Fixed-point Methods in Parametric Model Checking*

Michał Knapik[1] and Wojciech Penczek[1,3]

[1] Institute of Computer Science, PAS, Warsaw, Poland
{mknapik, penczek}@ipipan.waw.pl
[2] University of Natural Sciences and Humanities, II, Siedlce, Poland

Abstract We present a general framework for the synthesis of the constraints under which the selected properties hold in a class of models with discrete transitions, together with Boolean encoding - based method of implementing the theory. We introduce notions of parametric image and preimage, and show how to use them to build fixed-point algorithms for parametric model checking of reachability and deadlock freedom. An outline of how the ideas shown in this paper were specialized for an extension of Computation Tree Logic is given together with some experimental results.

1 Introduction

The analysis of the existing software- and hardware systems is already a daunting task, and due to their increasing proliferation in everyday life will only become more difficult, and important. The systems employed in the critical areas such as avionics, security, and medical applications need to be thoroughly tested and verified, with the testing and verification present from the possibly earliest stages of design. While a battery of tests is often able to reveal some errors present in a design, the formal verification aims to mathematically *guarantee* that the abstraction of the system is compliant with its specification.

The classical model checking [7] is a simple, yet powerful methodology of systems analysis. In this approach, the behaviour of the system is presented as a mathematical model $\mathcal{M}$ with the intended level of granularity; a property to be verified (e.g., the lack of deadlocks) is then expressed as a formula ϕ of selected modal logic. The compliance of the model with the property is denoted as $\mathcal{M} \models \phi$. Symbolic model checking applies various methods for an efficient representation of the state-space, allowing to verify large systems with more than 10^{20} states [6]. Model checking needs a fairly complete description of the model and the property to be verified, which limits its applicability when some of the details are still unknown. Parametric model checking (also called parameter synthesis) is an extension of model checking that allows for the presence of free variables in the model and/or the formula; the goal is to describe all the valuations of the free variables under which ϕ holds in $\mathcal{M}$. In this way the parametric model

* Work partially funded by $DEC - 2012/07/N/ST6/03426$ NCN Preludium grant

checking tool can provide pointers to system designer or analyst, e.g., on how to instantiate the unknown variables.

In this work we do not focus on any logic from the menagerie of known modal logics. Instead, we propose a general framework that encompasses a range of state-based models with discrete transition guards and logics with model-checking procedures based on the computation of the (pre)image with respect to the transition relation.

1.1 Related Work

The problem of the synthesis of the set of assignments under which a given formula becomes true in a selected model was introduced in [1] in the context of a parametric extension of Linear Temporal Logic (LTL), called Parametric Temporal Logic (PLTL). This was later explored for other parametric extensions of LTL, and Computation Tree Logic (CTL) in [9,11]. The [9] paper seems to be especially important, as it shows how to extend standard fixed-point - based algorithms for model checking to encompass quantified parameters in Parametric Real Time CTL (PRTCTL). In the analysis of real-time systems, parametric versions of Timed CTL (PTCTL) interpreted over parametric extensions of timed automata appear in [4,5] and [24,25]. The first group of papers presents the techniques based on a translation of durations of runs of a timed automaton to a formula of Presburger arithmetic, the second group extends explicit-state methods to a parametric version of region graph. In [16,18,22] SAT-based Bounded Model Checking methods have been adapted to the task of RTCTL and PRTCTL verification and to a limited parameter synthesis for parametric reachability in a selected class of Petri Nets. In [8] the authors introduce algorithms for verification of specifications for software product lines expressed in Feature CTL. This work is closest to what we present in this paper, as it employs fixed-point algorithms implemented using Binary Decision Diagrams.

1.2 Paper Outline

In Section 2 we present the framework for parameter synthesis together with all the relevant notions. Section 3 contains a brief description of how the algorithms can be implemented using Boolean encodings. In Section 4 we sketch how the introduced theory can be specialized on an example of a selected modal logic PARCTL; we also cite some earlier experimental results that show the efficiency of our approach. The paper ends with Conclusions.

2 Framework for Parameter Synthesis

In this work we do not focus on any logic from the menagerie of known modal logics. Instead, we propose a general framework that encompasses a range of state-based models with discrete transition guards and logics with model-checking procedures based on the computation of the (pre)image with respect to the transition relation.

2.1 Parametric Image and Preimage

Let us start with establishing the computation model. Definition 1 allows to interpret many of the state-based models encountered in the theory and practice of model checking.

Definition 1. *Let X be a finite set of propositional variables (called* parameters*) and let $\mathcal{L}_X$ denote the set of propositional formulae over X. A $\mathcal{L}_X$-*labeled transition system *(also called* a model*) is a tuple $\mathcal{M} = (S, s^0, \rightarrow)$, where S is a finite set of states, $s^0 \in S$ is the* initial state*, and $\rightarrow \in S \times \mathcal{L}_X \times S$ is a* transition relation*.*

In what follows we fix a set of parameters X and $\mathcal{L}_X$-labeled transition system $\mathcal{M} = (S, s^0, \rightarrow)$. The set of all valuations of X is denoted by Ω. If $t = (s, g, s') \in \rightarrow$, then let $src(t) = s$, $trgt(t) = s'$ and $guard(t) = g$. We abbreviate $(s, g, s') \in \rightarrow$ as $s \xrightarrow{g} s'$. Consider a sequence of interleaved states and transitions $\pi = (s_0, t_0, s_1, t_1, \ldots)$. If the sequence is finite, then we assume that it ends with a state. By $|\pi|$ we denote the floor of the length of the sequence divided by two; the i–th state and transition of π are denoted by $\pi_i^S = s_i$ and $\pi_i^T = t_i$, respectively. Let $\upsilon \in \Omega$, then π is a υ-path if $s_i \xrightarrow{t_i} s_{i+1}$ and $\upsilon \models guard(t_i)$ for all $i < |\pi|$. The set of all υ-paths in $\mathcal{M}$ is denoted by $\Pi(\mathcal{M}, \upsilon)$ and the set of all υ-paths starting from $s \in S$ is denoted by $\Pi(s, \mathcal{M}, \upsilon)$; we omit the model symbol when it is evident from the context. One way of thinking about υ-paths is that a given valuation υ enables all and only those transitions whose guards evaluate to *true*.

Example 1. Consider the model in Fig. 1 with the set of parameters $X = \{x_1, x_2, x_3\}$. The loop $\pi = (s_0, x_1 \wedge \neg x_3, s_1, x_1 \vee x_2, s_2, x_1 \wedge \neg x_2, s_0, \ldots)$ is a υ-path iff $\upsilon(x_1) = true$ and $\upsilon(x_2) = \upsilon(x_3) = false$. In fact, this is the only maximal υ-path for this valuation, and this is the only valuation under which $\Pi(s_0, \upsilon)$ contains an infinite path.

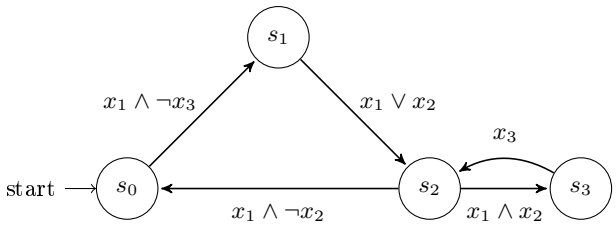

Figure 1: A simple model.

A function $\xi : S \rightarrow 2^\Omega$ is called a *characterization*. The set of all characterizations is denoted by *Chars*. A characterization ξ assigns to an each state a

set of valuations; intuitively, these are the constraints under which some given property holds. In what follows, for each $g \in \mathcal{L}_X$ by $[g]$ we denote the set of all valuations satisfying g.

Definition 2. *Let X be a set of parameters, and let $\mathcal{M} = (S, s_0, \rightarrow)$ be a $\mathcal{L}_X$-labeled transition system. We define the operators of* parametric image *and the* parametric preimage $Post, Pre \in Chars^{Chars}$ *with respect to $\rightarrow$ as follows. Let $\xi \in Chars$, then:*

$$Post(\xi)(s) = \bigcup_{t \in \rightarrow} \{[guard(t)] \cap \xi(src(t)) \mid trgt(t) = s\},$$
$$Pre(\xi)(s) = \bigcup_{t \in \rightarrow} \{[guard(t)] \cap \xi(trgt(t)) \mid src(t) = s\},$$

for all $s \in S$.

Let us assume that the characterization ξ defines the constraints under which some property in question holds. The set $Post(\xi)(s)$ consists of all the valuations that simultaneously enable the transition to s from some of its predecessors s' and ensure that the property ξ holds in s'. The set $Pre(\xi)(s)$ gathers all the valuations which simultaneously enable a transition to some successor s' of s and ensure that the property ξ holds in s'.

2.2 Reachability and Deadlock Freedom

The reachability analysis is arguably the most utilized formal methods-based task both in the research and the industry, employed in error detection [26], protocol analysis [12,19], planning [10], and many other areas. Algorithm 1 is a template that allows, depending on the passed operator *Oper* and initial property ξ, for the synthesis of constraints for reachability from the initial state (Lemma 1), reachability of a state satisfying a given property f (Lemma 2), and freedom from deadlocks in the model (Lemma 3).

Algorithm 1 $constrVals\,(\mathcal{M}, Oper, \xi)$

Input: $\mathcal{M} = (S, s^0, \rightarrow)$, $Oper \in \{Post, Pre\}$, $\xi \in Chars$

1: $f := \xi$
2: $h := \emptyset$
3: **while** $f \neq h$ **do**
4: $\quad h := f$
5: $\quad$ **for each** $s \in S$ **let** $f(s) := f(s) \cup Oper(f)(s)$
6: **end while**
7: **return** f

It is usually more feasible to use the parametric image (i.e., to test *forward reachability*) rather than the preimage operator (i.e., to test *backward reachability*) when synthesizing the constraints for the reachability from the initial state.

This is due to the possible chance of pruning the state-space by removing the trajectories that start from the states unreachable from the initial one. The state $s \in S$ is *reachable* from the initial state s^0 under valuation $\upsilon \in \Omega$ iff there exists a υ-path $\pi \in \Pi(s^0, \upsilon)$ such that $\pi_i^S = s$ for some $i \in \mathbb{N}$.

Lemma 1. *Let $\mathcal{M} = (S, s^0, \to)$ be a model. Denote $\xi_I := \{(s^0, \Omega)\} \cup \{(s, \emptyset) \mid s \in S \wedge s \neq s^0\}$, then a state $s \in S$ is reachable from s^0 under $\upsilon \in \Omega$ iff $\upsilon \in constrVals\,(\mathcal{M}, Post, \xi_I)\,(s)$.*

Proof. For each $i \in \mathbb{N}$ by f^i denote the value of the f variable before entering the 3–6 loop. For a while replace the stop condition test of the loop with *true*. We prove that for each $i \in \mathbb{N}$ the set $f^i(s)$ consists of all the valuations υ under which the state s can be reached in i or less steps from the initial state s^0, i.e., there exists $\pi \in \Pi(s^0, \upsilon)$ such that $\pi_j^S = s$ for some $j \leq i$. The proof follows by the induction with the trivial base case of f^0, as defined in Line 1. For the inductive step, notice that by the inductive assumption the state s is reachable under υ in $i+1$ or less steps iff $\upsilon \in f^i(s)$ or if for some $\pi \in \Pi(s^0, \upsilon)$ we have $\pi_{j+1}^S = s$ and $\upsilon \in f^i(\pi_j^S)$ and $\upsilon \models \pi_j^T$ for some $j \leq i$. The latter is in turn equivalent to $\upsilon \in f^i(s) \cup Post(f^i)(s)$, i.e., $\upsilon \in f^{i+1}(s)$ (Line 5). Now it suffices to notice that for each of a finite number of $s \in S$ the sequence $(f^i(s))_{i \in \mathbb{N}}$ is monotonically increasing and consists of subsets of a finite set Ω, thus for some $k \in \mathbb{N}$ we have $f^{j+k} = f^k$ for all $j \in \mathbb{N}$. Therefore the fixed-point is reached, the loop stops, and $f = f^k$ is the correct characterization of the reachability from s^0 for $\mathcal{M}$. □

Let $\xi \in Chars$ be a characterization, $\upsilon \in \Omega$, and $s \in S$. We say that ξ is reachable under υ from the state $s \in S$ iff there exists $\pi \in \Pi(s, \upsilon)$ such that $\upsilon \in \xi(\pi_i)$ for some $i < |\pi|$. To obtain the constraints for this type of reachability we employ the operator of the parametric preimage in Algorithm 1.

Lemma 2. *Let $\mathcal{M}$ be a model. A characterization $\xi \in Chars$ is reachable under $\upsilon \in \Omega$ from the state $s \in S$ iff $\upsilon \in constrVals\,(\mathcal{M}, Pre, \xi)\,(s)$.*

Proof. Again, for each $i \in \mathbb{N}$ by f^i we denote the value of the f variable before entering the loop 3–6. By the induction on $i \in \mathbb{N}$ we prove that for each $s \in S$ the set $f^i(s)$ consists of all the valuations under which for some $\pi \in \Pi(s, \upsilon)$ there exists $0 \leq j \leq i$ such that $\upsilon \in \xi(\pi_j)$. We omit the remaining details of the proof, as it follows very similarly to the proof of Lemma 1. □

One of the typical assumptions in the formal methods approach to systems analysis is that the model in question is free of deadlocks, i.e., every state reachable from the initial one has an enabled outgoing transition. A state $s \in S$ is in a *deadlock* under valuation $\upsilon \in \Omega$ iff there is no $s' \in S$, such that $s \xrightarrow{g} s'$ and $\upsilon \models g$, i.e., s has no successor under υ. A state is *deadlock-free* under $\upsilon \in \Omega$ iff it is not in a deadlock. In what follows, let $\xi^\Omega \in Chars$ be such a characterization that $\xi^\Omega(s) = \Omega$ for all $s \in S$. For each $s \in S$ the set $Pre(\xi^\Omega)(s)$ consists of all those valuations under which there is an outgoing transition from s. This means

that $Pre(\xi^{\Omega}) \in Chars$ is a characterization of *no-local-deadlock* property, what we want to obtain however, is the set of valuations under which no state in deadlock is reachable from a given state. Let $\xi \in Chars$ be a characterization, then we define its *complement* $\overline{\xi} \in Chars$ by $\overline{\xi}(s) := \Omega \setminus \xi(s)$ for all $s \in S$. Intuitively, $\overline{\xi}(s)$ gathers all those valuations under which the property ξ does not hold in s.

Lemma 3. *Let $\mathcal{M}$ be a model and $s \in S$. All states reachable from s under v are deadlock-free iff $v \in constrVals(\mathcal{M}, Pre, \overline{Pre(\xi^{\Omega})})(s)$.*

Proof. Let $s \in S$, then the set $\overline{Pre(\xi^{\Omega})}(s)$ consists of the valuations under which s is in a deadlock, i.e., $\overline{Pre(\xi^{\Omega})} \in Chars$ is a characterization of a deadlock. By Lemma 2, the set $constrVals(\mathcal{M}, Pre, \overline{Pre(\xi^{\Omega})})(s)$ gathers all those valuations under which a state in a deadlock is reachable from s, thus the complement of the set consists of valuations under which all reachable states are deadlock-free. □

3 Boolean Encodings

The notions introduced in this paper are geared towards symbolic parameter synthesis. In contrast to the explicit approach, in symbolic verification and synthesis we do not deal with single states and transitions; instead - we manipulate sets of states, functions and relations [6]. In this section we show how to express the problem of parameter synthesis using operations on propositional formulae.

3.1 Boolean Encodings and Operations

Let $\mathcal{M} = (S, s^0, \rightarrow)$ be a $\mathcal{L}_X$-labelled transition system, and let V be a set of propositional variables such that V is disjoint with X and $|V| = \lceil log(|S|) \rceil$. Recall that $\mathcal{L}_V$ denotes the set of propositional formulae over V. It is easy to see that we can encode the set of states S using the variables from V, i.e., for each $s \in S$ we define $enc(s) \in \mathcal{L}_V$ such that $enc(s)$ is satisfiable and $\not\models enc(s) \wedge enc(s')$ for all $s, s' \in S, s \neq s'$. We also use an auxiliary set of propositional variables V', disjoint with V and X and such that $|V'| = |V|$. Assume that $V = \{v_1, \ldots, v_k\}$ and $V' = \{v'_1, \ldots, v'_k\}$; if $g \in \mathcal{L}_V$ then $g' \in \mathcal{L}_{V'}$ denotes the formula obtained by replacing in g each v_i with its primed counterpart. The transition relation of $\mathcal{M}$ is then encoded as: $enc(\rightarrow) = \bigvee_{t \in \rightarrow} enc(src(t)) \wedge guard(t) \wedge enc'(trgt(tr))$.

Example 2. Let us encode the transition relation of the model in Fig. 1. Denote $V = \{v_1, v_2\}$ and $X = \{x_1, x_2, x_3\}$. Let us put $enc(s_0) = v_1 \wedge v_2$, $enc(s_1) = v_1 \wedge \neg v_2$, $enc(s_2) = \neg v_1 \wedge v_2$, $enc(s_3) = \neg v_1 \wedge \neg v_2$, then: $enc(\rightarrow) = \big((v_1 \wedge v_2) \wedge (x_1 \wedge \neg x_3) \wedge (v'_1 \wedge \neg v'_2)\big) \vee \big((v_1 \wedge \neg v_2) \wedge (x_1 \vee x_2) \wedge (\neg v'_1 \wedge v'_2)\big) \vee \big((\neg v_1 \wedge v_2) \wedge (x_1 \wedge \neg x_2) \wedge (v'_1 \wedge v'_2)\big) \vee \big((\neg v_1 \wedge v_2) \wedge (x_1 \wedge x_2) \wedge (\neg v'_1 \wedge \neg v'_2)\big) \vee \big((\neg v_1 \wedge \neg v_2) \wedge (x_3) \wedge (\neg v'_1 \wedge v'_2)\big)$.

If $A \subseteq \Omega$ then let $enc(A) \in \mathcal{L}_X$ be such that $[enc(A)] = A$. Let $f \in Chars$ be a characterization, then we define its encoding as: $enc(f) = \bigvee_{s \in S} enc(s) \wedge$

$enc(f(s))$. Let $\xi \in Chars$, then the operations of parametric image and preimage of ξ are performed symbolically as follows:

$$enc'(Post(\xi)) = \bigvee_{bf \in 2^V} \big(enc(\rightarrow) \wedge enc(\xi)\big)[v_1 \leftarrow bf(v_1), \ldots, v_k \leftarrow bf(v_k)],$$
$$enc(Pre(\xi)) = \bigvee_{bf \in 2^{V'}} \big(enc(\rightarrow) \wedge enc'(\xi)\big)[v'_1 \leftarrow bf(v'_1), \ldots, v'_k \leftarrow bf(v'_k)].$$

It is quite easy to see that the encoding of a join of two relations or characterizations is encoded as a disjunction of their encodings; the same follows for the meet, encoded as a conjunction. In order to implement the operation of complement, we make use of the following procedure.

Algorithm 2 *Complement*(f)

Input: $f \in Chars$; **Output:** $enc(\overline{f}) \in Chars$

1: **return** $\neg enc(f) \wedge \bigvee_{bf \in 2^{V'}} enc(f)[v'_1 \leftarrow bf(v'_1), \ldots, v'_k \leftarrow bf(v'_k)]$

The following lemma (first presented in [14]), applied to $F = enc(f)$ with f_i substituted with state encodings and g_i substituted with associated guards, proves that $enc(\overline{f}) = Complement(f)$.

Lemma 4. *Let $F = \bigvee_{i \in A} f_i \wedge g_i$, where A is a finite set of indices, and f_i, g_i are propositional formulae such that:*

- *sets of propositional variables present in f_i, g_i are disjoint,*
- *for all $i, j \in A$ such that $i \neq j$ we have that $\not\models f_i \wedge f_j$.*

Let $F' = \bigvee_{i \in A} f_i$, then $\neg F \wedge F' = \bigvee_{i \in A} f_i \wedge \neg g_i$.

Proof. Let $\upsilon \models \neg F \wedge F'$, then there exists exactly one $i \in A$ such that $\upsilon \models f_i$. On the other hand $\upsilon \not\models f_i \wedge g_i$, thus $\upsilon \models f_i \wedge \neg(f_i \wedge g_i)$. Now it suffices to notice that $f_i \wedge \neg(f_i \wedge g_i) \equiv f_i \wedge \neg g_i$. Now let $\upsilon \models \bigvee_{i \in A} f_i \wedge \neg g_i$. As previously, there exists exactly one $i \in A$ such that $\upsilon \models f_i \wedge \neg g_i$, and obviously $\upsilon \models F'$. It suffices to notice that if we had $\upsilon \models F$ then $\upsilon \models f_i \wedge g_i$ would hold. This is not possible, therefore $\upsilon \models \neg F$, and $\upsilon \models \neg F \wedge F'$. □

4 Application and Evaluation

In this section we present an outline of how the general ideas and models introduced so far have been specialized for a selected logic, providing parametric frameworks for action synthesis [17]. We have implemented the theory in stand-alone tool SPATULA [15,17]. The tool employs Reduced Ordered Binary Decision Diagrams (BDDs) CUDD package [23] for an efficient representation of

transition relation and operations on Boolean encodings. To provide the means for the comparison with the non-parametric approach we also implemented the *naïve*, enumerative parameter synthesis. In the naïve approach, a tool simply iterates through all the possible substitutions of parameters and records all those for which the result yields true.

4.1 Preliminaries

In what follows, we introduce a parameterized logic that allows for the presence of the parameters (i.e., free variables) in the formulae. In order to deal with the formulae with multiple parameters, we need to extend the notion of a characterization. Let ϕ be a formula with parameters $Pars$ (i.e., a modal logic formula, interpreted in a $\mathcal{L}_X$-labeled transition system $\mathcal{M}$). Let $\upsilon : Pars \to 2^X$ be a valuation of $Pars$ (we use the symbol Ω_{Pars} to denote the set of all such valuations). In this work, we assign to $Pars$ the subsets of the set of allowed actions (subsection 4.2). By $\phi[\upsilon]$ we denote the result of substitution of the variables in ϕ in accordance to υ, and by $\mathcal{M}, s \models_\upsilon \phi$ we denote that the ground formula $\phi[\upsilon]$ holds in the state s of $\mathcal{M}$ (we usually omit the model symbol). The methods presented in the previous section can be extended to allow for an efficient construction, for each formula ϕ of a chosen logic, of the function $f_\phi : S \to 2^{\Omega_{Pars}}$ such that: $\upsilon \in f_\phi(s)$ iff $s \models_\upsilon \phi$, for all $s \in S$. We only present a brief explanation of the relevant extensions, referring to the earlier work for a detailed description.

Let $\mathcal{M}$ be a model with the set of states S, and let $\mathcal{PV}$ be a finite set of fresh (i.e., not appearing in the model) propositions. A function $\mathcal{V} : S \to 2^{\mathcal{PV}}$ is called a *labeling*. Intuitively, $\mathcal{V}(s)$ denotes the set of all the propositions from $\mathcal{PV}$ that are true in the state $s \in S$.

4.2 Parametric Action-restricted CTL

Action-restricted CTL [20] (ARCTL) is a simple branching time logic with actions. The formulae of ARCTL limit the set of actions allowed along a given run. In [17] we have introduced a parametric extension of the logic, called PARCTL that allows for the presence of the variables in place of concrete sets of actions.

Definition 3 (PARCTL syntax). *Let $Acts$ be a finite set of* actions*, and $Pars$ be a finite set of* action variables*. The set of the formulae of Parametric Action-Restricted CTL is defined by the following grammar:*

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

where $p \in \mathcal{PV}$, $\alpha \in (2^{Acts} \cup Pars)$, and $Y \in Pars$.

The E path selector is read as *"there exists a path"*, and the superscript $_\alpha$ selects the actions allowed along a given run. The X modality stands for *"in a next state"*, the modalities G, G^ω stand for *"globally"* with the second one pertaining to the infinite paths only, and the U modality is read as *"until"*. In what follows, we

also use the universal path selector A (read as *"for all paths"*) and the temporal modality F (standing for *"in future"*); both of these can be derived from the already introduced notions. As to give an example, $E_{\{\text{left, right}\}}G(E_{\{\text{forward}\}}F \text{ safe})$ is a formula without parameters that 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*"*. A formula $E_YG(E_ZF \text{ safe})$ contains two parameters we seek to valuate, i.e., for a given state $s \in S$ we wish to obtain all valuations υ such that $E_{\upsilon(Y)}G(E_{\upsilon(Z)}F \text{ safe})$ holds in s.

We refer to [17] for the full description of the semantics of the logic and the construction of the function f_ϕ for each $\phi \in$ PARCTL. In general, the implementation of the Boolean operations is rather straightforward, and the following equalities hold: $E_YG^\omega\phi \equiv \phi \wedge E_YXE_YG^\omega\phi$, and $E_YG\phi \equiv \phi \wedge (\neg E_YXtrue \vee E_YXE_YG^\omega\phi)$, and $E_Y(\phi U\psi) \equiv \psi \vee (\phi \wedge E_YXE_Y(\phi U\psi))$, where $\phi, \psi \in$ PARCTL, and $Y \in Pars$. These equivalences can be converted into fixed-point algorithms [17] based on the consecutive computation of $f_{E_YX(\cdot)}$, which in turn can be obtained by a single application of parametric preimage operation with some additional variable relabeling.

4.3 Experimental Results

We have performed a batch of tests to establish the efficiency of our approach as compared to the naïve, enumerative one, as to our best knowledge there is no tool comparable to ours. In all the experiments, the timeout was set at 15 minutes.

In the first test, we used a version of Train-Gate-Controller, a classical benchmark [2] with the injected faulty behaviour, as inspired by [3]. The system consists of k trains and the controller that monitors the access to the tunnel. The safety requires that at most one train at a time enters the tunnel. In a non-faulty version of the model this is ensured by a simple protocol where a train gains an access if it reaches an agreement with the controller. In our version of the system, the (*red/green* light - based) communication between the selected faulty train and a controller can malfunction. We have tested the properties:

- $\psi_1 = A_YG(\neg \bigvee_{1 \le i < l \le k}(in_i \wedge in_l)) \wedge \bigwedge_{1 \le i \le k} E_YFin_i$, with the meaning 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"*;
- $\psi_2 = E_YFA_YG\big((\bigwedge_i^k \neg in_i) \wedge green\big)$, with the meaning that: *"it is possible for the system to execute in such a way that at some state, in all the possible executions of the system, all the trains remain outside the tunnel while the controller remains in the* green *state"*.

As we aim to synthesize actions that are used for communication (via synchronized actions) between the participating entities, the results of the synthesis can be interpreted as finding the sets of messages that have to be turned off in order to provide the compliance with a specification.

In Table 1 we provide the results on the relative time speedup of the parametric approach versus the naïve one. As it can be clearly seen, the benefits of moving

from an exponential number of simple tests operating on sets (enumerative synthesis) to a small number of complex tests operating on characteristic functions are very substantial. This is in line with the results reported in [8], where the relative speedup exceeded 750 for the emptiness and universality tests.

Property	**Speedup** (naïve/parametric time)			
	2 trains	3 trains	4 trains	5 trains
ψ_1	76.0	463.59	4021.68	17378.02
ψ_2	48.96	276.01	703.97	1553.73

Table 1: Speedup for Faulty Train-Gate-Controller.

In the second test, we analyze a pipeline network inspired by [21]. The model in question consists of the chain of k nodes, each of which has two states: *in* and *out*. A node can synchronize via shared action with up to four other surrounding ones, depending on its position in the pipeline. The first node can be perceived as a Producer, and the last one as a Consumer, and each of the intermediate tokens can obtain an information token from one of its predecessors by firing a shared action and moving to the *in* state. In this way the token can be moved from the Producer to the Consumer through a series of intermediate nodes. We have tested the properties:

- $\phi_1 = A_Y F(\bigwedge_{1 \leq i \leq \lfloor \frac{k}{2} \rfloor} out_i \wedge \bigwedge_{\lceil \frac{k}{2} \rceil \leq j \leq k} in_j)$, describing unavoidability of a configuration in which the first half of the nodes is in *out*- and the other half is in *in* states;
- $\phi_2 = A_Y G A_Y F(\bigwedge_{1 \leq i \leq k} in_i)$, expressing that the configuration with all the nodes simultaneously in their *in* states appears infinitely often or ends a path;
- $\phi_3 = E_Y F A_Y G(\bigwedge_{1 \leq i \leq \lceil \frac{k}{2} \rceil} in_{2i-1} \wedge \bigwedge_{1 \leq i \leq \lfloor \frac{k}{2} \rfloor} out_{2i})$, 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 point in the future.

As previously, we present the relative speedup of our approach in Table 2. The relative efficiency of our approach is even more evident in this case, as we were able to verify formulae for which the naïve approach timed out within the set time limits.

Property	**Speedup** (naïve/parametric time)			
	7 processes	8 processes	9 processes	10 processes
ϕ_1	1402.60	4115.96	9171.02	22669.83
ϕ_2	1202.53	3265.79	8723.40	$> 12344.49^{\dagger}$
ϕ_3	2985.93	7979.04	18633.09	$> 34531.71^{\dagger}$

($\dagger$ - the naïve approach exceeded set timeout of 15 minutes)

Table 2: Speedup for Generic Pipeline Paradigm.

We refer to [17] for the detailed presentation and analysis of the experimental results, as well as for some applications of our tool to concurrent systems security.

5 Conclusions

In this work we presented a framework for parametric model checking for models with discrete transitions. We have shown how to synthesize constraints for reachability by means of computing the fixpoint of consecutively applied sequence of parametric image operations, and how to synthesize constraints for deadlock-freedom by means of computing the fixpoint of consecutively applied sequence of operations of parametric preimage. We have also outlined how to implement the presented theory using Boolean encodings (typically, BDDs), and how to extend it to properties expressed in selected modal logics on an example of PARCTL. The benefits of the approach are illustrated on two scalable benchmarks.

Acknowledgements: Michał Knapik 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

1. Alur, R., Etessami, K., Torre, S.L., Peled, D.: Parametric Temporal Logic for "Model Measuring". ACM Trans. Comput. Log. 2(3), 388–407 (2001)
2. Alur, R., Henzinger, T., Vardi, M.: Parametric real-time reasoning. In: Proc. of the 25th Ann. Symp. on Theory of Computing (STOC'93). pp. 592–601. ACM (1993)
3. Belardinelli, F., Jones, A.V., Lomuscio, A.: Model checking temporal-epistemic logic using alternating tree automata. Fundam. Inform. 112(1), 19–37 (2011)
4. Bozzelli, L., La Torre, S.: Decision problems for lower/upper bound parametric timed automata. Form. Methods Syst. Des. 35(2), 121–151 (Oct 2009), `http://dx.doi.org/10.1007/s10703-009-0074-0`
5. Bruyère, V., Dall'Olio, E., Raskin, J.F.: Durations, parametric model-checking in timed automata with presburger arithmetic. In: Alt, H., Habib, M. (eds.) STACS. Lecture Notes in Computer Science, vol. 2607, pp. 687–698. Springer (2003)
6. Burch, J.R., Clarke, E., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98(2), 142–170 (1990)
7. Clarke, E.M.: The birth of model checking. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. Lecture Notes in Computer Science, vol. 5000, pp. 1–26. Springer (2008)
8. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A.: Symbolic model checking of software product lines. In: Proc. of the 33rd Int. Conf. on Software Engineering. pp. 321–330. ICSE '11, ACM, New York, NY, USA (2011)
9. Emerson, E.A., Trefler, R.: Parametric quantitative temporal reasoning. In: Proc. of the 14th Symp. on Logic in Computer Science (LICS'99). pp. 336–343. IEEE Computer Society (July 1999)

10. Ghallab, M., Nau, D.S., Traverso, P.: Automated planning - theory and practice. Elsevier (2004)
11. Giampaolo, B.D., La Torre, S., Napoli, M.: Parametric metric interval temporal logic. In: Dediu, A.H., Fernau, H., Martín-Vide, C. (eds.) LATA. Lecture Notes in Computer Science, vol. 6031, pp. 249–260. Springer (2010)
12. Holzmann, G.J.: Protocol design: Redefining the state of the art. IEEE Software 9(1), 17–22 (1992)
13. Jensen, K., Donatelli, S., Koutny, M. (eds.): Transactions on Petri Nets and Other Models of Concurrency IV, Lecture Notes in Computer Science, vol. 6550. Springer (2010)
14. Jones, A.V., Knapik, M., Penczek, W., Lomuscio, A.: Parametric computation tree logic with knowledge. In: Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'11). pp. 286–300. Białystok University of Technology (2011)
15. Knapik, M.: https://michalknapik.github.io/spatula
16. Knapik, M., Penczek, W., Szreter, M., Pólrola, A.: Bounded parametric verification for distributed time Petri nets with discrete-time semantics. Fundam. Inform. 101(1-2), 9–27 (2010)
17. Knapik, M., Męski, A., Penczek, W.: Action synthesis for branching time logic: Theory and applications. In: Proc. of the 14th Int. Conf. on Application of Concurrency to System Design (to appear). IEEE Computer Society (2014)
18. Knapik, M., Szreter, M., Penczek, W.: Bounded parametric model checking for elementary net systems. In: T. Petri Nets and Other Models of Concurrency [13], pp. 42–71
19. Lin, F.J., Chu, P.M., Liu, M.T.: Protocol verification using reachability analysis: The state space explosion problem and relief strategies. SIGCOMM Comput. Commun. Rev. 17(5), 126–135 (Aug 1987)
20. Pecheur, C., Raimondi, F.: Symbolic model checking of logics with actions. In: Proc. of MoChArt 2006. Springer Verlag (2006)
21. Peled, D.: All From One, One For All: On Model Checking Using Representatives. In: Proc. of CAV'93. pp. 409–423 (1993)
22. Penczek, W., Pólrola, A., Zbrzezny, A.: Sat-based (parametric) reachability for a class of distributed time petri nets. In: T. Petri Nets and Other Models of Concurrency [13], pp. 72–97
23. Somenzi, F.: CUDD: CU decision diagram package - release 2.3.1. http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html
24. Wang, F.: Parametric timing analysis for real-time systems. Inf. Comput. 130(2), 131–150 (1996)
25. Wang, F.: Parametric analysis of computer systems. Formal Methods in System Design 17(1), 39–60 (2000)
26. Xie, Y., Aiken, A.: Scalable error detection using boolean satisfiability. SIGPLAN Not. 40(1), 351–363 (Jan 2005)