Partial order reduction for model checking Markov decision processes under unconditional fairness

Henri Hansen
Department of Software Systems,
Tampere University of Technology
PO Box 553, FI-33101 Tampere, FINLAND
Email: henri.hansen@tut.fi

Marta Kwiatkowska and Hongyang Qu
Department of Computer Science, University of Oxford
Wolfson Building, Parks Road
Oxford, OX1 3QD, UK
Email: marta.kwiatkowska,hongyang.qu@cs.ox.ac.uk

***Abstract*—Fairness assumptions are needed to verify liveness properties of concurrent systems. In this paper we explore the so-called *unconditional fairness* in Markov decision processes (MDPs), which is a prerequisite for quantitative assume-guarantee reasoning. Unconditional fairness refers to executions where all processes are guaranteed to participate. We prove that realisability of unconditional fairness coincides with the absence of *partial deadlocks*, i.e., end components where a process suffers from starvation. We propose a weak variant of the stubborn set method to reduce MDPs, while preserving the realisability of unconditional fairness and maximal probabilities of reaching bottom end components under fair schedulers.**

***Keywords*-Probabilistic model checking; partial order reduction; Markov decision processes; unconditional fairness**

I. Introduction

Probabilistic model checking refers to a range of automated and systematic techniques for verification of systems exhibiting probabilistic behaviours. It has received much attention in the last two decades and has been applied to various problems, from communication protocols to biological systems. With continuous improvement of the technique, the size of systems that can be handled by probabilistic model checkers has increased. Nevertheless, the larger systems exceed the capacity of available tools. Recently, compositional verification has been introduced for probabilistic model checkers [1], [2]. In compositional verification, a system is verified by analysing modules of the system individually. Due to the probabilistic and concurrent characteristics of the problem, fairness conditions in general, and *unconditional fairness* in particular, are required for the verification of probabilistic liveness properties [2].

In this paper, we present and implement an approach to establish probabilistic properties under unconditional fairness. In our approach, we incorporate partial order reduction to tackle the so called state space explosion problem. Partial order reduction has been studied intensively for non-probabilistic system, such as [3], [4], [5], [6]. For probabilistic systems, only the ample set approach has been investigated [7], [8], [9], [10] so far as ample sets are comparably easier to compute than stubborn sets, in spite of the better reduction from the latter. [11] gave a survey on these methods. Our paper is the first to combine the stubborn set approach and probabilistic model checking. In addition, we adopt the weak stubborn set method, instead of the well-known (strong) stubborn set, to obtain more compact reduced models. Weak stubborn sets are often regarded as impractical, but Hansen and Wang have recently described an efficient and accessible method [12]. It is worth noting that our approach can also be applied to probabilistic verification without fairness constraints. Another contribution of this paper is the realisability check for unconditional fairness, which can be performed efficiently on the non-probabilistic version of the original model. This is very useful for examining the compatibility of the components in a parallel composition.

This paper is structured as follows. Section II presents the background knowledge about LTSs, MDPs and unconditional fairness (UF). In Section III, we discuss our approach for checking realisability of UF in MDPs, which can be reduced to the realisability of UF in LTSs. In Section IV, we use the *weak stubborn sets* to generate a reduced model and to compute the maximum/minimum probability of reaching a set of target states under unconditional fairness. The model checking algorithm for computing the max/min probability is explained in Section V. We briefly discuss some implementation choices and present experimental results in Section VI. We conclude the paper in Section VII.

II. Preliminaries

A. MDPs and LTSs

MDP An MDP is a tuple $\mathcal{M} = (S, \Sigma, \mathsf{Steps}, \overline{s})$ where

- S is a finite set of *states*,
- $\overline{s} \in S$ is the *initial state*,
- Σ is a finite set of *actions*,
- $\mathsf{Steps} : (S \times \Sigma) \to \mathsf{Dist}(S) \cup \{\perp\}$ is the *probabilistic transition function*.

The probabilistic transition function Steps maps each state $s \in S$ and action $a \in \Sigma$ to an element of $\mathsf{Dist}(S) \cup \{\perp\}$. For each state $s \in S$, Steps defines a set of enabled actions $en(s)$: $en(s) = \{a \in \Sigma \mid \mathsf{Steps}(s, a) \neq \bot\}$. We usually write $s \xrightarrow{a}$ to denote $a \in en(s)$. Steps associates a probability distribution function μ to each enabled action a at s. For convenience, we assume $\bot$ simply assigns the value zero to all states.

There are two steps to determine a successor state of s. First, an action is chosen non-deterministically from the set of enabled actions. Next, the probability of moving to state s' is decided by μ, i.e., $\mu(s')$. We assume there are no terminating states in MDPs.

An action a is said to be *deterministic* if for all states $\mathsf{Steps}(s, a)$ is either $\bot$ or the *Dirac-distribution*, i.e., it assigns the probability 1 to some state. If an action is not deterministic, we say that it is *probabilistic*.

A *path* in an MDP is a non-empty sequence of the form

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots$$

where for all $i \geq 0$, $s_i \in S$, and there is some μ_{i+1} such that $\mu_{i+1} = \mathsf{Steps}(s_i, a_{i+1})$ and $\mu_{i+1}(s_{i+1}) > 0$. We use $\omega(i)$ to denote the ith state in the path ω, ω_{fin} to denote a finite path, $last(\omega_{fin})$ to denote the last state in ω_{fin}, and $Path_s^{fin}$ ($Path_s$) to denote the set of all finite (infinite) paths starting in state s. Moreover, the ith action in ω is represented by $step(\omega, i)$, and the i-length prefix of ω is denoted ω_i.

The sequence of actions that occur in ω is denoted by $\Sigma(\omega)$. An *execution* of $\mathcal{M}$, from s to s_n, is denoted $s \xrightarrow{a_1 \cdots a_n} s_n$, and means that there is some path ω such that $\Sigma(\omega) = a_1 \cdots a_n$ and $\mathsf{last}(\omega) = s_n$.

The *probability measure* of a finite path, denoted $p(\omega)$, is defined as the product $\prod_{i=1}^{n} \mu_i(\omega(i))$. In addition, we define the probability of executions as

- $p(s \xrightarrow{a} s') = \mathsf{Steps}(s, a)(s')$,
- $p(s \xrightarrow{a_1 \cdots a_n} s') = \sum_{s^* \in S} p(s \xrightarrow{a_1 \cdots a_{n-1}} s^*) \mathsf{Steps}(s^*, a_n)(s')$.

To resolve the non-deterministic choices when we execute an MDP, we employ an *adversary* to select an action based on the history of choices made so far.

Adversary An adversary A of an MDP $\mathcal{M}$ is a function mapping every finite path ω_{fin} onto an element $A(\omega_{fin})$ of the set $en(last(\omega_{fin}))$. We use $Adv_{\mathcal{M}}$ to denote the set of all possible adversaries of the MDP and, for any adversary A, we use $Path_s^A$ denote the subset of $Path_s$ defined as

$$\{\omega \mid \omega(0) = s \wedge \forall i. \omega(i) \xrightarrow{A(\omega_i)} \omega(i+1)\}$$

We say that the paths in $Path_s^A$ are *controlled* by A.

End component An end component (EC) of $\mathcal{M}$ is a pair (R, A) such that $R \subseteq S$, $A : R \to 2^{\Sigma}$ and:

- $\emptyset \neq A(s) \subseteq en(s)$ for every $s \in R$,
- For each $a \in A(s)$ and $s \in R$, $\{s' \mid \mathsf{Steps}(s, a)(s') > 0\} \subseteq R$,
- R is strongly connected in the transition-state graph of $\mathcal{M}$.

A *bottom EC* is an end component with the condition $A(s) = en(s)$ for all states $s \in R$. Unless stated otherwise, we assume an EC to be maximal, i.e., that there is no end component (R', A') such that $R \subset R'$.

Parallel composition Let $M_i = (S_i, \Sigma_i, \mathsf{Steps}_i, \overline{s}_i)$ be MDPs. Then the parallel composition $M_1 || M_2$ is the MDP $(S, \Sigma, \mathsf{Steps}, \overline{s})$, where:

- $S = S_1 \times S_2$, • $\Sigma = \Sigma_1 \cup \Sigma_2$, • $\overline{s} = (\overline{s}_1, \overline{s}_2)$,
- $\mathsf{Steps}((s_1, s_2), a) = \mu_1 \times \mu_2$ iff
  - $a \in \Sigma_1 \cap \Sigma_2$, $\mathsf{Steps}_1(s_1, a) = \mu_1$ and $\mathsf{Steps}_1(s_2, a) = \mu_2$, or
  - $a \in \Sigma_1 \backslash \Sigma_2$, $\mathsf{Steps}_1(s_1, a) = \mu_1$, and $\mu_2 = \eta_{s_2}$, or
  - $a \in \Sigma_2 \backslash \Sigma_1$, $\mathsf{Steps}_2(s_2, a) = \mu_2$ and $\mu_1 = \eta_{s_1}$

where $\mu_1 \times \mu_2$ is the product of distributions μ_1 and μ_2, and $\eta_{s_i} \in Dist(S_i)$ is the Dirac-distribution on $s_i \in S_i$.

For a state s of the parallel composition of n components, we write $s(i)$ for the state of the ith component and $en_i(s) = \{a \in \Sigma_i \mid s(i) \xrightarrow{a}_i\}$.

LTSs are a special type of MDPs[1], where each distribution is a Dirac-distribution. To distinguish LTSs from the full class of MDPs, we write $L = (S, \Sigma, \Delta, \overline{s})$ for an LTS. An end component in LTSs is also called a *strongly connected component* (SCC).

B. Computing maximum reachability probabilities

Verification of MDPs usually requires properties to be specified in temporal logics, e.g., PCTL [13], [14] and LTL [15]. *Reachability probabilities* are one of the key properties central to probabilistic verification. In this paper, we focus on *maximum reachability probabilities* of reaching a set of target states $\mathcal{F} \subseteq S$ from s, over all possible adversaries:

$$p_s^{\max}(\mathcal{F}) = \sup_{A \in \mathsf{Adv}_{\mathcal{M}}} p_s^A(\mathcal{F})$$
$$p_s^A(\mathcal{F}) = \mathsf{Prob}_s^A(\{\omega \in \mathsf{Path}_s^A \mid \exists i \,.\, \omega(i) \in \mathcal{F}\}).$$

Our technique can be extended to *minimum reachability probabilities* in a straightforward manner.

Usually, $p_s^{\max}(\mathcal{F})$ can be computed by *value iteration* or *linear programing*. We use value iteration with partial order reduction, as we dynamically prune transitions to speed up the state space exploration and probability computation. In value iteration, a vector is used to store the current estimation of the probability of states. Iteratively, the vector is updated by the multiplication of itself and the probabilistic transition relation, which is represented as a matrix. Let

[1]Strictly speaking, LTSs allow non-deterministic actions, which does not affect our proof in next section. Due to the page limit, we do not provide the full definition of an LTS.

$p_s^{\max,k}$ be the probability of state s in the kth iteration. Initially, i.e., $k = 0$, the states in $\mathcal{F}$ have probability one in the vector, and other states have probability zero. For $k > 0$, $p_s^{\max,k}$ is defined as follows:

$$p_s^{\max,k} := \begin{cases} 1 & \text{if } s \in \mathcal{F} \\ \max\limits_{\mu \in \mathsf{Steps}(s)} \sum\limits_{s' \in S} \mu(s') \cdot p_{s'}^{\max,k-1} & \text{if } s \in S \backslash \mathcal{F}. \end{cases}$$

Eventually, $p_s^{\max,k}$ is guaranteed to converge to $p_s^{\max}(\mathcal{F})$. In practice, *precomputation*, such as $Prob0A$ [14] and $Prob1E$ [16], is performed before value iteration starts in order to identify states that have either probability one or zero. These states are excluded from value iteration to save time and space. However, precomputation cannot be applied before the state space is generated by partial order reduction. Therefore, we omit it in this paper. We also write $p_s^{\max}$, instead of $p_s^{\max}(\mathcal{F})$ for simplicity. In [17], [18], an MDP is decomposed into SCCs first and value iteration is carried out in each SCC in the reversed topological order, rather than in the whole model directly. Experimental results in both papers show that computation time can be dramatically reduced in this way.

C. Unconditional fairness

Unconditional fairness is crucial in probabilistic assume-guarantee reasoning [2], as it guarantees that the intermediate adversary computed during reasoning is not partial.

Unconditional fairness Given an MDP $\mathcal{M} = (S, \Sigma, \mathsf{Steps}, \overline{s})$ composed of n components $\mathcal{M}_1, \ldots, \mathcal{M}_n$, i.e., $\mathcal{M} = \mathcal{M}_1 \parallel \ldots \parallel \mathcal{M}_n$, an infinite path π is *unconditionally fair*, denoted $\pi \models fair$, if for each component $\mathcal{M}_i$ there exists an action $a \in \Sigma_i$ that occurs infinitely often in the path. An adversary A is *unconditionally fair*, or *fair* for short, if for every state $s \in S$, every infinite path starting at s controlled by A is unconditionally fair.

Realisability Unconditional fairness is *realisable* in an MDP $\mathcal{M}$ if there exists some fair adversary for $\mathcal{M}$.

A state is *visited* by an adversary if the state is reached from the initial state in an execution controlled by the adversary.

Generalised fair adversary When unconditional fairness is not realisable in an MDP M, an adversary A is *generalised fair* (or *fair* for short) if for every state $s \in M$ that is visited by A, all infinite paths starting at s are fair.

If there does not exist any fair adversary, then the maximum probabilities are restricted to *generalised fair* adversaries.

To compute the maximum probabilities for an MDP under unconditional fairness, we need to reduce the model, i.e., leave out the parts where fairness is not realisable. Basically, states that cannot be visited by any fair adversary should be removed, as suggested in [10], so that we only perform value iteration over the remaining states. We address this issue in Section V.

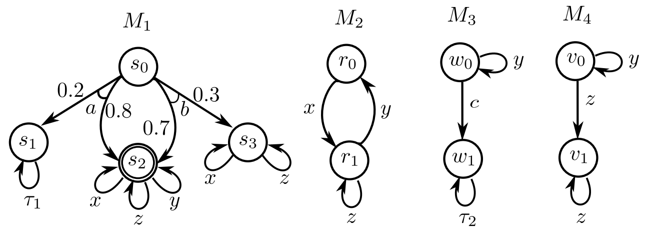

Figure 1. An MDP with four components M_1, M_2, M_3 and M_4

Example Figure 1 shows an MDP with four components. Actions x, y and z are synchronised, and the others are local. Unconditional fairness is not realisable in this model because, in any global state containing s_1, no actions in M_2 and M_4 can be executed. The maximum probability of reaching the target state s_2, marked by a double circle in the figure, is 0.8 if we do not consider fairness, while it becomes 0.7 under unconditional fairness.

III. Checking realisability

In the rest of the paper, let $L = (S, \Sigma, \Delta, \overline{s})$ be composed of n components $L_1, \ldots, L_n$, i.e., $L = L_1 \parallel \ldots \parallel L_n$, and $\mathcal{M} = \mathcal{M}_1 \parallel \ldots \parallel \mathcal{M}_n$. Let $\mathsf{Reach}(s) \subseteq S$ be the set of states that can be reached from state s.

Partial deadlock The process L_i is *partially deadlocked* in state $s \in S$ if for each $s' \in \mathsf{Reach}(s)$ and each $a \in \Sigma_i$ we have $s' \stackrel{a}{\not\to}$. A *partial deadlock* of L_i is a bottom SCC of L such that L_i is partially deadlocked in all the states of the SCC.

Theorem 3.1: Unconditional fairness is realisable in L iff, for all L_i, there does not exist a partial deadlock in L.

Proof: "$\Rightarrow$". Suppose there exists a partial deadlock Ω for L_i. Consequently, L_i is partially deadlocked in any state $s \in \Omega$. Then there does not exist an infinite path π starting at s such that action a appears infinitely often in π for some $a \in \Sigma_i$. This is in contradiction with the definition of unconditional fairness.

"$\Leftarrow$". Suppose unconditional fairness is not realisable in L. Thus we can find a state $s \in S$ where it holds that, for all infinite paths starting at s, there is some i such that no $a \in \Sigma_i$ occurs infinitely often. In particular, this holds for all infinite paths that lead from s to some bottom SCC C, and visit all the states of C in that SCC infinitely often. Therefore, no $a \in \Sigma_i$ is enabled in any state of C, and C is a deadlock for L_i. ■

In [10, Theorem 1], Baier et. al showed that a strong/weak fairness constraint is realisable if and only if, for every state s in an MDP $\mathcal{M}$, there exists a path with non-zero probability leading to a fair maximal end component (FMEC). Since unconditional fairness can be formulated as a strong fairness constraint, this result still holds. An FMEC in our case is a maximum end component that contains at least one action for each $\mathcal{M}_i$. Let $L_{\mathcal{M}} = (S_{\mathcal{M}}, \Sigma_{\mathcal{M}}, \Delta_{\mathcal{M}}, \overline{s}_{\mathcal{M}})$ be an LTS obtained by converting probabilistic choice in an MDP $\mathcal{M} = (S, \Sigma, \mathsf{Steps}, \overline{s})$ into non-deterministic choices as follows:

- $S_{\mathcal{M}} = S$,
- $\overline{s}_{\mathcal{M}} = \overline{s}$,
- For any $s \in S$ and any $a \in \Sigma$, $a_{s'} \in \Sigma_{\mathcal{M}}$ and $(s, a_{s'}, s') \in \Delta_{\mathcal{M}}$ iff $\mathsf{Steps}(s, a)(s') > 0$.

Synchronisation of actions in the resulting LTS-parallel composition is somewhat involved; all the actions resulting from actions with the same labels are synchronised, i.e., multi-way synchronisation, which does not affect partial order reduction [12].

Theorem 3.2: Unconditional fairness is realisable in an MDP $\mathcal{M}$ iff, for all $L_{\mathcal{M}_i}$, there does not exist a partial deadlock in $L_{\mathcal{M}}$.

The proof is similar to that of Theorem 3.1, and hence omitted here.

T-reduction Let $L = (S, \Sigma, \Delta, \overline{s})$ be an LTS, and let $T : S \to 2^{\Sigma}$ be some function. Then the *T-reduction* of L is the LTS $(S_T, \Sigma, \Delta_T, \overline{s})$, where S_T and Δ_T are minimal sets such that:

- $\overline{s} \in S_T$;
- if $s \in S_T$, $(s, a, s') \in \Delta$ and $a \in T(s)$, then $s' \in S_T$ and $(s, a, s') \in \Delta_T$.

We also use the subscript $s \xrightarrow{a}_T$ when talking about the reduced LTS.

Stubborn set Let $(S, \Sigma, \Delta, \overline{s})$ be an LTS, and let $T : S \to 2^{\Sigma}$. we say that the function T is a *stubborn set generator* if, for all $s \in S$, $T(s)$ satisfies the following properties:

D0 $T(s) \cap en(s) = \emptyset$ iff $en(s) = \emptyset$,

D1 For all $a \in T(s)$ and $b_1, \ldots, b_n \in \Sigma \backslash T(s)$, and $s' \in S$, if $s \xrightarrow{b_1 \cdots b_n a} s'$ then $s \xrightarrow{a b_1 \cdots b_n} s'$,

D2 Either $en(s) = \emptyset$, or there exists $a \in T(s)$ such that, for all $b_1, \ldots, b_n \in \Sigma \setminus T(s)$, $s \xrightarrow{b_1 \cdots b_n a} s'$. An action $a \in T(s)$ with this property is called a *key action*.

The stubborn set defined above is sometimes referred to as a *weak* stubborn set, in contrast to the (strong) stubborn set. The difference is in the condition **D2**, which for strong stubborn sets is replaced by a condition that requires that *all* enabled actions of $T(s)$ are key actions. If we reduce an LTS with a (weak or strong) stubborn set generator, it is known that the reduced LTS contains exactly the deadlocks of the original LTS [19]. A case where **D0–D2** are not enough to preserve *partial* deadlocks can also be found in Figure 1. In the state (s_0, r_0, w_1, v_0), the set $\{\tau_2\}$ is a legitimate stubborn set, but when M_1 is in state s_1, M_2 and M_4 are partially deadlocked. What is more, in the resulting reduced LTS, there is an "extra" partial deadlock induced, as the reduced LTS would stay in that state for ever, and all other processes would be "partially deadlocked".

The phenomenon is a manifestation of the so-called *ignoring problem* [20], [21], whereby a cycle of actions is considered completed, with some transition enabled on the way and never explored. When looking for global deadlocks, this is not a problem. However, the stubborn set conditions do not guarantee that we find all *partial* deadlocks, and thus the problem needs to be addressed.

To solve the problem, an additional constraint must be imposed on stubborn sets. This is the *safety condition* [22]:

S For every state s in S_T and every $b \in en(s)$, there exists some $a_1, \ldots, a_n$ such that $s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$, all a_i are key actions, and $b \in T(s_n)$.

Lemma 3.3 ([3], [22]): Let T be a stubborn set generator for L such that T satisfies **D0** – **D2** and **S**. Then, for every bottom SCC $C \subseteq S$, $C \cap S_T \neq \emptyset$.

Theorem 3.4: If the stubborn set generator T satisfies **D0** – **D2** and **S**, then the reduced LTS $L_T = (L_1 || \cdots || L_n)_T$ contains a partial deadlock if and only if $L = (L_1 || \cdots || L_n)$ does.

Proof: The "if"-part follows from Lemma 3.3. Let C be a bottom SCC of L_T that is a partial deadlock for L_j in L_T. For proof by contradiction, assume there is no partial deadlock for L_j in L. Let $s_0 \in C$ and $b \in \Sigma_j$ such that $s_0 \xrightarrow{a_1 \cdots a_n b}$ is minimal among states of C. An obvious contradiction comes if for some i, $a_i \in T(s_0)$, as taking minimal such i would give $s_0 \xrightarrow{a_i}_T s_0'$ and there would be a path of length $n-1$ to a state where b is enabled.

Therefore, assume that each $a_i \notin T(s_0)$. **S** guarantees a path of key actions $x_1, x_2, \cdots, x_m$ such that $s_0 \xrightarrow{x_1 \cdots x_m}_T s_0'$ and $a_1 \in T(s_0')$. **D1** guarantees, with a simple induction over m, that either one a_i is in a stubborn set in one of the states on the path, contradicting minimality that was previously assumed, or that minimality is contradicted by the state s^* such that $s_0' \xrightarrow{a_1}_T s^*$. ■

IV. Preserving max probabilities under unconditional fairness

A. Preserving terminal states

Probabilistic stubborn set Let $\mathcal{M} = (S, \Sigma, \mathsf{Steps}, \overline{s})$ be an MDP. We say that the function $T : S \to 2^{\Sigma}$ is a *probabilistic stubborn set generator* if it satisfies the following properties:

D0 $T(s) \cap en(s) = \emptyset$ if and only if $en(s) = \emptyset$.

D1 For every $a \in T(s)$ and $b_1, \ldots, b_n \notin T(s)$, $s \xrightarrow{b_1 \cdots b_n a} s'$ implies $p(s \xrightarrow{b_1 \cdots b_n a} s') = p(s \xrightarrow{a b_1 \cdots b_n} s')$.

D2 If $T(s) \neq \emptyset$, then there exists at least one $a \in T(s)$ such that for all $b_1, \ldots, b_n \notin T(s)$, $s \xrightarrow{b_1 \cdots b_n} s'$ implies $s' \xrightarrow{a}$. Such an action is called a *key action*.

Reduced MDP Let $\mathcal{M} = (S, \Sigma, \mathsf{Steps}, \overline{s})$ be an MDP and T be a stubborn set generator for $\mathcal{M}$. We define the *T-reduced* MDP, denoted $\mathcal{M}_T = (S_T, \Sigma, \mathsf{Steps}_T, \overline{s})$, as the minimal MDP that satisfies: if $s \in S_T$, $a \in T(s)$, then $\{s' \mid \mathsf{Steps}(s,a)(s') > 0\} \subseteq S_T$ and $\mathsf{Steps}_T(s,a) = \mathsf{Steps}(s,a)$.

To preserve safety properties and partial deadlocks for MDPs, we need to tackle the ignoring problem in end components. There are various ways of doing this in the context of partial order reduction for MDPs. Such conditions are often referred to as *cycle conditions*. Two variants in the literature [7], [8] include:

A3a If (R, A) is an EC of $\mathcal{M}_T$ and $a \in \bigcap_{s \in R} en(s)$ then $a \in \bigcup_{s \in R} T(s)$;

A3b On each cycle $s_0 \xrightarrow{a_1 \cdots a_n} s_n = s_0$ there exists a state s_i which is fully expanded, i.e., $T(s) = en(s)$.

Of the two, A3b is clearly stronger, but also easier to implement, which is why most practical implementations adopt it. However, we do not actually need such powerful conditions if we are concerned only with reachability and other safety properties. Instead, we use the **S** condition as it stands.

It is a well-known problem for maximal probabilities, first identified in [7], [9], that the usual non-probabilistic rules do not guarantee the preservation of maximal probabilities. This is true with rules **D0–D2** and **S**. The most common fix for this probabilistic branching problem is to impose a rule:

A5 If $s \xrightarrow{b_1 \cdots b_n c}$ such that $b_1, \ldots, b_n, c \notin T(s)$, and c is probabilistic, then $|T(s) \cap en(s)| = 1$.

Note that, for ample sets, in which we assume $T(s) \subseteq en(s)$, this rule seems to be different than for stubborn sets. The difference, however, is only superficial. Rule **D1** disallows actions outside the stubborn set to enable disabled actions within the stubborn set. The intricacies of the stubborn set method do allow, however, a more careful analysis of the condition. Rule A5 is hard to implement, as mentioned in [7], which is why a number of heuristics have been proposed for it. For instance, in [7], a stronger version is given, which requires that $T(s) \cap en(s)$ is a singleton unless it contains all the enabled actions.

We propose to replace A5 with another rule that is stronger in the context of stubborn sets, in the sense that, with the support of **D1**, this rule implies A5. This rule is significantly weaker than the simplistic rule of always requiring singleton sets, and should allow for more reduction.

PB If $|T(s) \cap en(s)| \neq 1$, then $T(s)$ contains all probabilistic actions in the model.

Lemma 4.1: If $T(s)$ satisfies **D0–D2** and **PB**, then it satisfies A5.

Proof: Let $|T(s) \cap en(s)| \neq 1$ and $s \xrightarrow{b_1 \cdots b_n c}$, c be probalistic and $b_1, \ldots, b_n$ deterministic. **PB** implies that $c \in T(s)$. **D1** guarantees that either c is enabled at s, or at least one $b_i \in T(s)$. In both cases, then, A5 is satisfied. ■

The exact relationship of **PB** and A5 is unknown to us. On the face of it, **PB** does seem stronger, because it requires that disabled probabilistic actions are inside the set, but definite evidence is lacking at this time.

Theorem 4.2: If $T(s)$ satisfies **D0–D2**, **S** and **PB**, then the maximum probability of reaching each bottom EC is preserved through T-reduction.

Proof: Let C be a bottom EC in the original MDP $\mathcal{M}$. Let us denote by $p_{s_0}^{\max}(C)$ the maximum probability of ending in C from s_0 in $\mathcal{M}$, and by $q_{s_0}^{\max}(C)$ the maximum probability in the reduced MDP $\mathcal{M}_T$. If $s_0 \in C$, then naturally $p_{s_0}^{\max}(C) = q_{s_0}^{\max}(C)$. This also holds if $p_{s_0}^{\max}(C) = 0$, i.e., C is not reachable from s_0. We assume this is not the case.

It is known that the maximum probability of reaching C can be realised by a memoryless adversary. Let $\mathcal{A}$ be such an adversary, and $\mathcal{A}(s_0)$ the only action chosen by $\mathcal{A}$ for s_0. We prove that there exists a memoryless adversary $\mathcal{A}_T$ in $\mathcal{M}_T$ such that $p_{s_0}^{\mathcal{A}}(C) = q_{s_0}^{\mathcal{A}_T}(C)$, where $p_{s_0}^{\mathcal{A}}(C)$ and $q_{s_0}^{\mathcal{A}_T}(C)$ are the probability of reaching C under $\mathcal{A}$ and $\mathcal{A}_T$ respectively.

If $\mathcal{A}(s_0) \in T(s_0)$, then $\mathcal{A}_T(s_0) = \mathcal{A}(s_0)$, and nothing else is needed. Otherwise, to show that $p_{s_0}^{\mathcal{A}}(C) = q_{s_0}^{\mathcal{A}_T}(C)$, we show that we can choose $\mathcal{A}_T(s_0) = b$ such that $p_{s_0}^{\max}(C) = \sum_{s'_0} \mu(s'_0) \cdot p_{s'_0}^{\max}(C)$, where $\mu = \mathsf{Steps}(s_0, b)$. **S** then guarantees that the execution can always make progress.

a) If $\mathcal{A}(s_0)$ is probabilistic, then $T(s_0) \cap en(s_0)$ is a singleton (**PB**). In this case, any sequence $s_0 \xrightarrow{a_1 \cdots a_n} s_n$ that does not contain b can be extended by b, and does not have any other actions from $T(s_0)$ (**D1**), and $p(s_0 \xrightarrow{b a_1 \cdots a_n} s'_n) = p(s_0 \xrightarrow{a_1 \cdots a_n b} s'_n)$. In particular, this holds for sequences where $a_{i+1} = \mathcal{A}(s_i)$, and thus if $s_n \in C$, then, because C is a bottom EC and also $s'_n \in C$, we obtain $p_{s_0}^{\max}(C) = \sum_{s'_0} \mu(s'_0) p_{s'_0}^{\max}(C)$.

b) **PB** guarantees that, if $T(s_0) \cap en(s_0)$ is not a singleton (which means that $\mathcal{A}(s_0)$ is not probabilistic), then in any sequence $s_0 \xrightarrow{a_1 \cdots a_n} s_n$ that starts with $\mathcal{A}(s_0)$, either the first probabilistic action or some action before it is in $T(s_0)$. Let k be the index of such an action. **D1** and the fact that a_i is deterministic for $i < k$ guarantee that:

1) $s_0 \xrightarrow{a_1 \cdots a_{k-1}} s_{k-1}$ is a deterministic sequence for some state s_{k-1}, and
2) for each s'_0 such that $\mathsf{Steps}(s_0, a_k)(s'_0) > 0$, we have $s'_0 \xrightarrow{a_1 \cdots a_{k-1}} s'_{k-1}$ for some state s'_{k-1}, and $\mathsf{Steps}(s'_{k-1}, a_k)(s'_k) = \mathsf{Steps}(s_0, a_k)(s'_0)$.

Again, this holds for all the sequences such that $\mathcal{A}(s_i) = a_{i+1}$, so we have that $p_{s_0}^{\max}(C) = \sum_{s'_0} \mu(s'_0) p_{s'_0}^{\max}(C)$.

When case b) happens, one action from the sequence leading to the SCC is executed. On the other hand, the a) case cannot happen indefinitely: **S** guarantees that, because $a_1 \in en(s)$, there is a sequence of key actions that lead to a state where a_1 is in the stubborn set, and in the a) case, there is only a single key action, so either a_1 is eventually in the stubborn set or the b) case materialises. ■

Theorem 4.2 does not consider the maximum probabilities under unconditional fairness. We have to prune actions that can lead to a partial deadlock in order to do so. The following theorem guarantees that the maximum probability under unconditional fairness in a reduced MDP is the same as in the original MDP.

Theorem 4.3: If $T(s)$ satisfies **D0–D2**, **S** and **PB**, then the maximum probability of reaching a bottom FMEC under unconditional fairness is preserved through T-reduction.

Proof: Note that a bottom FMEC is a bottom SCC that has no partial deadlock, or an FMEC from which only partially deadlocking bottom SCCs can be reached.

In the first case, we only need to consider states where we can reach a number of bottom SCCs, some of which are FMECs. Otherwise Theorem 4.2 suffices. Let $s \in S_T$ and let $\mathcal{A}_F$ be the fair adversary that maximises the probability. $\mathcal{A}_F$ will never assign an action that leads to a state from which the maximal probability of reaching a fair bottom SCC is less than one.

Using a construction similar to the one in the proof of Theorem 4.2, we can prove that either there is some action $b \in T(s)$ such that $\mathcal{A}_F$ might as well assign b, or that there is some sequence of key actions that lead to the execution of $\mathcal{A}_F(s)$, and hence the maximum probability of reaching the FMEC is the same. ■

V. Model checking algorithm

A. Generating basic stubborn sets

The concept of *independence* of actions is central in most partial order reduction methods. It is important also for the practical implementation of stubborn sets.

Traditionally, independence means that, when two actions are enabled at the same state, they commute in the sense that both actions can always be taken and the result is independent of the order in which they are executed. We shall refer to such independence as *strong independence*.

Strong Independence Let $\mathcal{M} = (S, \Sigma, \mathsf{Steps}, \overline{s})$ be an MDP and $a, b \in \Sigma$. We say that a and b are *strongly independent* iff, for all states $s \in S$ such that $a \in en(s)$ and $b \in en(s)$, we have:

- for all $s_1 \in S$, $s \xrightarrow{a} s_1 \Rightarrow b \in en(s_1)$,
- for all $s_1 \in S$, $s \xrightarrow{b} s_1 \Rightarrow a \in en(s_1)$,
- for all $s' \in S$ it is true that $p(s \xrightarrow{ab} s') = p(s \xrightarrow{ba} s')$.

If a and b are not strongly independent, they are *strongly dependent*.

A careful examination of the rules for stubborn sets, however, reveals that **D1** is not entirely symmetric. Commutativity of an inside action with some sequence of outside actions is required only if executing a sequence of outside actions, followed by an inside action, is possible to begin with. In fact, **D1** in no way rules out the possibility of a becoming disabled by a sequence of b-actions. What is more, **D2** requires only that we have *at least one* action that cannot be disabled by outside sequences.

The so-called *weak independence* is designed to capture this asymmetry. We require commutativity only if a can be executed after b. The relation does allow for b and a to disable one another.

Weak Independence Let $\mathcal{M} = (S, \Sigma, \mathsf{Steps}, \overline{s})$ be an MDP and $a, b \in \Sigma$. We say that a is *weakly independent* of b iff, for all states $s \in S$ such that $s \xrightarrow{ba}$, it holds that $a \in en(s)$ and:

- for all $s_1 \in S$, $s \xrightarrow{a} s_1 \Rightarrow b \in en(s_1)$,
- for all $s_1 \in S$, $s \xrightarrow{b} s_1 \Rightarrow a \in en(s_1)$,
- for all $s' \in S$, it is true that $p(s \xrightarrow{ab} s') = p(s \xrightarrow{ba} s')$.

If a is not weakly independent of b, we say that a is *weakly dependent* with b.

We emphasise the difference: the requirement that a and b commute is the same as in strong independence, but it is "triggered" only if a can be executed after b. In fact, if we know that b completely disables a, then a is weakly independent of b.

The concept of *causes* and causality is also helpful in designing a practical method for calculating stubborn sets. **D1** implies that, if a is a disabled inside action, then no sequence of outside actions may enable a. Verifying this can be computationally expensive, and, what is more, a simple binary relation for actions may not fully capture all the intricacies of action sequences enabling one another.

We introduce the following notational convention for 3-ary relations: if Q is a 3-ary relation, we write in the following $(a, c, -) \in Q$ if there exists some b such that $(a, c, b) \in Q$. We define the causality relation as follows.

Causality relation Let C be a finite set of *causes* at a state s. A *causality relation* at s is a 3-ary relation $Q \subseteq \Sigma \times C \times \Sigma$, such that for every $a \notin en(s)$, and $b_1, \ldots, b_n \neq a$ with $s \xrightarrow{b_1 \cdots b_n a}$, it holds that for every c such that $(a, c, -) \in Q$ we have $(a, c, b_i) \in Q$ for at least one $i \in \{1, \ldots, n\}$.

The intuition of this definition is as follows. For a disabled action a, there is some set of *causes* that need to be "triggered" before a can happen. This is some subset of C. Intuitively, C can be thought of as a set of guards or conditions that must be met in order for actions to become enabled. For example, in Petri nets, "actions" would be the structural transitions and "causes" would be the places that are empty. Now, if a is disabled in the current state, but enabled after some sequence of actions, then all the causes of a must be triggered by some action.

Note that the definition is conditioned on s, i.e., it is dynamic. The set of causes can of course be static; the only dynamic feature is when a cause is "active" (e.g., a true guard) and when it is "inactive" (e.g.., a false guard). In a parallel composition, for instance, the "inactive" causes can simply be the components where the corresponding action is disabled, and "active" causes are those where it is enabled. The action becomes enabled when all causes are active.

Example In Figure 1, actions a and b are strongly dependent of each other, as one disables the other. Similarly, y and z, as well as y and c are strongly dependent. Action c is weakly dependent of y, but not the other way around, because the first condition in the definition of weak independence is violated, i.e., $(s_2, r_1, w_0, v_0) \xrightarrow{c} (s_2, r_1, w_1, v_0) \not\ni y \in en(s_2, r_1, w_1, v_0)$. For the same reason, z is weakly dependent of y. Actions a and b are the causes of x, y and z (through component M_1); x is a cause of z, and y is a cause of x (through component M_2).

Strong and weak dependency and causality are relations on top of which we shall build what we call a *dependency graph*. The role of this graph is to capture the information in the relations so that we can calculate stubborn sets.

Dependency graph Let $\mathcal{M} = (S, \Sigma, \mathsf{Steps}, \overline{s})$ be an MDP and $s \in S$ be a state. A *dependency graph* for s is a tuple $(E, D, C, \rightsquigarrow_S, \rightsquigarrow_W, \rightsquigarrow_C)$ such that $(E \cup D \cup C, \rightsquigarrow_S \cup \rightsquigarrow_W \cup \rightsquigarrow_C)$ is a directed graph satisfying the following conditions:

- $E = en(s)$,
- $D = \Sigma \setminus en(s)$,
- $C \neq \emptyset \vee D = \emptyset$,
- $\rightsquigarrow_S \subseteq E \times \Sigma$,
- $\rightsquigarrow_W \subseteq E \times \Sigma$,
- $\rightsquigarrow_C \subseteq (D \times C) \cup (C \times \Sigma)$,
- $\forall a \in D : \exists c \in C : a \rightsquigarrow_C c$.
- for each $v \in E$ and $u \in \Sigma$, either $v \rightsquigarrow_S u$ or v and u are strongly independent;
- for each $v \in E$ and $u \in \Sigma$, either $v \rightsquigarrow_W u$ or v and u are weakly independent;
- there exists a causality relation Q such that for $a \in D$, $c \in C$, and $b \in \Sigma$, $(a, c, b) \in Q$ if and only if $a \rightsquigarrow_C c \rightsquigarrow_C b$.

In the above definition, E is the set of enabled actions, D the set of disabled actions, and C the set of "inactive" causes for the actions in D. The relation $\rightsquigarrow_C$ links the disabled actions with their causes and causes to actions that can activate them.

Stubborn set for dependency graph A set $V \subseteq \Sigma \cup C$ is a *stubborn set for G* if

1) Either $E = \emptyset$ or $V \cap E \neq \emptyset$,
2) If $V \cap E \neq \emptyset$, then there exists a $v \in V \cap E$, such that: $\{u \mid v \rightsquigarrow_S u\} \subseteq V$,
3) For each $v \in V \cap E$, it holds that either $\{u \mid v \rightsquigarrow_S u\} \subseteq V$, or $\{u \mid v \rightsquigarrow_W u\} \subseteq V$,
4) For each $v \in V \cap C$, it holds that $\{u \mid v \rightsquigarrow_C u\} \subseteq V$,
5) For each $v \in V \cap D$ we have $\{u \mid v \rightsquigarrow_C u\} \cap V \neq \emptyset$.

We call these conditions the *closure properties* of the stubborn set. The first requirement is natural; V must contain some enabled action if there are any. The second requirement is there to make sure some key action is in the set. The third one requires that for each enabled action in V, either the weakly or the strongly dependent actions are in V. The fourth requirement says that if a cause is in V, then every action that can activate that cause must be in V; and the last one that, if a disabled action is in V, at least one of its causes must also be in V.

Theorem 5.1: If V is a stubborn set of a dependency graph G at s, then $V \cap \Sigma$ is a stubborn set at s.

Proof: **D0** follows from the first condition in the definition of stubborn set for G. For **D1**, let $a \in V \cap \Sigma$ and let $b_1, \ldots, b_n \in \Sigma \setminus V$, and $s' \in S$ be a state such that $s \xrightarrow{b_1 \cdots b_n a} s'$. There are two possibilities. (1) If $a \in en(s)$, then we can readily infer commutativity by induction from (weak or strong) independence. (2) If $a \in \Sigma \setminus en(s)$, then let Q be the causality relation required by definition. From the definition of causality, it follows that for at least one $i \in \{1, \ldots, n\}$, $b_i \in V$, which leads to contradiction. **D2** follows from the fact that there is some $a_k \in V \cap en(s)$ such that $\{u \mid a_k \rightsquigarrow_S u\} \subseteq V$. ■

Relations for (weak or strong) dependency, and causality, can be safely approximated in a number of ways. In a parallel composition, a common heuristic is to consider all the actions inside a given component dependent on one another, but such relations will not benefit from weak dependency. One possibility is to analyse each component, as in [12], so that every component is searched for manifestations of dependency and causality. The causality is implemented so that, for every action a and every i such that $a \in \Sigma_i$, there is a "cause" c_a^i, and (a, c_a^i, b) is in the causality relation if b enables a in some state of the ith component.

For shared variable models with, e.g., guarded commands, the effects of actions on guards need to be assessed. A common heuristic is to consider all write accesses as dependencies, but, again, a more nuanced approach is needed to make use of weak dependency.

Once the dependency and causality relations are known, and we know which causes are relevant at the current state, we can generate a dependency graph.

Example In the initial state (s_0, r_0, w_0, v_0) in Figure 1, the dependency graph $G = (E, D, C, \rightsquigarrow_S, \rightsquigarrow_W, \rightsquigarrow_C)$ is as follows:

- $E = \{a, b, c\}$,
- $D = \{x, y, z, \tau_1, \tau_2\}$,
- $C = \{\tau_1^1, \tau_2^3, x^1, y^1, y^2, z^1, z^2\}$,
- $\rightsquigarrow_S = \{(a, b), (b, a), (c, y)\}$,
- $\rightsquigarrow_W = \{(c, y)\}$,
- $\rightsquigarrow_C = \{(\tau_1, \tau_1^1), (\tau_2, \tau_2^3), (x, x^1), (y, y^1), (y, y^2), (z, z^1), (z, z^2), (x^1, a), (x^1, b), (\tau_1^1, a), (\tau_2^3, c), (y^1, a), (y^1, b), (y^2, x), (z^1, a), (z^1, b), (z^2, x)\}$.

The stubborn set for G is $\{a, b\}$, where both a and b are the key actions.

The algorithm for calculating stubborn sets is given in Algorithm 1. The basic idea of generating stubborn sets is as follows. We have a candidate stubborn set Stub, initially the set of all actions. Det is the set of deterministic actions in the model, and $\mathit{Prob} = \Sigma \setminus \mathit{Det}$ is the set of probabilistic actions of the model. The set U is a set of *required* actions. Usually this is empty, and has no effect on the generation of the stubborn set. Sometimes, to satisfy the **S**-condition, we must require that certain actions will be guaranteed in the stubborn set.

Algorithm 1 $T(s, U)$

```
1: $(E, D, C, \rightsquigarrow_S, \rightsquigarrow_W, \rightsquigarrow_C)$ is a dependency graph for $s$
2: $\mathit{Stub} := \Sigma \cup C$; $\mathit{Keys} = en(s)$; $\mathit{Weak} = en(s)$
3: $\mathit{Untried} := en(s) \setminus U$
4: while $\mathit{Untried} \neq \emptyset$ do
5:    pick $a$ from $\mathit{Untried}$
6:    $\mathit{Untried} = \mathit{Untried} \setminus \{a\}$
7:    if $U = \emptyset \wedge \mathit{Single}(a)$ then
8:       return $(\{a\}, \{a\})$
9:    end if
10:   if $a \notin \mathit{Det}$ then continue
11:   $\mathit{Stub}' := \mathit{Stub}; \mathit{Keys}' := \mathit{Keys}; \mathit{Weak}' := \mathit{Weak}$
12:   $\mathit{Delete}(a)$
13:   if $\mathit{Keys} = \emptyset \vee \mathit{Prob} \not\subseteq \mathit{Stub} \vee U \not\subseteq \mathit{Stub}$ then
14:      $\mathit{Stub} := \mathit{Stub}'; \mathit{Keys} := \mathit{Keys}'; \mathit{Weak} := \mathit{Weak}'$
15:   end if
16: end while
17: return $(\mathit{Stub} \cap en(s), \mathit{Keys})$
```

Algorithm 2 $\mathit{Delete}(a)$

```
1: $\mathit{Stub} := \mathit{Stub} \setminus \{a\}$; $\mathit{Keys} := \mathit{Keys} \setminus \{a\}$
2: $\mathit{Weak} := \mathit{Weak} \setminus \{a\}$
3: for all $b$: $b \rightsquigarrow a$ s.t. $b \in \mathit{Stub}$ do
4:    if $b \in C$ then $\mathit{Delete}(b)$ end if
5:    if $b \in en(s)$ then
6:       if $b \rightsquigarrow_S a$ then $\mathit{Keys} := \mathit{Keys} \setminus \{b\}$ end if
7:       if $b \rightsquigarrow_W a$ then $\mathit{Weak} := \mathit{Weak} \setminus \{b\}$ end if
8:       if $b \notin \mathit{Weak} \wedge b \notin \mathit{Keys}$ then $\mathit{Delete}(b)$ end if
9:    end if
10:   if $b \in \Sigma \setminus en(s) \wedge \{x \mid b \rightsquigarrow_C x\} \cap \mathit{Stub} = \emptyset$ then
11:      $\mathit{Delete}(b)$
12:   end if
13: end for
```

We try each enabled action a in turn. If $\{a\}$ is a legitimate stubborn set, we return that. This is done using a predicate $\mathit{Single}(a)$, which checks the following conditions:

(a) $\{b \mid b \in en(s) \wedge a \rightsquigarrow_S b\} = \emptyset$
(b) for each $b \in \Sigma \setminus en(s)$, such that $a \rightsquigarrow_S b$, we have $\exists c \in C : \{d \mid b \rightsquigarrow_C c \rightsquigarrow_C d\} \cap \{x \mid a \not\rightsquigarrow_S x\} \subseteq \{a\}$.

That these conditions are sufficient results from the following: (a) makes sure that a is strongly independent with all other enabled actions, and (b) makes sure that, if b is an action that a is dependent with, b has a cause that cannot be satisfied without executing either a or some action that a is dependent with. One such action would need to be executed first, but this cannot happen without a, and thus $\{a\}$ is stubborn.

If either of these conditions fail, then we attempt to remove a deterministic a. (If a is not deterministic, we do not attempt to remove it as this could violate **PB**.) We start with the current candidate set by the recursive procedure $\mathit{Delete}(a)$. The procedure makes sure that the closure properties of stubborn sets are satisfied. It works as follows: after deleting an action (or condition) a from the set, we have possible violations of the closure properties. The deletion of a may cause a violation for any b such that $b \rightsquigarrow a$. (We write $b \rightsquigarrow a$ whenever $b \rightsquigarrow_X a$ for some $X \in \{S, W, C\}$) We check for such a violation, and if it happens, we also delete b.

If the removal of a would result in a void set, i.e., empty or without key actions, a set violating **PB**, or not containing the set U, the removal is cancelled. In such a case, the sets Stub, Keys, and Weak must be returned to what they were before the deletion of a. Otherwise we go with the new values.

Naturally we can optimize Delete, and stop the procedure if a probabilistic action or a member of U is deleted without the need to complete the run. Other optimizations include stopping the procedure when a member of U would be deleted. Once we are done, we return the enabled actions of the stubborn set, along with the key actions.

To show that this results in a stubborn set for the graph, we present the following reasoning. $\mathit{Delete}(a)$ removes a from the candidate set. The closure properties may be violated for b if $b \rightsquigarrow a$. Firstly, if $b \in C$, then b needs to be removed, as we know that b has no $\rightsquigarrow_W$-neighbours. Secondly, if $b \in en(s)$, we have to check that either all its weak neighbours are in the set or it is a key action. If neither is true, b needs to be deleted, otherwise its closure properties are intact. Thirdly, if $b \in \Sigma \setminus en(s)$, and all its causes have been removed, it must be removed as well. If not, its closure properties are still intact. Later we only need to check the existence of key actions and that **PB** is not violated.

B. Combining partial order reduction with probability computation

To check reachability properties, partial order reduction is usually implemented in depth-first search (DFS). In this section, we give a complete DFS algorithm for exploring reduced state space under unconditional fairness and computing maximum probability on-the-fly by SCC-based value iteration [17], [18]. Our algorithm is based on the Tarjan's algorithm [23], which is an efficient algorithm for identifying SCCs using DFS. Algorithm 3 combines partial order reduction with an optimised version of the Tarjan's algorithm proposed in [24], as well as on-the-fly reduction.

Algorithm 3 $por\_tarjan(s)$

```
1:  $s.index := index$; $s.lowlink := index$
2:  $index := index + 1$; $s.pdl := false$;
3:  if $s \in F$ then $s.p := 1$ else $s.p := 0$ end if
4:  $(Stub, Keys) := T(s, \emptyset)$; $s.sat := Stub$; $R := Stub$
5:  repeat
6:    while $Stub \neq \emptyset$ do
7:      pick $a$ from $Stub$; $Stub := Stub \setminus \{a\}$
8:      $\mu := \mathsf{Steps}(s, a)$
9:      for all $s'$ with $\mu(s') > 0$ do
10:       if $s'.index$ is undefined then
11:         $por\_tarjan(s')$
12:         if $s'.pdl$ is false then
13:           $s.lowlink := \min\{s.lowlink, s'.lowlink\}$
14:         else
15:           $R := R \setminus \{a\}$; break
16:         end if
17:       else
18:         if $s' \in stack$ then
19:           $s.lowlink := \min\{s.lowlink, s'.lowlink\}$
20:         else if $s'.pdl = true$ then
21:           $R := R \setminus \{a\}$; break
22:         end if
23:       end if
24:       if $a \in Keys$ then $s.sat := s.sat \cup s'.sat$
25:     end for
26:   end while
27:   if $en(s) \setminus s.sat \neq \emptyset$ then
28:     $(Stub, Keys) := T(s, en(s) \setminus s.sat)$;$R := R \cup Stub$
29:     $s.sat := s.sat \cup Stub$
30:   end if
31: until $Stub = \emptyset$
32: if $R = \emptyset$ then $s.pdl := true$
33: else
34:   if $s.lowlink = s.index$ then
35:     $scc := \{s\}$
36:     while $stack \neq \emptyset \wedge TOP(stack).index \geq s.index$ do
37:       $scc := scc \cup POP(stack)$
38:     end while
39:     if $scc$ is a partial deadlock then
40:       $s''.pdl := true$ for each $s'' \in scc$
41:     else
42:       compute $s''.p$ for each $s'' \in scc$
43:     end if
44:   else $PUSH(stack, s)$ end if
45: end if
```

In Algorithm 3, $stack$ is the stack for the depth-first search. TOP returns the element on top of $stack$, POP and $PUSH$ are the usual stack operations. Before the first call to the recursive function por_tarjan, $stack$ contains only the initial state $\overline{s}$ and the global variable $index$ is set to one. The attribute p of a state s is the probability in s, and pdl indicates if no FMEC can be reached from s. The process of action a is interrupted once we find it can lead to a partial deadlock with non-zero probability. Then a is removed from the set R, which initially represents the whole weak stubborn set. The role of $s.sat$ is to make sure the condition **S** for partial order reduction is satisfied: it denotes all the actions that have been executed after a path of key actions. When the actions of a stubborn set have been explored, if there are still "unsatisfied" actions left, we need to generate a new stubborn set such that they are included.

Identifying a partial deadlock can be done in two steps. (1) Identifying a bottom EC can be done by checking if there exists an action in the EC leading to states outside the EC. Note that an action that is not chosen by any fair adversaries cannot be counted as an action leaving the EC. (2) To check partial deadlock in a bottom EC, each state s in the EC is associated with a vector, one entry per process. If a process has an enabled action in s, then the corresponding entry is set to $true$. We perform an entry-wise disjunction operation on the vectors in the bottom EC. A partial deadlock is found if and only if an entry remains $false$ after the disjunction operation.

VI. Implementation and experiments

We implemented a prototype for the partial order reduction algorithm (Algorithm 3) in the probabilistic model checker PRISM [25]. The strong and weak independence relations are defined on the global state space, which we try to avoid to generate. Therefore, we compute over-approximations of these relations using the local state space of each component. In PRISM, each component has a set of local variables, which can be read in the guard of actions in other components. Therefore, it might be impossible to compute accurate local state spaces individually. Instead, we compute an over-approximation again, which contains the actual local state space, and may also include non-reachable local states. The over-approximated local state space is generated by removing all conjuncts containing non-local variables from the actions' guard, assuming each guard is specified in conjunction normal form.

The performance of our reduction technique can be demonstrated by a case study of the Zeroconf network configuration protocol [26]. Table I shows the number of states[2] and running time under unconditional fairness, with and without reduction respectively. These results clearly show the advantage of partial order reduction, even under the over-approximation of dependency relations. The time

[2]In this case study, the number of states under unconditional fairness is the same as that without fairness in both the original and reduced models, as the states in all partial deadlocks are explored.

Table I
EXPERIMENTAL RESULTS ON ZEROCONF.

Parameters	No reduction		With reduction	
(K)	States	Time (s)	States	Time (s)
2	13474	103.819	461	2.479
4	57960	1940.343	747	2.805
6	125697*	timeout	1033	3.582
8	163229*	timeout	1319	4.181
10	176316*	timeout	1605	4.929

*Computed using PRISM BDD engine

spent on computing these relations was compensated by the much smaller models after reduction.

VII. CONCLUSION

In this paper, we first proved that realisability of unconditional fairness in MDPs can be carried out on their non-probabilistic abstractions to enable fast check. Then we proved the maximum probability of reaching some bottom EC under unconditional fairness is preserved by the weak stubborn set T-reduction. We also presented data structures and the model checking algorithm for computing such probability through T-reduction.

There are several possible future directions. The first is to extend our approach to allow arbitrary target states, rather than just bottom FMECs. Secondly, we would like to optimise our initial implementation to improve the over-approximations of computing dependency relation and conduct more case studies to explore the potential of weak stubborn sets. Computing the maximum probability under strong/weak fairness constraints via partial order reduction has been discussed in [10]. It would be very interesting to compare our approach with theirs.

ACKNOWLEDGEMENTS

The first author was supported by the TEKES post-doctoral grant of 2010. The second and third authors are partly supported by EC FP 7 project CONNECT (IST 231167) and ERC Advanced Grant VERIWARE.

REFERENCES

[1] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume-guarantee verification for probabilistic systems," in *Proc. of TACAS'10*, vol. LNCS 6105. Springer, 2010, pp. 23–37.

[2] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Quantitative multi-objective verification for probabilistic systems," in *Proc. of TACAS'11*, vol. LNCS 6605. Springer, 2011, pp. 112–127.

[3] A. Valmari, "Stubborn sets for reduced state space generation," in *Proc. of the 10th International Conference on Application and Theory of Petri Nets*, vol. LNCS 483. Springer, 1989, pp. 491–515.

[4] P. Godefroid, "Using partial orders to improve automatic verification methods," in *Proc. of CAV'90*, vol. LNCS 531. Springer, 1990, pp. 176–185.

[5] D. A. Peled, "All from one, one for all: on model checking using representatives," in *Proc. of CAV'93*, vol. LNCS 697. Springer, 1993, pp. 409–423.

[6] K. McMillan and D. Probst, "A technique of state space search based on unfolding," *Formal Methods in System Design*, vol. 6, no. 1, 1995.

[7] C. Baier, M. Grösser, and F. Cieinski, "Partial order reduction for probabilistic systems," in *Proc. of QEST'04*. IEEE CS Press, 2004, pp. 230–239.

[8] C. Baier, P. D'Argenio, and M. Grösser, "Partial order reduction for probabilistic branching time," *ENTCS*, vol. 153, no. 2, pp. 97–116, 2006.

[9] P. R. D'Argenio and P. Niebert, "Partial order reduction on concurrent probabilistic programs," in *QEST'04*. IEEE CS Press, 2004, pp. 240–249.

[10] C. Baier, M. Größer, and F. Ciesinski, "Quantitative analysis under fairness constraints," in *Proc. of ATVA'09)*, vol. LNCS 5799. Springer, 2009, pp. 135–150.

[11] M. Größer and C. Baier, "Partial order reduction for markov decision processes: A survey," in *Proc. of FMCO'05*, vol. LNCS 4111. Springer, 2005, pp. 408–427.

[12] H. Hansen and X. Wang, "Compositional analysis for weak stubborn sets," in *Proc. of ACSD'11*. IEEE CS Press, 2011, to appear.

[13] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[14] A. Bianco and L. de Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Proc. of FSTTCS'95*, vol. LNCS 1026. Springer, 1995.

[15] A. Pnueli, "The temporal logic of programs," in *Proc. of FOCS'77*. IEEE CS Press, 1977, pp. 46–57.

[16] L. de Alfaro, "Formal verification of probabilistic systems," Ph.D. dissertation, Stanford University, 1997.

[17] F. Ciesinski, C. Baier, M. Größer, and J. Klein, "Reduction techniques for model checking Markov decision processes," in *Proc. of QEST'08*. IEEE CS Press, 2008, pp. 45–54.

[18] M. Kwiatkowska, D. Parker, and H. Qu, "Incremental quantitative verification for Markov decision processes," in *Proc. PDS'11*. IEEE CS Press, 2011, to appear.

[19] A. Valmari, "Stubborn set methods for process algebras," in *Proc. of POMIV'96*, ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 29, 1997, pp. 213–231.

[20] A. Valmari, "A stubborn attack on state explosion," *Formal Methods in System Design*, vol. 1, no. 1, pp. 297–322, 1992.

[21] S. Evangelista and C. Pajault, "Solving the ignoring problem for partial order reduction," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 2, pp. 155–170, 2010.

[22] A. Valmari, "The state explosion problem," in *Lectures on Petri Nets I: Basic Models*. Springer, 1998, vol. LNCS 1491, pp. 429–528.

[23] R. E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal of Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[24] E. Nuutila and E. Soisalon-soininen, "On finding the strongly connected components in a directed graph," *Information Processing Letters*, vol. 49, pp. 9–14, 1994.

[25] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *Proc. of TACAS'06*, vol. LNCS 3920. Springer, 2006.

[26] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston, "Performance analysis of probabilistic timed automata using digital clocks," *FMSD*, vol. 29, 2006.