Symbolic Model Checking of Stutter-invariant Properties Using Generalized Testing Automata

A.-E. Ben Salem[1,2,3], A. Duret-Lutz[1], F. Kordon[2,3], and Y. Thierry-Mieg[2,3]

[1] LRDE, EPITA, Le Kremlin-Bicêtre, France
ala@lrde.epita.fr, adl@lrde.epita.fr
[2] Sorbonne Universités, UPMC Univ. Paris 06,
UMR 7606, LIP6, F-75005, Paris, France
Fabrice.Kordon@lip6.fr, Yann.Thierry-Mieg@lip6.fr
[3] CNRS, UMR 7606, LIP6, F-75005, Paris, France

Abstract. In a previous work, we showed that a kind of ω-automata known as *Transition-based Generalized Testing Automata* (TGTA) can outperform the Büchi automata traditionally used for *explicit* model checking when verifying stutter-invariant properties.
In this work, we investigate the use of these generalized testing automata to improve *symbolic* model checking of stutter-invariant LTL properties. We propose an efficient symbolic encoding of stuttering transitions in the product between a model and a TGTA. Saturation techniques available for decision diagrams then benefit from the presence of stuttering self-loops on all states of TGTA. Experimentation of this approach confirms that it outperforms the symbolic approach based on (transition-based) Generalized Büchi Automata.

1 Introduction

Model checking for Linear-time Temporal Logic (LTL) is usually based on converting the negation of the property to check into an ω-automaton $\mathcal{B}$, composing that automaton with a model $\mathcal{M}$ given as a Kripke structure, and finally checking the language emptiness of the resulting product $\mathcal{B} \otimes \mathcal{M}$ [21].

One way to implement this procedure is the *explicit approach* where $\mathcal{B}$ and $\mathcal{M}$ are represented as explicit graphs. $\mathcal{B}$ is usually a Büchi automaton or a generalization using multiple acceptance sets. We use Transition-based Generalized Büchi Automata (TGBA) for their conciseness. When the property to verify is stutter-invariant [8], testing automata [13] should be preferred to Büchi automata. Instead of observing the values of state propositions in the system, testing automata observe the *changes* of these values, making them suitable to represent stutter-invariant properties. In previous work [1], we showed how to generalize testing automata using several acceptance sets, and allowing a more efficient emptiness check. Our comparison showed these Transition-based Generalized Testing Automata (TGTA) to be superior to TGBA for model-checking of stutter-invariant properties.

Another implementation of this procedure is the *symbolic approach* where the automata and their products are represented by means of decision diagrams (a concise way to represent large sets or relations) [3]. Symbolic encodings for *generalized* Büchi automata are pretty common [17]. With such encodings, we can compute, in one step, the sets of all direct successors (*PostImage*) or predecessors (*PreImage*) of any set of states. Using this technique, there have been a lot of propositions for symbolic emptiness-check algorithms [9, 19, 14]. These symbolic algorithms manipulate fixpoints on the transition relation which can be optimized using saturation techniques [4, 20].

However these approaches do not offer any reduction when verifying stutter-invariant properties. So far, and to the best of our knowledge, testing automata have never been used in symbolic model checking. Our goal is therefore to propose a symbolic approach for model checking using TGTA, and compare it to the symbolic approach using TGBA. In particular, we show that the computation of fixpoints on the transition relation of the product can be sped up with a dedicated evaluation of stuttering transitions. We exploit a separation of the transition relation into two terms, one of which greatly benefits from saturation techniques.

This paper is organized as follows. Section 2 presents the symbolic model-checking approach for TGBA. For generality we define our symbolic structures using predicates over state variables in order to remain independent of the decision diagrams used to actually implement the approach. Section 3 focuses on the encoding of TGTA in the same framework. We first show how a TGTA can be encoded, then we show how to improve the encoding of the Kripke structure and the product to benefit from saturation in the encoding of stuttering transitions in the TGTA. Finally, Section 4 compares the two approaches experimentally with an implementation that uses hierarchical Set Decision Diagrams (SDD) [20] (a particular type of Decision Diagrams on integer variables, on which we can apply user-defined operations). On our large, BEEM-based benchmark, our symbolic encoding of TGTA appears to to be superior to TGBA.

2 Symbolic LTL Model Checking using TGBA

We first recall how to perform the automata-theoretic approach to LTL model checking using symbolic encodings of TGBA and Kripke structures. This setup will serve as a baseline to measure our improvements from later sections.

Through the paper, let AP designate the finite set of *atomic propositions* of the model. Any state of the model is labeled by a valuation of these atomic propositions. Let $\Sigma = 2^{AP}$ denote the set of these valuations, which we interpret either as sets or as Boolean conjunctions. For instance if $AP = \{a,b\}$, then $\Sigma = 2^{AP} = \{\{a,b\},\{a\},\{b\},\emptyset\}$ or equivalently $\Sigma = \{ab, a\bar{b}, \bar{a}b, \bar{a}\bar{b}\}$. An *execution* of the model is an infinite sequence of such valuations, i.e., an element of Σ^ω.

2.1 Kripke Structures and their Symbolic Encoding

The executions of the model can be represented by a Kripke structure $\mathcal{M}$.

Definition 1 (Kripke Structure) *A* Kripke structure *over Σ is a tuple $\mathcal{M} = \langle S, S_0, R, L\rangle$, where:*

- *S is a finite set of states,*

– *$S_0 \subseteq S$ is the set of initial states,*
– *$R \subseteq S \times S$ is the transition relation,*
– *$L : S \to \Sigma$ is a state-labeling function.*

An execution $w = \ell_0\ell_1\ell_2 \ldots \in \Sigma^\omega$ is accepted *by $\mathcal{M}$ if there exists an infinite sequence $s_0, s_1, \ldots \in S^\omega$ such that $s_0 \in S_0$ and $\forall i \in \mathbb{N}$, $(L(s_i) = \ell_i) \wedge ((s_i, s_{i+1}) \in R)$. The* language *accepted by $\mathcal{M}$ is the set $\mathscr{L}(\mathcal{M}) \subseteq \Sigma^\omega$ of executions it accepts.*

In symbolic model checking we encode such a structure with predicates that represent sets of states or transitions [18]. These predicates are then implemented using decision diagrams [3].

Definition 2 (Symbolic Kripke Structure) *A Kripke structure $\mathcal{M} = \langle S, S_0, R, L\rangle$ can be encoded by the following predicates where $s, s' \in S$ and $\ell \in \Sigma$:*
– *$P_{S_0}(s)$ is true iff $s \in S_0$,*
– *$P_R(s, s')$ is true iff $(s, s') \in R$,*
– *$P_L(s, \ell)$ is true iff $L(s) = \ell$.*

In the sequel, we use the notations $S_0(s)$, $R(s, s')$ and $L(s, \ell)$ instead of $P_{S_0}(s)$, $P_R(s, s')$ and $P_L(s, \ell)$. A Symbolic Kripke structure *is therefore a triplet of predicates $K = \langle S_0, R, L\rangle$ on state variables.*

Variables s and s' used above are typically implemented using decision diagrams to represent either a state or a set of states. In a typical encoding [3], states are represented by conjunctions of Boolean variables. For instance if $S = \{0,1\}^3$, a state $s = (1,0,1)$ would be encoded as $s_1\bar{s}_2s_3$. Similarly, s_1s_3 would encode the set of states $\{(1,0,1),(1,1,1)\}$. With this encoding, S_0, R and L are propositional formulae which can be implemented with BDDs or other kind of decision diagrams. In our implementation, we used SDDs on integer variables [20].

2.2 TGBA and their Symbolic Encoding

Transition-based Generalized Büchi Automata (TGBA) [11] are a generalization of the Büchi Automata (BA) commonly used for model checking. In our context, the TGBA represents the negation of the LTL property to verify. We chose to use TGBA rather than BA since they allow a more compact representation of properties [7].

Definition 3 (TGBA) *A* TGBA *over the alphabet $\Sigma = 2^{AP}$ is a tuple $\mathcal{B} = \langle Q, Q_0, \delta, F\rangle$ where:*
– *Q is a finite set of states,*
– *$Q_0 \subseteq Q$ is a set of initial states,*
– *$\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, where each element (q, ℓ, q') represents a transition from state q to state q' labeled by the valuation ℓ,*
– *$F \subseteq 2^\delta$ is a set of acceptance sets of transitions.*

$\mathcal{B}$ accepts *an execution $\ell_0\ell_1 \ldots \in \Sigma^\omega$ if there exists an infinite path $(q_0, \ell_0, q_1)(q_1, \ell_1, q_2) \ldots \in \delta^\omega$ that visits each acceptance set infinitely often: $q_0 \in Q_0$ and $\forall f \in F, \forall i \in \mathbb{N}, \exists j \geq i, (q_j, \ell_j, q_{j+1}) \in f$.*
The language *accepted by $\mathcal{B}$ is the set $\mathscr{L}(\mathcal{B}) \subseteq \Sigma^\omega$ of the executions it accepts.*

We target TGBA in this paper because their use of generalized and transition-based acceptance make them more concise than traditional Büchi automata [11]. Generalized acceptance is classically used in symbolic model checking [9] and using transition-based acceptance is not a problem [17]. People working with (classical) Büchi automata can adjust to our definitions by "pushing" the acceptance of states to their outgoing transitions [7].

Any LTL formula φ can be converted into a TGBA whose language is the set of executions that satisfy φ [7]. Figure 1(a) shows a TGBA derived from the LTL formula $\mathsf{FG}a$. The Boolean expression over $AP = \{a\}$ that labels each transition represents the valuation of atomic propositions that hold in this transition (in this example, $\Sigma = \{a, \bar{a}\}$). Any infinite path in this example is accepted if it visits infinitely often the only acceptance set containing transition $(1, a, 1)$.

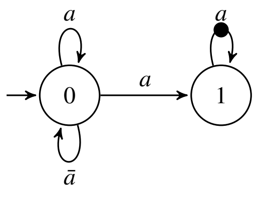
Like Kripke structures, TGBAs can be encoded by predicates [18] on state variables.

Definition 4 (Symbolic TGBA) *A TGBA $\langle Q, Q_0, \delta, F\rangle$ is symbolically encoded by a triplet of predicates $\langle Q_0, \Delta, \{\Delta_f\}_{f\in F}\rangle$ where:*
- *$Q_0(q)$ is true iff $q \in Q_0$,*
- *$\Delta(q, \ell, q')$ is true iff $(q, \ell, q') \in \delta$,*
- *$\forall f \in F$, $\Delta_f(q, \ell, q')$ is true iff $(q, \ell, q') \in f$.*

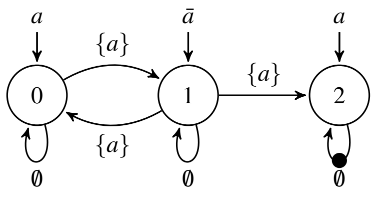
2.3 Symbolic Product of a TGBA with a Kripke structure

We now show how to build a synchronous product by composing the symbolic representations of a TGBA with that of a Kripke structure, inspired from Sebastian *et al.* [18].

Definition 5 (Symbolic Product for TGBA) *Given a Symbolic Kripke structure $K = \langle S_0, R, L\rangle$ and a Symbolic TGBA $A = \langle Q_0, \Delta, \{\Delta_f\}_{f\in F}\rangle$ sharing a set AP of atomic propositions, the Symbolic Product $K \otimes A = \langle P_0, T, \{T_f\}_{f\in F}\rangle$ is defined by the predicates P_0, T and T_f encoding respectively the set of initial states, the transition relation and the acceptance transitions of the product:*
- *(s, q) denotes the state variables of the product (s for the Kripke structure and q for TGBA),*
- *$P_0(s, q) = S_0(s) \wedge Q_0(q)$,*
- *$T((s, q), (s', q')) = \exists \ell \left[R(s, s') \wedge L(s, \ell) \wedge \Delta(q, \ell, q')\right]$, where (s', q') encodes the next state variables,*



(a) TGBA



(b) TGTA

Fig. 1. TGBA and TGTA for the LTL property $\varphi = \mathsf{FG}a$. Acceptance transitions are indicated by ●.

– $\forall f \in F, T_f((s,q),(s',q')) = \exists \ell \left[R(s,s') \wedge L(s,\ell) \wedge \Delta_f(q,\ell,q')\right]$.

The labels ℓ are used to ensure that a transition (q,ℓ,q') of A is synchronized with a state s of K such that $L(s,\ell)$. This way, we ensure that the product recognizes only the executions of K that are also recognized by A. However we do not need to remember how product transitions are labeled to check $K \otimes A$ for emptiness. A product can be seen as a TGBA without labels on transitions.

In symbolic model checkers, the exploration of the product is based on the following *PostImage* operation [18]. For any set of states encoded by a predicate P, *PostImage*(P) $(s',q') = \exists(s,q) \left[P(s,q) \wedge T((s,q),(s',q'))\right]$ returns a predicate representing the set of states reachable in one step a state in P.

Because in TGBA the acceptance conditions are based on transitions, we also define *PostImage*(P, f) to computes the successors of P reached using only transitions from an acceptance set $f \in F$: *PostImage*$(P, f)(s',q') = \exists(s,q) \left[P(s,q) \wedge T_f((s,q),(s',q'))\right]$.

These two operations are at the heart of the symbolic emptiness check presented in the next section.

2.4 Symbolic Emptiness Check

One way to check if a product is not empty is to find a reachable Strongly Connected Component that contains transitions from all acceptance sets (we call it an *accepting SCC*). Figure 2 shows such an algorithm implemented using symbolic operations. It mimics the algorithm FEASIBLE of Kesten et al. [14] and can be seen as a forward variant of OWCTY (One Way Catch Them Young [9]) that uses *PostImage* computations instead of *PreImage*. Line 3 computes the set P of all reachable states of the product. The main loop on lines 4–8 refines P at each iteration. Lines 5–6 keep only the states of P that can be reached from a cycle in P. Lines 7–8 then remove all cycles that never visit some acceptance set $f \in F$. Eventually the main loop will reach a fixpoint where P contains all states that are reachable from an accepting SCC. The product is empty iff that set is empty.

There are many variants of such symbolic emptiness checks. We selected this variant mainly for its simplicity, as our contributions are mostly independent of the chosen algorithm: essentially, we will improve the cost of computing *Reach*(P) (used lines 3 and 8).

```
1 Input: PostImage, $P_0$ and $F$
2 begin
3     $P \leftarrow$ Reach($P_0$)
4     while $P$ changes do
5         while $P$ changes do
6             $P \leftarrow$ PostImage($P$)
7         for $f$ in $F$ do
8             $P \leftarrow$ Reach(PostImage($P$, $f$))
9     return $P = \emptyset$
```

```
1 Reach($P$)
2     while $P$ changes do
3         $P \leftarrow P \cup$ PostImage($P$)
4     return $P$
```

Fig. 2. Forward-variant of OWCTY, a symbolic emptiness check.

3 Symbolic Approach Using TGTA

Testing automata [10] are a kind of automata that recognize only stutter-invariant properties. In previous work [1] we generalized them as Transition-based Generalized Testing Automata (TGTA). In this section, we show how to encode a TGTA for symbolic model checking.

Definition 6 *A property, i.e., a set of infinite sequences $\mathcal{P} \subseteq \Sigma^\omega$, is* stutter-invariant *iff any sequence $\ell_0\ell_1\ell_2\ldots \in \mathcal{P}$ remains in $\mathcal{P}$ after repeating any valuation ℓ_i or omitting duplicate valuations. Formally, $\mathcal{P}$ is stutter-invariant iff $\ell_0\ell_1\ell_2\ldots \in \mathcal{P} \iff \ell_0^{i_0}\ell_1^{i_1}\ell_2^{i_2}\ldots \in \mathcal{P}$ for any $i_0 > 0, i_1 > 0\ldots$*

Theorem 1 *An LTL property is stutter-invariant iff it can be expressed as an LTL formula that does not use the X operator [16].*

3.1 Transition-based Generalized Testing Automata

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 valuations of an execution, we say that a TGTA executes a *stuttering transition*.

If A and B are two valuations, $A \oplus B$ denotes the symmetric set difference, i.e., the set of atomic propositions that differ (e.g., $a\bar{b} \oplus ab = \{a\} \oplus \{a,b\} = \{b\}$). Technically, this can be implemented with an XOR operation on bitsets (hence the symbol $\oplus$).

Definition 7 *A TGTA over the alphabet Σ is a tuple $\mathcal{T} = \langle Q, Q_0, U, \delta, F\rangle$ where:*
- *Q is a finite set of states,*
- *$Q_0 \subseteq Q$ is a set of initial states,*
- *$U : Q_0 \to 2^\Sigma$ is a function mapping each initial state to a set of symbols of Σ,*
- *$\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, where each element (q,c,q') represents a transition from state q to state q' labeled by a* changeset *c interpreted as a (possibly empty) set of atomic propositions whose value must change between q and q',*
- *$F \subseteq 2^\delta$ is a set of acceptance sets of transitions,*

and such that all stuttering transitions (i.e., transitions labeled by $\emptyset$) are self-loops and every state has a stuttering self-loop. More formally, we can define a partition of $\delta = \delta_\emptyset \cup \delta_*$ where:
- *$\delta_\emptyset = \{(q,\emptyset,q) \mid q \in Q\}$ is the stuttering transition relation,*
- *$\delta_* = \{(q,\ell,q') \in \delta \mid \ell \neq \emptyset\}$ is the non-stuttering transition relation.*

An execution $\ell_0\ell_1\ell_2\ldots \in \Sigma^\omega$ is accepted by $\mathcal{T}$ if there exists an infinite path $(q_0, \ell_0 \oplus \ell_1, q_1)(q_1, \ell_1 \oplus \ell_2, q_2)(q_2, \ell_2 \oplus \ell_3, q_3)\ldots \in \delta^\omega$ where:
- *$q_0 \in Q_0$ with $\ell_0 \in U(q_0)$ (the execution is recognized by the path),*
- *$\forall f \in F, \forall i \in \mathbb{N}, \exists j \geq i, (q_j, \ell_j \oplus \ell_{j+1}, q_{j+1}) \in f$ (each acceptance set is visited infinitely often).*

The language accepted by $\mathcal{T}$ is the set $\mathscr{L}(\mathcal{T}) \subseteq \Sigma^\omega$ of executions it accepts.

Figure 1(b) shows a TGTA recognizing the LTL formula $\mathsf{F}\mathsf{G}a$. Acceptance sets are represented using dots as in TGBAs. Transitions are labeled by changesets: e.g., the transition $(0,\{a\},1)$ means that the value of a changes between states 0 and 1. Initial valuations are shown above initial arrows: $U(0)=\{a\}$, $U(1)=\{\bar{a}\}$ and $U(2)=\{a\}$. As an illustration, the execution $\bar{a};a;a;a;\ldots$ is accepted by the run $\textcircled{1}\xrightarrow{\{a\}}\textcircled{2}\xrightarrow{\emptyset}\textcircled{2}\xrightarrow{\emptyset}\textcircled{2}\cdots$ because the value a only changes between the first two steps.

Theorem 2 *Any stuttering invariant property can be translated into an equivalent TGTA [1].*

Note that Def. 7 differs from our previous work [1] because we now enforce a partition of δ such that stuttering transitions can only be self-loops. However, the TGTA resulting from the LTL translation we presented previously [1] already have this property. We will use it to optimize symbolic computation in section 3.3.

Finally, a TGTA's symbolic encoding is similar to that of a TGBA.

Definition 8 (Symbolic TGTA) *A TGTA $\mathcal{T}=\langle Q,Q_0,U,\delta,F\rangle$ is symbolically encoded by a triplet of predicates $\langle U_0,\Delta^{\oplus},\{\Delta^{\oplus}_f\}_{f\in F}\rangle$ where:*
- *$U_0(q,\ell)$ is true iff $(q\in Q_0)\wedge(U(q)=\ell)$*
- *$\Delta^{\oplus}(q,c,q')$ is true iff $(q,c,q')\in\delta$*
- *$\forall f\in F$, $\Delta^{\oplus}_f(q,c,q')$ is true iff $((q,c,q')\in f)$*

3.2 Symbolic Product of a TGTA with a Kripke structure

The product between a TGTA and a Kripke structure is similar to the TGBA case, except that we have to deal with changesets. The transitions (s,s') of a Kripke structure that must be synchronized with a transition (q,c,q') of a TGTA, are all the transitions such that the label of s and s' differs by the changeset c.

In order to reduce the number of symbolic operations when computing the Symbolic product of a TGTA with a Kripke structure, we introduce a changeset-based encoding of Kripke structure (only the transition relation changes).

Definition 9 (Changeset-based symbolic Kripke structure) *A Kripke structure $\mathcal{M}=\langle S,S_0,R,L\rangle$, can be encoded by the changeset-based symbolic Kripke structure $K^{\oplus}=\langle S_0,R^{\oplus},L\rangle$, where:*
- *the predicate $R^{\oplus}(s,c,s')$ is true iff $((s,s')\in R\wedge(L(s)\oplus L(s'))=c)$,*
- *the predicates S_0 and L have the same definition as for a Symbolic Kripke structure K (Def. 2).*

In practice, the (changeset-based or not) symbolic transition relation of the Kripke structure should be constructed directly from the model and atomic propositions of the formula to check. In Section 4.2, we discuss how we build such changeset-based Kripke structures in our setup.

The procedure requires reconstruction of the symbolic transition relation for each formula (or at least for each set of atomic propositions used in the formulas). However the cost of this construction is not significant with respect to the complexity of the overall model checking procedure (overall on our benchmark, less than 0.16% percent of total time was spent building these transition relations).

Adjusting the symbolic encoding of the Kripke structure to TGTA, allows us to obtain the following natural definition of the symbolic product using TGTA:

Definition 10 (Symbolic Product for TGTA) *Given a changeset-based Symbolic Kripke structure $K^{\oplus} = \langle S_0, R, L\rangle$ and a Symbolic TGTA $A^{\oplus} = \langle U_0, \Delta^{\oplus}, \{\Delta_f^{\oplus}\}_{f\in F}\rangle$ sharing the same set of atomic propositions AP, the Symbolic Product $K^{\oplus} \otimes A^{\oplus} = \langle P_0, T, \{T_f\}_{f\in F}\rangle$ is defined by the following predicates:*

- *The set of initial states is encoded by:* $P_0(s,q) = \exists \ell \left[S_0(s) \wedge L(s,\ell) \wedge U_0(q,\ell)\right]$
- *The transition relation of the product is:* $T((s,q),(s',q')) = \exists c \left[R^{\oplus}(s,c,s') \wedge \Delta^{\oplus}\,(q,c,q')\right]$
- *The definition of T_f is similar to T by replacing $\Delta^{\oplus}$ with $\Delta_f^{\oplus}$.*

The definitions of $\mathit{PostImage}(P)$ and $\mathit{PostImage}(P,f)$ are the same as in the TGBA approach, with the new expressions of T and T_f above.

As for the product in TGBA approach, the product in TGTA approach is a TGBA (or a TGTA) without labels on transitions, and the same emptiness check algorithm (Fig. 2) can be used for the two products.

3.3 Exploiting Stuttering Transitions to Improve Saturation in the TGTA Approach

Among symbolic approaches for evaluating a fixpoint on a transition relation, the *saturation* algorithm offers gains of one to three orders of magnitude [4] in both time and memory, especially when applied to asynchronous systems [5].

The saturation algorithm does not use a breadth-first exploration of the product (i.e., each iteration in the function `Reach` (Fig. 2) is not a “global" $\mathit{PostImage}()$ computation). Saturation instead recursively repeats “local” fixed-points by recognizing and exploiting transitions locality and identity transformations on state variables [5].

This algorithm considers that the system state consists of k discrete variables encoded by a Decision Diagram, and that the transition relation is expressed as a disjunction of terms called transition clusters. Each cluster typically only reads or writes a limited subset consisting of $k' \leq k$ variables, called the *support* of the cluster. During the least fixpoint computing the reachable states, saturation technique consists in reordering [12] the evaluation of (“local” fixed-points on) clusters in order to avoid the construction of (useless) intermediate Decision Diagram nodes.

The algorithm to determine an ordering for saturation is based on the support of each cluster.

We now show how to decompose the transition relation of the product $K^{\oplus} \otimes A^{\oplus}$ to exhibit clusters having a smaller support, favoring the saturation technique.

We base our decomposition on the fact that in a TGTA, all stuttering transitions are self-loops and every state has a stuttering self-loop ($\delta_{\emptyset}$ in Def. 7). Therefore, stuttering transitions in the Kripke structure can be mapped to stuttering transitions in the product regardless of the TGTA state.

Let us separate stuttering and non-stuttering transitions in the transition relation T of the product between a Kripke structure and a TGTA ($K^{\oplus} \otimes A^{\oplus}$):

$$T((s,q),(s',q')) = \left(R^{\oplus}(s,\emptyset,s') \wedge \Delta^{\oplus}(q,\emptyset,q')\right) \vee \left(\exists c\left[R_*^{\oplus}(s,c,s') \wedge \Delta_*^{\oplus}(q,c,q')\right]\right)$$

where $R_*^{\oplus}$ and $\Delta_*^{\oplus}$ encode respectively the non-stuttering transitions of the model and of the TGTA:

– $\Delta^{\oplus}_{*}(q,c,q')$ is true iff $(q,c,q') \in \delta_*$ (see Def. 7)
– $R^{\oplus}_{*}(s,c,s')$ is true iff $R^{\oplus}(s,c,s') \wedge (c \neq \emptyset)$

According to the definition of $\delta_\emptyset$ in Def. 7, the predicate $\Delta^{\oplus}(q,\emptyset,q')$ encodes the set of TGTA's self-loops and can be replaced by the predicate $equal(q,q')$, simplifying T:

$$T((s,q),(s',q')) = \underbrace{\left(R^{\oplus}(s,\emptyset,s') \wedge equal(q,q')\right)}_{T_\emptyset((s,q),(s',q'))} \vee \underbrace{\left(\exists c\left[R^{\oplus}_{*}(s,c,s') \wedge \Delta^{\oplus}_{*}(q,c,q')\right]\right)}_{T_*((s,q),(s',q'))} \quad (1)$$

The transition relation (1) is a disjunction of T_*, synchronizing updates of both TGTA and Kripke structure, and $T_\emptyset$, corresponding to the stuttering transitions of the Kripke structure. Since all states in the TGTA have a stuttering self-loop, $T_\emptyset$ does not depend on the TGTA state. In practice, the predicate $equal(q,q')$ is an identity relation for variable q [5] and is simplified away (i.e., the term $T_\emptyset$ can be applied to a decision diagram without consulting or updating the variable q [12]). Hence q is not part of the clusters supports in $T_\emptyset$ (while q is part of the clusters supports in T_*). This gives more freedom to the saturation technique for reordering the application of clusters in $T_\emptyset$.

Note that in the product of TGBA with Kripke structure (Def. 5) there is no $T_\emptyset$ that could be extracted since there is no stuttering hypothesis in general. This severely limits the possibilities of the saturation algorithm in the TGBA approach.

In the symbolic emptiness check presented in Fig. 2, the function `Reach` corresponds to a least fixpoint performed using saturation. As we shall see experimentally in the next section, the better encoding of $T_\emptyset$ (without q in its support) in the product of TGTA with Kripke structure, greatly favors the saturation technique, leading to gains of roughly one order of magnitude.

4 Experimentation

We now compare the approaches presented in this paper. The symbolic model-checking approach using TGBA, presented in Section 2 serves as our baseline. We first describe our implementation and selected benchmarks, prior to discussing the results.

4.1 Implementation

All approaches are implemented on top of three libraries[4]: Spot, SDD/ITS, and LTSmin.

Spot is a model-checking library providing several bricks that can be combined to build model checkers [7]. In our implementation, we reused the modules providing a translation form an LTL formula into a TGBA and into a TGTA [1].

SDD/ITS is a library for symbolic representation of state spaces in the form of Instantiable Transition Systems (ITS): an abstract interface for symbolic Labeled Transition Systems (LTS). The symbolic encoding of ITS is based on Hierarchical Set Decision Diagrams (SDD) [20]. SDDs allow a compact symbolic representation of states and transition relation.

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

The algorithms presented in this paper can be implemented using any kind of decision diagram (such as OBDD), but use of the SDD software library allows to easily benefit from the automatic saturation mechanism described in [12].

LTSmin [2] can generate state spaces from various input formalisms (μCRL, DVE, GNA, MAPLE, PROMELA, ...) and store the obtained LTS in a concise symbolic format, called Extended Table Format (ETF). We used LTSmin to convert DVE models into ETF for our experiments. This approach offers good generality for our tool, since it can process any formalism supported by LTSmin tool.

Our symbolic model checker inputs an ETF file and an LTL formula. The LTL formula is converted into TGBA or TGTA which is then encoded using an ITS. The ETF model is also symbolically encoded using an ITS (see Sec. 4.2). The two obtained ITSs are then composed to build a symbolic product, which is also an ITS. Finally, the OWCTY emptiness check is applied to this product.

4.2 Using ETF to build a changeset-based symbolic Kripke structure

An ETF file[5] produced by LTSmin is a text-based serialization of the symbolic representation of the transition relation of a model whose states consist in k integer variables. Transitions are described in the following tabular form:

```
0/1  0/1  *    *
1/2  *    0/1  *
...
```

where each column correspond to a variable, and each line describes the effect of a symbolic transition on the corresponding variables. The notation "in/out" means that the variable must have the value "*in*" for the transition to fire, and the value is then updated to "*out*". A "*" means that the variable is not consulted or updated by the transition. Each line may consequently encode a set of explicit transitions that differ only by the values of the starred variables: the support of a transition is the set of unstarred variables.

A changeset-based symbolic Kripke structure, as defined in Sec. 3.2, can be easily obtained from such a description. To obtain a changeset associated to a line in the file, it is enough to compute difference between values of atomic propositions associated to the *in* variables and the values associated to the *out* variables. Because they do not change, starred variables have no influence on the changeset.

Note that an empty changeset does not necessarily correspond to a line where all variables are starred. Even when *in* and *out* values are different, they may have no influence on the atomic propositions, and the resulting changeset may be empty. For instance if the only atomic proposition considered is $p = (v_1 > 1)$ (where v_1 denotes the first-column variable), then the changeset associated to the first line is $\emptyset$, and the changeset for the second line is $\{p\}$.

4.3 Benchmark

We evaluated the TGBA and TGTA approaches on the following models and formulae:

[5] `http://fmt.cs.utwente.nl/tools/ltsmin/doc/etf.html`

Table 1. Characteristics of our selected benchmark models. The stuttering-ratio represents the percentage of stuttering transitions in the model. Since the definition of stuttering depends on the atomic propositions of the formula, we give an average over the 200 properties checked against each model.

BEEM model	states $10^3 \times$	stut. ratio	BEEM model	states $10^3 \times$	stut. ratio
`at.5`	31 999	95%	`lann.6`	144 151	52%
`bakery.4`	157	83%	`lann.7`	160 025	64%
`bopdp.3`	1 040	91%	`lifts.7`	5 126	93%
`elevator.4`	888	74%	`peterson.5`	131 064	83%
`brp2.3`	40	79%	`pgm_protocol.8`	3 069	92%
`fischer.5`	101 028	89%	`phils.8`	43 046	89%
`lamport_nonatomic.5`	95 118	92%	`production_cell.6`	14 520	85%
`lamport.7`	38 717	93%	`reader_writer.3`	604	88%

– Our models come from the BEEM benchmark [15], a suite of models for explicit model checking, which contains some models that are considered difficult for symbolic model checkers [2]. Table 1 summarizes the 16 models we selected as representatives of the overall benchmark.
– BEEM provides a few LTL formulae, but they mostly represent safety properties and can thus be checked without building a product. Therefore, for each model, we randomly generated 200 stutter-invariant LTL formulae: 100 verified formulae (empty product) and 100 violated formulae (non-empty product). We consequently have a total 3200 pairs of (model, formula).

All tests were run on a 64bit Linux system running on an Intel Xeon E5645 at 2.40GHz. Executions that exceeded 30 minutes or 4GB of RAM were aborted and are reported with time and memory above these thresholds in our graphics.

In all approaches evaluated, symbolic products are encoded using the same variable ordering: we used the symbolic encoding named “log-encode with top-order” by Sebastiani et al. [18].

4.4 Results

The results of our experimental[6] comparisons are presented by the two scatter plot matrices of Fig. 3 and Fig. 4. The scatter plot highlighted at the bottom of Fig. 3 compares the time-performance of the TGTA-approach against the reference TGBA approach.[7] Each point of the scatter plot represents a measurement for a pair (*model*,*formula*). For the highlighted plot, the x-axis represents the TGBA approach and the y-axis represent the TGTA approach, so 3060 points below the diagonal correspond to cases where the TGTA approach is better, and the 131 points above the diagonal corresponds to points were the TGBA approach is better (In scatter plot matrices, each point below the diagonal is in favor of the approach displayed on the right, while each point above the

[6] The results, models, formulae and tools used in these tests, can be downloaded from `http://www.lrde.epita.fr/~ala/TACAS-2014/Benchmark.html`

[7] We recommend viewing these plots online.

diagonal is in favor of the approach displayed in the top). Axes use a logarithmic scale. The colors distinguish violated formulae (non-empty product) from verified formulae (empty products). In order to show the influence of the saturation technique, we also ran the TGBA and TGTA approaches with saturation disabled. In our comparison matrix, the labels “(sat)” and “(nosat)” indicate whether saturation was enabled or not. Fig. 4 gives the memory view of this experiment.

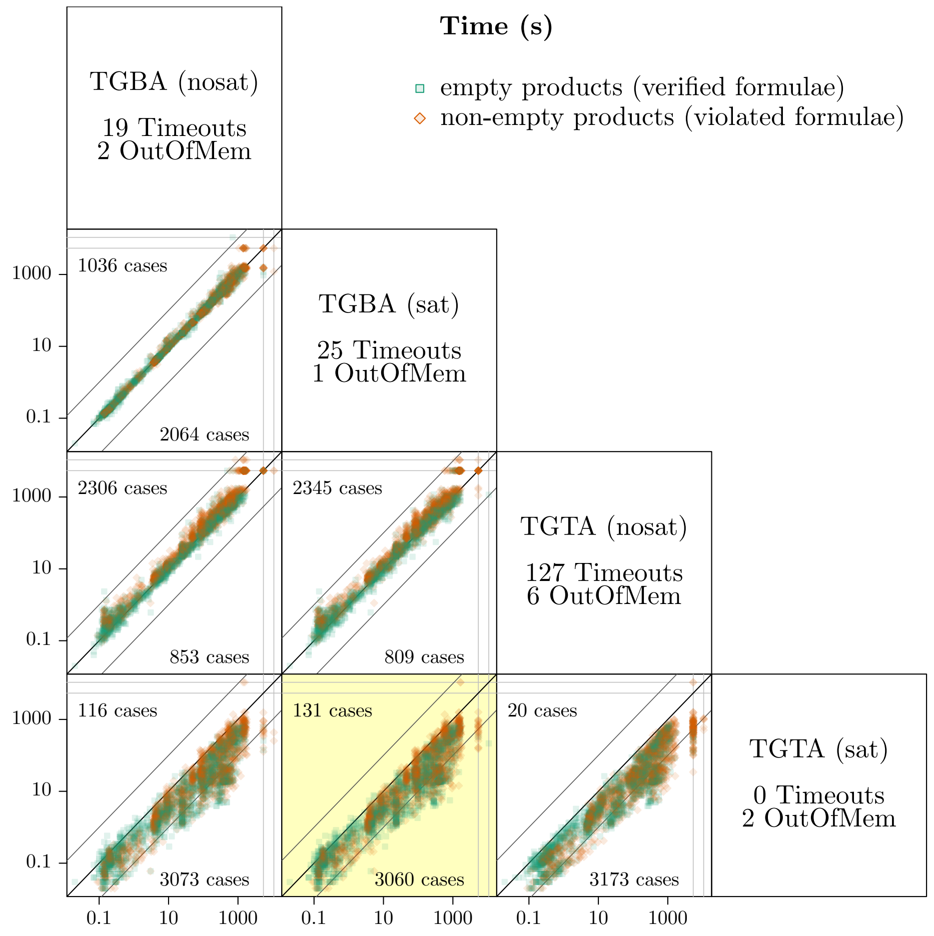


Fig. 3. Time-comparison of the TGBA and TGTA approaches, with saturation enabled “(sat)” or disabled “(nosat)”, on a set of 3199 pairs (*model*,*formula*). Timeouts and Out-of-memory errors are plotted on separate lines on the top or right edges of the scatter plots. Each plot also displays the number of cases that are above or below the main diagonal (including timeouts and out-of-memory errors), i.e., the number of (*model*,*formula*) for which one approach was better than the other. Additional diagonals show the location of $\times 10$ and $/10$ ratios. Points are plotted with transparency to better highlight dense areas, and lessen the importance of outliers.

As shown by the highlighted scatter plots in Fig. 3 and 4, the TGTA approach clearly outperforms the traditional TGBA-based scenario by one order of magnitude. This is due to the combination of two factors: saturation and exploration of stuttering.

The saturation technique does not significantly improve the model checking using TGBA (compare "TGBA (sat)" against "TGBA (nosat)" at the top of Fig. 3 and 4). In fact, the saturation technique is limited on the TGBA approach, because in the transition relation of Def. 5 each conjunction must consult the variable q representing the state of the TGBA, therefore q impacts the supports and the reordering of clusters evaluated by the saturation. This situation is different in the case of TGTA approach, where the $T_\emptyset$ term of the transition-relation of the product (equation (1)) does not involve the state q of the TGTA: here, saturation strongly improve performances (compare "TGTA (sat)" against "TGTA (nosat)").

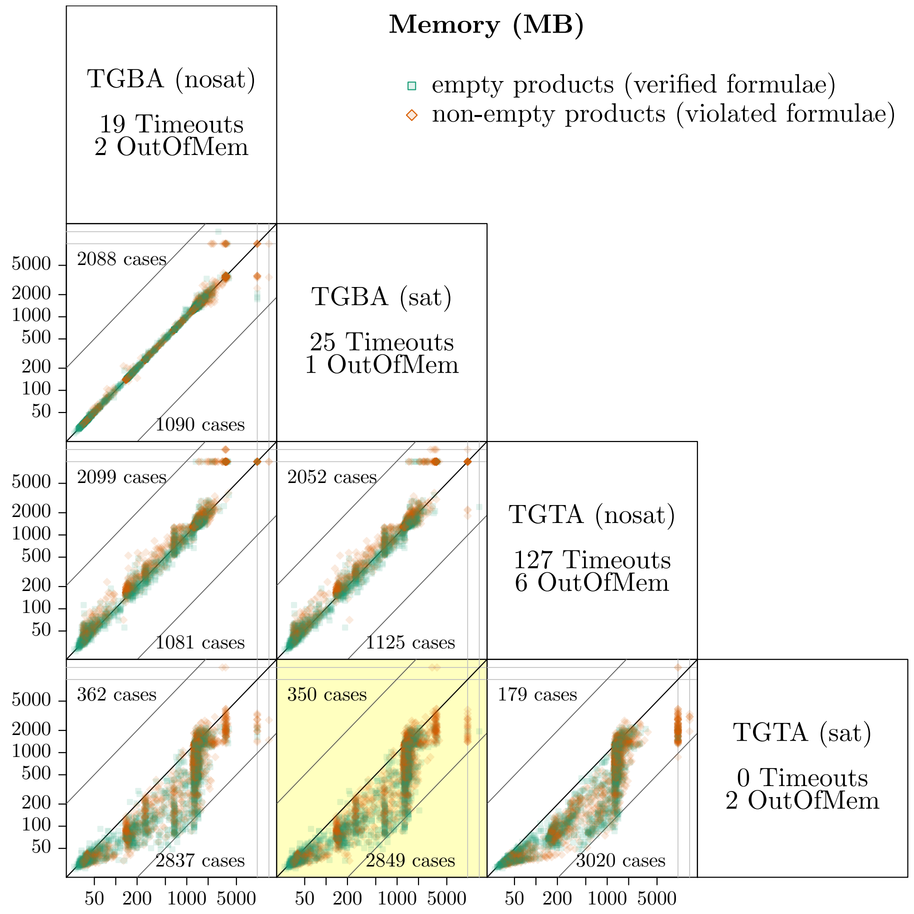


Fig. 4. Comparison of the memory-consumption of the TGBA and TGTA approaches, with or without saturation, on the same set of problems.

Overall the improvement to this symbolic technique was only made possible because the TGTA representation makes it easy to process the stuttering behaviors separately from the rest. These stuttering transitions represent a large part of the models transitions, as shown by the stuttering-ratios of Table 1. Using these stuttering-ratios, we can estimate in our Benchmark the importance of the term $T_\emptyset$ compared to T_* in equation (1).

5 Conclusion

Testing automata [10] are a way to improve the explicit model checking approach when verifying stutter-invariant properties, but they had not been used for symbolic model checking. In this paper, we gave the first symbolic approach using testing automata, with generalized acceptance (TGTA), and compare it to a more classical symbolic approach (using TGBA).

On our benchmark, using TGTA, we were able to gain one order of magnitude over the TGBA-based approach.

We have shown that fixpoints over the transition relation of a product between a Kripke structure and a TGTA can benefit from the saturation technique, especially because part of their expression is only dependent on the model, and can be evaluated without consulting the transition relation of the property automaton. The improvement was possible only because TGTA makes it possible to process stuttering behaviors specifically, in a way that helps the saturation technique.

In future work, we plan to evaluate the use of TGTA in the context of hybrid approaches, mixing both explicit and symbolic approaches [18, 6].

References

1. 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)*, 7400:94–112, 2012.
2. S. C. C. Blom, J. C. van de Pol, and M. Weber. LTSmin: Distributed and symbolic reachability. In *Computer Aided Verification, Edinburgh*, vol. 6174 of *LNCS*, pp. 354–359. Springer, July 2010.
3. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pp. 1–33. IEEE Computer Society Press, 1990.
4. G. Ciardo, R. M. Marmorstein, and R. Siminiceanu. Saturation unbound. In *Proc. of TACAS'03*, vol. 2619 of *LNCS*, pp. 379–393. Springer.
5. G. Ciardo and A. J. Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *Proc. of the 13 IFIP WG 10.5 international conference on Correct Hardware Design and Verification Methods*, vol. 3725 of *LNCS*, pp. 146–161. Springer, 2005.
6. 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 *Proc. of ATVA'11*, vol. 6996 of *LNCS*, pp. 336–350. Springer.

7. A. Duret-Lutz and D. Poitrenaud. SPOT: an extensible model checking library using transition-based generalized Büchi automata. In *Proc. of MASCOTS'04*, pp. 76–83. IEEE Computer Society Press.
8. K. Etessami. Stutter-invariant languages, ω-automata, and temporal logic. In *Proc. of CAV'99*, vol. 1633 of *LNCS*, pp. 236–248. Springer.
9. K. Fisler, R. Fraer, G. Kamhi, M. Y. Vardi, and Z. Yang. Is there a best symbolic cycle-detection algorithm? In *Proc. of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pp. 420–434. Springer, 2001.
10. J. Geldenhuys and H. Hansen. Larger automata and less work for LTL model checking. In *Proc. of SPIN'06*, vol. 3925 of *LNCS*, pp. 53–70. Springer.
11. D. Giannakopoulou and F. Lerda. From states to transitions: Improving translation of LTL formulæ to Büchi automata. In *Proc. of FORTE'02*, vol. 2529 of *LNCS*, pp. 308–326.
12. A. Hamez, Y. Thierry-Mieg, and F. Kordon. Hierarchical set decision diagrams and automatic saturation. In *Proc. of the 29th international conference on Applications and Theory of Petri Nets*, PETRI NETS '08, pp. 211–230. Springer, 2008.
13. H. Hansen, W. Penczek, and A. Valmari. Stuttering-insensitive automata for on-the-fly detection of livelock properties. In *Proc. of FMICS'02*, vol. 66(2) of *ENTCS*. Elsevier.
14. Y. Kesten, A. Pnueli, and L. on Raviv. Algorithmic verification of linear temporal logic specifications. In *Proc. of ICALP'98*, vol. 1443 of *LNCS*, pp. 1–16. Springer.
15. R. Pelánek. BEEM: benchmarks for explicit model checkers. In *Proc. of the 14th international SPIN conference on Model checking software*, Lecture Notes in Computer Science, pp. 263–267. Springer, 2007.
16. D. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63(5):243–246, Sept. 1995.
17. K. Y. Rozier and M. Y. Vardi. A multi-encoding approach for LTL symbolic satisfiability checking. In *Proc. of FM'11*, pp. 417–431. Springer.
18. R. Sebastiani, S. Tonetta, and M. Y. Vardi. Symbolic systems, explicit properties: on hybrid approches for LTL symbolic model checking. In *Proc. of CAV'05*, vol. 3576 of *LNCS*, pp. 350–363. Springer.
19. F. Somenzi, K. Ravi, and R. Bloem. Analysis of symbolic SCC hull algorithms. In *Proc. of FMCAD'02*, vol. 2517 of *LNCS*, pp. 88–105. Springer.
20. Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon. Hierarchical set decision diagrams and regular models. In *Proc. of TACAS'09*, vol. 5505 of *LNCS*, pp. 1–15. Springer.
21. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proc. of Banff'94*, vol. 1043 of *LNCS*, pp. 238–266. Springer.