ABSTRACT

SONI, ARVIND. Probabilistic and Nondeterministic Systems . (Under the direction of Professor Dr. S. Purushothaman Iyer).

Probabilistic and nondeterministic systems are important to model systems such as distributed network protocols, concurrent systems and randomized algorithms, where nondeterminism is inherently present along with probabilistic choices. Probabilistic transition systems without any nondeterminism have been explored over the past decade. Several logics have been proposed to express the probabilistic behavior of systems. Nondeterministic systems differ from their probabilistic counterparts in that there behavior needs a notion of scheduler which resolves the nondeterministic choices. The probability space of observations of such systems is dependent on the choice of scheduler. In the absence of unique probability space the system properties can only be measured in intervals. The methods proposed in literature for quantitative analysis of nondeterministic systems use approximations for conjunction and disjunction to avoid the nonlinearity in the equations for measures.

The contribution of this thesis is three fold. In the initial part we present a model checking method for quantitative analysis of nondeterministic systems. We generate a set of constraints and compute the minimum and maximum measure for the property without using any approximations. Secondly, we present an abstraction and probabilistic bisimulation based approach to model and analyze randomized token stabilization protocol. In the end we present a method for compositional verification of PNS where in we describe weak predicate transformers which can be used to generate sub-specification for the unknown systems from the specification of the composite system.

Probabilistic and Nondeterministic Systems

by

Arvind Soni

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2004

Approved By:

Dr. George N. Rouskas	Dr. Ting Yu
Dr. S. Purushothaman Iyer Chair of Advisory Committee	

To my friends and family

Biography

Arvind Soni was born on 2nd August, 1980 in Durg, India. He did his preliminary schooling at Vishwadeep Hr. Sec. School and attended High School at the Senior Sec. School Sector X. He completed his bachelors degree in Computer Science from the Indian Institute of Technology, Bombay, in 2002 and joined North Carolina State University in Fall 2002. His research interests include graph theory, algorithms, model checking and formal verification. His non-research interests include racquet ball, tennis, movies and cooking.

Acknowledgements

This thesis wouldn't have been possible without the support and guidance of my adviser, Dr. Purushothaman Iyer, who with a great deal of patience and kindness helped me through out the duration of this work. Many thanks to Burak Serdar, for his timely help on various occasions.

I take this opportunity to thank the faculty and staff at NCSU, particularly in the Computer Science department, for their help, advice and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Model checking [4, 13] has been established as an extremely successful mechanical tool for automated verification, especially in hardware design. Model checking, as such, is *qualitative* as it answers checks of satisfiability as either “yes” or “no”. There are a wide range of models and applications which require *quantitative* answer to satisfiability. For instance, randomized algorithms obtain high performance at the cost of obtaining correct answers with high probability. Hence models for randomized algorithms call for quantitative analysis. The interaction geometry of modern distributed systems and network protocols requires quantitative estimates of e.g. performance and cost measures. In such cases conventional model checking based on temporal logics, like CTL fails to deliver as many relevant properties are simply not true. Thus one is forced to consider suitable extensions of logics with quantitative information; in particular probabilistic logics.

The need for quantitative model checking calls for appropriate models and expressive logics. Broadly speaking, two models known as *probabilistic transition systems(PTS)* and *probabilistic-nondeterministic systems(PNS)* have been investigated in the literature. Probabilistic transition systems(PTS) are essentially discrete time Markov chains(DTMC) where the choice of next state on a particular action is resolved by a pre-defined probability distribution. As an example of PTS consider modeling a communication protocol where the messages are delivered correctly with probability (say) 0.999. The initial quantitative analysis of PTS consisted in studying which temporal logic properties are satisfied with probability 1 [15, 6] . Subsequently, [1] considered logics pCTL and pCTL* that expressed quantitative bounds on the probability of system evolution. The logics extend the temporal operators with probabilistic operator $\mathbb{P}$, with an interpretation that the formula $\mathbb{P}_{\geq p}(\phi)$ is true at a state s if the measure of traces which satisfy ϕ is at least p. The model checking algorithms proposed in [1] can be used to determine the validity of pCTL and pCTL* formulas for DTMCs. In [7] Huth et al propose algorithms to compute lower and upper bounds on the probabilities to satisfy modal μ properties for PTS. The probability bounds are computed by using approximations for conjunction and disjunction and by reducing the quantitative analysis to a linear optimization problem. In an attempt to compute exact probabilities for modal mu formulas on PTS, [14] propose a product graph based approach which computes the probabilities as the solution of a set of (non-linear) equations.

Probabilistic transition systems are well suited to describe sequential processes which have probabilistic resolution of choices, but with the increasing use of distributed and parallel architecture of systems, and use of randomized algorithms, it becomes necessary to incorporate non-determinism inherently built into such systems. The notion of concurrency and randomization call for models which don't resolve the choices probabilistically, but make use of *adversaries (schedulers)* to resolve the internal(daemonic) and external(angelic) non-determinism. The probabilistic-nondeterministic transition systems(PNS) are models in which from a given state, on a given action label, there is a non-deterministic choice of a set of probability distributions. The probability of reaching a state on an action label is the combined result of selecting a probability distribution and then going to the state as per the selected distribution. In this thesis we explore the model checking and compositional verification of PNS.

1.1 Contributions

In [14], Cleaveland et. al. describe the *Generalized Probabilistic Logic(GPL)* for Probabilistic Transition Systems. The logic provides operators to specify unique probability bounds for the properties. IN PNS, however, there is a set of probability distributions for a given state and given action label, so it is not possible to talk in terms of exact probabilities. We extend GPL to express the quantitative measures in terms of intervals in [0,1]. We develop a model checking algorithm, *modchk fuzzy*, to compute the quantitative measures of EGPL properties. We give an implementation of a quantitative model checking tool and use it for the quantitative analysis of randomized token stabilization protocol in [8]. We also extend the work on compositional verification of PTS to PNS.

The rest of the thesis is organized as follows, in Chapter 2 we formally define the models PTS and PNS, the probability space used, and the notion of adversaries. In Chapter 3 we revise the model checking algorithm of [14] for PTS to provide a extension for the PNS. In Chapter 4 we give details of our implementation and case studies. In Chapter 5 we present Larsen and Skou's([12]) work on compositional verification and present its extension to PNS. In the last chapter we provide conclusions and directions for future work.

Chapter 2

Probabilistic-Nondeterministic Systems

2.1 Models

Given a set S, a weighing function $\pi : S \rightarrow \mathbb{R}^+$ assigns positive real numbers to elements of S. A weighing function $\pi : S \rightarrow [0,1]$ is said to be a distribution over S provided $\sum_{s \in S} \pi(s) = 1$. Let $Dist(S)$ be the set of all the distributions over S. In the following, for all $S' \subseteq S, \pi(S') = \sum_{s \in S'} \pi(s)$. Furthermore, a distribution π is called *Dirac* provided there is a $s \in S$ such that $\pi(s) = 1$. The probabilistic transition systems are defined with respect to fixed sets *Act* and *Prop* of atomic *actions* and *propositions*, respectively. The former set records the interactions the system may engage in with its environment, while the latter provides information about the states the system may enter.

Definition 2.1.1. *A probabilistic transition system is a tuple* $(S, \delta, P, I, s_{init})$

- S *is a finite set of states*

- $\delta \subseteq S \times Act \times S$ *is the transition relation*

- $P : \delta \rightarrow (0, 1]$, *the transition probability distribution, satisfies:*
  - $\forall s \in S.\forall a \in Act.(\exists s'.(s, a, s') \in \delta) \implies P(s, a, s') \in (0, 1]$
  - $\forall s \in S.\forall a \in Act. \sum_{s':(s,a,s') \in \delta} P(s, a, s') = 1$
- $I : S \rightarrow 2^{Prop}$ *is the interpretation, which records the set of propositions true at a state.*
- $s_{init} \in S$ *is the initial state.*

A PTS in a state s responds to an action a enabled by an environment by probabilistically choosing one of the a-labeled transitions available at s. The quantity $P(s, a, s')$ represents the probability with which the transition (s, a, s') is selected as opposed to other transitions labeled by a emanating from state s. Note that the conditions on P ensure that for all actions and for all states the transition probability distribution is well-defined.

Definition 2.1.2. *A probabilistic-nondeterministic transition system(PNS) is a tuple* (S, Δ, I, s_{init})*:*

- S *is a finite set of states.*
- $s_{init} \in S$ *is initial state.*
- $\Delta \subseteq S \times Act \times Dist(S)$, *is the transition relation.*
- $I :\rightarrow 2^{Prop}$ *is the interpretation, which records the set of propositions true at a state.*

Fig.2.1 shows a simple PNS, which we use as a running example for this chapter, where $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$ and $\Delta = \{(s_0, a, \pi_1), (s_0, a, \pi_2), (s_0, b, \pi_3), (s_1, c, \pi_4)\}$

When an action a is requested of the system, the system responds, on its own accord, by following one of the a actions possible from the current state. Once such a transition (s, a, π) is selected the next state reached is dictated by the probability distribution π. For the PNS of Fig.2.1, s_0 on a a can select either π_1 or π_2, and this choice determines the probability of reaching the next state.

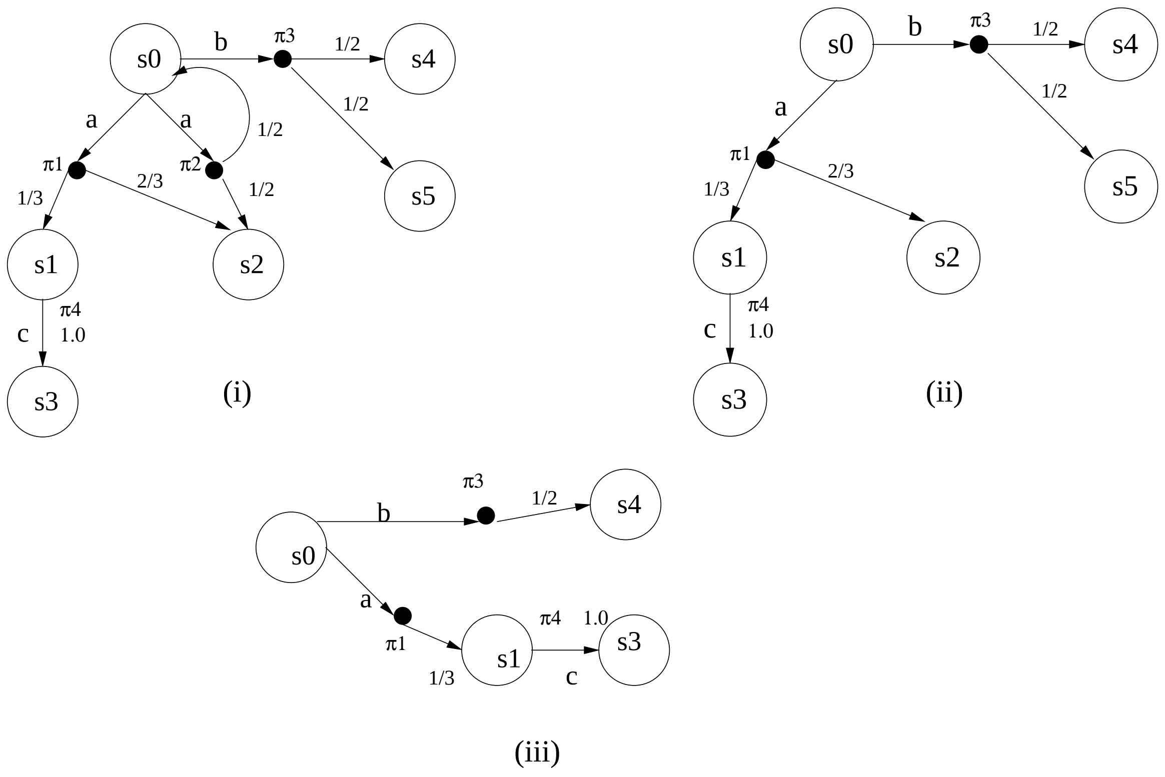

Figure 2.1: (i)A simple PNS (ii)A schedule for PNS (iii)An observation for schedule

Most of the known models of computation for non-determinism and probabilistic choice are subclasses of PNS. These include PTS which are deterministic, LTS which have no probabilistic choice, and Concurrent Markov chains which have no external non-determinism. Formally, we have:

Definition 2.1.3. *A PNS $P = (S, \Delta, s_{init})$ is said to be a* PTS *provided for each $s \in S$ and each $a \in Act$ there is at most one distribution π such that $s \xrightarrow{a} \pi$.*

A PNS is a labeled transition system *(LTS) provided all distributions are Dirac.*

A PNS reduces to a Concurrent Markov chain *when $|Act| = 1$. Finally, a PTS is a* Markov chain *if it is a PTS and $|Act| = 1$.*

The execution of PNS is defined in terms of schedulers or adversaries. A *scheduler(adversary)* is a function which resolves the non-deterministic choice. A *reactive sched-*

uler is a generalization of scheduler which takes the requested action into account while resolving the non-determinism. There can be various kinds of reactive schedulers, for example *deterministic* schedulers which at a given state and action label, always pick the same probability distribution, *policy based* schedulers which resolve the non-determinism as per pre-defined decision policies or *randomized schedulers* which resolve the nondeterminism using a random probability distribution. In this thesis we focus on randomized schedulers, and to this end we define the notion of *combined transition* of PNS that takes into account the randomized schedulers, by considering all possible linear combinations. For the following definitions we fix a PNS $P = (S, \Delta, I, s_{init})$.

Definition 2.1.4. *A* combined transition *is a triple* (s, a, π) *where* π *is a convex combination of a transitions enabled from* s*. Formally* $\pi = \sum_{t=(s,a,\pi_t)\in\Delta} \lambda_t{\cdot}\pi_t$ *where* $\sum_{t=(s,a,\pi_t)\in\Delta} \lambda_t = 1$.

For the simple PNS of fig. 2.1, a combined a transition from s_0 is $1/2\pi_1 + 1/2\pi_2$. Based on the notion of combined transition we will now define the notion of a path that a system might take:

Definition 2.1.5. *A path* σ *starting from a state* s_0 *is a possibly infinite sequence of the form* $s_0 \xrightarrow{a_0,\pi_0,r_0} s_1 \ldots \xrightarrow{a_{n-1},\pi_{n-1},r_{n-1}} s_n \ldots$*, where for all* $i \geq 0$ *the triple* (s_i, a_i, π_i) *is a combined transition and* $\pi_i(s_{i+1}) = r_i$.

Given a path σ *let* $\mathbf{fst}(\sigma) = s_0$ *denote the first state, and if* σ *is finite, let* $\mathbf{lst}(\sigma)$ *be the last state of the path. We will use* $s \xrightarrow{a,\pi,r} \sigma$ *to denote a path starting at* s *followed by* σ *provided* (s, a, π) *is a combined transition and* $\pi(\mathbf{fst}(\sigma)) = r$*. Similarly,* $\sigma \xrightarrow{a,\pi,r} s$ *denotes an extension of path* σ*. If* $\mathbf{fst}(\sigma') = \mathbf{lst}(\sigma)$*, the concatenation of two paths is denoted as* $\sigma.\sigma'$*. Path* σ' *is a prefix of* σ*, denoted by* $\sigma' \leq \sigma$*, if there exists a path* σ'' *such that* $\sigma = \sigma'.\sigma''$*. Let the set of all paths starting from state* s_0 *be denoted by* $\mathcal{P}(s_0)$*. A maximal path* σ *is either infinite, or* $\mathbf{lst}(\sigma) \not\xrightarrow{a}$ *for all* a.

For the simple PNS, $\{s_0 \xrightarrow{a,\pi_1,1/3} s_1 \xrightarrow{c,\pi_4,1.0} s_3,\ s_0 \xrightarrow{a,\pi_2,1/2} s_0 \xrightarrow{a\pi_2,1/2} \ldots\}$ are maximal paths. Sets of paths, under certain conditions, denote trees that can be looked upon as reactive schedules. We will first define a tree as a set of paths that identify it. Formally, we have:

Definition 2.1.6. *A set of s_0-rooted maximal paths $T \subseteq \mathcal{P}(s_0)$ is said to be a tree provided it is deterministic: If $\sigma \xrightarrow{a,\pi_1,r_1} s_1$ and $\sigma \xrightarrow{a,\pi_2,r_2} s_2$ are in T then $\pi_1 = \pi_2$. Define $root(T) = s_0$.*

We extend the transition notation to trees, i.e. $T \xrightarrow{a,\pi,r} T'$ if there is a $s_0 \xrightarrow{a,\pi,r} s$ in T and $T' = \{\sigma \in \mathcal{P}(s) \mid s_0 \xrightarrow{a,\pi,r} \sigma \in T\}$. If $T \xrightarrow{a,\pi,r} T'$ then T' is a *subtree* of T. A tree T is *maximal* provided there is no other tree T' such that $T \subseteq T'$.

For our running example, $\{s_0 \xrightarrow{a,\pi_1,1/3} s_1 \xrightarrow{c,\pi_4,1.0} s_3,\ s_0 \xrightarrow{a,\pi_1,2/3} s_2,\ \}$ is a tree. $\{s_0 \xrightarrow{a,\pi_1,1/3} s_1 \xrightarrow{c,\pi_4,1.0} s_3,\ s_0 \xrightarrow{a,\pi_1,2/3} s_2,\ s_0 \xrightarrow{b,\pi_3,1/2} s_4,\ s_0 \xrightarrow{b,\pi_3,1/2} s_5\}$ is a maximal tree, and a schedule. (see definition below).

Definition 2.1.7. *A s_0-rooted reactive schedule is a maximal s_0-rooted tree. Let $\mathcal{M}_{s_0}$ be the set of all maximal trees and, thus reactive schedules with s_0 as its root.*

Given a schedule T, which has resolved all non-determinism from the PNS P, we can consider sets of paths that resolve all probabilistic choice [14].

Definition 2.1.8. *Given a schedule T, a set of finite paths o is an* observation *of T if and only if:*

1. *for each $p \in o$, there exists $p' \in T$ such that p is a prefix of p', and*

2. *if $\sigma \xrightarrow{a,\pi,r} s$ and $\sigma \xrightarrow{a,\pi,r'} s'$ are in o, then $r = r'$ and $s = s'$.*

$\{s_0 \xrightarrow{a,\pi_1,1/3} s_1 \xrightarrow{c,\pi_4,1.0} s_3, s_0 \xrightarrow{b,\pi_3,1/2} s_4\}$, is an observation for the schedule, $\{s_0 \xrightarrow{a,\pi_1,1/3} s_1 \xrightarrow{c,\pi_4,1.0} s_3,\ s_0 \xrightarrow{a,\pi_1,2/3} s_2,\ s_0 \xrightarrow{b,\pi_3,1/2} s_4,\ s_0 \xrightarrow{b,\pi_3,1/2} s_5\}$(fig 2.1),

Let O_T be the set of all observations of a schedule T and let $O_s = \cup_{T \in \mathcal{M}_s} O_T$.

2.1.1 Measure space

To build a measure space for a reactive schedule T we can start with observations o and build basic cylindrical sets out of them as $B_o = \{\sigma \in T \mid \exists \sigma' \in o . \sigma' \leq \sigma\}$. The measure of such a basic cylindrical set would be the product of the probabilistic choices in o, and will be denoted as $\mathsf{m}_T(o)$. The smallest Borel field containing such basic cylindrical sets would then form the required probability space. These constructions are rather standard and can be found in [14].

In the next chapter we define the logic to specify the properties of PNS, define the function which maps the property to the set of observations which satisfy it.

Chapter 3

Extended Generalized Probabilistic Logic

The specification logic is required to be expressive enough to capture the properties of interest in a particular class of systems. The PNS allows us to model non-determinism along with the probabilistic choices. The behavior of PNS is determined by the schedules and their observations. Thus the first requirement of any specification logic for PNS is that it should be able to quantify over schedules. In the presence of multiple probability distributions for a given state and action label it is impossible to talk in terms of exact probabilities of properties. Therefore, the second requirement for the logic is that it should allow us to express range of probabilities for satisfying the property. We should be able to specify properties like *there exists a scheduler such that* $Pr(s \models \phi) \in [\alpha, \beta]$ *where* $[\alpha, \beta] \subseteq [0, 1]$. Kozen's modal μ-calculus [9] is well suited to express the qualitative properties of any system. We describe extended generalized probabilistic logic(EGPL) which extends the modal μ-calculus with quantifiers over schedules($\mathbf{E}$, $\mathbf{A}$) and describe the semantics of EGPL with respect to the observations of a schedule of the PNS. The nomenclature EGPL comes from GPL [14] which essentially extends the modal μ-calculus with probabilistic quantifiers $\mathbb{P}_{\{\geq, >\}}$ and is useful to specify properties of PTS where there is no nondeterminism.

3.1 Syntax of EGPL

The syntax for extended GPL (EGPL) is given by the following BNF-like grammar.

$$\psi \ ::= \ \langle a\rangle\psi \mid [a]\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \mu x.\psi \mid \nu x.\psi \mid \phi \mid X$$

$$\phi \ ::= \ \mathbf{E}_Y\psi \mid \mathbf{A}_Y\psi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid A \mid \neg A$$

where $Y \subseteq [0,1]$ and $A \in Prop$

The formulas generated from nonterminal ϕ are referred to as *state* formulas and ψ generate formulas as *path* formulas. The operators μ and ν bind the variables in the usual sense, and one may define the standard notion of free and bound variables. Also, we refer to an occurrence of a bound variable X in a formula as a μ-occurrence if the closest enclosing binding operator for X is μ and as a ν-occurrence otherwise. The following restrictions on GPL formulas are applied to the EGPL formulas:

- in the formula $\mathbf{E}_Y\psi$ or $\mathbf{A}_Y\psi$ the formula ψ can not contain any free variables, and
- no sub-formula of the form $\mu X.\psi(\nu X.\psi)$ may contain a free ν-occurrence(μ-occurrence) of a variable. In other words the formula must be alternation free.

3.2 Semantics of EGPL

The semantics of EGPL is described in terms of observations of schedules. Therefore, it is necessary to have quantifiers over schedules while talking about the properties of PNS. $\mathbf{E}$ is the existential quantification over the schedules and the specification $\mathbf{E}_Y\psi$ is satisfied if there exists a schedule of the PNS which satisfies the formula ψ with probability which is in the interval Y. $\mathbf{A}_Y$ formulas quantify over all schedulers and the PNS satisfies a $\mathbf{A}$ formula if all the schedules of the system satisfy the formula with probability in Y. An observation rooted at s satisfies a state formula if s satisfies the formula. The path formulas when interpreted against a schedule have a measurable set of observations that satisfy them. That is why they are also referred to as *fuzzy* formulas[14]. An observation satisfies the $\langle a\rangle\psi$ if the unique a-successor satisfies ψ. In order to formally specify the semantics of EGPL we define two mutually recursive functions as follows.

- $\Theta_s : \Psi \times \mathcal{M}_s \rightarrow O_s$, which returns the set of observations of a schedule, $T \in \mathcal{M}_s$, that satisfy a formula $\psi \in \Psi$.

- $\models \subseteq S \times \Phi$ which indicates whether a state formula is true at a state $s \in S$.

Fix a PNS $P = (S, \Delta, I, s_0)$, and fix a state $s \in S$.

Definition 3.2.1. *The function $\Theta_s(\psi, T)$ is $\emptyset$ if $\mathbf{root}(T) \neq s$. Thus, assuming that $\mathbf{root}(T) = s$ we have*

$\Theta_s(\langle a \rangle \psi, T) = \{o \in O_T \mid o \xrightarrow{a,\pi,r} o' \wedge T \xrightarrow{a,\pi,r} T' \wedge o' \in \Theta_{\mathbf{root}(T')}(\psi, T')\}$

$\Theta_s([a]\psi, T) = \{o \in O_T \mid (o \xrightarrow{a,\pi,r} o') \Rightarrow (T \xrightarrow{a,\pi,r} T' \wedge o' \in \Theta_{\mathbf{root}(T')}(\psi, T'))\}$

$\Theta_s(\psi_1 \wedge \psi_2, T) = \Theta_s(\psi_1, T) \cap \Theta_s(\psi_2, T)$

$\Theta_s(\psi_1 \vee \psi_2, T) = \Theta_s(\psi_1, T) \cup \Theta_s(\psi_2, T)$

$\Theta_s(X) = \Theta_s(\psi)$ *where* $X \mapsto \psi$

$\Theta_s(\mu X.\psi) = \bigcup_{i=0}^{\inf} M_i,$ *where* $M_0 = \emptyset$ *and* $M_{i+1} = \Theta_s(\psi[X \mapsto M_i], T)$

$\Theta_s(\nu X.\psi) = \bigcup_{i=0}^{\inf} N_i,$ *where* $N_0 = O_T$ *and* $N_{i+1} = \Theta_s(\psi[X \mapsto N_i], T)$

$$\Theta_s(\phi, T) = \begin{cases} O_T & \text{if } s \models \phi \\ \emptyset & \text{otherwise} \end{cases}$$

Once a schedule, T, resolves all the non-determinism in the PNS, the systems behave like a PTS, hence the following theorem follows directly from the result of measurability of Θ_s in [14].

Theorem 1. *For any given schedule T, $s \in S$ and $\psi \in \Psi$, $\Theta_s(\psi, T)$ is measurable.*

Definition 3.2.2. *The relation $\models \subseteq S \times \Phi$ captures the semantics of state formulas as follows:*

$s \models \mathbf{A}_Y\psi \ \textit{iff} \ \forall T \in \mathcal{M}_s, \ \mathsf{m}_T(\Theta_s(\psi, T)) \in Y$

$s \models \mathbf{E}_Y\psi \ \textit{iff} \ \exists T \in \mathcal{M}_s, \ \mathsf{m}_T(\Theta_s(\psi, T)) \in Y$

$s \models \phi_1 \wedge \phi_2 \ \textit{iff} \ s \models \phi_1 \ \textit{and} \ s \models \phi_2$

$s \models \phi_1 \vee \phi_2 \ \textit{iff} \ s \models \phi_1 \ \textit{or} \ s \models \phi_2$

$s \models A \ \textit{iff} \ A \in I(s)$

$s \models \neg A \ \textit{iff} \ A \notin I(s)$

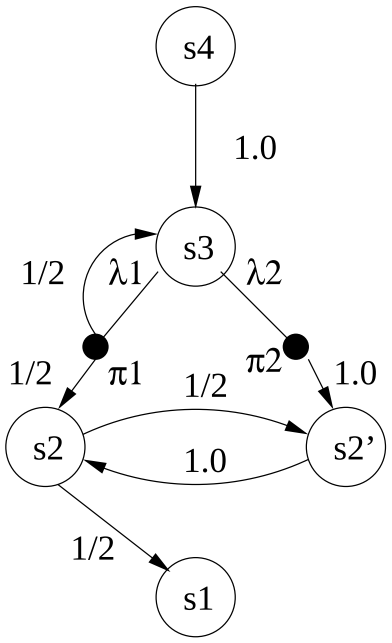

Figure 3.1: PNS for token stabilization algorithm

In order to illustrate the kind of properties that can be specified using EGPL, we consider the PNS of figure 3.1. The PNS is an abstraction of randomized token stabilization algorithm for four symmetrical processes given by Israeli and Jalofan in [8]. The system starts with all processes having the token($s4$) and tries to get to a state where exactly one process has the token($s1$). We will postpone the details of the abstraction and execution of the algorithm to chapter 5. Lets say we want to specify the following property for the system: Does the system eventually stabilize with minimum probability 1.0 ? Lets give a dummy action label a to the system, and define that all the transitions are made on a. From [5], $\psi = \mu X.((s1) \vee \langle a \rangle X)$ represents the *eventually stabilize* property. $\phi = \mathbf{A}_{[1.0,1.0]}\psi$ specifies that for all the schedulers the probability of satisfying ψ is 1.0. ϕ gives the required specification.

The quantitative analysis of EGPL properties, like the one discussed above, requires minimization(maximization) over the schedules to get the range of probability of satisfying the formula. In the next chapter we describe an algorithm to generate a system of equation whose minimum and maximum solution give the measure of the formula.

Chapter 4

Model Checking of PNS

The PNS allows to model the non-determinism and EGPL gives the specification which ranges over schedulers. The quantitative analysis of EGPL formula with respect to PNS, is essentially the computation of minimum(or maximum) probabilities of satisfying the formula over the set of schedulers. We intend to compute results of the form $Pr(s_0, \psi) \in [l, u]$ where $[l, u] \subseteq [0, 1]$. In [14], Cleaveland et. al. propose an algorithm *modchk-fuzzy*, for model checking GPL properties with respect to a PTS. The algorithm generates a product graph from the PTS and *Fisher-Ladner closure* of the GPL formula. A system of equations is generated from the product graph and one of the solutions of the equations gives the measure of the GPL formula for the PTS. A PNS and a PTS differ only in the transition probabilities where a PTS has a single probability associated with every transition, a PNS has a range of probability values corresponding to the various possible schedules of the PNS. This observation forms the basis of extending the algorithm for quantitative analysis of PNS. In the following sections we tailor the algorithm modchk-fuzzy to compute the probabilities for a restricted subset of EGPL properties with respect to PNS.

4.1 Outline of the approach

For a given start node s_0 and a schedule T of PNS L, the algorithm computes the measure of fuzzy formula ψ in three steps.

- From L, s_0, T, ψ construct a dependency graph
- From the dependency graph extract a set of constraints
- Minimize(maximize) the root variable with respect to constraints.

Before going into details of each step, two important comments are in order.

- The presence of multiple probability distributions in PNS, makes it impossible to talk in terms of exact probabilities of properties. Therefore, we are interested in computing the minimum and maximum probabilities of satisfying the property, where the minimum and maximum are computed over all possible schedules. A randomized schedule is uniquely identified by the random probability with which it resolves the non-deterministic choices, i.e., the set of values it assigns to the λ_i's(refer definition 2.1.7,2.1.4). These λ_i's specify the constraints over transition probabilities. So a maximization(minimization) of the root node of dependency graph subject to the constraints gives the schedule corresponding to the maximum(minimum) probability of satisfying the property. Thus the schedule, T, refers to the unknown schedule which gives the maximum(minimum) probability of satisfying the property. In the following discussion we assume the existence of such an optimal schedule T.
- We use EGPL without recursion for the specification of properties. The reason for the restriction is explained towards the end of the chapter. In the rest of the chapter EGPL refers to the restricted set of properties given by:
  $\psi \ ::= \ \langle a \rangle \psi \mid [a]\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \phi \mid X$
  $\phi \ ::= \ \mathbf{E}_Y \psi \mid \mathbf{A}_Y \psi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid A \mid \neg A$
  where $Y \subseteq [0,1]$ and $A \in Prop$

4.2 Graph Construction

Fix a state s_0 and a fuzzy(path) formula ψ. The first step in $modchk - fuzzy$ involves constructing a graph $PG(s_0, \psi)$ that describes the relationship between the quantity $\mathsf{m}_T(\Theta_{s_0}(\psi, T))$, that we wish to compute, and the quantities of the form $\mathsf{m}_{T'}(\Theta_s(\psi, T'))$, where s is a state reachable from s_0 and ψ' is an (appropriate) sub-formula of ψ. This graph has vertices of the form (s, F), where $s \in S$ and F is a set of fuzzy formula, The edges from (s, F) then provide "local" information regarding $\mathsf{m}_T(\Theta_s(\psi, T))$.

In order to define the graph formally we need the following notions.

Definition 4.2.1. *For a closed fuzzy formula ψ define the (Fisher-Ladner) closure, written as $Cl(\psi)$, as the smallest set of formula satisfying the following rules:*

- $\psi \in Cl(\psi)$
- *If $\psi' \in Cl(\psi)$ then*
  - *if $\psi' = \psi_1 \wedge \psi_2$ or $\psi_1 \vee \psi_2$ then $\psi_1, \psi_2 \in Cl(\psi)$*
  - *if $\psi' = \langle a \rangle \psi''$ or $[a]\psi''$ for some $a \in Act$, then $\psi'' \in Cl(\psi)$*

One may easily show that $Cl(\psi)$ contains no more elements than ψ contains sub formula.
The node set N in the graph is a subset of the set $S \times 2^{Cl(\psi)}$; that is, nodes have form (s, F), where $s \in S$ and $F \subseteq Cl(\psi)$. A node is classified according to the first rule, among the following, that it satisfies.

- (s, F) is an *empty node* if $F = \emptyset$.
- (s, F) is a *false node* if there exists a state formula $\phi \in F$ with $s \not\models \phi$ or if there exists a formula of the form $\langle a \rangle \psi'$ and there is no s' such that $s \xrightarrow{a} s'$.
- (s, F) is a *true node* if (a)it has at least one state formula $\phi \in F$ such that $s \models \phi$ or (b) it has at least one $[a]\psi \in F$ but there are no transitions of the form $s \xrightarrow{a} s'$ for any $s \in S$.
- (s, F) is an *and node* if there exists a formula $\psi_1 \wedge \psi_2 \in F$
- (s, F) is an *or node* if there exists a formula $\psi_1 \vee \psi_2 \in F$
- (s, F) is an *action node* if every formula in F has the form $\langle a \rangle \psi'$ or $[a]\psi'$

The edges in the graph are labeled by the elements drawn from the set $Act \cup \{\epsilon^+, \epsilon^-\}$ (assuming $\epsilon^+, \epsilon^- \notin Act$). Define the set of action labels in a set of formula F as $action(F) = \{a \in Act | \exists \psi . \langle a \rangle \psi \in F \vee [a]\psi \in F\}$. The edge set $E \subseteq N \times (Act \cup \{\epsilon^+, \epsilon^-\}) \times N$ is defined as follows.

- If $n = (s, F)$ is an empty node or a false node, then n is a sink node;

- else if (s, F) contains state formulas then $(s, F), \epsilon^+, (s, F)) \in E,$ where F' is F with all the state formulas deleted.
- else if $\psi = \psi_1 \wedge \psi_2 \in F$ then $((s, F), \epsilon^+, (s, F - \{\psi\} \cup \{\psi_1, \psi_2\})) \in E)$
- else if $\psi = \psi_1 \vee \psi_2 \in F$ then $((s, F), \epsilon^+, F - \{\psi\} \cup \{\psi_1\}) \in E$, $((s, F), \epsilon^+, F - \{\psi\} \cup \{\psi_2\}) \in E$, and $((s, F), \epsilon^-, (s, F - \{\psi\} \cup \{\psi_1, \psi_2\})) \in E)$;
- else if (s, F) is an action node, let $F_a = \{\psi' | \langle a \rangle \psi' \in For[a]\psi' \in F\}$. Then for any $a \in Act$ with $F_a \neq \emptyset$ and $s' \in S$ such that $\exists \pi.\ (s, a, \pi) \in \Delta \wedge \pi(s') > 0, ((s, F), a, (s', F_a)) \in E$.

The graph construction is guaranteed to terminate; this is due to the fact that all the formulas in the construction are in Fisher-Ladner closure of ψ [9].

The edges in the graph indicate a "local relationship" between its end nodes. To see this, first note that if (s, F) is false then $[[(s, F)]] = \emptyset$ and $\mathsf{m}_T(\Theta_s(F, T)) = 0$ and if it is an empty node then all observations of schedule T satisfy the formula F i.e.$[[(s, F)]] = O_T$ and $\mathsf{m}_T(\Theta_s(F, T) = 1$. If the node is an *or node* i.e. $F = F' \cup \{\psi_1 \vee \psi_2\}$ then the semantics of the logic entails that $\wedge F$ and $(\wedge F' \wedge \psi_1) \vee (\wedge F' \wedge \psi_2)$ are logically equivalent. Therefore we have

$$\mathsf{m}_T(\Theta_s(F, T)) = \mathsf{m}_T(\Theta_s(\wedge F' \wedge \psi_1, T)) + \mathsf{m}_T(\Theta_s(\wedge F' \wedge \psi_2, T)) - \mathsf{m}_T(\Theta_s(\wedge(F' \cup \{\psi_1, \psi_2\}), T))$$

This observation is encoded in ϵ^+, ϵ^- edges emanating from the or node. Similar observations can be made for other nodes except the action nodes. Under the assumption of existence of optimal schedule T the PNS reduces to PTS and we have the following results from [14], which relate the nodes to their measures.

Lemma 1. *For the product graph* $PG(s_0, \psi) = (N, E)$

- *If* (s, F) *is an empty node then*

  $\mathsf{m}_T([[s, F]]) = 1$

- *If* (s, F) *is a false node then*

  $\mathsf{m}_T([[s, F]]) = 0$

- *If (s,F) is an or-node with edges $((s,F),\epsilon^+,(s,F_1)),(((s,F),\epsilon^+,(s,F_2))$ and $(s,F),\epsilon^-,(s,F_3))$ then*

  $\mathsf{m}_T([[(s,F)]]) = \mathsf{m}_T([[(s,F_1)]]) + \mathsf{m}_T([[(s,F_2)]]) - \mathsf{m}_T([[(s,F_3)]])$

- *If (s,F) is an action node then* $\mathsf{m}_T([[(s,F)]]) = \prod_{a\in action(F)} \sum_{((s,F),a,(s',F'))\in E} Pr(s,a,s') \times \mathsf{m}_T([[(s',F')]])$

- *If (s,F) is any other node then it has a unique successor (s,F')*

  $\mathsf{m}_T([[(s,F)]]) = \mathsf{m}_T([[(s,F')]])$

4.3 Generating constraints from the graph

We give a method to generate a set of constraints from the graph. The constraints have one variable for each node of the graph. For every variable there is exactly one equation with the variable on the LHS. The transition probabilities are expressed as variables whose values are given by the values of λ_i's and corresponding probability distributions π_i. For example,(refer Fig. 3.1), $Pr(s_3,a,s_2) = y_{(s_3,a,s_2)} = \lambda_1 * \frac{1}{2} + \lambda_2 * 0$. The constraints are generated according to following rules.

1. If n is a *empty* node then $X_n = 1$;

2. If n is a false node then $X_n = 0$;

3. If there is an edge of the form (n,ϵ^+,n') then the equations for X_n is

$$X_n = \sum_{(n,\epsilon^+,n')\in E} X_{n'} - \sum_{(n,\epsilon^-,n')\in E} X_{n'}$$

4. If $n=(s,F)$ is an action node, let $A_n = \{a|(n,a,n')\in E\}$. Then,

$$\prod_{a\in A_n} \sum_{(n,a,(s',F'))\in E} (y_{s,a,s'} \cdot X_{(s',F')})$$

   where $\forall a \in A_n . \forall s' \in S.\ \ y_{s,a,s'} = \sum_i \lambda^i_{s,a} * \pi^i_{s,a}(s')$ where $\pi^i_{s,a}$ are the probability distributions from s on a and $\sum_i \lambda^i_{s,a} = 1$

The action node makes a transition with the probability defined by the scheduler. The λ_i's correspond to the unknown scheduler, T, which gives the maximum(minimum) value. The product part of the action node equation comes from the fact that transition with different action labels are independent of each other. Maximizing(minimizing) the root variable with respect to the constraints gives the maximum(minimum) measure over all possible schedules. Note that we use the optimization only for the root variable, that is we compute the minimum(maximum) value of satisfying the property starting at the root state. The probability values for internal nodes correspond to the schedule which minimizes(maximizes) the root node, and hence are not necessarily optimal for the corresponding node and sub formula pair.

Under the assumption of existence of the optimal schedule, T, the PNS becomes a PTS and the following lemma follows from Lemma 1.(refer Lemma 19 in [14])

Lemma 2. *Let* $\mathbf{\Upsilon} = \{X_n = \Upsilon_n\}$ *be the set of equations generated above, and let measure values corresponding to optimal schedule* T *be* $\mathbf{V} = \{X_n = \mathsf{m}_T(\Theta_s(\wedge F, T)\}$*, where* $n = (s, F)$*. Then* $\mathbf{V}$ *is a solution to* $\mathbf{\Upsilon}$*.*

4.4 Model Checking Example

We illustrate the model checking algorithm for PNS of fig.3.1 and the property $\phi = \mathbf{E}_{[0.25,1.0]}(\texttt{tt U[3] (s1)})$, where $\mathsf{U[k]}$ is the bounded until property defined as

$\phi_1 \mathsf{U[0]} \phi_2 = \phi_2$

$\phi_1 \mathsf{U[k]} \phi_2 = \phi_2 \vee (\phi_1 \wedge \langle \mathsf{a} \rangle (\phi_1 \mathsf{U[k-1]} \phi_2))$

The property specifies that there exists a schedule so that probability of reaching state $(s1)$ in 3 or less steps is at least 0.5. It suffices to check if the maximum probability is greater than equal to 0.25. The product graph is shown in fig.4.1. The root node is an *or node* and one of the disjuncts is a *false node* as $s_4 \not\models (s1)$. So node $(s_4, \{\langle a \rangle(\texttt{tt U[2] (s1)})\})$ is the only node to explore. Note that the nodes s_2' and s_3 can not reach s_1 in one step hence they are denoted as false nodes. Constructing the product graph in a similar fashion we get the following set of constraints to maximize the root variable.

MAX x_4;

$\mathsf{x}_4 = \mathsf{y}_1 * \mathsf{x}_3; \mathsf{y}_1 = 1.0$

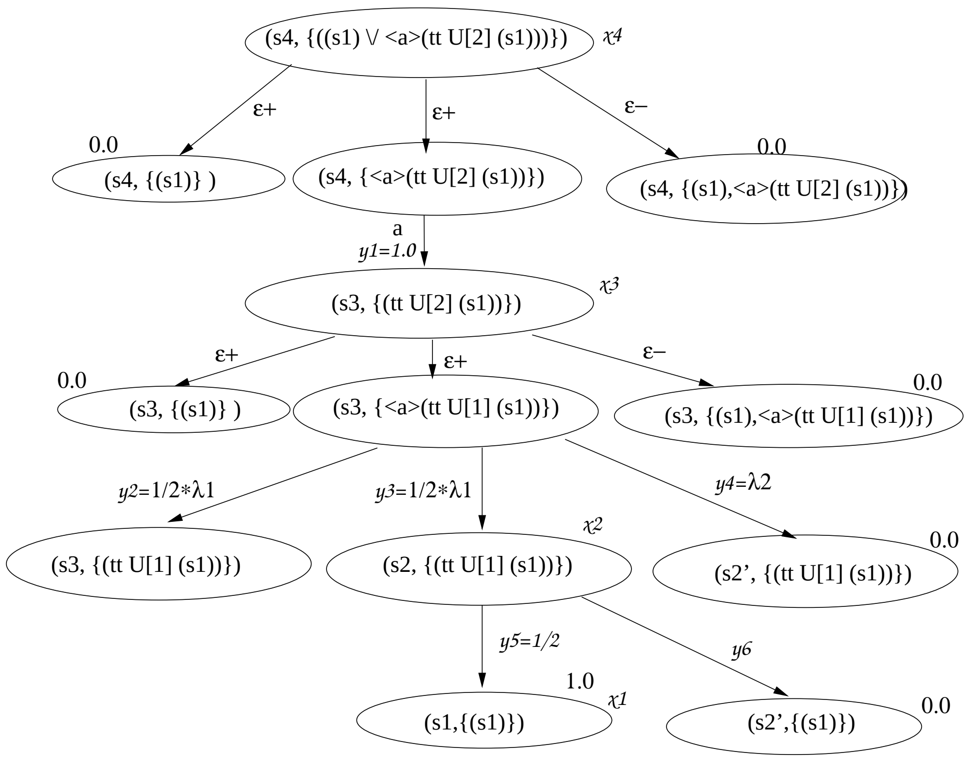

Figure 4.1: Product graph

$\mathsf{x}_3 = \mathsf{y}_2 * 0 + \mathsf{y}_3 * \mathsf{x}_2 + \mathsf{y}_4 * 0; \mathsf{y}_3 = \lambda_1/2$

where $\lambda_1 + \lambda_2 = 1.0$

$\mathsf{x}_2 = \mathsf{y}_5 * \mathsf{x}_1 + \mathsf{y}_6 * 0; \mathsf{y}_5 = 1/2$

$\mathsf{x}_1 = 1.0$

The maximum value of $x_4 = 0.25$ is obtained for the scheduler which assigns $\lambda_1 = 1.0, \lambda_2 = 0$.

The complexity of the product graph construction is linear with respect to the size of the product set of states of PNS and the Fisher-Ladner closure of the specification i.e. $O(|S| \times |Cl(\psi)|)$ There is one variable per node of the product graph so number of variables and the number of equations is also bounded by $O(|S| \times |Cl(\psi)|)$. The algorithm is an explicit graph approach, hence it is expected to be memory intensive. However it allows one to incorporate the independence of transitions having different action labels, which is not the case with the symbolic MTBDD based approach of [11] where $|Act| = 1$.

Model checking the complete EGPL logic with fix points cannot be done using the above method based on product graph. The quantitative analysis of μ and ν properties requires computation of maximum(minimum) for a set of variables corresponding to the strongly connected components in the product graph(refer [14]). The back substitution of these maximum(minimum) values does not ensure the optimal probability measures for the nodes dependent on the variables of strongly connected component.

Current systems work in composition with each other, therefore before modeling any real world problem into PNS we need composition operators for PNS. In the next chapter we develop such a calculus for PNS. We also give results on how to decompose the specification for a composite system into individual specification for subsystems. The requirement for the decomposition is that the composite system satisfies the specification if and only if the subsystems satisfy their individual specification. Such a decomposition provides a top-down approach to building a composite system and allows the subcomponents to be developed independently.

Chapter 5

Implementation and Case Studies

In the last chapters we presented the mathematical machinery needed for quantitative analysis of PNS. In this chapter we present an implementation of tool which enables us to perform quantitative analysis of EGPL properties with respect to a given PNS. Then we present a case study of a randomized self-stabilization protocol, where we model the system as PNS and compute probabilities for reachability properties. A self-stabilizing protocol for a network of processes is a protocol which when started from some possibly illegal start state returns to a legal/stable state without any outside intervention within finite number of steps. The stable states are those where there is exactly one process designated as "privileged"(has a token). The property of interest for such protocols is the minimum probability of reaching the stable state within a given time bound and maximum expected time to reach the stable state (given that the probability of reaching the stable state is 1). The protocol we consider for case study is by Israeli and Jalofan [8]. It is a token based self-stabilization protocol, where the stable state corresponds to exactly one process having the token. Before we go into the details of the protocol we give a brief description of the tool and modeling language.

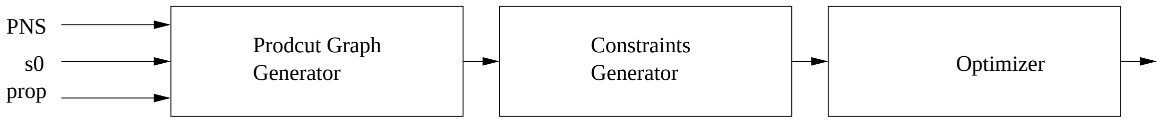

Figure 5.1: Quantitative Model Checker

5.1 Quantitative Model Checker

The model checker has three components as seen in Fig.5.1. The *product graph generator*, takes a PNS model, a start state s_0 and a path property of PNS, and generates the product graph of Fisher-Ladner closure of the property and the PNS. This product graph is used by the *constraint generator* which generates a set of constraints, and the *optimizer* maximizes(minimizes) the variable corresponding to the root node of product graph with respect to these constraints. The PNS is described using a PRISM[10] like language (see appendix A for sample input file). States are uniquely identified by the set of atomic propositions they satisfy and they are hashed to positive integers. Transitions are identified with the start state, next state action label and probability interval. A typical transition input looks like: `[]s=0 ->[a] 0.5:s=1 + 0.5:s=0` , where a is the action label. The non-determinism in the PNS is specified by allowing the model to have multiple lines where a given state on a given label goes to different probability distributions. The parser generates an interval graph from the input PNS which is used by the product graph generator. The key word `COMPUTE` provides the start state and the path property to be analyzed. We use the optimization tool LINGO[1] to solve the non-linear optimization and generate the probability measure. In the next section we present the modeling and quantitative analysis of the token based randomized self-stabilization protocol.

5.2 Randomized Self-Stabilizing Protocol(RSSP)

The protocol we consider in this section is due to Israeli and Jalofan [8]. It is a token based protocol used by a system of n processes connected in an oriented ring and having bidirectional communication. The goal of the protocol is to reach the "stable" state

[1]© LINGO is the property of LINDO SYSTEMS INC.

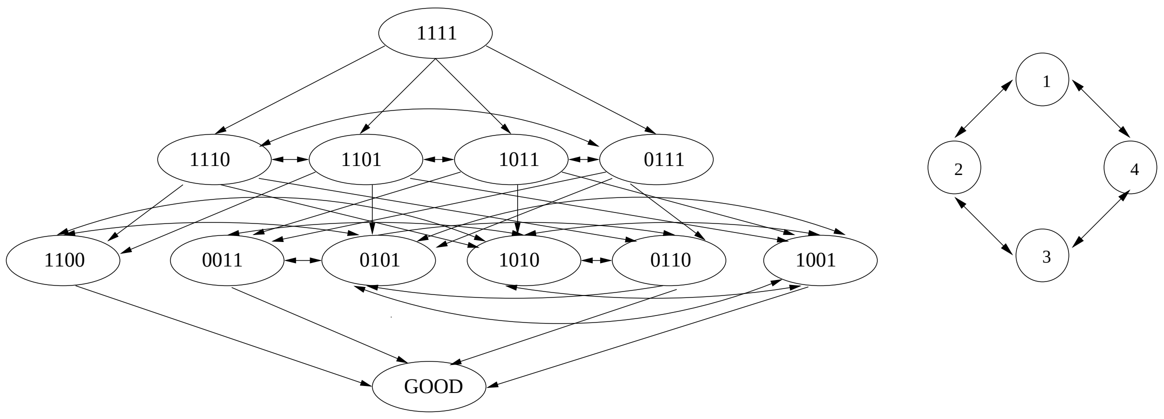

Figure 5.2: PNS for 4 processes token based randomized self-stabilization protocol

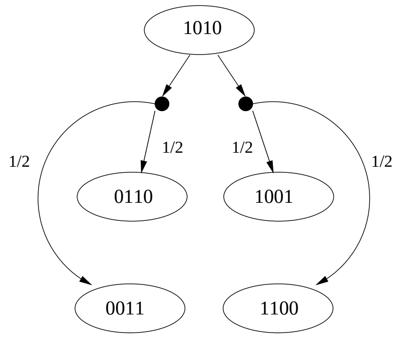

Figure 5.3: A typical execution step in the protocol

starting from an ”illegal” state. Stable state is the one where exactly one process has the token and an illegal state is the one where more than one processes have token. The processes having the token are called *active* processes. The scheduler randomly picks one of the active processes and this process propagates the token towards its left or right with equal probability. When a processes has more than one token the tokens are merged into a single token.

A typical graph based model of such a system with 4 processes is shown in figure 5.2. To illustrate a transition of the system, consider state (1010), the scheduler randomly picks one of the active processes 1 or 3 (processes numbered left to right 1 to n). If process 1 is selected then with 1/2 probability it passes the token to process 4, i.e. next state is

(0011), or with 1/2 the probability it goes to state (0110). Similar execution can be seen for the choice of process 3 in fig.5.3

It is easy to see that the size of the graph is exponential in the number of processes, $O(2^n)$. Therefore, even a small model of such system would generate large of number states in the product graph which will proportionately generate large number of constraints and variables. We focus on systems with 4 and 5 processes and exploit symmetry to reduce the number of states while preserving the quantitative behavior of the system.

5.2.1 Abstraction based model

The evolution of RSSP from a particular state depends on the neighborhood structure of each process. By neighborhood structure we mean whether the process has active or inactive process to its left and right.[2]

- If an active process has an active process on its left and right then the system evolves to a state with one less number of tokens with probability 1.
- If it is adjacent to one active and one inactive process then with half probability it evolves to one less number of tokens and half probability it goes to state with the same number of tokens.
- If both its neighbors are inactive then with probability 1 it goes to a state with same number of tokens.

So the first requirement of an abstraction for RSSP is that the set of states grouped together must have same number of tokens The second requirement is that they should have the same neighborhood structure for all the component processes. The abstraction described in the section is based on the observation that if n processes are arranged in an oriented ring where m are active and $n-m$ are inactive then, finite number of rotations of the processes merely relabels the processes while preserving their neighborhood structures. The finite number of rotations refers to the bijective function which relabels the active(inactive) processes of one state to the active(inactive) processes of another state while preserving the neighborhood structure.

[2]Note that every process has the same sense of left and right as the ring is oriented.

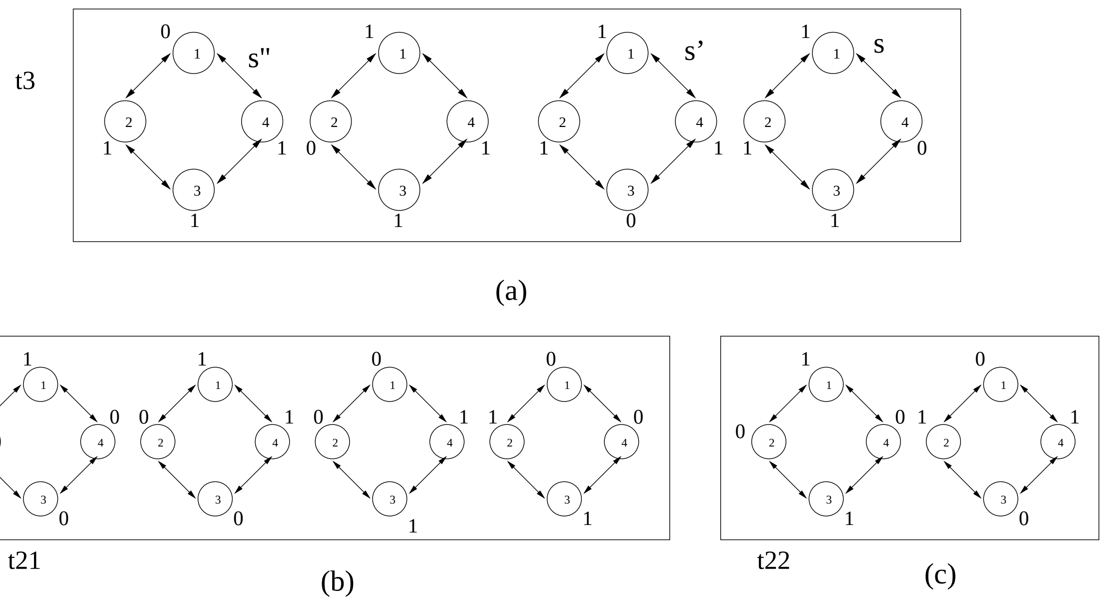

Figure 5.4: Equivalent states of 4 process PNS

Consider the 4 process PNS for RSSP, the state space can be partitioned into four disjoint subsets, $\{t_4, t_3, t_2, t_1\}$, where t_i corresponds to the state having i tokens (see fig.5.4). There are $\binom{4}{i}$ states corresponding to t_i for $i = 1$ to 4. There are 4 states where system can have 3 tokens and 6 states where the system has 2 tokens. We claim that all the 4 states of t_3 are identical with respect to qualitative properties. To see this note that all the 4 states can be obtained by rotation of each other. For example consider state $s = (1, 1, 1, 0)$, a rotation takes the state to $s' = (1, 1, 0, 1)$ or $s" = (0, 1, 1, 1)$, depending on the direction of rotation (fig.5.4). In s, process p_3 has the token and is adjacent to process p_4 on right which doesn't have the token and process p_2 on left which has the token. Similar configuration can be seen for process $p2$ in s' and process $p4$ in $s"$. So an execution of PNS which chooses p_3 in s can be simulated by choice of $p_2(p_4)$ in $s'(s")$. Similar mappings can be established for p_1, p_2 and p_4. Hence all the states in t_3 are identical with respect to qualitative behavior of the system. However, subset t_2 has to be partitioned into subsets t_{21} and t_{22}, because elements in t_{21} cannot be obtained by finite rotation of elements in t_{22} and vice versa. So, we have the following partition of state space $\mathsf{P(S)} = \{\mathsf{t_4, t_3, t_{21}, t_{22}, t_1}\}$. The following theorem establishes that the partition is probabilistic bi-simulation as described in [3].

Theorem 2. *Consider the relation, R, induced by the partition, $\mathsf{P}(\mathsf{S})$, sRs' iff $s, s' \in t_i$. R is a probabilistic bi-simulation.*

Proof.

- R is an equivalence relation.
- Let sRs'. The transition probability, $Pr(s, a, r) = \sum_i \lambda_i Pr(p_i, a, r)$ where $1 \leq i \leq 4$ is the number of active processes in s. The same transition probability can be obtained from s' as $Pr(s, a, r') = \lambda_1 * Pr(\mathcal{B}(p_1), a, r) + \lambda_2 * Pr(\mathcal{B}(p_2), a, r) + \lambda_3 * Pr(\mathcal{B}(p_3), a, r)$. where rRr' and $\mathcal{B}$ is the bijective function corresponding to finite rotation which relabels the processes in s to processes in s'.

$\square$

5.2.2 Experimental Results

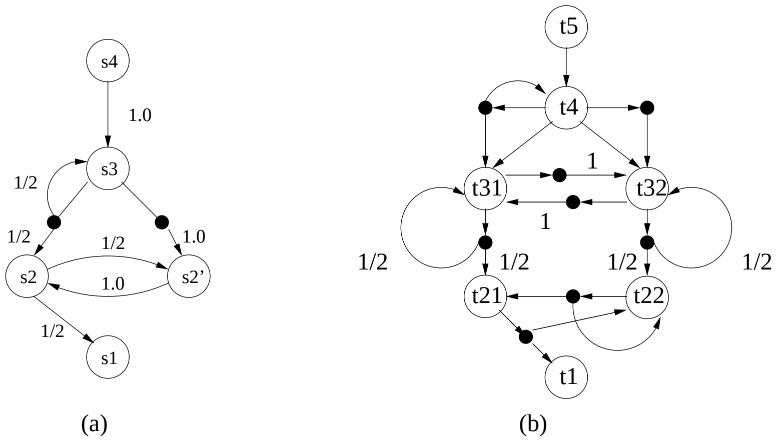

Figure 5.5: Abstract model of RSSP (a) 4 process system (b)5 process system

The model obtained by using the abstraction for 4 process RSSP is given in Fig.(5.5(a)). We use the above abstraction to compute probability of reaching the stable state in at most x steps, starting from the state where all the 4 processes have tokens(t_4).

Each step corresponds to the scheduler randomly picking up an active processes and the process then passing the token to its left or right with equal probability. The property is specified using the bounded until operator as follows

$$(\mathtt{tt}\ \mathsf{U}_{\mathsf{x}}\ (\mathsf{t}_1 = 1))$$

Table 5.1 gives the minimum(MIN) and maximum(MAX) probability of reaching the stable state in x steps. An interesting property to note from Table 5.1 is that the minimum (maximum) probabilities of satisfying the bounded until properties under randomized schedulers is the same as minimum(maximum) probabilities obtained by considering non-deterministic schedulers[3]. This stands as a witness to the theorem in [2], which states that for bounded until properties it is sufficient to minimize or maximize over the non-deterministic schedules.

The number of variables and constraints grow linearly with respect to the number of steps (see fig5.9). This is expected since the number of nodes in the product graph varies linearly with the closure of formula, which keeps increasing with steps, if the system is kept constant (refer chapter 4). The memory usage of LINGO varies linearly with the number of constraints and variables(see fig5.7). The system has been optimized to reduce the number of redundant equations of the form $\mathsf{x}_\mathsf{i} = \mathsf{x}_\mathsf{j}$ and $\mathsf{x}_\mathsf{i} = 0$. This optimization is possible in the absence of recursion as every node(variable) of the product graph is referred exactly once in the LHS and exactly once in the RHS of the constraints. The non-linear constraints correspond to the randomized scheduling at state t_3. The number of nodes corresponding to t_3 in the product graph, increase linearly with the number of steps and hence the number of non-linear constraints also increase linearly.

Fig. 5.10 gives a comparison of the minimum probability of stabilizing 4 process and 5 process RSSP. The number of steps required to stabilize with probability 1 is almost doubled from 4 process to 5 process system.

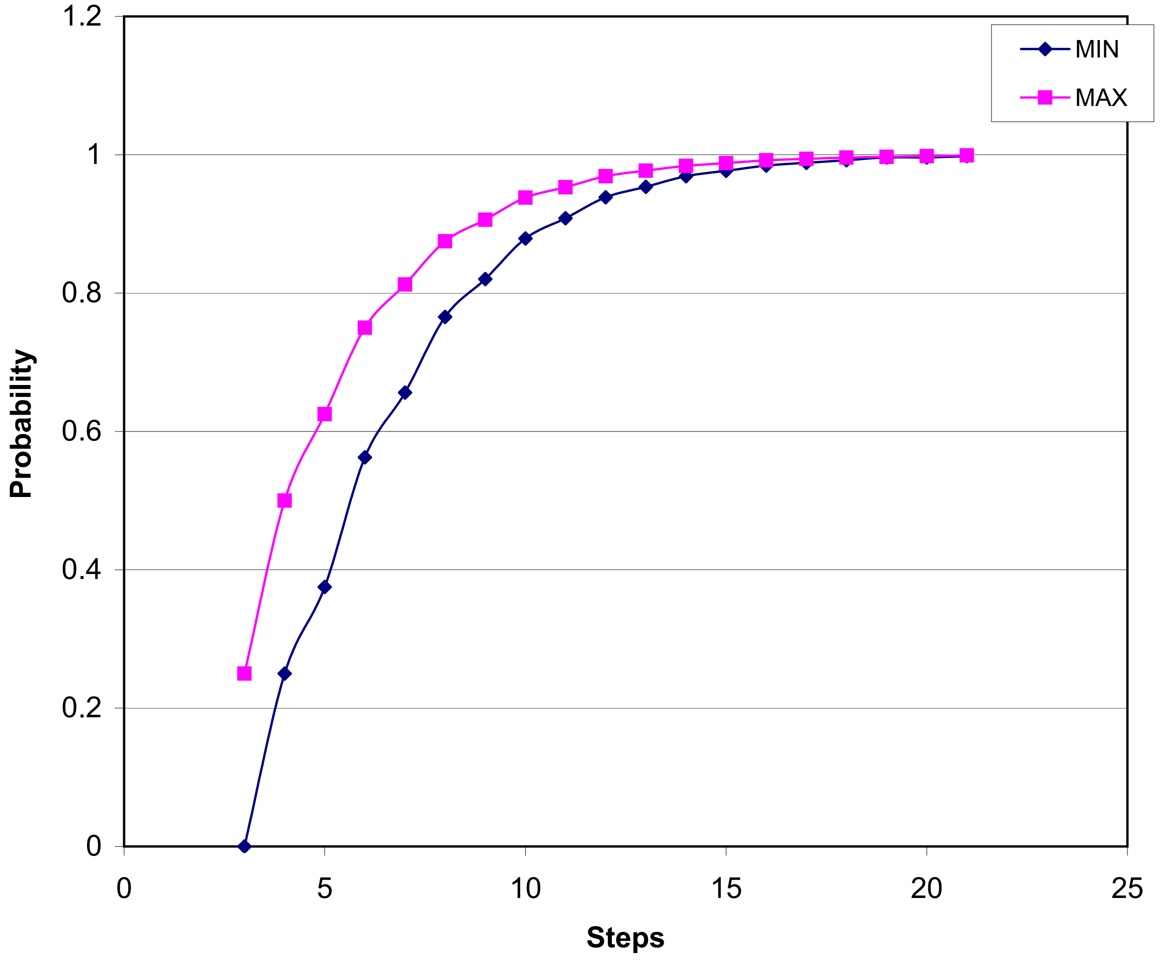

Figure 5.6: Minimum and maximum probability of stabilizing in x steps for 4 process RSSP

steps	MIN	MAX	#vars	#cons	#nlcons	memory(K)
3	0	0.25	14	28	1	11
4	0.25	0.5	25	48	2	18
5	0.375	0.625	36	68	4	25
6	0.5625	0.75	47	88	6	32
7	0.65625	0.8125	58	108	8	39
8	0.765625	0.875	69	128	10	47
9	0.8203125	0.906	80	148	12	54
10	0.8789062	0.938	91	168	14	61
11	0.9082031	0.953	102	188	16	68
12	0.9384765	0.969	113	208	18	75
13	0.9536133	0.977	124	228	20	83
14	0.9689941	0.984	135	248	22	91
15	0.9766846	0.988	146	268	24	97
16	0.984436	0.992	157	288	26	104
17	0.9883118	0.994	168	308	28	111
18	0.9922028	0.996	179	328	30	118
19	0.9960938	0.997	190	348	32	125
20	0.9962158	0.998	201	368	34	132
21	0.9980469	0.999	212	388	36	139

Table 5.1: Quantitative Analysis of RSSP for 4 process system

steps	MIN	MAX	#vars	#cons	#nlcons	memory(K)
4	0	0.125	43	78	4	26
6	0.094	0.406	89	160	14	55
8	0.141	0.633	135	242	24	83
9	0.211	0.703	158	283	29	98
10	0.324	0.76	181	324	34	112
12	0.431	0.843	227	406	44	139
15	0.664	0.917	296	529	59	185
18	0.755	0.956	365	652	74	226
20	0.826	0.968	411	734	84	253
22	0.871	0.979	457	816	94	280
25	0.922	0.99	526	939	109	322
30	0.968	0.997	641	1144	134	399
35	0.987	0.999	756	1349	159	470
40	0.995	1	871	1554	184	540

Table 5.2: Quantitative Analysis of RSSP for 5 process system

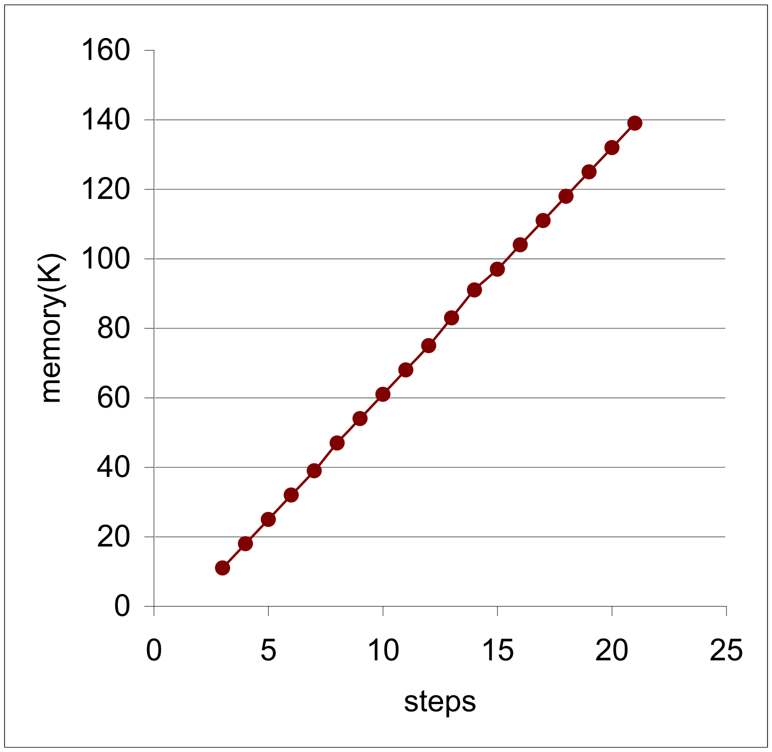

Figure 5.7: Memory usage of LINDO to compute probability of stabilizing in x steps for 4 process RSSP

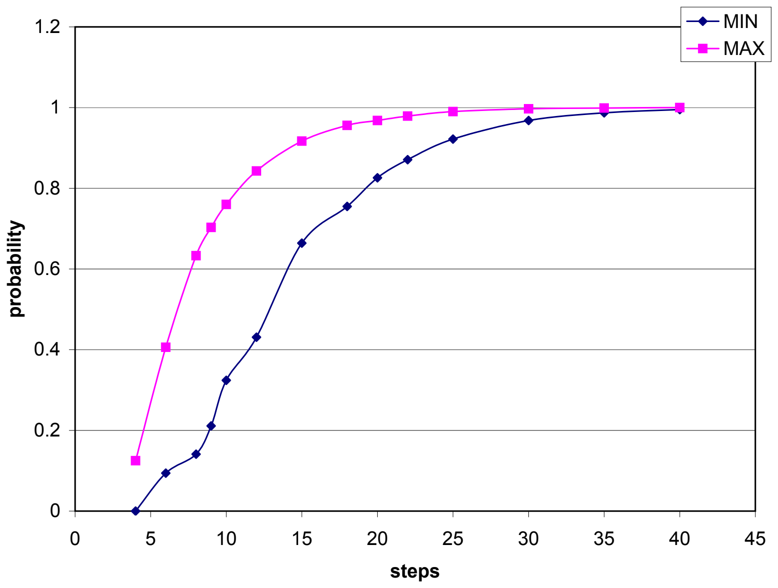

Figure 5.8: Minimum and maximum probabilities of stabilizing in x steps for 5 process RSSP

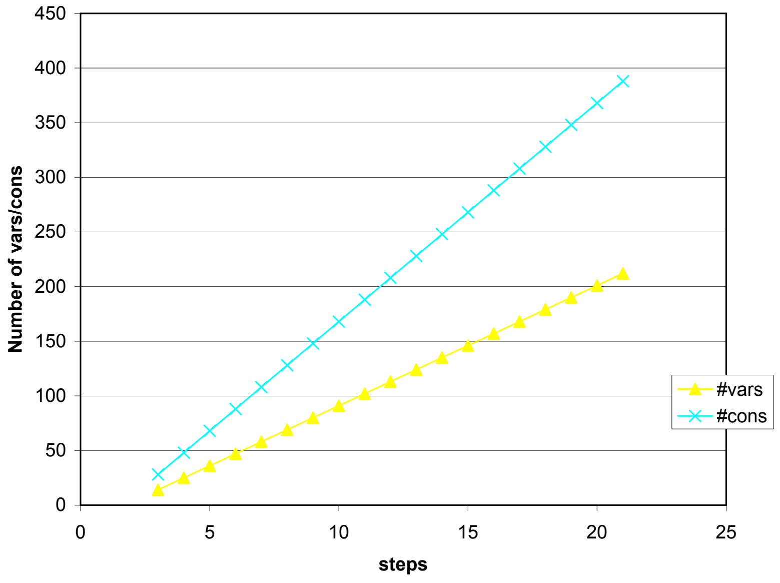

Figure 5.9: Number of variables and constraints generated from the product graph

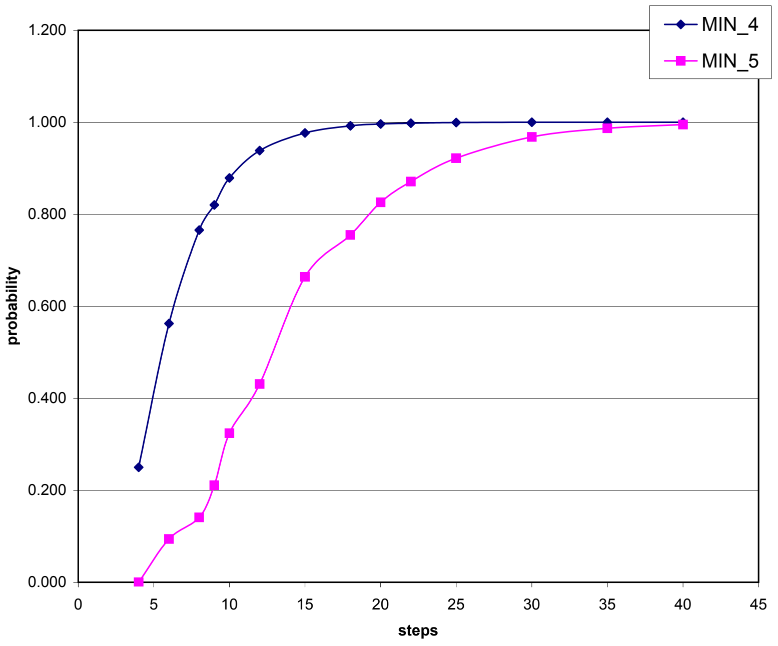

Figure 5.10: Comparison of probabilities of stabilizing in x steps for 4 and 5 process RSSP

Chapter 6

Compositional Verification of PNS

The study of compositionality is an essential component of top-down design methodology. In this approach we wish to decompose the specification of a composite system into sufficient and necessary specifications of the components of a system. Consider a composite system $P \triangle X$, where $\triangle$ is a process algebra operator and X is the system is to be designed(say X). Let F be the required specification for the composite system. We wish to compute individual sub-specifications F_X, for the unknown component such that the following is satisfied.

$$X \models F_X \Leftrightarrow P \triangle X \models F$$

Clearly, while maintaining soundness we want the specification $F_X$to be as weak as possible so that we have greater latitude with possible implementation of X. In [12], Larsen et.al. proposed a decomposition scheme for extended PML formula and SCCS like algebra for PTS. In this work we develop a similar calculus $CNPP$, and add a non-deterministic composition operator for the PNS composition. We then define the decomposition operator $\mathcal{W}_\triangle$ which gives the sub specification for the unknown component. In the end we prove a theorem which states that the decomposition is sound and is sufficiently weak.

6.1 Calculus for PNS

Definition 6.1.1. *Let Act be a set of actions. Then the calculus of non-deterministic probabilistic processes (CNPP) over a given set of actions Act has the following syntax*

$$P ::= \mathbf{0} | a.P | P \oplus Q | P \oplus_{\mu} Q | P \times Q$$

In the above definition $\mathbf{0}$ denotes completely inactive process, whereas $a.P$ can perform the action a and then behave like P. The process $P \times Q$ can only make a transition $c = (a, b)$ when both the components P and Q perform their corresponding actions. The probabilistic summation construct $P \oplus_{\mu} Q$, selects the process $P(Q)$ with probability $\mu(1-\mu)$ if both P and Q have transition on the action label a, else it selects the process having the transition on a with probability 1. If neither of the processes has a transition on a then it selects neither of them. The non-deterministic summation operator $P \oplus Q$ selects P with probability $\lambda \in [0,1]$ and Q with probability $1 - \lambda$ if both have transitions on action a, otherwise the behavior is same as probabilistic summation.

Definition 6.1.2. *Let Act be a set of actions. In the presence of action based non-deterministic choices we cannot calculate the exact probabilities, therefore we define the transition probabilities in terms of intervals in* $[0, 1]$*. Let NPr be the set of processes. The transition probability intervals π are defined as follows:*

$$\begin{aligned}
\pi(0, a, P) &= [0,0] \text{ for all } a \in \mathit{Act} \text{ and for } P \in NPr \\
\pi(a.P, b, Q) &= \begin{cases} [1,1] & \text{if } P = Q, b = a \\ [0,0] & \text{otherwise} \end{cases} \\
\pi(P \times Q, \langle a, \bar{a} \rangle, P' \times Q') &= \begin{cases} [l, u] \quad \text{where} & l = l_p * l_q, \\ & u = u_p * u_q, \\ & [l_p, u_p] = \pi(P, a, P'), \\ & [l_q, u_q] = \pi(Q, \bar{a}, Q'] \\ [0,0] \quad \text{otherwise} & \end{cases}
\end{aligned}$$

$$\pi(P \oplus Q, a, R) = \begin{cases} [l,u] & \text{where} \quad l = min(l_p, l_q), \\ & \qquad\quad\; u = max(u_p, u_q), \\ & \qquad\quad\; [l_p, u_p] = \pi(P, a, R), \\ & \qquad\quad\; [l_q, u_q] = \pi(Q, a, R) \text{ if } P \rightarrow^a, Q \rightarrow^a \\ [l_p, u_p] & \text{if } Q \not\rightarrow^a \\ [l_q, u_q] & \text{if } P \not\rightarrow^a \\ [0,0] & \text{otherwise} \end{cases}$$

$$\pi(P \oplus_\mu Q, a, R) = \begin{cases} [\mu l_p + (1-\mu) l_q, \mu u_p + (1-\mu) u_q] & \text{where } [l_p, u_p] = \pi(P, a, R), \\ & [l_q, u_q] = \pi(Q, a, R) \text{ if } P \rightarrow^a, Q \rightarrow^a \\ [l_p, u_p] & \text{if } Q \not\rightarrow^a \\ [l_q, u_q] & \text{if } P \not\rightarrow^a \\ [0,0] & \text{otherwise} \end{cases}$$

For $\pi(P, a, P') = [l, u]$, define $min(max)\pi(P, a, P') = l(u)$. For the non-deterministic summation operator, the probability $\pi(P \oplus Q, a, R) = [l, u]$ where l is the minimum and u is the maximum of the two probabilities $\pi(P, a, R)$ and $\pi(Q, a, R)$, which corresponds to combined transition of the PNS where the next state transition probabilities are defined by the convex combination of available probability distributions.

In the next section we define a variant of extended PML([12]) which is used as the specification logic.

6.2 A variant of extended PML

Syntax of the logic is given as follows

$$F ::= \texttt{tt} | c | F \wedge G | \neg F | [\langle a \rangle_{[x_1, y_1]} F_1, ..., \langle a \rangle_{[x_k, y_k]} F_k \; where \; \phi(\mathbf{x}, \mathbf{y})] | \sigma c.F \text{ where } \sigma = \{\mu, \nu\}$$

The logic essentially puts constraints on the minimum and maximum transition probabilities at each step. These constraints, as we will see, define the intervals of transition for the unknown system. Semantics of the logic is given with respect to the set of probabilistic non-deterministic processes NPr.

- $[[\texttt{tt}]] = NPr$
- $[[c]] = [[F]][c \mapsto F]$

- $[[\sigma c.F]] = [[F]]$ where $[\sigma c.F/c]$
- $[[F \wedge G]] = [[F]] \cap [[G]]$
- $[\![[\langle a\rangle_{[x_1,y_1]}F_1, ..., \langle a\rangle_{[x_k,y_k]}F_k \; where \; \phi(x_1, y_1, \dots, x_k, y_k)]]\!]$
  $= \{P \in V| \;\; \phi(min\{\pi(P, a, [[F_1]])\}/x_1, max\{\pi(P, a, [[F_1]])\}/y_1, \dots,$
  $min\{\pi(P, a, [[F_k]])\}/x_k, max\{\pi(P, a, [[F_k]])\}/y_k)\}$

We now have the framework to answer the following question.
Given a PNS P, an unknown PNS X, and the composition operator $\triangle \in \{a., P\oplus_\mu, P\oplus, P\times\}$, compute $\mathcal{W}_\triangle(F)$ such that $\triangle X \models F$ iff $X \models \mathcal{W}_\triangle(F)$

6.3 Decomposing Formulas

Definition 6.3.1. *Let $a \in Act$, $\mu \in]0, 1[$, and P a process. We define the transformers $\mathcal{W}_\triangle$ inductively as follows:*

$$\mathcal{W}_\triangle(tt) = tt$$

$$\mathcal{W}_\triangle(c) = \mathcal{W}_\triangle(F) \; where \; [c \mapsto F]$$

$$\mathcal{W}_\triangle(\sigma c.F) = \mathcal{W}_\triangle(F)[c \mapsto \sigma c.F]$$

$$\mathcal{W}_\triangle(F \wedge G) = \mathcal{W}_\triangle(F) \wedge \mathcal{W}_\triangle(G)$$

$$\mathcal{W}_\triangle(\neg F) = \neg\mathcal{W}_\triangle(F)$$

For $F = [< a >_{[x_1,y_1]} F_1, ..., < a >_{[x_n,y_n]} F_n \; where \; \phi(\mathbf{x}, \mathbf{y})]$

$$\mathcal{W}_{b.}(F) = \begin{cases} tt & ; b \neq a, \phi(\mathbf{0},\mathbf{0}) \\ ff & ; b \neq a, \neg\phi(\mathbf{0},\mathbf{0}) \\ ff & ; b = a, \Gamma = \{\} \\ \bigvee_{\nu\in\Gamma} (\bigwedge_{\nu_i=1} F_i \wedge \bigwedge_{\nu_i=0} \neg F_i) & ; otherwise \end{cases}$$

where Γ *denotes the set of tuples* $\nu = (\nu_1, \ldots, \nu_n)$ *where each* $\nu_i = x_i = y_i = 0$ *or* 1 *such that* $\phi(\mathbf{x}, \mathbf{y})$ *holds.*

$$\mathcal{W}_{P\oplus_\mu}(F) = \begin{cases} F & ; P \overset{a}{\not\to} \\ \neg < a >_{[0,1]} tt \vee G_{P,F} & ; P \xrightarrow{a} \ and\ P \models F \\ G_{P,F} & ; P \xrightarrow{a} \ and\ P \not\models F \end{cases}$$

where $G_{P,F} = [< a >_{[x_1,y_1]} F_1, ..., < a >_{[x_n,y_n]} F_n$ *where* $\phi(\ldots, (\mu * min\{\pi(P, a, [[F_i]])\} + (1-\mu) * x_i)/x_i, \ldots, (\mu * max\{\pi(P, a, [[F_i]])\} + (1-\mu) * y_i)/y_i, \ldots)]$

$$\mathcal{W}_{P\oplus}(F) = \begin{cases} F & ; P \overset{a}{\not\to} \\ \neg < a >_{[0,1]} tt \vee G_{P,F} & ; P \xrightarrow{a} \ and\ P \models F \\ G_{P,F} & ; P \xrightarrow{a} \ and\ P \not\models F \end{cases}$$

where $G_{P,F} = [< a >_{[x_1,y_1]} F_1, ..., < a >_{[x_n,y_n]} F_n$ *where*

$\phi(\ldots, min(min\{\pi(P, a, [[F_i]])\}, x_i)/x_i, \ldots, max(max\{\pi(P, a, [[F_i]])\}, y_i)/y_i, \ldots)]$

In the following $< a >$ *denotes the ordered pair* $< b, c >$*.*

$$\mathcal{W}_{P\times}(F) = \begin{cases} tt & ; P \overset{b}{\not\to} \ and\ \phi(\mathbf{0},\mathbf{0}) \\ ff & ; P \overset{b}{\not\to} \ and\ \neg\phi(\mathbf{0},\mathbf{0}) \\ \bigwedge_{1\le i\le M, 1\le j\le n} < c >_{[x_{i,j}, y_{i,j}]} \mathcal{W}_{P_i\times}(F_j) \ where\ \phi' \ holds & ; P \xrightarrow{b} \end{cases}$$

where, P_i*'s are the* $b-derivatives$ *of* P *such that,* $P \underset{[\alpha_i,\beta_i]}{\xrightarrow{b}} P_i$ *and*

$\phi' = \phi(\ldots, \underset{i=1\ to\ M}{min}\{\alpha_i * x_{i,j}\}/x_j, \ldots, \underset{i=1\ to\ M}{max}\{\beta_i * y_{i,j}\}/y_j, \ldots)$

Theorem 3.

$$\triangle(X) \models F \textit{ iff } X \models \mathcal{W}_{\triangle}(F)$$

Proof. By induction on the structure of F.

- $F = \mathtt{tt}$, vacuously true.

- $F = c$, under assumption that there are no unbounded variables, let $[c \mapsto F']$

  $\triangle(X) \models c$

  $\Leftrightarrow \triangle(X) \models F'$

  $\Leftrightarrow X \models \mathcal{W}_{\triangle}(F'),$ by ind. hyp.

- $F = \neg F$

  $\triangle(X) \models \neg F$

  $\Leftrightarrow \triangle(X) \not\models F$

  $\Leftrightarrow X \not\models \mathcal{W}_{\triangle}(F)$, by ind. hyp.

  $\Leftrightarrow X \models \neg\mathcal{W}_{\triangle}(F)$

- $F = F \wedge G$

  $\triangle(X) \models F \wedge G$

  $\Leftrightarrow \triangle(X) \models F \wedge \triangle(X) \models G$

  $\Leftrightarrow X \models \mathcal{W}_{\triangle}(F) \wedge X \models \mathcal{W}_{\triangle}(G)$, by ind. hyp.

  $\Leftrightarrow X \models \mathcal{W}_{\triangle}(F) \wedge \mathcal{W}_{\triangle}(G)$

- $F = [< a >_{[x_1,y_1]} F_1, \ldots, < a >_{[x_1,y_1]} F_1$ where $\phi(\mathbf{x}, \mathbf{y})]$

  There are following four cases depending on the operator.

1. $b.X \models F$ iff $X \models \mathcal{W}_{b.}(F)$.

   The base cases are trivially true. Suppose that $X \models \bigvee_{\nu \in \Gamma} (\bigwedge_{\nu_i = 1} F_i \wedge \bigwedge_{\nu_i = 0} \neg F_i)$ where $\nu \in \Gamma$ is the set of n-tuples such that $\nu_i = x_i = y_i = 0$ *or* 1[1] and $\phi(\mathbf{x}, \mathbf{y})$ holds. Then on making the transition on b, we enter process X which models exactly those F_i such that the condition of ϕ holds. Hence, $b.X \models F$.

   Suppose that $b.X \models F$, and let i range over the formulas such that $X \models F_i$ and j range over the formulas such that $X \not\models F_j$. Then $X \models \bigwedge_i F_i \wedge \bigwedge_j \neg F_j$. Now the condition ϕ determines the set of F_i's and F_j's. Note that there can various such partitions satisfying the condition ϕ, and the set Γ ranges over all such tuples. Hence $X \models \mathcal{W}_{b.}(F)$.

2. $P \oplus X \models F$ iff $X \models \mathcal{W}_{\triangle}(F)$

   If $P \overset{a}{\not\rightarrow}$ then X has to model F, as the scheduler will always schedule X. If $P \xrightarrow{a} andP \models F$ then either $X \models \neg < a >_{[0,1]}$ i.e. $\overset{a}{\not\rightarrow}$ or $X \models G_{P,F}$. Suppose that $X \models G_{P,F}$. $G_{P,F}$ is the same as F except that x_i is replaced by $min(min\{\pi(P, a, [\![F_i]\!])\}, x_i)$ and y_i by $max(max\{\pi(P, a, [\![F_i]\!])\}, y_i)$. By definition of $\oplus$, $\pi(P \oplus X, a, [\![F_i]\!]) = [l_i, u_i]$ where $l_i = min(min\{\pi(P, a, [\![F_i]\!])\}, x_i)$ and $u_i = max(max\{\pi(P, a, [\![F_i]\!])\}, y_i)$. From the supposition, $\phi(\mathbf{l}, \mathbf{u})$ is true, hence $P \oplus X \models F$.

   Similar argument holds for the only if part.

3. $P \oplus_\mu X \models F$ iff $X \models \mathcal{W}_{\triangle}(F)$

   Follows from 2.

[1]Since $b.$ is a deterministic operator minimum and maximum probabilities are equal

4. $P \times X \models F$ iff $X \models \mathcal{W}_{\triangle}(F)$

If $P \overset{b}{\not\rightarrow}$ then by definition of $P \times X$, there is no transition on $< b, c >$. So, if $\phi(\mathbf{0}, \mathbf{0}$ is true then P can be lock-step composed with any X and the system will satisfy F. Otherwise, if $\phi(\mathbf{0}, \mathbf{0})$ is false then no system X when coupled with P would satisfy the formula.

If $P \xrightarrow{b}$, then suppose

$X \models \bigwedge_{1 \leq i \leq M, 1 \leq j \leq n} < c >_{[x_{i,j}, y_{i,j}]} \mathcal{W}_{P_i \times}(F_j)$ where ϕ' holds

where, P_i's are the $b - derivatives$ of P such that, $P \underset{[\alpha_i, \beta_i]}{\xrightarrow{b}} P_i$ and

$\phi' = \phi(\ldots, \underset{i=1 \text{ to } M}{min}\{\alpha_i * x_{i,j}\}/x_j, \ldots, \underset{i=1 \text{ to } M}{max}\{\beta_i * y_{i,j}\}/y_j, \ldots)$

By the definition of the $\times$ operator, $\pi(P \times X, < b, c >, P_i \times [[\mathcal{W}_{P_i \times}(F_j)]]) = [\alpha_i * x_{i,j}, \beta_i * y_{i,j}]$ and by induction hypothesis, $P_i \times [[\mathcal{W}_{P_i \times} F_j]] \models F_j$. So $\underset{i=1 \text{ to } M}{min}\{\alpha_i * x_{i,j}\}$ gives the minimum probability of $P \times X$ satisfying $< b, c > F_j$ and by ϕ' this minimum probability satisfies ϕ. Similar argument holds for the maximum probability. Hence $P \times X \models F$.

$P \times X \models F$

$\Leftrightarrow P_i \times X' \models F_j$ for all $1 \leq i \leq M, 1 \leq j \leq n$

$\Leftrightarrow X' \models \mathcal{W}_{P_i \times}(F_j)$ for all $1 \leq i \leq M, 1 \leq j \leq n$ by ind. hyp.

$\Leftrightarrow X \models \bigwedge_{1 \leq i \leq M, 1 \leq j \leq n} < c >_{[x_{i,j}, y_{i,j}]} \mathcal{W}_{P_i \times}(F_j)$

$\pi(P \times X, < b, c >, P_i \times [[\mathcal{W}_{P_i \times}(F_j)]]) = [\alpha_i * x_{i,j}, \beta_i * y_{i,j}]$

$\therefore min(\pi(P \times X, < b, c >, [[F_j]]) = min\{\alpha_i * x_{i,j}\}$

$similarly\ max(\pi(P \times X, < b, c >, [[F_j]]) = max\{\beta_i * y_{i,j}\})$

$\therefore \phi \Rightarrow \phi'$

Hence $X \models \mathcal{W}_{P\times}(F)$

$\square$

6.4 A Signal Handler Example

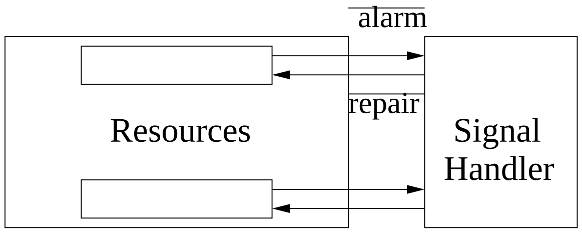

Figure 6.1: Resource and signal handler system

In order to illustrate the above decomposition scheme, we consider a system consisting of a signal handler and resources(Fig6.1). The resources fail with certain probability(2/3) and they generate an alarm($\overline{alarm}$). The signal handler on receiving the alarm can either buffer it and wait for more alarms or buffer it and switch to serving the alarms. Since the alarms are generated pretty frequently we would like the signal handler to buffer and serve more frequently rather than buffer and wait for more alarms. We model a 2 resource scenario, where process P_0 denotes both the resources working and $P_0 \models (Good)$, where $(Good)$ is a atomic proposition. P_1 and P_2 denote failure of 1 and 2 resources respectively. P_1 one resource is failed and the other is working, so it can either generate another alarm and go to P_2, or it can remain in P_1. P_1 and P_2 on receiving the *repair* signal go back to P_0. In terms of process algebra the system is described as

$$\begin{aligned}
P_0 &= \langle\overline{alarm}\rangle.P_0 \oplus_{1/9} (\langle\overline{alarm}\rangle.P_1 \oplus_{1/2} \langle\overline{alarm}\rangle.P_2) \\
P_1 &= (\langle\overline{alarm}\rangle.P_1 \oplus_{2/3} \langle\overline{alarm}\rangle.P_2) \oplus \langle repair\rangle.P_0 \\
P_2 &= \langle repair\rangle.P_0
\end{aligned}$$

We want the composite system of signal handler and resources to satisfy the property that an *alarm* signal generated by the resources should be received by the signal handler(SH) with high probability and following it the *repair* signal should be generated by signal handler and received by resources with high probability. The composite system should revert to the state which models the proposition (*Good*). The property can be specified in extended PML as:

$F = [\langle\overline{alarm}, alarm\rangle_{[x,y]}(F_1)$ where $\phi(x,y) = (y > \lambda)]$ where λ is a constant in $[0,1]$ and
$F_1 = [\langle\overline{repair}, repair\rangle_{[x',y']}(Good)$ where $\Omega(x',y') = (x' = 1.0 \wedge y' = 1.0))]$

We derive a specification for SH by using our results on compositional verification: The composite system $P_0 \times SH \models F$ iff $SH \models \mathcal{W}_{P_0\times}(F)$.

1. $\mathcal{W}_{P_0\times}(F)$
$= \langle alarm\rangle_{[x_0,y_0]}\mathcal{W}_{P_0\times}(F_1) \wedge \langle alarm\rangle_{[x_1,y_1]}\mathcal{W}_{P_1\times}(F_1) \wedge \langle alarm\rangle_{[x_2,y_2]}\mathcal{W}_{P_2\times}(F_1)$
where $\phi'(x_0,y_0,x_1,y_1,x_2,y_2) = \phi(max\{4/9*y_1, 4/9*y_2, 1/9*y_0\}/y) = max\{4/9*y_1, 4/9*y_2, 1/9*y_0\} > \lambda)$, since $P_0 \xrightarrow[{[4/9,4/9]}]{\overline{alarm}} P_1$, $P_0 \xrightarrow[{[4/9,4/9]}]{\overline{alarm}} P_2$, and $P_0 \xrightarrow[{[1/9,1/9]}]{\overline{alarm}} P_0$

2. $\mathcal{W}_{P_0\times}(F_1) = \texttt{ff}\ P_0 \not\xrightarrow{repair}$

3. $\mathcal{W}_{P_1\times}(F_1)$
$= \langle\overline{repair}\rangle_{[x_1',y_1']}\mathcal{W}_{P_0\times}((Good))$ where $\Omega'(x_1',y_1') = \Omega(min\{1.0*x_1'\}/x', max\{1.0*y_1'\}/y') = (x_1' = 1.0 \wedge y_1' = 1.0)$, since $P_1 \xrightarrow[{[1.0,1.0]}]{repair} P_0$
$= \langle\overline{repair}\rangle_{[1,1]}\mathcal{W}_{P_0\times}((Good))$

4. $\mathcal{W}_{P_2\times}(F_1)$
$= \langle\overline{repair}\rangle_{[x_1',y_1']}\mathcal{W}_{P_0\times}((Good))$ where $\Omega'(x_1',y_1') = \Omega(min\{1.0*x_1'\}/x', max\{1.0*y_1'\}/y') = (x_1' = 1.0 \wedge y_1' = 1.0)$, since $P_2 \xrightarrow[{[1.0,1.0]}]{repair} P_0$
$= \langle\overline{repair}\rangle_{[1,1]}\mathcal{W}_{P_0\times}((Good))$

5. $\mathcal{W}_{P_0\times}((Good)) = \texttt{tt}$, since $P_0 \models (Good) \Rightarrow P_0 \times X \models (Good)$ where $X \models \texttt{tt}$

6. From 1 to 5, we get
$SH \models \langle alarm\rangle_{[0,0]}\texttt{ff} \wedge \langle alarm\rangle_{[x_1,y_1]}\langle\overline{repair}\rangle_{[1,1]}\texttt{tt} \wedge \langle alarm\rangle_{[x_2,y_2]}\langle\overline{repair}\rangle_{[1,1]}\texttt{tt}$ where $max\{4/9*$

$$y_1, 4/9 * y_2\} > \lambda \Leftrightarrow SH \models \langle alarm\rangle_{[x,y]}\langle\overline{repair}\rangle_{[1,1]}\texttt{tt} \text{ where } 4/9 * y > \lambda$$

The specification generated for SH provides constraints for the design of the system. The SH should have two states: the first state should wait for *alarm* signals and then make a transition to the repair state. In the repair state it should always generate the $\overline{repair}$. Suppose that we design the SH such that it goes to the repair state every time it receives an *alarm* signal, i.e. the probability $x = y = 1.0$, then the maximum probability of the composite system handling the alarms successfully is 4/9. The maximum probability for the composite system to successfully handle the alarms is bounded by 4/9. The bound was generated from the known components of the system, that is the *resources* and the nature of composition operator, $\times$. Thus the results on compositional verification provide an important tool to generate the specification for unknown systems and analyze the limitations imposed by the nature of interfacing and behavior of known components.

Chapter 7

Conclusion and Future Work

PNS is an expressive model which incorporates the non-determinism along with probabilistic choices. We have used PNS to model the non-determinism in randomized systems and to reason about composite systems with non-determinism arising due to concurrency. Randomization is usually used to break symmetry, as in the token stabilization protocol, or to make decisions in the absence of any guided choice, for example selection of pivot element in quick sort. The advantage of using a randomization based approach is that it achieves high performance because it doesn't invest resources to inspect the consequences of choices it makes, However, random resolution of choices make it difficult to estimate the lower bounds on resources needed to achieve the goal. For instance, for the randomized token stabilization protocol, it is important to know the probability of stabilization in a given amount of time (number of steps). The model checking algorithm, *modchk-fuzzy*, provides a generic method for quantitative analysis of reachability properties using randomized schedulers.

The results on *sufficiently weak* decomposition of the original specification for a composite system enables for flexible and independent design of subcomponents. Usually, some parts of the composite system are known and the specification for the entire system is available and we need a method to generate the sub-specifications for the unknown system. When the components work concurrently then the problem is two fold, firstly we have to generate constraints for the behavior of the scheduler and secondly we have generate the sub-specification for the unknown system under the scheduler constraints. The extended PML specifications suit very well for the decomposition because they incorporate the scheduler constraints as interval bounds on each transition. Thus the use of extended PML allows us to generate the sub-specification recursively at each transition of the composite system.

We used token based randomized stabilization protocol to see the efficiency of modchk fuzzy algorithm. Leaving apart the inherent exponential blow up of the system, the method is competitive with respect to resources and time needed to perform the quantitative analysis. Moreover the algorithm allows to utilize the symmetry of the system, which greatly reduces the state space. Token rings are very important for several distributed protocols, and randomized stabilization protocol is necessary to recover from illegal states (like more than one token). Therefore it would be interesting to explore the generalization of symmetry based abstraction approach to see if it reduces the state space for n process systems. Firstly, the generalization requires the number of different configurations of the token ring when m out of n processes are active. Two configurations are same if one can be obtained from another by rotation (relabeling) of processes. This will give an estimate of the reduction in state space. Secondly, there is need for methods to automate the process of abstraction and generation of probabilistically bi-similar models of RSSP from the original model. Another approach to quantitative analysis would be to exploit the trend in the minimum and maximum probabilities of stabilization to come up with approximate functions of such measures with respect to n and x, the number of transitions.

To summarize the future work we would like to explore and generalize the results based on symmetry for such systems as RSSP. We would like to develop a set of benchmark problems with nondeterminism where modalities are used to specify the independence of transitions. Lastly theres need to expand the work to symbolic techniques as explicit methods are handicapped when it comes to bigger systems from real life scenarios.

Bibliography

[1] Adnan Aziz, Vigyan Singhal, and Felice Balarin. It usually works: The temporal logic of stochastic systems. In *Proceedings of the 7th International Conference on Computer Aided Verification*, pages 155–165. Springer-Verlag, 1995.

[2] Christel Baier and Marta Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[3] B.Jonsson, K. Larsen, and W. Yi. *Handbook of Process Algebra*, chapter Probabilistic Extension of Process Algebras. Elsevier Science, North-Holland, 2001.

[4] J. R. Burch, E. M. Clarke, and K. L. McMillan. Symbolic model checking: 10^{20} states and beyond. In *Proc. of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, PA, 1990.

[5] Edmund M. Clark, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000.

[6] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Proc. FOCS'88*, pages 338–345, 1988.

[7] Michael Huth and Marta Kwiatkowska. Quantitative analysis and model checking. In *Proceedings of the 12th Symposium on Logic in Computer Science (LICS '97)*, page 111. IEEE Computer Society, 1997.

[8] Amos Israeli and Marc Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 119–131. ACM Press, 1990.

[9] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(1):333–354, 1983.

[10] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 2004. To appear.

[11] Marta Z Kwiatkowska, Gethin Norman, David A Parker, and Roberto Segala. Symbolic model checking of concurrent probabilistic systems using MTBDDs and simplex. Technical Report CSR-99-1, University of Birmingham, 1999.

[12] Kim Guldstrand Larsen and Arne Skou. Compositional verification of probabilistic processes. In *Proceedings of the Third International Conference on Concurrency Theory*, pages 456–471. Springer-Verlag, 1992.

[13] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[14] M. Narasimha, R. Cleaveland, and P. Iyer. Probabilistic temporal logics via the modal mu-calculus. In W. Thomas, editor, *Foundations of Software Science and Computation Structures, Lecture Notes in Computer Science*, volume 1578, pages 288–305. Springer-Verlag, March 1999.

[15] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *IEEE Symposium on Foundations of Computer Science*, pages 327–338, 1985.

Appendix A

Input Language for PNS

```
MODULE

s1 : [0..1];

s2 : [0..1];

s3 : [0..1];

s4 : [0..1];

ACTION a;

[] s4=1 ->[a] 0.5 : (s3=1) + 0.5 : (s4=1);

[] s4=1 ->[a] 1.0 : (s3=0) ;

[] s3=1 ->[a] 0.5 : (s2=1) + 0.5 : (s3=0);

[] s3=1 ->[a] 1.0 : (s2=0) ;

[] s2=1 ->[a] 0.5 : (s2=0) + 0.5 : (s1=1);

[] s2=0 ->[a] 0.5 : (s2=0) + 0.5 : (s2=1);

ENDMODULE
```

```
COMPUTE(s4=1, Mu{x}(s1=1 | <a>x));
```