Action Synthesis for Branching Time Logic: Theory and Applications

Michał Knapik*, Artur Męsk† and Wojciech Penczek‡

* Institute of Computer Science, PAS, Poland: {knapik,meski,penczek}@ipipan.waw.pl
† University of Natural Sciences and Humanities, II, Poland
‡ University of Łódź, FMCS, Poland

Abstract—We introduce a parametric extension of Action-Restricted Computation Tree Logic. 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 correct parameter values. The time complexity of the problem and of the algorithm 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 in comparison with a naive solution to the problem.

I. INTRODUCTION

Parameter synthesis is a generalisation of the model checking problem, where a formula [1], [2], [3] and/or a model [4], [5] are augmented with parameters, and aims at computing the values of the parameters guaranteeing that the formula holds 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 formulae of a branching time temporal logic. We build upon Action Restricted Computation Tree Logic (ARCTL) [6], which we augment with parameters corresponding to the sets of actions, by defining the logic pmARCTL. To solve the synthesis problem for pmARCTL we propose a fixed-point based algorithm, inspired by [7], 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 paper. 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 the 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: faulty Train-Gate-Controller and a Generic Pipeline Processing network.

As far as the related work is concerned the problem of synthesis of the valuations under which a given modal property holds was first investigated in [1] in the context of a parametric version of LTL. In [2] and [3] the authors analyse parametric extensions of MITL and TCTL, respectively. In [7] the authors consider synthesis of agent groups for the CTLK properties in a multi-agent setting. In [8] 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 parameterised logics, their work shares the same difficulties as the problems we deal with in this paper: both the statespace and the set of solutions are susceptible to exponential blowup. The experimental results of [8] show that the symbolic verification of fCTL can be up to 766-times faster than the brute-force. We extend these results to pmARCTL parameter synthesis, where the relative speedup can exceed 35000. The work presented here is also related to parametric model checking with parameters in models [4], [9], [5], and model synthesis from a specification [10], [11].

The rest of the paper is organised as follows. In the next section we introduce the syntax and the semantics of pmARCTL. The algorithms for parameter synthesis are given in Section 3, and an experimental evaluation is provided in Section 4. The paper concludes with a brief summary and conclusions.

II. MIXED TRANSITION SYSTEMS AND PMARCTL

In this section we recall some basic definitions and present the syntax and semantics of the logic pmARCTL used in the paper. Mixed Transition Systems [6] are essentially Kripke structures with the transitions labelled with actions. The labels serve us to express branching-time properties with the selected set of actions allowed along a given run.
Definition 1 (MTS). Let \( PV \) be a set of propositional variables. A mixed transition system (MTS, for short) is a 5-tuple \( M = (S, s^0, A, T, V_s) \), where:

- \( S \) is a non-empty finite set of states,
- \( s^0 \in S \) is the initial state,
- \( A \) is a non-empty finite set of actions,
- \( T \subseteq S \times A \times S \) is a transition relation,
- \( \forall_s : S \to 2^{PV} \) is a (state) valuation function.

As usually, we write \( s \xrightarrow{a} s' \) if \((s, a, s') \in T \). Let \( \chi \subseteq A \) be a finite 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 \( \pi \) if \( \pi \) is finite, and \( \omega \) if \( \pi \) is infinite. A sequence \( \pi \) is a path over \( \chi \) if \( s_i \xrightarrow{a_i} s_{i+1} \) and \( a_i \in \chi \) for each \( i < |\pi| \) and if it is maximal with respect to this condition. Note that if a path \( \pi \) is finite, then its final state does not have a \( \chi \)-successor state in \( S \), i.e., \( \pi = (s_0, a_0, s_1, a_1, \ldots, s_m) \) and there is no \( s' \in S \) and \( a \in \chi \) s.t. \( s_m \xrightarrow{a} s' \).

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

Example 1. A simple mixed transition system with \( PV = \{ p, \text{safe} \} \), actions \( A = \{ \text{left}, \text{right}, \text{forward}, \text{back} \} \), and the initial state \( s_0 \) is presented in Fig. 1. The path \((s_0, \text{left}, s_1, \text{right}, s_4) \) belongs to \( \Pi(\{\text{left}, \text{right}\}) \), but it does not belong to \( \Pi(\{\text{left}, \text{right}, \text{back}\}) \). The reason is that while \((s_0, \text{left}, s_1, \text{right}, s_4) \) is a maximal path over \( \{\text{left}, \text{right}\} \) as it can be extended e.g. into an infinite path \((s_0, \text{left}, s_1, \text{right}, s_4, \text{back}, s_0, \ldots) \in \Pi(\{\text{left}, \text{right}, \text{back}\}) \).

The MTSs defined in this paper slightly differ from these introduced in [6], where the actions that label the transitions are treated as propositions. The difference is not essential however, as in [6] the propositional formulae over actions serve only to select sets of actions allowed along considered runs. We obtain the same result by the explicit description of allowed actions.

A. Parametric ARCTL

The logic presented here is a parametric extension of Action-Restricted Computation Tree Logic (ARCTL) [6]. The language of ARCTL consists of the CTL-like branching-time formulae. The main difference between ARCTL and CTL is that each path quantifier is superscripted with a set of actions. The superscripts are used in path selection, e.g., \( E_{\{\text{left}, \text{right}\}} G (E_{\text{forward}} F \text{safe}) \) 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”.

Parametric ARCTL (paARCTL) extends ARCTL by allowing free variables in place of sets of actions, e.g., \( E_{y}(G(E_{z} F \text{safe})) \).

Definition 2 (paARCTL syntax). Let \( A \) be a finite set of actions, \( \text{ActVars} \) be a finite set of variables, and \( PV \) be a set of propositional variables. The set of the formulae of Parametric Action-Restricted CTL is defined by the following grammar:

\[
\phi ::= p | \neg \phi | \phi \lor \psi | E_{\alpha} X \phi | E_{\alpha} G \phi | E_{\alpha} G^\omega \phi | E_{\alpha}(\phi U \psi),
\]

where \( p \in PV, \alpha \in \text{ActSets} \cup \text{ActVars} \), and \( Y \in \text{ActVars} \).

The \( E \) path quantifier is read as “there exists a path”. The superscript \( \alpha \) restricts the quantification to the path over \( \alpha \). The \( X \) modality stands for “in the next state”. The state modalities \( G \) and \( G^\omega \) are two versions of the “globally” modality, where \( G \) is applied to all paths while \( G^\omega \) pertains to infinite ones only. The modality \( U \) stands for “until”.

Since the considered formulae contain free variables, their validity needs to be defined with respect to provided valuations of \( \text{ActVars} \). Let \( \text{ActSets} = 2^A \setminus \{\emptyset\} \). A function \( v : \text{ActVars} \to \text{ActSets} \) is called an action valuation and the set of all action valuations is denoted by \( \text{ActVals} \). By \( M, s \models v \phi \), we denote that the formula \( \phi \) holds in the state \( s \) of the model \( M \) under the valuation \( v \), as formalised in Definition 3 (we omit the model symbol where it is clear from the context). In what follows, by \( \pi_i \) we denote the \( i \)-th state of \( \pi \).

In order to limit the indices range of \( \pi \), which can be either finite or infinite, we define the relation:

\[
\leq_k \defeq \begin{cases} 
\leq & \text{if } k < \omega \\
\leq & \text{if } k = \omega,
\end{cases}
\]

where \( k \in \mathbb{N} \cup \{\omega\} \). For conciseness, if \( v \) is an action valuation, then let:

\[
v(\alpha) \defeq \begin{cases} 
\chi & \text{if } \alpha = \chi \subseteq A, \\
v(Y) & \text{if } \alpha = Y \in \text{ActVars}.
\end{cases}
\]

Lastly, for \( O \in \{U, G, X, \Pi\} \) we assume that \( O^\omega = O \).

Definition 3 (paARCTL semantics). Fix an MTS \( M = (S, s^0, A, T, V_s) \) and let \( v \in \text{ActVals} \) be an action valuation. The relation \( \models_v \) is defined as follows:

- \( s \models_v p \text{ iff } p \in V_s(s) \),
- \( s \models_v \neg \phi \text{ iff } s \not\models_v \phi \),
- \( s \models_v \phi \lor \psi \text{ iff } s \models_v \phi \text{ or } s \models_v \psi \),
where the set of formulae model selection, a correct model synthesis from a general be explored with many goals in mind, including a minimal deadlock attribute. The space of synthesised valuations can state of $\text{s} \cup \text{A}$ and $\text{E}$. The $\text{E}_\text{r}$ stands for

\( \text{E}_\text{r} \), for some $\pi \in \Pi(\nu(\alpha), s)$, $\text{E}_\text{r} \), for all $\nu(\alpha), s$, $\text{E}_\text{r} \), for some $\pi \in \Pi(\nu(\alpha), s)$, $\text{E}_\text{r} \), for all $0 \leq j \leq i$, for some $\pi \in \Pi(\nu(\alpha), s)$, where $p \in \mathcal{PV}$, $\phi, \psi \in \text{pmARCTL}$, and $r \in \{\omega, \epsilon\}$, and $\alpha \in \text{ActSets} \cup \text{ActVars}$. Next, we define several derived modalities. Let $\phi, \psi \in \text{pmARCTL}$ and denote:

1. $\text{E}_\alpha X^{\omega} \phi \overset{\text{def}}{=} \text{E}_\alpha X(\phi \land \text{E}_\alpha G^{\omega} \text{true}),$
2. $\text{E}_\alpha (\text{U} \psi) \overset{\text{def}}{=} \text{E}_\alpha (\phi \land \text{E}_\alpha G^{\omega} \text{true}),$
3. $\text{E}_\alpha (\text{F} \phi) \overset{\text{def}}{=} \text{E}_\alpha (\phi \land \text{E}_\alpha F \phi),$
4. $\text{A}_\alpha X^{\omega} \phi \overset{\text{def}}{=} \text{E}_\alpha X^{\omega} \phi,$
5. $\text{A}_\alpha G^{\omega} \phi \overset{\text{def}}{=} \text{E}_\alpha G^{\omega} \phi,$
6. $\text{A}_\alpha (\text{U} \psi) \overset{\text{def}}{=} \text{E}_\alpha (\phi \land \text{E}_\alpha G^{\omega} \phi),$
7. $\text{A}_\alpha (\text{F} \phi) \overset{\text{def}}{=} \text{E}_\alpha (\phi \land \text{E}_\alpha F \phi),$ where $\alpha \in \text{ActSets} \cup \text{ActVars}$, and $r \in \{\omega, \epsilon\}$. The $U^{\omega}$ and $X^{\omega}$ modalities are counterparts of $U$ and $X$, defined on the infinite paths. $\text{F}^{(\omega)}$ stands for “in some future state (on an infinite path)”. $A$ stands for “for each path”. The semantics of the derived modalities is consistent with the intuition.

**Example 2.** Consider the MTS from Fig. 1 and the formulae $\phi_1 = AY Gp$ and $\phi_2 = AY G^p p$. It is easy to check that for $\nu(\alpha), s \in \text{Actvals}$ such that $\nu(\alpha), s \in S$, the set $\Pi(\nu(\alpha), v)$ consists of infinite paths only, and we have $s_0 \models \nu, s_1 \models \nu, s_2$ and $s_3$ for each state reachable from $s$. For $v(\alpha), s \in \text{Actvals}$ satisfying $v(\alpha), s \in S$, the set $\Pi(\nu(\alpha), s)$ contains finite paths along which $p$ does not hold globally (e.g., $v(\alpha), s_4, s_5$ and $s_6$) therefore $s_0 \models \nu, s_1 \models \nu$. $s_2$ while $s_0 \models \nu, s_2$. A. Algorithms for computing $f_{\phi}$

**Propositional variables and boolean operations.** Let $p \in \mathcal{PV}$ be a propositional variable and $s \in S$ be a state. It is easy to notice that the set $\nu(p, s)$ consists of either all the action valuations if $s$ is labelled with $p$, or is empty otherwise, thus $\nu(p, s) = \text{Actvals}$ if $p \in \nu(\alpha), s$, and $\nu(p, s) = \emptyset$ if $p \notin \nu(\alpha), s$. Now, let $p \in \text{pmARCTL}$ and $\nu$ be given. Then the set $\nu(p, s)$ consists of all the action valuations $v$ such that $s \models \nu, p$. From the inductive assumption this is equivalent to $v \not\models \nu(p, s)$, from which follows: $\nu(p, s) = \text{Actvals} \setminus \nu(p, s)$. To deal with the boolean connectives, assume that $\phi, \psi \in \text{pmARCTL}$ and $f_{\phi}$ and $f_{\psi}$ are given. Recall that from definition $s \models \nu, \phi \lor \psi$ if $s \models \nu, \phi$ or $s \models \nu, \psi$. By the inductive assumption, $s \models \nu, \phi$ or $s \models \nu, \psi$ is equivalent to $v \in f_{\phi}(s)$ or $v \in f_{\psi}(s)$, therefore: $f_{\phi \lor \psi}(s) = f_{\phi}(s) \cup f_{\psi}(s)$.

**Parametric preimage and neXt.** Let $f : S \to 2^\text{Actvals}$ be a function. The existential parametric preimage of $f$ with respect to $Y \in \text{Actvals}$ is defined as the function $\nu(p, s)$ such that:

\[ \text{parPre}_{\nu}^3(f)(s) = \left\{ v \mid \exists s' \in S \exists a \in \nu(\alpha, Y) \left( s \overset{a}{\rightarrow} s' \land v \in f(s') \right) \right\} \]

for each $s \in S$.

It follows immediately from the (s) condition that for each $\phi \in \text{pmARCTL}$ the set $\nu(p, s)$ consists of all such action valuations $v$ that some state $s'$ such that $s' \models \nu \phi$ can be reached by firing an action from $v(Y)$.

**Lemma 1.** For each $s \in S$, $\phi \in \text{pmARCTL}$, and $Y \in \text{Actvals}$, and $v \in \text{Actvals}$, the following condition holds:

\[ s \models \nu E_Y X \phi \iff v \in \text{parPre}_{\nu}^3(f_{\phi})(s) \]

**Proof:** From the definition $s \models \nu, E_Y X \phi$ iff there exists a path $\pi \in \Pi(\nu(\alpha), s)$ such that $|\pi| > 1$ and $\pi_1 \models \nu, \phi$, which is equivalent to $\exists s' \in S \exists a \models \nu(\alpha, Y) \left( s \overset{a}{\rightarrow} s' \land s' \models \nu, \phi \right)$ as $s, a, s'$ can be extended to a path. This, in turn, is equivalent to $\exists s' \in S \exists a \models \nu(\alpha, Y) \left( s \overset{a}{\rightarrow} s' \land v \in f(s') \right)$, i.e., $v \in \text{parPre}_{\nu}^3(f_{\phi})$.

The meaning of the above lemma can be expressed as: $f_{E_Y X \phi} = \text{parPre}_{\nu}^3(f_{\phi})$ for each $\phi \in \text{pmARCTL}$.

**Example 4.** Consider the MTS from Fig. 1. By case-by-case analysis one can see that $s_1 \models \nu, E_Y F \text{safe} \iff \text{forward} \in v(Y)$ and $s_2 \models \nu, E_Y F \text{safe} \iff \forall v \in \text{Actvals}$, thus $f_{E_Y F \text{safe}(s_1)} = \{ v \mid \text{forward} \in v(Y) \}$ and $f_{E_Y F \text{safe}(s_2)} = \text{Actvals}$. To compute $\text{parPre}_{\nu}^3(f_{E_Y X (E_Y F \text{safe})})(s_0)$ notice i.e., $f_{\phi}(s)$ returns all the valuations under which $\phi$ holds in $s$.

**Example 3.** Consider the model in Fig. 1 and the formula $\phi_1 = E_Y p U(p \land \text{safe})$. By hand calculations one can check that $s_0 \models \nu, \phi_1 \iff \{ \text{forward, left, right} \} \subseteq v(Y)$.

In what follows, given a model, by writing the function $f_{\phi}$ for $\phi$ of pmARCTL, we assume that $f_{\phi}$ satisfies the condition (s). Next, we show how to compute the function $f_{\phi}$ by means of the recursive compositions, preimage, and fixpoints. Throughout this section let $M$ be a fixed MTS and $Y \in \text{Actvals}$.
that in order to reach $s_1$ or $s_2$ from $s_0$ the actions forward or left should be fired, respectively. Therefore:

$$\begin{align*}
\text{parPre}_Y^3(f_{E^vX(E^vF_{safe}))})(s_0) &= \bigcup_{i \in \{1,2\}} \{v \mid \exists \pi \in \Pi(Y) \ s_0 \xrightarrow{\pi} s_i \} \\
\cap f_{E^vF_{safe}}(s_i) &= \{v \mid \text{forward} \in v(Y)\}.
\end{align*}$$

Two versions of the \textit{Globally} modality. We employ the equivalence $E^v G^2 \phi \equiv \phi \land E^v X E^v G^2 \phi$ to obtain Algorithm 1. Note the similarity to its non-parametric counterpart. The case of $E^v G$ is more interesting, as the non-totality of the transition relation needs to be covered. To this end in Algorithm 2 we consecutively keep adding action valuations under which the given states satisfy the considered formula but are \textit{deadlocked}, i.e., have no successors.

\textbf{Algorithm 1} \textit{Synth}_{E^vG^2}(f_\phi,Y)

Input: $f_\phi \in (2^{ActVals})^S$  
Output: $f_{E^vG^2\phi} \in (2^{ActVals})^S$

1: $f := f_\phi$; $h := \emptyset$  
2: while $f \neq h$ do  
3: : $h := f$  
4: : $f := f \cap \text{parPre}_Y^3(h)$  
5: end while  
6: return $f$

\textbf{Algorithm 2} \textit{Synth}_{E^vG}(f_\phi,Y)

Input: $f_\phi \in (2^{ActVals})^S$  
Output: $f_{E^vG\phi} \in (2^{ActVals})^S$

1: $f := f_\phi$; $h := \emptyset$  
2: $D := f_{\phi \land \sim E^v X \text{true}}$  
3: while $f \neq h$ do  
4: : $h := f$  
5: : $f := (f \cap \text{parPre}_Y^3(h)) \cup D$  
6: end while  
7: return $f$

\textbf{Lemma 2.} Let $\phi$ be a pmARCTL formula, $r \in \{\omega, \epsilon\}$, and $Y \in \text{ActVals}$. For all $s \in S$ and $v \in \text{ActVals}$ holds that: $s \models_v E^v G^r \phi$ iff $v \in \text{Synth}_{E^vG^r}(f_\phi,Y)(s)$

\textbf{Proof:} Let us first prove that $s \models_v E^v G^r \phi$ iff $v \in \text{Synth}_{E^vG^r}(f_\phi,Y)(s)$. For a while, replace the condition in Line 2 of Algorithm 1 with \textit{true}. In this way, the while loop 2–5 becomes infinite, and we can define $f_\phi$ for each $i \in N$ as the value of the $v$ variable after the $i$–th run and $f_0 = f_\phi$. First, prove that:

$$f_i(s) = \{v \mid \exists \pi \in \Pi(y,y),s) \mid \pi \geq i \land \forall 0 \leq j \leq i \pi_j \models v, \phi)\}$$

for each $s \in S$ and $i \in N$. The base case for $i = 0$ follows immediately from the definition. For the inductive step notice that $f_{i+1} = f_i \cap \text{parPre}_Y^3(f_i)$ and $\text{parPre}_Y^3(f_i)(s) = \{v \mid \exists \pi \in \Pi(y,y),s) \mid \pi \geq i + 1 \land \forall 0 \leq j \leq i \pi_j \models v, \phi)\}$, from which it follows that $f_i(s) \cap \text{parPre}_Y^3(f_i)(s) = \{v \mid \exists \pi \in \Pi(y,y),s) \mid \pi \geq i + 1 \land \forall 0 \leq j \leq i \pi_j \models v, \phi)\}$.

Observe that $s \models_v E^v G^2 \phi$ iff $v \in \bigcap_{i \in N} f_i(s)$. Notice that $f_{i+1}(s) \subseteq f_i(s)$ for all $i \in N$, $s \in S$, and $f_i$ is a fixedpoint 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, i.e., prove that $s \models_v E^v G \phi$ iff $v \in \text{Synth}_{E^vG}(f_\phi,Y)(s)$. Let $\phi \in \text{pmARCTL}$ and notice that $D = f_{\phi \land \sim E^v X \text{true}}$ is a constant, and $D(s) = \{v \mid (s \models \phi) \land \sim \exists \pi \leq s \Pi(y,y),s) \pi \models v, \phi)\}$ for each $s \in S$. The set $D(s)$ consists of all the action valuations under which $\phi$ holds in $s$ and $s$ has no successor. By $f_\phi$ we denote the value of the $v$ variable after the $i$–th run of the 3–6 loop of Algorithm 2. Also, let $f_0 = f_\phi$, as given in Line 1. We prove that $f_i(s) = A^\phi_F(s) \cup A^\omega_F(s)$ for each $i \in N$, $s \in S$, where:

$$A^\phi_F(s) = \{v \mid \exists \pi \in \Pi(y,y),s) \mid \pi \geq i \land \forall 0 \leq j \leq i \pi_j \models v, \phi)\}$$

$$A^\omega_F(s) = \{v \mid \exists \pi \in \Pi(y,y),s) \mid \pi > i \land \forall 0 \leq j \leq i \pi_j \models v, \phi)\}$$

i.e., $A^\phi_F(s)$ consists of action valuations under which there exists a finite path of length smaller than or equal to $i$ along which $\phi$ holds, whereas $A^\omega_F(s)$ contains all such valuations that along some path of length greater than $i$ the $\phi$ formula holds up to its $i$–th state. The base case of $f_0(s)$ follows immediately from the definition of $f_0$ (note that $D(s) \subseteq f_0(s)$ for all $s \in S$). For the inductive step, first notice that (Line 5) $f_{i+1} = (f_i \cap \text{parPre}_Y^3(f_i)) \cup D$, and that for each $s \in S$ we have $\text{parPre}_Y^3(f_i)(s) = \{v \mid \exists \pi \in \Pi(y,y),s) \mid \pi > i \land \forall 0 \leq j \leq i \pi_j \models v, \phi)\}$ and $\text{parPre}_Y^3(f_i)(s) = \{v \mid \exists \pi \in \Pi(y,y),s) \mid \pi > i \land \forall 0 \leq j \leq i \pi_j \models v, \phi)\}$. We can now easily derive that $(f_0(s) \cap \text{parPre}_Y^3(f_i)(s)) \cup D(s) = A^\omega_F(s) \cup A^\phi_F(s)$. The sequence $(A^\phi_F(s))_{i \in N}$ is increasing, therefore it eventually stabilizes at the fixpoint $A^\phi_F(s)$ consisting of all the action valuations under which $\phi$ holds along some finite path starting from $s$. The sequence $(A^\omega_F(s))_{i \in N}$ decreases until it reaches a fixpoint $A^\omega_F(s)$, consisting of all action valuations under which $\phi$ holds along an infinite path beginning at $s$. As $\text{Synth}_{E^vG}(f_\phi,Y)(s) = A^\phi_F(s) \cup A^\omega_F(s)$, this concludes the proof.

From Lemma 2: $f_{E^vG^2\phi} = \text{Synth}_{E^vG^2}(f_\phi,Y)$, $f_{E^vG\phi} = \text{Synth}_{E^vG}(f_\phi,Y)$.

\textbf{Untill} modality. Similarly as in the case of CTL, the equivalence $E^v (\phi \psi) \equiv \psi \phi \land E^v X E^v (\phi \psi)$ motivates the following fixpoint algorithm.

\textbf{Algorithm 3} \textit{Synth}_{E^vU}(f_\phi,f_\psi)

Input: $f_\phi,f_\psi \in (2^{ActVals})^S$
Output: $f_{E^v\phi \psi} \in (2^{ActVals})^S$

1: $f := f_\phi$; $h := \emptyset$  
2: while $f \neq h$ do  
3: : $h := f$  
4: : $f := f_\psi \cup (f_\phi \cap \text{parPre}_Y^3(h))$  
5: end while  
6: return $f$

\textbf{Lemma 3.} For each $s \in S$, $\phi,\psi \in \text{pmARCTL}$, $Y \in \text{ActVals}$, and $v \in \text{ActVals}$ the following condition holds:
Proof: For now, assume that the while loop 2–4 of Algorithm 3 is infinite (i.e., put true in place of the f ≠ 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_v (as given in Line 1). First, let us prove that:

\[ f_i(s) = \{ v \mid \exists x \in \Pi_v(y), s \exists j \leq (\pi_j \leq v \psi \land \forall 0 \leq k < j \pi_k \leq v \phi) \} \]

for each s ∈ S. In the case of f_0 the above equality follows immediately from the definition of f_v. For the inductive step, notice that due to the substitution in Line 3 we have that

\[ f_i+1 = f_v \cup (f_0 \cap \text{parPre}_i(f_1)) \]

Now, observe that:

\[ \text{parPre}_i(f_1)(s) = \{ v \mid \exists x \in S \exists x \in \Pi_v(y)(s \exists s' \land v \in f_i(s')) \} \]

\[ = \{ v \mid \exists x \in S \exists x \in \Pi_v(y)(s \exists s' \land \exists x \in \Pi_v(y), s' \exists j \leq (\pi_\psi \land \forall 0 \leq k < j \pi_k \leq v \phi)) \}
\]

The above formula can be simplified to:

\[ = \{ v \mid \exists \pi_v(y), s \exists \pi \leq i \leq 1 (\pi \leq v \psi \land \forall 0 \leq k < j \pi_k \leq v \phi) \} \]

and therefore \( f_i(s) \cap \text{parPre}_i(f_1)(s) = \{ v \mid \pi_v(y), s \exists \pi \leq i \leq 1 (\pi \leq v \psi \land \forall 0 \leq k < j \pi_k \leq v \phi) \} \).

From the above we finally have:

\[ f_i(s) \cup (f_0(s) \cap \text{parPre}_i(f_1)(s)) = \{ v \mid \exists x \in \Pi_v(y), s \exists \pi \leq i \leq 1 (\pi \leq v \psi \land \forall 0 \leq k < j \pi_k \leq v \phi) \} \]

Now observe that \( s \models E_Y \phi \psi \psi \iff v \in U_{\phi \psi \psi} f_i(s) \). As \( f_i(s) \leq f_{i+1}(s) \) 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 \( f_k(s) = U_{\phi \psi \psi} f_i(s) \). The fixpoint however is always reached and the algorithm stops, because there is only a finite number of functions in \( (\mathcal{X} \text{ActVals})^S \) and \( f_i(s) \in \mathbb{N} \) a monotonic sequence of sets.

Following the chosen convention we have:

\[ f_{Y \phi \psi \psi} = \text{Synth}_{EU}(f_v, f_v, Y, \psi) \]

and from the definition it also follows that:

\[ f_{Y \phi \psi \psi} = f_{Y \phi \psi \psi} \text{true}_U \text{true}_U \]

Overall algorithm. The last algorithm presented in this section provides the entry point for the computation of the \( f_\phi \) function, given \( \phi \in \text{pmARCTL} \).

Algorithm 4 Synth_{true}(\phi)

Input: \( \phi \in \text{pmARCTL} \)
Output: \( f_\phi \in (\mathcal{X} \text{ActVals})^S \)

1: if \( \phi = E_Y \psi \) then
2: \quad return \text{parPre}_1(Synth_{true}(\psi))
3: else if \( \phi = E_Y G' \psi \) where \( r \in \{w, \} \) then
4: \quad return Synth_{true}(Synth_{true}(\psi), Y)
5: else if \( \phi = E_Y [\psi \text{true}_U] \) then
6: \quad return Synth_{EU}(Synth_{true}(\phi), Synth_{true}(\psi), Y)
7: else \{ propositional and non-parametric modalities omitted for simplicity \}
8: \quad return \{ \}
9: end if

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

Theorem 1. For each model \( \mathcal{M} \), formula \( \phi \in \text{pmARCTL} \), state \( s \in S \) and action valuation \( v \in \text{ActVals} \) holds that:

\[ \mathcal{M}, s \models \phi \iff v \in \text{Synth}_{true}(\phi)(s) \]

Proof: Follows immediately from Lemmas 1–3.

B. 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 \( v \) such that \( \mathcal{M}, s^0 \models \phi \). It is a well-defined decision problem, called the emptiness problem for pmARCTL.

Theorem 2. The emptiness problem for pmARCTL is NP-complete.

Proof: The proof follows via reduction from 3SAT. Let \( \mathcal{P} \) be a set of propositional variables, and let \( \mathcal{P} = \mathcal{P} \cup \{ p \mid p \in \mathcal{P} \} \) be the set of literals. Let \( n \in \mathbb{N} \), and let \( \mu = (a_1 \lor a_2 \lor a_3 \lor \ldots \lor (a_n \lor a_2 \lor a_3) \) be a propositional formula in 3CNF, where \( a_i \in \mathcal{P} \) for all \( 1 \leq i \leq n, 1 \leq j \leq 3 \).

![Figure 2: A model for 3SAT formula \( \mu = (a \lor \neg b \lor \neg c) \land (\neg a \lor d \lor \neg e) \) that is used in the proof. The dashed arcs are labelled with jmp.]
a^j such that (st_{i+1}, a^j, mst_j^i) \in \mathcal{T} for some 1 \leq j \leq 3, and (3) the set v(Y) does not contain a transition labelled with literal and transition labelled with its negation (this would create a path leading to sink_i, for some 1 \leq i \leq n, which is not labelled by p). For an action valuation v satisfying conditions 1-3 let \omega_v be a valuation (of propositional) s.t. \omega_v(a) is true iff a \in v(Y) for each a \in \mathcal{A}, and notice that \omega_v = \mu. Conversely, let \omega be s.t. \omega = \mu, define the action valuation \omega_v s.t. jmp \in v_v(Y) and a \in v_v(Y) iff \omega_v(a) is true for each a \in \mathcal{A}, and observe that v_v = \phi_\mu. Therefore \omega_v = \mu iff \mathcal{M} = \mathcal{M}_v, \phi_\omega.

Note that the presented reduction is polynomial. On the other hand, the ARCTL verification can be attained in polynomial time by a nondeterministic Turing machine. Therefore, let \phi \in \text{pmARCTL} contain k free variables. In order to estimate the time complexity of parPre^3(\phi) let us fix s, s' \in \mathcal{S} and let \phi(s') = \{v_1, \ldots, v_k\}. Let (s, s, s') \in \mathcal{T} and notice that parPre^3(\phi)(s) gathers such valuations from \phi(s') that a \in v_i(Y). As |\phi(s')| can be at most of \text{poly}(k), the worst case complexity of parPre^3(\phi) is in O(|\mathcal{S}| + |\mathcal{T}| \cdot \text{poly}(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 \text{poly}(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 for pmARCTL is in O(|\mathcal{S}|^2 \cdot \text{poly}(k) + |\mathcal{S}| |\mathcal{T}| \cdot \text{poly}(2k)).

The complexity of nonparametric ARCTL model checking is equal to that of CTL verification, therefore the complexity of the naive approach, based on enumerative checking of all the possible action valuations is in O(|\mathcal{S}| + |\mathcal{T}|^k \cdot \text{poly}(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 [12], 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 formulae in the naive approach, still \text{poly}(k) cases need to be separately analysed. As we show in the next section, our symbolic algorithm for pmARCTL substantially outperforms the naive approach.

IV. IMPLEMENTATION AND EVALUATION

In this section we present an evaluation of our implementation of the theory presented in this paper. We use parallel compositions of MTSs as models, with disjunctive location labelling, i.e., a given vector of locations is labelled with a proposition p if any of its components is labelled with p.

Definition 4. Let \{1, \ldots, k\} be a finite set of indices, and for each \i in \{1, \ldots, k\} let \mathcal{M}_i = (\mathcal{S}_i, s^0_i, \mathcal{A}, T, \mathcal{V}_i) be an MTS. We define the product with the disjunctive location labelling of a network \{\mathcal{M}_i\}_{i \in \{1, \ldots, k\}} as an MTS \mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, T, \mathcal{V}) such that: \mathcal{S} = \bigcup_{i \in \{1, \ldots, k\}} \mathcal{S}_i, and s^0 = (s^0_1, \ldots, s^0_k), and \mathcal{A} = \bigcup_{i \in \{1, \ldots, k\}} \mathcal{A}_i, and the transition \mathcal{T} satisfies:

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

and the labelling \mathcal{V} such that for each proposition p \in \mathcal{P}: p \in \mathcal{V}_\mathcal{S}(l_1, \ldots, l_k) iff p \in \mathcal{V}_\mathcal{S}(l_i) for some \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 [13], which implements the parameter synthesis methods. The tool uses CUDD [14] package providing operations on Reduced Ordered Binary Decision Diagrams (ROBDD) to represent the statespace and action valuations. SPATULA allows for modelling 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, therefore, for the sake of comparison, we implemented a naive engine, which enumerates over all the 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 [8].

The memory usage results for the naive cases are omitted from the figures, as they overlap with the parametric ones until the timeout, which was set at 15 mins. The experiments have been performed on an Intel P6200 dual core 2.13 GHz machine, running Linux operating system.

A. Scalability on faulty Train-Gate-Controller

The system presented in Fig. 3 is a version of the classical model from [4] with the modifications inspired by [15]. It consists of \(k\) trains and the controller monitoring the access to the tunnel.

![Figure 3: Faulty Train-Gate-Controller.](image-url)
tunnel (\textit{out}), approaching the tunnel (\textit{appr}), or inside the tunnel (\textit{in}). If the controller is in the \textit{red} state, then no train is allowed in the tunnel, otherwise if the controller is in the \textit{green} state, then the trains are allowed in the tunnel. The \textit{j}-th train is assumed to be faulty and its communication with the controller is malfunctioning, i.e., it can perform a faulty action which does not change the controller state when entering the tunnel (\textit{in}_j^F) or leaving the tunnel (\textit{out}_j^F).

The network is described using the SPATULA’s simple modelling language as shown below. The language is essentially a graph network description language with a set of convenient, C-inspired flow control constructs.

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

    /* correct behaviour */
    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("in", "out", inlabel);
        ctr = ctr + 1;
    }

    /* faulty behaviour */
    labelF = "Train" + trainNo + "F";
    outlabelF = "outF" + faultyTrainNo;
    inlabelF = "inF" + faultyTrainNo;
    apprlabel = "appr" + trainNo;
    inlabel = "in" + trainNo;
    outlabel = "out" + trainNo;
    join_with("s1", "s0", outlabel);
    join_with("s0", "s1", inlabel);
    /* correct behaviour */
    faultyTrainNo = j;
    trainsNo = k;
    bloom("s1");
    bloom("s0");
    mark_with("s1", "red");
    mark_with("s0", "green");
    mark_with("s0", "initial");
    bloom("s0");
    /* control nodes */
    outlabel = "out" + trainNo;
    inlabel = "in" + trainNo;
    apprlabel = "appr" + trainNo;
    join_with("in", "out", outlabel);
    join_with("out", "in", inlabel);
    /* faulty behaviour */
    if(trainNo == faultyTrainNo) {
        outlabel = "out" + trainNo;
        inlabel = "in" + trainNo;
        apprlabel = "appr" + trainNo;
        join_with("s1", "s0", outlabel);
        join_with("s0", "s1", inlabel);
        /* faulty behaviour */
        faultyTrainNo = j;
        trainsNo = k;
        bloom("s1");
        bloom("s0");
        mark_with("s1", "red");
        mark_with("s0", "green");
        mark_with("s0", "initial");
        bloom("s0");
    }
```

Figure 4: SPATULA template for \textit{Controller} and \textit{Train}_i.

We have tested the following two properties: (1) \( \psi_1 = A_Y G(\neg \bigwedge_{1 \leq i \leq k} (\text{in}_i \land \text{in}_i)) \land A_Y F \text{in}_i \), 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) \( \psi_2 = E_Y F A_Y G(\bigwedge_{1 \leq i \leq j} \neg \text{in}_i) \land \text{green} \), expressing that it is possible for the system to execute in such a way that at some state, all the possible executions of the system, all the trains remain outside the tunnel while the controller remains in the \textit{green} state.

The parametric approach clearly outperforms the naïve one (see Fig. 5); 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, 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 one second, and within the specified time bound it managed to obtain the results for the systems with up to 13 trains, with the state-space of size \( 3 \cdot 10^6 \) and the space of possible solutions of size \( 2^{10} - 1 \).

![Figure 5](image.png)

**Figure 5:** Faulty Train-Gate-Controller results.

<table>
<thead>
<tr>
<th>Property</th>
<th>Speedup ( naïve/parametric time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \psi_1 )</td>
<td>2 trains</td>
</tr>
<tr>
<td>( \psi_2 )</td>
<td>48.96</td>
</tr>
</tbody>
</table>

Table I: Speedup for Faulty Train-Gate-Controller.
Table II: Speedup for Generic Pipeline Paradigm.

<table>
<thead>
<tr>
<th>Property</th>
<th>$\phi_1$</th>
<th>$\phi_2$</th>
<th>$\phi_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 processes</td>
<td>1402.60</td>
<td>3265.79</td>
<td>9171.02</td>
</tr>
<tr>
<td>9 processes</td>
<td>1202.53</td>
<td>3265.79</td>
<td>8723.19</td>
</tr>
<tr>
<td>10 processes</td>
<td>2985.93</td>
<td>7979.09</td>
<td>18633.09</td>
</tr>
</tbody>
</table>

(1 - the naïve approach exceeded set timeout of 10 minutes)

In the next benchmark we analyse three different cases of the ratio of the space growth of the synthesised solutions to the growth of the model.

B. Scalability on Generic Pipeline Paradigm

The network in Fig. 6, inspired by the Generic Pipeline Paradigm [16], consists of $k > 3$ processing nodes. A node can synchronise 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 the actions from the set $\{ret_i, act_i, act_{min(i+1,k)}, act_{min(i+2,k)}\}$).

![Generic Pipeline Paradigm Network](image)

Figure 6: Generic Pipeline Paradigm Network.

We have tested the following three properties: (1) $\phi_1 = A_Y F(\bigwedge_{1 \leq i \leq \lfloor \frac{k}{2} \rfloor} out_i \wedge \bigwedge_{\lfloor \frac{k}{2} \rfloor < j \leq k} in_j)$ that describes 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) $\phi_2 = A_Y GA_Y F(\bigwedge_{1 \leq i \leq k} in_i)$, expressing that the configuration with all the nodes simultaneously in their $in$ states appears infinitely often or ends a path; (3) $\phi_3 = E_Y F A_Y G(\bigwedge_{1 \leq i \leq \lfloor \frac{k}{2} \rfloor} in_{2i-1} \wedge \bigwedge_{1 \leq i \leq \lfloor \frac{k}{2} \rfloor} out_{2i})$, describing that the configuration with the first half of the nodes 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.

The naïve approach becomes infeasible for more than 9 nodes, whereas the parametric approach managed to compute the results for $\psi_1$, $\psi_2$ up to 20 nodes, and for $\psi_3$ up to 21 nodes. Moreover, using the parametric approach, the results up to 16 nodes are computed in under one second. The model with $k$ nodes consists of $2^k$ states and there are $2k$ separate actions, which gives $2^{2^k} - 1$ possible action valuations. 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 well in all the cases: for a fixed number of correct (SAT) action valuations ($\phi_1$), for a slowly growing number of SAT valuations ($\phi_2$), and for an exponentially growing number of SAT valuations ($\phi_3$).

C. Peterson’s algorithm analysis

In this section we analyse a solution to the mutual exclusion problem for two processes, proposed in [17]. For reference, we include (Fig. 8) a pseudocode for Peterson’s solution to the mutual exclusion problem for two processes.

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 $i$-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 the operations on the variables are atomic.
Variable initialisation

\[ B_0 := \text{False}; B_1 := \text{False} \]

Process 0

\[ B_0 := \text{True} \]
\[ B_2 := \text{True} \]
\[ \text{while } B_1 = \text{True} \text{ and } B_2 = \text{False} \text{ do} \]
\[ \text{pass } \{ \text{busy wait} \} \]
\[ \text{end while} \]
\[ \{ \text{critical section} \} \]
\[ B_0 := \text{False} \]

Process 1

\[ B_1 := \text{True} \]
\[ B_2 := \text{False} \]
\[ \text{while } B_0 = \text{True} \text{ and } B_2 = \text{False} \text{ do} \]
\[ \text{pass } \{ \text{busy wait} \} \]
\[ \text{end while} \]
\[ \{ \text{critical section} \} \]
\[ B_1 := \text{False} \]

![Figure 8: Peterson’s algorithm.](image1)

We model Peterson’s algorithm as a network of MTSs (see Figure 9). The process components do not share any actions (apart from the interrupt calls) and synchronise solely via the shared variables \( B_0, B_1, B_2 \) modelled as two-state MTSs. In order to analyse more in-depth properties of the algorithm, each state \( s_j \) of each process is joined by the interrupt request \( \text{irq} \) (transitions with dashed arcs) with its static counterpart \( s_j \) that preserves the labelling: the returning transition is labelled with \( \text{irqret} \) and marked with dotted arc. The \( dm \) (dummy) nodes are unreachable and used for the variable access consistency. The monitor is a component that activates with the \( \text{irq} \) request. After this, there exists a determined, unique three transitions long sequence that ends in an internal state. In order to model the current state of the variables, and then sets them to the current state of variables the monitor sets new actions. Denote \( \mathcal{A}_{\text{hdnis}} = \{ B_i \text{hdnis}_{ij} \mid i, j \in \{0,1\} \} \) and \( \mathcal{A}_{\text{hdnset}} = \{ B_i \text{hdnset}_{ij} \mid i, j \in \{0,1\} \} \). We first analyse the property:

\[ \phi_{\text{dtct}} = E_{A_{\text{norm}}} FA_Y G(\text{dtct} \implies (\text{trying}_0 \lor \text{trying}_1 \lor \text{critical}_0 \lor \text{critical}_1)) \land E_Y F\text{dtct} \]

with the switchable actions set \( \mathcal{A}_{\text{hdnis}} \). The meaning of \( \phi_{\text{dtct}} \) 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 sec. and 5.40 MB of BDD memory, and the naive approach took 1.06 sec. and 5.56 MB of BDD memory. The resulting set is empty. In practice, this means that Peterson’s protocol is not susceptible to eavesdropping, i.e., 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 we analyse is:

\[ \phi_{\text{nfrcAX}} = E_{A_{\text{norm}}} FA_Y G(\text{nfrc} \implies A(\text{irqret})X A_{\text{norm}}X (\text{trying}_0 \lor \text{trying}_1 \lor \text{critical}_0 \lor \text{critical}_1)) \land E_Y F\text{done}. \]

![Figure 9: Peterson’s algorithm with monitor network.](image2)
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 sec. and 5.56 MB of BDD memory, and the naïve approach took 87.41 sec. and 5.72 MB 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 labelled 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 the possible configurations of joins of propositions from $\bigcup_{i \in \{0,1\}} \{ \text{trying}_{i}, \text{critical} \}$ we found a single one that yields a nonempty set of valuations:

$$\phi_{nfrcAF} = E_{A_{norm}} FA G (nfrc \implies A_{irqret} X A_{norm} F (\text{trying}_0 \land \text{trying}_1)) \land EY F done.$$  

The $\phi_{nfrcAF}$ pose 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 sec. and 5.52 MB of BDD memory, and the naïve approach took 79.36 sec. and 6.04 MB of BDD memory. There are 21 possible substitutions for $Y$ (of 4095) under which $\phi_{nfrcAF}$ holds. Some of these substitutions are redundant from the practical point of view, e.g., in the set of solutions there is an action valuation $\nu$ that contains both $B_2hdnset_0$ and $B_2hdnset_1$ actions, and there are action valuations $\nu'$ and $\nu''$ that differ from $\nu$ only in that they contain $B_2hdnset_i$ for a single $i \in \{0, 1\}$. The $\nu$ 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 $Y$, analysis of which enables a concise recipe for malicious monitor behavior shown in Fig. 10.

\begin{verbatim}
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
\end{verbatim}

Figure 10: Malicious active monitor.

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

V. CONCLUSIONS

In this paper 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 have shown that despite this, the ROBDD-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 [8] in the context of model checking of software product lines. We argue that our approach is worth recommending to both the researchers and the industrial system designers.

Acknowledgements: Michal Knapił is supported by the Foundation for Polish Science under Int. PhD Projects in Intelligent Computing. Project financed from the EU within the Innovative Economy OP 2007-2013 and ERDF.

REFERENCES