under organised attacks or with respect to unintentional faults. Within the last year, the world had to recover from at least three major "e-epidemics". One concerned the SoBig.F virus that has led to our mailboxes being cluttered with junk (but dangerous) mail; the Economist (30 August 2003) estimated that some 500,000 machines were infected at the peak of the epidemic. The second one involved the Blaster worm which was intended to mount a distributed denial of service attack on a Microsoft web page. Most recently the Mydoom worm has been in the news. These three were just the latest in a long line of high profile viruses/worms to infect the Internet.

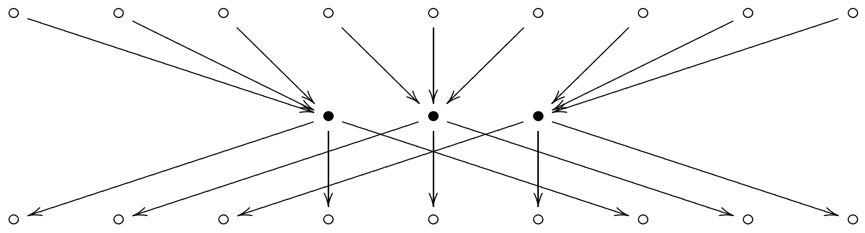
In this paper we propose a framework for network (vulnerability) analysis where static analysis techniques are used to predict the properties of a distributed system. These techniques are based on a formalisation of the system in a suitable programming language provided with a mathematically rigorous operational semantics. A novel aspect of this analysis is to be found in its *quantitative approach*. In recent years, such an approach is gaining more prominence over the traditional qualitative one. In particular, aspects related to the probabilistic description and analysis of systems seem to become important to allow for a more realistic analysis. This is due to several factors. One of these is the difficulty or impossibility to obtain an exact description or analysis – for example because of the size or interconnectedness of these systems – while an approximate solution to these problems looks achievable; another reason is the need to guarantee not only functional but also non-functional properties of systems: besides the question of whether a certain service will (eventually) be provided, it is practically useful to ask also for a certain performance level, e.g. that the response time for a database enquiry is within certain limits.

Computer security is where a quantitative approach has proved to be particularly well-suited. Since perfect security is often unachievable, a more realistic way to address security problems would ask questions such as how much protection effort leads to what kind of security level, e.g. expected average damage. This leads to a quantitative risk or vulnerability analysis rather than an absolute security certification.

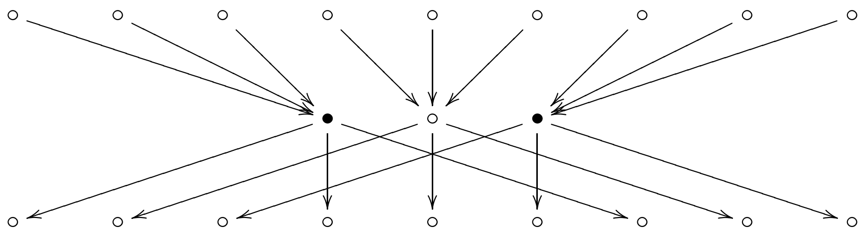
In previous work (Di Pierro *et al.*, 2002; Di Pierro *et al.*, 2004a; Di Pierro *et al.*, 2003) we have been looking into the issue of quantifying the confinement level of concurrent systems based on a quantitative analysis of their behaviour. The aim of this paper is to extend this approach to distributed systems by developing a semantics based framework for the quantitative analysis of computer networks. As an example we will look at the problem of how information is being disseminated in distributed systems, addressing not only the problem whether some information will eventually be transmitted from one node in a network to another one, but also the probability that this will happen within a given number of time steps.

If one thinks of the information transmitted as of some malicious code – a computer virus or worm – then this kind of quantitative analysis effectively addresses the question of how fast a computer virus with a certain infection mechanism will spread in a network with a given topology. The intention is that such an analysis will allow for the identification of weaknesses of certain network topologies with respect to some infection mechanisms.

As an illustrative example, consider the scenario described by the following network (assuming that all nodes in each layer are fully connected to each other):



Obviously, if the three nodes in the middle layer are virus-resistant then the infection will never spread from the top network to the lower network. If one node in the middle layer can be infected, then it is a matter of time until the infection spreads to the lower part. The question is how long it will take in the average. If there are two un-protected nodes in the middle layer then it should go faster, etc.



This scenario reflects quite closely, although in a simplified way, one of the main problems with the structure of today's worldwide network. As reported in Albert *et al.* (2000), a study has shown that the Internet's reliance on a few key nodes makes it especially vulnerable to organised attacks by hackers and terrorists. The rate at which the virus spreads is essential in a real world situation as it might determine the time given to system managers to react and to put counter-measures in place.

In order to answer quantitative questions such as the one in the above example we will use static analysis techniques based on Probabilistic Abstract Interpretation (PAI) (Di Pierro & Wiklicky, 2000). This requires that we represent the collecting semantics of a network as a linear operator. Starting from such a concrete semantics – usually represented by a highly complex matrix – the PAI technique allows us to define a simplified abstract, so-called induced semantics on which we can base our analysis with a guarantee of optimality: the abstract results we get are the *closest* in quantitative terms to the concrete ones. This is guaranteed by the use of a mathematical structure (the Moore-Penrose pseudo-inverse) which can be seen as the analogue of a Galois connection – which is essential in the formulation of classical Abstract Interpretation (Cousot & Cousot, 1977) – in a linear algebraic setting. The possibility of defining a "safe" abstract semantics is particularly essential in a distributed setting where the presence of interaction and communication mechanisms between different local semantics makes it intrinsically quite complex to analyse the semantics of the overall network.

We will show how the use of a particular construction, namely the tensor product, for modelling probabilistic node composition at the concrete level allows for a "compositional" definition of the abstract induced semantics which greatly simplifies the analysis: Each single node can be analysed individually, while the properties of the tensor product and of the PAI technique ensure that we can get a correct abstract analysis of the overall network by simply composing the abstract individual analysis of each node.

The paper is organised as follows. We start with a formal description of a network of active nodes. Rather than invent a new language for this purpose we have adopted KLAIM (Kernel Language for Agents Interaction and Mobility) (De Nicola *et al.*, 1998). This is an experimental language that was designed for programming distributed systems. We identify a simple core language and propose a probabilistic extension to this core. The language together with its Structural Operational Semantics is presented in the next section.

We introduce some background material and present the linear operator semantics for pcKLAIM in sections 3 and 4. Probabilistic Abstract Interpretation is introduced in section 5. The semantics of networks is expressed using tensor products; in section 6 we investigate how tensors and probabilistic abstract interpretation interact and show that nodes can be individually analysed and the results then combined without loss of precision – this provides the basis for a compositional approach to analysis. The final main section (section 7) of the paper constructs an analysis of the simple infection scenario discussed above.

2 Probabilistic core KLAIM

We define a probabilistic version of core KLAIM (Bettini *et al*., 2003), a restricted version of KLAIM (De Nicola *et al*., 1998), which has no higher order features and simple parameter passing rather than the more sophisticated, Linda-like pattern matching mechanism of the full language. A probabilistic version of the full KLAIM language (without sub-nets) has been presented in Di Pierro *et al.* (2004c).

The original KLAIM language was designed for the programming of distributed systems. A KLAIM program is a network of located processes and data. The located data represents a distributed tuple space. The fact that we will not consider higher order features for the time being is part of the reason for our decision to use KLAIM as the basis for our formalisation and not various probabilistic versions of, for example, the π-calculus (Priami, 1995; Herescu & Palamidessi, 2000).

In our restricted model, processes can only **in**(put) and **out**(put) data and nodes are only capable of storing a single datum. Thus each node may be considered to be a pair consisting of a process and a store which is either empty or contains a single "token". In this restricted setting the data are *localities*, that is network references of nodes or subnets, although in a more realistic calculus structured data might be desirable.

We introduce probabilities in two distinct ways: first through the scheduler at both the network and local level and secondly through probabilistic allocation environments which, at the local level, map locality variables to distributions over localities.

In order to keep things as simple as possible we will consider only a discrete time version, i.e. at each time step only one process will be executed according to the scheduling probability. This leads to a simple discrete Markov chain model. Although this introduces some element of synchronicity, communication is still asynchronous as **in** and **out** actions do not communicate directly with each other.

2.1 Syntax

The syntax of pcKLAIM networks is defined in Table 1, where the data D and processes P are defined in Tables 2 and 3, respectively.

Localities and Locality Names. We call a node *simple* if it is not a subnet. We use l (possibly subscripted or primed) as a metavariable to represent localities, drawn

Table 1. *pcKLAIM network syntax*

N	$::=$	$\|_{i=1}^{m} n_i$	composition
n	$::=$	$l ::_{\varrho}^{p} P$	process node
	$\mid$	$l ::^{p} D$	data node
	$\mid$	$l ::^{p} N$	subnet

Table 2. *pcKLAIM data syntax*

D	$::=$	$\langle\rangle$	empty store
	$\mid$	$\langle l \rangle$	located data

from $\mathscr{L}$ and representing network addresses. We assume that each locality $l \in \mathscr{L}$ is used to indicate at most one physical location; in our syntax this corresponds to the parallel composition of at most one process node and at most one data node, or to a subnet. We allow for the omission of nodes with trivial processes or empty stores, i.e. we avoid to explicitly introduce in a network term nodes of the form $l ::_{\varrho}^{p}$ **nil** and $l ::^{p} \langle\rangle$. Our model allows for subnetted networks, where portions of the network share the same address. Thus, localities are either a simple network reference (as in the original KLAIM) or a compound reference. For two-level networks, a compound locality is of the form $l.l'$, where l refers to a subnet and l' to a node in the subnet. The form of a compound locality reflects the splitting of the host portion of an IP address into a subnet portion and a host portion usually achieved via an address mask. According to the syntax in Table 1 a compound locality $l.l'$ is the locality associated to a node of the form:

$$l :: (\ldots \parallel l' ::_{\varrho} P \parallel \ldots) \quad \text{or} \quad l :: (\ldots \parallel l' :: D \parallel \ldots).$$

Analogously, for a simple node in a k-level subnetted network, the associated compound locality is of the form $l.l_1.l_2.\cdots.l_{k-1}$, meaning that the node is in a subnet at l_{k-2} which is part of a subnet at l_{k-3}, and so on.

Locality variables, $\mathscr{U}$, are ranged over by x, y, z. Locality names, $\mathscr{N} = \mathscr{L} \cup \mathscr{U}$, are represented by ℓ (again possibly subscripted or primed).

Scheduling Probabilities. The global scheduling probability of the nodes in a (sub-)network, as well as the local scheduling probability of parallel processes running at the same node, and the local probabilistic choice of the locality addressed by an action are defined as distributions on localities, i.e. functions $d : \mathscr{L} \to [0,1]$ which we denote by

$$d = \{\langle l_1, p_1 \rangle, \ldots, \langle l_n, p_n \rangle\}$$

with $p_i \in [0,1]$ and $\sum_{i=1}^{n} p_i = 1$. The probability of scheduling a node or a subnet is specified by the superscript p in the syntax of nodes. If a process node $l ::_{\varrho}^{p} P$ and a data node $l ::^{p} D$ share the same locality l then they must also have the same probability p. For a given net $N = \|_{i=1}^{m} n_i$ the probabilities p_i associated to the nodes n_i define a probability distribution on localities. We assume that $\sum_{i=1}^{m} p_i = 1$; this

Table 3. *pcKLAIM process syntax*

P	$::=$	**nil**	null process
	$\mid$	**delay**.P	delay action prefix
	$\mid$	**out**$(\ell')@\ell.P$	output action prefix
	$\mid$	**in**$(x)@\ell.P$	input action prefix
	$\mid$	$\|_{i=1}^{n}\ q_i : P_i$	probabilistic parallelism
	$\mid$	A	process call

can always be achieved by (static) normalisation. We will write $N[d]$ to denote a network N with a prescribed distribution d on its localities.

We call $\mathscr{D}$ the set of all locality distributions and we denote by u the uniform distribution. For a network $N[u]$ we will sometimes use a shorthand notation where we omit the superscript p in the specification of nodes.

Allocation environments ϱ associate distributions over localities to locality variables, i.e. $\varrho : \mathscr{U} \to \mathscr{D}$. We extend ϱ to a function on $\mathscr{N}$ by defining $\varrho(l)$ for $l \in \mathscr{L}$ as the distribution which associates 1 to the locality l and 0 to any other locality in $\mathscr{L}$. It is useful to introduce the following *insert* operation on distributions over $\mathscr{N}$, which defines the composition of distributions associated to nested sub-nets. Let $\varrho = \{\langle s_i, p_i\rangle\}_{i\in I}$ and $\sigma = \{\langle t_j, q_j\rangle\}_{j\in J}$ be two distributions over $\mathscr{N}$. Then we define $\sigma \rhd_z \varrho$ as

$$(\sigma \rhd_z \varrho)(s_k) = \begin{cases} \varrho(s_k) & \text{if } k \in I \setminus \{z\} \\ \varrho(s_k)\sigma(t_j) & \text{if } k = z \text{ and } \ j \in J. \end{cases}$$

This operation defines the probability associated to a compound locality $l_1.l_2.\cdots.l_k$ as the product of the probabilities associated to the single localities l_i, $i = 1, 2, \ldots, k$. In fact, if ϱ is the distribution associated to a network N and $\sigma = \{\langle t_j, q_j\rangle\}_{j\in J}$ the distribution associated to a subnet N' of N located at s_z, then $\sigma \rhd_z \varrho$ defines the probability of the compound localities $s_z.t_j$, for all $j \in J$.

Processes. The syntax of pcKLAIM process terms is given in Table 3. The parameter x in an input action prefix will be bound to a locality when the action succeeds, that is when some data is present at the node associated to the selected locality ℓ. Symmetrically, an output action succeeds when the addressed node contains the empty store. The parameter ℓ' of an **out** can be either a (simple or compound) locality or any prefix of a compound locality; in this case the information sent is a subnet reference rather than a simple node address. Alternatively, ℓ' in an **out** might be a bound variable, that is a variable x which occurs in an action prefix **in**$(x)@\ell.P$. We will also omit the target address in an **out** or an **in** process when it refers to the same node where the process is located; thus, the notation **out**(x) and **in**(x) will stand for **out**$(x)@$**self** and **in**$(x)@$**self**, where **self** is (a reference to) the locality of the node the process is executed at. Moreover, we will write a^{ℓ} to refer to an action of the form **out**$(\ell')@\ell$ or **in**$(x)@\ell$.

The probabilistic parallel operator schedules the sub processes according to the probabilities q_i. We do not require that the numbers q_i specify directly a distribution: normalisation procedures which we introduce later will achieve this. This allows programmers to specify "relative scheduling priorities", e.g. $1 : P_1 \mid 2 : p_2 \mid \ldots$ specifies that the scheduler will select P_2 with twice the probability of P_1 (provided both processes are "active") independently of other processes which might run in parallel with these two. The process call syntax admits the possibility of defining named parameterless processes which may then be invoked – invocation is essentially macro substitution.

In order to define the semantics of a pcKLAIM program, it is useful to introduce the notion of a network in normal form. A pcKLAIM network term is in *normal form* if there is no local parallelism and there are no subnets. More formally this is defined as follows.

Definition 1

A pcKLAIM network term $N = (\|_{i=1}^{m} n_i)[d]$ is in *normal form* if for all i, n_i is a simple node, that is $n_i \equiv l ::_{\varrho_i} Q$ or $n_i \equiv l :: D$, and processes Q are defined by the restricted syntax

$$Q ::= \mathbf{nil} \mid \mathbf{delay}.Q \mid \mathbf{out}(\ell')@\ell.Q \mid \mathbf{in}(x)@\ell.Q.$$

In the next section we will show that every pcKLAIM network can be reduced to a normal form.

2.2 Operational semantics

In the following we will denote by *env*(n) the environment ρ at node n, and by *addr*(n), *data*(n) and *proc*(n) respectively the locality, the data and the process at node n.

We define the operational semantics of pcKLAIM by means of a structural congruence $\equiv$ and a probabilistic transition relation $\longrightarrow_p$. The first relation allows us to identify two networks which "behave" in the same way although syntactically different and is defined by the rules in Table 4. The second relation represents the dynamical evolution of a pcKLAIM program and is defined by the rules in Table 5.

The congruence rules in Table 4 establish that **nil** is the unit for the probabilistic parallel operator, rule **C1**, that process invocation is a macro substitution, rule **C2**, and that the order of composition of nodes in a network is immaterial, rule **C3**. Rule **C4** defines the distribution resulting from nested probabilistic parallel compositions of subnets. The last rule, **C5**, equates probabilistic parallelism with the creation of a subnet whose nodes contain each a process which was an operand of the parallel operator.

The congruence rules of Table 4 allows us to transform every pcKLAIM network into an equivalent one in normal form.

Proposition 1

For every pcKLAIM network term N there exists a pcKLAIM network term N' in normal form such that $N \equiv N'$.

Table 4. *Congruence on pcKLAIM terms*

C1	$l ::_{\varrho}^{p} P \equiv l ::_{\varrho}^{p} (q_1 : P \mid q_2 : \textbf{nil})$
C2	$l ::_{\varrho}^{p} A \equiv l ::_{\varrho}^{p} P$ with the declaration $A :\!-P$
C3	$\Vert_{i\in S}\, n_i \equiv \Vert_{i\in S}\, n_{\pi(i)}$ for any permutation $\pi : S \rightarrow S$
C4	$(\Vert_{i\in S\setminus\{j\}}\, n_i \parallel l_j :: (\Vert_{k\in T} n_k)[d'])[d] \equiv (\Vert_{i\in S\setminus\{j\}}\, n_i \Vert (\Vert_{k\in T} l_j.l_k :: n_k))[d' \triangleright_j d$
C5	$(\Vert_{i\in S\setminus\{j\}}\, n_i \parallel l_j ::_{\varrho}^{p} P_j)[d] \equiv (\Vert_{i\in S\setminus\{j\}}\, n_i \parallel l_j ::^{p} N_j[d'])[d]$ with $P_j = \Vert_{k=1}^{m}\, q_k : P_k$ and $N_j = \Vert_{k=1}^{m}\, l_k ::_{\varrho}^{q_k} P_k$, l_k fresh localities, and $d'(l_k) = q_k$, for $k = 1 \ldots m$ and $d'(l) = 0$ otherwise

Proof
By using rule **C5**, we transform N in an equivalent network $\bar{N}$ with no local parallelism. Then using **C4** we lift each node in a subnet at level k up to the previous level $k-1$. More formally, we proceed by structural induction. Suppose that $\bar{N} \equiv (\Vert_{i=1}^{m}\, n_i)[d]$ and there exists $j \in [1, m]$ such that $n_j \equiv l :: N''$. Since by the inductive hypothesis N'' is in normal form, by applying **C4** iteratively to $\bar{N}$ (note that we have only finite nesting of sub-networks) we get N' which is also in normal form and equivalent to N. $\square$

The normal form of a network is obviously only unique up to permutation, because of congruence **C3** in Table 4. In the following we will assume a particular normal form of a network obtained by fixing an enumeration of the localities in N'.

Example 1
The following pcKLAIM program

$$l_1 :: \left(\frac{1}{4} : \textbf{out}(l_1)@l_2 \,\middle|\, \frac{3}{4} : \textbf{out}(l_2)@l_2\right) \parallel l_2 :: \langle\rangle$$

represents a network of two nodes. Since the first node contains a parallel process this network is not in normal form, but we can rewrite it as follows:

$$\begin{aligned}
& l_1 :: \left(\frac{1}{4} : \textbf{out}(l_1)@l_2 \,\middle|\, \frac{3}{4} : \textbf{out}(l_2)@l_2\right) \parallel l_2 :: \langle\rangle \equiv \\
&\equiv \left(l_1 :: \left(\frac{1}{4} : \textbf{out}(l_1)@l_2 \,\middle|\, \frac{3}{4} : \textbf{out}(l_2)@l_2\right) \,\middle\Vert\, l_2 :: \langle\rangle\right)[u] \\
&\equiv (l_1 :: (l_1' :: \textbf{out}(l_1)@l_2 \parallel l_2' :: \textbf{out}(l_2)@l_2)[d] \parallel l_2 :: \langle\rangle)[u] \\
&\equiv (l_1.l_1' :: \textbf{out}(l_1)@l_2 \parallel l_1.l_2' :: \textbf{out}(l_2)@l_2 \parallel l_2 :: \langle\rangle)[d']
\end{aligned}$$

with distributions:

$$u(l_1) = \frac{1}{2} \quad \text{and} \quad u(l_2) = \frac{1}{2}$$
$$d(l_1') = \frac{1}{4} \quad \text{and} \quad d(l_2') = \frac{3}{4}$$

The distribution d' is therefore given by

$$\begin{aligned} d'(l_1.l'_1) &= \frac{1}{4}\cdot\frac{1}{2} = \frac{1}{8} \\ d'(l_1.l'_2) &= \frac{3}{4}\cdot\frac{1}{2} = \frac{3}{8} \\ d'(l_2) &= \frac{1}{2}. \end{aligned}$$

The network reduction is described for normalised networks N with associated distribution d by the rules in Table 5 which define the probabilistic transition relation $\longrightarrow_p$. The probability p of a single step transition depends on both the global scheduling d and the local probabilistic choice defined by the environment at each node n in N. If ℓ is the name addressed in a **out** or **in** action a on some node $n \equiv l ::_\varrho a.P$, then executing such an action leads to a new configuration of the whole network with a probability given by the distribution $\tilde{d}(l)\cdot(\tilde{\varrho}(\ell))$ obtained via a *normalisation* process as follows. We will write $n \in N$ to denote that n is a node in the network N, that is $N = (\|_{i=1}^{m} n_i)[d]$ and $n = n_i$ for some $i \in [1, m]$.

Define the set of *active* nodes

$$Active_N = \{n \in N \mid n \equiv l ::_\varrho a^\ell.P \text{ and } Active_a(\varrho,\ell) \neq \emptyset\},$$

where $Active_a(\varrho,\ell)$ is defined for $a = \textbf{delay}$, $a = \textbf{out}$ and $a = \textbf{in}$ respectively by:

$$\begin{aligned} Active_{\textbf{delay}}(\varrho,\ell) &= \{n \in N\} \\ Active_{\textbf{out}}(\varrho,\ell) &= \{n \in N \mid n \equiv l :: \langle\rangle \text{ and } \varrho(\ell)(addr(n)) \neq 0\} \\ Active_{\textbf{in}}(\varrho,\ell) &= \{n \in N \mid n \equiv l :: \langle l'\rangle, l' \in \mathscr{L} \text{ and } \varrho(\ell)(addr(n)) \neq 0\}. \end{aligned}$$

Consider the probability defined by

$$Gprob = \sum\{d(l') \mid l' = addr(n), n \in Active_N\}.$$

Then the normalised distribution $\tilde{d}$ is defined as $\tilde{d}(l) = \frac{d(l)}{Gprob}$.

The normalisation of the distribution $\varrho(\ell)$ assigned to a locality name ℓ by a local environment ϱ depends on the particular action a being performed and is defined in a similar way, namely $(\tilde{\varrho}(\ell))(l) = \frac{\varrho(\ell)(l)}{Lprob}$, where

$$Lprob = \sum\{\varrho(\ell)(l') \mid l' = addr(n), n \in Active_{Out}\}$$

if a is of the form $\textbf{out}(l'')@\ell$, and

$$Lprob = \sum\{\varrho(\ell)(l') \mid l' = addr(n), n \in Active_{In}\}$$

if a is of the form $\textbf{in}(x)@\ell$.

The transition relation $\longrightarrow_p$ is defined on network terms in normal form. We assume that each node may contain at most one process term and/or one data term. We identify $l ::_\rho P \equiv l ::_\rho P \parallel l :: \langle\rangle$ and $l :: \langle l'\rangle \equiv l :: \textbf{nil} \parallel l :: \langle l'\rangle$.

We now present a number of simple examples. We have omitted the allocation environment when the process contains no locality variables.

Table 5. *The transition system for pcKLAIM*

R1	$(\Vert_{i \in S \setminus j}\, n_i \parallel l_j ::_{\varrho} \mathbf{delay}.P)[d] \xrightarrow[\bar{d}(l_j)]{} (\Vert_{i \in S \setminus j}\, n_i \parallel l_j ::_{\varrho} P)[d]$
R2	$(\Vert_{i \in S \setminus \{j,s\}}\, n_i \parallel l_j ::_{\varrho} \mathbf{out}(l')@\ell.P \parallel n_s)[d] \xrightarrow[\bar{d}(l_j)(\bar{\varrho}(\ell))(l_s)]{} (\Vert_{i \in S \setminus \{s,j\}}\, n_i \parallel l_j ::_{\varrho} P \parallel l_s :: \langle l' \rangle)[d]$ if $n_s \in Active_{\mathbf{out}}(\varrho, \ell_j)$ with $addr(n_s) = l_s$
R3	$(\Vert_{i \in S \setminus \{s,j\}}\, n_i \parallel l_j ::_{\varrho} \mathbf{in}(x)@\ell.P \parallel n_s)[d] \xrightarrow[\bar{d}(l_j)(\bar{\varrho}(\ell))(l_s)]{} (\Vert_{i \in S \setminus \{s,j\}}\, n_i \parallel l_j ::_{\varrho} P[l'/x] \parallel l_s :: \langle\rangle)[d]$ if $n_s \in Active_{\mathbf{in}}(\varrho, \ell_j)$, $addr(n_s) = l_s$ and $data(n_s) = \langle l'$
R4	$\dfrac{N \equiv N_1 \quad N_1 \xrightarrow[p]{} N_2 \quad N_2 \equiv N'}{N \xrightarrow[p]{} N'}$

Example 2
Consider the normal form of the network in Example 1

$$N = (l_1.l_1' :: \mathbf{out}(l_1)@l_2 \parallel l_1.l_2' :: \mathbf{out}(l_2)@l_2 \parallel l_2 :: \langle\rangle)[d']$$

where d' is the distribution

$$d' = \left\{ \left\langle l_1.l_1', \frac{1}{8} \right\rangle, \left\langle l_1.l_2', \frac{3}{8} \right\rangle, \left\langle l_2, \frac{1}{2} \right\rangle \right\}.$$

Since $Active_{Out} = \{l_2 :: \langle\rangle\}$, we have that $Active_N = \{l_1.l_1' :: \mathbf{out}(l_1)@l_2, l_1.l_2' :: \mathbf{out}(l_2)@l_2\}$. By applying **R2** the following two reductions of N are possible:

$$N \xrightarrow[p_1]{} (l_1.l_1' :: \mathbf{nil} \parallel l_1.l_2' :: \mathbf{out}(l_2)@l_2 \parallel l_2 :: \langle l_1 \rangle)[d']$$

and

$$N \xrightarrow[p_2]{} (l_1.l_1' :: \mathbf{out}(l_1)@l_2 \parallel l_1.l_2' :: \mathbf{nil} \parallel l_2 :: \langle l_2 \rangle)[d'],$$

where $p_1 = \frac{1/8}{4/8} \cdot 1 = \frac{1}{4}$ and $p_2 = \frac{3/8}{4/8} \cdot 1 = \frac{3}{4}$

No further transitions are now possible as there are no active nodes.

Example 3
The following network exemplifies the use of an allocation environment:

$$N \equiv (l_1 ::_{\varrho} \mathbf{out}(l)@\ell \parallel l_1 :: \langle l' \rangle \parallel l_2 :: \langle\rangle \parallel l_3 :: \langle\rangle)[u]$$

with $\varrho(\ell)(l_1) = \frac{1}{4}$, $\varrho(\ell)(l_2) = \frac{1}{2}$ and $\varrho(\ell)(l_3) = \frac{1}{4}$.

We have that $Active_N = \{l_1 ::_{\varrho} \mathbf{out}(l)@\ell\}$ and $Active_{Out} = \{l_2 :: \langle\rangle, l_3 :: \langle\rangle\}$. Thus, the possible transitions are:

$$N \xrightarrow[p_1]{} (l_1 ::_{\varrho} \mathbf{nil} \parallel l_2 :: \langle l \rangle \parallel l_3 :: \langle\rangle)[u]$$

and

$$N \xrightarrow[p_2]{} (l_1 ::_{\varrho} \mathbf{nil} \parallel l_2 :: \langle\rangle \parallel l_3 :: \langle l \rangle)[u],$$

where $p_1 = \frac{1/3}{1/3} \cdot \frac{1/2}{3/4} = \frac{2}{3}$, and $p_2 = \frac{1/3}{1/3} \cdot \frac{1/4}{3/4} = \frac{1}{3}$.

Example 4
The following network illustrates communication across the network using an intermediate node as a "buffer":

$$l_1 :: \mathbf{out}(l)@l_2.P \parallel l_2 :: \langle\rangle \parallel l_3 :: \mathbf{in}(x)@l_2.Q.$$

It is easy to see that this network reduces in two steps to

$$l_1 :: P \parallel l_2 :: \langle\rangle \parallel l_3 :: Q[l/x]$$

with probability 1.

Example 5
Consider the following network:

$$l_1 :: \langle l_2\rangle \parallel l_2 :: \langle l_2\rangle \parallel l_3 ::_\varrho \mathbf{in}(x)@\ell.P$$

with $\varrho(\ell)(l_1) = \frac{1}{3}$ and $\varrho(\ell)(l_2) = \frac{2}{3}$.

$Active_{In} = \{l_1 :: \langle l_2\rangle, l_2 :: \langle l_2\rangle\}$, and $Active_N = \{l_3 ::_\varrho \mathbf{in}(x)@\ell\}$. Thus the network reduces to

$$l_1 :: \langle\rangle \parallel l_2 :: \langle l_2\rangle \parallel l_3 ::_\varrho P[l_2/x]$$

with probability $\frac{1}{3}$, and to

$$l_1 :: \langle l_2\rangle \parallel l_2 :: \langle\rangle \parallel l_3 ::_\varrho P[l_2/x]$$

with probability $\frac{2}{3}$.

3 Linear spaces and operators

We will define a non-standard denotational semantics for pcKLAIM which is based on notions from linear algebra and functional analysis. We recall the required notions and we refer to widely available text books and monographs on linear algebra, functional analysis and operator theory for details.

Vector spaces play an important role in modelling systems in many scientific and/or technological fields. In computer science they are much less popular. It is probably futile to speculate about a reason for this, but it might be that the theory of linear spaces and their morphisms (i.e. linear maps) provides a convenient combination of a simple algebraic theory together with a quantitative element whilst computer science structures are often purely qualitative.

3.1 Linear and Hilbert spaces

A *vector space* $\mathcal{V}$ is defined algebraically as an additive group $(\mathcal{V}, +)$ together with a scalar multiplication by elements in the base field $\mathbb{K}$ which distributes over the vector addition. We will concentrate in this paper on real and/or complex vector spaces, i.e. assume $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$. As $\mathbb{R} \subseteq \mathbb{C}$ the distinction between real and complex vector spaces is for many purposes not essential.

For practical computational purposes it is important that one can see vector or linear spaces not just as abstract algebraic structures but also as some concrete tuple spaces, i.e. that any vector can be expressed as a *linear combination* of a certain set of *base* vectors, or just via their coordinates (even though this representation depends on the chosen base and in general might be non-unique for infinite dimensional spaces). Vice versa we can also take any finite or countable set X as a vector space base and generate (algebraically) a vector space $\mathscr{V}(X)$.

Definition 2
The *vector space* $\mathscr{V}(X)$ over a (countable) set X is the set of formal linear combinations, $\sum_{x \in X} c_x x$, of elements in X with coefficients $c_x \in \mathbb{C}$ which we can represent as possibly infinite sequences in $\mathbb{C}$ indexed by elements in X:

$$\mathscr{V}(X) = \{(c_x)_{x \in X} \mid c_x \in \mathbb{C}\}.$$

Besides its algebraic structure we also need to introduce an appropriate topological structure for a vector space. For finite dimensional vector spaces the algebraic structure is essentially forcing a unique topological structure: since a n-dimensional (real or complex) vector space is isomorphic to $\mathbb{R}^n$ or $\mathbb{C}^n$; a possible choice is the product topology. For infinite dimensional spaces the choice of the topological structure is essential, as this is in general not unique.

A popular way to impose a topological structure on a vector space is via a *norm*, i.e. a function $\|.\| : \mathscr{V} \to \mathbb{R}$ which measures the "length of a vector", or a *inner product*, i.e. a function $\langle .,. \rangle : \mathscr{V} \times \mathscr{V} \to \mathbb{C}$ measuring the "angle" between two vectors.

Definition 3
A *Hilbert space* $\mathscr{H}$ is a complex vector space together with an *inner product* $\langle .,. \rangle$: $\mathscr{H} \times \mathscr{H} \to \mathbb{C}$ with (i) $\langle x, y \rangle = \overline{\langle y, x \rangle}$, (ii) $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$, (iii) $\langle x + z, y \rangle = \langle x, y \rangle + \langle z, y \rangle$, and (iv) $\langle x, x \rangle \geqslant 0$ and $\langle x, x \rangle = 0 \iff x = o$ for all $\alpha \in \mathbb{C}$ and $x, y, z \in \mathscr{H}$ (where $\overline{c}$ denotes the complex conjugation in $\mathbb{C}$, and o is the null vector in $\mathscr{H}$) such that the topology induced by the norm $\|x\| = \sqrt{\langle x, x \rangle}$ is complete.

Similarly to the the situation before we can generate a Hilbert space using any countable set X formally as base vectors. For finite sets we can simply take $\mathscr{V}(X) = \mathscr{H}(X)$, but for infinite countable sets we have to impose additional conditions to guarantee that the topology is acceptable, i.e. we have only $\mathscr{H}(X) \subseteq \mathscr{V}(X)$.

Definition 4
Let X be any countable set. The *Hilbert space* $\mathscr{H}(X) \subseteq \mathscr{V}(X)$ over X is the set of formal linear combinations of elements in X with coefficients $c_x \in \mathbb{C}$ which are square summable, i.e.

$$\mathscr{H}(X) = \left\{ (c_x)_{x \in X} \;\middle|\; c_x \in \mathbb{C} \text{ and } \sum_{x \in X} |c_x|^2 < \infty \right\}.$$

It is easy to show that $\mathscr{H}(X)$ is indeed a vector space. This is usually referred to as $\ell^2(X)$. In fact, one can also show that every separable Hilbert space is isomorphic to the "standard" Hilbert space $\ell^2 = \ell^2(\mathbb{N})$ with standard norm on ℓ^2 defined as $\|\vec{x}\|_2 = \|(x_i)_{i \in \mathbb{N}}\|_2 = \sqrt{\sum_{i \in \mathbb{N}} |x_i|^2}$ (see e.g. Corollary 2.2.13 in Kadison & Ringrose, 1997).

3.2 Linear and bounded operators

A map $\mathbf{T} : \mathscr{V} \to \mathscr{W}$ between two vector spaces is *linear* if it respects the algebraic structure, i.e. $\mathbf{T}(x+y) = \mathbf{T}(x) + \mathbf{T}(y)$ and $\mathbf{T}(\alpha x) = \alpha\mathbf{T}(x)$. One usually refers to linear maps $\mathbf{T} : \mathscr{W} \to \mathscr{W}$ as (linear) operators. The set of linear operators on a vector space forms in a natural way itself a vector space. This is in fact a linear algebra, as functional composition defines an algebra product of linear maps.

Linear maps are not just abstract algebraic objects. They also have a computationally useful representation with respect to a basis in the vector space(s) as a *matrix* $\mathbf{M}$. We will denote by $\mathscr{M}$ the set of all matrices, and by $\mathscr{M}(n,m)$ the set of all $n \times m$ matrices with $n, m \in \mathbb{N}$. With this representation we can "perform" the abstract algebraic operations on concrete objects. The application of a linear map to a vector $\mathbf{T}(x)$ and the product of linear maps $\mathbf{T} \cdot \mathbf{S}$ can then be represented in the usual way by *matrix multiplication.*

On a Hilbert space we can define the so called *adjoint* $\mathbf{T}^*$ of an operator $\mathbf{T}$ via the requirement:

$$\langle \mathbf{T}(x), y \rangle = \langle x, \mathbf{T}^*(y) \rangle$$

for all elements $x, y \in \mathscr{H}$. The adjoint operator for bounded linear operators on a Hilbert space always exists and is unique. For finite dimensional operators with a matrix representation $\mathbf{M} \in \mathscr{M}$, the matrix representation of the adjoint is $\mathbf{M}^* = (\overline{\mathbf{M}})^t$, i.e. the transpose complex conjugate matrix. In the case of linear maps on Hilbert spaces we can use the notion of the "length" of vectors (i.e. the vector norm) to describe the "expansiveness" of linear maps to obtain a norm for linear maps $\mathbf{T} \in \mathscr{L}(\mathscr{H}_1, \mathscr{H}_2)$.

Definition 5
The *norm* of a linear map $\mathbf{T} : \mathscr{H}_1 \to \mathscr{H}_2$ is defined as

$$\|\mathbf{T}\| = \sup_{x \in \mathscr{H}_1} \frac{\|\mathbf{T}(x)\|}{\|x\|}.$$

A linear map $\mathbf{T}$ is *bounded* iff $\|\mathbf{T}\| < \infty$. We denote by $\mathscr{B}(\mathscr{H}_1, \mathscr{H}_2) \subseteq \mathscr{L}(\mathscr{H}_1, \mathscr{H}_2)$ the set of bounded linear maps on Hilbert spaces.

For finite dimensional spaces, all linear maps not only respect the algebraic structure of the underlying vector spaces but also preserve their topological structure, i.e. are continuous. In the infinite dimensional case linearity and continuity are different concepts. For Hilbert spaces (and other normed vector spaces) there is however a very convenient way to characterise the set of linear and continuous maps, e.g. Proposition 1.1 in Conway (1990).

Theorem 1
A linear map $\mathbf{T} : \mathscr{H}_1 \to \mathscr{H}_2$ is continuous if and only if it is bounded.

The theory of continuous, i.e. bounded, linear maps and operators on infinite dimensional Hilbert spaces generalises many of the concepts from linear algebra. Our aim is to exploit such an "operator algebraic" approach in order to define a useful quantitative/probabilistic program semantics together with a framework for static program analysis.

3.3 Tensor products

As the tensor product plays a central role in the development of our semantics as well as our analysis, let us recall some of the important facts about the tensor product of vectors, spaces, operators and algebras.

Let $\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_n$ and $\mathscr{W}$ be linear spaces. A map $f : \mathscr{V}_1 \times \mathscr{V}_2 \times \ldots \times \mathscr{V}_n \rightarrow \mathscr{W}$ is called *multilinear* if f is linear in each of its arguments. We denote by $\mathscr{L}(\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_n; \mathscr{W})$ the set of multilinear maps. The algebraic tensor product of vector spaces is then defined via a universal property as follows (see e.g. Definition 1.10.1 in (Palmer, 1994)).

Definition 6
The algebraic tensor product of vector spaces $\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_n$ is given by a vector space $\bigotimes_{i=1}^n \mathscr{V}_i$ and a map $p = \otimes_{i=1}^n \in \mathscr{L}(\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_n; \bigotimes_{i=1}^n \mathscr{V}_i)$ such that if $\mathscr{W}$ is any vector space and $f \in \mathscr{L}(\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_n; \mathscr{W})$ then there exists a unique map $h : \bigotimes_{i=1}^n \mathscr{V}_i \rightarrow \mathscr{W}$ satisfying $f = h \circ p$.

Again, this algebraic construction is sufficient for finite dimensional vector spaces. It is easy to show that in the finite dimensional case we have: $\mathscr{V}(X \times X) \cong \mathscr{V}(X) \otimes \mathscr{V}(X)$. However, 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, Kadison & Ringrose (1997), Fillmore (1996) or Appendix T in Wegge-Olsen (1993)) it is, however, possible to construct from the algebraic tensor product of Hilbert spaces $\mathscr{H}_1, \mathscr{H}_2, \ldots, \mathscr{H}_n$, a Hilbert space which is the tensor product $\bigotimes_{i=1}^n \mathscr{H}_i$.

Furthermore, we can extend the construction of the Hilbert space tensor product to bounded linear maps. For the tensor product of bounded linear maps between Hilbert spaces we have the following result (cf. e.g. Proposition 2.6.12 in Kadison & Ringrose (1997)).

Proposition 2
If $\mathscr{H}_1, \ldots, \mathscr{H}_n$ are Hilbert spaces and $\mathbf{A}_i \in \mathscr{B}(\mathscr{H}_i)$ with $i = 1, \ldots, n$ bounded linear operators, then there exists a unique bounded linear operator $\mathbf{A} \in \mathscr{B}(\mathscr{H}_1 \otimes \ldots \otimes \mathscr{H}_n)$ 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 $x_i \in \mathscr{H}_i$.

This (unique) operator is called the *tensor product* of $\mathbf{A}_i$ and denoted by

$$\mathbf{A} = \mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n.$$

Proposition 3
The tensor product of bounded linear operators $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$ 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 Kadison & Ringrose (1997).

As before we are not only interested in the abstract algebraic and topological properties of the tensor product, but also in a more computationally useful representation. Given the tuple representation of vectors or matrix representation of linear maps one can compute their tensor products simply by their so called *Kronecker product*:

$$(x_1, \ldots, x_n) \otimes (y_1, \ldots, y_m) = (x_1y_1, \ldots, x_1y_m, \ldots, \ldots, x_ny_1, \ldots, x_ny_m)$$

$$\begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mn} \end{pmatrix} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \ldots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \ldots & a_{mn}\mathbf{B} \end{pmatrix}.$$

This means that the tensor product of an n dimensional and an m dimensional vector results in an nm dimensional vector; while the tensor product of an $n \times m$ matrix with an $l \times k$ dimensional one results in an $nl \times mk$ dimensional matrix.

4 Linear operator semantics for pcKLAIM

The probabilistic transition relation $\longrightarrow_p$ introduced in Section 2.2 defines the evolution of a network expressed as a pcKLAIM program and can be encoded via a linear operator on an appropriate vector space representing all the possible configurations of the network. In this section we will define such an encoding in terms of a bounded operator on the Hilbert space over the set of all network configurations.

4.1 Representation of configurations

We denote by $\mathscr{P}$ the set of all pcKLAIM processes in normal form (i.e. according to the syntax in Definition 1) – thus $\mathscr{P}$ contains only the null process and the action prefixed processes – and by $\mathscr{S}$ the set of all local data – that is $\mathscr{S} = \{\langle l \rangle \mid l \in \mathscr{L}\} \cup \{\langle \rangle\}$.

Note that the set $\mathscr{P}$ is countable and that assuming a countable $\mathscr{L}$ the same holds for $\mathscr{S}$. We assume fixed enumerations $\iota : \mathscr{P} \to \mathbb{N}$ and $\kappa : \mathscr{S} \to \mathbb{N}$. We will take the Hilbert spaces $\mathscr{H}(\mathscr{P})$ and $\mathscr{H}(\mathscr{S})$ generated by $\mathscr{P}$ and $\mathscr{S}$ as the central elements in our representation of node and network configurations. If we consider only finite subsets of $\mathscr{P}$ and $\mathscr{S}$ we can identify $\mathscr{V}(\mathscr{P}) \simeq \mathscr{H}(\mathscr{P})$ and $\mathscr{V}(\mathscr{S}) \simeq \mathscr{H}(\mathscr{S})$.

For a process term P we denote by $\vec{P}$ the vector with coordinates (p_i) in $\mathscr{V}(\mathscr{P})$ or $\mathscr{H}(\mathscr{P})$ with

$$p_i = \begin{cases} 1 & \text{for } i = \iota(P) \\ 0 & \text{otherwise.} \end{cases}$$

Similarly we define $\vec{D}$ for data terms.

Our aim is to utilise these vector spaces in order to represent not only particular configurations but also distributions over node and network configurations. This is necessary as the stochastic execution model we use for pcKLAIM means that after each execution step we might be faced not with a single successor state but a whole set of possible configurations together with the probability that a particular configuration will be the result of this computational step.

4.1.1 *Local configurations*

We recall that every node in a pcKLAIM network in normal form may contain at most one process term and/or one data term.

Definition 7
A *local configuration* is a pcKLAIM term of the form:

$$l ::_{\varrho} Q \parallel l :: D$$

with Q and D as in Definition 1.

We denote by $\mathscr{LC}$ the set of local configurations. Obviously, the set of local configurations $\mathscr{LC}$ is isomorphic to $\mathscr{P} \times \mathscr{S}$. A *distribution over local configurations* is a map $\mu : \mathscr{LC} \rightarrow [0,1]$ which is non-zero only for finitely many local configurations and such that $\sum_{c \in \mathscr{LC}} \mu(c) = 1$. Note that as long as we are interested in the configurations after a finite number of steps we only have to accommodate for a finite number of reachable configurations (as we have a finitely branching language). One could generalise the concept of a distribution to that of a measure on $\mathscr{C}$ to deal with infinite sets of local configurations.

We can now define the representation of a local configuration and of distributions over configurations.

Definition 8
For a local configuration $l ::_{\varrho} Q \parallel l :: D$ we define its representation as:

$$[\![l ::_{\varrho} Q \parallel l :: D]\!] = \vec{Q} \otimes \vec{D} \in \mathscr{V}(\mathscr{P}) \otimes \mathscr{V}(\mathscr{S}).$$

For a distribution over local configurations $\mu : \mathscr{LC} \rightarrow [0,1]$ we define its representation by

$$[\![\mu]\!] = \sum_{c \in \mathscr{LC}} \mu(c)[\![c]\!] \in \mathscr{V}(\mathscr{P}) \otimes \mathscr{V}(\mathscr{S}).$$

Note that in the finite case we have $\mathscr{V}(\mathscr{P} \times \mathscr{S}) \simeq \mathscr{V}(\mathscr{P}) \otimes \mathscr{V}(\mathscr{S})$.

Example 6
Consider the network of Example 1 in normal form:

$$(l_{11} :: \mathbf{out}(l_1)@l_2 \parallel l_{12} :: \mathbf{out}(l_2)@l_2 \parallel l_2 :: \langle\rangle)[d]$$

with $d(l_{11}) = \frac{1}{8}$, $d(l_{12}) = \frac{3}{8}$ and $d(l_2) = \frac{1}{2}$.

We enumerate all possible local configurations. The relevant (reachable) local configurations in this example are:

1. $\mathbf{out}(l_1)@l_2$
2. $\mathbf{out}(l_2)@l_2$
3. $\mathbf{nil}$

and the reachable stores:

1. $\langle\rangle$
2. $\langle l_1\rangle$
3. $\langle l_2\rangle$.

Our representation of local configurations will therefore be a vector in the tensor product of $\mathscr{V}(\mathscr{P}) \simeq \mathbb{R}^3$ and $\mathscr{V}(\mathscr{S}) \simeq \mathbb{R}^3$. In particular, we have:

$$\begin{aligned}
[\![l_{11} :: \mathbf{out}(l_1)@l_2]\!] &= (1,0,0) \otimes (1,0,0)\\
[\![l_{12} :: \mathbf{out}(l_2)@l_2]\!] &= (0,1,0) \otimes (1,0,0)\\
[\![l_2 :: \langle\rangle]\!] &= (0,0,1) \otimes (1,0,0).
\end{aligned}$$

4.1.2 Global configurations

Given a network N in normal form, we construct the vector space Dom of all global configurations of N as

$$Dom = \left(\mathscr{V}(\mathscr{P}) \otimes \mathscr{V}(\mathscr{S})\right)^{\otimes m},$$

where m is the number of nodes in N. For finite subsets of $\mathscr{P}$ and $\mathscr{S}$ we have that $Dom \simeq (\mathscr{V}(\mathscr{P} \times \mathscr{S}))^{\otimes m} \simeq \mathscr{V}((\mathscr{P} \times \mathscr{S})^m)$.

Denoting by $\mathscr{N}\mathscr{C}$ the set of all network configurations (with m nodes in normal form) and defining distributions over global configurations in a similar way as for local configurations, we can now represent networks and distributions over network configurations.

Definition 9
For a network configuration in normal form

$$N \equiv \left(\|_{i=1}^m\ l_i ::_{\rho_i} Q_i\ \|_{i=1}^m\ l_i :: D_i\right)[d]$$

we define its representation as:

$$[\![N]\!] = \bigotimes_{i=1}^{m} \left[\!\left[l_i ::_{\rho_i} Q_i \parallel l_i :: D_i\right]\!\right] \in (\mathscr{V}(\mathscr{P}) \otimes \mathscr{V}(\mathscr{S}))^{\otimes m}.$$

For a distribution over global configurations $\mu : \mathscr{N}\mathscr{C} \rightarrow [0,1]$ we define its representation by

$$[\![\mu]\!] = \sum_{N \in \mathscr{N}\mathscr{C}} \mu(N)[\![N]\!] \in (\mathscr{V}(\mathscr{P}) \otimes \mathscr{V}(\mathscr{S}))^{\otimes m}.$$

Note that we do not encode the global scheduling probability given by d in $[\![N]\!]$ because it is a static information which will not change during the execution of the program.

Example 7
For Example 1 in normal form the initial configuration

$$(l_{11} :: \mathbf{out}(l_1)@l_2 \parallel l_{12} :: \mathbf{out}(l_2)@l_2 \parallel l_2 :: \langle\rangle)[d]$$

is represented by the vector

$$((1,0,0) \otimes (1,0,0)) \otimes ((0,1,0) \otimes (1,0,0)) \otimes ((0,0,1) \otimes (1,0,0)).$$

4.2 Basic operators

As a first step towards the definition of a linear operator representing the operational semantics of pcKLAIM, we look at local transitions, i.e. how local configurations change.

Definition 10
Given a pcKLAIM network in normal form $N \equiv \|_{i=1}^{m}(l_i ::_{\rho_i} Q_i \parallel l_i :: D_i)$, we define a *local transition relation* as the restriction of $\xrightarrow[p]{}$ to local configurations, i.e

$$\left(l_i ::_{\rho_i} Q_i \parallel l_i :: D_i\right) \xrightarrow[p]{} \left(l_i ::_{\rho_i} Q_i' \parallel l_i :: D_i'\right)$$

iff $N \xrightarrow[p]{} N'$ with $N' \equiv \|_{i=1}^{m}(l_i ::_{\rho_i} Q_i' \parallel l_i :: D_i')$.

In general we will be only interested in proper local transitions, that is transitions where the local configuration of a node is indeed changing, i.e. where $(l_i ::_{\rho_i} Q_i \parallel l_i :: D_i) \not\equiv (l_i ::_{\rho_i} Q_i' \parallel l_i :: D_i')$. From the semantics in Table 5 we see that each global transition is implemented via at most two proper local transitions.

Proposition 4
Given a pcKLAIM network in normal form $N \equiv \|_{i=1}^{m}(l_i ::_{\rho_i} Q_i \parallel l_i :: D_i)$ such that $N \xrightarrow[p]{} N'$ with $N' \equiv \|_{i=1}^{m}(l_i ::_{\rho_i} Q_i' \parallel l_i :: D_i')$ then there are at most two proper local transitions associated with this transition.

Proof
Follows directly from the inspection of all rules in Table 5. □

4.2.1 Operators on processes

We can distinguish between local transitions which involve just the process component or just the data component of a local configuration. In order to model the transitions which involve a process term at a node by a linear operator we define the following prefix operators.

Local **delay** transitions are represented by the operator $\mathbf{W} : \mathscr{V}(\mathscr{P}) \to \mathscr{V}(\mathscr{P})$, whose matrix representation is given by:

$$\mathbf{W}_{P_1,P_2} = \begin{cases} 1 & P_1 \equiv \mathbf{delay}.P_2 \\ 0 & \text{otherwise.} \end{cases}$$

The local transition of a node performing an **out** action can be modelled by the operator $\mathbf{P} : \mathcal{V}(\mathcal{P}) \to \mathcal{V}(\mathcal{P})$, whose matrix representation is for each $l \in \mathcal{L}$:

$$\mathbf{P}(l)_{P_1,P_2} = \begin{cases} 1 & P_1 \equiv \mathbf{out}(l).P_2 \\ 0 & \text{otherwise.} \end{cases}$$

The local transition of the process performing an **in** action is represented by $\mathbf{G} : \mathcal{V}(\mathcal{P}) \to \mathcal{V}(\mathcal{P})$. For every locality variable x we define it as:

$$\mathbf{G}(x)_{P_1,P_2} = \begin{cases} 1 & P_1 \equiv \mathbf{in}(x).P_2 \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we define a substitution operator $\mathbf{S} : \mathcal{V}(\mathcal{P}) \to \mathcal{V}(\mathcal{P})$ which models the substitution of a variable x by a locality $l \in \mathcal{L}$:

$$\mathbf{S}(l,x)_{P_1,P_2} = \begin{cases} 1 & P_2 \equiv P_1[l/x] \\ 0 & \text{otherwise.} \end{cases}$$

4.2.2 Data operators

The following operators on $\mathcal{V}(\mathcal{S})$ implement the local transitions involving the data component of a local transformation.

We define an adding or joining operator $\mathbf{J} : \mathcal{V}(\mathcal{S}) \to \mathcal{V}(\mathcal{S})$ by:

$$\mathbf{J}(l)_{s_1,s_2} = \begin{cases} 1 & s_2 = \langle l \rangle \ \wedge \ s_1 = \langle \rangle \\ 0 & \text{otherwise.} \end{cases}$$

and a removal or deletion operator $\mathbf{R} : \mathcal{V}(\mathcal{S}) \to \mathcal{V}(\mathcal{S})$ by

$$\mathbf{R}(l)_{s_1,s_2} = \begin{cases} 1 & s_1 = \langle l \rangle \ \wedge \ s_2 = \langle \rangle \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mathbf{R}(l) = \mathbf{J}^*(l)$.

We defined all the basic operators on process as well as data configurations as infinite matrices, i.e. as linear operators on $\mathcal{V}(\mathcal{P})$ and $\mathcal{V}(\mathcal{S})$ respectively. However it is straightforward to show that they are not only linear operators but also continuous ones on $\mathcal{H}(\mathcal{P})$ and $\mathcal{H}(\mathcal{S})$ respectively.

Proposition 5
The linear operators defined above restricted to $\mathcal{H}(\mathcal{P})$ and $\mathcal{H}(\mathcal{S})$ respectively, i.e.

$$\begin{aligned} \mathbf{W} &: \mathcal{H}(\mathcal{P}) \to \mathcal{H}(\mathcal{P}) \\ \mathbf{P}(l) &: \mathcal{H}(\mathcal{P}) \to \mathcal{H}(\mathcal{P}) \\ \mathbf{G}(x) &: \mathcal{H}(\mathcal{P}) \to \mathcal{H}(\mathcal{P}) \\ \mathbf{S}(l,x) &: \mathcal{H}(\mathcal{P}) \to \mathcal{H}(\mathcal{P}) \end{aligned}$$

as well as

$$\begin{aligned} \mathbf{J}(l) &: \mathcal{H}(\mathcal{S}) \to \mathcal{H}(\mathcal{S}) \\ \mathbf{R}(l) &: \mathcal{H}(\mathcal{S}) \to \mathcal{H}(\mathcal{S}) \end{aligned}$$

for all $l \in \mathcal{L}$ and variables $x \in \mathcal{U}$ are bounded operators.

Proof
Obviously, all process operators **W**, **P**, **G** and **S** are “deterministic”, i.e. there is at most one successor process to each process; for example **delay**.P can only make a move to P. The infinite matrix representing these operators thus has only one entry in each row. From this it follows that these operators are bounded.

The same argument holds also for the data operators, i.e. **J** and **R**. □

Example 8
Consider the following (deterministic) pcKLAIM network N in normal form:

$$l_1 :: \mathbf{out}(l)@l_2.\mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2 \parallel l_1 :: \langle\rangle \parallel l_2 :: \mathbf{nil} \parallel l_2 :: \langle\rangle\,.$$

We have for N the following global transitions:

$$\begin{aligned}
&l_1 :: \mathbf{out}(l)@l_2.\mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2 \parallel l_1 :: \langle\rangle \parallel l_2 :: \mathbf{nil} \parallel l_2 :: \langle\rangle\\
&\quad \xrightarrow[1]{} \quad l_1 :: \mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2 \parallel l_1 :: \langle\rangle \parallel l_2 :: \mathbf{nil} \parallel l_2 :: \langle l\rangle\\
&\quad \xrightarrow[1]{} \quad l_1 :: \mathbf{out}(x)@l_2[x/l] \parallel l_1 :: \langle\rangle \parallel l_2 :: \mathbf{nil} \parallel l_2 :: \langle\rangle\\
&\quad \equiv \quad l_1 :: \mathbf{out}(l)@l_2 \parallel l_1 :: \langle\rangle \parallel l_2 :: \mathbf{nil} \parallel l_2 :: \langle\rangle\\
&\quad \xrightarrow[1]{} \quad l_1 :: \mathbf{nil} \parallel l_1 :: \langle\rangle \parallel l_2 :: \mathbf{nil} \parallel l_2 :: \langle l\rangle\,.
\end{aligned}$$

We enumerate the relevant (reachable) process terms as follows:

1. $\mathbf{out}(l)@l_2.\mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2$
2. $\mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2$
3. $\mathbf{out}(x)@l_2$
4. $\mathbf{out}(l)@l_2$
5. **nil**

and stores by:

1. $\langle\rangle$
2. $\langle l\rangle$.

Some of the operators needed to implement the local transitions of N involving data components are:

$$\mathbf{J}(l) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad \mathbf{R}(l) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

and involving the process component:

$$\mathbf{G}(l) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{S}(x,l) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Considering the local transitions

$$l_2 :: \mathbf{nil} \parallel l_2 :: \langle l\rangle \xrightarrow[1]{} l_2 :: \mathbf{nil} \parallel l_2 :: \langle\rangle\,,$$

in the sequence of global transitions above, we can represent the data terms involved by vectors

$$\begin{aligned}[\![l_2 :: \mathbf{nil} \parallel l_2 :: \langle l \rangle]\!] &= (0,0,0,0,1) \otimes (0,1) \\ [\![l_2 :: \mathbf{nil} \parallel l_2 :: \langle \rangle]\!] &= (0,0,0,0,1) \otimes (1,0),\end{aligned}$$

and it is easy to see that

$$[\![l_2 :: \mathbf{nil} \parallel l_2 :: \langle l \rangle]\!] \cdot (\mathbf{I} \otimes \mathbf{R}(l)) = [\![l_2 :: \mathbf{nil} \parallel l_2 :: \langle \rangle]\!].$$

Similarly, we can represent a local transformation involving the process part like

$$l_1 :: \mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2 \parallel l_1 :: \langle \rangle \xrightarrow[1]{} l_1 :: \mathbf{out}(l)@l_2 \parallel l_1 :: \langle \rangle .$$

The vector representation of these two nodes is given by:

$$\begin{aligned}[\![l_1 :: \mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2 \parallel l_1 :: \langle \rangle]\!] &= (0,1,0,0,0) \otimes (1,0) \\ [\![\mathbf{out}(l)@l_2 \parallel l_1 :: \langle \rangle]\!] &= (0,0,0,1,0) \otimes (1,0),\end{aligned}$$

and using the prefix operator $\mathbf{G}(x)$ and substitution operator $\mathbf{S}(x,l)$ we get:

$$[\![l_1 :: \mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2 \parallel l_1 :: \langle \rangle]\!] \cdot (\mathbf{G}(x) \cdot \mathbf{S}(x,l) \otimes \mathbf{I}) = [\![\mathbf{out}(l)@l_2 \parallel l_1 :: \langle \rangle]\!]$$

4.3 Net semantics

To illustrate how in principle we can combine local transitions in order to obtain global ones, let us consider again Example 8.

Example 9
It is easy to see that the global transition in Example 8:

$$\begin{aligned}& l_1 :: \mathbf{in}(x)@l_2.\mathbf{out}(x)@l_2 \parallel l_1 :: \langle \rangle \parallel l_2 :: \mathbf{nil} \parallel l_2 :: \langle l \rangle \\ & \xrightarrow[1]{} l_1 :: \mathbf{out}(l)@l_2 \parallel l_1 :: \langle \rangle \parallel l_2 :: \mathbf{nil} \parallel l_2 :: \langle \rangle\end{aligned}$$

is realised by the operator

$$\begin{aligned}& (\mathbf{G}(x) \cdot \mathbf{S}(x,l) \otimes \mathbf{I}) \otimes (\mathbf{I} \otimes \mathbf{R}(l)) \\ & = ((\mathbf{G}(x) \cdot \mathbf{S}(x,l) \otimes \mathbf{I}) \otimes (\mathbf{I} \otimes \mathbf{I})) \cdot ((\mathbf{I} \otimes \mathbf{I}) \otimes (\mathbf{I} \otimes \mathbf{R}(l)))\end{aligned}$$

as it transforms the representation of the initial global configuration

$$((0,1,0,0,0) \otimes (1,0)) \otimes ((0,0,0,0,1) \otimes (0,1))$$

into the vector corresponding to the successor network:

$$((0,0,0,1,0) \otimes (1,0)) \otimes ((0,0,0,0,1) \otimes (1,0))$$

Although this example is rather simple – it does not involve allocation environments, scheduling probabilities, blocked actions, etc. – it is possible to generalise it in order to construct a linear operator representation of all possible global transitions out of the basic operators introduced above.

Given an operator $\mathbf{M}$ on $\mathcal{V}(\mathcal{P})$ and an operator $\mathbf{N}$ on $\mathcal{V}(\mathcal{S})$, we define the following operator on *Dom* which expresses "localisation" of configurations:

$$\mathbf{At}(i, \mathbf{M} \otimes \mathbf{N}) = (\mathbf{I} \otimes \mathbf{I})^{\otimes(i-1)} \otimes (\mathbf{M} \otimes \mathbf{N}) \otimes (\mathbf{I} \otimes \mathbf{I})^{\otimes(m-i)}.$$

Denoting by $\mathbf{i} = (1, \ldots, 1) \otimes (1, \ldots, 1)$ and by $empty = ((1, \ldots, 1) \otimes (1, 0, \ldots, 0))$ and by $full = ((1, \ldots, 1) \otimes ((0, 1, 0, \ldots, 0) + (0, 0, 1, 0, \ldots, 0) + \ldots))$, we define two (diagonal) test operators:

$$\mathbf{Tst}_{empty}(l_j) = \operatorname{diag}\left(\mathbf{i}^{\otimes(j-1)} \otimes \text{empty} \otimes \mathbf{i}^{\otimes(m-j)}\right)$$
$$\mathbf{Tst}_{full}(l_j) = \operatorname{diag}\left(\mathbf{i}^{\otimes(j-1)} \otimes \text{full} \otimes \mathbf{i}^{\otimes(m-j)}\right)$$

which represent projection operators. Note that $full$ could be defined equivalently as $((1, \ldots, 1) \otimes (0, 1, \ldots, 1))$.

We then define the operator associated with a node n_i which describes the global transitions of the network initiated by a local transition of n_i:

$$\mathbf{T}(n_i) = \mathbf{Nil}(n_i) + \mathbf{Delay}(n_i) + \mathbf{Out}(n_i) + \mathbf{In}(n_i).$$

Each of the operators $\mathbf{Nil}(n_i)$, $\mathbf{Delay}(n_i)$, $\mathbf{Out}(n_i)$ and $\mathbf{In}(n_i)$ models the global changes of the network when the process at n_i is **nil**, **delay**.P, **out**(.).P, or **in**(.).P, respectively. Each of these operators has an effect if and only if the process at n_i is of the correct form. Note that this way all possible cases according to the syntax of Q in Definition 1 are covered.

Concretely, the operators $\mathbf{Nil}(n_i)$, $\mathbf{Delay}(n_i)$, $\mathbf{Out}(n_i)$ and $\mathbf{In}(n_i)$ are defined as follows.

$$\mathbf{Nil}(n_i) = \mathbf{At}(i, \mathbf{E}_{\mathbf{nil},\mathbf{nil}} \otimes \mathbf{I}).$$

In other words, at node n_i there is a local transition $l_i ::_{\rho_i} \mathbf{nil} \parallel l_i :: D \xrightarrow[1]{} l_i ::_{\rho_i} \mathbf{nil} \parallel l_i :: D$ while all other nodes stay unchanged.

$$\mathbf{Delay}(n_i) = \mathbf{At}(i, \mathbf{W} \otimes \mathbf{I}).$$

This implements the transition $l_i ::_{\rho_i} \mathbf{delay}.P \parallel l_i :: D \xrightarrow[1]{} l_i ::_{\rho_i} P \parallel l_i :: D$ at node n_i and leaves all other local configurations unchanged.

While $\mathbf{Nil}(n_i)$ and $\mathbf{Delay}(n_i)$ are unblockable transitions, for **in** and **out** actions we need to consider always the case that the target node is already full or empty respectively. To model this we will need the above introduced testing operators.

$$\mathbf{Out}(n_i) = \sum_{l,\ell,j=1\ldots m} env(n_i)(\ell)(addr(n_j)) \cdot \mathbf{DoOut}(i, j, l).$$

This expresses the idea that an **out** action involves the initiating node n_i – where the prefix **out**(l) for all possible $l \in \mathcal{L}$ is removed – and one of the other nodes n_j with $j = 1 \ldots m$ – where the data l is inserted. The actual pair of nodes n_i and n_j involved in the transition depends on the given allocation environment and in particular on the distribution $env(n_i)(\ell)$. The actual transformations are implemented by:

$$\begin{aligned}&\mathbf{DoOut}(i, j, l)\\ &\quad = \mathbf{Tst}_{empty}(addr(n_j)) \cdot \mathbf{At}(i, \mathbf{P}(l) \otimes \mathbf{I}) \cdot \mathbf{At}(j, \mathbf{I} \otimes \mathbf{J}(l)) \cdot\\ &\qquad + \mathbf{Tst}_{full}(addr(n_j)).\end{aligned}$$

This operator tests if the data l can be placed at node n_j, i.e. if it is empty. If this is the case, then the prefix removal at node n_i and the data insertion at node n_j are performed; otherwise the operator does nothing (note that $\mathbf{Tst}_{full}(addr(n_j)) = \mathbf{Tst}_{full}(addr(n_j)) \cdot \mathbf{At}(i, \mathbf{I} \otimes \mathbf{I}) \cdot \mathbf{At}(j, \mathbf{I} \otimes \mathbf{I})$).

We define the operator representing **in** actions in a similar way.

$$\mathbf{In}(n_i) = \sum_{l,\ell,j=1\ldots m} env(n_i)(\ell)(addr(n_j)) \cdot \mathbf{DoIn}(i,j,l,x),$$

with

$$\begin{aligned}\mathbf{DoIn}(i,j,l,x) &= \\ &= \mathbf{Tst}_{full}(addr(n_j)) \cdot \mathbf{At}(i, \mathbf{G}(x) \cdot \mathbf{S}(data(n_j), x) \otimes \mathbf{I}) \cdot \mathbf{At}(j, \mathbf{I} \otimes \mathbf{R}(l)) \\ &\quad + \mathbf{Tst}_{empty}(addr(n_j)).\end{aligned}$$

Proposition 6

The operators $\mathbf{T}(n_i)$ are bounded linear operators on $(\mathscr{H}(\mathscr{P}) \otimes \mathscr{H}(\mathscr{S}))^{\otimes m}$.

Proof

For all $i \in [1,m]$, the operator $\mathbf{T}(n_i)$ is defined as the finite sum and finite tensor product of the basic operators on processes and data. From Proposition 5 we know that these operators are bounded. It follows that $\mathbf{T}(n_i)$ is bounded too. □

A single step of computation in the network is defined as:

$$\mathbf{T} = \mathscr{N}\left(\sum_{n \in N} d(addr(n)) \cdot \mathbf{T}(n)\right),$$

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

$$(\mathscr{N}(\mathbf{A}))_{ij} = \begin{cases} \frac{\mathbf{A}_{ij}}{a_j} & \text{if } a_j = \sum_i \mathbf{A}_{ij} \neq 0 \\ 1 & \text{if } a_j = 0 \ \wedge \ i = j \\ 0 & \text{if } a_j = 0 \ \wedge \ i \neq j. \end{cases}$$

This expresses the idea that at each computational step the scheduler is choosing one of the nodes n_i which is able to initiate a global network update according to the normalised scheduling probabilities. If a network reaches a final state, i.e. no node can initiate any further update, the normalisation introduces a diagonal entry 1 in $\mathbf{T}$ which simply reproduces the final network state with certainty.

The normalisation procedure $\mathscr{N}$ guarantees that the operator $\mathbf{T}$ is *stochastic*, i.e. that the entries in each row sum up to one. $\mathbf{T}$ therefore defines a so called *discrete time Markov chain* (e.g. see Bause & Kritzinger, 2002). Introducing transition rates in place of transition probabilities allows us to define an analogue *continuous time* version of pcKLAIM (Di Pierro *et al.*, 2004b).

Proposition 7
The operator $\mathbf{T}$ is a bounded linear operator on $(\mathscr{H}(\mathscr{P}) \otimes \mathscr{H}(\mathscr{S}))^{\otimes m}$.

Proof
As all $\mathbf{T}(n)$ are bounded operators it follows immediately that the finite sum $\tilde{\mathbf{T}} = \sum_{n \in N} d(addr(n)) \cdot \mathbf{T}(n)$ is also bounded. The normalisation is well defined as in each row $\sum_i \tilde{\mathbf{T}}_{ij}$ exists as all entries are positive and because $\tilde{\mathbf{T}}$ is bounded — actually one could show from the construction that $\sum_i \tilde{\mathbf{T}}_{ij} \leqslant 1$. □

From the construction of $\mathbf{T}$ it should be clear that $\mathbf{T}$ encodes all the single computational steps as defined by the operational semantics in Table 5, i.e. if there is a (global) transition step $N_1 \xrightarrow[p]{} N_2$ then the entry in $\mathbf{T}$ corresponding to the network configurations N_1 and N_2 is $\mathbf{T}_{N_1,N_2} = p$.

The semantics of the network is then obtained by the iterated application of $\mathbf{T}$ to some initial configuration of the network. In particular, for finite computations we can recover the final configurations and their probabilities via:

$$\lim_{i \to \infty} \mathbf{T}^i [\![N_0]\!].$$

Note that in general this limit need not exist nor represent a distribution. This reflects the fact that some computations might not terminate.

For real networks the semantics we have defined may result in very large matrices whose dimensions grow exponentially with the number of nodes (because of the tensor product operation). Although these matrices are in general very sparse, they are not easy to deal with and calculation might result prohibitive even with the more advanced mathematical tools available for matrix manipulation. Thus, the definition of an abstract semantics which is simpler and more tractable is a necessary step before analysing a network. The abstraction mechanism we will use relies on the Probabilistic Abstract Interpretation framework which we will briefly recall in Section 5.

4.4 Some examples

In order to illustrate the concrete form of the linear operator semantics introduced above we will briefly present two very simple examples.

Example 10
Consider the following pcKLAIM network consisting of just two nodes:

$$l_1 :: \mathbf{out}(l)@l_2 \parallel l_2 :: \mathbf{out}(l)@l_1$$

In this network each of the processes running at either location tries to **out** the token l on the other location.

We use the following enumeration of the relevant process terms:

1. $\mathbf{out}(l)@l_1$
2. $\mathbf{out}(l)@l_2$
3. **nil**

and data terms:

1. $\langle\rangle$
2. $\langle l\rangle$.

The operator specifying the linear operator semantics of network updates initiated by the first node n_1 is then given by:

$$\begin{aligned}
\mathbf{T}(n_1) = &\left(\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\\
&+\left(\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix}\otimes\begin{pmatrix}0&1\\0&0\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\\
&+\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}0&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\\
&+\left(\begin{pmatrix}0&0&0\\0&0&1\\0&0&0\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}0&1\\0&0\end{pmatrix}\right)\\
&+\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}0&0\\0&1\end{pmatrix}\right)
\end{aligned}$$

The first term in this sum describes the network changes when the process at node l_1 is the **nil** process, i.e. the operator **Nil**(n_1): The process transition at l_1 is in this case **nil** $\longrightarrow$ **nil** while the data at location l_1 stays in all cases unchanged, at location l_2 the process as well as the data part remain what they are.

The remaining terms specify the operator **Out**(n_1). The second and third term in the above sum describe the transitions for **out**$(l)@l_1$ running at l_1: If there is "space" at location l_1, i.e. the data term at l_1 is $\langle\rangle$, there is a transition **out**$(l)@l_1 \longrightarrow$ **nil** for the process part at l_1 and a transition $\langle\rangle \longrightarrow \langle l\rangle$ for the data part while there is no changes at locations l_2; if there is "no space" at l_1, i.e. the data part at l_1 is $\langle l\rangle$, then no transitions happen at either location. The third and forth term similarly describe the situation when **out**$(l)@l_2$ is running at location l_1.

The operator $\mathbf{T}(n_2)$ is of a similar form:

$$\begin{aligned}
\mathbf{T}(n_2) = &\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\\
&+\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix}\otimes\begin{pmatrix}0&1\\0&0\end{pmatrix}\right)\\
&+\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}0&0\\0&1\end{pmatrix}\right)
\end{aligned}$$

$$+\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}0&1\\0&0\end{pmatrix}\right)\otimes\left(\begin{pmatrix}0&0&0\\0&0&1\\0&0&0\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)$$
$$+\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}0&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right).$$

The linear operator semantics of the whole pcKLAIM network is then given by:

$$\mathbf{T} = \mathbf{T}(l_1 :: \mathbf{out}(l)@l_2 \parallel l_2 :: \mathbf{out}(l)@l_1) = \mathcal{N}\left(\frac{1}{2}\cdot\mathbf{T}(n_1) + \frac{1}{2}\cdot\mathbf{T}(n_2)\right)$$

based on an implicit uniform distribution for scheduling either node.

Example 11
Consider the following pcKLAIM network consisting of two nodes:

$$l_1 ::_{\varrho_1} \mathbf{out}(l)@\ell \parallel l_2 :: \langle\rangle$$

with the following enumeration of the relevant process terms:

1. $\mathbf{out}(l)@\ell$
2. **nil**

and data terms:

1. $\langle\rangle$
2. $\langle l\rangle$.

and the probabilistic allocation environments:

$$\varrho_1(\ell)(l_1) = \frac{1}{3}$$
$$\varrho_1(\ell)(l_2) = \frac{2}{3}.$$

In other words, in this network the process at location l_1 tries to **out** the token l either to the same location l_1 with probability $1/3$ or with probability $2/3$ at the location l_2.

In this case we get for the linear operator $\mathbf{T}(n_1)$ the following representation:

$$\begin{aligned}\mathbf{T}(n_1) = {} & \left(\begin{pmatrix}0&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\\
&+\frac{1}{3}\cdot\left(\begin{pmatrix}0&1\\0&0\end{pmatrix}\otimes\begin{pmatrix}0&1\\0&0\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\\
&+\frac{1}{3}\cdot\left(\begin{pmatrix}1&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}0&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\\
&+\frac{2}{3}\cdot\left(\begin{pmatrix}0&1\\0&0\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}0&1\\0&0\end{pmatrix}\right)\\
&+\frac{2}{3}\cdot\left(\begin{pmatrix}1&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\otimes\left(\begin{pmatrix}1&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}0&0\\0&1\end{pmatrix}\right).\end{aligned}$$

This operator has essentially the same structure as the operators in the previous example except for the fact that the second and third term are weighted with probability 1/3 indicating the fact that the location variable ℓ will be referring to location l_1 with this probability, and that terms four and five are weighted with the corresponding probability 2/3. In each case we have again one term for the successful and the blocked execution of **out**$(l)@\ell$.

5 Probabilistic abstract interpretation

Classical abstract interpretation provides general techniques for the analysis of programs which are based on the construction of *safe* approximations of the concrete semantics of programs via the notion of a *Galois connection* (Cousot & Cousot, 1992; Nielson *et al.*, 1999).

Probabilistic abstract interpretation re-casts these techniques in a probabilistic setting where linear spaces replace the classical order-theoretic domains, and the notion of the so-called *Moore-Penrose pseudo-inverse* of a linear operator replaces the classical notion of a Galois connection. The abstractions we get this way are *close* approximations of the concrete semantics. Thus, closeness is a quantitative replacement for classical safety which does not require any approximation ordering.

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, i.e. in our setting the Hilbert space $\mathscr{H}(S) = \ell^2(S)$ (or in the finite dimensional case simply $\mathscr{V}(S)$).

Definition 11

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

In this paper we will use the probabilistic analogue of a classical abstract interpretation technique which allows for the definition of an *induced abstract semantics* starting from the abstraction function and the concrete semantics. In the classical framework, this corresponds to the *best correct approximation* for the given concrete semantics, that is the most precise among all correct approximations (the relative precision being left unquantified). In the probabilistic abstract interpretation framework, this technique consists in the following. Given a linear operator Φ on some Hilbert space $\mathscr{V}$ expressing the probabilistic semantics of a concrete system, and a linear abstraction function $\mathbf{A} : \mathscr{V} \mapsto \mathscr{W}$ from the concrete domain into an abstract domain $\mathscr{W}$, we compute the Moore-Penrose pseudo-inverse $\mathbf{G} = \mathbf{A}^\dagger$ of $\mathbf{A}$. Then, the abstract semantics defined as the linear operator on the abstract domain $\mathscr{W}$

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

is the *closest* one to the concrete semantics. This "closeness" property expresses both the "safety" of the approximation and its optimality, the latter being guaranteed by a minimality property of the Moore-Penrose pseudo-inverse relative to the problem of finding solutions to inconsistent linear equations (Deutsch, 2001; Ben-Israel & Greville, 2003). More precisely, given a linear equation $x\mathbf{A} = y$ an (exact) solution x_* is a vector for which $\|x_*\mathbf{A} - y\| = 0$; in the case that no such solution vector x_* exists (the equation is inconsistent), an approximate solution is the so-called "least-squares solution" defined as a vector minimising the Euclidean norm of the residual vector $y - x\mathbf{A}$. Among all the least-squares solutions of $x\mathbf{A} = y$ the one constructed using the Moore-Penrose pseudo-inverse, i.e. $x_0 = y\mathbf{A}^\dagger$ is the one of minimal norm (Ben-Israel & Greville, 2003, Corollary 3).

In our program analysis setting, this result can be rephrased as follows. We can apply a concrete semantics Φ to a concrete vector x and abstract the result giving $x\Phi\mathbf{A}$ or we can apply the abstract operator to an abstract vector giving $x\mathbf{A}\mathbf{A}^\dagger\Phi\mathbf{A}$. Ideally, we would like these to be equal. If $\mathbf{A}$ is invertible then its Moore-Penrose pseudo-inverse is identical to the inverse and we are done. In program analysis $\mathbf{A}$ is never a square matrix and thus $\mathbf{A}\mathbf{A}^\dagger$ in $x\mathbf{A}\mathbf{A}^\dagger\Phi\mathbf{A}$ will lead to some loss of precision. The Moore-Penrose pseudo-inverse is as close as possible to an inverse if the matrix is not invertible and thus for the particular choice of $\mathbf{A}$, $\mathbf{A}^\dagger\Phi\mathbf{A}$ is the best approximation of Φ that we can have. Moreover, by choosing an appropriate notion of distance we can measure this closeness to get a quantitative estimate of the information lost in the abstraction.

We will now introduce in some more details the central notion of Moore-Penrose pseudo-inverse.

5.1 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. Böettcher & Silbermann (1999, Section 4.7); Campbell & Meyer (1979, Definition 1.1.1); Deutsch (2001, Section 8.43)). This is sufficient for the finite-dimensional setting, while for dealing with the infinite-dimensional case we will need some topological considerations which we will use for a more concrete definition.

Definition 12
An element $\mathbf{A} \in \mathcal{B}(\mathcal{H})$ is said to be *Moore-Penrose invertible* if there exists an element $\mathbf{B} \in \mathcal{B}(\mathcal{H})$ such that:

(i) $\mathbf{ABA} = \mathbf{A}$,
(ii) $\mathbf{BAB} = \mathbf{B}$,
(iii) $(\mathbf{AB})^* = \mathbf{AB}$,
(iv) $(\mathbf{BA})^* = \mathbf{BA}$.

If an element $\mathbf{A} \in \mathcal{B}(\mathcal{H})$ is Moore-Penrose invertible then there exists a unique element $\mathbf{B} = \mathbf{A}^\dagger$, the *Moore-Penrose pseudo-inverse* of $\mathbf{A}$, which fulfils the above conditions (Böettcher & Silbermann (1999, Proposition 4.20).

An alternative but equivalent definition is given in Deutsch (2001, Section 8.43) (see also Campbell & Meyer (1979, Definition 1.1.2)).

Definition 13
Let $\mathscr{C}$ and $\mathscr{D}$ be two Hilbert spaces and $\mathbf{A} : \mathscr{C} \mapsto \mathscr{D}$ a bounded linear map between them. A bounded linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathscr{D} \mapsto \mathscr{C}$ 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 $\mathbf{G}$.

For finite-dimensional matrix algebras $\mathscr{M}(n)$ every operator is Moore-Penrose pseudo-invertible (Beutler, 1965; Campbell & Meyer, 1979; Böttcher & Silbermann, 1999; Deutsch, 2001).

For infinite dimensional operators (i.e. operator algebras over infinite dimensional Hilbert spaces) there exist results which guarantee the existence of the Moore-Penrose pseudo-inverse. We mention here the one in Böettcher & Silbermann (1999, Theorem 4.24), which also states how one can "construct" the Moore-Penrose Pseudo-Inverse.

Proposition 8
An operator $\mathbf{A} \in \mathscr{B}(\mathscr{H})$ is Moore-Penrose invertible if and only if it is *normally solvable*, i.e. the range $\{\mathbf{A}x \mid x \in \mathscr{H}\}$ is closed.

In this case $\mathbf{A}^*\mathbf{A} + \mathbf{P}$ – with $\mathbf{P}$ the orthogonal projection of $\mathscr{H}$ onto the kernel of $\mathbf{A}$, i.e. onto $\{x \in \mathscr{H} \mid \mathbf{A}x = o\}$ – is invertible and

$$\mathbf{A}^\dagger = (\mathbf{A}^*\mathbf{A} + \mathbf{P})^{-1}\mathbf{A}^*.$$

It is easy to see that if the range of an operator is finite dimensional then it is normally solvable.

For the finite dimensional case, various algorithms are known for the construction of the Moore-Penrose pseudo-inverse (Campbell & Meyer, 1979) or (Ben-Israel & Greville, 2003). A general technique for computing the Moore-Penrose pseudo-inverse of infinite operators is to approximate them by a sequence of finite dimensional operators.

Given an operator $\mathbf{A} \in \mathscr{B}(\ell^2)$ and a sequence of (orthogonal) projections $\pi_n : \ell^2 \to \ell^2$ onto the first n coordinates of ℓ^2 with $\pi_n^2 = \pi_n = \pi_n^*$, we call $\mathbf{A}_n = \pi_n \mathbf{A} \pi_n$ a *finite section* of $\mathbf{A}$. It corresponds effectively to taking the $n \times n$ sub-matrix in the upper left corner of the matrix representing $\mathbf{A}$. The sequence $(\mathbf{A}_n)_n$ is an *approximating sequence* for $\mathbf{A}$ in the sense that $\mathbf{A}$ is the *strong limit* of this sequence (Böettcher & Silbermann (1999, Section 2.1).

For operators $\mathbf{A}$ with an approximating sequence $(\mathbf{A}_n)_n$ we can construct the Moore-Penrose pseudo-inverse as established by the following proposition (Böettcher & Silbermann (1999, Corollary 4.34).

Proposition 9
Let $\mathscr{H}$ be a separable Hilbert space, $\mathbf{A} \in \mathscr{B}(\mathscr{H})$ and $\mathscr{A}_n$ a sequence of finite dimensional operators $\mathscr{A}_n \in \mathscr{M}(n)$ with $\sup_n \|\mathscr{A}\| < \infty$ and such that $\mathbf{A}_n \to \mathbf{A}$ and $\mathbf{A}_n^* \to \mathbf{A}^*$ strongly. Then $\mathbf{A}$ is normally solvable and $\mathbf{A}_n^\dagger$ converges strongly to $\mathbf{A}^\dagger$ $(\mathbf{A}_n^\dagger \to \mathbf{A}^\dagger)$.

In other words if we can approximate $\mathbf{A}$ by a sequence $(\mathbf{A}_n)_n$ and the sequence $(\mathbf{A}_n^\dagger)_n$ of Moore-Penrose pseudo-inverse converges in the strong operator topology then $\mathbf{A}^\dagger$ exists and is identical to the limit of $(\mathbf{A}_n^\dagger)_n$.

6 Network analysis

Using the PAI framework we can construct induced abstractions or approximations of individual processes. We have followed this approach before in various papers (Di Pierro & Wiklicky, 2000; Di Pierro & Wiklicky, 2001).

In the current context we need a way to analyse *networks* of processes. This task will be simplified thanks to the following properties of the tensor product which allow us to de-compose a network into its individual processes and to analyse their interaction in order to obtain an "integrated" behaviour of the network as a whole.

Proposition 10
Given two bounded linear operators $\mathbf{A}_1$ and $\mathbf{A}_2$ on a Hilbert space, then

$$(\mathbf{A}_1 \otimes \mathbf{A}_2)^\dagger = \mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger.$$

Proof
We show that the Moore-Penrose conditions from Definition 12 hold. We use the properties of the tensor product as stated in Proposition 3.

We denote by $\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2$ and by $\mathbf{B} = \mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger$, in order to show that $\mathbf{A}^\dagger = \mathbf{B}$ we have to show the the following four conditions are fulfilled:

$\mathbf{ABA} = \mathbf{A}$: Exploiting property (i) in Proposition 3 we get:

$$\begin{aligned}
\mathbf{ABA} &= (\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger)(\mathbf{A}_1 \otimes \mathbf{A}_2)\\
&= \mathbf{A}_1\mathbf{A}_1^\dagger\mathbf{A}_1 \otimes \mathbf{A}_2\mathbf{A}_2^\dagger\mathbf{A}_2\\
&= \mathbf{A}_1 \otimes \mathbf{A}_2\\
&= \mathbf{A}.
\end{aligned}$$

$\mathbf{BAB} = \mathbf{B}$: Again by property (i) in Proposition 3 we see that:

$$\begin{aligned}
\mathbf{BAB} &= (\mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger)(\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger)\\
&= \mathbf{A}_1^\dagger\mathbf{A}_1\mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger\mathbf{A}_2\mathbf{A}_1^\dagger\\
&= \mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger\\
&= \mathbf{B}.
\end{aligned}$$

$(\mathbf{AB})^* = \mathbf{AB}$: Exploiting properties (i) and (iv) in Proposition 3 we obtain:

$$\begin{aligned}
(\mathbf{AB})^* &= ((\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger))^*\\
&= (\mathbf{A}_1\mathbf{A}_1^\dagger \otimes \mathbf{A}_2\mathbf{A}_2^\dagger)^*\\
&= (\mathbf{A}_1\mathbf{A}_1^\dagger)^* \otimes (\mathbf{A}_2\mathbf{A}_2^\dagger)^*\\
&= \mathbf{A}_1\mathbf{A}_1^\dagger \otimes \mathbf{A}_2\mathbf{A}_2^\dagger\\
&= (\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger)\\
&= \mathbf{AB}.
\end{aligned}$$

$(\mathbf{BA})^* = \mathbf{BA}$: Using properties (i) and (iv) in Proposition 3 it follows that:

$$\begin{aligned}
(\mathbf{BA})^* &= ((\mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger)(\mathbf{A}_1 \otimes \mathbf{A}_2))^* \\
&= (\mathbf{A}_1^\dagger\mathbf{A}_1 \otimes \mathbf{A}_2^\dagger\mathbf{A}_2)^* \\
&= (\mathbf{A}_1^\dagger\mathbf{A}_1)^* \otimes (\mathbf{A}_2^\dagger\mathbf{A}_2)^* \\
&= \mathbf{A}_1^\dagger\mathbf{A}_1 \otimes \mathbf{A}_2^\dagger\mathbf{A}_2 \\
&= (\mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger)(\mathbf{A}_1 \otimes \mathbf{A}_2) \\
&= \mathbf{BA}.
\end{aligned}$$

□

6.1 Compositional analysis

The above result allows us to construct the Moore Penrose pseudo-inverse of an abstraction which is given by a tensor product in a simple compositional way. Next we need to combine this result with the fact that the semantics of pcKLAIM networks as we have defined it before is also a tensor product in order to obtain an effective way to construct an abstract network semantics based on "node abstractions".

The semantics of pcKLAIM, i.e. the transition relation given in Table 5, is represented by an infinite matrix, or more precisely by a bounded operator $\mathbf{T}$ on a Hilbert space $\mathscr{H}$. For practical computations – when dealing with a single network – we can restrict this operator to one on a Hilbert space generated by just the network configurations reachable from the initial configuration.

Given an initial configuration N_0 of a pcKLAIM network we define the set of *reachable configurations* by:

$$\mathscr{R}(N_0) = \{N \mid N_0 \longrightarrow_p^* N \text{ with } p > 0\}$$

Definition 14

For a pcKLAIM network N_0 we define the *reachable restriction* $\overline{\mathbf{T}}_{N_0}$ of the linear operator semantics $\mathbf{T}$ by

$$\overline{\mathbf{T}}_{N_0} = \pi_{N_0}\mathbf{T}\pi_{N_0}$$

where π_{N_0} is the projection onto $\mathscr{V}(\mathscr{R}(N_0))$.

If we have a finite computation, i.e. if the set of reachable states is finite, we get a finite dimensional matrix, i.e. a linear operator on some finite dimensional vector space.

For any network N_0 the restricted operator $\overline{\mathbf{T}}_{N_0}$ contains enough information to serve as a replacement of the full semantics $\mathbf{T}$ as we have

$$\mathbf{T}^i[\![N_0]\!] = \overline{\mathbf{T}}^i_{N_0}[\![N_0]\!]$$

if we embed $\mathscr{H}(\mathscr{R}(N_0))$ in the obvious way in $\mathscr{H}$.

An important fact for the compositional analysis of pcKLAIM program is that we can express $\mathbf{T}$ and in particular $\overline{\mathbf{T}}$ as a sum of binary communications. This corresponds in the operational semantics to the fact that in each step there are at most two local configurations which change in each of the rules in Table 5.

In the following we will use $\mathbf{I} \otimes \mathbf{I} = \mathbf{I}$ instead of the more correct notation $\mathbf{I}_n \otimes \mathbf{I}_m = \mathbf{I}_{nm}$, where the subscript indicates the dimension of the identity operator, i.e. $\mathbf{I}_n \in \mathscr{M}(n,n)$, $\mathbf{I}_m \in \mathscr{M}(m,m)$ and $\mathbf{I}_{nm} \in \mathscr{M}(nm,nm)$.

Proposition 11

Given an initial network N_0 (with at least two nodes), the operator $\overline{\mathbf{T}} = \overline{\mathbf{T}}_{N_0}$ can be written

$$\overline{\mathbf{T}} = \sum_{i,j=1}^{m} \mathbf{C}(i,j),$$

where $\mathbf{C}(i,j)$ represents the interaction or communication between two nodes, and is of the form

$$\mathbf{C}(i,j) = \mathbf{I}^{\otimes(i-1)} \otimes (\mathbf{M}_i \otimes \mathbf{N}_i) \otimes \mathbf{I}^{\otimes(j-i-1)} \otimes (\mathbf{M}_j \otimes \mathbf{N}_j) \otimes \mathbf{I}^{\otimes(m-j-1)}.$$

Proof

This follows from the definition of the linear operator semantics. $\mathbf{T}$ is defined as sum of $\mathbf{T}(n_i)$ which in turn are defined as the sum of tensor products of operators. It is easy to see that in all four cases the operators are of the form required for $\mathbf{C}(i,j)$: For $\mathbf{Nil}(n_i)$ we have, for example, $\mathbf{N}_i = \mathbf{E}_{\mathbf{nil},\mathbf{nil}}$ and $\mathbf{M}_i = \mathbf{I}$, and we can take any other node j and take $\mathbf{M}_j = \mathbf{I}$ and $\mathbf{N}_j = \mathbf{I}$. Essentially the same argument holds for $\mathbf{Delay}(n_i)$. For $\mathbf{Out}(n_i)$ and $\mathbf{In}(n_i)$ we just have to observe that the operators $\mathbf{DoOut}(i,j,l)$ and $\mathbf{DoIn}(i,j,l,x)$ are the product of an $\mathbf{At}(i,\ldots)$ and an $\mathbf{At}(j,\ldots)$ operator. The properties of the tensor product then imply immediately that $\mathbf{At}(i,\ldots) \cdot \mathbf{At}(j,\ldots)$ is of the form required for $\mathbf{C}(i,j)$ (if $i = j$ we actually return to a case similar to **Nil** and **Delay**).

Note that the probabilistic weighting, the application of test projections as well as the normalisation operation do not change the structure of these operators. Their effect is merely to adjust the numerical entries in the factors of the $\mathbf{C}(i,j)$. □

This means that in order to construct $\mathbf{T}$ or $\overline{\mathbf{T}}$ we have only to consider interactions between two nodes, i.e. operators of the form:

$$(\mathbf{M}_i \otimes \mathbf{N}_i) \otimes (\mathbf{M}_j \otimes \mathbf{N}_j).$$

Suppose we have an abstraction operator $\mathbf{A}$ for single nodes, i.e. a map $\mathbf{A} : \mathscr{V}(\mathscr{P}) \otimes \mathscr{V}(\mathscr{D}) \to \mathscr{A}$, where $\mathscr{A}$ is vector space representing some abstract property of nodes. For example, the abstraction $\mathbf{A}$ could be an operator which marks nodes as infected/sick or not infected/health, with $\mathscr{A} = \mathscr{V}(\{s,h\})$. We can then construct a global abstraction and concretisation for a whole network:

$$\bigotimes_{i=1}^{m} \mathbf{A} = \mathbf{A}^{\otimes m} \text{ and } \left(\bigotimes_{i=1}^{m} \mathbf{A}\right)^{\dagger} = \bigotimes_{i=1}^{m} \mathbf{A}^{\dagger} = (\mathbf{A}^{\dagger})^{\otimes m}.$$

Such a compositional abstraction of a whole network induces the following abstract semantics:

$$\begin{aligned}
\left(\bigotimes_{i=1}^{m}\mathbf{A}^{\dagger}\right)\overline{\mathbf{T}}\left(\bigotimes_{i=1}^{m}\mathbf{A}\right) &= \left(\bigotimes_{i=1}^{m}\mathbf{A}^{\dagger}\right)\sum_{i,j=1}^{m}\mathbf{C}(i,j)\left(\bigotimes_{i=1}^{m}\mathbf{A}\right)\\
&= \sum_{i,j=1}^{m}\left(\bigotimes_{i=1}^{m}\mathbf{A}^{\dagger}\right)\mathbf{C}(i,j)\left(\bigotimes_{i=1}^{m}\mathbf{A}\right)\\
&= \sum_{i,j=1}^{m}(\mathbf{A}^{\dagger})^{\otimes m}(\mathbf{I}^{\otimes(i-1)}\otimes(\mathbf{M}_i\otimes\mathbf{N}_i)\otimes\mathbf{I}^{\otimes(j-i-1)}\otimes(\mathbf{M}_j\otimes\mathbf{N}_j)\\
&\quad \otimes\mathbf{I}^{\otimes(m-j-1)})\mathbf{A}^{\otimes m}\\
&= \sum_{i,j=1}^{m}\mathbf{I}^{\otimes(i-1)}\otimes(\mathbf{A}^{\dagger}(\mathbf{M}_i\otimes\mathbf{N}_i)\mathbf{A})\otimes\mathbf{I}^{\otimes(j-i-1)}\otimes(\mathbf{A}^{\dagger}(\mathbf{M}_j\otimes\mathbf{N}_j)\mathbf{A})\\
&\quad \otimes\mathbf{I}^{\otimes(m-j-1)}\\
&= \sum_{i,j=1}^{m}\mathbf{C}^{\#}(i,j)
\end{aligned}$$

if we denote by $\mathbf{C}^{\#}(i,j) : \mathscr{A}^{\otimes m}\rightarrow\mathscr{A}^{\otimes m}$ the abstraction of the basic communication operators $\mathbf{C}(ij)$ defined by:

$$\begin{aligned}
\mathbf{C}^{\#}(i,j) &= (\mathbf{A}^{\dagger})^{\otimes m}\cdot(\mathbf{C}(i,j))\cdot\mathbf{A}^{\otimes m}\\
&= \mathbf{I}^{\otimes(i-1)}\otimes(\mathbf{A}^{\dagger}(\mathbf{M}_i\otimes\mathbf{N}_i)\mathbf{A})\otimes\mathbf{I}^{\otimes(j-i-1)}\otimes(\mathbf{A}^{\dagger}(\mathbf{M}_j\otimes\mathbf{N}_j)\mathbf{A})\otimes\mathbf{I}^{\otimes(m-j-1)}.
\end{aligned}$$

In other words, the abstraction $\overline{\mathbf{T}}^{\#} : \mathscr{A}^{\otimes m}\rightarrow\mathscr{A}^{\otimes m}$ of the reachable operator semantics $\overline{\mathbf{T}}$ given by $\overline{\mathbf{T}} = \sum_{i,j=1}^{m}\mathbf{C}(i,j)$ induced by a node abstraction $\mathbf{A}$ is given by

$$\overline{\mathbf{T}}^{\#} = \sum_{i,j=1}^{m}\mathbf{C}^{\#}(i,j).$$

This means that to understand what the abstraction is doing we need to study essentially only the operator

$$\begin{aligned}
(\mathbf{A}^{\dagger}(\mathbf{M}_i\otimes\mathbf{N}_i)\mathbf{A})\otimes(\mathbf{A}^{\dagger}(\mathbf{M}_j\otimes\mathbf{N}_j)\mathbf{A}) &= (\mathbf{A}\otimes\mathbf{A})^{\dagger}((\mathbf{M}_i\otimes\mathbf{N}_i)\otimes(\mathbf{M}_j\otimes\mathbf{N}_j))(\mathbf{A}\otimes\mathbf{A})\\
&= (\mathbf{A}^{\dagger}\otimes\mathbf{A}^{\dagger})((\mathbf{M}_i\otimes\mathbf{N}_i)\otimes(\mathbf{M}_j\otimes\mathbf{N}_j))(\mathbf{A}\otimes\mathbf{A}).
\end{aligned}$$

6.2 Interaction matrices

A particular case of a node abstraction is a one dimensional abstraction $\mathbf{A} : \mathscr{V}(\mathscr{P})\otimes\mathscr{V}(\mathscr{D})\rightarrow\mathbb{R}$. This is useful if the aim is to assign to a node a single property. This can be as usual a property the node might have or not (a qualitative property) or as in our quantitative setting the "strength" of this property or the probability that the node exhibits the property in question.

The fact that for one dimensional spaces $\mathscr{A}$ we have

$$\bigotimes_{i=1}^{m}\mathscr{A} = \mathscr{A}^{\otimes m}\simeq\mathscr{A}^{m},$$

and in particular that we can describe the tensor product representing $\mathbf{C}^{\#}(i,j)$ by a $m \times m$ matrix allows us to present the interaction between several nodes by a communication or interaction matrix.

Example 12

To illustrate the construction and use of such a simple abstraction of the network interactions we depart from pcKLAIM and consider a more abstract situation.

Assume that we have three nodes n_1, n_2, and n_3 which can be in just two states $s \in \mathscr{S}$ which we refer to as:

1. healthy
2. sick

We can represent the state of each node by a vector in $\mathscr{V}(\mathscr{S}) \simeq \mathbb{R}^2$ and the overall state of the three node network by a vector in $\mathscr{V}(\mathscr{S}) \otimes \mathscr{V}(\mathscr{S}) \otimes \mathscr{V}(\mathscr{S}) \simeq (\mathbb{R}^2)^{\otimes 3} = \mathbb{R}^8$. For example

$$(1,0) \otimes (0,1) \otimes (1,0)$$

represents a network where the first and third node are healthy while the middle one is sick.

The aim is now to describe a (concrete) dynamics of this network under the assumption that sick nodes make their neighbours sick. Of course one could also think of more complicated dynamics as in the Conway's Game of Life or other cellular automata (Toffoli & Margolus, 1987). Translating the pcKLAIM specific definitions into this more general setting we can define operators $\mathbf{T}(n_i)$'s which describe the overall change of the network if it is triggered by node n_i. For example we get:

$$\begin{aligned}
\mathbf{T}(n_1) &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
&+ \frac{1}{2} \cdot \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
&+ \frac{1}{2} \cdot \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}.
\end{aligned}$$

This expresses the fact that a healthy node leaves the other two unchanged, while a sick node will infect one of its neighbours, each with a 50% chance. Note that we assume – as with pcKLAIM's one-by-one interactions – that each $\mathbf{T}(n_i)$ is the sum of some $\mathbf{C}(i,j)$s describing the interaction between two nodes (one of these is actually "degenerated", involving only one node, as we have it in the **Nil** and **Delay** cases).

The operators $\mathbf{T}(n_2)$ and $\mathbf{T}(n_3)$ are defined similarly. With this we can define a global update operator in which each of the three nodes gets the same chance to trigger the update by:

$$\mathbf{T} = \frac{1}{3} \cdot \mathbf{T}(n_1) + \frac{1}{3} \cdot \mathbf{T}(n_2) + \frac{1}{3} \cdot \mathbf{T}(n_3).$$

The node abstraction operator $\mathbf{A} : \mathbb{R}^2 \to \mathbb{R}$:

$$\mathbf{A} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \text{ with } \mathbf{A}^\dagger = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

models the "sickness" of each node.

We can now construct abstract interaction operators $\mathbf{C}^\#(i,j)$ which describe how a sick node can infect healthy ones. First we need a decomposition

$$\mathbf{T} = \sum_{i,j=1}^{3} \mathbf{C}(i,j).$$

From the definition of $\mathbf{T}(n_1)$ above, we have for example:

$$\begin{aligned}
\mathbf{C}(1,1) &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
\mathbf{C}(1,2) &= \frac{1}{2} \cdot \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
\mathbf{C}(1,3) &= \frac{1}{2} \cdot \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}
\end{aligned}$$

etc. Therefore the abstracted interaction matrices are given by

$$\begin{aligned}
\mathbf{C}^\#(1,1) &= \mathbf{A}^\dagger \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \mathbf{A} \otimes \mathbf{A}^\dagger \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{A} \otimes \mathbf{A}^\dagger \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{A} \\
&= \begin{pmatrix} 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \end{pmatrix} = \begin{pmatrix} 0 \end{pmatrix} \\
\mathbf{C}^\#(1,2) &= \frac{1}{2} \cdot \mathbf{A}^\dagger \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{A} \otimes \mathbf{A}^\dagger \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \mathbf{A} \otimes \mathbf{A}^\dagger \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{A} \\
&= \frac{1}{2} \begin{pmatrix} 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \end{pmatrix} \\
\mathbf{C}^\#(1,3) &= \frac{1}{2} \cdot \mathbf{A}^\dagger \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{A} \otimes \mathbf{A}^\dagger \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{A} \otimes \mathbf{A}^\dagger \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \mathbf{A} \\
&= \frac{1}{2} \begin{pmatrix} 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \end{pmatrix}
\end{aligned}$$

as for 1×1 matrices the tensor (or Kronecker) product degenerates to the normal multiplication and because we have:

$$\mathbf{A}^\dagger \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{A} = \begin{pmatrix} 1 \end{pmatrix} \quad \text{and} \quad \mathbf{A}^\dagger \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \mathbf{A} = \begin{pmatrix} 1 \end{pmatrix}$$

$$\mathbf{A}^\dagger \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \mathbf{A} = \begin{pmatrix} 0 \end{pmatrix} \quad \text{and} \quad \mathbf{A}^\dagger \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{A} = \begin{pmatrix} 1 \end{pmatrix}$$

The abstractions $\mathbf{C}^\#(i,j)$ essentially describe the chances that a sick node i causes an infection at node j – ignoring that i itself stays sick. We can re-arrange the

$\mathbf{C}^{\#}(i,j)$ to form an “infection matrix” $\mathbf{V}$. In our example we get

$$\mathbf{V} = (\mathbf{C}^{\#}(i,j))_{ij} = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$$

The infection matrix $\mathbf{V} = (\mathbf{C}^{\#}(i,j))_{ij}$ itself is not an abstraction of the network dynamics. It is only a convenient way to denote the risks that an infected node i could cause an infection of a previously not infected node j. In other words, $\mathbf{V}$ only describes new infections. Therefore we have, for example, a zero diagonal in $\mathbf{V}$; no infected node can cause a new infection of itself.

Despite the fact that $\mathbf{V}$ does not describe the full abstract network dynamics, we can still utilise it in order to obtain some information about the dynamics of the infection. If we start, for example, with a infected node n_2 and represent this situation by the vector $(0,1,0)$ then one application of this matrix gives a vector $(\frac{1}{2},0,\frac{1}{2})$. This indicates the fact that the chances of each of the two other nodes getting infected is 50% (provided they were both healthy before). Adding both vectors, we obtain the distribution $(\frac{1}{2},1,\frac{1}{2})$, which gives the probabilities that in this case after one step we find an infection in each of the three nodes.

However, as $\mathbf{V}$ only describes the partial ‘new infection’ dynamics, it would be wrong to apply $\mathbf{V}$ directly to this vector in order to obtain the chances of an infection after two steps. We know from the beginning that n_2 is infected and it will stay this way after two, three, etc. steps. The chances of e.g. n_1 to be infected after two steps is given by the chances that it was infected at the beginning, i.e. zero, plus the chances of a first infection after one step, i.e. a half, and the chances of a new infection in the second step. These chances of a new infection in the second step are thus $\frac{1}{2}$, i.e. that it had been still healthy after the first step, times the first component of the distribution we obtain when we apply $\mathbf{V}$ to $(\frac{1}{2},1,\frac{1}{2})$. The same applies for n_3 and we thus obtain $(\frac{7}{8},1,\frac{7}{8})$ describing the health status of the network after two steps.

7 Analysis examples

We present in this section some examples of pcKLAIM networks and show how to analyse their properties using the techniques developed before. The aim is to illustrate the features of our framework, not to analyse real-world networks. However, even in these relatively simple cases we will see that abstraction is essential to overcome the problems associated with the exponential growth of models for distributed systems (due to the tensor product representation). The following examples also illustrate the use of pcKLAIM specifications and the probabilistic aspects captured by the operational and linear operator semantics introduced above.

7.1 Simple one-by-one interaction

We will consider here simple networks similar to the ones described in section 1, where nodes are either *open* – capable of becoming infected – or *closed*. The worm is represented by a recursive process that outputs a token to some locality and then restarts:

$$I \equiv \mathbf{out}(v)@\ell.I.$$

We start by considering very simple networks in which open nodes just input a value and then behave like the worm, while closed nodes silently loop:

$$O \equiv \mathbf{in}(x).I$$
$$C \equiv \mathbf{delay}.C.$$

Given allocation environments $\varrho_1(\ell) = \{\langle l_2, 1\rangle\}$ and $\varrho_2(\ell) = \{\langle l_1, 1\rangle\}$ and the network

$$N_o \equiv l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} O,$$

the possible transitions are:

- With probability 1, i.e. with certainty, the network can make the step $N_o \xrightarrow[1]{} l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} O \parallel l_2 :: \langle v\rangle$, since O is blocked on the **in** action (as node $l_2 ::_{\varrho_2} O$ is not active).
- With probability 1 this state makes a further transition to $l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} I$, i.e. the open process consumes the token v and gets itself infected.
- With probability $\frac{1}{2}$ each, this state makes a transition to $l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} I \parallel l_2 :: \langle v\rangle$ triggered by l_1 and to $l_1 ::_{\varrho_1} I \parallel l_1 :: \langle v\rangle \parallel l_2 ::_{\varrho_2} I$ by executing the infection process at l_2.
- Finally, both these states are transformed into $l_1 ::_{\varrho_1} I \parallel l_1 :: \langle v\rangle \parallel l_2 ::_{\varrho_2} I \parallel l_2 :: \langle v\rangle$.
- After that no further transitions are possible because both processes are blocked.

If instead we consider the network

$$N_c \equiv l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C,$$

with the same allocation environments ϱ_1 and ϱ_2 as before, the possible transitions are:

- With probability $\frac{1}{2}$ this network loops back to itself caused by a network updated triggered by l_2, i.e. a transition initiated by C.
- With probability $\frac{1}{2}$ we have a transition form this initial state N_c to $l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C \parallel l_2 :: \langle v\rangle$.
- The network $l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C \parallel l_2 :: \langle v\rangle$ then loops with probability 1 because the worm is now blocked and further network updates are only caused by the process C.
- On the long run, after infinitely many updates, the network will end up in this second configuration with probability 1 as the probability of reaching this state in one step is $\frac{1}{2}$, in two steps is $\frac{1}{4}$, after three is $\frac{1}{8}$, etc.

So we see that the open nodes eventually block and the closed nodes continue to operate as intended (albeit with a virus token in their local store). A different choice of allocation environment for the open node would allow the virus to propagate.

7.2 Linear semantics of simple interactions

If we recast the above situation using linear operators we have to construct the operators $\overline{\mathbf{T}}_o = \mathbf{T}_{N_o}$ and $\overline{T}_c = \mathbf{T}_{N_c}$ representing the dynamics of the two network configurations. To allow a uniform treatment we will consider a single operator $\overline{T}$ which is the restriction of $\mathbf{T}$ to all reachable configurations for both nets (further restrictions would then result in $\overline{\mathbf{T}}_o$ and $\overline{\mathbf{T}}_c$. Furthermore we present $\overline{\mathbf{T}}$ not in the original form given in Section 4 but in the way introduced in Proposition 11.

We enumerate the reachable process and date store states at each node as follows:

1. $C \equiv \mathbf{delay}.C$
2. $O \equiv \mathbf{in}(x).I$
3. $I \equiv \mathbf{out}(v)@\ell.I$

1. $\langle\rangle$
2. $\langle v\rangle$.

With this we can define an interaction matrices which describes the interaction of a node i with a node j (as in Proposition 11 assuming $i < j$ and similarly for $i > j$) as:

$$\begin{aligned}\mathbf{C}(i,j) = {} & \mathbf{I}^{\otimes(i-1)} \otimes (\mathbf{M}_i^d \otimes \mathbf{N}_i^d) \otimes \mathbf{I}^{\otimes(j-i-1)} \otimes (\mathbf{M}_j^d \otimes \mathbf{N}_j^d) \otimes \mathbf{I}^{\otimes(m-j-1)} \\ & + \mathbf{I}^{\otimes(i-1)} \otimes (\mathbf{M}_i^o \otimes \mathbf{N}_i^o) \otimes \mathbf{I}^{\otimes(j-i-1)} \otimes (\mathbf{M}_j^o \otimes \mathbf{N}_j^o) \otimes \mathbf{I}^{\otimes(m-j-1)} \\ & + \mathbf{I}^{\otimes(i-1)} \otimes (\mathbf{M}_i^i \otimes \mathbf{N}_i^i) \otimes \mathbf{I}^{\otimes(j-i-1)} \otimes (\mathbf{M}_j^i \otimes \mathbf{N}_j^i) \otimes \mathbf{I}^{\otimes(m-j-1)}\end{aligned}$$

where the matrices representing a **delay** transition at the sources node i and the target node are given by

$$\mathbf{M}_i^d = \begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix} \quad \mathbf{N}_i^d = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \quad \mathbf{M}_j^d = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \quad \mathbf{N}_j^d = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$

the matrices encoding **out** actions are:

$$\mathbf{M}_i^o = \begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 1\end{pmatrix} \quad \mathbf{N}_i^o = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \quad \mathbf{M}_j^o = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \quad \mathbf{N}_j^o = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}$$

and the operators for the **in** process are defined to be:

$$\mathbf{M}_i^i = \begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{pmatrix} \quad \mathbf{N}_i^i = \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix} \quad \mathbf{M}_j^i = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \quad \mathbf{N}_j^i = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}.$$

With this we can construct $\overline{\mathbf{T}}$ as a 36×36 matrix (as $36 = (3 \cdot 2) \cdot (3 \cdot 2)$):

$$\overline{\mathbf{T}} = \mathcal{N}\left(\frac{1}{2}\mathbf{C}(1,2) + \frac{1}{2}\mathbf{C}(2,1)\right).$$

This operator $\overline{\mathbf{T}}$ encodes all possible probabilistic transitions between all the 36 networks containing two nodes with processes C, O or I and stores $\langle\rangle$ or $\langle v\rangle$ possible. In particular, we can look at the dynamics of a vector encoding the network N_o. We get as expected the following sequence of distributions over network configurations:

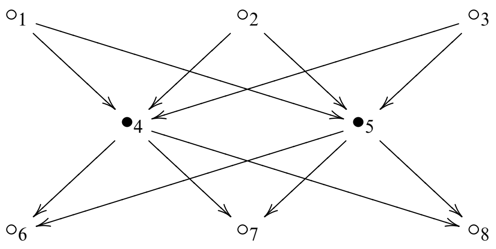
$$
\begin{array}{lcl}
\overline{\mathbf{T}}^0[\![N_o]\!] & = & \{\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} O, 1\rangle\} \\
\overline{\mathbf{T}}^1[\![N_o]\!] & = & \{\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} O \parallel l_2 :: \langle v\rangle, 1\rangle\} \\
\overline{\mathbf{T}}^2[\![N_o]\!] & = & \{\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} I, 1\rangle\} \\
\overline{\mathbf{T}}^3[\![N_o]\!] & = & \left\{\left\langle l_1 ::_{\varrho_1} I \parallel l_1 :: \langle v\rangle \parallel l_2 ::_{\varrho_2} I, \frac{1}{2}\right\rangle, \left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} I \parallel l_2 :: \langle v\rangle, \frac{1}{2}\right\rangle\right\} \\
\overline{\mathbf{T}}^4[\![N_o]\!] & = & \{\langle l_1 ::_{\varrho_1} I \parallel l_1 :: \langle v\rangle \parallel l_2 ::_{\varrho_2} I \parallel l_2 :: \langle v\rangle, 1\rangle\} \\
\overline{\mathbf{T}}^5[\![N_o]\!] & = & \{\langle l_1 ::_{\varrho_1} I \parallel l_1 :: \langle v\rangle \parallel l_2 ::_{\varrho_2} I \parallel l_2 :: \langle v\rangle, 1\rangle\} \\
\dots & & \dots
\end{array}
$$

and for the network N_c we get:

$$
\begin{array}{lcl}
\overline{\mathbf{T}}^0[\![N_c]\!] & = & \{\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C, 1\rangle\} \\
\overline{\mathbf{T}}^1[\![N_c]\!] & = & \left\{\left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C, \frac{1}{2}\right\rangle, \left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C \parallel l_2 :: \langle v\rangle, \frac{1}{2}\right\rangle\right\} \\
\overline{\mathbf{T}}^2[\![N_c]\!] & = & \left\{\left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C, \frac{1}{4}\right\rangle, \left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C \parallel l_2 :: \langle v\rangle, \frac{3}{4}\right\rangle\right\} \\
\overline{\mathbf{T}}^3[\![N_c]\!] & = & \left\{\left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C, \frac{1}{8}\right\rangle, \left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C \parallel l_2 :: \langle v\rangle, \frac{7}{8}\right\rangle\right\} \\
\overline{\mathbf{T}}^4[\![N_c]\!] & = & \left\{\left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C, \frac{1}{16}\right\rangle, \left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C \parallel l_2 :: \langle v\rangle, \frac{15}{16}\right\rangle\right\} \\
\overline{\mathbf{T}}^5[\![N_c]\!] & = & \left\{\left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C, \frac{1}{32}\right\rangle, \left\langle l_1 ::_{\varrho_1} I \parallel l_2 ::_{\varrho_2} C \parallel l_2 :: \langle v\rangle, \frac{31}{32}\right\rangle\right\}. \\
\dots & & \dots
\end{array}
$$

7.3 Network dynamics of simple interactions

In order to put the simple interaction pattern of the previous example into context let us investigate the worm propagation in a network with a similar topology to the networks presented in Section 1. Consider the network



where we assume that the nodes in each layer are fully connected. The pcKLAIM specification of this network is as follows:

$$
\begin{array}{c}
l_1 ::_{\varrho_1} P_1 \parallel l_2 ::_{\varrho_2} P_2 \parallel l_3 ::_{\varrho_3} P_3 \\
\parallel l_4 ::_{\varrho_4} P_4 \parallel l_5 ::_{\varrho_5} P_5 \parallel \\
l_6 ::_{\varrho_6} P_6 \parallel l_7 ::_{\varrho_7} P_7 \parallel l_8 ::_{\varrho_8} P_8
\end{array}
$$

using the allocation environments:

$$\begin{aligned}
\varrho_1(\ell) &= \left\{\left\langle l_2, \frac{1}{4}\right\rangle, \left\langle l_3, \frac{1}{4}\right\rangle, \left\langle l_4, \frac{1}{4}\right\rangle, \left\langle l_5, \frac{1}{4}\right\rangle\right\}\\
\varrho_2(\ell) &= \left\{\left\langle l_1, \frac{1}{4}\right\rangle, \left\langle l_3, \frac{1}{4}\right\rangle, \left\langle l_4, \frac{1}{4}\right\rangle, \left\langle l_5, \frac{1}{4}\right\rangle\right\}\\
\varrho_3(\ell) &= \left\{\left\langle l_1, \frac{1}{4}\right\rangle, \left\langle l_2, \frac{1}{4}\right\rangle, \left\langle l_4, \frac{1}{4}\right\rangle, \left\langle l_5, \frac{1}{4}\right\rangle\right\}\\
\varrho_4(\ell) &= \left\{\left\langle l_5, \frac{1}{4}\right\rangle, \left\langle l_6, \frac{1}{4}\right\rangle, \langle l_7, \frac{1}{4}\rangle, \langle l_8, \frac{1}{4}\rangle\right\}\\
\varrho_5(\ell) &= \left\{\left\langle l_4, \frac{1}{4}\right\rangle, \left\langle l_6, \frac{1}{4}\right\rangle, \langle l_7, \frac{1}{4}\rangle, \langle l_8, \frac{1}{4}\rangle\right\}\\
\varrho_6(\ell) &= \left\{\left\langle l_7, \frac{1}{2}\right\rangle, \left\langle l_8, \frac{1}{2}\right\rangle\right\}\\
\varrho_7(\ell) &= \left\{\left\langle l_6, \frac{1}{2}\right\rangle, \left\langle l_8, \frac{1}{2}\right\rangle\right\}\\
\varrho_8(\ell) &= \left\{\left\langle l_6, \frac{1}{2}\right\rangle, \left\langle l_7, \frac{1}{2}\right\rangle\right\}.
\end{aligned}$$

The processes P_i with $i = 1\ldots, 8$ are C, O or I. In particular, we will be interested in networks where $P_1 = P_2 = P_3 = I$ and $P_6 = P_7 = P_8 = O$, while P_4 and P_5 are either O (indicated by '$\circ$') or C (indicated by '$\bullet$').

Although the number of nodes in each layer are reduced compared to the networks in section 1, the operator representing the concrete operator $\overline{\mathbf{T}}$ for this 8 node network would be a $6^8 \times 6^8$, i.e. a 1679616×1679616, matrix. Although most of its 2821109907456 entries are zero – thus allowing us the usage of sparse matrix techniques – the size of this operator prohibits a direct analysis of its dynamic behaviour.

To overcome this problem we will use a probabilistic abstraction of the dynamics of the individual nodes based on our PAI technique and the fact that it is compositional with respect to the tensor product. This will allow us to obtain a computationally feasible representation of $\overline{\mathbf{T}}$. Reducing the dimension of the individual $\mathbf{M}$ and $\mathbf{N}$ matrices in the definition of interaction operators $\mathbf{C}(i, j)$ from 6×6 to, for example, 3×3 leads to reducing the dimension of $\overline{\mathbf{T}}$ dramatically to a mere $3^8 \times 3^8$ (i.e. 6561×6561) matrix.

One idea for an abstraction would be to ignore the contents of the store and to concentrate only on the process part. However, this kind of abstraction would lead to considering any node with an O process as infected although its infection state actually depends on whether its store contains the virus v or not. We thus had to treat open nodes behind a protective wall of closed nodes as eventually infected although the infection would be unable to reach those open nodes. We will therefore use a different abstraction for the six possible node configurations, namely:

$$\begin{aligned}
l_i ::_\varrho C \parallel l_i :: \langle\rangle &\mapsto \text{healthy}\\
l_i ::_\varrho C \parallel l_i :: \langle v\rangle &\mapsto \text{healthy}
\end{aligned}$$

$$\begin{aligned} l_i ::_\varrho O \parallel l_i :: \langle\rangle &\mapsto \text{vulnerable} \\ l_i ::_\varrho O \parallel l_i :: \langle v\rangle &\mapsto \text{sick} \\ l_i ::_\varrho I \parallel l_i :: \langle\rangle &\mapsto \text{sick} \\ l_i ::_\varrho I \parallel l_i :: \langle v\rangle &\mapsto \text{sick} \end{aligned}$$

Encoding this abstraction into an abstraction matrix and computing its Moore-Penrose pseudo-inverse yields:

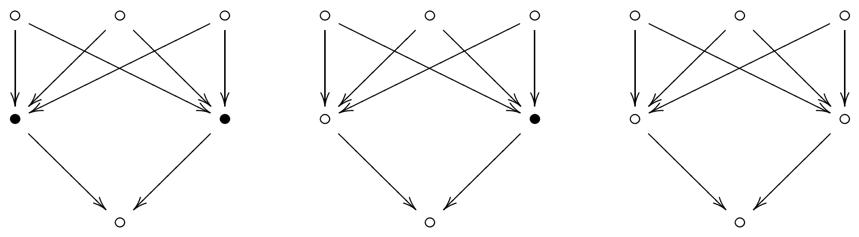
$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{A}^\dagger = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3}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

We can now define abstract interaction matrices $\mathbf{C}^\#(i,j)$ in exactly the same way as in the case of the concrete semantics. We only have to replace the $\mathbf{M} \otimes \mathbf{N}$ and $\mathbf{I}$ factors in the defining tensor product by their abstract versions $\mathbf{A}^\dagger(\mathbf{M} \otimes \mathbf{N})\mathbf{A}$ and $\mathbf{A}^\dagger\mathbf{I}\mathbf{A}$ (which simply gives a 3×3 identity matrix). The operator $\overline{\mathbf{T}}^\#$ is defined as

$$\overline{\mathbf{T}}^\# = \sum_{(i,j)\in N} \mathbf{C}^\#(i,j),$$

where $(i,j) \in N$ indicates that node i and j are connected in the network topology.

We will consider three prototypical situations: in the first one the open nodes in the lower layer are protected by a layer of closed nodes; in the second one a middle layer contains open and closed nodes; and the third one is when the network has only open nodes. In order to further simplify our analysis we will consider the infection risk of only a single node in the bottom layer, i.e. we consider the following three networks:



The operator $\overline{\mathbf{T}}^\#$ representing the common abstract semantics for all three networks is given by a 'small' 729×729 matrix. We represent the configuration of each of the three networks by three vectors in $\mathbb{R}^{729}$.

Assuming that all nodes in the top layer are infected the chances that the bottom node is infected after i iterations in each of the three networks N_c (closed), N_v (vulnerable) and N_o (open) is depicted in the following table:

i	0	1	2	3	4	5	6	100
N_o	0.0	0.0	0.0476	0.1285	0.2271	0.3310	0.4316	0.9999
N_v	0.0	0.0	0.0150	0.0414	0.0758	0.1159	0.1598	0.9998
N_c	0.0	0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

7.4 Probabilistic one-by-one interaction

A more sophisticated definition for O and C is a more symmetrical one which makes (the nodes of) O and C both vulnerable but with different (complementary) probabilities:

$$\begin{aligned} O &\equiv p : \mathbf{in}(x).I \mid (1-p) : S \\ C &\equiv (1-p) : \mathbf{in}(x).I \mid p : S \\ S &\equiv \mathbf{delay}.S. \end{aligned}$$

When p tends towards 1, then O has a high probability of becoming infected but also has the opportunity to behave as a local system (S), whilst the situation is reversed in C. Consider the following two nodes network:

$$N \equiv l_1 ::_\varrho I \parallel l_2 ::_\varrho C$$

with $\varrho(\ell) = \varrho_1(\ell) = \{\langle l_2, 1\rangle\}$. We first transform N in normal form[1] and we get:

$$N \equiv (l_1 ::_\varrho I \parallel l_{21} ::_\varrho \mathbf{in}(x).I \parallel l_{22} ::_\varrho S)[d]$$

with $d = \{\langle l_1, \frac{1}{2}\rangle, \langle l_{21}, \frac{1-p}{2}\rangle, \langle l_{22}, \frac{p}{2}\rangle\}$, or equivalently

$$N \equiv l_1 ::_\varrho^{\frac{1}{2}} I \parallel l_{21} ::_\varrho^{\frac{1-p}{2}} \mathbf{in}(x).I \parallel l_{22} ::_\varrho^{\frac{p}{2}} S$$

with allocation environment $\varrho(\ell)(l_1) = 0$, $\varrho(\ell)(l_{21}) = 1-p$, and $\varrho(\ell)(l_{22}) = p$.

Indicating the three nodes in N by $n_1 \equiv l_1 ::_\varrho I$, $n_2 \equiv l_{21} ::_\varrho \mathbf{in}(x).I$, and $n_3 \equiv l_{22} ::_\varrho S$, we have that $Active_N = \{n_1, n_2\}$ with $Active_{\mathbf{out}}(\varrho, \ell) = \{n_2, n_3\}$. Thus, the following transitions are possible.

- With probability $q_1 = \frac{p}{1+p}$, the network N loops because the transition on node n_3 is chosen.
- Otherwise two transitions can occur depending on the choice of the target node for the **out** at node n_1, namely

$$N \xrightarrow[q_2 = \frac{1-p}{1+p}]{} (n_1 \parallel l_{21} :: \langle v\rangle \parallel l_{21} ::_\varrho \mathbf{in}(x).I \parallel n_3)[d] \equiv N', \text{ and}$$

$$N \xrightarrow[q_3 = \frac{p}{1+p}]{} (n_1 \parallel n_2 \parallel l_{22} :: \langle v\rangle \parallel l_{22} ::_\varrho S)[d] \equiv N''.$$

- Now we have that $Active_{N'} = \{n_1, n_2, n_3\}$ with $Active_{\mathbf{out}}(\varrho, \ell) = \{n_3\}$ and $Active_{\mathbf{in}}(\varrho, \ell) = \{n_2\}$, while $Active_{N''} = \{n_1, n_2, n_3\}$ with $Active_{\mathbf{out}}(\varrho, \ell) = \{n_2\}$ and $Active_{\mathbf{in}}(\varrho, \ell) = \{n_3\}$.

[1] We use the shorthand notation l_{21} and l_{22} for $l_2.l_1$ and $l_2.l_2$ respectively.

- Both states N' and N'' can loop with probabilities $q_4 = \frac{p}{2}$ and $q_5 = \frac{p}{2}$ respectively.
- A further transition from N' with probability $q_7 = \frac{1}{2}$ and from N'' with probability $q_9 = \frac{1}{2}$ will lead to the same network

$$N''' \equiv (n_1 \parallel l_{21} :: \langle v \rangle \parallel l_{21} ::_{\varrho} \mathbf{in}(x).I \parallel l_{22} :: \langle v \rangle \parallel l_{22} ::_{\varrho} S)[d],$$

- Finally, N' with probability $q_6 = \frac{(1-p)}{2}$ and N'' with probability $q_8 = \frac{(1-p)}{2}$ lead to the same network

$$N'''' \equiv (n_1 \parallel l_{21} ::_{\varrho} I \parallel l_{22} ::_{\varrho} S)[d]$$

- The state N''' can loop with probability $q_{11} = p$ or become with probability $(1-p)$ a network where the worm is present in one of the nodes n_2 or n_3. More precisely:

$$N''' \xrightarrow[q_{13}=(1-p)^2]{} (n_1 \parallel l_{21} ::_{\varrho} I \parallel l_{22} :: \langle v \rangle \parallel l_{22} ::_{\varrho} S)[d] \equiv N_{f1}, \text{ and}$$

$$N''' \xrightarrow[q_{15}=(1-p)p]{} (n_1 \parallel l_{21} ::_{\varrho} I \parallel l_{21} :: \langle v \rangle \parallel n_3)[d] \equiv N_{f2},$$

  with $(1-p)^2 + (1-p)p = 1-p$.
- The network N'''' loops with probability $q_{10} = \frac{p}{2}$, leads to N_{f1} with probability $q_{12} = \frac{(2-p)(1-p)}{2}$ and N_{f2} with probability $q_{14} = \frac{(2-p)p}{2}$.
- The states N_{f1} and N_{f2} loop with probability $\frac{p}{2}$ (q_{16} and q_{17}) or, with probability $\frac{(2-p)}{2}$ (q_{18} and q_{19}) – which actually corresponds to the sum of the probabilities of two transitions (see above) — becomes

$$N_f \equiv (l_1 ::_{\varrho} I \parallel l_{21} ::_{\varrho} I \parallel l_{21} :: \langle v \rangle \parallel l_{22} ::_{\varrho} S) \parallel l_{22} :: \langle v \rangle)[d].$$

  In particular, N_f is reached both with a further transition from N_{f1} with probability

$$d(l_1)\varrho(l_{21}) + d(l_{21})\varrho(l_{21}) = \frac{1-p}{2} + \frac{1}{2} = 1 - \frac{p}{2},$$

  and from N_{f2} with probability

$$d(l_1)\varrho(l_{22}) + d(l_{21})\varrho(l_{22}) = \frac{1-p}{2} + \frac{1}{2} = 1 - \frac{p}{2}.$$

- The final state N_f loops with probability 1 since both worms are now blocked.

The composition of the worm with the closed system gives rise to a similar transition system, as the probabilistic versions of O and C only differ in the role of p.

7.5 *Linear semantics of probabilistic interactions*

As with the previous definitions of O and C we only need to understand the one-by-one interaction between I and the probabilistic versions of O and C to analyse more complex networks. We therefore now consider the linear operator semantics of these two node networks. By way of illustration, we continue to consider the interaction of the worm with the closed network. The semantics of a single step is given by:

$$\mathbf{T} = \mathcal{N}\left(\frac{1}{2}\cdot\mathbf{T}(n_1) + \frac{p}{2}\cdot\mathbf{T}(n_2) + \frac{(1-p)}{2}\cdot\mathbf{T}(n_3)\right)$$

where $addr(n_1) = l_1, addr(n_2) = l_{21}, addr(n_3) = l_{22}$. We will just consider the definition of $T(n_1)$.

Using the definition of the semantics gives

$$\mathbf{T}(n_1) = \mathbf{Nil}(n_1) + \mathbf{Delay}(n_1) + \mathbf{Out}(n_1) + \mathbf{In}(n_1)$$

The definitions of $\mathbf{Nil}(n_1)$ and $\mathbf{Delay}(n_1)$ are standard. For the input and output actions, we have that $env(n_1)(\ell)(addr(n_2)) = 1 - p$, $env(n_1)(\ell)(addr(n_3)) = p$ and all other values are 0. Thus

$$\begin{aligned}\mathbf{Out}(n_1) &= (1-p)(\mathbf{DoOut}(1,2,v)) + p(\mathbf{DoOut}(1,3,v))\\ &= (1-p)(\mathbf{Tst}_{empty}(l_{21})\cdot\mathbf{At}(1,\mathbf{P}(v)\otimes\mathbf{I})\cdot\mathbf{At}(2,\mathbf{I}\otimes\mathbf{J}(v))\\ &\quad + \mathbf{Tst}_{full}(l_{21}))\\ &\quad + p(\mathbf{Tst}_{empty}(l_{22})\cdot\mathbf{At}(1,\mathbf{P}(v)\otimes\mathbf{I})\cdot\mathbf{At}(2,\mathbf{I}\otimes\mathbf{J}(v))\\ &\quad + \mathbf{Tst}_{full}(l_{22}))\end{aligned}$$

The definition of $\mathbf{In}(n_1)$ is similar.

The construction for $\mathbf{T}(n_2)$ and $\mathbf{T}(n_3)$ are similar. This leads to a large tensor product which can then be simplified using the techniques from the preceding section. This results in the operator $\overline{\mathbf{T}}$ defined below. We enumerate the globally reachable states as follows:

1. $N \equiv (l_1 ::_\varrho I \parallel l_{21} ::_\varrho \mathbf{in}(x).I \parallel l_{22} ::_\varrho S)[d]$
2. $N' \equiv (l_1 ::_\varrho I \parallel l_{21} :: \langle v\rangle \parallel l_{21} ::_\varrho \mathbf{in}(x).I \parallel l_{22} ::_\varrho S)[d]$
3. $N'' \equiv (l_1 ::_\varrho I \parallel l_{21} ::_\varrho \mathbf{in}(x).I \parallel l_{22} :: \langle v\rangle \parallel l_{22} ::_\varrho S)[d]$
4. $N''' \equiv (l_1 ::_\varrho I \parallel l_{21} :: \langle v\rangle \parallel l_{21} ::_\varrho \mathbf{in}(x).I \parallel l_{22} :: \langle v\rangle \parallel l_{22} ::_\varrho S)[d]$
5. $N'''' \equiv (l_1 ::_\varrho I \parallel l_{21} ::_\varrho I \parallel l_{22} ::_\varrho S)[d]$
6. $N_{f1} \equiv (l_1 ::_\varrho I \parallel l_{21} ::_\varrho I \parallel l_{22} :: \langle v\rangle \parallel l_{22} ::_\varrho S)[d]$
7. $N_{f2} \equiv (l_1 ::_\varrho I \parallel l_{21} ::_\varrho I \parallel l_{21} :: \langle v\rangle \parallel l_{22} ::_\varrho S)[d]$
8. $N_f \equiv (l_1 ::_\varrho I \parallel l_{21} ::_\varrho I \parallel l_{21} :: \langle v\rangle \parallel l_{22} ::_\varrho S) \parallel l_{22} :: \langle v\rangle)[d]$.

Then we have:

$$\overline{\mathbf{T}} = \begin{pmatrix} q_1 & q_2 & q_3 & 0 & 0 & 0 & 0 & 0\\ 0 & q_4 & 0 & q_7 & q_6 & 0 & 0 & 0\\ 0 & 0 & q_5 & q_9 & q_8 & 0 & 0 & 0\\ 0 & 0 & 0 & q_{11} & 0 & q_{13} & q_{15} & 0\\ 0 & 0 & 0 & 0 & q_{10} & q_{12} & q_{14} & 0\\ 0 & 0 & 0 & 0 & 0 & q_{16} & 0 & q_{18}\\ 0 & 0 & 0 & 0 & 0 & 0 & q_{17} & q_{19}\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

We then abstract processes as healthy or sick. A reasonable abstraction would be to consider the first four states as healthy and the second four as infected. This gives rise to the probabilistic abstract interpretation $(\mathbf{A}, \mathbf{G})$, where

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}. \text{ and } \mathbf{G} = \mathbf{A}^{\dagger} = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

such that the abstraction of $\overline{\mathbf{T}}$ is given by:

$$\mathbf{G} \cdot \overline{\mathbf{T}} \cdot \mathbf{A} = \begin{pmatrix} \frac{1+p}{2} & \frac{1-p}{2} \\ 0 & 1 \end{pmatrix}.$$

Thus, if p is close to 1 and the network is healthy, it has a high probability of staying so, as indicated in our earlier discussion.

8 Conclusions

Our aim in this paper has been to develop a framework for the quantitative analysis of distributed systems. One important application of this framework is the modelling of computer worms and viruses. These have become one of the major banes of early twenty first century life. The economic cost of viruses is already incalculable and rising. Our approach provides the basis for using semantics-based techniques to analyse the propagation of worms and viruses.

More concretely, we have presented a probabilistic discrete time version of a restricted KLAIM. The KLAIM language has been designed for describing distributed systems; we have based our work on extensions to a simple, but powerful, core calculus. The extensions that we have introduced include probabilistic local and global parallelism and probabilistic allocation environments. We presented both the structural operational semantics of pcKLAIM and a linear operator semantics. The latter extends our earlier work by introducing the use of tensors to model network parallelism.

We have presented simple abstractions of worms and systems that are more or less susceptible to attacks. The analysis of such systems uses the technique of probabilistic abstract interpretation. Probabilistic abstract interpretation has not previously been used in a setting involving tensor products. We have shown that, by concentrating on the set of reachable states, we can get a tractable analysis of virus propagation. Our analysis is currently handcrafted; an automated analysis would probably require us to work with a superset of reachable states.

Further work should deal with a continuous time, truly asynchronous, version of probabilistic KLAIM. For this we would have to describe not just the probability that something happens but also specify the chance over time when something could happen. This leads to models like Stochastic Petri Nets (Bause & Kritzinger, 2002), and continuous time Markov chains (Tijms, 1994; Norris, 1997), and would resemble approaches used in performance analysis where tensor or Kronecker products are used for a compositional approach (Donatelli, 1993; Buchholz & Kemper, 2004).

References

Albert, R., Jeong, H. and Barabási, A.-L. (2000) Attack and error tolerance in complex networks. *Nature*, **406**, 387–482.

Bause, F. and Kritzinger, P. S. (2002) *Stochastic petri nets – an introduction to the theory*, 2nd ed. Vieweg Verlag.

Ben-Israel, A. and Greville, T. N. E. (2003) *Generalised inverses – theory and applications*, 2nd ed. CMS Books in Mathematics, vol. 15. Springer Verlag.

Bettini, L., Bono, V., De Nicola, R., Ferrari, G., Gorla, D., Loreti, M., Moggi, E., Pugliese, R., Tuosto, E. and Venneri, B. (2003) The KLAIM project: Theory and practice. In: C.Priami (ed), *Global Computing: Programming environments, languages, security and analysis of systems: Lecture Notes in Computer Science 2874*. Springer-Verlag.

Beutler, F. J. (1965) The operator theory of the pseudo-inverse. *J. Math. Anal. Applic.* **10**, 451–493.

Böttcher, A. and Silbermann, B. (1999) *Introduction to large truncated Toeplitz matrices*. Springer-Verlag.

Buchholz, P. and Kemper, P. (2004) *Kronecker based matrix representations for large Markov models*. Technical report, Universität Dortmund.

Campbell, S. L. and Meyer, D. (1979) *Generalized inverse of linear transformations*. Constable and Co.

Conway, J. B. (1990) *A course in functional analysis*. 2nd ed. Graduate Texts in Mathematics, vol. 96. Springer-Verlag.

Cousot, P. and Cousot, R. (1977) Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. *Proceedings of POPL'77*, pp. 238–252.

Cousot, P. and Cousot, R. (1992) Abstract Interpretation and Applications to Logic Programs. *J. Logic Program.* **13**(2-3), 103–180.

De Nicola, R., Ferrari, G. L. and Pugliese, R. (1998) KLAIM: A kernel language for agents interaction and mobility. *IEEE Trans. Softw. Eng.* **24**(5), 315–330.

Deutsch, F. (2001) *Bet Approximation in Inner Product Spaces*. CMS Books in Mathematics, vol. 7. Springer-Verlag.

Di Pierro, A. and Wiklicky, H. (2000) Concurrent constraint programming: towards probabilistic abstract interpretation. *Proceedings of PPDP'00*, pp. 127–138. Montréal, Canada. ACM.

Di Pierro, A. and Wiklicky, H. (2001) Measuring the precision of abstract interpretations. *Proceedings of LOPSTR'00: Lecture Notes in Computer Science 2042*, pp. 147–164. Springer-Verlag.

Di Pierro, A., Hankin, C. and Wiklicky, H. (2002) Approximate non-interference. *Proceedings of CSFW'02*, pp. 3–17. Cape Breton, Canada. IEEE.

Di Pierro, A., Hankin, C. and Wiklicky, H. (2003) Quantitative relations and approximate process equivalences. In: Lugiez, D. (editor), *Proceedings of CONCUR'03 – International Conference on Concurrency Theory: Lecture Notes in Computer Science 2761*, pp. 508–522. Springer-Verlag.

Di Pierro, A., Hankin, C. and Wiklicky, H. (2004a) Approximate non-interference. *J. Comput. Security*, **1**(12), 37–81.

Di Pierro, A., Hankin, C. and Wiklicky, H. (2004b) Continuous-time probabilistic KLAIM. *Secco'04 – Concur workshop on Security issues in Coordination models, languages, and systems.* Electronic Notes in Theoretical Computer Science. Elsevier.

Di Pierro, A., Hankin, C. and Wiklicky, H. (2004c) Probabilistic KLAIM. In: De Nicola, R., Ferrari, G. and Meredith, G. (editors), *Proceedings of Coordination 2004: Lecture Notes in Computer Science 2949*, pp. 119–134. Springer-Verlag.

Donatelli, S. (1993) Superposed stochastic automata: A class of stochastic Petri nets with parallel solution and distributed state space. *Perf. Eval.* **18**, 21–36.

Fillmore, P. A. (1996) *A User's Guide to Operator Algebras.* Wiley.

Herescu, M. and Palamidessi, C. (2000) Probabilistic asynchronous π-calculus. In: Tiuryn, J. (editor), *Proceedings of FOSSACS 2000: Lecture Notes in Computer Science 1784*, pp. 146–160. Springer-Verlag.

Kadison, R. V. and Ringrose, J. R. (1997) *Fundamentals of the theory of operator algebras: Volume I – Elementary Theory.* Graduate Studies in Mathematics, vol. 15. AMS.

Nielson, F., Nielson, H. R. and Hankin, C. (1999) *Principles of Program Analysis.* Springer-Verlag.

Norris, J. R. (1997) *Markov chains.* Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.

Palmer, T. W. (1994) *Banach algebras and the general theory of $*$-algebras – volume i: Algebras and Banach Algebras.* Encyclopedia of Mathematics and Its Applications, vol. 49. Cambridge University Press.

Priami, C. (1995) Stochastic π-calculus. *Comput. J.* **38**(7), 578–589.

Tijms, H. C. (1994) *Stochastic Models – An algorithmic approach.* Wiley.

Toffoli, T. and Margolus, N. (1987) *Cellular Automata Machines: A new environment for modelling.* MIT Press.

Wegge-Olsen, N. E. (1993) *K-theory and C^*-algebras: A friendly approach.* Oxford University Press.