Electronic Communications of the EASST
Volume 22 (2009)

Proceedings of the Third International Workshop on Formal Methods for Interactive Systems (FMIS 2009)

Markov Abstractions for Probabilistic π-Calculus

Hugh Anderson and Gabriel Ciobanu

17 pages

Guest Editors: Michael Harrison, Mieke Massink
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Markov Abstractions for Probabilistic π-Calculus

Hugh Anderson[1] **and Gabriel Ciobanu**[2]

[1] hugh.anderson@acm.org
Wellington Institute of Technology,
New Zealand
[2] gabriel@iit.tuiasi.ro
Romanian Academy, Institute of Computer Science, Iaşi
and A.I.Cuza University of Iaşi, Romania.

Abstract: This paper presents a range of approaches to the analysis and development of program specifications that have been expressed in a probabilistic process algebra. The approach explores Markovian processes as a high-level abstraction tool to reason about system specifications. The abstractions include ones to check the structure of specifications, analyze the long-term stability of the system, and provide guidance to improve the specifications if they are found to be unstable. The approach could present interest to the formal methods and critical-systems development community, as it leads to an automatic analysis of some subtle properties of complex systems. We illustrate some aspects by analyzing the Monty Hall game, and a probabilistic protocol.

Keywords: Abstraction, probabilistic π-calculus, Markov processes.

1 Introduction

The process of effective abstraction underlies most facets of software production and analysis. When reasoning about software systems, we abstract out areas of interest, and reason only about those. When writing software, we use class declarations to encapsulate abstract notions. The benefit is that it could be easier to manipulate abstract notions. We can then make use of apriori knowledge about the abstraction. In the case of automatic analysis or automatic software production, we may also benefit from higher-level abstractions, the automatic tool using pre-proved transformations.

The approach presented here builds on earlier work in [MM01], where Markov Decision Processes (MDPs) are used as an abstraction in the context of the quantitative analysis of program predicate transformers, and in [NPPW07], where automated verification for probabilistic π-calculus is outlined.

We show the development of high-level abstractions based on Markovian processes. In the context presented here, the abstractions are used to assist in the analysis of specifications given in the probabilistic process algebra, useful for the analysis and development of probabilistic algorithms, and also the analysis of critical-systems in which we include estimates of the likelihood of failure. Such systems are commonly found in modern distributed computer systems, and a feature of this approach is that it supports mechanical formal analysis of the systems.

The approach starts with the generation of a Markov transition matrix for the program specification. After generating the transition matrix, we can reason directly about the matrix, or establish the long-term behaviour/equilibrium state of the system, or analyze it directly by using the eigenvectors of the matrix. In each case, we use known properties of the matrix to pick out an area of the specification to fix/modify.

In order to follow this technical sequence, we need to provide effective abstractions of the elements of the approach. Section 2 provides background material on Markovian processes and transition matrices, as well as the probabilistic π-calculus. Section 3 continues with a discussion on suitable abstractions. Section 4 shows various worked examples, including the analysis of a probabilistic protocol. Section 5 concludes the paper.

2 Preliminaries

In this section, the underlying components of the approach, probabilistic process algebras and Markov chains, are briefly introduced and defined.

A process algebra is a technique for mathematically modeling systems constructed of interacting concurrent processes. The technique deserves the term algebra, as it is concerned with axioms and algebraic transformations of expressions in the process algebra.

The most well-known process algebras are the Communicating Sequential Processes (CSP), the Calculus of Communicating Systems (CCS) and the π-calculus. CSP is presented in [BHR84]; CCS and the π-calculus are presented in [Mil99]. Each of these process algebras is useful in its own right, concerned mostly with notions of equivalence between expressions in the process algebras, or with notions of satisfaction between a process algebra expression, and some property expressed in a modal logic.

Probabilistic process algebras have been used for modelling complex systems. Such systems may include elements of interaction where the environment introduces uncertainty; for example, the behaviour of people interacting with the processes, or communication bit error rates, or speed.

The advantage of model description in process algebra is compositionality, i.e., that complex models can be built from smaller ones. With probabilistic process algebra, it is generally understood that the underlying quantitative semantics of the concurrent model of computation is a discrete or continuous-time Markov chain [Hil94]. The analysis of continuous-time Markov chains with large populations can be computationally quite expensive, and so deterministic models can be more efficient.

2.1 Probabilistic π-calculus

Probabilistic π-calculus is an extension of the π-calculus introduced in [NPPW07] with the aim of modeling performances of dynamically reconfigurable systems. It inherits all the syntax of π-calculus, and extends it with the possibility of associating to each action a probability distribution. This means that it is possible to associate to each prefix a quantitative value, represented by the value of a random variable, which follows the above mentioned probability distribution. Distributed systems often have probabilities associated with them that can be represented in the probabilistic π-calculus.

Many works have pointed out the usefulness of probabilistic and stochastic versions of the π-calculus in modeling various systems (see, e.g., [PRSS01, Car08, CP07]). Mainly it could be applied to labeled transition systems representing concurrent systems; this is not possible by using differential equation models. For instance, the use of process algebras in systems biology is natural and provides double advantages: on the one hand, the representation is incremental and compositional; on the other hand, the models support formal verification techniques such as behavioral equivalences and model checking [Cio04, Cio08, NPPW07].

Definition 1 (Probabilistic π-calculus) Let $\mathcal{N} = \{a, b, \ldots, x, y, \ldots\}$ be an infinite set of names. A probabilistic π-calculus process is an expression using the following grammar:

$$P, Q ::= 0 \mid \sum_i \langle \pi_i, p_i \rangle .P_i \mid (\nu x)P \mid [x = y]P \mid P|Q \mid P(y_1, \ldots, y_n)$$

where π is either $x(y)$ or $\bar{x}y$ or τ, and $p \in [0, 1]$.

The intuitive meaning of the operators is essentially the same as in π-calculus. $x(y)$ denotes that we are waiting for a message on the channel x and y acts as a placeholder, which will be replaced with the received message. $\bar{x}y$ represents the output of the message y on the channel x. τ is the silent action. Standard considerations about free and bound names hold. In general, inputs are binding operators on the arguments. This means that in the process $x(y).P$ the name y is bound in P, and not accessible from outside P. Restriction $(\nu n)P$ of the name n makes that name private and unique to P: the name n becomes bound in P. Recursion models infinite behaviour by assuming the existence of a set of equations of the form $A(x) = P$ such that $x \in fn(P)$, where $fn(P)$ stands for the usual free names of P. The definition of $fn(P)$ is standard taking into account that the only binding operators are inputs and restriction.

0 represents the inactive process. The probabilistic process $\langle \pi, p \rangle .P$ is used in a probabilistic choice operation, so the process $\langle \tau, 0.5 \rangle .P + \langle \tau, 0.5 \rangle .Q$ will either continue with process P or Q, with equal probability. If a probability p is 1.0, then we may omit it. In this probabilistic π-calculus, we do not consider nondeterminism.

2.2 Markovian Processes

There are many different kinds of processes, however a particular subset of all processes, the Markov processes, have been studied in great detail for many years.

Markov processes are viewed as a set of random variables $\{X_t\}$, where the time index t runs through an ordered set. The set of all possible values of the variables is known as the *state space* of the process. For one-dimensional state spaces, we classify Markov processes into four distinct categories:

	Time	State space
1	discrete	discrete
2	continuous	discrete
3	discrete	continuous
4	continuous	continuous

A Markov chain is an abstraction $\mathscr{M}$ representing a probabilistic process in terms of a set of states $\mathscr{S}$, and a probability transition matrix $\mathscr{T}$ from one set of states to another. Transitions from state to state occur at discrete time intervals. The future behaviour of a Markov chain is not dependent on the previous path arriving at the current state, although we can specify an initial state according to some pre-defined probability over $\mathscr{S}$.

In elementary probability theory, given an event B, and an exhaustive set of mutually exclusive events $\{C_i\}$, then $\text{prob}(B) = \sum_i \text{prob}(B \mid C_i)\,\text{prob}(C_i)$. In a continuous state space, the conditional probability density function at time n given the state occupied at time m ($l < m < n$) is:

$$p_{X_n}(x \mid X_l = z) = \int_{-\infty}^{\infty} p_{X_m}(y \mid X_l = z)\, p_{X_n}(x \mid X_m = y, X_l = z)\, dy \tag{1}$$

where the X_l are random variables specifying the states of a probabilistic process. A probabilistic process is a Markov Process if, for arbitrary times $l < m < n$,

$$p_{X_n}(x \mid X_m = y, X_l = z, \ldots) = p_{X_n}(x \mid X_m = y) \tag{2}$$

That is, probability density functions are only dependant on the most recent of the time points. Given equation (2), we can rewrite equation (1) as:

$$p_{X_n}(x \mid X_l = z) = \int_{-\infty}^{\infty} p_{X_m}(y \mid X_l = z)\, p_{X_n}(x \mid X_m = y)\, dy \tag{3}$$

This is called the Chapman-Kolmogorov equation, and it indicates that it is possible to build up probability density functions over a long period of time $(l \ldots n)$ from short time periods $(l \ldots m)$ and $(m \ldots n)$. We can express this for discrete states as:

$$p^n_{jk} = \sum_{i=0}^{\infty} p^{n-1}_{ji}\, p_{ik}$$

and note that in process algebras we are dealing with discrete state spaces and hence Markov *chains*, rather than the continuous state space and hence Markov *processes*.

A Markov Decision Process extends the notion of the Discrete Time Markov chain (DTMC) by allowing the process to be controlled through *actions* at each state. MDPs also provide the notion of reward for each taken action and each pair of present and subsequent states. By contrast, in the case of DTMCs, the process cannot be controlled. The subsequent states are only chosen in a probabilistic fashion, with respect to some prescribed transition matrix. Thus DTMCs only allow probabilistic choice (through the transition matrix), while MDPs allow both probabilistic choice (through the probability function) and nondeterministic choice (through the set of possible actions from each state).

If a system eventually settles to a state of statistical equilibrium represented by a state distribution vector $\pi = (\pi_1, \ldots, \pi_n)$, where $\pi_1 + \ldots + \pi_n = 1$, then $\pi = \pi\mathscr{T}$ or alternatively $\pi(\mathscr{I} - \mathscr{T}) = 0$. This gives us a homogeneous set of equations, which will have a solution if $\mathscr{I} - \mathscr{T}$ vanishes. There are various standard methods for solving such a set of equations.

Another notion is that of class solidarity: a state of a given type can only intercommunicate with states of the same type. Furthermore they must have the same recurrence period. A set of states that communicate only with each other are called a closed set - an absorbing set of states. The decomposition theorem states that we may divide any Markov chain into two sets - the recurrent and the transient states. The recurrent set may be decomposed into closed sets. Within each closed set, all states communicate with same period.

The Perron-Frobenius theorem asserts that if A is of order $h \times h$ and if A has $t > 1$ eigenvalues equal in modulus to λ_1 (the largest one), then A can be reduced to a cyclic form by a permutation applied to both rows and columns. The import of this is that it allows us to differentiate between cyclic and stochastic processes.

Hansson and Jonsson have developed PCTL in [HJ94]. It is a probabilistic real-time computation tree logic for checking discrete time Markov chains. PCTL path and state formulas represent properties of states and sequences of states. The PCTL formula are applied to discrete time Markov chains, yielding judgements over them that indicate if the chain *satisfies* the formula. This model-checking approach is not considered here.

3 Markov Abstractions

The principal abstraction is just the (perhaps obvious) one that the transition matrix derived from the specification is an abstraction of the specification. Observations about the matrix apply to the specification. We begin with some general observations about the analysis of transition matrices:

1. Given a Markov transition matrix $\mathscr{T}$, the solution of the equation $p\mathscr{T} = p$ can give p, the steady-state (or equilibrium) state value matrix[1].

2. The least or greatest expected returns from a state p_n, can be easily calculated using the transition matrix $\mathscr{T}$. The calculations are easy to do, but the *meaning* associated with them needs to be given an intuition. This intuition is clarified by means of examples: "Given that we are at p_n what is the least expected cost of a decision taken here?"

3. More complex distributions are possible. For example - the t value may itself be a MDP, making the decision based on (say) a dial on the black box. This suggests a *testing* process - given a specification, and a particular MDP suggesting how another process/tester will interact with it.

We also have the following abstractions over the transition matrix, which can be tied back to the particular state(s) that give rise to the effect. This gives us a method for specification improvement, involving testing the transition matrix for each effect, and then relating this back to the causative states.

Given a Markov transition matrix $\mathscr{T}$,

1. determine which (if any) states are *absorbing* (i.e. capturing states of the system).

2. determine which (if any) states are *ephemeral* (i.e. unreachable).

[1] There are two possible kinds of outcomes - cyclic ones, and steady state ones.

3. determine if the system is *cyclic* or *ergodic*. Is it expected to be cyclic? Is it expected to be a distribution?
4. if it is cyclic, determine the *states* of the cyclic behaviour.
5. determine if the system is a single Markov chain or two (or more) Markov chains.
6. if it describes two (or more) Markov chains, determine the *states* belonging to each of the chains.
7. the distribution of *eigenvalues* of $\mathscr{T}$ gives the following insights:
   (a) If one is 0, then the transitions are not invertible
   (b) If two are close together, then it may take a long time to stabilize
   (c) The maximum value gives the long term behaviour
   (d) The case when the behaviour is cyclic (i.e. no equilibrium)

For each abstraction, we develop an intuition or interpretation, and show how it can be mechanically calculated using conventional linear programming tricks.

3.1 Markov Classification of States

The states of a Markov chain are classified as follows:

Type of state	**Definition**
Ephemeral	Cannot be reached from any other state
Absorbing	Cannot ever leave this state
Periodic	Return to this state cyclically
Aperiodic	Not periodic
Recurrent	Eventual return certain
Transient	Eventual return uncertain
Positive-recurrent	Recurrent, finite mean recurrence time
Null-recurrent	Recurrent, infinite mean recurrence time
Ergodic	Aperiodic, positive recurrent

Each of these types of states has some relevance to the analysis of probabilistic process algebra. For example an ephemeral state (if it is not the first state in the system), may indicate an unreachable part of a specification. An absorbing state may indicate a deadlock condition.

4 Direct Analysis of Systems

In this section, some simple examples are used to demonstrate how systems expressed in a probabilistic process algebra may be expressed as transition matrices, which are then examined directly in terms of the Markov classification of states, in order to discover various properties.

In each example, we also give a motivation or intuition about the applicability of the particular technique used.

4.1 Periodic versus Ergodic

It is useful to discover if an arbitrary process algebra expression is periodic or ergodic. This may not be apparent from the process algebra expression, and if it is found to be (say) periodic when we expect it to be random, then this may indicate a problem with the expression. Let us begin with a very simple example:

$$\begin{aligned} P &\stackrel{def}{=} \langle a, 0.5\rangle .P + \langle b, 0.5\rangle .Q \\ Q &\stackrel{def}{=} \langle a, 0.5\rangle .P + \langle b, 0.5\rangle .Q \end{aligned}$$

The prefixes $\langle x, p\rangle$ indicate a pair consisting of the prefix, and its probability. This one's transition matrix is $\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$ with an equilibrium state vector of $(0.5, 0.5)$. Consider also the expression:

$$\begin{aligned} P &= \langle a, 0\rangle .P + \langle b, 1.0\rangle .Q \\ Q &= \langle a, 1.0\rangle .P + \langle b, 0\rangle .Q \end{aligned}$$

with the transition matrix $\begin{bmatrix} 0 & 1.0 \\ 1.0 & 0 \end{bmatrix}$ and, again, an equilibrium state vector of $(0.5, 0.5)$.

Can we differentiate between these two? They *are* quite different, the first representing a probabilistic process, producing sequences including actions a and b, while the second is a deterministic process defined by alternating actions a and b. If we look at the eigenvalues of the two matrices, we see that the first matrix returns the eigenvalues 1 and 0, whereas the second produces 1 and -1. From the Perron-Frobenius theorem, we can immediately tell from the two eigenvalues that this matrix represents a cyclic process. In summary:

> *The direct analysis of the eigenvalues of a transition matrix allows us to differentiate between two different types of process - specifically cyclic and stochastic processes.*

This analysis may be useful in either case - when we expect a stochastic process and get a cyclic one, and *vice-versa*.

4.2 Ephemeral States

The trivial ephemeral state S_i in a Markov chain is a state in which the i-th column of the transition matrix is *all-zeroes*. This indicates that you can only pass out of this state, and never get back into it.

The intuition in process algebra terms is that we might consider these states to be correct if they are initial states, but otherwise they are *dead-code*. You can never get to an ephemeral state if you do not start in the ephemeral state. In this example, column 4 is all-zeroes, indicating that

S_4 is ephemeral:

$$P = \begin{bmatrix} 0.3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.7 \\ 0.0 & 0.1 & 0.1 & 0.0 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.8 & 0.0 & 0.0 & 0.2 \\ 0.2 & 0.0 & 0.0 & 0.0 & 0.4 & 0.4 \\ 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.5 \\ 0.1 & 0.3 & 0.2 & 0.0 & 0.0 & 0.4 \end{bmatrix}$$

The concept of the ephemeral state may be extended to an ephemeral state-set. This is a set of states which you can only pass out of. In this case, there can be no periodic structure in the ephemeral state-set, and one of the states must therefore be a first ephemeral state. If it is removed from the transition matrix, we are left with a transition matrix with a (possibly null) ephemeral state-set, and the rest of the matrix. In this way, we can step-wise remove the ephemeral state set from the transition matrix, identifying all its elements as we go.

$$P' = \begin{bmatrix} 0.3 & 0.0 & 0.0 & 0.0 & 0.7 \\ 0.0 & 0.1 & 0.1 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.8 & 0.0 & 0.2 \\ 0.0 & 0.5 & 0.0 & 0.0 & 0.5 \\ 0.1 & 0.3 & 0.2 & 0.0 & 0.4 \end{bmatrix} \qquad P'' = \begin{bmatrix} 0.3 & 0.0 & 0.0 & 0.7 \\ 0.0 & 0.1 & 0.1 & 0.8 \\ 0.0 & 0.0 & 0.8 & 0.2 \\ 0.1 & 0.3 & 0.2 & 0.4 \end{bmatrix}$$

In this example, by removing S_4 we discovered a second ephemeral state S_4'. Removing this state removes the ephemeral state-set entirely. In summary:

> *The ephemeral states should correspond only to initial states. If there are any ephemeral states that are not the initial ones, then these indicate that the states are dead states, i.e. ones that will never be visited.*

This analysis is useful in finding parts of a specification that are not required. If so they may as well be removed.

4.3 Absorbing States and State Sets

An absorbing state S_i in a Markov chain is one in which the i-th row is *all-zeroes*. This indicates that you can only pass into this state, and never get back out of it. The intuition in process algebra terms is that we might consider these states to be deadlock ones - our process has arrived at this state and can never exit it. The concept of the absorbing state may be extended to an absorbing state-set, which may either be a chain of states ending in an absorbing state, or a set of states which may be either ergodic or periodic, but in any case do not ever exit.

The first option is easily handled in a manner analogous to that used in the presentation for ephemeral states; the second is a little more difficult. However, the Markov techniques provide again a solution. By permuting the rows and columns, we transform the matrix until it is $P = \begin{bmatrix} \cdots & \cdots \\ 0 & [K] \end{bmatrix}$. In this matrix, the bottom right hand corner K is a square matrix, with all-zero entries to the left of it. If the states contributing to the rows of K do not contain an initial state, we can deduce that this is a set of states that are absorbing. In summary:

> *The absorbing states should correspond only to final states. If there are any absorbing states or state sets that are not final, then these indicate that the states are deadlock states.*

This analysis is useful in finding parts of a specification that cause deadlocks. If so, these are better removed early (at the specification stage) than later.

4.4 Direct Computation

It is relatively easy to manipulate the matrices directly to discover properties of a process. This is illustrated by examining a process algebra expression intended to capture the elements of the Monty-Hall game. The rules of the Monty-Hall game are as follows:

> *A contestant appears in a TV show. The announcer for the show (Monty Hall) hides a prize in an alcove behind one of three curtains. The contestant is asked to select one of the curtains, and then announce the selection. Since the announcer for the show knows where the prize is located, he opens one of the other two curtains, showing the contestant that the prize is not behind it. The contestant is then asked if she wants to change her mind - she can either stick to the original curtain, or change to the other curtain.*

The question is: which of these options should the contestant choose? The surprising answer is that she should change her mind.

4.4.1 Parallel Formulation

We begin with a probabilistic π-calculus expression of the game, played with two interacting processes representing Monty Hall (monty), a contestant (contestant). Beginning with the Monty Hall process, we have Monty selecting a random prize curtain, and signalling the contestant on channel x. The contestant replies with a curtain selection on channel z. Monty then signals the contestant on channel y with a revealed curtain. Finally the contestant replies with a choice on channel w. Monty then signals either the win or lose on channel public:

$$
\begin{array}{lcl}
\text{game} & \stackrel{\text{def}}{=} & \text{monty} \mid \text{contestant} \\
\text{monty} & \stackrel{\text{def}}{=} & \sum_{i\in\{1,2,3\}} \left\langle \bar{x}, \frac{1}{3} \right\rangle .z(c).\text{select}_{ic} \\
\text{select}_{ic} & \stackrel{\text{def}}{=} & \begin{cases} \left\langle \bar{y}(c_l), \frac{1}{2} \right\rangle .w(f).\text{signal}_{if} + \left\langle \bar{y}(c_r), \frac{1}{2} \right\rangle .w(f).\text{signal}_{if} & \text{if } i = c \\ \bar{y}(c_o).w(f).\text{signal}_{if} & \text{if } i \neq c \end{cases} \\
\text{signal}_{if} & \stackrel{\text{def}}{=} & \begin{cases} \overline{\text{public}}(\text{win}).!\text{monty} & \text{if } i = f \quad (\text{a win!}) \\ \overline{\text{public}}(\text{lose}).!\text{monty} & \text{if } i \neq f \quad (\text{a lose}) \end{cases}
\end{array}
$$

The process is replicated, and continues forever, or until we switch off the TV. The other processes may be defined as follows, where the contestant makes the deliberate decision to choose the *other* curtain (that is the contestant changes curtains):

$$
\text{contestant} \stackrel{\text{def}}{=} x.\left(\sum_{j\in\{1,2,3\}} \left\langle \bar{z}(c_j), \frac{1}{3} \right\rangle .y(m).\bar{w}(c_o).!\text{contestant} \right)
$$

To model the the case when the contestant decides not to change curtains, our contestant process would be

$$\text{contestant} \stackrel{\text{def}}{=} x.\left(\sum_{j\in\{1,2,3\}} \left\langle \bar{z}(c_j), \frac{1}{3}\right\rangle .y(m).\bar{w}(c_j).!\text{contestant}\right)$$

We can model the whole system using a simple (single) process algebra expression, only consisting of choice, and no parallel composition. The following section shows such an expression.

4.4.2 Expansion of Parallel Composition - 1

We can expand the parallel composition of the expression using the modified expansion law as shown below. We assume that we have actions (a_1, a_2, a_3) representing the selection of Monty Hall selecting one of the three alcoves. The contestant then can indicate with one of three actions (b_1, b_2, b_3) which alcove/curtain she wishes to choose by pressing the corresponding button. Monty can then open one of the two remaining curtains, the left one or the right one if there is a choice (c_l, c_r), or the only remaining one if there is not (c_o). Finally, the contestant can choose either to change to the other curtain by selecting action (d) or not. In the following expression, the contestant tries the *changing* algorithm - that is, she changes her selection. There are also two extra states, indicating *winning* and *losing* which we are interested in:

$$\begin{aligned}
\text{hide} &\stackrel{\text{def}}{=} \sum_{i\in\{1,2,3\}} \left\langle a_i, \frac{1}{3}\right\rangle .\text{select}_i \\
\text{select}_i &\stackrel{\text{def}}{=} \sum_{j\in\{1,2,3\}} \left\langle b_j, \frac{1}{3}\right\rangle .\text{monty}_{ij} \\
\text{monty}_{ij} &\stackrel{\text{def}}{=} \begin{cases} \langle c_l, 0.5\rangle .\text{change}_l + \langle c_r, 0.5\rangle .\text{change}_r & \text{if } i = j \\ c_0.\text{change}_o & \text{if } i \neq j \wedge k \neq i \wedge k \neq j \end{cases} \\
\text{change}_i &\stackrel{\text{def}}{=} \begin{cases} d.\text{lose} & \text{if } i = l \\ d.\text{lose} & \text{if } i = r \\ d.\text{win} & \text{if } i = o \end{cases} \\
\text{lose} &\stackrel{\text{def}}{=} \text{hide} \\
\text{win} &\stackrel{\text{def}}{=} \text{hide}
\end{aligned}$$

We can now use this to generate a simple Markov transition matrix, as seen in section 4.4.4.

4.4.3 Expansion of Parallel Composition - 2

Another technique may be used to expand out the parallel composition of probabilistic processes. In [HK01], Hillston et al. show how the Kronecker representation of a parallel composition of Markovian processes is an efficient representation, and allows us to create the generator matrix as required. The Kronecker representation of a parallel composition of N component processes is represented by a computation over their transition matrices R_i, and including factors related to their *interaction* ($P_{i,\alpha}$, which represents a probability transition matrix for each component i

associated with each interaction α), and *normalization* ($\overline{P}_{i,\alpha}$, which normalizes the interaction matrices for each component i and each interaction α, where r_α is the minimum of the rates of action α):

$$Q \stackrel{\text{def}}{=} \bigoplus_{i=1}^{N} R_i + \sum_{\alpha \in Z} r_\alpha \left(\bigotimes_{i=1}^{N} P_{i,\alpha} - \bigotimes_{i=1}^{N} \overline{P}_{i,\alpha} \right)$$

In our example, if the matrix for monty was $\mathscr{M}$ and the matrix for the contestant was $\mathscr{C}$, then the Kronecker representation is $Q = \mathscr{M} \oplus \mathscr{C} + f \times [\mathscr{M}_i \otimes \mathscr{C}_i - \mathscr{M}_n \otimes \mathscr{C}_n]$. Note that this representation is compositional, and this is of use in that the sort of sparse matrices generated by our process algebra expressions are *compactly* represented by the Kronecker expression. In addition, the unexpanded expression may *itself* give insights into the behaviour of the interacting processes - for example, the interaction matrix may tell you about the *degree of binding* between processes.

4.4.4 Markov Transition System

In our example, the first process expansion technique applied to the process expression results in 18 states - i.e. the size of the transition matrix is 18×18. The matrix is:

$$M = \begin{array}{r} \\ \text{hide} \\ \text{select}_1 \\ \text{select}_2 \\ \text{select}_3 \\ \text{monty}_{11} \\ \text{monty}_{12} \\ \text{monty}_{13} \\ \text{monty}_{21} \\ \text{monty}_{22} \\ \text{monty}_{23} \\ \text{monty}_{31} \\ \text{monty}_{32} \\ \text{monty}_{33} \\ \text{change}_l \\ \text{change}_r \\ \text{change}_o \\ \text{lose} \\ \text{win} \end{array} \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Noting that the process cycles after four states, we calculate game $=$ init $\times M^4$ where init is the initial state vector. From this we get a vector representing the probability of being in each state after four cycles. The vector is game $= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\frac{1}{3},\frac{2}{3})$. The values associated with lose and win are $\frac{1}{3}$ and $\frac{2}{3}$ respectively, indicating that with this strategy, we will

(in the long term) win! By contrast, if we change the definition of the change$_{ij}$ state to represent the choice of not changing curtains:

$$\text{change}_{ij} \stackrel{\text{def}}{=} \begin{cases} \text{win} & \text{if } i = l \\ \text{win} & \text{if } i = r \\ \text{lose} & \text{if } i = o \end{cases}$$

By examination of this new matrix, we establish that the values associated with lose and win are $\frac{2}{3}$ and $\frac{1}{3}$ respectively, indicating that this is not as good a strategy as the previous one.

4.4.5 Comment on Direct Evaluation

The transition laws for $S\pi$ may be directly applied to the original expression of the game, but they do not result in an efficient reduction of the expression. Initially there are three possible transitions, and each must be separately evaluated - i.e. the expression expands rather than reduces. The point here is that the matrix representation is simpler to handle, and reduces mechanically and quickly. In this section various techniques for evaluating the properties of probabilistic processes were given. In summary:

> *The reduction of a parallel composition of probabilistic processes to a single Markov transition matrix or chain allows a compact representation of the behaviour of the process. This matrix/chain may then be evaluated using either algebraic or arithmetic techniques to discover properties of the original processes.*

This analysis is useful in directly evaluating the probabilistic behaviour of a process without having to apply the more complex $S\pi$ rules a step at a time. The evaluation is done in a simple manner on the transition matrix alone.

4.5 First Passage Probabilities

In a previous example, we were interested in the time taken to get to the absorbing state set. In Markov terms, this is known as the first passage probability. The first passage probability from state i to state j at time t is defined by the conditional probability that state j is entered at time t, and state j is not entered *before* time t, this being conditional on starting at state i. The mean first passage probability M_{ij} from state i to state j in transition matrix $\mathscr{T}$ is derived from the mean first passage matrix, which is given by

$$M = (I - Z + EZ_{\text{diag}})D$$

where I is the identity matrix, Z is the fundamental matrix, E is a matrix containing ones everywhere, Z_{diag} is the matrix containing in its diagonal the components of the fundamental matrix (and zeros everywhere else) and finally, D contains in its diagonal $1/\alpha_i$. (1 divided by the components of the limit matrix). Z, the fundamental matrix, is computed by $Z = (I - (P - \mathscr{T}))^{-1}$.

This gives us a technique for finding the mean passage time - the expected number of transitions/steps performed to get from state i to state j. We begin by constructing the transition matrix for our probabilistic expression. We next ensure that it is irreducible and stochastic, and we may

then calculate the mean first passage probability matrix M. This matrix may be used to evaluate mean first passage times for any transition of interest. Note that once again, this computation is performed by manipulations of the transition matrix $\mathscr{T}$, which is used to derive the mean first passage matrix. This matrix may in turn be used to derive specific useful properties as seen in the next two sections.

4.5.1 Commute Time

A useful property that may be of interest is the *commute* time of a system. The commute time between state i and state j is the expected time to return to state i after visiting state j at least once. This is derived from the mean first passage matrix:

$$C_{ij} = M_{ij} + M_{ji}$$

An intuition about the usefulness of *commute* time may be gleaned from the following:

> *Consider the evaluation of two competing specifications for a probabilistic algorithm. A particular cycle of states in the algorithm is of interest, as it is time critical. The* commute*-time analysis will yield the faster specification.*

This quantitative evaluation of two competing designs may assist in the correct selection of a better design for implementation.

4.5.2 Cover Time

The *cover* time is the expected time to visit all components of a system. Again this has an interpretation in the software design, specification and analysis field. The calculation of the cover time is not as simple as the commute time, and involves analysis of the spanning tree for the transition graph. However tight bounds may be efficiently calculated as seen in [Fei95]. An intuition about the usefulness of *cover* time may be gleaned from the following:

> *Consider the evaluation of two competing specifications for a probabilistic algorithm. Each of the states in the algorithm is of interest, and must be visited at least once, and this is time critical. The* cover*-time analysis may yield the better one of the two competing specifications.*

Again, quantitative evaluation of two competing designs helps in the correct selection of a better design for implementation.

4.6 Equilibrium

Another core concept in the analysis of ergodic Markov processes is that of equilibrium. The calculation of the equilibrium of a Markov chain gives insight into the long-term behaviour of the process. As an example of this, we consider an expression of a probabilistic algorithm for a non-repudiation protocol. The purpose of this protocol is to ensure that two processes agree that a transaction has taken place. At some time, one of the processes cannot violate the protocol and later claim that it did not. The protocol probabilistically ensures that neither process

can repudiate an agreement. We consider two expressions of the protocol, one in which both processes act honourably, and a second in which the second process attempts to cheat by refusing to send an acknowledgement. By looking at the long-term behaviour of the two expressions, using an infinite Markov chain, we see that in the first case the likelihood of agreement is unity. However in the second case the likelihood of agreement approaches zero.

This protocol is simplified for the purposes of the paper. The sender process randomly chooses a message n out of M possible message numbers, and elects to transmit the (encrypted) transaction during this message. A friendly receiver immediately responds with a acknowledgement containing the message, and then proceeds to decrypt the message (this process taking much longer than the round-trip time of the messages). When the message decrypts correctly, the protocol has ended, and the receiver cannot later attempt to deny the transaction, as it sent the acknowledgement before it attempted to decrypt. An unfriendly receiver flips a coin to decide if it wishes to NOT send an acknowledgement. If the process is lucky enough to do this when it receives the correctly encrypted transaction, then it can deny the transaction ever took place. We model the system with a sender and a receiver. The x channel is used for the messages from the sender to the receiver, the y channel contains the acknowledgement. Beginning with the sender process, and then defining a good and a bad receiver process, we have:

$$
\begin{array}{lcl}
\text{protocol} & \stackrel{\text{def}}{=} & \text{sender} \mid \text{receiver} \\
\text{sender} & \stackrel{\text{def}}{=} & \left\langle \bar{x}(t,d(t)), \frac{1}{M} \right\rangle .y.0 + \left\langle \bar{x}(q,d(r)), 1-\frac{1}{M} \right\rangle .y.!\text{sender} \\
\text{goodie} & \stackrel{\text{def}}{=} & \begin{cases} \langle x(t,q), 1.0 \rangle .\bar{y}.\overline{\text{public}}(T).0 & \text{if } d(q) = t \\ \langle x(t,q), 1.0 \rangle .\bar{y}.!\text{goodie} & \text{if } d(q) \neq t \end{cases} \\
\text{baddie} & \stackrel{\text{def}}{=} & \begin{cases} \left\langle x(t,q), \frac{1}{2} \right\rangle .\overline{\text{public}}(T).0 + \left\langle x(t,q), \frac{1}{2} \right\rangle .\bar{y}.!\text{baddie} & \text{if } d(q) = t \\ \left\langle x(t,q), \frac{1}{2} \right\rangle .\overline{\text{public}}(F).0 + \left\langle x(t,q), \frac{1}{2} \right\rangle .\bar{y}.!\text{baddie} & \text{if } d(q) \neq t \end{cases}
\end{array}
$$

Choosing an arbitrary (finite) value for M, it is easy to calculate that the probability for the *goodie* to succeed will be 1.0, and for the *baddie* reduces to zero:

$$P(T) = \frac{1}{M}\sum_{i=1}^{M} 1 \text{ if } d(q) = t \quad (=1) \quad \text{and} \quad P(T) = \frac{1}{M}\sum_{i=1}^{M} \left(\frac{1}{2}\right)^i \text{ if } d(q) = t \quad (\to 0)$$

In summary:

> *The equilibrium for a stochastic process is easily derived from the transition matrix, and allows us to make assertions about the long term behaviour of the process.*

The analysis is completely mechanical, and relies on an early use of abstraction, reducing a relatively complicated process algebra expression to manipulations on a Markov chain.

4.7 Evaluation of System Behaviour: a *hard* problem

The final approach to the analysis of communicating systems suggested by the Markov abstraction is the prediction and evaluation of system behaviour even when we cannot complete the quantification of the transition matrix. The view here is that we may not be able to quantify the probabilities of certain events, or, because of the interaction with other events, be unable to externally monitor them. In this situation, we can still generate a Markov transition matrix, although the matrix will contain variables (unknowns) in certain places.

Consider a tiny example of a transition matrix $\mathscr{T}$: $\mathscr{T} = \begin{bmatrix} p & 1-p \\ q & 1-q \end{bmatrix}$. To discover the eigenvalues of this, we need to solve $\det\left(\begin{bmatrix} p-\lambda & 1-p \\ q & 1-q-\lambda \end{bmatrix}\right) = 0$ for λ. This is simple for the given problem: $\lambda^2 + (q-p-1)\lambda + (p-q) = 0 \Rightarrow \lambda = 1, p-q$. Note that the single large value 1 indicates that this is a stochastic matrix if $p,q \notin \{0,1\}$. The solutions for λ are then used to generate the eigenvectors for the matrix $\mathscr{T}$ by solving $v\mathscr{T} = \lambda v$ for v. However, in a more general case, this is normally considered a *hard* problem, particularly if there are a large number of variables and the matrix is large. Instead, this sort of problem is solved using arithmetic and algorithmic techniques rather than algebraic ones. Again there are many well-known arithmetic techniques for quickly finding solutions to large sets of equations. Let us now consider how this sort of evaluation can assist in a software engineering process:

> *Consider the evaluation of a specification for a probabilistic algorithm in which some of the probabilities are unknown. By constructing a transition matrix, and then solving the eigenvector equations, we derive a compact set of information that can be used to make assertions about the specification. (For example - in the trivial example above, if $p = 1$ and $q = 0$ we can immediately tell that the process is cyclic with length 2. If $p = q$ we can immediately tell that the process has an ephemeral state. For any other values for p and q, the process is stochastic).*

In this example, we see how the Markov abstraction can lead to a better understanding of a process even in the absence of quantitative values.

5 Conclusion

In this paper we have concentrated on the mechanical calculation of properties of probabilistic process algebra expressions using a matrix P which defines a Markov decision process for the expression. The key point of the approach is that instead of reasoning about the detailed structure of the process algebra expression, we reason at a higher level of abstraction using Markovian abstractions. These abstractions include ones to predict long term behaviour of a system, identify deadlock states, identify ephemeral states, and calculate system properties by direct manipulation of P. Note that this approach is different from previous approaches, which do not concentrate on the element of abstraction suggested by the use of Markovian processes.

This abstraction allows us to compactly represent aspects of the behaviour of processes. For example, a single vector (the eigenvalues of the matrix) can be used to infer useful properties of a process. In addition, efficient libraries and procedures for calculating probabilities or rates have been developed over the long history of Markov processes, and we can obtain results using these efficient arithmetic techniques. The results can not only give direct quantitative assessments of the behaviour of a design or software element, but also can lead to comparisons between competing designs. If two designs/implementations had similar properties in other areas, but one was better in it's *cover* time, then we might choose to pick this one.

In summary, we have outlined ways in which Markovian processes may be used as a high-level abstraction tool to reason about program specifications expressed in a probabilistic π-calculus.

The abstractions include ones to check the structure of specifications, analyze the long-term stability of the system, and provide guidance to improve the specifications.

References

[BHR84] S. Brookes, C. Hoare, A. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM* 31(3):560–599, 1984.
doi:10.1145/828.833

[Car08] L. Cardelli. On Process Rate Semantics. *Theor. Comput. Sci.* 391(3):190–215, 2008.
doi:10.1016/j.tcs.2007.11.012

[Cio04] G. Ciobanu. Software Verification of Biomolecular Systems. In *Modelling in Molecular Biology*. Pp. 40–59. Springer-Verlag, 2004.

[Cio08] G. Ciobanu. From Gene Regulation to Stochastic Fusion. In *UC '08: Proceedings of the 7th international conference on Unconventional Computing*. Pp. 51–63. Springer-Verlag, Berlin, Heidelberg, 2008.
doi:10.1007/978-3-540-85194-3_7

[CP07] K. Chatzikokolakis, C. Palamidessi. A Framework for Analyzing Probabilistic Protocols and its Application to the Partial Secrets Exchange. *Theor. Comput. Sci.* 389(3):512–527, 2007.
doi:10.1016/j.tcs.2007.09.006

[Fei95] U. Feige. A Tight Lower Bound on the Cover Time for Random Walks on Graphs. *RSA: Random Structures and Algorithms* 6:51–54,433–438, 1995.
doi:10.1002/rsa.3240060406

[Hil94] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.
doi:10.2277/0521673534

[HJ94] H. Hansson, B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing* 6(5):512–535, 1994.
citeseer.nj.nec.com/hansson94logic.html

[HK01] J. Hillston, L. Kloul. An Efficient Kronecker Representation for PEPA models. In Alfaro and Gilmore (eds.), *Proceedings of the first joint PAPM-PROBMIV Workshop*. Lecture Notes in Computer Science 2165, pp. 120–135. Springer-Verlag, 2001.
doi:10.1.1.62.8157

[Mil99] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.
doi:10.2277/0521658691

[MM01] C. Morgan, A. McIver. Cost Analysis of Games, Using Program Logic. In *8th Asia-Pacific Software Engineering Conference (APSEC 2001)*. P. 351 (Abstract only). 2001.

[NPPW07] G. Norman, C. Palamidessi, D. Parker, P. Wu. Model Checking the Probabilistic π-Calculus. In *Proc. 4th International Conference on Quantitative Evaluation of Systems (QEST'07)*. Pp. 169–178. IEEE Computer Society, 2007. doi:10.1109/QEST.2007.27

[PRSS01] C. Priami, A. Regev, E. Y. Shapiro, W. Silverman. Application of a Stochastic Name-Passing Calculus to Representation and Simulation of Molecular Processes. *Information Processing Letters* 80(1):25–31, 2001. doi:10.1016/S0020-0190(01)00214-9