POWVER

Technical Report 2019-01

Title: **Syntactic Partial Order Compression for Probabilistic Reachability**

Author: Gereon Fox, Daniel Stan, Holger Hermanns

Report Number: 2019-01

ERC Project: Power to the People. Verified.

ERC Project ID: 695614

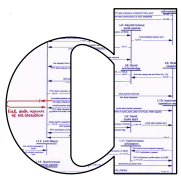
Funded Under: H2020-EU.1.1. – EXCELLENT SCIENCE

Host Institution: Universität des Saarlandes, Dependable Systems and Software
Saarland Informatics Campus

Published In: VMCAI 2019

Please cite this publication as follows:

Gereon Fox, Daniel Stan, Holger Hermanns.
Syntactic Partial Order Compression for Probabilistic Reachability.
Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-19, 2019, Proceedings. Lecture Notes in Computer Science 11388, 2019, ISBN 978-3-030-11245-5. 446-467.



POWER TO THE PEOPLE.
VERIFIED.
erc

Syntactic Partial Order Compression for Probabilistic Reachability

Gereon Fox[1 2(✉)], Daniel Stan[1], and Holger Hermanns[1]

[1] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
{fox,dstan,hermanns}@depend.uni-saarland.de
[2] Saarbrücken Graduate School of Computer Science, Saarbrücken, Germany

Abstract. The state space explosion problem is among the largest impediments to the performance of any model checker. Modelling languages for compositional systems contribute to this problem by placing each instruction of an instruction sequence onto a dedicated transition, giving concurrent processes opportunities to interleave after every instruction. Users wishing to avoid the excessive number of interleavings caused by this default can choose to explicitly declare instruction sequences as *atomic*, which however requires careful considerations regarding the impact this might have on the model as well as on the properties that are to be checked. We instead propose a preprocessing technique that automatically identifies instruction sequences that can safely be considered atomic. This is done in the context of concurrent variable-decorated Markov Decision Processes. Our approach is compatible with any off-the-shelf probabilistic model checker. We prove that our transformation preserves maximal reachability probabilities and present case studies to illustrate its usefulness.

Keywords: state space explosion · atomicity · partial order reduction · concurrency · interleavings · model checking

1 Introduction

Concurrency problems are notoriously difficult to study. One important reason is that activities local to a single component can interleave in arbitrary order with those of others, which is a root cause of the state-space explosion problem.

One of the earliest attempts to alleviate this problem is a feature of Holzmann's language Promela [16], exploited in the model checker Spin. Promela contains an `atomic` keyword, to be used by the modeller to group sequences of computations so that they are executed *atomically*, i.e. without the need to store intermediate states, which would otherwise contribute to state space explosion. Nowadays, model checkers like Uppaal allow the modeller to place entire C-code fragments as effects on single transitions [4]. As another example, Lnt, the modern language of the Cadp model checking framework [10], provides a dedicated "procedure" construct, the semantics of which ensure that the effect of each procedure body does not span more than a single transition. Apart from

interesting semantic questions (what if the body does not terminate, what if it cannot be made atomic?), these solutions burden the modeller with deciding which computations to group together.

Instead, partial order reduction techniques (POR) take over the task of identifying local computations that can be considered independent, and thus need to be explored in only *one* of their interleavings as opposed to *all* interleavings. POR techniques usually fall into one of two categories: *Static* POR [12,18,21], running mostly *before* state space exploration (hence the name "static"), computes so-called *persistent sets*, that are subsets of the transitions enabled in the different states of the system. Exploration then does not follow *all* the transitions enabled in a state, but only the ones in the persistent set of that state. *Dynamic* POR [1,9] on the other hand aims at materialising persistent sets *during* state space exploration, which promises to be more precise, as information about concrete executions is available and does not need to be overapproximated as much. POR has also been adapted to the probabilistic setting [2,3,7,8,11].

The present paper describes a technique that achieves reductions comparable to those of POR, but is actually geared towards automated placement of `atomic`'s in the model source code. Executed as a preprocessing step prior to the actual model checking, our approach analyses the model statically, in order to identify instruction sequences inside components that can safely be made `atomic`. This places the instructions on a single transition in the state space, instead of taking multiple steps that would multiply when interleaved.

We present our technique for variable-decorated, concurrent probabilistic models and probabilistic model checking. In this setting, the model checker usually constructs a Markov decision process that needs to be stored explicitly and very often is of prohibitive size. Symbolic or SAT-based approaches do not work particularly well here [15,20], basically because numerical computations need to be performed at the end.

We call our approach "syntactic partial order compression", because it has the same goal as partial order reduction, i.e. to rule out redundant interleavings. In general, POR techniques appear more capable of reducing the number of interleavings than the placement of `atomic`'s, because POR can freely choose between interleavings that involve several components of the system, whereas an `atomic` sequence is restricted to allowing only one component to make progress. We expect that the latter will often not suffice to preserve the property of interest, while the former still can, i.e. POR can still find opportunities for reduction when placing an `atomic` is just not sound anymore. In contrast to most POR techniques however, our approach does not require the model checker to cooperate (e.g. by pruning state space exploration) and thus can be used with any off-the-shelf tool. Furthermore, traditional POR techniques do not work well in our particular setting, rooted in the fact that probabilistic decisions (inside components) alternate with interleaving of components, and the resulting tree-shaped executions contain cross-dependencies that are difficult to account for [11]. Therefore, POR techniques have so far not delivered true success stories for probabilistic models. Recent work in this realm [8] did explore POR to

find and syntactically enforce particular representative interleavings — but the results of this approach do not translate to `atomic`'s, which we aim at.

Our main contribution is a perspicuous three-step approach that transforms a parallel composition of processes without ever materializing the product process: First, a *generation step* builds up a set of so-called *chains* for each process. A chain is a sequence of alternating nondeterministic and probabilistic decisions in one component, with the main requirement being that schedulers that never interrupt this sequence by any transitions from other components still achieve the same maximum reachability probability for the state-formula in question. We give a wellformedness condition for these chain sets, largely independent of how they are actually generated. Second, as a *filtering step* we describe an optimization problem aimed at finding a subset of the generated chains that admits as few interleavings as possible without changing said probability. Lastly, the *fusion step* turns the remaining chains into proper probabilistic transitions again, whereby sequences of decisions are compacted into atomic decisions, obtaining a syntactic representation of the transformed system.

2 Preliminaries

We start off by reviewing the basics of variable-decorated Markov decision processes in the style of MODEST [5,13], building upon [14], which we recommend for further details and elaborate discussions of the concepts.

Basic structures. Throughout the paper we assume a finite set Var of variables with countable domains $\mathrm{Dom}(x)$ for all $x \in Var$ and a set Exp of expressions over these variables, the detailed syntax and semantics of which are not of importance. We assume $Bxp \subseteq Exp$ to comprise boolean expressions ranging over $\{\mathrm{tt}, \mathrm{ff}\}$ and $Axp \subseteq Exp$ to denote arithmetic expressions ranging over $\mathbb{Q}$. The set $Val := Var \to \mathrm{Dom}(Var)$ of valuations contains all mappings of variables to values of their domains, i.e. for each $v \in Val$ we have $\forall x \in Var : v(x) \in \mathrm{Dom}(x)$.

For $x \in Var$ and $e \in Exp$ we call $x := e$ an *assignment*. Two assignments $x_1 := e_1$ and $x_2 := e_2$ are called *consistent* iff $x_1 \neq x_2$ or $[\![e_1]\!](v) = [\![e_2]\!](v)$ for all $v \in Val$, where $[\![e]\!](v) \in Val \to Val$ is the function that evaluates expression e given valuation v. The set Upd of all updates contains all sets of pairwise consistent assignments. Consistency allows the definition of a function $[\![u]\!] \in Val \to Val$ updating valuations after simultaneous evaluation of the assignments in u.

For a set S we call elements of $S \to Axp$ *symbolic probability distributions over* S and the values these functions return *weight expressions*.
Given a function f the set $\mathrm{Supp}(f)$ contains the arguments f is defined for.
For all sets S of the form $S = A \times B$ we define $S^\perp := (A \times B)^\perp := A^\perp \times B^\perp$ and if S is not a cartesian product we set $S^\perp := S \cup \{\perp\}$, where $\perp \notin S$ is a distinct placeholder element. Tuples $(\perp, \ldots, \perp)$ will be abbreviated with $\perp$. For sequences $s, t \in S^*$ we call s a *prefix* of t and write $s \sqsubseteq t$ iff $\exists t' : t = s \cdot t'$.

For $a, b \in \mathbb{N}$ we define $[a : b] := \{i \in \mathbb{N} \mid a \leq i \leq b\}$ and for vector variables we usually use the notation $\vec{v} = (v_1, \ldots, v_n)$.

MDPs with Variables. Our models are based on parallel compositions of Markov Decision Processes (MDP), enhanced with variables, guards and probability expressions [13,14]:

Definition 1 (VMDP). *A* Markov decision process with variables *(VMDP) is a tuple* (Loc, A, E, l_{init}) *where Loc is a finite set of* locations*,* $A \supseteq \{\tau\}$ *is a finite* alphabet*, including the* silent action τ*, and* $l_{init} \in Loc$ *is the* initial location*. Furthermore,* $E \in Loc \to \mathscr{P}(Bxp \times A \times (Upd \times Loc \to Axp))$ *maps each location to a set of* transitions*, which each consist of a* guard $g \in Bxp$*, a* label $\alpha \in A$ *and a symbolic probability distribution* $m \in Upd \times Loc \to Axp$ *that weighs pairs of updates and* target locations*. We denote the set of* all *transitions by* $\mathbb{T} := \bigcup_{l \in Loc} E(l)$*.*

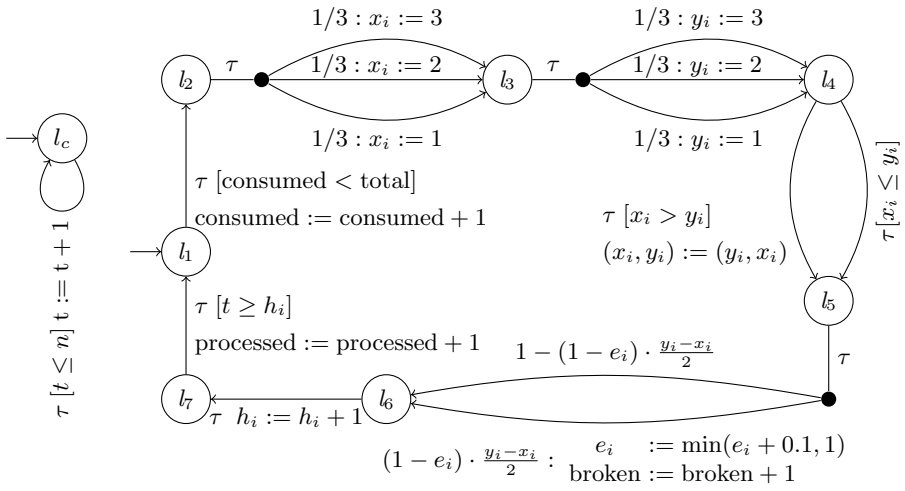


Fig. 1. A VMDP network modelling Example 1

As examples, consider the two processes depicted in Fig. 1: Locations are connected by transitions, which are decorated with their label and nontrivial guard. Distributions m with $|\mathrm{Supp}\,(m)\,| > 1$ are represented by splitting up transitions at $\bullet$ and labeling arcs with their probability expressions and updates. Each process transitions between locations: Given its current location l and a valuation v, a transition $(g, \alpha, m) \in E(l)$ is picked *nondeterministically*, under the condition that its guard expression is satisfied by v. Then, the weights assigned by m are evaluated over v, turning weight expressions into actual probabilities, based on which the new location l', along with an update u is chosen. The process then transitions to l' and updates v to v', according to u.

The fact that probability distributions are *symbolic* (i.e. probabilities depend on the current valuation of variables) contributes to the expressiveness of the

formalism, as we will see in Example 1. For the purposes of this paper, we require that all symbolic probability distributions m be well-formed, i.e. they satisfy

$$\forall v \in Val, (u,l) \in \mathrm{Supp}\,(m) : 0 \leq [\![m(u,l)]\!](v) \wedge \textstyle\sum_{(u,l)\in \mathrm{Supp}(m)} [\![m(u,l)]\!](v) = 1$$

Violations of this condition will be considered modelling errors in practise, causing MODEST to reject the model.

Networks of VMDPs. So far, we have not said a lot about the action labels $\alpha \in A$. They become significant when we consider *networks* of VMDPs. From now on we will deal with finite sets of VMDP's $\mathcal{C}_i = (Loc_i, A_i, E_i, l_{init_i})$:

Definition 2 (VMDP network semantics). *A* VMDP network *is a tuple* $\mathcal{N} = (\mathcal{C}, v_{init})$ *consisting of an initial valuation* $v_{init} \in Val$ *and a finite set* $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$ *of VMDP* components. *We call* $\mathbb{S} := (Loc_1 \times \ldots \times Loc_N) \times Val$ *the state space of* $\mathcal{N}$*, with* initial state $s_{init} := ((l_{init_1}, \ldots, l_{init_N}), v_{init}) \in \mathbb{S}$.
We write $(\vec{l}, v) \xrightarrow{\vec{g},\vec{\alpha},\vec{p},\vec{u}} (\vec{l'}, v')$ *iff there is* $\alpha \in A$ *and* $P \subseteq [1:N]$*, such that*

1. *If* $\alpha = \tau$*, then* $|P| = 1$*, otherwise* $P = \{i \mid \alpha \in A_i\}$.
2. *For all* $i \in P$ *we have* $\alpha_i = \alpha$ *and* $(g_i, \alpha, m) \in E_i(l_i)$ *for some* m *with* $m(u_i, l'_i) = p_i$*, such that* $[\![g_i]\!](v) = \mathrm{tt}$ *and* $[\![p_i]\!](v) > 0$.
3. *For all* $i \in [1:N] \setminus P$ *we have* $l_i = l'_i$ *and* $(g_i, \alpha_i, p_i, u_i) = \bot$
4. $v' = [\![\bigcup_{i\in P} u_i]\!](v)$

A (finite) path in $\mathcal{N}$ to s_n *is a sequence of the form*

$$\pi = s_{init} \xrightarrow{\vec{g}_0,\vec{\alpha}_0,\vec{p}_0,\vec{u}_0} s_1 \xrightarrow{\vec{g}_1,\vec{\alpha}_1,\vec{p}_1,\vec{u}_1} s_2 \ldots \xrightarrow{\vec{g}_{n-1},\vec{\alpha}_{n-1},\vec{p}_{n-1},\vec{u}_{n-1}} s_n \in \mathrm{Paths}$$

and we set $\mathrm{state}(\pi) := s_n$.

In a network, the executions of VMDPs interleave, i.e. the order in which they progress is chosen nondeterministically. Definition 2 however mandates common-alphabet synchronization in the style of Hoare [6], making steps in the network fall into one of two categories: Either a *silent* τ-step is made by one single process, or a different, *nonsilent* action label is chosen, requiring all processes with this label in their alphabet to participate and execute a step *synchronously*. In addition to communicating via synchronizing actions, processes also share all the variables in Var and can read and write them to exchange information. Similar to the wellformedness of symbolic probability distributions above we require that networks never lead to a state in which updates according to rule 4 would be inconsistent. Even after strengthening this condition to make it easily implementable (e.g. in MODEST), it remains a very light restriction and is relevant only for non-τ steps.

Example 1 (Factory). A factory receives a delivery of machine parts that are to be welded together in pairs. Part sizes are randomly distributed. A number of workers start out with a certain amount of experience with this type of task. Working in parallel, they grab pairs of parts and process them, which takes one hour per pair. The less experienced a worker is and the more two parts differ in size, the more likely it is that the worker makes a mistake and breaks the parts.

Fig. 1 depicts a model for Example 1: A global clock keeps track of time during an n-hour day, while each worker is represented by a dedicated process (indexed with i): Starting in l_1, the worker obtains a pair of parts, unless all have been obtained already. From l_2 to l_4 he measures the parts. From l_4 to l_5 measures are sorted: x_i should refer to the shorter part length. Based on the difference in part lengths and his current level of experience, the worker will then either succeed or fail in welding parts together, the latter of which will make him gain experience. Both success and failure take one hour (l_6 to l_7) which is why the worker can obtain another job only after the clock has progressed.

Note that at each location l_i a worker process might be nondeterministically interrupted by others. To avoid the enormous number of interleavings that these locations give rise to the modeller might want to manually compact subsequent transitions into one. This error-prone transformation would require duplicating parts of the model and thus lead to a less concise representation, justifying the desire for a tool that performs such a transformation "under the hood".

Probabilistic Reachability. Model analysis is often concerned with the question of reachability, i.e. whether there is a path of the system such that at some point of the path the system state satisfies a given condition. We denote such queries by $\Diamond\,\varphi$, where φ is a predicate over states. By $\mathrm{Paths}(\Diamond\,\varphi)$ we denote the set of finite paths containing a state s satisfying φ.

Since exploring paths of the system usually involves probabilistic choices it makes sense to not only ask for the existence of paths satisfying $\Diamond\,\varphi$, but for the probability $\mathrm{P}(\Diamond\,\varphi)$ with which one of these paths is chosen. These probabilities depend on the way that nondeterministic choices are resolved and thus can be defined only given a *scheduler* $\mathfrak{S} \in \mathit{Sched}$ which is a function $\mathbb{S} \to \mathbb{T}_1^{\perp} \times \ldots \times \mathbb{T}_N^{\perp}$, such that whenever $\mathfrak{S}((\vec{l}, v)) = (g_1, \alpha_1, m_1, \ldots, g_N, \alpha_N, m_N)$ we have

$$(\vec{l}, v) \xrightarrow{\begin{pmatrix} g_1 \\ \vdots \\ g_N \end{pmatrix}, \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{pmatrix}, \vec{p}, \vec{u}} (\vec{l'}, v')$$

for some $\vec{p}, \vec{u}, \vec{l'}, v'$ with $\forall i \in [1:N] : m_i(u_i, l_i'') = p_i$.
Intuitively, a scheduler thus resolves nondeterministic choices by selecting transitions that are enabled in the current network state. We define $\mathrm{Paths}(\mathfrak{S})$ to be the smallest set satisfying the following conditions:

1. $s_{init} \in \mathrm{Paths}(\mathfrak{S})$
2. If $s_{init} \xrightarrow{\vec{g}_0, \vec{\alpha}_0, \vec{p}_0, \vec{u}_0} s_1 \ldots \xrightarrow{\vec{g}_{n-1}, \vec{\alpha}_{n-1}, \vec{p}_{n-1}, \vec{u}_{n-1}} s_n \in \mathrm{Paths}(\mathfrak{S})$ and we have $\mathfrak{S}(s_n) = (\vec{g}, \vec{\alpha}, \vec{m})$, then for all steps $s_n \xrightarrow{\vec{g}, \vec{\alpha}, \vec{p}, \vec{u}} s_{n+1}$ we also have

$$s_{init} \xrightarrow{\vec{g}_0, \vec{\alpha}_0, \vec{p}_0, \vec{u}_0} s_1 \ldots \xrightarrow{\vec{g}_{n-1}, \vec{\alpha}_{n-1}, \vec{p}_{n-1}, \vec{u}_{n-1}} s_n \xrightarrow{\vec{g}, \vec{\alpha}, \vec{p}, \vec{u}} s_{n+1} \in \mathrm{Paths}(\mathfrak{S})$$

Remark 1. In full generality, schedulers base their decisions not only on the current state, but the complete sequence of visited states and transitions, and can randomize their decision. In our context this does however not add any power [19], which is why we restrict to *pure memoryless* schedulers.

This is what we need to properly define reachability probabilities:
For $\pi = (\vec{l}_0, v_0) \xrightarrow{\vec{g}_0,\vec{\alpha}_0,\vec{p}_0,\vec{u}_0} (\vec{l}_1, v_1) \dots \xrightarrow{\vec{g}_{n-1},\vec{\alpha}_{n-1},\vec{p}_{n-1},\vec{u}_{n-1}} (\vec{l}_n, v_n) \in \text{Paths}$:

$$\mathrm{P}(\pi) := \prod_{0 \le i < n} \prod_{1 \le j \le N} [\![p_{i,j}]\!](v_i) \quad \text{and} \quad \mathrm{P}^{\mathfrak{S}}(\Diamond\, \varphi) := \sum_{\pi \in \mathrm{Paths}(\mathfrak{S}) \cap \mathrm{Paths}(\Diamond\, \varphi)} P(\pi)$$

$\mathrm{P}(\pi)$ can be seen as a short-hand notation for the measure value $\mathrm{P}(\mathrm{Cyl}(\pi))$ of the cylinder $\mathrm{Cyl}(\pi)$ composed of all maximal paths prefixed by π. It is easy to see that this definition meets Carathodory's extension theorem so that P defines a unique *measure* over the σ-algebra generated by the finite paths. In particular, the set $\max_{\sqsubseteq}(\mathrm{Paths}(\mathfrak{S}))$ of (possibly infinite) prefix-maximal paths is a measurable set and $\mathrm{Paths}(\Diamond\, \varphi)$ identified as $\mathrm{Cyl}(\mathrm{Paths}(\Diamond\, \varphi))$ is also measurable. Moreover, $\mathrm{P}^{\mathfrak{S}}(\cdot) = \mathrm{P}(\max_{\sqsubseteq}(\mathrm{Paths}(\mathfrak{S})) \cap \cdot)$ is a *probability* measure.
In this paper we will focus on *quantitative reachability properties*:

$$\mathrm{P}_{\max}(\Diamond\, \varphi) := \max_{\mathfrak{S} \in Sched} \mathrm{P}^{\mathfrak{S}}(\Diamond\, \varphi) \qquad \text{or} \qquad \mathrm{P}_{\min}(\Diamond\, \varphi) := \min_{\mathfrak{S} \in Sched} \mathrm{P}^{\mathfrak{S}}(\Diamond\, \varphi)$$

Example 2. For the scenario modelled in Fig. 1 interesting queries include the probability $\mathrm{P}_{\max}(\Diamond(\text{processed} = \text{total} \wedge \mathrm{t} \le b))$ of finishing work within a certain time horizon and the probability $\mathrm{P}_{\max}(\Diamond(\text{processed} = \text{total} \wedge \text{broken} = 0))$ of not breaking any parts. Since model checkers are able to output the schedulers that achieve extremal probabilities, these queries correspond to finding strategies of how to distribute work packages among the workers.

Chains. The key data structure of our approach is the concept of a *chain*:

Definition 3 (Branches, links and chains). *For a transition $t = (g, \alpha, m)$ we denote the* branches of t *by* $\mathrm{Br}(t) := \{(u, l', p) \mid m(u, l') = p\}$. *Each branch $b \in \mathrm{Br}(t)$ corresponds to a* link $t \cdot b$ *and we denote the set of all links in the network by $\mathbb{L}$. Two links $t_1 \cdot (u, l', p)$, $t_2 \cdot b_2$ are called consecutive iff $t_2 \in E(l')$. A* chain *is a finite sequence $t_0 \cdot b_0 \cdot t_1 \cdot b_1 \cdots t_n \cdot b_n$ of consecutive links.*

Example 3. Consider the arrows from l_2 to l_3 in Fig. 1: Together they depict a transition that we refer to as t. The three segments between $\bullet$ and l_3 represent the branches of t. Every route from l_2 to l_3 depicts a link. A link itself is already a chain. But also any finite route in Fig. 1 that starts at a location and ends at a location is a chain, as it is a sequence of links.

If $|\mathrm{Br}(t)| = 1$ we call t *Dirac* and write $t \in T_{\mathrm{nd}}$, omitting the $\bullet$ in the graphical representation. We call a location l *pure* iff $l \in Loc_p := \{l \in Loc \mid |E(l)| \le 1\}$.

The goal of our approach is to identify certain chains as "uninterruptible", i.e. to establish that while a process is inside such a chain, one can refrain from switching control to a different process, without losing behavior relevant for the reachability property in question. By collapsing these chains into single transitions one obtains a new network that thus admits fewer interleavings than the original one, while preserving the property.

Mobility. To establish chains as "uninterruptible" we use the concept of mobility:

Definition 4 (Mobility). *Let $i, j \in [1 : N]$ with $i \neq j$ and $t_1 \in E_i(l_1), t_2 \in E_j(l_2)$ with $b_1 \in \mathrm{Br}(t_1), b_2 \in \mathrm{Br}(t_2)$. We say that $t_1 \cdot b_1$ is* forward-commutative with *$t_2 \cdot b_2$ iff for all states s and all step sequences $s \xrightarrow{\vec{g}_1,\vec{\alpha}_1,\vec{p}_1,\vec{u}_1} s_1' \xrightarrow{\vec{g}_2,\vec{\alpha}_2,\vec{p}_2,\vec{u}_2} s''$ that involve $t_1 \cdot b_1$ followed by $t_2 \cdot b_2$ (i.e. projecting the first step on process i gives $t_1 \cdot b_1$ and projecting the second step on process j gives $t_2 \cdot b_2$) we also have $s \xrightarrow{\vec{g}_2,\vec{\alpha}_2,\vec{p}_2,\vec{u}_2} s_2' \xrightarrow{\vec{g}_1,\vec{\alpha}_1,\vec{p}_1,\vec{u}_1} s''$ for some s_2' and*

$$\mathrm{P}\left(s \xrightarrow{\vec{g}_1,\vec{\alpha}_1,\vec{p}_1,\vec{u}_1} s_1' \xrightarrow{\vec{g}_2,\vec{\alpha}_2,\vec{p}_2,\vec{u}_2} s''\right) = \mathrm{P}\left(s \xrightarrow{\vec{g}_2,\vec{\alpha}_2,\vec{p}_2,\vec{u}_2} s_2' \xrightarrow{\vec{g}_1,\vec{\alpha}_1,\vec{p}_1,\vec{u}_1} s''\right).$$

If $s \not\models \varphi \ \wedge \ s_1' \models \varphi \ \wedge \ s'' \not\models \varphi$ we also require $s_2' \models \varphi$. Backward-commutativity *is defined by swapping* 1 *and* 2 *above. $t_1 \cdot b_1$ is called* mobile *iff it is forward-commutative and backward-commutative with* all *links from other processes.*

Mobility of links is what allows us to reorder paths such that the links of a chain appear as one contiguous sub-path in the proofs of Section 4. Since it ensures that we neither lose behavior, nor miss states that satisfy φ, we can be sure to preserve maximal reachability probabilities.

Remark 2. Definition 4 is related to the notion of independence usually found in POR literature [1,3,17]: Intuitively, two transitions t_1, t_2 are independent if for any *state* where they are both enabled, both executions $t_1 \cdot t_2$ and $t_2 \cdot t_1$ can be performed and lead to the same *state* distribution. The difference is that while mobility is defined on the syntactic level of links, independence talks about steps between concrete *states* (i.e. *vectors* of locations, paired with valuations).

3 Algorithm

Our syntactic partial order compression approach comprises three steps: In Step A we identify a set of candidate chains that are sufficient for spanning the relevant behavior. Since some of these chains are redundant, Step B is then concerned with selecting a subset of chains that eliminates a maximal number of interleavings. Finally Step C compiles the remaining chains into a new VMDP network. We assume a given input VMDP network $\mathcal{N}$ to start with.

Step A: Chain Generation

The objective of this step is to collect a finite set C of chains that can be turned into atomic transitions without altering probabilistic reachability. C should satisfy the following requirements:

Definition 5 (Chain set validity). *Given a property of the form $\mathrm{P}_{\max}(\Diamond\, \varphi)$, a chain set C is called* valid *iff it satisfies the following conditions:*

1. *For each chain $c = t_1 \cdot b_1 \cdots t_n \cdot b_n \in C$ we have:*

(a) *c does not contain any link more than once.*
(b) *If $n > 1$, c comprises τ-actions only and all its links are mobile.*
(c) *There is $k \in [1:n]$, s.t. $b_1, \ldots, b_{k-1}$ are Dirac and $l_k, \ldots, l_n \in Loc_p$.*
(d) *For each state s with $s \not\models \varphi$ and every path segment*

$$s \xrightarrow{\vec{g}_0,\vec{\alpha}_0,\vec{p}_0,\vec{u}_0} s_1 \xrightarrow{\vec{g}_1,\vec{\alpha}_1,\vec{p}_1,\vec{u}_1} s_2 \ldots \xrightarrow{\vec{g}_{n-1},\vec{\alpha}_{n-1},\vec{p}_{n-1},\vec{u}_{|c|-1}} s_{|c|}$$

that involves c as a sequence of steps in one of the processes of $\mathcal{N}$, we have $s_{|c|} \not\models \varphi \to \forall i \in [1:|c|-1] : s_i \not\models \varphi$.

2. *For each process $\mathcal{C}_i$ and each sequence λ of consecutive links originating in l_{init_i}, there is $\gamma \in C^*$, s.t. $\lambda \sqsubseteq \gamma$.*
3. *For each chain $t_1 \cdot b_1 \cdots t_k \cdot b_k \cdot t_{k+1} \cdot b_{k+1} \cdots t_n \cdot b_n \in C$ and $b'_k \in \mathrm{Br}(t_k) \setminus \{b_k\}$, there is $t_1 \cdot b_1 \cdots t_k \cdot b'_k \cdot t'_1 \cdot b'_1 \cdots t'_m \cdot b'_m \in C$ for some $t'_1, b'_1, \ldots t'_m, b'_m$.*

Condition 1a ensures acyclicity of chains. Condition 1b allows us to reorder paths of $\mathcal{N}$ such that chain c emerges as one contiguous path segment, see Section 4. Condition 1c rules out chains that contain a probabilistic choice before a nondeterministic one. This is necessary because chains will eventually be turned into atomic transitions again, that according to Definition 2 need to be picked nondeterministically *before* their probabilistic choices are made, thus not allowing schedulers to base their nondeterministic decision on the outcome of the probabilistic choice, which is not yet available. This condition echoes the need to add additional conditions to Peled's ample set approach [18] to preserve all relevant schedulers in probabilistic partial order reduction [2,3,7]. Condition 1d rules out chains during the execution of which φ is briefly satisfied in some intermediate state, without being also satisfied upon entering or leaving the chain. This prevents such intermediate states from being lost when chains are converted to atomic links. Condition 2 makes sure that control flow inside components is preserved, i.e. all paths through the locations of a component can still be realized by sequences of chains. Together with Condition 3, that preserves the probabilistic branching structure of components, it ensures that we can produce a syntactically well-formed result in Step C. Notice that all links bearing a nonsilent label will only be represented in C as chains of length 1.

Example 4. In Fig. 1 the only link in the clock process, as well as the links from l_1 to l_2 and from l_7 to l_1 in the worker constitute chains of length 1. The 9 paths from l_2 to l_4 constitute chains of length 2. The 4 paths from l_4 to l_7 are chains of length 3. This chain set satisfies the conditions of Definition 5. Condition 1c rules out chains running through l_4.

We give one possibility of generating a set C according to the above conditions in Algorithm 1: It basically performs a depth-first search through every component, extending chain prefixes as long as possible and starting new empty prefixes at locations where a chain ended. The latter already ensures condition 2.

The predicate $\mathrm{valid}(c, t \cdot b)$ is implemented such that the concatenation $c \cdot t \cdot b$ satisfies conditions 1b, 1c, 1d and does not contain more than one link with the same source location, to satisfy 1a and ensure termination. Conditions 1a and 1c

Algorithm 1 Chain generation

```
function CollectChains(component C_i)
    C ← ∅                                              ▷ Collected chains
    T ← {l_init} ; V ← ∅                               ▷ Locations to visit, locations visited
    function Continue(prefix,l)
        terminal ← tt                                  ▷ Is |E(l)| = 0 ?
        total ← tt                                     ▷ Was prefix continued for all outgoing links of l?
        for all t ∈ E(l), b = (u, l', p) ∈ Br(t) do
            terminal ← ff
            if valid(prefix, t, b) then Continue(prefix · t · b, l')
            else total ← ff
        if prefix ≠ ϵ ∧ (terminal ∨ ¬total) then       ▷ Yield prefix as chain?
            C ← C ∪ {prefix}
            T ← T ∪ ({l} \ V)
    while T ≠ ∅ do
        l ← pop(T); T ← T \ {l}
        V ← V ∪ {l}
        Continue(ϵ, l)
    return C
```

are local syntactic checks. Condition 1b can be conservatively overapproximated in many ways. Our implementation considers both orders in which two links can be combined and performs syntactic checks that imply the necessary conditions in Definition 4. Similarly, we overapproximate Condition 1d, by ensuring that chains contain at most one link that writes to variables occuring in φ.

A careful look at Condition 3 reveals that it is actually not ensured by Algorithm 1: For example, there might be a chain c covering a sequence of transitions, as depicted in Fig. 2. Since c contains the link $t \cdot b$, the condition requires a counterpart c' that agrees with c up to l and then contains the probabilistic alternative $t \cdot b'$. This counterpart is missing in Fig. 2. However, since Algorithm 1 starts new chains at exactly those locations where previously generated chains end and because it satisfies Condition 2, we know that chains c^1 and c^2 as in the figure must have been generated, which also entails the generation of c^3. Thus there is at least a subset of C that does preserve all conditions including 3, which is sufficient as an input for the next step.

Step B: Chain Filtering

The output of Step A is a chain set C that satisfies Definition 5, except for Condition 3. To enforce the latter we need to select a subset of C that satisfies it, a problem that can always be solved as we have argued in the previous section. Since we strive for the selection of a subset that reduces the number of interleavings as far as possible, we are dealing with an optimization problem that we cast as a $\{0, 1\}$-weighted MAX-SAT instance.

However, Condition 3 is not the only reason for optimizing a subset of C: A simple iteration over the syntax as in Algorithm 1 will generate redundant

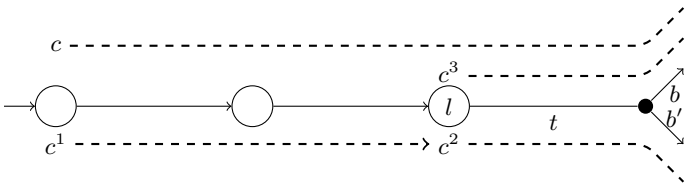


Fig. 2. c violates Condition 3, because $\neg\,\mathrm{valid}(c^1, t \cdot b')$. Algorithm 1 thus generated c^1 as a chain, which put l into its agenda T and caused generation of c^2 and c^3.

chains in C, because it has to start new chains in all directions at the end of *every* chain generated so far, unless sophisticated bookkeeping keeps track of the path-segments already covered, complicating step A significantly. As an example see Fig. 2, where c is subsumed by $c^1 \cdot c^3$. Experiments have shown that not eliminating these redundancies by optimization can not only prevent reduction of the number of interleavings but even increase it, because there are more nondeterministic choices to enumerate.

Constraints. We need to preserve the conditions of Definition 5. Condition 1 is trivially satisfied for all subsets of C and thus does not need to be encoded. Condition 3 can be encoded easily, because for every chain c the number of chains c' that need to be selected as a consequence of selecting c according to the condition is usually rather small.

Condition 2 does need to be encoded, because discarding chains in C might make certain control flow paths unrealizable. For this purpose we propose Algorithm 2. This algorithm attaches so-called *bundles* to the links of each process. A bundle B attached to a link $t \cdot b$ represents a set of finite paths that all start in the initial location and end with $t \cdot b$. A solution to the optimization problem must select a set S of chains that all contain $t \cdot b$. The chains selected in this way are the ones remaining at the end of step B.

The algorithm starts by attaching bundles to the initial links. For every newly created bundle attached to a link, successor bundles for all successor links are created, which is why branches and loops lead to links being attached more tha none bundle. The constraints generated for bundles make sure that the sets S for subsequent bundles "fit together": A chain c can be selected for a bundle attached to $t \cdot b$ only if c is select for the predecessor bundle as well ("continuation"), or if $t \cdot b$ is the very first link in c and the predecessor bundle selected a chain ending in the link that it is attached to ("initiation"). Termination is guaranteed because for every link there is only a finite number of chains that contain it, bounding the number of bundles attached to it. Correctness of Algorithm 2 can be proven by structural induction over paths.

Objective. Intuitively, the goal of Step B is to minimize the number of times a process can be "interrupted" by the scheduler, giving other processes opportunity to interfere and thus burdening the model checker with a great number of

Algorithm 2 Generation of constraints that encode chain set invariant 2

for all $t \in \mathbb{T}$, $b \in \mathrm{Br}(t)$ **do** $\mathrm{bundles}(t \cdot b) \leftarrow \emptyset$
for all $t \in E(l_{init})$, $b \in \mathrm{Br}(t)$ **do**
  $B \leftarrow \{c \in C \mid \exists c' : c = t \cdot b \cdot c'\}$
  $\mathrm{bundles}(t \cdot b) \leftarrow \mathrm{bundles}(t \cdot b) \cup \{B\}$
  AddConstraint($\bigvee_{c \in B} u_{B,c}$) ▷ At least one chain must be used to cover $t \cdot b$
$k \leftarrow 0$
repeat
  for all consecutive links $t_i \cdot b_i$, $t \cdot b$ and $B \in \mathrm{bundles}(t_i \cdot b_i)$ **do**
    $B_T \leftarrow \{c \in B \mid \exists c' : c = c' \cdot t_i \cdot b_i\}$ ▷ Chains terminated with $t_i \cdot b_i$
    $B'_C \leftarrow \{c \in B \mid \exists \pi, \sigma : c = \pi \cdot t_i \cdot b_i \cdot t \cdot b \cdot \sigma\}$ ▷ Chains continued from $t_i \cdot b_i$
    $B'_I \leftarrow \{c \in C \mid \exists c' : c = t \cdot b \cdot c'\}$ ▷ Chains initiated with $t \cdot b$
    $B' \leftarrow B'_C \cup B'_I$
    $\mathrm{bundles}(t \cdot b) \leftarrow \mathrm{bundles}(t \cdot b) \cup \{B'\}$
    AddConstraint($I_k \vee C_k$) ▷ Initiate new chain, or Continue previous one
    AddConstraint($I_k \leftrightarrow \bigvee_{c \in B'_I} u_{B',c}$)
    AddConstraint($C_k \leftrightarrow \bigvee_{c \in B'_C} u_{B',c}$)
    AddConstraint($I_k \rightarrow \bigvee_{c \in B_T} u_{B,c}$)
    AddConstraint($\forall c \in B'_C : u_{B',c} \rightarrow u_{B,c}$)
    $k{+}{+}$
until fixpoint ▷ Exists because there can be only finitely many B

interleavings. More technically this means that when given the choice of how to "emulate" a path $\lambda \in \mathbb{L}^*$ by a concatenation $\gamma \in C^*$ of chains, we should attempt to pick γ such that it contains a minimal number of positions in which one chain ends and another begins, because these are exactly the positions where a scheduler might choose to interrupt a process. To encode this intuition, we chose the summation over all variables I_k generated by Algorithm 2 as the primary objective to be minimized. This choice is justified by the fact that setting a variable I_k to true basically means that there is a set of path prefixes that can only be continued by leaving one chain and entering into a new one. Of course counting interruptions with this heuristic is biased by being performed on the syntactic representation of the model, but we expect it to lead to reasonable results as long as the control flow of processes is not too unusual. As a secondary objective we minimize the total number of selected chains, hoping to obtain a compact syntactic representation of the transformation result.

Step C: Chain Fusion

Having obtained a chain set C that satisfies Definition 5, all that remains is to compile chains to a proper VMDP network again, for which we give Algorithm 3. Starting at the initial location, the algorithm follows chains to reachable locations, building up the transformed network along the way.

ndchoices(l) contains subsets of the chains starting in location l. It reflects the possible control flow choices a scheduler can make inside a component: A

Algorithm 3 Compilation of a set of chains to a VMDP network

```
function COMPILE(component (Loc, A, E, l_init), chain set C)
    T ← ∅                                              ▷ Locations to visit
    L ← ∅ ; E′ ← ∅                                     ▷ New locations, New transitions
    function MAP(location l)
        l′ ← L[l]
        if l′ = ⊥ then
            l′ = NEW LOCATION()
            T ← T ∪ {l′}
        return l′
    l′_init ← MAP(l_init)
    while T ≠ ∅ do
        l ← pop(T); T ← T \ {l}
        l′ ← MAP(l)
        E(l′) ← { FUSE(MAP, cs) | cs ∈ ndchoices(l)}
    return ({L[l] | l ∈ Loc}, A, E, l′_init)
```

subset cs is in ndchoices(l) iff there is a mapping of locations to transitions, such that cs is precisely the set of all chains starting in l that contain only selected transitions:

$$\begin{aligned} cs(S) &:= \quad \{t_0 \cdot b_0 \cdot \ldots t_n \cdot b_n \in C \mid t_0 \in E(l) \wedge \forall i \in [0:n] : t_i \in S\} \\ \text{ndchoices}(l) &:= \quad \{cs(S) \mid S \subseteq \mathbb{T} \wedge \forall l \in Loc : |S \cap E(l)| = min(|E(l)|, 1)\} \end{aligned}$$

FUSE(MAP, cs) converts each such subset cs into a transition of the new model: Every $c \in cs$ is traversed, maintaining a record of the updates seen so far, in order to substitute variables in guards, weight expressions and subsequent updates. In the case of guards, the results of the substitutions are then conjoined, while in the case of weight expressions they are multiplied. After all chains have been traversed in this fasion, for each chain there is thus one guard g, one combined update u and one weight expression p. The guard of the resulting transition is the conjunction of all g. Weight expressions need to be normalized before they can be combined with updates and target locations to form the branches of the new transition. Since all nonatomic chains contain only τ-labels synchronization does not complicate FUSE.

Remark 3. Instead of compiling a new model in this fashion, we could as well resort to the textual representation of the original model (if any), in order to declare instruction sequences along chains `atomic`. This, however, can turn out to be even more cumbersome than the very technical computation in Algorithm 3, depending on the generated chains and the syntactic structure of the modelling language. As an example, consider a piece of code like

```
a; b; if (c) then d else e; f; g;
```

If the sequence `a b d f g` can safely be made atomic, but `a b e f g` cannot, it is not clear what parts of the code to enclose in an `atomic` block. Chains that span across loops can be even harder to represent in syntax.

4 Correctness

In this section, we prove that the transformation given in the previous section is sound, i.e. that maximal reachability probabilities are preserved.

In the following, mathematical objects indexed by r will refer to the reduced network. Our goal in this section is to relate $\mathrm{P}^{\mathfrak{S}}(\Diamond\varphi)$ and $\mathrm{P}^{\mathfrak{S}_r}(\Diamond\varphi)$, respectively for $\mathfrak{S} \in \mathit{Sched}$ and $\mathfrak{S}_r \in \mathit{Sched}_r$. Note that despite its similar notation, $\Diamond\varphi$ refers to different sets of paths $\mathrm{Paths}(\Diamond\varphi)$ and $\mathrm{Paths}_r(\Diamond\varphi)$. The reduced system is composed of transitions built from the filtered chain set $C_f \subseteq C$.

For a transition $t \in \mathbb{T}_i$, we set $\downarrow t := (\bot, \ldots, \bot, t, \bot \ldots \bot) \in (\mathbb{T}_1^\bot \times \ldots \times \mathbb{T}_N^\bot)^T$ and define $\downarrow b$ equivalently for branches b. We extend the notation to links and sets or sequences thereof. For any path π we denote similarly by $\downarrow\pi$ the underlying sequence of vectors of transitions and branches.

Finally, we extend this translation to the whole set C_f:

$$\begin{aligned}\Downarrow C_f = &\{\downarrow c \mid c \in C_f \text{ with } \tau\text{-actions}\} \uplus \\ &\{(t_1, \ldots) \cdot (b_1, \ldots) \mid \alpha \in A, (t_i, b_i) \in C_f \text{ if } \alpha \in A_i, (t_i, b_i) = \bot \text{ otherwise}\}\end{aligned}$$

Note that special care has to be taken for chains of length 1 that represent non-silent actions, that have to be "synchronized" with each other in $\Downarrow C_f$.

Without loss of generality, and for the sake of clarity, we assume in this section that all schedulers are *history*-dependent. This means, that a scheduler $\mathfrak{S}$ is defined for a whole path π instead of only its last state. This assumption does not interfere with the optimal value of the reachability property, while at the same time allows us to reason and manipulate the sub-tree of a set of paths, without interfering with previous transitions taken by the scheduler.

Soundness. In order to establish correctness of the transformation, we want to simulate any run of the reduced network in the original one, which basically means that the transformation introduced no new behavior. However, we can already notice that our reduction may introduce new deadlock situations, that are fortunately harmless for the reachability properties we are considering:

Example 5. Composing the process on the left of Fig. 3 with itself (under synchronization over actions a and b) yields a system that is free of deadlocks. Reducing the depicted chain in both components results in a composition of the process on the right with itself. Here, deadlock is possible. Fortunately, because we are interested in reachability properties, this newly introduced deadlock doesn't influence the value of $\mathrm{P}_{\max}(\Diamond(\mathrm{x} = 1))$.

Lemma 1 (Simulation of the reduced system). *Let $\mathfrak{S}_r \in \mathit{Sched}_r$ a scheduler of the reduced system. We can construct $\mathfrak{S} \in \mathit{Sched}$ for the original system such that $\mathrm{P}^{\mathfrak{S}}(\Diamond\varphi) = \mathrm{P}^{\mathfrak{S}_r}(\Diamond\varphi)$.*

Proof. Basically, $\mathfrak{S}$ has to *follow* the moves made by $\mathfrak{S}_r$. For this purpose, we define $\mathfrak{S}(\pi)$ for any $\pi \in \mathrm{Paths}$ by induction on $|\pi|$. The following invariant will hold during the induction:

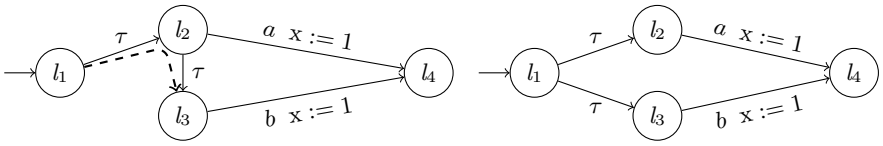


Fig. 3. Reducing the chain over $l_1 l_2 l_3$ turns the process on the left into the process on the right.

Invariant 1 *If* $\mathrm{P}^{\mathfrak{S}}(\pi) > 0$ *then* $\pi = \pi' \cdot \rho$ *for some* π',ρ *and there are* $\pi_r \in \mathrm{Paths}_r$, cs *returned from* ndchoices *and* $c \in \mathrm{cs} \subseteq C_f$ *, s.t.*

$$\downarrow\pi' \in \Downarrow {C_f}^* \ \wedge\ \downarrow\rho \sqsubseteq \downarrow c \ \wedge\ |\rho| < |c| \ \wedge\ \mathrm{state}(\pi_r) = \mathrm{state}(\pi') \\ \wedge\ \mathfrak{S}(\pi_r) = t_{\mathrm{cs}} \ \wedge\ \mathrm{P}^{\mathfrak{S}}(\pi') = \mathrm{P}^{\mathfrak{S}_r}(\pi_r)$$

where t_{cs} *denotes the transition resulting from* FUSE(MAP, *cs).*

Intuitively, π' represents the already chain-reconstructed path in $\mathcal{N}$ matched by π_r in $\mathcal{N}_r$, while ρ is the next prefix of a valid chain to be added to π. The empty path starting in the initial state satisfies the invariant. Let us consider π satisfying the invariant.

- If $\mathrm{P}^{\mathfrak{S}}(\pi) = 0$, we define $\mathfrak{S}(\pi)$ arbitrarily, and easily check that any successor of π still satisfies the invariant, since it is also not reachable.
  From now on, we assume that $\mathrm{P}^{\mathfrak{S}}(\pi) > 0$ and define π', ρ, π_r as above.
- If $|\rho| = 0$ and $\mathfrak{S}_r(\pi_r)$ is a transition of the original system, we simply define $\mathfrak{S}(\pi) = \mathfrak{S}_r(\pi_r)$ and check that the invariant is still satisfied by all successors (we can add the same branch to π,π' and π_r).
- Otherwise, $\downarrow\rho \sqsubseteq \downarrow c$ which means a chain has to be continued. We define $\mathfrak{S}(\pi) = (\downarrow c)[|\downarrow\rho| + 2] = \vec{t}$ as the next transition vector to be played. For any branch $b \in \mathrm{Br}(t)$, there exists $c' \in \mathrm{cs}$ such that $\downarrow\rho \cdot \downarrow(t, b) \sqsubseteq \downarrow c'$. Moreover if the equality holds, the chain is over. In this case, in order to preserve the invariant, we define $\rho' = \epsilon$, and due to the way c' is compiled into a simple link in $\mathcal{N}_r$ by Algorithm 3, the probability stays equal to probability of the sequence of individual links in c' from $\mathrm{state}(\pi)$ in $\mathcal{N}$.

Completeness. We now establish that our transformation is *complete*, i.e. that any run in the initial network can still be reproduced in the reduced system, up to interleavings that do not contribute to the reachability property in question. This direction is more challenging, and essentially relies on the conditions in Definition 5. Most of all, the key ingredient is the ability for a scheduler to postpone any Dirac transition after a probabilistic one:

Lemma 2. *Let* $\mathfrak{S} \in$ *Sched,* $t \cdot b$ *Dirac and mobile and* $\pi \in \mathrm{Paths}$*. Let* $s \in \mathbb{S}$ *such that* $\pi \xrightarrow{\downarrow t, \downarrow b} s$ *and* $\vec{t'} := \mathfrak{S}(\pi \xrightarrow{\downarrow t, \downarrow b} s)$*. We can rewrite* $\mathfrak{S}$ *into* $\mathfrak{S}'$ *such that:*

- *For all* $\pi' \in \mathrm{Paths}$ *with* $\pi \not\sqsubseteq \pi'$*,* $\mathfrak{S}(\pi') = \mathfrak{S}'(\pi')$ *and* $\mathrm{P}^{\mathfrak{S}}(\pi') = \mathrm{P}^{\mathfrak{S}'}(\pi')$

– *For all* π'*, and for all* $\vec{b'}, s'$, $\mathfrak{S}'(\pi \xrightarrow{\vec{t'},\vec{b'}} s' \xrightarrow{\downarrow t,\downarrow b} \pi') = \mathfrak{S}(\pi \xrightarrow{\downarrow t,\downarrow b} s \xrightarrow{\vec{t'},\vec{b'}} \pi')$ *and* $\mathrm{P}^{\mathfrak{S}}(\pi \xrightarrow{\downarrow t,\downarrow b} s \xrightarrow{\vec{t'},\vec{b'}} \pi') = \sum_{s'' \in \mathbb{S}} \mathrm{P}^{\mathfrak{S}'}(\pi \xrightarrow{\vec{t'},\vec{b'}} s'' \xrightarrow{\downarrow t,\downarrow b} \pi')$.

Proof. Intuitively, the state is already determined to be s after $\downarrow t$, so the scheduler can postpone this transition and play the (possibly) probabilistic one first, thanks to mobility.

Lemma 3. *Let* $\mathfrak{S} \in$ *Sched. We can build* $\tilde{\mathfrak{S}} \in$ *Sched such that* $\mathrm{P}^{\mathfrak{S}}(\Diamond\varphi) \leq \mathrm{P}^{\tilde{\mathfrak{S}}}(\Diamond\varphi)$ *and such that* $\forall \pi : \mathrm{P}^{\tilde{\mathfrak{S}}}(\pi) > 0 \;\Rightarrow\; \downarrow\pi \in \Downarrow {C_f}^*$.

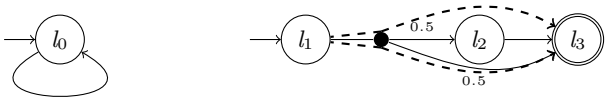


Fig. 4. Example of an increasing reachability probability after applying Lemma 3: Consider the scheduler running the second component once, then the first component forever. Probability of eventually reaching l_3 is 0.5 as the second component may remain in l_2 forever. After transformation, the scheduler has to follow each chain entirely which yields possible reachability probabilities 0 and 1.

Proof. We define $\tilde{\mathfrak{S}}(\pi)$ by induction on $|\pi|$. See Fig. 5 for illustration.

– If $\mathfrak{S}(\pi) = \vec{t}$ is a transition of the original system that was kept in the reduced one, we immediately define $\tilde{\mathfrak{S}}(\pi) = \vec{t}$ which satisfies the property.
– Otherwise, $\mathfrak{S}(\pi)$ is a transition that has now been subsumed by a set of chains. More precisely, for each $i \in [1 : N]$, the next transition run on component $\mathcal{C}_i$ will represent the beginning of a fixed given chain c_i. Let us consider the maximal extension $\pi\pi'$ such that π' contains only Dirac links from one of the c_i. (not necessarily from the same component). Two cases can occur:
  • Either the system is deadlocked: $\pi\pi'$ has no successor. We swap the Dirac transitions, which are assumed to be mobile, in order to write $\downarrow\pi' = \downarrow c'_1 \cdots \downarrow c'_N$ with for each i, c'_i a prefix of c_i. We then apply Condition 1d to show that if φ doesn't hold at state(π), it cannot hold anywhere in π' either, so that for any scheduler $\tilde{\mathfrak{S}}$ already defined up to π and arbitrarily defined later, $\mathrm{P}^{\mathfrak{S}}(\Diamond\varphi) \leq \mathrm{P}^{\tilde{\mathfrak{S}}}(\Diamond\varphi)$.
  • Otherwise, we consider $i \in [1 : N]$ such that $\mathcal{C}_i$ is the first process to have fired the last Dirac transition of its current chain. Let us consider the set of chains cs $\subseteq C_f$ corresponding to the sequence of non-deterministic choice made by $\mathcal{C}_i$. The rest of the run for $\mathcal{C}_i$ is indeed determined: It consists in a finite sequence of probabilistic transitions according to the chains of cs. We apply Lemma 2 recursively on each link of cs to move

it towards the beginning of the trace. This is possible because each link is assumed to be mobile and has to be swapped only with probabilistic transitions of other components. Thus, we have defined a scheduler $\tilde{\mathfrak{S}}$ such that $\mathrm{P}^{\tilde{\mathfrak{S}}}(\pi\pi')$ implies that there exists $c \in \mathrm{cs}$ such that $\downarrow c \sqsubseteq \downarrow \pi'$.

Remark 4. The transformation as stated in Lemma 3 may strictly increase the reachability probability. As an example consider Fig. 4. We note however that this can only happen for non-optimal schedulers, as pointed out by Lemma 1 so maximal reachability probability is still preserved.

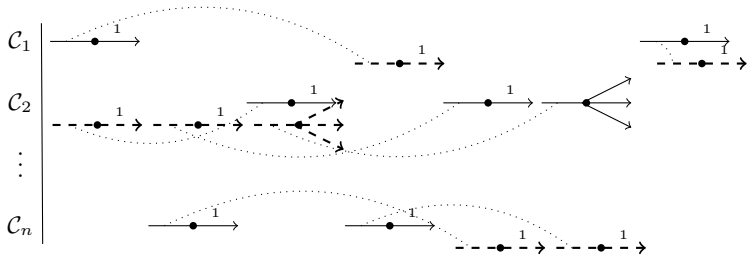


Fig. 5. Illustration of the proof strategy for Lemma 3. Here, the first component to terminate a chain is $\mathcal{C}_2$, represented by the first non-Dirac transition. All links of the chain are then "pulled" to the beginning of the run. This transformation is achieved by permuting links in $\mathcal{C}_2$ with links of the other components. The latter links are indeed assumed by minimality to be Dirac and so can always be postponed with Lemma 2.

Remark 5. The reduced model may lack deadlocks exhibited by the original system: Consider the network comprising only the process on the left of Fig. 6, with only one non-trivial chain going through $l_1 l_2 l_3$. Clearly, the system can deadlock in location l_2, which is why $\mathrm{P}_{\min}(\Diamond(\mathrm{x} = 2)) = 0$. After transformation (depicted on the right), the guard becomes ff which leads to $\mathrm{P}_{\min}(\Diamond(\mathrm{x} = 2)) = 1$. This is why we restrict ourselves to maximal reachability probabilities, where a scheduler has no incentive to trigger a deadlock as it will only reduce the reachability probability.

We conclude this section by establishing the correctness of our transformation:

Theorem 1. *The following equality holds for reachability objectives $\Diamond\varphi$:*

$$\mathrm{P}_{\max,\mathcal{N}_r}(\Diamond\varphi) = \mathrm{P}_{\max,\mathcal{N}}(\Diamond\varphi)$$

Proof. Lemma 1 ensures that our transformation does not introduce new behavior and thus $\mathrm{P}_{\max,\mathcal{N}_r}(\Diamond\varphi) \leq \mathrm{P}_{\max,\mathcal{N}}(\Diamond\varphi)$. Equality follows with Lemma 3, allowing us to convert any $\mathcal{N}$-scheduler into an $\mathcal{N}_r$-scheduler with a reachability probability at least as high as the one in the original system.

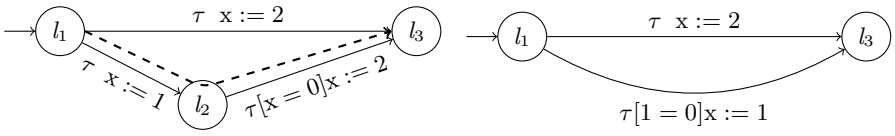


Fig. 6. A network comprising only a single process, with one nontrivial chain, before and after transformation

5 Case Studies

We have implemented our approach in the model-checking framework Modest [5,13]. Experiments were conducted on an OpenSUSE Linux machine with an Intel Core i7-6700 CPU (3.40GHz) and 32GB of working memory. We considered two different exemplary cases: On the one hand, we looked at the Factory example, depicted in Fig. 1. This is a genuine VMDP network and it is not manually hand-optimized for model-checking, contrary to essentially all existing case studies around. In addition, we looked at an example on flow control, from the context of the SPIN model checker. This is a non-probabilistic case study, and we use it to demonstrate in how far our approach can place `atomic`s mechanically.

Factory. To illustrate the nature of our transformation and its ability to reduce state space size, we applied it to instances of the network in Fig. 1. The algorithm generated the chain set described in Example 4 . We had Modest answer the query $\mathrm{P}_{\max}(\Diamond(\text{processed} = 4 \wedge \text{broken} = 0))$ on both the original and the transformed model. Table 1 shows our results for different numbers of workers: While the exponential growth in both state space size as well as model-checking runtime caused by the additional interleavings that each new worker introduces can in principle not be avoided by our transformation, the reduction ratio increases with the number of workers. For more than 4 workers, Modest gives up for lack of memory on the original system, while it still succeeds to answer the query on the transformed system for 5 and for 6 workers.

For 6 workers our algorithm took only about one second of runtime, including the optimization problem in step B, and used only negligible amounts of working memory. Even more encouraging, the performance of our algorithm would not be affected at all by changes to constants such as the total number of work items in the factory, that have a dramatic impact on the state space size.

Flow Control. The model checker Spin comes, among others, with a Promela example file `pftp.pml`. In this model, a sender and a receiver communicate over a lossy channel, supported by a flow control layer that prevents message loss and reordering. Holzmann discusses the model in the original Spin book [16] and notes that whenever possible, one should surround parts of the code by `atomic` blocks, to "reduce the complexity of the validation". Our goal in this case study was to omit the `atomic` keyword and have our algorithm infer the corresponding chains automatically.

Table 1. Resource usage for model checking the network from Fig. 1 before and after our transformation

	# Workers	1	2	3	4	5	6
Original	**# States**	1 558	164 264	5 207 980	59 873 864	?	?
	MC time	< 1s	1.2s	48s	10m50s	?	?
Transformed	**# States**	719	56 291	1 187 248	9 994 337	38 657 750	104 937 279
	MC time	< 1s	0.7s	16s	2m19s	9m07s	25m25s
Reduced by	**# States**	54%	66%	77%	83%	?	?
	MC time	0%	42%	67%	79%	?	?

We slightly modified the model: Spin's special `timeout` predicate (that is used to resend messages when no other transition in the system is enabled) was replaced by a constant `true`, because its non-compositional semantics is difficult to capture in Modest. Furthermore, we changed the model by disabling `white` messages, so as to have the MDP-focussed model-checking engine of Modest scale better in the absence of (nontrivial) probabilities. We then translated the Spin model to an isomorphic VMDP network, ignoring all the `atomic` keywords.

Applying our algorithm to the translated model had the expected effects: Most of the `atomic` blocks of the original could be recovered in the form of chains, with one restriction: Chains stop at channel accesses, because we had to model those by reads and writes to global variables. Notably, Holzmann's `atomic`s can span across channel accesses, but lose atomicity (only) in case a channel access blocks (which again is a non-compositional feature). A small number of the chains we generate are hard to translate into Promela `atomic`s, e.g. because they start in front of a loop and end inside it.

Despite not recovering the original `atomic`s precisely, we achieve a dramatic reduction in state space size: Checking whether any assertions in the model are violated Modest explored 15 922 533 states of the original model, but only 4 588 039 on the transformed one, a reduction by 70%. In total the algorithm generated 99 chains, all of which remained after filtering, which used 72 056 MAXSAT variables and 147 176 clauses. Again, steps A, B and C together took about one second of runtime and negligible amounts of memory.

For comparison we applied Spin to our variants of `pftp.pml`. Table 2 shows that the `atomic` keywords we manually derived from the generated chains reduce state space considerably (about 24%), though not as much as the original `atomic`'s in [16]. The first column shows that, surprisingly, Spin benefits from `atomic` keywords far less than Modest benefits from our transformation. This may again be rooted in the fact that Modest is optimized for probabilistic model checking. The second column reveals that even under partial order reduction the usage of `atomic` keywords can reduce state space size considerably. This suggests that one should use our technique in addition to POR.

Table 2. SPIN state space sizes for our variants of `pftp.pml`, with and without POR

	Without POR	**With POR**
No `atomics`	97 652	15 062
Chain `atomics`	74 442	12 088
Holzmann's `atomics`	46 773	11 248

6 Conclusion

We have presented an automated approach to fusing transition executions prior to model checking, within models of concurrent probabilistic processes, so as to alleviate the state space explosion problem. The probabilistic setting makes this task particularly challenging, owed to the tree-shaped structure of probabilistic executions. However, our approach is readily applicable to the nonprobabilistic setting, too, where it effectively yet mechanically detects instruction sequences that can be made atomic without altering reachability properties, as demonstrated on Holzmann's flow control example. Furthermore, the *Factory* case study demonstrates that on concurrent probabilistic examples the state-space compression factor achieved becomes more drastic the larger the models get.

A comparison to SPIN's implementation of partial order reduction demonstrated that while our approach achieves less state space reduction than POR, using it in addition to existing POR implementations can still increase the reduction ratio considerably. Notably, we were unable to report successes of our techniques on established examples, mainly because those have been highly optimised by hand, leaving no room for detecting further transitions to fuse.

There are several avenues extending this first work on *syntactic partial order compression*. Condition 1b constraining chains in Definition 5 could for instance be relaxed by replacing it by a more precise analysis. As a matter of fact, some transitions may never happen concurrently so that less independence requirements have to be met in order to establish a chain. We are working on an abstract interpretation-based approach to collect state information statically and exploit it for justifying chains even across synchronizing actions and accesses to global variables. Furthermore, we plan to investigate how to extend beyond reachability properties. Our approach does at this point add deadlocks to the model (see Remark 5), which implies that minimum probabilities are not preserved.

Acknowledgments. This work is partly supported by the DFG as part of CRC 248 (see perspicuous-computing.science) and by the ERC Advanced Investigators Grant 695614 (POWVER).

References

1. Abdulla, P.A., Aronis, S., Jonsson, B., Sagonas, K.: Source sets: A foundation for optimal dynamic partial order reduction. J. ACM **64**(4), 25:1–25:49 (Aug 2017). https://doi.org/10.1145/3073408

2. Baier, C., Grosser, M., Ciesinski, F.: Partial order reduction for probabilistic systems. In: First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. pp. 230–239 (Sept 2004). https://doi.org/10.1109/QEST.2004.1348037
3. Baier, C., D'Argenio, P., Groesser, M.: Partial order reduction for probabilistic branching time. Electronic Notes in Theoretical Computer Science **153**(2), 97 – 116 (2006). https://doi.org/https://doi.org/10.1016/j.entcs.2005.10.034, proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005)
4. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA. pp. 125–126. IEEE Computer Society (2006). https://doi.org/10.1109/QEST.2006.59
5. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.: MODEST: A compositional modeling formalism for hard and softly timed systems. IEEE Trans. Software Eng. **32**(10), 812–830 (2006). https://doi.org/10.1109/TSE.2006.104
6. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984). https://doi.org/10.1145/828.833
7. D'Argenio, P.R., Niebert, P.: Partial order reduction on concurrent probabilistic programs. In: 1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The Netherlands. pp. 240–249. IEEE Computer Society (2004). https://doi.org/10.1109/QEST.2004.1348038
8. Díaz, Á.F., Baier, C., Earle, C.B., Fredlund, L.: Static partial order reduction for probabilistic concurrent systems. In: Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012. pp. 104–113. IEEE Computer Society (2012). https://doi.org/10.1109/QEST.2012.22
9. Flanagan, C., Godefroid, P.: Dynamic partial-order reduction for model checking software. In: Palsberg, J., Abadi, M. (eds.) Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005. pp. 110–121. ACM (2005). https://doi.org/10.1145/1040305.1040315
10. Garavel, H., Lang, F., Serwe, W.: From LOTOS to LNT. In: Katoen, J., Langerak, R., Rensink, A. (eds.) ModelEd, TestEd, TrustEd - Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 10500, pp. 3–26. Springer (2017). https://doi.org/10.1007/978-3-319-68270-9_1
11. Giro, S., D'Argenio, P.R., Fioriti, L.M.F.: Partial order reduction for probabilistic systems: A revision for distributed schedulers. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5710, pp. 338–353. Springer (2009). https://doi.org/10.1007/978-3-642-04081-8_23
12. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem, Lecture Notes in Computer Science, vol. 1032. Springer (1996). https://doi.org/10.1007/3-540-60761-7
13. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods in System Design **43**(2), 191–232 (2013). https://doi.org/10.1007/s10703-012-0167-z

14. Hartmanns, A.: On the analysis of stochastic timed systems. Ph.D. thesis, Saarland University (2015). https://doi.org/10.22028/D291-26597
15. Hermanns, H., Kwiatkowska, M.Z., Norman, G., Parker, D., Siegle, M.: On the use of mtbdds for performability analysis and verification of stochastic systems. J. Log. Algebr. Program. **56**(1-2), 23–67 (2003). https://doi.org/10.1016/S1567-8326(02)00066-8
16. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice-Hall, Englewood Cliffs, New Jersey (1991)
17. Katz, S., Peled, D.A.: Defining conditional independence using collapses. Theor. Comput. Sci. **101**(2), 337–359 (1992). https://doi.org/10.1016/0304-3975(92)90054-J
18. Peled, D.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) Computer Aided Verification. pp. 409–423. Springer Berlin Heidelberg, Berlin, Heidelberg (1993). https://doi.org/10.1007/3-540-56922-7_34
19. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1994). https://doi.org/10.1002/9780470316887
20. Teige, T.: Stochastic satisfiability modulo theories: a symbolic technique for the analysis of probabilistic hybrid systems. Ph.D. thesis, Carl von Ossietzky University of Oldenburg (2012), https://oops.uni-oldenburg.de/id/eprint/1389
21. Valmari, A.: Stubborn sets for reduced state space generation. In: Rozenberg, G. (ed.) Advances in Petri Nets 1990. pp. 491–515. Springer Berlin Heidelberg, Berlin, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_36