Action Synthesis for Branching Time Logic: Theory and Applications

MICHAŁ KNAPIK, Institute of Computer Science, PAS, Warsaw, Poland
ARTUR MĘSKI, Institute of Computer Science, PAS, Warsaw, and FMCS, University of Łódź, Poland
WOJCIECH PENCZEK, Institute of Computer Science, PAS, Warsaw, and University of Natural Sciences and Humanities, Siedlce, Poland

64

The article introduces a parametric extension of Action-Restricted Computation Tree Logic called pmARCTL. A symbolic fixed-point algorithm providing a solution to the exhaustive parameter synthesis problem is proposed. The parametric approach allows for an in-depth system analysis and synthesis of the correct parameter values. The time complexity of the problem and the algorithm is provided. An existential fragment of pmARCTL (pmEARCTL) is identified, in which all of the solutions can be generated from a minimal and unique base. A method for computing this base using symbolic methods is provided. The prototype tool SPATULA implementing the algorithm is applied to the analysis of three benchmarks: faulty Train-Gate-Controller, Peterson's mutual exclusion protocol, and a generic pipeline processing network. The experimental results show efficiency and scalability of our approach compared to the naive solution to the problem.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification

General Terms: Algorithms, Reliability, Verification

Additional Key Words and Phrases: Parametric model checking, parameter synthesis, parametric verification

ACM Reference Format:
M. Knapik, A. Męski, and W. Penczek. 2015. Action synthesis for branching time logic: Theory and applications. ACM Trans. Embed. Comput. Syst. 14, 4, Article 64 (September 2015), 23 pages.
DOI: http://dx.doi.org/10.1145/2746337

1. INTRODUCTION

Parameter synthesis is a generalization of the model checking problem, where a formula [Alur et al. 2001; Bruyére et al. 2008; Giampaolo et al. 2010] and/or a model [Alur et al. 1993; Hune et al. 2002] are augmented with parameters, and aims at computing the values of the parameters that make the formula hold in the model. The parametric approach may be useful at the design phase to support decisions in software and hardware production, as it may provide the exact values for tunable parameters or sets of rules that govern the system execution, often saving time spent on tedious experiments with the possible parameter valuations.

In this work, we focus on the action synthesis problem for the parameters introduced to the formulas of a branching-time temporal logic. We build upon Action-Restricted Computation Tree Logic (ARCTL) [Pecheur and Raimondi 2006], which we augment

M. Knapik is supported by the Foundation for Polish Science under Int. Ph.D. Projects in Intelligent Computing. The project was financed from the EU within the Innovative Economy OP 2007-2013 and ERDF, and was partially funded by a DEC-2012/07/N/ST6/03426 NCN Preludium 4 grant.
Authors' addresses: M. Knapik, A. Męski, and W. Penczek, Institute of Computer Science, PAS, Warsaw, Poland; emails: {knapik, meski, penczek}@ipipan.waw.pl.

DOI: http://dx.doi.org/10.1145/2746337

with parameters corresponding to the sets of actions, by defining the logic parametric ARCTL (pmARCTL). To solve the synthesis problem for pmARCTL, we propose a fixed-point based algorithm, inspired by Jones et al. [2012], that processes the verified formula recursively and labels each state of the model with the valuations of the parameters under which the formula holds in this state. A novel framework for the parameter synthesis for pmARCTL, consisting of a theory, an implementation, and the tool SPATULA, is the main contribution of our article. We are also the first to demonstrate how to efficiently apply the exhaustive parameter synthesis in systems design and analysis by showing the potential of the method in identifying possible attack scenarios. We demonstrate this on Peterson's mutual exclusion algorithm by presenting what instructions need to be injected into the memory monitor to expose a subtle weakness. We also prove that the emptiness problem for pmARCTL is NP-complete and provide the complexity results for the proposed algorithms. Even though the problem is of a prohibitive theoretical complexity, our implementation significantly outperforms the naive approach and makes the method quite practical as we demonstrate on two scalable examples: a faulty Train-Gate-Controller and a generic pipeline processing network.

The problem of synthesis of the valuations under which a given modal property holds was first investigated in Alur et al. [2001] in the context of a parametric version of LTL. In Bruyére et al. [2008] and Giampaolo et al. [2010], the authors analyze parametric extensions of MITL and TCTL, respectively. In Jones et al. [2012], the problem of synthesis for agent groups of the CTLK properties in a multiagent setting was considered. In Classen et al. [2011], the authors focus on a verification of *feature* CTL (fCTL) properties for software product lines with the extended validity check providing constraints on when a given property does not hold. Despite the fact that the authors do not consider parameterized logics, their work shares the same difficulties as the problems we deal with in this article: both the state space and the set of solutions are susceptible to exponential blowup. The experimental results of Classen et al. [2011] show that the symbolic verification of fCTL can be up to more than 700 times faster than a brute-force approach. We extend these results to pmARCTL parameter synthesis, where the relative speedup can exceed 8,000. The work presented here is also related to parametric model checking with parameters in models [Alur et al. 1993; André et al. 2012; Hune et al. 2002] and model synthesis from a specification [Clarke and Emerson 1981; Katz and Peled 2010].

The rest of the article is organized as follows. In Section 2, we introduce the syntax and the semantics of pmARCTL. The algorithms for the parameter synthesis are given in Section 3. An experimental evaluation is provided in Section 4, followed by a summary and concluding remarks.

2. MIXED TRANSISTION SYSTEMS AND pmARCTL

In this section, we recall some basic definitions and present the syntax and semantics of the logic pmARCTL used in the article. Mixed transition systems (MTSs) [Pecheur and Raimondi 2006] essentially are Kripke structures with the transitions labeled with actions. The labels serve us to express branching-time properties with the selected set of actions allowed along a given run.

Definition 2.1 (*MTS*). Let $\mathcal{PV}$ be a set of propositional variables. A *mixed transition system* is a 5-tuple $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{V}_s)$, where

—$\mathcal{S}$ is a nonempty finite set of states,
—$s^0 \in \mathcal{S}$ is the initial state,
—$\mathcal{A}$ is a nonempty finite set of actions,
—$\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is a transition relation, and
—$\mathcal{V}_s : \mathcal{S} \to 2^{\mathcal{PV}}$ is a (state) valuation function.

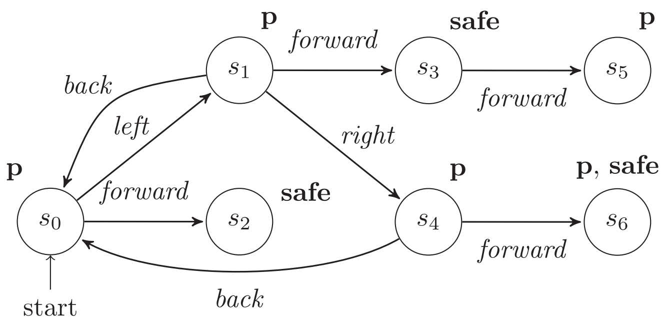

Fig. 1. A simple MTS used in Examples 2.2. through 3.3.

As usual, we write $s \xrightarrow{a} s'$ if $(s, a, s') \in \mathcal{T}$. Let $\chi \subseteq \mathcal{A}$ be a nonempty set of actions. Let $\pi = (s_0, a_0, s_1, a_1, \ldots)$ be a finite or infinite sequence of interleaved states and actions; by $|\pi|$ we denote the number of the states of π if π is finite, and by ω if π is infinite. A sequence π is a *path* over χ if (1) $s_i \xrightarrow{a_i} s_{i+1}$ and $a_i \in \chi$ for each $i < |\pi|$ and (2) π is maximal with respect to condition (1). Note that if a path π is finite, then its final state does not have a χ-successor state in $\mathcal{S}$—that is, if $\pi = (s_0, a_0, s_1, a_1, \ldots, s_m)$, then there is no $s' \in \mathcal{S}$ and $a \in \chi$ such that $s_m \xrightarrow{a} s'$.

The set of all paths over $\chi \subseteq \mathcal{A}$ in a model $\mathcal{M}$ is denoted by $\Pi(\mathcal{M}, \chi)$, whereas the set of all paths $\pi \in \Pi(\mathcal{M}, \chi)$ starting from a given state $s \in \mathcal{S}$ is denoted by $\Pi(\mathcal{M}, \chi, s)$. We omit the model symbol if it is clear from the context, simply writing $\Pi(\chi)$ and $\Pi(\chi, s)$. By $\Pi^{\omega}(\chi)$ and $\Pi^{\omega}(\chi, s)$, we mean the corresponding sets restricted to the infinite paths only.

Example 2.2. Figure 1 presents a simple MTS with $\mathcal{PV} = \{p, \text{safe}\}$, actions $\mathcal{A} = \{\text{left, right, forward, back}\}$, and the initial state s_0. The path $(s_0, \text{left}, s_1, \text{right}, s_4)$ belongs to $\Pi(\{\text{left, right}\})$, but it does not belong to $\Pi(\{\text{left, right, back}\})$, because although $(s_0, \text{left}, s_1, \text{right}, s_4)$ is a maximal path over $\{\text{left, right}\}$, it is not maximal over $\{\text{left, right, back}\}$, as it can be extended, for example, into an infinite path $(s_0, \text{left}, s_1, \text{right}, s_4, \text{back}, s_0, \ldots) \in \Pi(\{\text{left, right, back}\})$.

The MTSs defined in this article slightly differ from those introduced in Pecheur and Raimondi [2006], where the actions that label the transitions are treated as propositions. The difference is not essential, however, as in Pecheur and Raimondi [2006], the propositional formulas over actions serve only to select sets of actions allowed along considered runs. Here we describe the actions allowed explicitly.

2.1. Parametric ARCTL

The presented logic is a parametric extension of ARCTL [Pecheur and Raimondi 2006]. The language of ARCTL consists of the CTL-like branching-time formulas. The main difference between ARCTL and CTL is that each path quantifier is subscripted with a set of actions. The subscripts are used in path selection—for example, $E_{\{\text{left, right}\}}G(E_{\{\text{forward}\}}F$ safe) may be read as "there exists a path over *left* and *right*, on which it holds globally that a state satisfying safe is reachable along some path over *forward*." pmARCTL extends ARCTL by allowing free variables in place of sets of actions, such as $E_Y G(E_Z F \text{safe})$.

Definition 2.3 (pmARCTL *syntax*). Let $\mathcal{A}$ be a finite set of actions, $\text{ActSets} = 2^{\mathcal{A}} \setminus \{\emptyset\}$, ActVars be a finite set of variables, and $\mathcal{PV}$ be a set of propositional variables. The set of formulas of *parametric Action-Restricted Computation Tree Logic* is defined by the following grammar:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid E_\alpha X\phi \mid E_\alpha G\phi \mid E^\omega_\alpha G\phi \mid E_\alpha(\phi\, U\phi),$$

where $p \in \mathcal{PV}$, $\alpha \in$ ActSets $\cup$ ActVars.

The E path quantifier is read as "there exists a path." The superscript $^\omega$ restricts the quantification to the infinite paths, whereas the subscript $_\alpha$ restricts the quantification to the paths over α. The X modality stands for "in the next state." The state modality G is the "globally" modality. The modality U stands for "until."

Since the formulas considered contain free variables, their validity needs to be defined with respect to the provided valuations of ActVars. A function υ : ActVars $\rightarrow$ ActSets is called an *action valuation*, and the set of all action valuations is denoted by ActVals. By $\mathcal{M}, s \models_\upsilon \phi$, we denote that the formula ϕ holds in the state s of the model $\mathcal{M}$ under the valuation υ, as formalized in Definition 2.4 (we omit the model symbol when it is clear from the context). In what follows, by π_i we denote the i-th state of π. For conciseness, if υ is an action valuation, then let

$$\upsilon(\alpha) \stackrel{def}{=} \begin{cases} \chi & \text{if } \alpha = \chi \subseteq \mathcal{A}, \\ \upsilon(Y) & \text{if } \alpha = Y \in \text{ActVars}. \end{cases}$$

Moreover, in the following notations, for $O \in \{E, A, \Pi\}$ we assume that $O^\epsilon = O$.

Definition 2.4 (pmARCTL *semantics*). Let $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{V}_s)$ be an MTS and $\upsilon \in$ ActVals be an action valuation. The relation $\models_\upsilon$ is defined as follows:

—$s \models_\upsilon p$ if and only if $p \in \mathcal{V}_s(s)$,
—$s \models_\upsilon \neg\phi$ if and only if $s \not\models_\upsilon \phi$,
—$s \models_\upsilon \phi \vee \psi$ if and only if $s \models_\upsilon \phi$ or $s \models_\upsilon \psi$,
—$s \models_\upsilon E_\alpha X\phi$ if and only if there exists $\pi \in \Pi(\upsilon(\alpha), s)$ such that $|\pi| > 1$ and $\pi_1 \models_\upsilon \phi$,
—$s \models_\upsilon E^r_\alpha G\phi$ if and only if there exists $\pi \in \Pi^r(\upsilon(\alpha), s)$ such that $\pi_i \models_\upsilon \phi$ for all $i < |\pi|$,
—$s \models_\upsilon E_\alpha(\phi\, U\psi)$ if and only if there exists $\pi \in \Pi(\upsilon(\alpha), s)$ such that $\pi_i \models_\upsilon \psi$ for some $i < |\pi|$ and $\pi_j \models_\upsilon \phi$ for all $0 \leq j < i$,

where $p \in \mathcal{PV}$, $\phi, \psi \in$ pmARCTL, $r \in \{\omega, \epsilon\}$, and $\alpha \in$ ActSets $\cup$ ActVars.

Next we define several derived modalities. Let $\phi, \psi \in$ pmARCTL and denote the following:

(1) $E^\omega_\alpha X\phi \stackrel{def}{=} E_\alpha X(\phi \wedge E^\omega_\alpha G \text{ true})$,
(2) $E^\omega_\alpha(\phi\, U\psi) \stackrel{def}{=} E_\alpha(\phi\, U(\psi \wedge E^\omega_\alpha \text{ true}))$,
(3) $E^r_\alpha F\phi \stackrel{def}{=} E^r_\alpha(\text{true}\, U\phi)$,
(4) $A^r_\alpha X\phi \stackrel{def}{=} \neg E^r_\alpha X\neg\phi$,
(5) $A^r_\alpha G\phi \stackrel{def}{=} \neg E^r_\alpha F\neg\phi$,
(6) $A^r_\alpha(\phi\, U\psi) \stackrel{def}{=} \neg(E^r_\alpha(\neg\psi U \neg(\phi \vee \psi)) \vee E^r_\alpha G\neg\psi)$,
(7) $A^r_\alpha F\phi \stackrel{def}{=} \neg E^r_\alpha G\neg\phi$,

where $\alpha \in$ ActSets $\cup$ ActVars and $r \in \{\omega, \epsilon\}$. The modality F stands for "in some future state," A_α stands for "for each path over α," and A^ω_α stands for "for each infinite path over α." The semantics of the derived modalities is consistent with the intuition.

Example 2.5. Consider the MTS from Figure 1 and the formulas $\phi_1 = A_Y Gp$ and $\phi_2 = A^\omega_Y Gp$. It is easy to check that for $\upsilon \in$ ActVals such that $\upsilon(Y) = \{\text{left, right, back}\}$, the set $\Pi(\upsilon(Y), s_0)$ consists of infinite paths only, and we have $s_0 \models_\upsilon \phi_1$ and $s_0 \models_\upsilon \phi_2$. On the other hand, for $\upsilon' \in$ ActVals satisfying $\upsilon'(Y) = \{\text{left, right, back, forward}\}$, the set $\Pi(\upsilon(Y), s_0)$ contains finite paths along which p does not hold globally (e.g., (s_0, s_1, s_3, s_5)), therefore $s_0 \not\models_{\upsilon'} \phi_1$ while $s_0 \models_{\upsilon'} \phi_2$.

3. ACTION SYNTHESIS FOR pmARCTL

Consider a formula $\phi_{ddk} = A_Y G(E_Y X\,\text{true})$. This property expresses lack of deadlock—that is, $s \models_\upsilon \phi_{ddk}$ if and only if each state reachable from s via transitions labeled with actions from $\upsilon(Y)$ has a $\upsilon(Y)$-successor. The complete description of the set of valuations under which ϕ_{ddk} holds in the initial state of $\mathcal{M}$ can provide crucial information about the safety of the modeled system. If the model is *overspecified*, then these valuations can be used for its pruning, such as removing unnecessary actions while preserving the no-deadlock attribute. The space of synthesized valuations can be explored with many goals in mind, such as a minimal model selection, a correct model synthesis from a general skeleton, and failure resistance.

The main focus of this article is therefore on the automatic and efficient synthesis of the subset of the action valuations under which a given formula holds. More formally, for a given model $\mathcal{M}$ and a formula ϕ of pmARCTL, we define the function $f_\phi : \mathcal{S} \to 2^{\text{ActVals}}$ satisfying for all $s \in \mathcal{S}$ the condition

$$\upsilon \in f_\phi(s) \text{ if and only if } s \models_\upsilon \phi \text{—} \tag{$\star$}$$

that is, $f_\phi(s)$ returns all valuations under which ϕ holds in s.

Example 3.1. Consider the model in Figure 1 and the formula $\phi_1 = E_Y(pU(p \wedge \text{safe}))$. By hand calculations, one can check that $s_0 \models_\upsilon \phi_1$ if and only if $\{\text{forward, left, right}\} \subseteq \upsilon(Y)$.

In what follows, given a model, by writing the function f_ϕ for ϕ of pmARCTL, we assume that f_ϕ satisfies the condition ($\star$).

3.1. Algorithms for Computing f_ϕ

Next we show how to compute the function f_ϕ by means of the recursive compositions, preimage, and fixpoints. Throughout this section, let $\mathcal{M}$ be a fixed MTS and $Y \in$ ActVars.

3.1.1. Propositional Variables and Boolean Operations. Let $p \in \mathcal{PV}$ be a propositional variable and $s \in \mathcal{S}$ be a state. Notice that the set $f_p(s)$ consists of either all action valuations if s is labeled with p or is empty otherwise, and thus

$$f_p(s) = \begin{cases} \text{ActVals} & \text{if } p \in \mathcal{V}_s(s), \\ \emptyset & \text{if } p \notin \mathcal{V}_s(s). \end{cases}$$

Now let $\phi \in$ pmARCTL and f_ϕ be given. Then the set $f_{\neg\phi}(s)$ consists of all action valuations υ such that $s \not\models_\upsilon \phi$. From the inductive assumption, this is equivalent to $\upsilon \notin f_\phi(s)$, from which follows $f_{\neg\phi}(s) = \text{ActVals} \setminus f_\phi(s)$. To deal with the Boolean connectives, assume that $\phi, \psi \in$ pmARCTL and f_ϕ and f_ψ are given. Recall that from definition $s \models_\upsilon \phi \vee \psi$ if and only if $s \models_\upsilon \phi$ or $s \models_\upsilon \psi$. By the inductive assumption, $s \models_\upsilon \phi$ or $s \models_\upsilon \psi$ is equivalent to $\upsilon \in f_\phi(s)$ or $\upsilon \in f_\psi(s)$, and therefore $f_{\phi\vee\psi}(s) = f_\phi(s) \cup f_\psi(s)$.

3.1.2. Parametric Preimage and neXt. Let $f : \mathcal{S} \to 2^{\text{ActVals}}$ be a function. The *existential parametric preimage* of f with respect to $Y \in$ ActVars is defined as the function $\text{parPre}^{\exists}_Y(f) : \mathcal{S} \to 2^{\text{ActVals}}$ such that for each $s \in \mathcal{S}$,

$$\text{parPre}^{\exists}_Y(f)(s) = \left\{\upsilon \mid \exists_{s' \in \mathcal{S}}\, \exists_{a \in \upsilon(Y)}\, s \xrightarrow{a} s' \wedge \upsilon \in f(s')\right\}.$$

It follows immediately from the $(\star)$ condition that for each $\phi \in$ pmARCTL, the set $\mathrm{parPre}^{\exists}_{Y}(f_\phi)(s)$ consists of all such action valuations υ that some state s' such that $s' \models_\upsilon \phi$ can be reached by firing an action from $\upsilon(Y)$.

LEMMA 3.2. *For each $s \in \mathcal{S}$, $\phi \in$ pmARCTL, $Y \in$ ActVars, and $\upsilon \in$ ActVals, the following condition holds: $s \models_\upsilon E_Y X\phi$ if and only if $\upsilon \in \mathrm{parPre}^{\exists}_{Y}(f_\phi)(s)$.*

PROOF. From the definition $s \models_\upsilon E_Y X\phi$ if and only if there exists a path $\pi \in \Pi(\upsilon(Y), s)$ such that $|\pi| > 1$ and $\pi_1 \models_\upsilon \phi$, which is equivalent to $\exists_{s' \in \mathcal{S}} \exists_{a \in \upsilon(Y)} (s \xrightarrow{a} s' \wedge s' \models_\upsilon \phi)$ as (s, a, s') can be extended to a path. This, in turn, is equivalent to $\exists_{s' \in \mathcal{S}} \exists_{a \in \upsilon(Y)} (s \xrightarrow{a} s' \wedge \upsilon \in f_\phi(s'))$—that is, $\upsilon \in \mathrm{parPre}^{\exists}_{Y}(f_\phi)(s)$. □

The meaning of the preceding lemma can be expressed as $f_{E_Y X\phi} = \mathrm{parPre}^{\exists}_{Y}(f_\phi)$ for each $\phi \in$ pmARCTL.

Example 3.3. Consider the MTS from Figure 1. By case-by-case analysis, one can see that $s_1 \models_\upsilon E_Y F\mathrm{safe}$ if and only if forward $\in \upsilon(Y)$ and $s_2 \models_\upsilon E_Y F\mathrm{safe}$ for all $\upsilon \in$ ActVals, and thus $f_{E_Y F\mathrm{safe}}(s_1) = \{\upsilon \mid \text{forward} \in \upsilon(Y)\}$ and $f_{E_Y F\mathrm{safe}}(s_2) =$ ActVals. To compute $\mathrm{parPre}^{\exists}_{Y}(f_{E_Y X(E_Y F\mathrm{safe})})(s_0)$, notice that to reach s_1 or s_2 from s_0, the actions *left* or *forward* should be fired, respectively. Therefore,

$$\begin{aligned}\mathrm{parPre}^{\exists}_{Y}\big(f_{E_Y X(E_Y F\mathrm{safe})}\big)(s_0) &= \bigcup_{i \in \{1,2\}} \big(\{\upsilon \mid \exists_{a \in \upsilon(Y)}\, s_0 \xrightarrow{a} s_i\} \cap f_{E_Y F\mathrm{safe}}(s_i)\big) \\ &= \{\upsilon \mid \text{forward} \in \upsilon(Y)\}.\end{aligned}$$

3.1.3. Two Versions of the Globally Modality. We employ the equivalence $E^{\omega}_{Y} G\phi \equiv \phi \wedge E_Y X E^{\omega}_{Y} G\phi$ to obtain Algorithm 1. Note the similarity to its nonparametric counterpart. The case of $E_Y G$ is more interesting, as a potential lack of the totality of the transition relation needs to be taken into account. To this end, Algorithm 2 consecutively keeps adding action valuations under which the given states satisfy the considered formula, but are *deadlocked* (i.e., have no successors). This corresponds to the addition of the sink state, used to deal with the nontotality of the transition relation in the classical theory.

ALGORITHM 1: $\mathit{Synth}_{E^{\omega}G}\,(f_\phi, Y)$

Input: $f_\phi \in \left(2^{\mathrm{ActVals}}\right)^{\mathcal{S}}$
Output: $f_{E^{\omega}_{Y} G\phi} \in \left(2^{\mathrm{ActVals}}\right)^{\mathcal{S}}$
1: $f := f_\phi$; $h := \emptyset$
2: **while** $f \neq h$ **do**
3: $\quad h := f$
4: $\quad f := f_\phi \cap \mathrm{parPre}^{\exists}_{Y}(h)$
5: **end while**
6: **return** f

LEMMA 3.4. *Let ϕ be a* pmARCTL *formula, $r \in \{\omega, \epsilon\}$, and $Y \in$* ActVars. *For all $s \in \mathcal{S}$ and $\upsilon \in$* ActVals, *we have $s \models_\upsilon E^{r}_{Y} G\phi$ if and only if $\upsilon \in \mathit{Synth}_{E^{r}G}(f_\phi, Y)(s)$.*

PROOF. Let us first prove that $s \models_\upsilon E^{\omega}_{Y} G\phi$ if and only if $\upsilon \in \mathit{Synth}_{E^{\omega}G}(f_\phi, Y)(s)$. For a while, replace the condition in line 2 of Algorithm 1 with true. In this way, the while loop 2–5 becomes infinite, and we can define f_i for each $i \in \mathbb{N}$ as the value of the f variable after the i-th run and $f_0 = f_\phi$. First, we prove that

$$f_i(s) = \{\upsilon \mid \exists_{\pi \in \Pi(\upsilon(Y), s)} (|\pi| \geq i \wedge \forall_{0 \leq j \leq i}\, \pi_j \models_\upsilon \phi)\}$$

ALGORITHM 2: $Synth_{EG}\left(f_{\phi}, Y\right)$

Input: $f_{\phi} \in \left(2^{\text{ActVals}}\right)^{\mathcal{S}}$
Output: $f_{E_Y G\phi} \in \left(2^{\text{ActVals}}\right)^{\mathcal{S}}$
1: $f := f_{\phi}$; $h := \emptyset$
2: $D := f_{\phi \wedge \neg E_Y X\text{true}}$
3: **while** $f \neq h$ **do**
4: $\quad h := f$
5: $\quad f := (f_{\phi} \cap \text{parPre}^{\exists}_{Y}(h)) \cup D$
6: **end while**
7: **return** f

for each $s \in \mathcal{S}$ and $i \in \mathbb{N}$. The base case for $i = 0$ follows immediately from the definition. For the inductive step, notice that $f_{i+1} = f_{\phi} \cap \text{parPre}^{\exists}_{Y}(f_i)$ and $\text{parPre}^{\exists}_{Y}(f_i)(s) = \{\upsilon \mid \exists_{\pi \in \Pi(\upsilon(Y),s)}(|\pi| \geq i+1 \wedge \forall_{0<j\leq i+1}\ \pi_j \models_{\upsilon} \phi)\}$, from which it follows that $f_{\phi}(s) \cap \text{parPre}^{\exists}_{Y}(f_i)(s) = \{\upsilon \mid \exists_{\pi \in \Pi(\upsilon(Y),s)}(|\pi| \geq i+1 \wedge \forall_{0\leq j\leq i}\ \pi_j \models_{\upsilon} \phi)\}$. Now observe that $s \models_{\upsilon} E^{\omega}_{Y} G\phi$ if and only if $\upsilon \in \bigcap_{i\in\mathbb{N}} f_i(s)$. Notice that $f_{i+1}(s) \subseteq f_i(s)$ for all $i \in \mathbb{N}$, $s \in \mathcal{S}$, and the (common) codomain of f_i is finite. This means that the monotonic sequence $(f_i(s))_{i\in\mathbb{N}}$ stabilizes—in other words, there exists $k \in \mathbb{N}$ such that $f_i = f_k$ for all $i \geq k$. Obviously, $f_k(s) = \bigcap_{i\in\mathbb{N}} f_i(s)$ and f_k is the fixpoint of the loop and the value returned by Algorithm 1. This concludes the proof of the first case.

Let us move to the second case—that is, prove that $s \models_{\upsilon} E_Y G\phi$ if and only if $\upsilon \in Synth_{EG}(f_{\phi}, Y)(s)$. Let $\phi \in$ pmARCTL and notice that $D = f_{\phi\wedge\neg E_Y X\text{true}}$ is a constant, and $D(s) = \{\upsilon \mid (s \models_{\upsilon} \phi) \wedge \neg\exists_{s'\in\mathcal{S}}\exists_{a\in\upsilon(Y)}\ s \xrightarrow{a} s'\}$ for each $s \in \mathcal{S}$. The set $D(s)$ consists of all action valuations under which ϕ holds in s and s has no successor. By f_i, we denote the value of the f variable after the i-th run of the 3–6 loop of Algorithm 2. Additionally, let $f_0 = f_{\phi}$, as given in line 1. We prove that $f_i(s) = A^i_F(s) \cup A^i_{\infty}(s)$ for each $i \in \mathbb{N}$, $s \in \mathcal{S}$, where

$$A^i_F(s) = \{\upsilon \mid \exists_{\pi\in\Pi(\upsilon(Y),s)}(|\pi| \leq i \wedge \forall_{0\leq j\leq|\pi|}\ \pi_j \models_{\upsilon} \phi)\},$$
$$A^i_{\infty}(s) = \{\upsilon \mid \exists_{\pi\in\Pi(\upsilon(Y),s)}(|\pi| > i \wedge \forall_{0\leq j\leq i}\ \pi_j \models_{\upsilon} \phi)\}$$

that is, $A^i_F(s)$ consists of action valuations under which there exists a finite path of length smaller than or equal to i along which ϕ holds, whereas $A^i_{\infty}(s)$ contains all such valuations that along some path of length greater than i the ϕ formula holds up to its i-th state. The base case of $f_0(s)$ follows immediately from the definition of f_{ϕ} (note that $D(s) \subseteq f_{\phi}(s)$ for all $s \in \mathcal{S}$). For the inductive step, first notice that (line 5) $f_{i+1} = (f_{\phi} \cap \text{parPre}^{\exists}_{Y}(f_i)) \cup D$, and that for each $s \in \mathcal{S}$ we have $\text{parPre}^{\exists}_{Y}(f_i)(s) = \{\upsilon \mid \exists_{\pi\in\Pi(\upsilon(Y),s)}(1 \leq |\pi| \leq i+1 \wedge \forall_{0<j\leq|\pi|}\pi_j \models_{\upsilon} \phi)\} \cup \{\upsilon \mid \exists_{\pi\in\Pi(\upsilon(Y),s)}(|\pi| > i+1 \wedge \forall_{0<j\leq i+1}\pi_j \models_{\upsilon} \phi)\}$. We can now easily derive that $(f_{\phi}(s) \cap \text{parPre}^{\exists}_{Y}(f_i)(s)) \cup D(s) = A^{i+1}_F(s) \cup A^{i+1}_{\infty}(s)$. The sequence $(A^i_F(s))_{i\in\mathbb{N}}$ is increasing, and therefore it eventually stabilizes at the fixpoint $A_F(s)$, consisting of all action valuations under which ϕ holds along some finite path starting from s. The sequence $(A^i_{\infty}(s))_{i\in\mathbb{N}}$ decreases until it reaches a fixpoint $A_{\infty}(s)$, consisting of all action valuations under which ϕ holds along an infinite path beginning at s. As $Synth_{EG}(f_{\phi}, Y)(s) = A_F(s) \cup A_{\infty}(s)$, this concludes the proof. $\square$

From Lemma 3.4, $f_{E^{\omega}_Y G\phi} = Synth_{E^{\omega}G}(f_{\phi}, Y)$, $f_{E_Y G\phi} = Synth_{EG}(f_{\phi}, Y)$.

3.1.4. Until Modality. Similarly as in the case of CTL, the equivalence $E_Y(\phi U \psi) \equiv \psi \vee (\phi \wedge E_Y X E_Y(\phi U \psi))$ motivates the following fixpoint algorithm.

ALGORITHM 3: $Synth_{EU}\left(f_\phi, f_\psi, Y\right)$

Input: $f_\phi, f_\psi \in \left(2^{\mathrm{ActVals}}\right)^{\mathcal{S}}$
Output: $f_{E_Y(\phi U \psi)} \in \left(2^{\mathrm{ActVals}}\right)^{\mathcal{S}}$
1: $f := f_\psi; h := \emptyset$
2: **while** $f \neq h$ **do**
3: $\quad h := f;\;\; f := f_\psi \cup (f_\phi \cap \mathrm{parPre}^{\exists}_{Y}(h))$
4: **end while**
5: **return** f

Lemma 3.5. *For each $s \in \mathcal{S}$, $\phi, \psi \in$ pmARCTL, $Y \in$ ActVars, and $\upsilon \in$ ActVals, the following condition holds: $s \models_\upsilon E_Y(\phi U \psi)$ if and only if $\upsilon \in Synth_{EU}(f_\phi, f_\psi, Y)(s)$.*

Proof. For now, assume that the while loop 2–4 of Algorithm 3 is infinite (i.e., put true in place of the $f \neq h$ condition). Let f_i denote the value of the variable f after the i-th run of the loop, and let $f_0 = f_\psi$ (as given in line 1). First, let us prove that $f_i(s) = \{\upsilon \mid \exists_{\pi \in \Pi(\upsilon(Y),s)} \exists_{j \le i}(|\pi| \ge i \wedge \pi_j \models_\upsilon \psi \wedge \forall_{0 \le k < j} \pi_k \models_\upsilon \phi)\}$ for each $s \in \mathcal{S}$. In the case of f_0, the preceding equality follows immediately from the definition of f_ψ. For the inductive step, notice that due to the substitution in line 3, we have that $f_{i+1} = f_\psi \cup (f_\phi \cap \mathrm{parPre}^{\exists}_{Y}(f_i))$. Now observe that $\mathrm{parPre}^{\exists}_{Y}(f_i)(s) = \{\upsilon \mid \exists_{s' \in \mathcal{S}} \exists_{a \in \upsilon(Y)}(s \xrightarrow{a} s' \wedge \upsilon \in f_i(s'))\} = \{\upsilon \mid \exists_{s' \in \mathcal{S}} \exists_{a \in \upsilon(Y)}(s \xrightarrow{a} s' \wedge \exists_{\pi \in \Pi(\upsilon(Y),s')} \exists_{j \le i}(\pi_j \models_\upsilon \psi \wedge \forall_{0 \le k < j} \pi_k \models_\upsilon \phi))\} = \{\upsilon \mid \exists_{\pi \in \Pi(\upsilon(Y),s)} \exists_{0 < j \le i+1}(\pi_j \models_\upsilon \psi \wedge \forall_{0 < k < j} \pi_k \models_\upsilon \phi)\}$, and therefore $f_\phi(s) \cap \mathrm{parPre}^{\exists}_{Y}(f_i)(s) = \{\upsilon \mid \exists_{\pi \in \Pi(\upsilon(Y),s)} \exists_{0 < j \le i+1}(\pi_j \models_\upsilon \psi \wedge \forall_{0 \le k < j} \pi_k \models_\upsilon \phi)\}$. From the preceding, we finally have the following:

$$f_\psi(s) \cup (f_\phi(s) \cap \mathrm{parPre}^{\exists}_{Y}(f_i)(s)) = \{\upsilon \mid \exists_{\pi \in \Pi(\upsilon(Y),s)} \exists_{0 \le j \le i+1}(\pi_j \models_\upsilon \psi \wedge \forall_{0 \le k < j} \pi_k \models_\upsilon \phi)\}.$$

Now observe that $s \models_\upsilon E_Y \phi U \psi$ if and only if $\upsilon \in \bigcup_{i=0}^{\infty} f_i(s)$. As $f_i(s) \subseteq f_{i+1}(s)$ for all $i \in \mathbb{N}$, we have that if the fixpoint in line 2 is reached for some k-th run of the loop, then $f_k(s) = \bigcup_{i=0}^{\infty} f_i(s)$. The fixpoint, however, is always reached and the algorithm stops, because there is only a finite number of functions in $(2^{\mathrm{ActVals}})^{\mathcal{S}}$ and $(f_i(s))_{i \in \mathbb{N}}$ is a monotonic sequence of sets. □

Following the chosen convention, we have $f_{E_Y(\phi U \psi)} = Synth_{EU}(f_\phi, f_\psi, Y)$.

3.1.5. Overall Algorithm. Algorithm 4 provides the entry point for the computation of the f_ϕ function, given a formula $\phi \in$ pmARCTL.

ALGORITHM 4: $Synth_{full}(\phi)$

Input: $\phi \in$ pmARCTL
Output: $f_\phi \in \left(2^{\mathrm{ActVals}}\right)^{\mathcal{S}}$
if $\phi = E_Y X \psi$ **then**
$\quad$ **return** $\mathrm{parPre}^{\exists}_{Y}(Synth_{full}(\psi))$
else if $\phi = E^r_Y G \psi$ where $r \in \{\omega, \epsilon\}$ **then**
$\quad$ **return** $Synth_{E^r G}\left(Synth_{full}(\psi), Y\right)$
else if $\phi = E_Y(\xi U \psi)$ **then**
$\quad$ **return** $Synth_{EU}\left(Synth_{full}(\xi), Synth_{full}(\psi), Y\right)$
else {propositional and nonparametric modalities omitted for simplicity}
$\quad$ **return** f_ϕ
end if

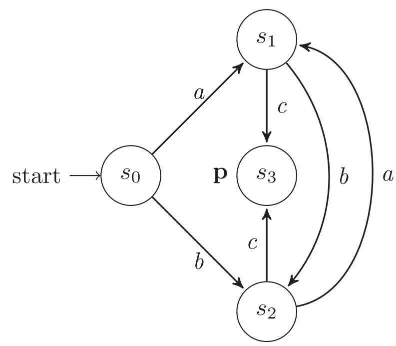

Fig. 2. A counterexample for the action monotonicity.

The validity of the results obtained with Algorithm 4 is summarized by the following theorem.

THEOREM 3.6. *For each model $\mathcal{M}$, formula $\phi \in$ pmARCTL, state $s \in \mathcal{S}$, and action valuation $\upsilon \in$ ActVals, we have $\mathcal{M}, s \models_\upsilon \phi$ if and only if $\upsilon \in Synth_{full}(\phi)(s)$.*

PROOF. Follows immediately from Lemmas 3.2 through 3.5. □

3.2. Synthesis of Minimal Sets of Constraints

Section 3.1 shows how to synthesize the complete set of action valuations under which a given formula holds. As we present in what follows, in some special cases the sought valuations can be represented as all supersets of a certain family of sets. This requires restricting the language of pmARCTL such that the operator $E_\alpha G$ is disallowed, as it is not distributive over unions and intersections of α's (see the following example).

Example 3.7. Consider the MTS from Figure 2, where $\mathcal{V}_s(s_3) = \{p\}$, $\mathcal{V}_s(s_i) = \emptyset$ for all $i \neq 3$, and $\mathcal{A} = \{a,b,c\}$. It is not difficult to see that $s_0 \models A_{\{a,c\}}Fp$ and $s_0 \models A_{\{b,c\}}Fp$. On the other hand, $s_0 \not\models A_{\{a,c\}\cup\{b,c\}}Fp$ (due to the existence of the loop between s_1 and s_2) and $s_0 \not\models A_{\{a,c\}\cap\{b,c\}}Fp$ (as $p \notin \mathcal{V}_s(s_0)$). Similarly, we have both $s_0 \models E_{\{a\}}G\neg p$ and $s_0 \models E_{\{c\}}G\neg p$, but $s_0 \not\models E_{\{a,c\}}G\neg p$.

Let $\upsilon, \upsilon' \in$ ActVals. We write $\upsilon \prec \upsilon'$ if $\upsilon(Y) \subseteq \upsilon'(Y)$ for all $Y \in$ ActVars. For a given property ϕ, if $(\mathcal{M}, s^0 \models_\upsilon \phi$ and $\upsilon \prec \upsilon')$ implies $\mathcal{M}, s^0 \models_{\upsilon'} \phi$ for all $\upsilon, \upsilon' \in$ ActVals and all models $\mathcal{M}$, then ϕ is called *action monotone*. The language is *action monotone* if all of its formulas are as well.

Definition 3.8 (pmEARCTL *syntax*). The language of *parametric existential Action-Restricted Computation Tree Logic* is defined by the following grammar:

$$\phi ::= p \mid \neg p \mid \phi \vee \phi \mid \phi \wedge \phi \mid E_\alpha X\phi \mid E^\omega_\alpha G\phi \mid E_\alpha(\phi\ U \phi),$$

where $p \in \mathcal{PV}$, $\alpha \in$ ActSets $\cup$ ActVars.

Observe that if $\phi, \psi \in$ pmEARCTL and $r \in \{\omega, \epsilon\}$, then the derived modalities $E^\omega_\alpha X\phi$, $E^\omega_\alpha(\phi\ U \psi)$, and $E^r_\alpha F\phi$ also belong to pmEARCTL.

LEMMA 3.9. *The language of pmEARCTL is action monotone.*

PROOF. The proof proceeds by the induction on the structure of $\phi \in$ pmEARCTL. Let us assume that $s \models_\upsilon \phi$ and $\upsilon \prec \upsilon'$. The base case of $\phi = p \in \mathcal{PV}$ and the cases of the conjunction, disjunction, and the negation of a proposition are straightforward. If $\phi \in \{E_\alpha X\phi, E_\alpha(\phi U \phi)\}$, then it suffices to observe that each path $\pi \in \Pi(\upsilon(\alpha), s)$ is a prefix of some path in $\Pi(\upsilon'(\alpha), s)$ and apply the inductive assumption. Similarly, if $\phi = E^{\omega}_{\alpha} G\phi$, notice that $\Pi^{\omega}(\upsilon(\alpha), s) \subseteq \Pi^{\omega}(\upsilon'(\alpha), s)$ and apply the inductive assumption.

Let $\Upsilon \subseteq \text{ActVals}$, and let $minVals(\Upsilon)$ denote the set of the valuations in Υ minimal with respect to $\prec$. Formally,

$$minVals(\Upsilon) = \{\upsilon \in \Upsilon \mid \forall_{\upsilon' \in \Upsilon}(\upsilon' \prec \upsilon \Rightarrow \upsilon' = \upsilon)\}.$$

By Lemma 3.9, for $\phi \in$ pmEARCTL the set $f_{\phi}(s^0)$ can be generated by (typically much smaller) $minVals(f_{\phi}(s^0))$.

In our approach, parameter synthesis is performed by means of manipulations of Boolean formulas. The process of building such formulas does not differ much from the nonparametric case [Baier and Katoen 2008], with the main difference consisting of the encoding of action valuations. With some notational abuse, let us treat the set of the actions $\mathcal{A}$ as propositional variables. Propositional formulas over $\mathcal{A}$ can be perceived as indicator functions for the subsets of ActSets—that is, let α be a propositional formula over $\mathcal{A}$ and

$$[\alpha] = \{\{a_1, \ldots, a_m\} \subseteq \mathcal{A} \mid\ \models \alpha[a_1/\text{true}, \ldots, a_m/\text{true}]\}.$$

The formula α encodes the set of all subsets of $\mathcal{A}$ that make α hold when all of their elements are set to true. It is easy to show that for each $A \subseteq$ ActSets, there exists a formula α such that $[\alpha] = A$. To encode the subsets of ActVals, we introduce a set of fresh propositional variables $\mathcal{A}_{\text{ActVars}} = \bigcup_{Y \in \text{ActVars}}\{a_Y \mid a \in \mathcal{A}\}$—that is, for each variable Y, we introduce a copy of each element of $\mathcal{A}$ subscripted by Y. We use propositional formulas over $\mathcal{A}_{\text{ActVars}}$ to represent sets of action valuations as follows. Let β be a propositional formula over $\mathcal{A}_{\text{ActVars}}$, then

$$[\beta]_{\text{ActVars}} = \{f \in \text{ActVals} \mid \exists_{\upsilon \in [\beta]} \forall_{Y \in \text{ActVars}}(a \in f(Y) \iff a_Y \in \upsilon)\}.$$

It is straightforward to prove that for each set of functions $F \subseteq$ ActVals, there exists a propositional formula over $\mathcal{A}_{\text{ActVars}}$, denoted by $enc(F)$, such that $[enc(F)]_{\text{ActVars}} = F$.

Example 3.10. Let $\mathcal{A} = \{a, b, c\}$ and ActVars $= \{Y, Z\}$, and let $\beta = (a_Y \wedge b_Y \wedge a_Z \wedge b_Z \wedge c_Z) \vee (\neg a_Z \wedge \neg b_Z \wedge c_Z)$. As we have $[\beta] = \{\{a_Y, b_Y, a_Z, b_Z, c_Z\}, \{a_Y, b_Y, c_Y, a_Z, b_Z, c_Z\}\} \cup \{A \cup \{c_Z\} \mid A \subseteq \{a_Y, b_Y, c_Y\}\}$, then $f \in [\beta]_{\text{ActVars}}$ if and only if (1) $\{a, b\} \subseteq f(Y)$ and $f(Z) = \{a, b, c\}$, or (2) $f(Z) = \{c\}$ and $f(Y)$ is any subset of $\mathcal{A}$. There are $2 + 2^3$ functions in $[\beta]_{\text{ActVars}}$.

Let κ be a conjunction of literals (i.e., propositions or their negations) from $\mathcal{A}$. We assume that the empty conjunction of literals is equivalent to false. If $\models \kappa \Rightarrow \beta$, then κ is called an *implicant* of β. Notice that the empty conjunction is an implicant of each formula. An implicant κ is called *prime* if it does not subsume a shorter implicant of β [Quine 1952]. A set *Cov* of prime implicants of β is called its *prime covering* if and only if $(\bigvee_{\kappa \in Cov} \kappa) \equiv \beta$. Let $val(\kappa)$ denote such action valuation that $val(\kappa)(Y) = \{a \in \mathcal{A} \mid a_Y \in \kappa\}$ for each $Y \in$ ActVars.

Lemma 3.11. *Let* $\phi \in$ pmEARCTL *and Cov be a prime covering of* $enc(f_{\phi}(s^0))$. *Then, Cov is unique, and* $minVals(f_{\phi}(s^0)) = \bigcup_{\kappa \in Cov}\{val(\kappa)\}$.

Proof. Notice that from the definition of prime covering, we have $\bigcup_{\kappa \in Cov}[\kappa]_{\text{ActVars}} = f_{\phi}(s^0)$. In addition, observe that if $\kappa \in Cov$ contains a negative literal a_Y, then none of the valuations of $[\kappa]_{\text{ActVars}}$ would assign a set of actions containing a to the variable Y. On the other hand, we know from Lemma 3.9 that pmEARCTL is action monotone, and thus if κ' denotes κ, where the literal is removed, then $[\kappa]_{\text{ActVars}} \subseteq [\kappa']_{\text{ActVars}} \subseteq f_{\phi}(s^0)$. As $\models \kappa' \Rightarrow enc(f_{\phi}(s^0))$ and κ subsumes κ', we get a contradiction with the assumption that κ is a prime implicant, and therefore κ can only contain positive literals. This means that $enc(f_{\phi}(s^0))$ is monotone (i.e., contains only positive literals), and hence *Cov*

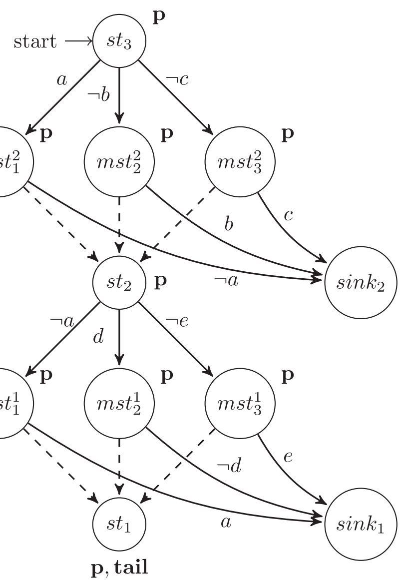

Fig. 3. A model for 3SAT formula $\mu = (a \vee \neg b \vee \neg c) \wedge (\neg a \vee d \vee \neg e)$. The dashed arcs are labeled with jmp.

is unique [Goldsmith et al. 2008]. As $val(\kappa)$ is the smallest element of $[\kappa]_{\text{ActVars}}$ with respect to $\prec$, the proof is complete. □

From Lemma 3.11, it follows that to build a set of minimal valuations for any formula of pmEARCTL, it suffices to collect prime implicants of its encoding. There are many known methods for obtaining the set of prime implicants of a Boolean function [Coudert et al. 1993]. In our work, we employ the facilities built in the CUDD BDD package to iterate over all prime implicants. The method can be in practice generalized as follows: if $enc(f_\phi(s^0))$ for $\phi \in$ pmARCTL does not contain negations, then ϕ is action monotone and $minVals(f_\phi(s^0)) = \bigcup_{\kappa \in Cov}\{val(\kappa)\}$, where Cov is the prime covering of $enc(f_\phi(s^0))$.

3.3. Complexity

Let us consider the question of whether for a given model $\mathcal{M}$ with the initial state s^0 and a formula $\phi \in$ pmARCTL, there exists an action valuation υ such that $\mathcal{M}, s^0 \models_\upsilon \phi$. It is a well-defined decision problem, called the *emptiness problem* for pmARCTL.

THEOREM 3.12. *The emptiness problem for pmARCTL is NP-complete.*

PROOF. The proof follows via reduction from 3SAT (Figure 3). Let $\mathcal{PV}$ be a set of propositional variables, and let $\mathcal{PL} = \mathcal{PV} \cup \{\neg p \mid p \in \mathcal{PV}\}$ be the set of literals over $\mathcal{PV}$. Let $n \in \mathbb{N}$, and let $\mu = (a_1^1 \vee a_2^1 \vee a_3^1) \wedge \cdots \wedge (a_1^n \vee a_2^n \vee a_3^n)$ be a propositional formula in 3CNF, where $a_j^i \in \mathcal{PL}$ for all $1 \le i \le n$, $1 \le j \le 3$.

We build a model $\mathcal{M}$ and a formula $\phi_\mu \in$ pmARCTL such that there is a correspondence between the valuations satisfying μ and action valuations satisfying ϕ_μ in $\mathcal{M}$. In what follows, we fix a proposition $p \in \mathcal{PV}$. Let $C_i = a_1^i \vee a_2^i \vee a_3^i$ denote the i-th clause of μ for each $1 \le i \le n$. With a slight notational abuse (i.e., the literals from $\mathcal{PL}$ are treated as actions, and double negations are reduced whenever possible) let

—$\mathcal{S}_i = \{st_i, st_{i+1}, sink_i, mst_1^i, mst_2^i, mst_3^i\}$,
—$\mathcal{A}_i = \{a_1^i, a_2^i, a_3^i, \neg a_1^i, \neg a_2^i, \neg a_3^i\}$,
—$\mathcal{T}_i = \bigcup_{j=1}^{3}\{(st_{i+1}, a_j^i, mst_j^i), (mst_j^i, \neg a_j^i, sink_i), (mst_j^i, jmp, st_i)\}$.

Now let $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{V}_s)$, where $\mathcal{S} = \bigcup_{i=1}^{n} S_i$, $s^0 = st_{n+1}$, $\mathcal{A} = \mathcal{PL} \cup \{jmp\}$, $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$, and $\mathcal{V}_s(s) = \{p\}$ for all $s \notin \bigcup_{i=1}^{n}\{sink_i\} \cup \{st_{n+1}\}$ and $\mathcal{V}_s(st_1) = \{p, tail\}$.

Consider the formula $\phi_\mu = A_Y Gp \wedge E_Y F\text{tail}$, and notice that $st_{n+1} \models_\upsilon \phi_\mu$ if and only if (1) $\upsilon(Y)$ contains the action jmp, (2) for each $1 \leq i \leq n$ the set $\upsilon(Y)$ contains at least one action a^i_j such that $(st_{i+1}, a^i_j, mst^i_j) \in \mathcal{T}$ for some $1 \leq j \leq 3$, and (3) the set $\upsilon(Y)$ does not contain a transition labeled with literal and transition labeled with its negation (this would create a path leading to $sink_i$, for some $1 < i \leq n$, which is not labeled by p). For an action valuation υ satisfying conditions 1 through 3, let ω_υ be a valuation (of propositionals) such that $\omega_\upsilon(a) = \text{true}$ if and only if $a \in \upsilon(Y)$ for each $a \in \mathcal{A}$, and observe that $\omega_\upsilon \models \mu$. Conversely, let ω be such that $\omega \models \mu$, define the action valuation υ_ω such that $jmp \in \upsilon_\omega(Y)$ and $a \in \upsilon_\omega(Y)$ if and only if $\omega(a) = \text{true}$ for each $a \in \mathcal{A}$, and observe that $\upsilon_\omega \models \phi_\mu$.

Note that the presented reduction is polynomial. On the other hand, the ARCTL verification can be attained in polynomial time, and there is a finite number of possible valuations; therefore, the emptiness problem can be solved in polynomial time by a nondeterministic Turing machine. $\square$

In view of the preceding information, it is not surprising that the time complexity of the presented algorithms is high. Recall that $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{V}_s)$, and let $\phi \in$ pmARCTL contain k free variables. To estimate the time complexity of $\text{parPre}^{\exists}_{Y}(f_\phi)$, let us fix $s, s' \in \mathcal{S}$ and let $f_\phi(s') = \{\upsilon_1, \ldots, \upsilon_k\}$. Let $(s, a, s') \in \mathcal{T}$, and notice that $\text{parPre}^{\exists}_{Y}(f_\phi)(s)$ gathers such valuations from $f_\phi(s')$ that $a \in \upsilon_i(Y)$. As $|f_\phi(s')|$ can be at most of $2^{|\mathcal{A}|k}$ size, the worst-case complexity of $\text{parPre}^{\exists}_{Y}(f_\phi)$ is in $O(|\mathcal{S}| + |\mathcal{T}| \cdot 2^{|\mathcal{A}|k})$. The proposed algorithms are based on fixed-point computations in the space consisting of pairs composed of a state and a set of action valuations. For a fixed state, its associated set of action valuations is altered by exclusively adding or removing new elements. As there can be at most $2^{|\mathcal{A}|k}$ such changes for a given state and the preimage computation is the main operation in the body of each loop, the total complexity of the parameter synthesis is in $O(|\mathcal{S}|^2 \cdot 2^{|\mathcal{A}|k} + |\mathcal{S}||\mathcal{T}| \cdot 2^{|\mathcal{A}|2k})$. (Note that k corresponds to the number of the parametric modalities in ϕ).

In general, the problem of computing the full set of minimal action valuations is difficult as well. The time complexity of selection of minimal elements of a partially ordered set is polynomial with respect to the size of the set [Daskalakis et al. 2011]; in our case, however, the size of the set (i.e., ActVals) is exponential with respect to the number of actions. Concerning the proposed prime implicant-based technique, it is known that there is no output-polynomial time algorithm for finding all prime implicants of a given monotone function unless $P = NP$ [Goldsmith et al. 2008]. Despite these obstacles, the minimization algorithm performs very well, as shown in the experimental part of this work.

The complexity of ARCTL model checking is equal to that of CTL verification, and therefore the complexity of the naive approach, based on enumerative checking of all possible action valuations, is in $O((|\mathcal{S}|+|\mathcal{T}|)k \cdot 2^{|\mathcal{A}|k})$. Note that in the naive approach, the worst-case complexity is equal to the expected one. The symbolic verification of (nonparametric) ARCTL is in PSPACE, similarly as in the case of CTL [Schnoebelen 2002]; in this case, however, the practical complexity is well documented to be lower, and symbolic model checkers typically outperform nonsymbolic verification tools. However, even if efficient symbolic verification methods are used for the verification of instantiations of ARCTL formulas in the naive approach, still $2^{|\mathcal{A}|k}$ cases need to be separately analyzed. As we show in the next section, our symbolic algorithm for pmARCTL substantially outperforms the naive approach.

4. IMPLEMENTATION AND EVALUATION

In this section, we present an evaluation of our implementation of the theory presented in this article. We use parallel compositions of MTSs as models, with disjunctive location labeling—that is, a given vector of locations is labeled with a proposition p if *any* of its components is labeled with p.

Definition 4.1. Let $I = \{1, \ldots, k\}$ for some $k \in \mathbb{N}$ be a finite set of indices, and for each $i \in I$, let $\mathcal{M}_i = (\mathcal{S}_i, s^0{}_i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{V}_{si})$ be an MTS. We define the *product with the disjunctive location labeling* of a network $\{\mathcal{M}_i\}_{i\in I}$ as an MTS $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{V}_s)$ such that $\mathcal{S} = \prod_{i\in I} \mathcal{S}_i$, and $s^0 = (s^0{}_1, \ldots, s^0{}_k)$, and $\mathcal{A} = \bigcup_{i\in I} \mathcal{A}_i$, and the transition relation $\mathcal{T}$ satisfies

—$(l_1, \ldots, l_k) \xrightarrow{a} (l'_1, \ldots, l'_k)$ if and only if for each $i \in I$, we have $l_i \xrightarrow{a} l'_i$ if $a \in \mathcal{A}_i$ and $l_i = l'_i$ otherwise,

and the labeling $\mathcal{V}_s$ such that for each proposition $p \in \mathcal{PV}$: $p \in \mathcal{V}_s((l_1, \ldots, l_k))$ if and only if $p \in \mathcal{V}_{si}(l_i)$ for some $i \in I$.

We present a preliminary evaluation of feasibility of the parameter synthesis of action valuations performed on two scalable examples followed by an analysis of Peterson's algorithm for mutual exclusion. As a companion to this work, we release a freely available open-source program SPATULA [Knapik 2014], which implements the parameter synthesis methods. The tool uses CUDD [Somenzi 2012] package providing operations on reduced ordered binary decision diagrams (BDDs) to represent the state space and action valuations. SPATULA allows for modeling the input systems in a simple description language. To the best knowledge of the authors, there is no other tool allowing the parameter synthesis for pmARCTL, and therefore for the sake of comparison, we implemented a *naive* engine, which enumerates all possible action valuations and performs nonparametric verification of resulting substitutions. We also record the speedup times of symbolic parametric synthesis versus brute-force parametric verification, following in this way the methodology presented in Classen et al. [2011]. SPATULA allows division of the actions into two disjoint sets—fixed and switchable—and to synthesize only those valuations that contain all fixed actions.

The memory usage results for the naive cases are omitted from the figures, as they are very similar to the results for the parametric ones up to the defined timeout, which was set to 15 minutes. The experiments have been performed on an Intel P6200 dual-core 2.13GHz machine with 3.5GB RAM, running on a Linux operating system. It should be noted that the alpha version of SPATULA, presented in Knapik et al. [2014], did not implement the dynamic reordering of the BDD variables. In the current version, the dynamic reordering is enabled in CUDD, which makes the speedup of the parametric approach even more noticeable.

4.1. Scalability on Faulty Train-Gate-Controller

The system presented in Figure 4 is a version of the classical model from Alur et al. [1993] with the modifications inspired by Belardinelli et al. [2011]. It consists of k trains and the controller monitoring the access to the tunnel.

It is required that there is at most one train at a time in the tunnel. For $1 \leq i \leq k$, the i-th train can be either outside the tunnel ($\mathbf{out}_i$), approaching the tunnel, or inside the tunnel ($\mathbf{in}_i$). If the controller is in the **red** state, then no train is allowed to enter the tunnel; otherwise, if the controller is in the **green** state, then the trains are allowed to enter the tunnel. The j-th train is assumed to be faulty and its communication with the controller is malfunctioning—that is, it can perform a faulty action that does not change the controller state when entering the tunnel (in^F_j) or leaving the

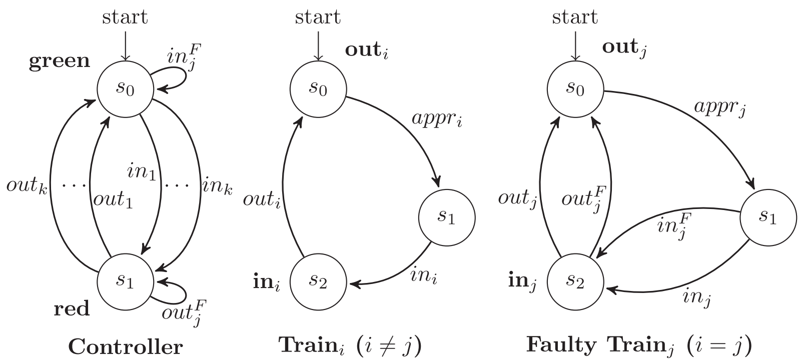

Fig. 4. Faulty Train-Gate-Controller.

tunnel (out_j^F). The network is described using the SPATULA's modeling language as shown in Figure 5. The language essentially is a graph network description language with a set of convenient, C-inspired flow control constructs.

We have tested the following properties:

(1) $\psi_1 = A_Y G(\neg \bigvee_{1 \le i < l \le k} (\mathrm{in_i} \wedge \mathrm{in_l})) \wedge \bigwedge_{1 \le i \le k} E_Y F \mathrm{in_i}$, expressing that it is not possible for any pair of trains to be in the tunnel at the same time, and each train will eventually be in the tunnel;
(2) $\psi_2 = E_Y F A_Z G((\bigwedge_{1 \le i \le k} \neg \mathrm{in_i}) \wedge \mathrm{green})$, expressing that it is possible for the system to execute actions from Y in such a way that at some state, in all possible executions of the system that use only the actions from Z, all trains remain outside the tunnel while the controller remains in the **green** state;
(3) $\psi_3 = E_Y^{\omega} G E F_Y (\mathrm{in_1} \wedge \mathrm{in_j})$, expressing the existence of an execution such that the first and the faulty train are infinitely often simultaneously present in the tunnel; and
(4) $\psi_4 = E_Y^{\omega} G E F_Z (\mathrm{in_1} \wedge \mathrm{in_j})$, expressing that there exists an execution of the system labeled by actions from Y such that the state with the first and the faulty train present in the tunnel is infinitely often reachable via actions from Z. Note that this is a version of ψ_3 with allowed two parameters.

The formulas ψ_3 and ψ_4 belong to pmEARCTL. The minimization of the constraints obtained in these cases was performed similarly as in Section 3.2. In both cases, we assume that the first train is not faulty. The parametric approach typically outperforms the naive one (Figure 6); this is especially noticeable when comparing the speedup of the method in Table I.

The observed state space explosion combined with the exponential blowup of the space of the solutions makes the iterative approach infeasible for the considered properties, as it is able to compute the results for the systems only with up to five trains. The parametric approach is clearly superior, as the results for the system consisting of five trains are obtained in less than 10 seconds, and within the specified time bound, the tool managed to obtain the results for the systems with up to 28 trains, with the state space of size $\approx 4.6 \cdot 10^{13}$ and the space of possible solutions of size $\approx 2^{172}$ (for the properties with two free variables). Notice that the space of solutions (correct (SAT) action valuations in Figure 6) grows at an exponential rate with respect to the number of nodes. The minimization of the constraints took less than 1 second in both cases, and therefore it is omitted from Figure 6.

```
module Controller:
  trainsNo = k;
  faultyTrainNo = j;

  /* correct behavior */
  bloom("s0");
  mark_with("s0", "initial");
  mark_with("s0", "green");
  bloom("s1");
  mark_with("s1", "red");
  ctr = 1;
  while(ctr <= trainsNo) {
     outlabel = "out" + ctr;
     inlabel  = "in" + ctr;
     join_with("s0", "s1", inlabel);
     join_with("s1", "s0", outlabel);
     ctr = ctr + 1;
  }

  /* faulty behavior */
  inlabelF = "inF" + faultyTrainNo;
  outlabelF = "outF" + faultyTrainNo;
  join_with("s0", "s0", inlabelF);
  join_with("s1", "s1", outlabelF);

module Train_i:
  trainNo = i;
  faultyTrainNo = j;

  bloom("out");
  mark_with("out", "initial");
  bloom("approaching");
  bloom("in");

  outlabel = "out" + trainNo;
  inlabel  = "in" + trainNo;
  apprlabel  = "appr" + trainNo;
  join_with("in", "out", outlabel);
  join_with("out", "approaching", apprlabel);
  join_with("approaching", "in", inlabel);

  /* faulty behavior */
  if(trainNo == faultyTrainNo) {
    inlabelF = "inF" + faultyTrainNo;
    outlabelF = "outF" + faultyTrainNo;
    join_with("in", "out", outlabelF);
    join_with("approaching", "in", inlabelF);
  }

  /* label the nodes */
  outmark = "Train" + trainNo + "out";
  inmark = "Train" + trainNo + "in";
  apprmark = "Train" + trainNo + "approaching";
  mark_with("out", outmark);
  mark_with("in", inmark);
  mark_with("approaching", apprmark);
```

Fig. 5. SPATULA template for Controller and Train$_i$.

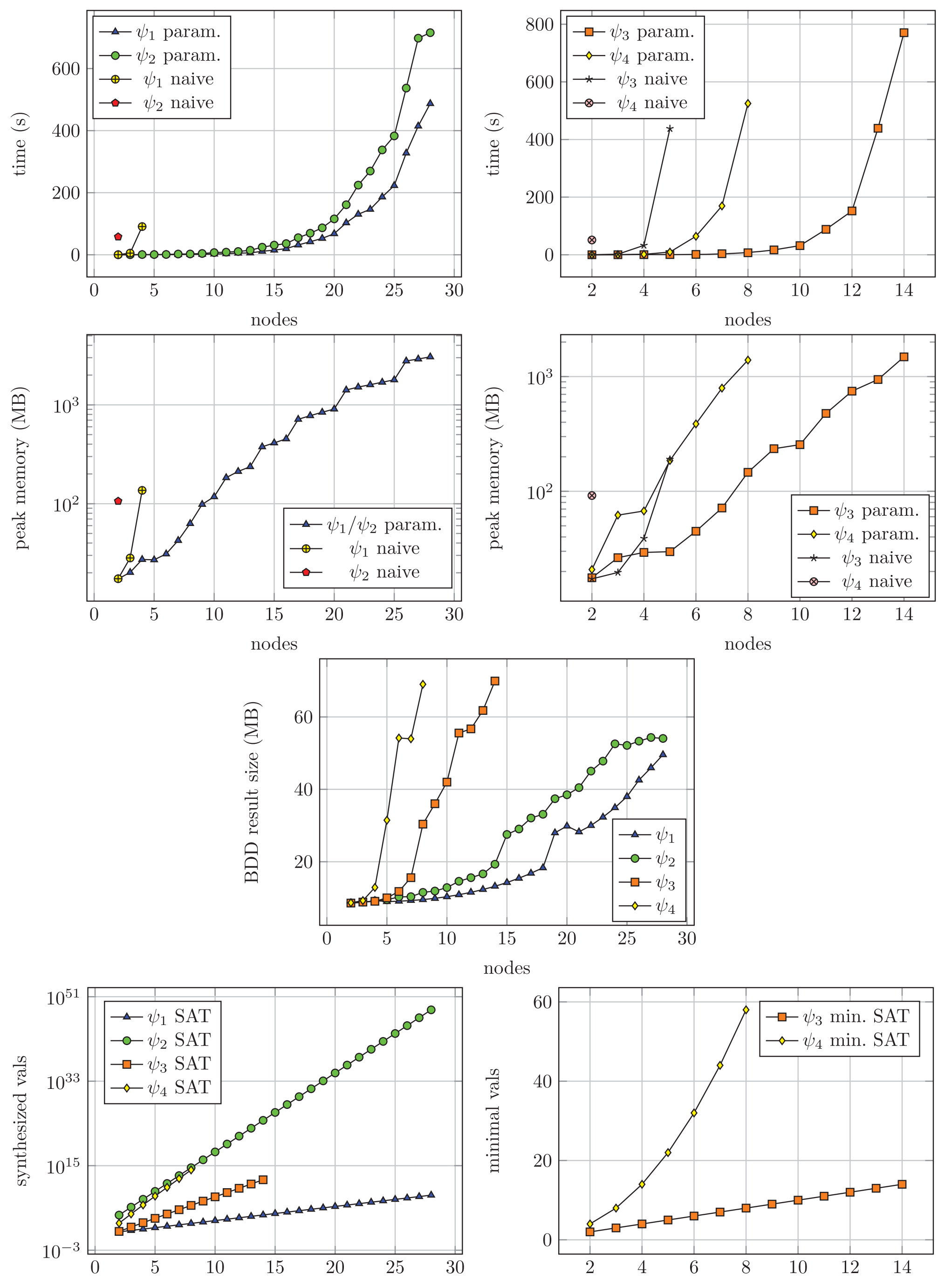

Fig. 6. Faulty Train-Gate-Controller results.

Table I. Speedup for Faulty Train-Gate-Controller

	Speedup (Naive/Parametric Time)				
Property	2 trains	3 trains	4 trains	5 trains	6 trains
ψ_1	1, 250.0	263.93	94.19011	>5,483.020†	>2,711.61†
ψ_2	3, 251.31	>7,999.64†	>1,358.39†	>1,379.55†	>992.42†
ψ_3	0.7	17.90	97.07	722.35	>740.13†
ψ_4	10.06	20.07	211.84	>96.64†	>14.04†

†, naive approach exceeds set timeout of 15 minutes.

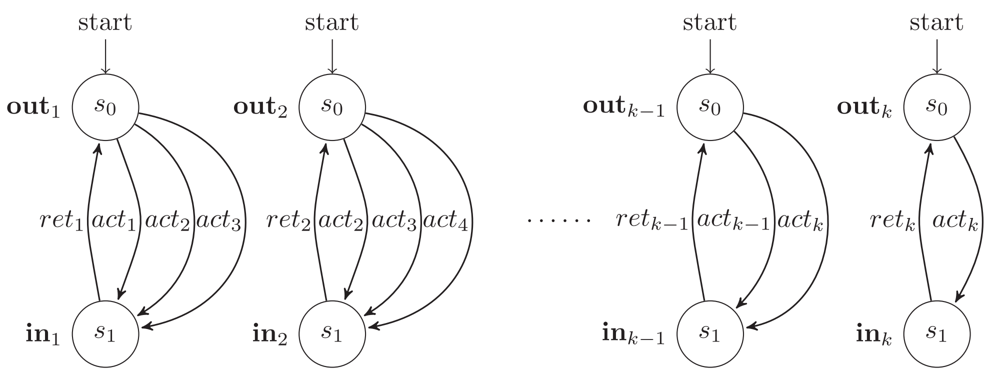

Fig. 7. Generic pipeline paradigm network.

Table II. Speedup for the Generic Pipeline Paradigm

	Speedup (Naive/Parametric Time)						
Property	4 proc.	5 proc.	6 proc.	7 proc.	8 proc.	9 proc.	10 proc.
ϕ_1	5.46	20.04	49.73	76.18	380.68	1306.71	>3687.69†
ϕ_2	12.2	29.56	149	204.15	977.92	2213	>5424.1†
ϕ_3	6.52	11.6	22.53	169.41	880.06	1468.03	>1640.05†
ϕ_4	345.89	>429†	>83.72†	>7.6†	>1.24†		

†, the naive approach exceeded set timeout of 15 minutes.

4.2. Scalability on the Generic Pipeline Paradigm

The network in Figure 7, inspired by the generic pipeline paradigm [Peled 1993], consists of $k > 3$ processing nodes (Table II). A node can synchronize via shared actions with up to four other surrounding ones, depending on its position in the pipeline (if $1 \leq i \leq k$, then the i-th node admits all actions from the set $\{ret_i, act_i, act_{min(i+1,k)}, act_{min(i+2,k)}\}$).

We have tested the following properties:

(1) $\phi_1 = A_Y F(\bigwedge_{1\leq i\leq \lfloor \frac{k}{2} \rfloor} \mathrm{out_i} \wedge \bigwedge_{\lceil \frac{k}{2} \rceil < j \leq k} \mathrm{in_j})$, describing the unavoidability of the configuration in which the first half of the nodes is in **out** and the other half is in **in** states;

(2) $\phi_2 = E_Y F A_Y G(\bigwedge_{1\leq i\leq \lceil \frac{k}{2} \rceil} \mathrm{in_{2i-1}} \wedge \bigwedge_{1\leq i\leq \lfloor \frac{k}{2} \rfloor} \mathrm{out_{2i}})$, describing the configuration such that the odd nodes are in their **in** and the even are in their **out** states becomes persistent starting from some state in the future;

(3) $\phi_3 = E^{\omega}_Y G E_Y F(\bigwedge_{1\leq i\leq k} \mathrm{in_i})$, expressing that the configuration with all nodes simultaneously in their **in** states is Y-reachable infinitely often; and

(4) $\phi_4 = E^{\omega}_Y G E_Z F(\bigwedge_{1\leq i\leq k} \mathrm{in_i})$, a version of ϕ_3 with two parameters.

The parametric synthesis for ϕ_1, ϕ_2, and ϕ_3 was interrupted after reaching the set time limit, and the synthesis for ϕ_4 has been stopped due to exceeding 3.5GB of memory usage (Figure 8). The naive synthesis has been stopped due to timeout in all cases.

In the case of ϕ_1, ϕ_2, and ϕ_3, the naive approach becomes infeasible for more than 9 nodes, whereas the parametric approach managed to compute the results for ϕ_1, ϕ_2 up to 48 nodes. For the formula ϕ_4, the timeout of the naive approach is almost immediate (i.e., reached at five processes), whereas the parametric approach allows computation of solutions up to eight processes. The model with k nodes consists of 2^k states and there are $2k$ separate actions, which gives $\approx 2^{2k}$ possible action valuations for the single-parameter formulas and $\approx 2^{4k}$ for two-parameter properties. The huge size of the space of the valuations explains why the enumerative approach quickly becomes infeasible. On the other hand, the symbolic fixed-point verification scales reasonably well.

4.3. Peterson's Algorithm Analysis

In this section, we analyze a solution to the mutual exclusion problem for two processes, which are proposed in Peterson [1981]. For reference, we include a pseudocode for Peterson's solution to the mutual exclusion problem for two processes (Figure 9). Both processes from Figure 9 are placed in infinite loops, omitted from the figure for clarity.

The algorithm employs three binary variables: B_0, B_1, B_2, where B_0, B_1 are used as red/green lights allowing a process $0, 1$ (respectively) to enter the critical section; the entry of the i-th process can also be granted by setting the B_2 variable to i. The process 0 can read the state of B_1 and write on B_0, the process 1 can read the state of B_0 and write on B_1, and both processes can read and write the variable B_2. All operations on the variables are atomic.

We model Peterson's algorithm as a network of MTSs (Figure 10). The process components do not share any actions (apart from the interrupt calls) and synchronize solely via the shared variables B_0, B_1, B_2 modeled as two-state MTSs (s_0 and s_1 correspond to *True* and *False*, respectively). To analyze more in-depth properties of the algorithm, each state s_j of each process is joined by the interrupt request *irq* (transitions with dashed arcs) with its static counterpart is_j that preserves the labeling; the returning transition is labeled with *irqret* and marked with a dotted arc. The *dm* (dummy) nodes are unreachable and used for the variable access consistency. The *monitor* is a component that activates with the *irq* request. After this, there exists a determined, unique three transitions long sequence that ends in an internal state marked with the current values of B_0, B_1, B_2. The wavy lines in Figure 10 marked with HI are used to cover parts of monitor omitted due to its size: a wavy line between awk and $di_0i_1i_2$ means that the latter is reached from the former via a sequence of actions $B_0hdnis_{i_0}, B_1hdnis_{i_1}, B_2hdnis_{i_2}$. After establishing the current state of variables, the monitor sets new values of B_0, B_1, B_2; this is done in a manner similar to the earlier state detection. A wavy line marked with HS and joining atk and $ei_0i_1i_2$ means that to reach the latter from the former, the sequence of actions $B_0hdnset_{i_0}, B_1hdnset_{i_1}, B_2hdnset_{i_2}$ should be fired. After this, the monitor terminates by firing the *irqret* transition. The labels on remaining unmarked arcs are not relevant to the example. Using the nonparametric component of the tool, we have verified that the model with the interrupts turned off satisfies the basic properties of mutual exclusion, the lack of deadlocks, liveness, nonblocking, and no strict sequencing [Baier and Katoen 2008]. The synthesis has taken only 0.07 seconds and 5.02MB of BDD memory (as reported by CUDD).

Now we move to the parameter synthesis for Peterson's algorithm. Denote $\mathcal{A}_{\text{hdnis}} = \{B_ihdnis_j \mid i, j \in \{0, 1\}\}$ and $\mathcal{A}_{\text{hdnset}} = \{B_ihdnset_j \mid i, j \in \{0, 1\}\}$. Let us also denote

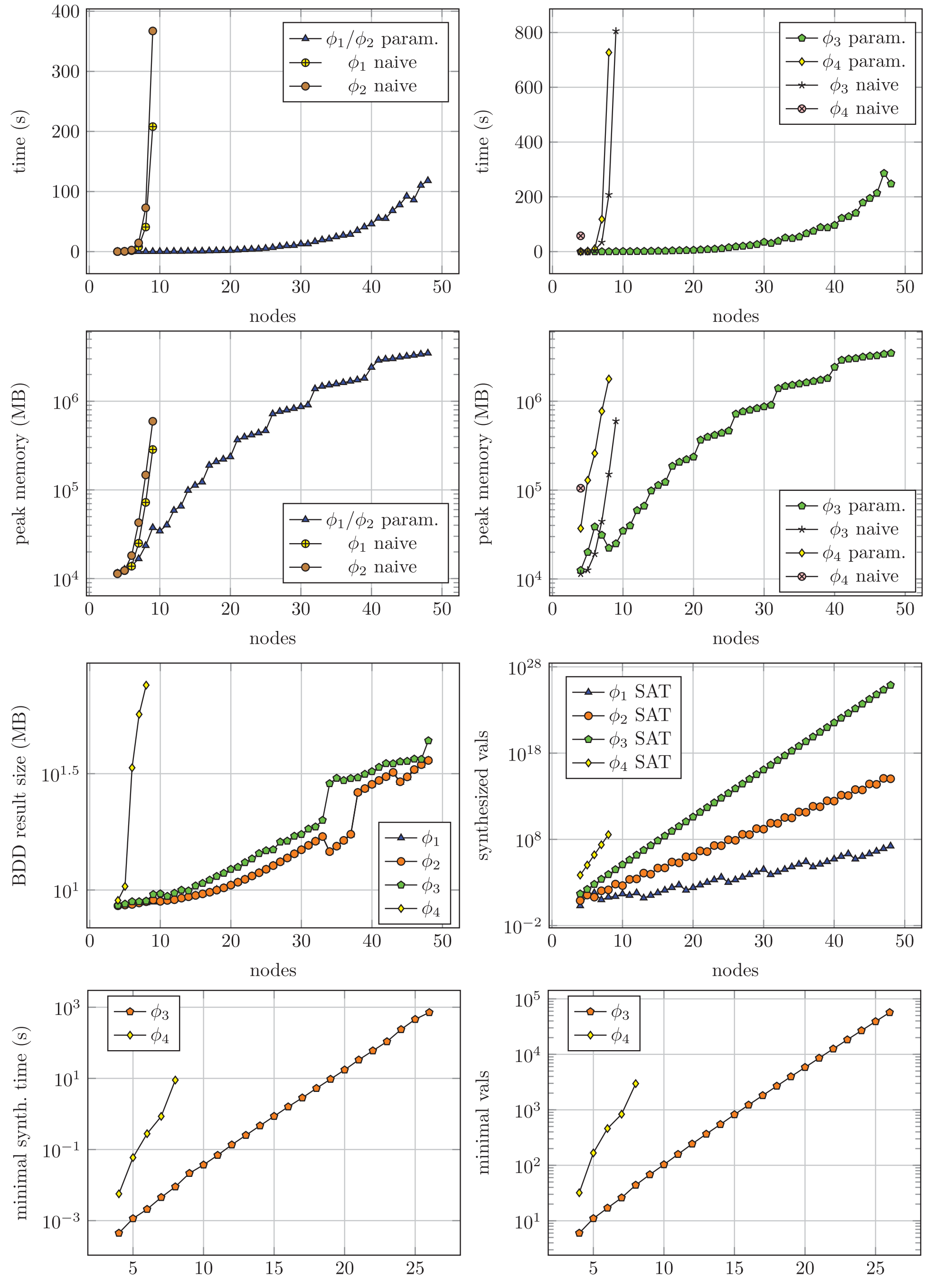

Fig. 8. Generic pipeline paradigm synthesis results.

Variable initialization

$B_0 := False; B_1 := False$

Process 0	Process 1

$B_0 := True$
$B_2 := True$
while $B_1 = True$ and $B_2 = True$ **do**
  $pass$ {busy wait}
end while
{critical section}
$B_0 := False$
	$B_1 := True$
	$B_2 := False$
	while $B_0 = True$ and $B_2 = False$ **do**
	  $pass$ {busy wait}
	end while
	{critical section}
	$B_1 := False$

Fig. 9. Peterson's algorithm.

$\mathcal{A}_{\text{norm}} = \bigcup_{i,j\in\{0,1\}}\{B_i set_j, B_i is_0\} \cup \{B_2 is_1\}$. We first analyze the property:

$$\phi_{dtct} = \boldsymbol{E}_{\mathcal{A}_{\text{norm}}}\boldsymbol{F}\boldsymbol{A}_Y\boldsymbol{G}(\text{dtct} \Rightarrow (\text{trying}_0 \vee \text{trying}_1 \vee \text{critical}_0 \vee \text{critical}_1)) \wedge \boldsymbol{E}_Y\boldsymbol{F}\text{dtct}$$

with the switchable actions set $\mathcal{A}_{\text{hdnis}}$. The meaning of ϕ_{dtct} is whether the monitor can infer only by looking at the values of B_0, B_1, B_2 if any of the two processes is attempting to enter or have already entered the critical section. The synthesis took 0.04 seconds and 5.40MB of BDD memory, and the naive approach took 1.06 seconds and 5.56MB of BDD memory. The resulting set is empty. In practice, this means that Peterson's protocol is not susceptible to eavesdropping—that is, a third party cannot tell just by looking at the values of the shared variables what the current state of the involved processes is.

In what follows, we assume the set $\mathcal{A}_{\text{hdnis}} \cup \mathcal{A}_{\text{hdnset}}$ of switchable actions. We move to the active monitor mode, where the monitor during the interrupt first detects the current state of the variables and then sets them to arbitrary values. The next property that we analyze is

$$\phi_{\text{nfrcAX}} = \boldsymbol{E}_{\mathcal{A}_{\text{norm}}}\boldsymbol{F}\boldsymbol{A}_Y\boldsymbol{G}(\text{nfrc} \Rightarrow \boldsymbol{A}_{\{irqret\}}\boldsymbol{X}\boldsymbol{A}_{\mathcal{A}_{\text{norm}}}\boldsymbol{X}$$
$$(\text{trying}_0 \vee \text{trying}_1 \vee \text{critical}_0 \vee \text{critical}_1)) \wedge \boldsymbol{E}_Y\boldsymbol{F}\text{done}.$$

In this way, we pose the question whether the monitor can test and set B_0, B_1, B_2 in such a way that after the return from the interrupt and a single step of the algorithm at least one of the processes attempts to enter or have already entered the critical section. The synthesis took 0.07 seconds and 5.56MB of BDD memory, and the naive approach took 87.41 seconds and 5.72MB of BDD memory. Again, the set of resulting valuations is empty. This means that despite the full control over the shared variables, a third party is not able to ensure in any circumstances that any of the processes is in a labeled location in an immediate successor to the current state of the system.

We alter the previous property by allowing an arbitrary number of steps after the return from the interrupt. After trying out all possible configurations of joins of propositions from $\bigcup_{i\in\{0,1\}}\{\text{trying}_i, \text{critical}_i\}$, we found a single one that yields a nonempty set of valuations:

$$\phi_{\text{nfrcAF}} = \boldsymbol{E}_{\mathcal{A}_{\text{norm}}}\boldsymbol{F}\boldsymbol{A}_Y\boldsymbol{G}(\text{nfrc} \Rightarrow \boldsymbol{A}_{\{irqret\}}\boldsymbol{X}\boldsymbol{A}_{\mathcal{A}_{\text{norm}}}\boldsymbol{F}(\text{trying}_0 \wedge \text{trying}_1)) \wedge \boldsymbol{E}_Y\boldsymbol{F}\text{done}.$$

The ϕ_{nfrcAF} poses a question whether the monitor can test and set the shared variables in such a way that, in the case of a positive test, after the return from the interrupt it will be unavoidable that both processes simultaneously attempt to enter the critical section. The synthesis took 0.08 seconds and 5.52MB of BDD memory, and the naive approach took 79.36 seconds and 6.04MB of BDD memory. There are 21 possible substitutions for Y (of $4,095$) under that ϕ_{nfrcAF} holds. Some of these substitutions are redundant from the practical point of view—for example, in the set of solutions, there is an action

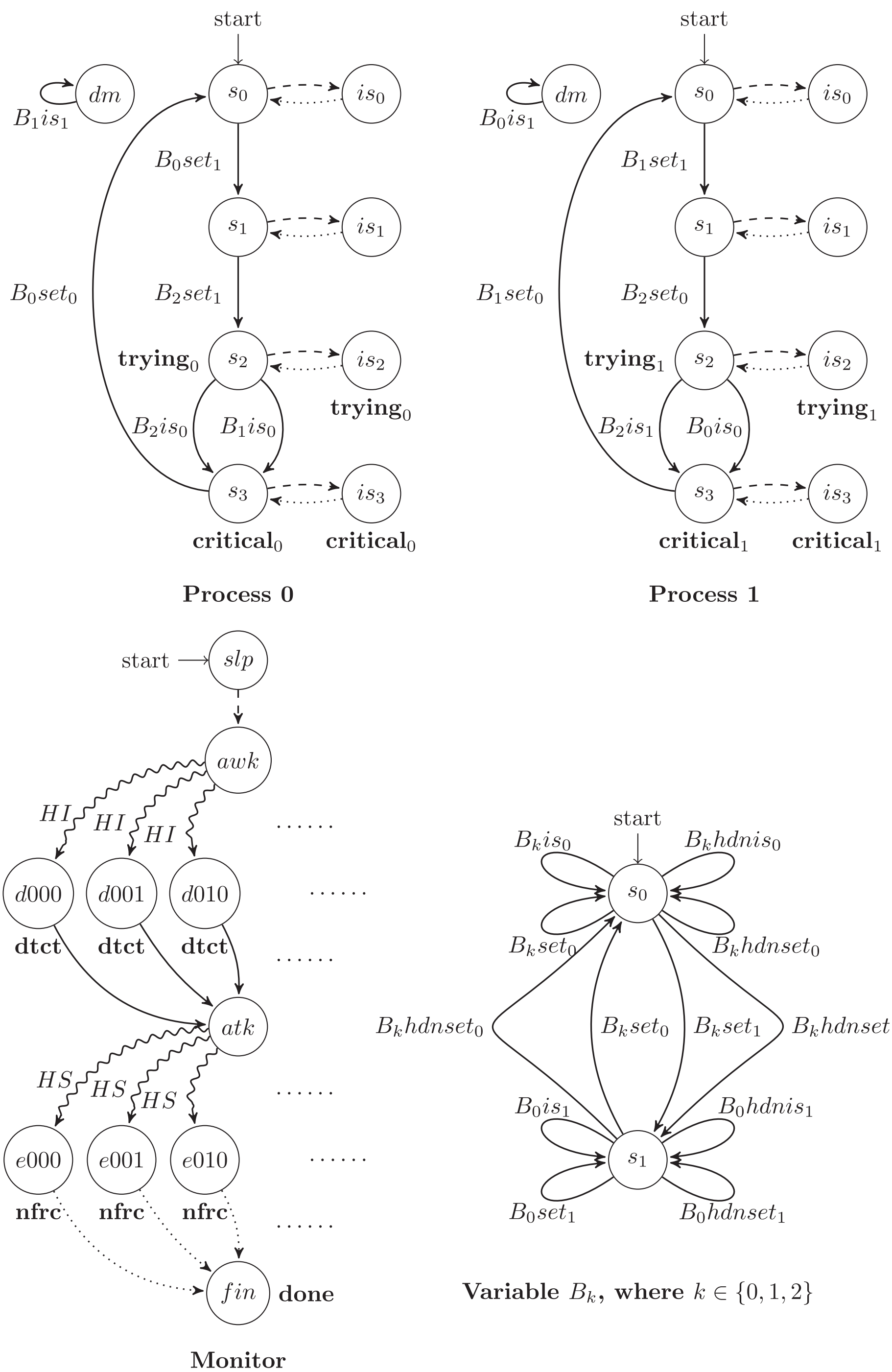

Fig. 10. Peterson's algorithm with monitor network.

if $(B_0, B_1, B_2) \in \{(0,0,0),(0,0,1),(0,1,1),(1,0,0)\}$ **then**
  set (B_0, B_1) to $(1,1)$
end if

Fig. 11. Malicious active monitor.

valuation υ that contains both $B_2hdnset_0$ and $B_2hdnset_1$ actions, and there are action valuations υ' and υ'' that differ from υ only in that they contain $B_2hdnset_i$ for a single $i \in \{0, 1\}$. The υ action is therefore not needed, as it expresses a nondeterministic choice where the deterministic one is possible. After the removal of unnecessary actions, we obtain 8 deterministic substitutions for Y, the analysis of which enables a concise recipe for malicious monitor behavior shown in Figure 11.

The preceding program guarantees that in 50% of the cases (i.e., possible configurations) after interrupt, the situation in which both the processes are simultaneously trying to enter the critical section is unavoidable. Note that among all possible internal states of Peterson's protocol, this one is arguably the most volatile and prone to attacks.

5. CONCLUSIONS

In this article, we proposed a new symbolic approach to the parameter synthesis for pmARCTL. The action valuations under which a given property holds are typically selected from a huge set, which makes the exhaustive enumeration intractable. We showed that despite this, the BDD-based implementation of fixed-point algorithms presented in this work can deal with small- to medium-sized models reasonably fast. This observation is in line with the results presented in Classen et al. [2011] in the context of model checking of software product lines. Our experimental results also demonstrate that our approach is promising for industrial system designers.

REFERENCES

R. Alur, K. Etessami, S. La Torre, and D. Peled. 2001. Parametric temporal logic for "model measuring." *ACM Transactions on Computational Logic* 2, 3, 388–407.

R. Alur, T. Henzinger, and M. Vardi. 1993. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*. ACM, New York, NY, 592–601.

É. André, L. Fribourg, U. Kühne, and R. Soulat. 2012. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM 2012: Formal Methods*. Lecture Notes in Computer Science, Vol. 7436. Springer, 33–36.

C. Baier and J.-P. Katoen. 2008. *Principles of Model Checking*. MIT Press, Cambridge, MA.

F. Belardinelli, A. V. Jones, and A. Lomuscio. 2011. Model checking temporal-epistemic logic using alternating tree automata. *Fundamenta Informaticae: Concurrency Specification and Programming* 112, 1, 19–37.

V. Bruyére, E. Dall'Olio, and J. Raskin. 2008. Durations and parametric model-checking in timed automata. *ACM Transactions on Computational Logic* 9, 2, 1–23.

E. Clarke and E. A. Emerson. 1981. Design and synthesis of synchronization skeletons for branching-time temporal logic. In *Logics of Programs*. Lecture Notes in Computer Science, Vol. 131. Springer, 52–71.

A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. 2011. Symbolic model checking of software product lines. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. ACM, New York, NY, 321–330.

O. Coudert, J. C. Madre, H. Fraisse, and H. Touati. 1993. Implicit prime cover computation: An overview. In *Proceedings of the Synthesis and Simulation Meeting and International Interchange (SASIMI'93)*.

C. Daskalakis, R. M. Karp, E. Mossel, S. Riesenfeld, and E. Verbin. 2011. Sorting and selection in posets. *SIAM Journal on Computing* 40, 3, 597–622.

B. Di Giampaolo, S. La Torre, and M. Napoli. 2010. Parametric metric interval temporal logic. In *Language and Automata Theory and Applications*. Lecture Notes in Computer Science, Vol. 6031. Springer, 249–260.

J. Goldsmith, M. Hagen, and M. Mundhenk. 2008. Complexity of DNF minimization and isomorphism testing for monotone formulas. *Information and Computation* 206, 6, 760–775.

T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. 2002. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming* 52–53, 183–220.

A. V. Jones, M. Knapik, A. Lomuscio, and W. Penczek. 2012. Group synthesis for parametric temporal-epistemic logic. In *Proceedings of the 11th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'12)*. 1107–1114.

G. Katz and D. Peled. 2010. Code mutation in verification and automatic code correction. In *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, Vol. 6015. Springer, 435–450.

M. Knapik. 2014. SPATULA, Simple Parametric Verification Tool. Retrieved August 10, 2015, from https://michalknapik.github.io/spatula.

M. Knapik, A. Męski, and W. Penczek. 2014. Action synthesis for branching time logic: Theory and applications. In *Proceedings of the 14th International Conference on Application of Concurrency to System Design (ACSD'14)*. IEEE, Los Alamitos, CA, 1–10.

C. Pecheur and F. Raimondi. 2006. Symbolic model checking of logics with actions. In *Model Checking and Artificial Intelligence*. Lecture Notes in Computer Science, Vol. 4428. Springer, 113–128.

D. Peled. 1993. All from one, one for all: On model checking using representatives. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 697. Springer, 409–423.

G. L. Peterson. 1981. Myths about the mutual exclusion problem. *Information Processing Letters* 12, 3, 115–116.

W. Quine. 1952. The problem of simplifying truth functions. *American Mathematical Monthly* 59, 8, 521–531.

P. Schnoebelen. 2002. The complexity of temporal logic model checking. In *Advances in Modal Logic*, Vol. 4. World Scientific Publishing, 393–436.

F. Somenzi. 2012. CUDD: CU Decision Diagram Package. Retrieved August 10, 2015, from http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html.

Received October 2014; revised March 2015; accepted March 2015