Quantitative Analysis of Distributed Randomized Protocols

Christel Baier Frank Ciesinski Marcus Groesser

Institut für Informatik I
Universität Bonn
Römerstrasse 164
53117 Bonn
Germany
{baierlciesinsklgroesser}@cs.uni-bonn.de

ABSTRACT

A wide range of coordination protocols for distributed systems, internet protocols or systems with unreliable components can formally be modelled by Markov decision processes (MDP). MDPs can be viewed as a variant of state-transition diagrams with discrete probabilities and nondeterminism. While traditional model checking techniques for non-probabilistic systems aim to establish properties stating that all (or some) computations fulfill a certain condition, the verification problem for randomized systems requires reasoning about the quantitative behavior by means of properties that refer to the probabilities for certain computations, for instance, the probability to find a leader within 5 rounds or the probability for not reaching an error state.

The paper starts with a brief introduction into modelling randomized systems with MDPs and the modelling language Probmela which is a guarded command language with features of imperative languages, nondeterminism, parallelism, a probabilistic choice operator and lossy channels. We summarize the main steps for a quantitative analysis of MDPs against linear temporal logical specifications. The last part will report on the main features of the partial order reduction approach for MDPs and its implementation in the model checker LiQuor.

Categories & Subject Descriptors

Software **[Software Engineering]**: Software/Program Verification, Model Checking, Reliability, Formal methods,
Mathematics of Computing **[Probability and Statistics]**: Markov processes, Stochastic processes

FMICS'05, September 5–6, 2005, Lisbon, Portugal.

General Terms

Reliability, Languages, Theory, Verification

1. INTRODUCTION

In the past, several methods for analyzing quantitative aspects of probabilistic systems have been suggested. These range from proof techniques to model checking algorithms for formal models with discrete or continuous probabilities, time-abstract, discrete-time or continuous-time models, and fully probabilistic models or models where nondeterminism and probabilism coexists, see e.g. [5] for an overview.

We concentrate here on techniques for Markov decision processes (MDP), an operational model for systems with discrete probabilism and nondeterminism. For instance, MDPs occur as a natural formal model for coordination algorithms for distributed systems that use coin tossing for breaking symmetry or for communication protocols that operate with unreliable channels (see e.g. [21] for several randomized mutual exclusion, leader election or data link protocols).

To specify the behavior of probabilistic concurrent systems several process algebras, automata-models and other specification languages have been suggested. We follow here the guarded command language approach [14, 22, 23] and present the main ideas of the modelling language Probmela [1], which resembles SPIN's input language Promela [17] and provides a simple and intuitive way to specify the stepwise behavior of protocols that may behave probabilistically, e.g., since they use coin tossing as algorithmic concept or since they have exceptional faulty behaviors that occur with some known probabilities. Probmela-processes are described by standard constructs of imperative languages (assignments, conditional commands, loops) and nondeterministic and probabilistic choices. They may communicate over shared variables or through message passing along perfect or lossy channels. Probmela has a formal semantics that describes the stepwise behavior of the parallel composition of several processes by a MDP. The latter can be constructed with an on-the-fly technique and serves as basis for verification purposes.

To specify quantitative properties such as "the deadlock probability is smaller than 0.015" or "the chance to select a leader within the next 5 rounds is at least 95%" several temporal logics have been suggested in the literature. Corresponding model checking algorithms mainly rely on a combination of graph algorithms and

automata-based techniques as they are known for non-probabilistic systems combined with techniques for solving linear programs [9, 7, 10, 12, 6]. Thus, verifying probabilistic systems against quantitative properties is even harder than the purely qualitative verification problem for non-probabilistic system. In particular, the state space explosion problem is even more critical for MDPs than in the non-probabilistic case. To handle the state space explosion problem several methods that have been developed for non-probabilistic systems were adapted to Markovian models. Most popular is the symbolic approach with multi-terminal binary decision diagrams [2] which has been implemented in the model checker Prism and successfully applied to many types of probabilistic systems [20]. The MDP model checker Rapture [19] uses an iterative abstraction-refinement algorithm which groups states into blocks and iteratively refines the blocks with a bisimulation-like partitioning technique. These approachs focus on the verification of branching time properties. Recently, it has been shown that also the partial order reduction approach can be adapted to MDPs and linear [4, 11] or branching [3] time properties. The goal is reducing the operational model to be analyzed by ignoring certain transitions (and states) that are redundant due to the interleaving semantics of distributed systems. Partial order reduction yields the basis of the forthcoming model checker LiQuor that takes as input a Probmela-program P and a linear temporal logical formula φ and calculates the probability that φ holds for P by solving a linear program for the MDP of a reduced model.

Organisation of the paper. In this paper, we summarize the main concepts of the forthcoming model checker LiQuor that implements the partial order reduction technique in the context of a quantitative analysis of randomized protocols against ω-regular linear time specifications. Section 2 provides with a brief introduction to modelling randomized protocols by the specification language Probmela which yields an operational semantics by means of an MDP. The main steps to compute the (extremal) probabilities for linear time properties are summarized in Section 3. Section 4 explains how these techniques can be supported by the partial order reduction approach and how they are realized in the model checker LiQuor.

2. THE MODELLING LANGUAGE PROBMELA

A probabilistic variant of SPIN's input language Promela [17], called Probmela, has been introduced in [1] as description language for probabilistic concurrent programs. The idea is to describe the processes that run in parallel by a guarded command language, similar to the high-level description languages pGCL [22, 23] and MoDeST [8]. A Probmela-program $P = P_1 \| \ldots \| P_n$ consists of finitely many processes P_i that run in parallel and might communicate over shared variables or (synchronous/fifo) channels.

$$
\begin{array}{rcl}
P & ::= & \mathsf{skip} \mid x := \mathsf{expr} \mid x := \mathsf{random}(V) \mid P_1 ; P_2 \mid \\
& & \underline{\mathbf{IF}} \;\; :: \; \mathsf{bexpr}_1 \Rightarrow P_1 \ldots \; :: \; \mathsf{bexpr}_n \Rightarrow P_n \;\; \underline{\mathbf{FI}} \mid \\
& & \underline{\mathbf{DO}} \;\; :: \; \mathsf{bexpr}_1 \Rightarrow P_1 \ldots \; :: \; \mathsf{bexpr}_n \Rightarrow P_n \;\; \underline{\mathbf{OD}} \mid \\
& & \underline{\mathbf{PIF}} \; [\pi_1] \; \Rightarrow P_1 \ldots [\pi_n] \; \Rightarrow P_n \; \underline{\mathbf{FIP}}
\end{array}
$$

The main ingredients to specify the behavior of the processes that run in parallel are deterministic and randomized assignments, sequential composition, conditional commands (IF-FI), loops (DO-OD) and a probabilistic choice operator (PIF-FIP).

The abstract syntax of this basic language is shown above where x is a program variable (e.g., type boolean or integer), expr an expression of the same type as x, bexpr_i boolean expressions (called *guard*) and π_i "probabilities", i.e., values in the interval $[0,1]$ such that $\pi_1 + \ldots + \pi_n = 1$. Probmela also allows for the special guard "else" with the intuitive meaning that no other guard is satisfed. In addition, Probmela has several other features like atomic regions, process instantiation or substochastic distributions that are not explained here. Message passing over channels will be explained later.

The atomic command skip means an atomic step which does not change the values of the variables or contents of the channels. The meaning of deterministic assignment is obvious. For the randomized assignment $x := \mathsf{random}(V)$ we assume that V is a finite subset of the domain of variable x and that a value $v \in V$ is assigned to x according to a uniform distribution.

The intuitive behavior of $P = \underline{\mathbf{PIF}} \; [\pi_1] \; \Rightarrow P_1 \ldots [\pi_n] \; \Rightarrow P_n \; \underline{\mathbf{FIP}}$ is that first P resolves a probabilistic choice by which P_i is selected with probability π_i. The meaning of conditional commands and DO-OD-loops is roughly as in Promela.

The terms $\mathsf{bexpr}_i \Rightarrow P_i$ are called guarded commands with their usual meaning: if the boolean expression bexpr_i is true then subprocess P_i can be executed. More precisely, in $P = \underline{\mathbf{IF}} \;\; :: \;\; \mathsf{bexpr}_1 \Rightarrow P_1 \;\; \ldots \;\; :: \;\; \mathsf{bexpr}_n \Rightarrow P_n \; \underline{\mathbf{FI}}$ there is a nondeterministic choice between the processes P_i where the corresponding guard bexpr_i holds for the current variable evaluation. If none of the guards bexpr_i holds then process P "blocks" which essentially means that P waits until another process changes the values of the program variables such that one of the guards bexpr_i evaluates to true.

The meaning of $P = \underline{\mathbf{DO}} \;\; :: \;\; \mathsf{bexpr}_1 \Rightarrow P_1 \;\; \ldots \;\; :: \;\; \mathsf{bexpr}_n \Rightarrow P_n \; \underline{\mathbf{OD}}$ is similar, the difference being that the nondeterministic choice between the processes P_i with valid guards continues until none of the guards holds, in which case P terminates.

The guards of a $\underline{\mathbf{IF}} - \underline{\mathbf{FI}}$ or $\underline{\mathbf{DO}} - \underline{\mathbf{OD}}$ statement need not to be disjoint. The nondeterminism between the enabled guarded commands can be useful, for instance, for underspecification (where alternatives for the allowed behaviors are specified) or for modelling the interface with an unpredictable environments (e.g., other programs or human users).

Message passing via channels is provided in Probmela in form of atomic communication actions $c!\mathsf{expr}$ (sending the current value of expr along c) and $c?x$ (receiving a value for variable x). The classical (synchronous) handshaking of two processes is obtained by declaring c as a synchronous channel, while for asynchronous message passing c has be declared as a FIFO channel with fixed (and finite) capacity. In addition, FIFO channels can be perfect or unreliable. In the latter case, the user might specify the probability for losing a message while inserting into the buffer. Other variants of faulty channels, e.g., FIFO channels that might lose or corrupt buffered messages, could be handled in a similar way. The terms $c!\mathsf{expr}$ and $c?x$ can also be used in the guards in which case they are treated as boolean expressions with the meaning that a communication guard fails if there is no communication partner in the case of synchronous communication or if the corresponding write- or read-operation for a FIFO channel c is not enabled, since the buffer is full or empty, respectively.

For an example, Fig.1 shows the Probmela-code for the randomized leader election protocol a la [18]. The goal is to choose a leader among n processes $P_1, \ldots, P_n$ that are arranged in a ring by a sequence of random decisions that are made by each participating process independently. Each process owns a channel out of an array c of n channels, and has a process id in $\{1, \ldots, n\}$. We refer to the process id and the predeccor by *MY_PID* and *MY_PRED*. In addition, any process uses the local boolean variables *tmp* and *is_active*. Furthermore, we use a shared variable *noa* which serves

```
1   DO   ::  is_active ⇒
2                PIF   [0.5]   ⇒ c[MY_PID]!0; my_choice := 0
3                      [0.5]   ⇒ c[MY_PID]!1; my_choice := 1
4                FIP
5                IF    ::      c[MY_PRED]?1 ∧ my_choice = 0
6                                ⇒ noa := noa − 1; is_active := false
7                      ::      else ⇒ skip
8                FI
9        ::  ¬is_active ⇒ c[MY_PRED]?tmp; c[MY_PID]!tmp
10  OD
```

Figure 1: Randomized leader election a la [18]

as counter for the active processes which means the number of processes that are still candidates for the leadership. The initial value of *noa* is N. At the beginning each process is in state active (line 1), picks a value *my_choice* $\in \{0,1\}$ randomly and sends it over its channel, stated by the output command $c[\ldots]!$ (lines 2 and 3). The successor reads this choice and goes to passive mode (where it only passes incoming messages, lines 5 and 9) when its value was 0 and its predecessor's value was 1, in which case variable *noa* is decremented. Eventually *noa* becomes 1 which means that a leader has been elected.

Operational semantics.. The stepwise interleaving behavior of a Probmela program can be formalized by a Markov decision process (MDP). We skip the formal semantics by means of SOS-rules for the processes and programs here, which (together with other examples such as the IPv4zeroconf protocol) can be found in [1]. Instead we provide the definition of a MDP and sketch the ideas for the operational semantics.

A *Markov decision process* is a tuple $M = (S, \mathsf{Act}, \mathbf{P}, s_0, \mathsf{AP}, \mathsf{label})$ where S is a finite set of states, Act a finite set of actions, $\mathbf{P} : S \times \mathsf{Act} \times S \to [0,1]$ is the three-dimensional transition matrix which specifies the enabled actions in state s and their probabilistic effect, $s_0 \in S$ the initial state, AP a set of atomic propositions and $\mathsf{label} : S \to 2^{\mathsf{AP}}$ the labeling function that assigns to any state s the set of atomic propositions that are assumed to hold in s. We require that for all states s and actions a, $\sum_{t \in S} \mathbf{P}(s,a,t) \in \{0,1\}$.

$\mathsf{Act}(s) = \{a \in \mathsf{Act} : \exists t \in S.\mathbf{P}(s,a,t) > 0\}$ denotes the set of enabled actions in state s. Intuitively, if the current state is s then there is a nondeterministic choice between the enabled actions in s. If action $a \in \mathsf{Act}(s)$ is selected then $\mathbf{P}(s,a,t)$ denotes the probability for reaching state t.

Given a Probmela-program $P = P_1 \| \ldots \| P_n$ then the states in the associated MDP have the form $s = \langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle$ where ℓ_i is the current location of process P_i (value for the control variable for process P_i), η is an evaluation for the variables (i.e., a function that assigns values to the variables) and ξ an evaluation for the FIFO channels (i.e., a function that assigns to any FIFO channel c the word describing c's current content).

For the initial state, we require that the ℓ_i's are the starting locations for the processes, all FIFO channels are empty and the variables have certain default values. The transition relation describes the one-step behavior of P which could be an individual step of one process P_i with no synchronous communication action or a handshaking of two processes P_i and P_j via complementary synchronous communication actions. Non-trivial distributions with two or more successor states can be obtained if P_i performs a probabilistic choice (PIF-FIP), a randomized assignment or a write-operation into a lossy FIFO channel.

The action names for the transitions are only needed for technical reasons. The atomic propositions that serve as labels for the states can be any assertion about the locations of the processes, the values of the program variables or the contents of the channels. For instance, there might be atomic propositions stating that "the current location of P_3 is line 167 of P_3's Probmela-code" or "$x < y - 5$" or "the buffer of channel c is empty".

3. VERIFYING QUANTITATIVE PROPERTIES

We now turn to the question how to reason about the probabilitistic behavior of a randomized protocol modelled in Probmela or some other formalism with an operational MDP-semantics. As it is standard for the verification of distributed systems, the most popular approach performs a "worst-case" analysis that ranges over *all* potential resolutions of the nondeterminism, i.e., the selection of one of the enabled actions in the current state. If none of the processes of a Probmela-program contains nondeterministic choices then the selection of one action per state means chosing the process which performs the next step (two processes in case of a synchronous handshaking).

For MDPs the standard notion of a probability measure assumes a fixed instance, often called scheduler, policy, strategy or adversary, that resolves the nondeterministic choices by chosing one enabled action for the current state s and the system history (formalized by a finite path from the initial state to s). Given a scheduler D, M's behavior under D can be described by a Markov chain on which we may use the standard probability measure on its maximal paths. Here, a maximal path denotes an alternating sequence of consecutive transitions $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ such that $s_i \in S$, $a_i \in \mathsf{Act}(s_i)$ and $\mathbf{P}(s_i, a_i, s_{i+1}) > 0$ which is either infinite or ends in a terminal state, i.e., a state s_i with $\mathsf{Act}(s_i) = \emptyset$. Assuming a formalism like linear temporal logic (LTL) that specifies path-properties, we then may speak about the maximal or minimal probability for a path-property to hold for an MDP.

Let us briefly summarize the main concepts of LTL, its semantics over MDPs and the model checking algorithm. LTL formulas are built from the grammar

$$\varphi ::= \mathsf{true} \mid p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \phi_1 U \phi_2$$

where $p \in \mathsf{AP}$ is an atomic proposition, $\wedge$ and $\neg$ denote standard conjunction and negation, while $\bigcirc$ denotes the next step operator and U the until operator. Given a maximal path σ in a MDP, the satisfaction relation $\sigma \models \varphi$ is defined as in the non-probabilistic case (see e.g. [10]). For instance, $\bigcirc p$ states that p holds in the second state, while pUq holds for any path that reaches a q-states via finitely many (possibly zero) p-states. The derived operators $\Diamond$ (eventually) and $\Box$ (always) are obtained as usual through $\Diamond\varphi = \mathsf{true} U \varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$. The combination of them yields e.g. the property $\Box(\mathsf{send} \rightarrow \Diamond \mathsf{received})$ stating that any sent message will eventually be received or $\Box\Diamond$"process P_1 is a leader" stating that process P_1 will be elected infinitely often as the leader. The property $\Diamond$"P_1 is a leader" is used in the leader-election example to denote that eventually process P_1 will be elected.

Given a scheduler D for a MDP M and a LTL formula φ, the probability $\Pr^D(\varphi)$ denotes the probability for the maximal paths starting in the initial state and satisfying φ. Depending on whether φ describes the "bad" or "good" behaviors, a quantitative analysis with respect to formula φ amounts calculating the extremal probability $\sup_D \Pr^D(\varphi)$ or $\inf_D \Pr^D(\varphi)$ where D ranges over all schedulers. For instance, for a leader protocol one might aim at a high (worst-case) probability $1 - \varepsilon$ that a leader is elected within the first 10

rounds for all schedulers, which amounts showing that

$$\inf_D \Pr^D\big(\Diamond(\texttt{noa} = 1 \wedge \#\texttt{rounds} \leq 10)\big) \;\geq\; 1-\varepsilon.$$

where "$\texttt{noa} = 1$" stands for the property "a leader has been elected", while #rounds is a counter for the number of attempts to find a leader (not shown in Fig.1). Equivalent to this is the requirement that the maximal probability for the event given by the formula

$$\Box\big(\#\texttt{rounds} \leq 10 \rightarrow \texttt{noa} \neq 1\big)$$

is smaller or equal ε. In fact, due to the fact that

$$\inf_D \Pr^D(\varphi) = 1 - \sup_D \Pr^D(\neg\varphi)$$

it suffices to provide an algorithm for calculating maximal probabilities for LTL formulas under all schedulers. The idea for this relies on an automata-based approach [28, 29, 10, 12, 6]. Using standard techniques, one first constructs a deterministic ω-automata A_φ for the given LTL formula φ. Then, one generates the (reachable fragment of the) product $M \times A_\varphi$ which is again an MDP. The states in $M \times A_\varphi$ have the form $\langle s,z\rangle$ where s is a state in M and z an automata-state. The paths in $M \times A_\varphi$ encode the pairs $\langle \sigma,\zeta\rangle$ consisting of a path σ in M and their runs ζ in A_φ. Then, one can show that the maximal probability for φ in M agrees with the maximal probability for the paths in $M \times A_\varphi$ where the acceptance condition of A_φ holds. The latter depends on the chosen type of deterministic ω-automaton. For instance, for Streett automaton the acceptance condition is a strong fairness condition which can be treated in a rather simple way: by analysing the strongly connected components, the Streett acceptance condition allows to derive a set C of states in the product such that

$$\sup_D \Pr^D_M(\varphi) \;=\; \sup_D \Pr^D_{M\times A_\varphi}(\Diamond C).$$

That is, the automata-approach provides a reduction of LTL-model checking problem for MDPs to the problem of computing maximal reachability probabilities in MDPs. The latter is solvable by means of a linear program [9, 7].

To give an idea for the linear programming approach, assume that M is a MDP with state space S and we ask for the maximal probability to reach a set $C \subseteq S$. The idea is now to use variables x_s for all states $s \in S$ for the maximal probability to reach C from s. We first may perform a graph analysis for computing the sets S^0 of all states that cannot reach C and the set S^1 of states that reach C almost surely under some scheduler. Then, $x_s = 0$ if $s \in S^0$ and $x_s = 1$ if $s \in S^1$. For the remaining states, we use the inequality

$$x_s \;\geq\; \sum_{t\in S} \mathbf{P}(s,a,t)\cdot x_t$$

for all actions $a \in \mathsf{Act}(s)$. This system of inequalities has a unique minimal solution with $x_s \in [0,1]$ and this minimal solution agrees with the maximal reachability probabilities.

4. THE MODEL CHECKER LIQUOR

The modeling language Probmela described in section 2 yields the input language of the tool LiQuor,[1] which supports the analysis of probabilistic reactive systems by means of model checking linear-time properties in a quantitative setting. More precisely, the functionality of LiQuor is to take a Probmela-program $P = P_1 \| \ldots \| P_n$ and a linear temporal logic (LTL) formula φ as input and to calculate the maximal or minimal probability that φ holds for P.

[1] **Li**near Time **Qu**alitative and **Qu**antitative Analysis **o**f **r**eactive Systems

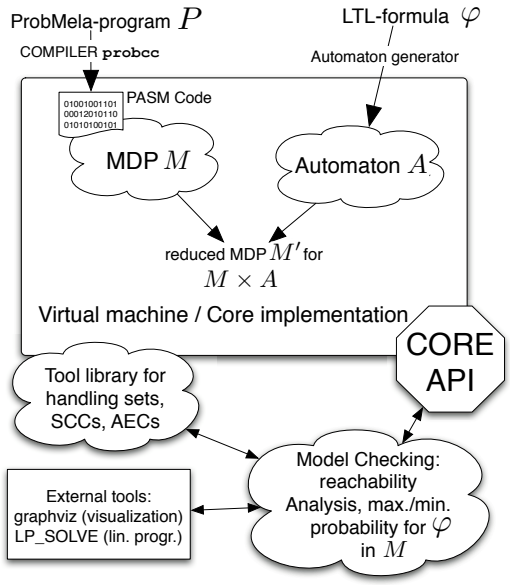

Figure 2: Schema for LiQuor

A system description in Probmela is translated into an intermediate XML-like format, called PASM (Probabilistic Assembler like Language), for a virtual machine that serves for generating the MDP-semantics. PASM is an easily extendable low-level guard-oriented system description language and moreover contains elements to control the model checking process. PASM serves as the target language for the Probmela compiler that was implemeneted by constructing an ANTLR [24] grammar out of Probmela's syntax and semantics.

To tackle the state explosition problem, LiQuor generates the MDP for P with an on-the-fly technique, similar to those realized in the popular non-probabilistic model checker SPIN, and using *partial order reduction* criteria. The idea of the partial order reduction approach [25, 15, 27] is to analyse only a fragment of the state space, rather than the full system model, using the redundancies in interleaving-based representations of asynchronous systems that origin from the commutativity of independent actions executed by processes that run in parallel. Independent actions are those that can be executed in any order without affecting each other's enabledness and with the same effect as they were executed in parallel. Typically, this is the case for "local, internal" actions of different processes, i.e., actions that do not refer to shared variables and do not depend on the communication possibilities.

In [4, 11], Peled's partial order reduction approach with so-called ample-sets, has been adapted to the probabilistic setting for verifying linear time properties. For this, one aims at a sub-MDP M' of M, which means a MDP where the state space is a subset of M's state space and where the enabled actions of a state s' in M', called the ample-set of s, are contained in $\mathsf{Act}(s)$, and which can be generated with an on-the-fly technique, without generating the full MDP M. The underlying notion for the independence of two actions α and β requires that for any state s with $\{\alpha,\beta\} \subseteq \mathsf{Act}(s)$ we have (i) β is enabled in any α-successor of s, and vice versa, and (ii) the probabilistic effect of the action sequences $\alpha\beta$ and $\beta\alpha$ agree, that is,

$$\sum_{t\in S} \mathbf{P}(s,\alpha,t)\cdot\mathbf{P}(t,\beta,w) \;=\; \sum_{u\in S} \mathbf{P}(s,\beta,u)\cdot\mathbf{P}(u,\alpha,w)$$

A1 If $\mathsf{ample}(s) \neq \mathsf{Act}(s)$ then all actions $\alpha \in \mathsf{ample}(s)$ are stutter actions.

A2 If $s \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_n \xrightarrow{\beta} \ldots$ is a path in M where β is dependent on some action in $\mathsf{ample}(s)$ then $\alpha_i \in \mathsf{ample}(s)$ for some $i \in \{1,\ldots,n\}$.

A3 For each end component (T,A) in M' there is a state $t \in T$ with $\mathsf{ample}(t) = \mathsf{Act}(t)$.

A4 If $s \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_n \xrightarrow{\gamma} \ldots$ is a path in M where

$$\alpha_1,\ldots,\alpha_n,\gamma \notin \mathsf{ample}(s)$$

and γ is probabilistic then $|\mathsf{ample}(s)| = 1$.

Figure 3: Conditions for the ample-sets

H1 = **A1**

H2 Consider as ample set candidates only $\mathsf{ample}(s) = \mathsf{Act}(P_i, s)$ all enabled actions of a process P_i in state s, and assure that:

If $\mathsf{ample}(s) \neq \mathsf{Act}(s)$ then each action $\alpha \in \mathsf{ample}(s)$ is independent of all other processes $P_j, j \neq i$ (future) actions.

H3 Let $\ldots \xrightarrow{\alpha_{n-1}} s_{n-1} \xrightarrow{\alpha_n} s_n \xrightarrow{\alpha_{n+1}} s_{n+1} \ldots \xrightarrow{\alpha_j} s_j$ be the DFS-stack in the construction of M', $\mathsf{ample}(s_n) = \mathsf{Act}(s_n)$ and

$$\nexists n' : n < n' < j : \mathsf{ample}(s_{n'}) = \mathsf{Act}(s_{n'}).$$

If a backward edge is discovered from s_j to a state s_i where $n < i < j$, then put $\mathsf{ample}(s_j) = \mathsf{Act}(s_j)$.

H4 Assure that $\mathsf{ample}(s) = \mathsf{Act}(s)$ or $|\mathsf{ample}(s)| = 1$.

Figure 4: Heuristics for the ample-sets

for all states w.

To ensure that M and M' have the same extremal probabilities for LTL formulas, the chosen ample-sets have to fulfill conditions **A1** -**A4** in Fig.3. (In addition, we have to require that the ample-set is non-empty for any non-terminal state.) Conditions **A1**, **A2** and **A3** are exactly as in the non-probabilistic case. **A1** requires that for any state s in M' that is not fully expanded, all ample actions are stutter actions, which means that they do not change the state-labels. The most difficult condition is **A2**. Speaking roughly, condition **A2** ensures that any path σ in the full MDP M can be transformed into a path σ' in M' by switching the order of independent actions. To ensure that through infinitely many permutations of independent actions the orginal path σ and the new path σ' fulfill the same LTL formulas, condition **A3** is needed. The concept of end components has been introduced in [13]. They can be understood as the MDP-counterpart of strongly connected subgraphs or ergodic sets of Markov chains. They are defined as strongly connected sub-MDPs N where certain fair schedulers can ensure that once N is entered, N is never left and almost surely all states of N are visited infinitely often.

While **A1**-**A3** are as in the non-probabilistic case, the last condition **A4** is special for the probabilistic approach. It ensures that the above path-transformation can be applied "simultaneously" to the paths generated by a given scheduler D for M, thus yielding a scheduler D' for M' such that the probabilities for all LTL-formulas under D and D' agree.

In order to apply the ample set method in practice the conditions in Fig.3 are checked during DFS-based model construction. While condition **A1** is local and easy to check conditions **A2**, **A3** and **A4** are not as easy to handle. For this reasons, **A2** - **A4** are replaced with stronger and computationally simpler conditions. See Fig. 3. Peled proved (see [25]) that deciding **A2** is of the same computational complexity as deciding the reachability problem. For this reason, **A2** is replaced with the stronger condition **H2** which is easy to check with the independence relation. The independence relation itself is established by analysing for each action-pair (α,β) the occurrence of variables that are written in α and read in β (or vice versa). In this case the actions are considered as dependent.[2] Similarly, **A3** is replaced by the simpler sufficient condition **H3** which looks for DFS-backward edges and can simply be integrated in the on-the-fly construction of the reduced MDP M'. The correctness of **H3** follows by the fact that any end component contains a cycle. To guarantee that **A4** holds we switch to condition **H4** which is stronger but easy to implement.

Experimental results. The leader election protocol can be considered as representative for a setting where asynchronous (distributed) processes communicate information under unreliable conditions in order to establish a certain goal. In Fig.5 results for the leader election protocol with a various number of processes are presented. The columns "lp-size" show the number of inequalities in the linear program that has to be solved to calculate the measure of all paths that satisfy the formula under consideration.

(unreduced model)

procs	states	transitions	lp-size
2	414	892	427
3	10584	32611	14784
4	214219	865928	499121
5	4386319	22159973	16520905
6	89042275	552863314	not calculated

(reduced with partial order red.)

procs	states	transitions	lp-size
2	333	653	319
3	8274	22130	9136
4	151186	500212	259462
5	2819281	10599317	7264947
6	51874770	223645588	not calculated

Figure 5: The asynchronous leader election protocol with property $\Diamond$(process P_1 is a leader).

According to these results it seems that the reduction on the number of transitions has a greater impact on the lp-reduction than the reduction on the number of states. This is as we expected because for each state that is ommited there is the possibility of "saving" several actions which do not occur as inequalities anymore. The runtime is omitted in the table since our implementation does not include full optimization for speed, yet.

[2] Note of course that this is a heuristic, too. There could be common variables with just the right content with α and β still being independent.

5. CONCLUSION AND FUTURE WORK

We described the main concepts for verifying randomized protocols with the partial order approach and a Promela-like modelling language to specify parallel process-based models that can contain both nondeterministic and probabilistic choices and is able to handle standard data elements like communication-channels and variables. Experiments with randomized protocols (such as leader election and other coordination algorithms for distributed systems) show promising reductions of the state space. This leads to smaller linear programs that necessarily go along with quantitative analysis of probabilistic reactive systems. Reductions on the size of the linear program directly impact on computation times.

A broader list of LiQuor's planned features include dynamic process creation and destruction, the use of Streett and Rabin-automata, improvents on the search of end components, improvements on the partial order heuristics and an efficient connection to linear program solvers. All of these features are currently under development, some of them will be included in a forthcoming public version of LiQuor.

6. REFERENCES

[1] C. Baier, F. Ciesinski, and M. Größer. Probmela: a modeling language for communicating probabilistic systems. MEMOCODE'04, pages 57–66. IEEE CS Press, 2004.

[2] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. ICALP'97, LNCS 1256, pages 430–440, 1997.

[3] C. Baier, P. D'Argenio, and M. Größer. Partial order reduction for probabilistic branching time. QAPL'05, To appear in ENTCS.

[4] C. Baier, M. Größer, and F. Ciesinski. Partial order reduction for probabilistic systems. QEST'04, IEEE CS Press, pages 230–239, 2004.

[5] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, editors. *Validation of Stochastic Systems*, LNCS 2925, 2003.

[6] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11:125–155, 1998.

[7] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. FST & TCS'95, LNCS 1026, pages 499–513, 1995.

[8] H. Bohnenkamp, H. Hermanns, J.-P. Katoen, and R. Klaren. The modest modeling tool and its implementation. *Computer Performance Evaluation/TOOLS*, pages 116–133, 2003.

[9] C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events (extended abstract). ICALP'90, LNCS 443, pages 336–349, 1990.

[10] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[11] P.R. D'Argenio and P. Niebert. Partial order reduction on concurrent probabilistic programs. QEST'04, IEEE CS Press, pages 230–239, 2004.

[12] L. de Alfaro. Temporal logics for the specification of performance and reliability. STACS'97, LNCS 1200, pages 165–179, 1997.

[13] L. de Alfaro. Formal Verification of Probabilistic Systems. Ph.D. thesis, Stanford University , 1997.

[14] E. Dijkstra. Guarded commands, non-determinacy and the formal derivation of programs. Comm. ACM, 18:453-457, 1975

[15] P. Godefroid. On the costs and benefits of using partial-order methods for the verification of concurrent systems. In *[26]*, pages 289–303, 1996.

[16] P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems: An Approach to the State Explosion Problem*, LNCS 1032, 1996.

[17] G. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*. Add.Wes., 2003.

[18] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87, 1990.

[19] B.Jeannet, P.d'Argenio and K.G. Larsen. RAPTURE: A tool for verifying Markov Decision Processes. *Proc.Tools Day / CONCUR'02*. Tech.Rep. FIMU-RS-2002–05,84–98, 2002.

[20] 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), to appear*, 2004.

[21] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, CS, 1996. MIT.

[22] C. Morgan and A. McIver. pGCL: formal reasoning for random algorithms. *Proc. WOFACS'98, Spec. Iss.of SACJ*, 22:14–27, 1999.

[23] C. Morgan and A. McIver. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.

[24] Terence Parr. *The ANTLR Reference Manual*. 2.6 edition, 1999.

[25] D. Peled. Partial order reduction: Linear and branching time logics and process algebras. In *[26]*, pages 79–88, 1996.

[26] D. Peled, V. Pratt, and G. Holzmann, editors. *Partial Order Methods in Verification*, volume 29(10) of *DIMACS*. American Mathematical Society, 1997.

[27] A. Valmari. Stubborn set methods for process algebras. In *[26]*, pages 79–88, 1996.

[28] M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. FOCS'85, IEEE CS Press, pages 327–338, 1985.

[29] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). LICS'86, pages 332–344, IEEE Computer Society Press, 1986.