Combining Explicit and Symbolic LTL Model Checking Using Generalized Testing Automata

Ala Eddine Ben Salem
Laboratoire de Recherche et
Développement de l'EPITA (LRDE)
Le Kremlin-Bicêtre, France
Email: ala@lrde.epita.fr

Mohamed Graiet
Institut Supérieur d'Informatique
et de Mathématiques de Monastir
Monastir, Tunisie
Email: Mohamed.Graiet@imag.fr

***Abstract*—In automata-theoretic model checking, there are mainly two approaches: *explicit* and *symbolic*. In the explicit approach [1], the state-space is constructed explicitly and lazily during exploration (i.e., on-the-fly). The symbolic approach [2] tries to overcome the state-space explosion obstacle by symbolically encoding the state-space in a concise way using decision diagrams. However, this symbolic construction is not performed on-the-fly as in the explicit approach. In order to take advantage of the best of both worlds, *hybrid approaches* [3, 4, 5] are proposed as combinations of explicit and symbolic approaches. A hybrid approach is usually based on an on-the-fly construction of an explicit graph of symbolic nodes, where each symbolic node encodes a subset of states by means of binary decision diagrams.**

An alternative to the standard Büchi automaton, called Testing automaton [6] has never been used before for hybrid model checking. In addition, in previous work [7, 8], we have shown that *Generalized Testing Automata* (TGTA) can outperform the Büchi automata for explicit and symbolic model checking of stutter-invariant LTL properties. In this work, we investigate the use of these TGTA to improve hybrid model checking. We show how traditional hybrid approaches based on Generalized Büchi Automata (TGBA) can be adapted to obtain TGTA-based hybrid approaches. Then, each original approach is experimentally compared against its TGTA variant. The results show that these new variants are statistically more efficient.

I. Introduction

Automata-theoretic approach [9] is traditionally used for the model checking of Linear-time Temporal Logic (LTL) properties. In this approach, a Kripke structure $\mathcal{K}$ is used to represent the state-space of the system, and the property to be checked is expressed as an LTL formula φ, then its negation is converted into a Büchi automaton $A_{\neg\varphi}$. The third operation is the synchronization between $\mathcal{K}$ and $A_{\neg\varphi}$. This constructs a synchronous product automaton $\mathcal{K} \otimes A_{\neg\varphi}$ whose language, $\mathscr{L}(\mathcal{K}) \cap \mathscr{L}(A_{\neg\varphi})$, is the set of executions of $\mathcal{K}$ invalidating φ. The last operation is the emptiness check algorithm that explores the product to tell whether it accepts or not an infinite word, called a counterexample. The system satisfies φ iff $\mathscr{L}(\mathcal{K} \otimes A_{\neg\varphi}) = \emptyset$.

The main problem of model checking is the well known state-space explosion problem. In particular, the performance of the automata-theoretic approach mainly depends on the size of the explored part during the emptiness check of the product automaton. The latter is often too large to be explored in a reasonable run time and memory.

There are two variants of the automata-theoretic approach: explicit and symbolic.

- In explicit model checking, the product automaton is explicitly constructed by enumerating its states. "On-the-fly" emptiness check algorithms avoid the construction of the entire product and system's state-space by building them lazily during exploration. These on-the-fly algorithms are more efficient because they stop as soon as they find a counterexample and therefore possibly before building the entire product, thereby reducing the amount of memory and time used by the emptiness check.
- The symbolic model checking tries to overcome the state-space explosion obstacle by representing the product automaton symbolically [2] by means of Binary Decision Diagrams (BDDs), i.e., concise data structures used to represent large sets of states. The intersection and union of sets of states are translated into conjunction ($\wedge$) and disjunction ($\vee$) of Boolean functions efficiently performed using BDDs. Using these symbolic operations, the product space is computed as a least fix-point on its symbolic transition relation. The emptiness check [10, 11] of this symbolic product is not performed on-the-fly.

The *hybrid* approaches [12, 13, 5, 4, 3] to LTL model checking combine ideas from both *explicit* and *symbolic* approaches in order to benefit from the advantages of both worlds, such as encoding the set of states in a concise way using decision diagrams as in the symbolic approach, and an emptiness check performed on-the-fly as in the explicit approach. In these hybrid approaches, the property automaton is described explicilty because its size is not large in most cases (and can be reduced by means of several optimizations proposed for explicit automata). However, the state-space of the system is typically very large and must be encoded symbolically.

Most of hybrid approaches are usually based on an on-the-fly construction of an explicit graph of symbolic nodes, called aggregates. Each aggregate symbolically encodes a set of states of the Kripke structure or of the product.

For instance, the hybrid approach of Biere et al. [12] replaces the Kripke structure $\mathcal{K}$ by a graph of aggregates such that each aggregate contains the set of states of $\mathcal{K}$ having the same atomic propositions values. The constructed graph is an abstraction of $\mathcal{K}$ that preserves the properties of $\mathcal{K}$ (in particular, it preserves the result of the emptiness-check of the product $\mathcal{K} \otimes A_{\neg\varphi}$).

The hybrid approach SOG (*Symbolic Observation Graph*) [13, 4] can be seen as an optimization of the approach of Biere et al. [12] for the verification of stutter-invariant [14] properties. It tries to allow further aggregation by exploiting the fact that only a subset of the atomic propositions of $\mathcal{K}$ are observed by the LTL property φ. The constructed SOG is an abstraction of the Kripke structure where consecutive states are aggregated if they share the same values of the *observed* atomic propositions.

The hybrid approach of Sebastiani et al. [5] called Property-Driven Partitioning (PDP) is based on the partitioning of the symbolic state-space of $\mathcal{K}$ according to the different explicit states of the property automaton $A_{\neg\varphi}$. This PDP approach represents the product space using an array of symbolic aggregates, one aggregate of $\mathcal{K}$ states for each explicit state of $A_{\neg\varphi}$. However, the symbolic emptiness check of this approach is not performed on-the-fly.

SLAP (*Self-Loop Aggregation Product*) [3] is an hybrid approach that combines the idea of PDP with an on-the-fly emptiness-check. SLAP is an aggregation graph alternative to the traditional product automaton. In a SLAP, the Kripke structure states are aggregated according to the valuations of the self-loops in the property automaton.

In this work, we focused on the two hybrid approaches : SOG and SLAP. Indeed, the SOG and SLAP techniques allow to manipulate the graph of aggregates simply as a variant of Büchi automata and to build this graph on-the-fly (they use the on-the-fly emptiness-check algorithms traditionally employed in automtata-based explicit approaches), moreover the computation of each aggregate is efficiently performed using symbolic fix-point algorithms. In addition, according to [4, 3], these approaches (especially SLAP) often outperform other existing (hybrid or fully symbolic) approaches.

The property automaton $A_{\neg\varphi}$ is usually a Büchi Automaton (BA) or a generalization using multiple acceptance sets, such as *Transition-based Generalized Büchi Automata* (TGBA). An alternative to these different variants of Büchi automata, called Testing Automata (TA) [15, 6] are proposed for the explicit model checking of stutter-invariant [14] properties.

In previous work [7, 8], we have shown that an improvement of TA, called *Transition-based Generalized Testing Automaton* (TGTA) is superior to TA, BA and TGBA for both explicit [7] and symbolic [8] model checking of stutter-invariant properties. The goal of this work is to show how the two hybrid approaches SOG and SLAP (that are based on TGBA) can be improved using TGTA.

II. Preliminaries

Let AP be a finite set of atomic propositions, a valuation ℓ over AP is an assignment of truth value to each atomic proposition of AP). We denote by $\Sigma = 2^{AP}$ the set of all valuations over AP, where a valuation $\ell \in \Sigma$ is interpreted either as the set of atomic propositions that are true, or as a Boolean conjunction. For instance if $AP = \{a, b\}$, then $\Sigma = 2^{AP} = \{\{a, b\}, \{a\}, \{b\}, \emptyset\}$ or equivalently $\Sigma = \{ab, a\bar{b}, \bar{a}b, \bar{a}\bar{b}\}$.

The formalization of hybrid approaches requires the following definitions introduced in [4]. We firstly recall the formalization of propositional formulas, then we present the definition of TGBA that use these propositional formulas as labels.

- $\mathbb{B} = \{\bot, \top\}$ represents the Boolean values.
- $\mathbb{B}(AP)$ is the set of all propositional formulas over AP. The formulas of $\mathbb{B}(AP)$ are built inductively from the propositions AP, $\mathbb{B}$, and the logical operators $\wedge$, $\vee$,...
- Let $\ell \in 2^{AP'}$ (i.e., a valuation over AP') and $AP \subseteq AP'$, the notation $\ell \overset{AP}{=} \ell'$ is equivalent to $\ell_{|AP} = \ell'_{|AP}$, where $\ell_{|AP}$ denotes the restriction of the valuation ℓ to the subset of atomic propositions AP. In other words, $\ell \overset{AP}{=} \ell'$ means that the valuations ℓ and ℓ' match on the atomic propositions of AP.

Duret-Lutz et al. [3] use a definition of TGBA suitable to define and implement the hybrid approach SLAP. In these TGBA, each transition is labeled with a propositional formula ϕ over AP.

A. TGBA labeled with propositional formulas

A Transition-based Generalized Büchi Automaton (TGBA) [16, 17] is a variant of a Büchi automaton that has multiple acceptance conditions on transitions.

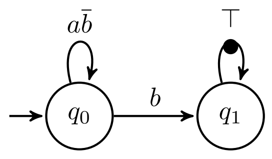
Definition 1 (TGBA)**.** *A Transition-based Generalized Büchi Automaton (TGBA) over the alphabet $\Sigma = 2^{AP}$ is a tuple $\mathcal{G} = \langle \mathcal{Q}, \mathcal{I}, \delta, \mathcal{F} \rangle$ where*

- *$\mathcal{Q}$ is a finite set of states,*
- *$\mathcal{I} \subseteq \mathcal{Q}$ is a set of initial states,*
- *$\mathcal{F} \neq \emptyset$ is a finite and non-empty set of acceptance conditions,*
- *$\delta \subseteq \mathcal{Q} \times \mathbb{B}(AP) \times 2^{\mathcal{F}} \times \mathcal{Q}$ is a transition relation, where each element $(q, \phi, F, q') \in \delta$ represents a transition from state q to state q' labeled by a propositional formula $\phi \in \mathbb{B}(AP)$, and a set of acceptance conditions $F \in 2^{\mathcal{F}}$. In the following, an element $(q, \phi, F, q') \in \delta$ will be denoted $q \xrightarrow{\phi, F} q'$*

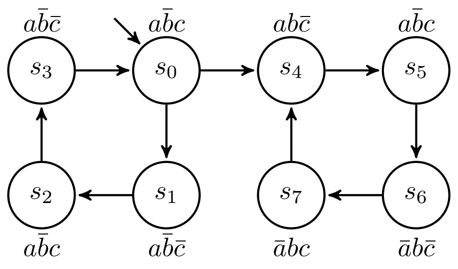
An infinite word $\sigma = \ell_0\ell_1\ell_2 \ldots \in \Sigma^\omega$ is accepted by $\mathcal{G}$ if there exists an infinite sequence of transitions $\pi = (q_0, \phi_0, F_0, q_1)(q_1, \phi_1, F_1, q_2) \cdots (q_i, \phi_i, F_i, q_{i+1}) \cdots \in \delta^\omega$ (π is called a run of $\mathcal{G}$) where:

- *$q_0 \in \mathcal{I}$, and $\forall i \in \mathbb{N}$, $\ell_i \models \phi_i$ (i.e., the infinite word σ is recognized by the run π)*
- *$\forall f \in \mathcal{F}, \forall i \in \mathbb{N}, \exists j \geq i, f \in F_j$ (i.e., the run π is accepting* iff *it visits each acceptance condition infinitely often).*

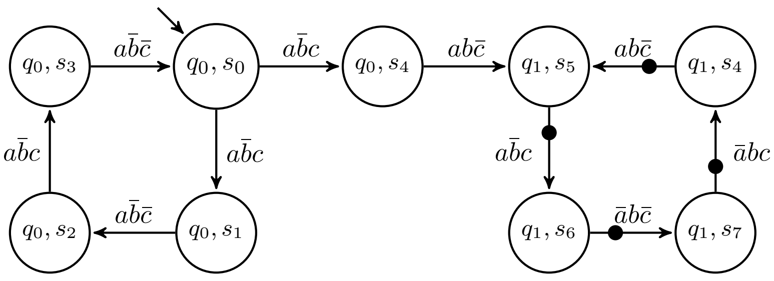
In this TGBA, each transition is labeled with a propositional formula $\phi \in \mathbb{B}(AP)$ instead of a valuation $\ell \in 2^{AP}$. This formula ϕ represents the maximal set of valuations $\{\ell_0, \ell_1, \ldots, \ell_n\} \subseteq 2^{AP}$ such that $\forall i \leq n$, $\ell_i \models \phi$. In other words, the set of valuations $\{\ell_0, \ell_1, \ldots, \ell_n\}$ is the set of all models of ϕ. For example, in the TGBA of Figure 1a, the



(a) TGBA $\mathcal{G}$ for $a\,\mathbf{U}\,b$, with acceptance conditions $\mathcal{F} = \{\bullet\}$



(b) Kripke structure $\mathcal{K}$



(c) TGBA of product $\mathcal{G} \otimes \mathcal{K}$

Fig. 1: Examples using TGBA [4]

formula $\phi = b$ labeling the transition $q_0 \xrightarrow{b} q_1$ represents the set of valuations $\{\bar{a}b, ab\}$.

Figure 1a shows the TGBA of the LTL formula $a\,\mathbf{U}\,b$. In this example, the unique acceptance condition is indicated by the black dot on the self-loop $q_1 \xrightarrow{\top,\{\bullet\}} q_1$ ($\mathcal{F} = \{\bullet\}$). The transitions are labeled with the formulas $a\bar{b}$, b and $\top$ (which encodes all the valuations over $AP = \{a, b\}$).

B. Kripke Structure

The state-space of a system can be represented by a directed graph, called Kripke structure, where vertices represent the states of the system and edges are the transitions between these states. In addition, each vertex is labeled by a valuation that represents the set of atomic propositions that are true in the corresponding state.

Definition 2 (Kripke Structure)**.** *A* Kripke structure *over a set of atomic propositions AP' is a tuple $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l \rangle$, where:*

- *$\mathcal{S}$ is a finite set of states,*
- *$\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states,*
- *$\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is the transition relation,*
- *$l : \mathcal{S} \to 2^{AP'}$ is a labeling function that maps each state s to a valuation that represents the set of atomic propositions (of AP') that are true in s.*

Fig. 1b represents an example of Kripke structure over $AP' = \{a, b, c\}$.

The synchronous product between a TGBA $\mathcal{G}$ and a Kripke structure $\mathcal{K}$ is also a TGBA ($\mathcal{G} \otimes \mathcal{K}$) that only accepts the words accepted by both $\mathcal{G}$ and $\mathcal{K}$.

Definition 3 (Synchronous product of a TGBA with a Kripke structure)**.** *For a TGBA $\mathcal{G} = \langle \mathcal{Q}, \mathcal{I}, \delta, \mathcal{F} \rangle$ over the alphabet $\Sigma = 2^{AP}$ and a Kripke structure $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l \rangle$, the product $\mathcal{G} \otimes \mathcal{K}$ is the reachable part of the TGBA $\langle \mathcal{Q}_\otimes, \mathcal{I}_\otimes, \delta_\otimes, \mathcal{F} \rangle$ over the alphabet $\Sigma = 2^{AP}$ where*

- *$\mathcal{Q}_\otimes = \mathcal{Q} \times \mathcal{S}$,*
- *$\mathcal{I}_\otimes = \mathcal{I} \times \mathcal{S}_0$,*
- *$\delta_\otimes \subseteq \mathcal{Q}_\otimes \times \mathbb{B}(AP) \times 2^{\mathcal{F}} \times \mathcal{Q}_\otimes$ where*

$$\delta_\otimes = \left\{ (q_1, s_1) \xrightarrow{l(s_1),F} (q_2, s_2) \;\middle|\; \begin{array}{l} (s_1, s_2) \in \mathcal{R} \text{ and} \\ \exists \phi \in \mathbb{B}(AP) \text{ s.t.} \\ q_1 \xrightarrow{\phi,F} q_2 \in \delta \text{ and} \\ l(s_1) \models \phi \end{array} \right\}$$

Figure 1c is an illustration of Definition 3. It shows an example of a synchronous product $\mathcal{G} \otimes \mathcal{K}$ between a TGBA $\mathcal{G}$ of $a\,\mathbf{U}\,b$ (Figure 1a) and an example of Kripke structure $\mathcal{K}$ over $AP' = \{a, b, c\}$ shown in Figure 1b. The initial state of the product is (q_0, s_0). Then, the successors $\{s_1, s_4\}$ of s_0 in $\mathcal{K}$ are synchronized with the state q_0 of $\mathcal{G}$, because the TGBA self-loop $q_0 \xrightarrow{a\bar{b}} q_0$ is labeled by the formula $\phi = a\bar{b}$ and $l(s_0) = a\bar{b}c \models \phi$. From state (q_0, s_4), the product move to state (q_1, s_5) through the TGBA transition $q_0 \xrightarrow{b} q_1$ because $l(s_4) = ab\bar{c} \models b$. From the product state (q_1, s_5), the TGBA state q_1 only requires to verify $\top$ (i.e, any valuation) to explore the self-loop labeled with the acceptance condition $\bullet$. Therefore, any cycle of $\mathcal{K}$ starting in s_5 corresponds to an accepting cycle in the product.

C. TGTA labeled with propositional formulas

In previous work [7, 8] we introduced a new kind of automata, called Transition-based Generalized Testing Automata (TGTA) that only recognize stutter-invariant [14] LTL properties.

While a TGBA observes the value of the atomic propositions AP, a TGTA observes the *changes* in these values. If a valuation of AP does not change between two consecutive states, we say that a TGTA executes a *stuttering transition*.

If A and B are two valuations, $A \oplus B$ denotes the symmetric set difference (called "changeset"), i.e., the set of atomic propositions that differ (e.g., $a\bar{b} \oplus ab = \{a\} \oplus \{a, b\} = \{b\}$).

Similar to TGBA, we present in this section a definition of a TGTA labeled with propositional formulas, more suited than the definition of [7] to define and implement the TGTA-based hybrid approaches presented later in this paper.

Definition 4 (TGTA [8]). *A Transition-based Generalized Testing Automaton (TGTA) over the alphabet $\Sigma = 2^{AP}$ is a tuple $\mathcal{T} = \langle \mathcal{Q}, \mathcal{I}, U, \delta, \mathcal{F} \rangle$ where*

- *$\mathcal{Q}$ is a finite set of states,*
- *$\mathcal{I} \subseteq \mathcal{Q}$ is a set of initial states,*
- *$U : \mathcal{I} \to \mathbb{B}(AP)$ is a function mapping each initial state to a propositional formula $\phi \in \mathbb{B}(AP)$,*
- *$\mathcal{F} \neq \emptyset$ is a finite and non-empty set of acceptance conditions,*
- *$\delta \subseteq \mathcal{Q} \times \mathbb{B}(AP) \times 2^{\mathcal{F}} \times \mathcal{Q}$ is a transition relation, where each element $(q, \phi, F, q') \in \delta$ represents a transition from state q to state q' labeled by a propositional formula $\phi \in \mathbb{B}(AP)$, and a set of acceptance conditions $F \in 2^{\mathcal{F}}$.*
  The propositional formula ϕ encodes a set of changesets *$\{k_0, k_1, \dots, k_n\} \subseteq 2^{AP}$, where $\forall i \leq n$, each* changeset *k_i is interpreted as a set of atomic propositions whose values change between q and q',*
- *δ has to satisfy the following* stuttering-normalization constraint *[8]:*
  All stuttering transitions are self-loops, and every state has a stuttering self-loop: $\forall (q, q') \in \mathcal{Q}^2 : (\exists (\phi, F) \in \mathbb{B}(AP) \times 2^{\mathcal{F}}, \emptyset \models \phi \wedge (q, \phi, F, q') \in \delta) \iff (q = q')$

An infinite word $\sigma = \ell_0 \ell_1 \ell_2 \ldots \in \Sigma^\omega$ is accepted by $\mathcal{T}$ iff *there exists an infinite sequence of transitions $\pi = (q_0, \phi_0, F_0, q_1)(q_1, \phi_1, F_1, q_2) \cdots (q_i, \phi_i, F_i, q_{i+1}) \cdots \in \delta^\omega$ (π is called a run of $\mathcal{T}$) where:*

- *$q_0 \in \mathcal{I}$, and $\ell_0 \models U(q_0)$*
- *$\forall i \in \mathbb{N}, (\ell_i \oplus \ell_{i+1}) \models \phi_i$ (i.e., the infinite word σ is recognized by the run π)*
- *$\forall f \in \mathcal{F}, \forall i \in \mathbb{N}, \exists j \geq i, f \in F_j$ (i.e., the run π is accepting).*

The difference between TGBA and TGTA resides mainly in the interpretation of the transition relation $\delta \subseteq \mathcal{Q} \times \mathbb{B}(AP) \times 2^{\mathcal{F}} \times \mathcal{Q}$. In TGTA, for each transition $(q, \phi, F, q') \in \delta$, the formula ϕ encodes a set of changesets $\{k_0, k_1, \dots, k_n\} \subseteq 2^{AP}$, where $\forall i \leq n$, each changeset $k_i \models \phi$. In the following, we will use (q, ϕ, F, q') and $(q, \{k_0, \dots, k_n\}, F, q')$ interchangeably as transitions of δ. For example, in Figure 2a, the formula $\phi = b$ labeling the transition $q_0 \xrightarrow{b} q_1$ represents the set of changesets $\{\{b\}, \{a, b\}\}$ (meaning that the value of b changes between q_0 and q_1 and we "do not care" about a).

Figure 2a shows the TGTA of the LTL formula $a \,\mathbf{U}\, b$. The valuations of the initial states are $U(q_0) = a\bar{b}$ and $U(q_1) = b = \{ab, \bar{a}b\}$. The formula $\bar{a}\bar{b}$ labeling the stuttering self-loop on q_0 is the empty changeset $\emptyset$. The formula b labeling the transition $q_0 \xrightarrow{b} q_1$ is obtained by merging the two changesets of the transitions $q_0 \xrightarrow{\{b\}} q_1$ and $q_0 \xrightarrow{\{a,b\}} q_1$ (because $\{\{b\}, \{a, b\}\} \models b$). On the self-loop $q_1 \xrightarrow{\top, \{\bullet\}} q_1$, the formula $\top$ is obtained by merging the set of all changesets over $AP = \{a, b\}$, and the acceptance condition is indicated by the black dot ($\mathcal{F} = \{\bullet\}$).

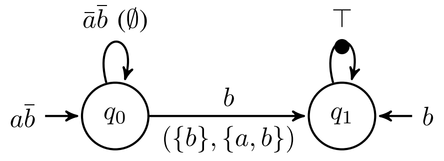
Using this definition of TGTA, we obtain the following definition of the synchronous product between a TGTA and a Kripke structure. This synchronous product is also a TGTA:

Definition 5 (Synchronous product of a TGTA with a Kripke structure). *For a TGTA $\mathcal{T} = \langle \mathcal{Q}, \mathcal{I}, U, \delta, \mathcal{F} \rangle$ over the alphabet $\Sigma = 2^{AP}$ and a Kripke structure $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l \rangle$, the product $\mathcal{T} \otimes \mathcal{K}$ is the reachable part of the TGTA $\langle \mathcal{Q}_\otimes, \mathcal{I}_\otimes, U_\otimes, \delta_\otimes, \mathcal{F} \rangle$ over the alphabet $\Sigma = 2^{AP}$ where*

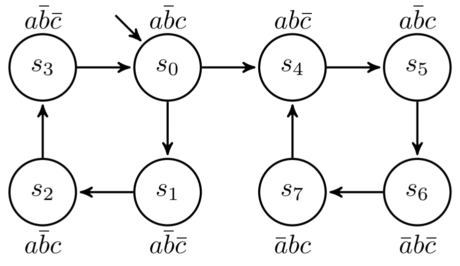
- *$\mathcal{Q}_\otimes = \mathcal{Q} \times \mathcal{S}$,*
- *$\mathcal{I}_\otimes = \{(q, s) \in \mathcal{I} \times \mathcal{S}_0 \mid l(s) \models U(q)\}$*
- *$\forall (q, s) \in \mathcal{I}_\otimes, U_\otimes((q, s)) = l(s)$,*
- *$\delta_\otimes \subseteq \mathcal{Q}_\otimes \times \mathbb{B}(AP) \times 2^{\mathcal{F}} \times \mathcal{Q}_\otimes$ where:*

$$\delta_\otimes = \left\{ (q_1, s_1) \xrightarrow{l(s_1) \oplus l(s_2)_{|AP}, F} (q_2, s_2) \;\middle|\; \begin{array}{l} (s_1, s_2) \in \mathcal{R} \text{ and} \\ \exists \phi \in \mathbb{B}(AP) \text{ s.t.} \\ q_1 \xrightarrow{\phi, F} q_2 \in \delta \wedge \\ l(s_1) \oplus l(s_2) \models \phi \end{array} \right\}$$

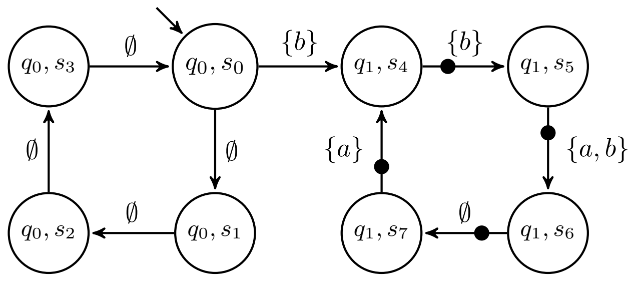
Figure 2c is an illustration of Definition 5. It shows an example of a synchronous product $\mathcal{T} \otimes \mathcal{K}$ between the TGTA $\mathcal{T}$ of $a \,\mathbf{U}\, b$ (Figure 2a) and the Kripke structure $\mathcal{K}$ (Figure 2b). The initial state of the product is (q_0, s_0) because $l(s_0) = a\bar{b}c \models U(q_0) = a\bar{b}$. Then, (q_0, s_0) have two successors: the first successor is (q_0, s_1) because $\mathcal{K}$ has a transition $s_0 \to s_1$ with $l(s_0) \oplus l(s_1) = \emptyset \models \bar{a}\bar{b}$ and $\mathcal{T}$ have a stuttering self-loop $q_0 \xrightarrow{\bar{a}\bar{b}} q_0$; the second successor is (q_1, s_4) because in $\mathcal{K}$ we have $s_0 \to s_4$ with $l(s_0) \oplus l(s_4) = \{b, c\} \models b$ and the TGTA have a transition $q_0 \xrightarrow{b} q_1$ labeled with $\phi = b$. From the product state (q_1, s_4), the TGTA can explore any changeset through the self-loop labeled with $\top$ and the acceptance condition $\bullet$. Therefore, the TGTA state q_1 can be synchronized with any reachable state from s_4 in $\mathcal{K}$ and the cycle $(q_1, s_4) \to (q_1, s_5) \to (q_1, s_6) \to (q_1, s_7) \to (q_1, s_4)$ is an accepting cycle in the product. In the obtained product $\mathcal{T} \otimes \mathcal{K}$, each transition is labeled with the changeset $((l(s_1) \oplus l(s_2))_{|AP})$ between the states of $\mathcal{K}$. These changesets are computed according to the set of atomic propositions $AP = \{a, b\}$ observed by $\mathcal{T}$. The product transitions also bear the acceptance conditions coming from the TGTA $\mathcal{T}$.



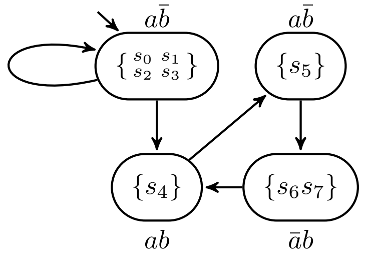
(a) TGTA $\mathcal{T}$ for $a\,\mathbf{U}\,b$. The formula $a\bar{b}$ labeling the self-loop on q_0 represents the empty changeset $\emptyset$. The formula b labeling the transition $q_0 \xrightarrow{b} q_1$ represents the set of changesets $\{\{b\},\{a,b\}\}$. The label $\top$ of the self-loop on q_1 encodes the set of all changesets $\{\emptyset,\{a\},\{b\},\{a,b\}\}$.



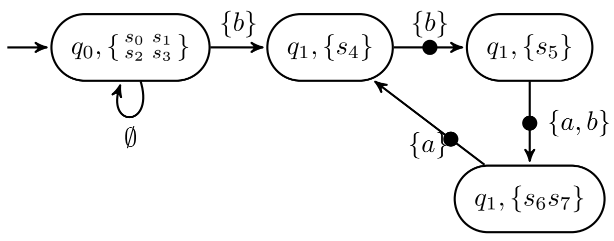
(b) Kripke structure $\mathcal{K}$



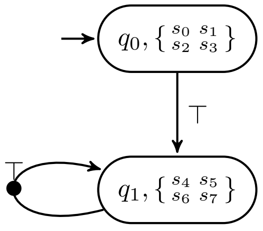
(c) TGTA of product $\mathcal{T} \otimes \mathcal{K}$



(d) SOG for TGTA: a SOG-TGTA $\widehat{\mathcal{K}}_{\{a,b\}}$



(e) Product $\mathcal{T} \otimes \widehat{\mathcal{K}}_{\{a,b\}}$ (TGTA $\otimes$ SOG-TGTA)



$q_0, \{ {s_0\ s_1 \atop s_2\ s_3} \}$
$\top$
$\top$
$q_1, \{ {s_4\ s_5 \atop s_6\ s_7} \}$

(f) SLAP-TGTA $\mathcal{T} \boxtimes \mathcal{K}$

Fig. 2: Examples Using TGTA

III. Symbolic Observation Graph for TGTA (SOG-TGTA)

In this section, we propose an adaptation of the SOG hybrid approach for use with TGTA instead of TGBA.

An SOG [13, 4] is a transformation of a Kripke structure allowing to aggregate states according to the set AP of atomic propostions observed in the property automaton. This transformation only preserves stutter-invariant properties. The constructed SOG is an explicit graph where each node is a symbolic set of states. Theses states are aggregated because they share the same values for the atomic propositions of AP (they may have different values for the other atomic propositions of the system (in AP') that are not in AP).

In hybrid approaches, symbolic data structures are used to represent sets of states of the Kripke structure. The following symbolic operations are introduced in [4] to manipulate this symbolic aggregate of states.

Notations Let $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l \rangle$ a Kripke structure, for a set of states $a \subseteq \mathcal{S}$ and a propositional formula $\phi \in \mathbb{B}(AP)$, the symbolic operations $\mathrm{SuccF}(a,\phi)$ and $\mathrm{ReachF}(a,\phi)$ are defined as follows:

$\mathrm{SuccF}(a,\phi) = \{s' \in \mathcal{S} \mid \exists s \in a, (s \to s' \in \mathcal{R}) \wedge l(s') \models \phi\}$, i.e., the set of the **Succ**essors of states of aggregate a, **F**iltered to keep only those satisfying ϕ. $\mathrm{ReachF}(a,\phi)$ computes the least subset of $\mathcal{S}$ satisfying:

- $a \subseteq \mathrm{ReachF}(a,\phi)$,
- $\mathrm{SuccF}(\mathrm{ReachF}(a,\phi),\phi) \subseteq \mathrm{ReachF}(a,\phi)$.

$\mathrm{ReachF}(a,\phi)$ is implemented using symbolic least fixed-points on Decision Diagrams.

Definition 6 (Homogeneous aggregate [4]). *Let $a \in 2^{\mathcal{S}} \setminus \{\emptyset\}$ be a subset of states of $\mathcal{K}$. We say that a is* a homogeneous aggregate *w.r.t.* (*with* ***r****espect* ***t****o) a given set of atomic propositions AP* iff $\forall s, s' \in a, l(s) \overset{AP}{=} l(s')$. *In other words, all the* *states of the aggregate a have the same valuation for all the atomic propositions in AP.*

For a homogeneous aggregate a w.r.t. AP, we write $l_{AP}(a) = l(s)_{|AP}$ for any state $s \in a$ (i.e., the valuation of a is the valuation of any one of its states).

The variant of SOG used in a TGBA-based approach was detailed in [4]. In this section, in our TGTA approach, we use another variant of SOG proposed in [13], this variant is called in this work SOG-TGTA because it is better suited for being employed in a TGTA-based approach. This SOG-TGTA does not use the divergent states used in the variant of SOG proposed in [4]. Instead of using divergent states, this SOG-TGTA has a self-loop on each aggregate that contains a cycle [13]. Because in a TGTA, all stuttering transitions are self-loops (see the *stuttering-normalization constraint* of Definition 4), then the synchronization between these stuttering self-loops and the self-loops of SOG-TGTA only produces self-loops in the product automaton, and therefore does not generate new states in this product.

Definition 7 (Symbolic Observation Graph [13] for TGTA (SOG-TGTA))**.** *Let $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l\rangle$ be a Kripke structure over AP'. A* SOG-TGTA *over $AP \subseteq AP'$ of $\mathcal{K}$ is defined as $\widehat{\mathcal{K}}_{AP} = \langle \widehat{\mathcal{S}}, \widehat{\mathcal{S}}_0, \widehat{\mathcal{R}}, \widehat{l}\rangle$ a Kripke structure over AP satisfying:*

1) $\widehat{\mathcal{S}} = \left\{ a \in 2^{\mathcal{S}} \setminus \{\emptyset\} \;\middle|\; \begin{array}{l} a \textit{ is homogeneous w.r.t. } AP \\ a = \mathrm{ReachF}(a, l_{AP}(a)) \end{array} \right\}$
2) $\forall a \in \widehat{\mathcal{S}},\ \widehat{l}(a) = l_{AP}(a)$
3) $\widehat{\mathcal{R}} = \{a \to a' \in \widehat{\mathcal{S}} \times \widehat{\mathcal{S}} \mid$
$\quad a' = \mathrm{ReachF}(\mathrm{SuccF}(a, \widehat{l}(a')) \setminus a, \widehat{l}(a'))\}$
$\quad \cup \{a \to a \in \widehat{\mathcal{S}} \times \widehat{\mathcal{S}} \mid a \text{ contains a cycle } \}$
4) $\widehat{\mathcal{S}}_0 = \{\mathrm{ReachF}(\{s_0\}, l(s_0)_{|AP}) \mid s_0 \in \mathcal{S}_0\}$.

The above Definition details how to build a SOG-TGTA $\widehat{\mathcal{K}}_{AP}$. The set $\widehat{\mathcal{S}}_0$ of initial states of $\widehat{\mathcal{K}}_{AP}$ is composed by the set of homogenous aggregates a_{s_0} satisfying $a_{s_0} = \mathrm{ReachF}(\{s_0\}, l(s_0)_{|AP})$, i.e., for each initial state s_0 of $\mathcal{K}$, a_{s_0} is the set of reachable states s' from s_0 in $\mathcal{K}$ as long as $l(s') \overset{AP}{=} l(s_0)$.

The set $\widehat{\mathcal{S}}$ of states of $\widehat{\mathcal{K}}_{AP}$ is composed of homogenous aggregates a satisfying $a = \mathrm{ReachF}(a, l_{AP}(a))$.

For the transition relation of $\widehat{\mathcal{K}}_{AP}$, $\widehat{\mathcal{R}}$ is composed of two kinds of edges:

1) Case a and a' are two aggregates of $\widehat{\mathcal{S}}$:
$a \to a' \in \widehat{\mathcal{R}}$ iff $l_{AP}(a) \neq l_{AP}(a')$ and a' is the least subset of $\mathcal{S}$ that contains every state $s' \in \mathcal{S}$ satisfying $l(s')_{|AP} = \widehat{l}(a')$ and $s \to s' \in \mathcal{R}$ with $s \in (a \cup a')$.
2) Case a is an aggregates of $\widehat{\mathcal{S}}$:
$a \to a \in \widehat{\mathcal{R}}$ iff a contains a cycle (i.e., the Kripke structure $\mathcal{K}$ contains a cycle visiting only states in a). In other words, the divergent states of the SOG proposed in [4] are replaced in SOG-TGTA by adding a self-loop on each aggregate that contains a cycle (see point 3 of Definition 7). These cycles are computed using symbolic least fixed-points. The obtained SOG-TGTA contains only one kind of nodes: homogenous aggregates.

Figure 2d shows an example of a SOG-TGTA $\widehat{\mathcal{K}}_{\{a,b\}}$ built from the Kripke structure $\mathcal{K}$ of Figure 1b according to $AP = \{a, b\}$ (ignoring the atomic proposition c of $\mathcal{K}$). The initial state of $\widehat{\mathcal{K}}_{\{a,b\}}$ is an aggregate $\{s_0, s_1, s_2, s_3\}$ because they agree on the value of atomic propositions observed in $AP = \{a, b\}$: $l(s_0)_{|\{a,b\}} = l(s_1)_{|\{a,b\}} = l(s_2)_{|\{a,b\}} = l(s_3)_{|\{a,b\}} = a\bar{b}$. In addition, the initial aggregate of $\widehat{\mathcal{K}}_{\{a,b\}}$ has a self-loop because it contains a cycle. The constructed SOG $\widehat{\mathcal{K}}_{\{a,b\}}$ is also a Kripke structure, that allows to check any stutter-invariant property over the alphabet $2^{\{a,b\}}$. As example, Figure 2e presents the product $\mathcal{T} \otimes \widehat{\mathcal{K}}_{\{a,b\}}$ of $\widehat{\mathcal{K}}_{\{a,b\}}$ with the TGTA $\mathcal{T}$ of $a \,\mathbf{U}\, b$.

Theorem 1 ([13])**.** *Given a Kripke Structure $\mathcal{K}$ defined on AP', then the SOG-TGTA $\widehat{\mathcal{K}}_{AP}$ of $\mathcal{K}$ built over $AP \subseteq AP'$ preserves any stutter-invariant property $\mathcal{A}$ on AP. In other words: $\mathscr{L}(\mathcal{A} \otimes \mathcal{K}) \neq \emptyset \iff \mathscr{L}(\mathcal{A} \otimes \widehat{\mathcal{K}}_{AP}) \neq \emptyset$.*

IV. Self-Loop Aggregation Product for TGTA (SLAP-TGTA)

SLAP [3] is a hybrid synchronous product, in which the aggregation of Kripke structure states is based on the self-loops of the property automaton.

Definition 8. *Given a TGBA $\mathcal{G} = \langle \mathcal{Q}, \mathcal{I}, \delta, \mathcal{F}\rangle$ or a TGTA $\mathcal{T} = \langle \mathcal{Q}, \mathcal{I}, U, \delta, \mathcal{F}\rangle$, for a state $q \in \mathcal{Q}$,* $\mathrm{SF}(q)$ *encodes the* ***S****elf-loop* ***F****ormulas labeling edges $q \to q$. Formally,*

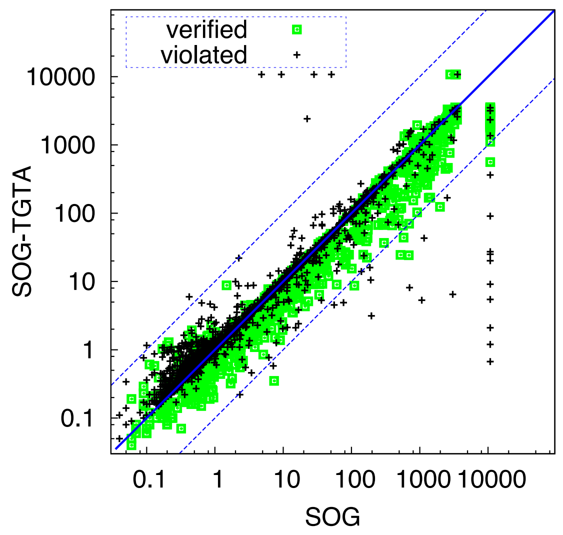
$$\mathrm{SF}(q) = \bigvee_{q \xrightarrow{\phi, F} q \in \delta} \phi$$

A SLAP [3] is a hybrid product between a Kripke structure $\mathcal{K}$ and a TGBA $\mathcal{G}$. The states of the SLAP are pairs of the form (q, a) composed of a state q of $\mathcal{G}$ and an aggregate a containing successive states of $\mathcal{K}$ aggregated as long as they model $\mathrm{SF}(q)$. These aggregates are computed as symbolic least fixed-points.

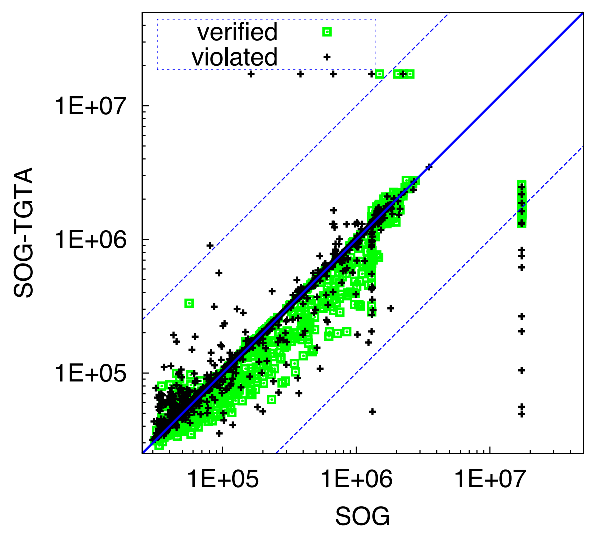
The SLAP-TGTA is a variant of SLAP based on TGTA instead of TGBA. In SLAP-TGTA, the states of the Kripke structure are aggregated according to the changesets labeling the TGTA transitions. In particular, each $\mathrm{SF}(q)$ represents the set of changesets encoded by the **S**elf-loop **F**ormulas labeling edges $q \to q$ of the TGTA. Therefore, in SLAP-TGTA the successive states of the Kripke structure are aggregated as long as they change according to the changesets encoded by $\mathrm{SF}(q)$. These aggregates are computed as least fixed-points based on changesets using the symbolic operations $\mathrm{Succ}^{\oplus}$ and $\mathrm{Reach}^{\oplus}$ defined as follows:

Let $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l\rangle$ a Kripke structure. For a set of states $a \subseteq \mathcal{S}$ and a propositional formula $\phi \in \mathbb{B}(AP)$, we define the following symbolic operations:

- $\mathrm{Succ}^{\oplus}(a, \phi) = \{s' \in \mathcal{S} \mid \exists s \in a, (s \to s' \in \mathcal{R}) \wedge ((l(s) \oplus l(s')) \models \phi)\}$, i.e, the set of the **Succ**essors states of a Filtered to keep only those satisfying $k \models \phi$ where $k = l(s) \oplus l(s')$ is the changeset between $l(s)$ and $l(s')$.
- $\mathrm{Reach}^{\oplus}(a, \phi)$ computes the least subset of $\mathcal{S}$ satisfying:
  - ∘ $a \subseteq \mathrm{Reach}^{\oplus}(a, \phi)$,
  - ∘ $\mathrm{Succ}^{\oplus}(\mathrm{Reach}^{\oplus}(a, \phi), \phi) \subseteq \mathrm{Reach}^{\oplus}(a, \phi)$.



(a) Time



(b) Memory

Fig. 3: Performance comparison of SOG-TGTA against SOG. Left: time (in seconds); Right: memory (in kilobytes).

Definition 9 (SLAP of a TGTA and a Kripke structure). *Given a TGTA $\mathcal{T} = \langle \mathcal{Q}, \mathcal{I}, U, \delta, \mathcal{F} \rangle$ over AP and a Kripke structure $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l \rangle$, the* SLAP-TGTA *of $\mathcal{T}$ and $\mathcal{K}$ is the TGTA denoted $\mathcal{T} \boxtimes \mathcal{K} = \langle \mathcal{Q}_\boxtimes, \mathcal{I}_\boxtimes, \delta_\boxtimes, \mathcal{F} \rangle$ where:*

- $\mathcal{Q}_\boxtimes = \mathcal{Q} \times (2^{\mathcal{S}} \setminus \{\emptyset\})$
- $\delta_\boxtimes = \left\{ (q_1, a_1) \xrightarrow{\top, F} (q_2, a_2) \;\middle|\; \begin{array}{l} \exists \phi \in \mathbb{B}(AP) \textit{ such that} \\ q_1 \xrightarrow{\phi, F} q_2 \in \delta, \textit{ with} \\ (q_1 = q_2) \Rightarrow (F \neq \emptyset) \wedge \\ a_2 = \mathrm{Reach}^\oplus(\\ \quad \mathrm{Succ}^\oplus(a_1, \phi), \mathrm{SF}(q_2)) \end{array} \right\}$
- $\mathcal{I}_\boxtimes = \{(q_0, \mathrm{Reach}^\oplus(\{s_0\}, \mathrm{SF}(q_0))) \mid$
  $(q_0, s_0) \in \mathcal{I} \times \mathcal{S}_0$ *and* $l(s_0) \models U(q_0)\}$

We have $\mathscr{L}(\mathcal{T} \otimes \mathcal{K}) \neq \emptyset \iff \mathscr{L}(\mathcal{T} \boxtimes \mathcal{K}) \neq \emptyset$ by construction.

For the same reason as in SLAP [3], the SLAP-TGTA transitions are only labeled with $\top$ because these labels are irrelevant when checking language emptiness of SLAP-TGTA.

The reachable states of a SLAP-TGTA are of the form (q, a) where q is a state of the TGTA and a is an aggregate of states of the Kripke structure such that: for each state $s \in a$, if s' is a successor of s in the Kripke structure with $l(s) \oplus l(s') \models \mathrm{SF}(q)$, then $s' \in a$.

Figure 2f presents $\mathcal{T} \boxtimes \mathcal{K}$, an example of SLAP-TGTA computed from the Kripke structure $\mathcal{K}$ and the TGTA $\mathcal{T}$ of $a \,\mathbf{U}\, b$. Because $l(s_0) = a\bar{b}c \models U(q_0) = a\bar{b}$, the initial state of $\mathcal{T} \boxtimes \mathcal{K}$ is the pair (q_0, a_1), where a_1 is computed from s_0 by iteratively aggregates successors that change according to a changeset belonging to (the set of changesets encoded by) $\mathrm{SF}(q_0)$. Formally, $a_1 = \mathrm{Reach}^\oplus(\{s_0\}, \mathrm{SF}(q_0)) = \{ \begin{smallmatrix} s_0 & s_1 \\ s_2 & s_3 \end{smallmatrix} \}$ (because $l(s_0) \oplus l(s_1) = l(s_1) \oplus l(s_2) = l(s_2) \oplus l(s_3) = \emptyset \models \mathrm{SF}(q_0) = \bar{a}\bar{b}$). Then, in order to compute the successors of (q_0, a_1), we explore the transition $q_0 \xrightarrow{b} q_1$ of TGTA. We obtain only one successor (q_1, a_2) with the aggregate $a_2 = \mathrm{Reach}^\oplus(\mathrm{Succ}^\oplus(a_1, b), \mathrm{SF}(q_1))$ is computed as follows:

- $\mathrm{Succ}^\oplus(a_1, b) = \{s_4\}$ because s_4 is a successor of $s_0 \in a_1$ and it is the unique successor of states of a_1 that satisfies $l(s_0) \oplus l(s_4) \models b$ ($l(s_0) \oplus l(s_4) = \{b, c\}$),
- $\mathrm{SF}(q_1) = \top$ because $q_1 \xrightarrow{\top} q_1$
- Thus, $a_2 = \mathrm{Reach}^\oplus(\{s_4\}, \top) = \{s_4, s_5, s_6, s_7\}$ because a_2 contains all the reachable states from s_4 through any changeset.

Finally, we compute the successors of (q_1, a_2) by exploring the TGTA transition $q_1 \xrightarrow{\top, \bullet} q_1$. This TGTA accepting self-loop also generates an accepting self-loop on state (q_1, a_2) of the SLAP-TGTA. Indeed, the unique successor of (q_1, a_2) is itself because:
$\mathrm{Reach}^\oplus(\mathrm{Succ}^\oplus(a_2, \top), \top) = a_2$ (the states of a_2 are in a cycle).

V. Experimental Comparison of Hybrid Approaches using TGBA vs. TGTA

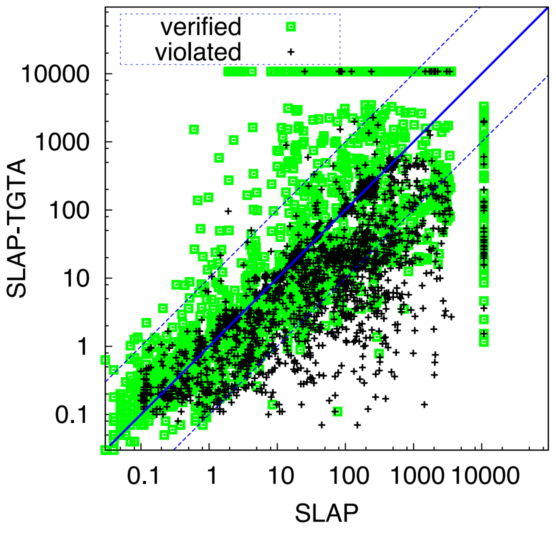
This section presents an experimental evaluation conducted to compare each hybrid approach (SOG and SLAP) with its variant based on TGTA (SOG-TGTA and SLAP-TGTA). This experimentation is based on BEEM benchmark [18]. It reuses the same benchmark inputs, formulas and models used in [8] to evaluate the symbolic approach using TGTA.

A. Implementation

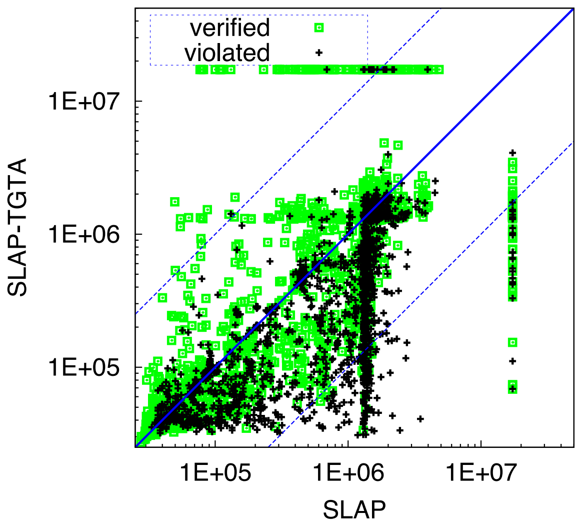
We have implemented SOG-TGTA and SLAP-TGTA in the same tool LTL-ITS [1] that already contains SOG and SLAP. LTL-ITS tool is built on top of libraries [2]: SDD/ITS, Spot,

[1]http://ddd.lip6.fr

[2]Respectively http://ddd.lip6.fr, http://spot.lip6.fr, and http://fmt.cs.utwente.nl/tools/ltsmin.



(a) Time



(b) Memory

Fig. 4: Performance comparison of SLAP-TGTA against SLAP. Left: time (in seconds); Right: memory (in kilobytes).

and LTSmin. These three libraries were already presented in the previous work [8].

The DVE variant of LTSmin [19] is used to produce an ETF file representing the transition relation of each BEEM model. These ETF files are loaded by the SDD/ITS [20] library to encode the symbolic transition relations of the Kripke structures, used to implement the symbolic operations ReachF. For SLAP-TGTA, it is a changeset-based [8] symbolic transition relations that are built from the ETF files. It is used to implement the symbolic operation $\mathrm{Reach}^{\oplus}$ for the SLAP-TGTA aggregates computation.

The Spot library [17] is used to translate the LTL properties into TGBA or TGTA. It is also used to perform the emptiness check of explicit graphs, such as the synchronous products between TGBA and SOG and between TGTA and SOG-TGTA. In addition, the hybrid synchronous products SLAP and SLAP-TGTA are also handled by the emptiness check of Spot. Indeed, they are explicit graphs in which each node stores a set of states encoded as a Decision Diagram. These sets of states are computed using least fixed-points (ReachF or $\mathrm{Reach}^{\oplus}$).

A SOG-TGTA is implemented in the same way as a SOG, as a concrete class of the Kripke structure interface provided by Spot. During the emptiness check of the products TGBA/SOG and TGTA/SOG-TGTA, the SOG and the SOG-TGTA nodes are constructed on-the-fly using the implementation of the symbolic operation ReachF.

Similar to SLAP, we have implemented SLAP-TGTA as concrete classes of the synchronous product interface of Spot. During the emptiness check, the nodes of these two hybrid products are built on-the-fly from the states of the property automaton (TGBA or TGTA) and using the symbolic operations: FReach for SLAP [3] and $\mathrm{Reach}^{\oplus}$ for SLAP-TGTA.

In the next section, the results of our experimental comparisons are presented as scatter plots using logarithmic scale. In these experimentations, we check that the different approaches (SOG, SLAP, SOG-TGTA and SLAP-TGTA) always gave the same result on each pair (formula, model).

B. *SOG versus SOG-TGTA*

The scatter plots of Figure 3 compare the performance of two hybrid approaches: the first is based on TGBA and SOG (called just SOG approach in the following); the second is based on TGTA and SOG-TGTA (called SOG-TGTA approach). Each point of the left and right scatter plots compares respectively the time and memory used in the model checking of each pair (formula, model). The x-axis represents the performance of the SOG approach and the y-axis shows the performance of the SOG-TGTA approach, so the points below the diagonal correspond to cases where the SOG-TGTA approach is better. Symmetrically, the points above the diagonal correspond to points were the SOG approach is better. The points represented by gray squares correspond to verified formulas (empty products), and the black crosses correspond to violated formulas (non-empty products).

In the two scatter plots, we observe that for verified formulas (gray points), the SOG-TGTA approach outperforms the SOG approach (in time and memory). This result is similar to the comparison between the explicit approaches based on TGBA versus TGTA, presented in [7]. This similarity is justified by the fact that these approaches are based on traditional explicit synchronous products.

For violated formulas, the SOG approach outperforms the SOG-TGTA approach for only the cases where the execution time is less than one second. On the contrary, for hard cases, there are more cases that failed using SOG than when using SOG-TGTA (compare the “aligned” black points at top and on right of the scatter plots).

In total in these scatter plots, SOG failed in 53 cases solved by SOG-TGTA, while the SOG-TGTA approach failed for only

9 cases solved by SOG. In addition, 875 cases are not solved by any of the two approaches within the time and memory limits. In the experiments that have not failed, SOG-TGTA was at least ten times faster than SOG in 20 cases, and two times faster in 287 cases. A contrario, SOG was at least ten times faster than SOG-TGTA in 4 cases, and two times faster in 144 cases.

C. *SLAP versus SLAP-TGTA*

The scatter plots of Figure 4 compare the performance of SLAP against SLAP-TGTA. The left scatter plot compares the time performance and the right concerns the memory consumption. The points below the diagonal correspond to the cases where SLAP-TGTA is better.

The interpretation of the scatter plots results depends on the colors of the points:

- For black crosses that correpond to violated formulas, in most cases, SLAP-TGTA is more efficient than SLAP to find a counterexample. Therefore, SLAP-TGTA should be a valuable help to debug systems (i.e. when counterexamples are *expected* in the "debugging phase").
- For the gray squares representing the verified formulas, the scatter plots are difficult to interpret, there are many cases on both sides of the diagonal.

In total, we observe a relative advantage for SLAP-TGTA in cases where the two approaches were successful. Indeed, on the one hand, SLAP-TGTA was at least a hundred times faster than SLAP in 85 cases, ten times faster in 646 cases, and twice times faster in 1605 cases. On the other hand, SLAP was at least one hundred times faster than SLAP-TGTA in 12 cases, ten times faster in 95 cases, and two times faster in 434 cases only. However, for the failed cases, SLAP-TGTA failed in 277 experiments solved by SLAP, while SLAP failed for only 85 cases solved by SLAP-TGTA.

We believe that the two approaches SLAP and SLAP-TGTA are complementary and very different because of the fact that SLAP-TGTA aggregates are based on changesets and therefore are very different from SLAP aggregates that are based on valuations. Thus, these two approaches can be considered complementary and can be launched in parallel in order to retrieve the result of the fastest approach.

VI. Conclusion

In previous work [7, 8], we have shown that *Transition-based Generalized Testing Automata* (TGTA) are a way to improve the explicit and symbolic model checking approach when verifying stutter-invariant properties, but they had not been used for hybrid model checking. In this paper, we gave the first hybrid approaches using a variant of Testing Automata, and compare it to more classical hybrid approaches (using TGBA). We propose two hybrid approaches using TGTA: SOG-TGTA and SLAP-TGTA.

SOG-TGTA is a variant of SOG without divergent states, which are replaced in SOG-TGTA by adding a self-loop on each aggregate that contains a cycle. Adding these self-loop in SOG-TGTA is better than adding divergent states because in TGTA all stuttering transitions are self-loops, and therefore adding self-loops in SOG-TGTA does not generate new states in the product between TGTA and SOG-TGTA.

SLAP-TGTA is an adaptation of SLAP to use TGTA instead of TGBA. The two variants (SLPA and SLAP-TGTA) are based on the aggregation of the states of a Kripke structure according to the self-loops of the formula automaton (i.e., TGBA for SLAP and TGTA for SLAP-TGTA). However, the obtained aggregates are very different between the two variants. In a SLAP-TGTA, the states of an aggregate change according to the changesets of the TGTA self-loops (instead of satisfying the valuations labeling the TGBA self-loops in the case of SLAP).

We implemented and experimentally compared the performance of each TGTA based hybrid approach (SOG-TGTA and SLAP-TGTA) against its reference variant (SOG and SLAP). The obtained results show that SOG-TGTA is statistically better than SOG, especially to prove that a property is verified. For SLAP versus SLAP-TGTA, we observed an advantage for SLAP-TGTA to find a counterexample for violated properties (in the "debugging phase"). The results are more difficult to interpret for verified properties: there are many cases where SLAP-TGTA is better than SLAP and vice versa. This difference between SLAP and SLAP-TGTA can be explained by the fact that the aggregates computed using TGBA and TGTA are very different (in SLAP-TGTA, the aggregates are based on changesets instead of valuations as in SLAP). We believe that the two variants (using TGBA versus TGTA) are complementary, and the best solution is to run the two variants in parallel, then take the result of the faster one.

In future work, we plan to evaluate the use of TGTA in another hybrid approach called SSP (*Symbolic Synchronized Product*) [21]. Similar to SLAP, the SSP replaces the product automaton $\mathcal{K} \otimes A_{\neg\varphi}$ by a smaller product graph that preserves the result of the emptiness-check. In order to perform this reduction, the SSP approach aggregates "symmetrically equivalent" states by exploiting the symmetries in the model according to the states of the property automaton.

Another future work is to combine the TGTA-based approaches with other techniques that propose state-space optimizations specific to stutter-invariant properties, such as the partial order reduction [22, 23, 24]. TGTA and partial order reduction are complementary. Indeed, while the TGTA-based approaches focus on optimizing the property automata, the partial order techniques try to reduce the state-space of the model.

Finally, TGTA are less expressive than TGBA since they are able to represent only stutter-invariant LTL properties (LTL$\setminus$X). Another future work is to extend TGTA to represent all LTL properties, and therefore extend the SLAP-TGTA hybrid approach to check any LTL property.

References

[1] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis, "Memory-efficient algorithm for the verification of temporal properties," in *Proceedings of the 2nd international workshop on Computer Aided Verification (CAV'90)*, ser. LNCS, E. M. Clarke and R. P. Kurshan, Eds., vol. 531. Springer-Verlag, 1991, pp. 233–242.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. Hwang, “Symbolic model checking: 10^{20} states and beyond,” in *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*. Washington, D.C.: IEEE Computer Society Press, 1990, pp. 1–33.

[3] A. Duret-Lutz, K. Klai, D. Poitrenaud, and Y. Thierry-Mieg, “Self-loop aggregation product — a new hybrid approach to on-the-fly LTL model checking,” in *Proceedings of the 9th International Symposium on Automated Technology for Verification and Analysis (ATVA’11)*, ser. LNCS, vol. 6996. Taipei, Taiwan: Springer, Oct. 2011, pp. 336–350.

[4] A. Duret-Lutz, K. Klai, D. Poitrenaud, and Y. Thierry-Mieg, “Combining explicit and symbolic approaches for better on-the-fly LTL model checking,” arXiv, Tech. Rep. 1106.5700, Jun. 2011, extended version of our ATVA’11 paper, presenting two new techniques instead of one.

[5] R. Sebastiani, S. Tonetta, and M. Y. Vardi, “Symbolic systems, explicit properties: on hybrid approches for LTL symbolic model checking,” in *Proceedings of 17th International Conference on Computer Aided Verification (CAV’05)*, ser. LNCS, K. Etessami and S. K. Rajamani, Eds., vol. 3576. Edinburgh, Scotland, UK: Springer, Jul. 2005, pp. 350–363.

[6] J. Geldenhuys and H. Hansen, “Larger automata and less work for LTL model checking,” in *Proceedings of the 13th International SPIN Workshop (SPIN’06)*, ser. LNCS, vol. 3925. Springer, 2006, pp. 53–70.

[7] A. E. Ben Salem, A. Duret-Lutz, and F. Kordon, “Model Checking using Generalized Testing Automata,” *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC VI)*, vol. 7400, pp. 94–122, 2012.

[8] A. E. Ben Salem, A. Duret-Lutz, F. Kordon, and Y. Thierry-Mieg, “Symbolic Model Checking of stutter invariant properties Using Generalized Testing Automata,” in *20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’14)*, ser. LNCS, vol. 8413. Grenoble, France: Springer, April 2014, pp. 440–454.

[9] M. Y. Vardi, “Automata-theoretic model checking revisited,” in *Proceedings of the 8th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI’07)*, ser. LNCS, vol. 4349. Nice, France: Springer, Jan. 2007, pp. 137–150, invited paper.

[10] K. Fisler, R. Fraer, G. Kamhi, M. Y. Vardi, and Z. Yang, “Is there a best symbolic cycle-detection algorithm?” in *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’01)*, ser. LNCS, vol. 2031. Springer-Verlag, 2001, pp. 420–434.

[11] F. Somenzi, K. Ravi, and R. Bloem, “Analysis of symbolic SCC hull algorithms,” in *Proc. of FMCAD’02*, ser. LNCS, vol. 2517. Springer, 2002, pp. 88–105.

[12] A. Biere, Y. Zhu, and E. Clarke, “Multiple state and single state tableaux for combining local and global nodel checking,” in *Correct System Design*, ser. LNCS, vol. 1710. Springer Berlin Heidelberg, 1999, pp. 163–179.

[13] K. Klai and D. Poitrenaud, “MC-SOG: An LTL model checker based on symbolic observation graphs,” in *Proceedings of the 29th International Conference on Application and Theory of Petri Nets (ICATPN’08)*, ser. LNCS, vol. 5062. Xi’an, China: Springer, June 2008, pp. 288–306.

[14] K. Etessami, “Stutter-invariant languages, ω-automata, and temporal logic,” in *Proceedings of the 11th International Conference on Computer Aided Verification (CAV’99)*, ser. LNCS, N. Halbwachs and D. Peled, Eds., vol. 1633. Springer-Verlag, 1999, pp. 236–248.

[15] H. Hansen, W. Penczek, and A. Valmari, “Stuttering-insensitive automata for on-the-fly detection of livelock properties,” in *Proceedings of the 7th International ERCIM Workshop in Formal Methods for Industrial Critical Systems (FMICS’02)*, ser. Electronic Notes in Theoretical Computer Science, R. Cleaveland and H. Garavel, Eds., vol. 66(2). Málaga, Spain: Elsevier, Jul. 2002.

[16] D. Giannakopoulou and F. Lerda, “From states to transitions: Improving translation of LTL formulæ to Büchi automata,” in *Proceedings of the 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE’02)*, ser. LNCS, D. Peled and M. Vardi, Eds., vol. 2529, Houston, Texas, Nov. 2002, pp. 308–326.

[17] A. Duret-Lutz and D. Poitrenaud, “SPOT: an extensible model checking library using transition-based generalized Büchi automata,” in *Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS’04)*. IEEE Computer Society Press, Oct. 2004, pp. 76–83.

[18] R. Pelánek, “BEEM: benchmarks for explicit model checkers,” in *Proceedings of the 14th international SPIN conference on Model checking software*, ser. LNCS. Springer, 2007, pp. 263–267.

[19] S. C. C. Blom, J. C. van de Pol, and M. Weber, “LTSmin: Distributed and Symbolic Reachability,” in *Computer Aided Verification, Edinburgh*, ser. LNCS, T. Touili, B. Cook, and P. Jackson, Eds., vol. 6174. Berlin: Springer Verlag, July 2010, pp. 354–359.

[20] Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon, “Hierarchical set decision diagrams and regular models,” in *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’09)*, ser. LNCS, vol. 5505. Springer-Verlag, 2009, pp. 1–15.

[21] S. Baarir and A. Duret-Lutz, “Emptiness check of powerset Büchi automata,” in *Proceedings of the 7th International Conference on Application of Concurrency to System Design (ACSD’07)*. IEEE Computer Society, Jul. 2007, pp. 41–50.

[22] A. Valmari, “Stubborn sets for reduced state space generation,” in *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (ICATPN’91)*, ser. LNCS, vol. 618. London, UK: Springer-Verlag, 1991, pp. 491–515.

[23] P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, ser. LNCS, J. van Leeuwen, J. Hartmanis, and G. Goos, Eds. Springer-Verlag, 1996, vol. 1032.

[24] D. Peled, “Combining partial order reductions with on-the-fly model-checking,” in *Proceedings of the 6th International Conference on Computer Aided Verification (CAV’94)*, ser. LNCS, vol. 818. Springer-Verlag, 1994, pp. 377–390.