# Extending Testing Automata to All LTL

Ala Eddine Ben Salem(✉)

LRDE, EPITA, Le Kremlin-Bicêtre, France
ala@lrde.epita.fr

**Abstract.** An alternative to the traditional Büchi Automata (BA), called Testing Automata (TA) was proposed by Hansen et al. [8, 6] to improve the automata-theoretic approach to LTL model checking. In previous work [2], we proposed an improvement of this alternative approach called TGTA (Generalized Testing Automata). TGTA mixes features from both TA and TGBA (Generalized Büchi Automata), without the disadvantage of TA, which is the second pass of the emptiness check algorithm. We have shown that TGTA outperform TA, BA and TGBA for explicit and symbolic LTL model checking. However, TA and TGTA are less expressive than Büchi Automata since they are able to represent only stutter-invariant LTL properties ($LTL\setminus$X) [13]. In this paper, we show how to extend Generalized Testing Automata (TGTA) to represent any LTL property. This allows to extend the model checking approach based on this new form of testing automata to check other kinds of properties and also other kinds of models (such as Timed models). Implementation and experimentation of this extended TGTA approach show that it is statistically more efficient than the Büchi Automata approaches (BA and TGBA), for the explicit model checking of LTL properties.

## 1 Introduction

The model checking of a behavioral property on a finite-state system is an automatic procedure that requires many phases. The first step is to formally represent the system and the property to be checked. The formalization of the system produces a model $M$ that formally describes all the possible executions of the system. The property to be checked is formally described using a specification language such as Linear-time Temporal Logic (LTL). The next step is to run a model checking algorithm that takes as inputs the model $M$ and the LTL property $\varphi$. This algorithm exhaustively checks that all the executions of the model $M$ satisfy $\varphi$. When the property is not satisfied, the model checker returns a *counterexample*, i.e., an execution of $M$ invalidating $\varphi$, this counterexample is particularly useful to find subtle errors in complex systems.

The *automata-theoretic approach* [16] to LTL model checking represents the state-space of $M$ and the property $\varphi$ using variants of $\omega$-automata, i.e., an extension of the classical finite automata to recognize words having infinite length (called $\omega$-words).

The *automata-theoretic approach* splits the verification process into four operations:

1. Computation of the state-space of $M$. This state-space can be represented by a variant of $\omega$-automaton, called Kripke structure $\mathcal{K}_M$, whose language $\mathscr{L}(\mathcal{K}_M)$, represents all possible infinite executions of $M$.
2. Translation of the negation of the LTL property $\varphi$ into an $\omega$-automaton $A_{\neg\varphi}$ whose language, $\mathscr{L}(A_{\neg\varphi})$, is the set of all infinite executions that would invalidate $\varphi$.

© IFIP International Federation for Information Processing 2015
S. Graf and M. Viswanathan (Eds.): FORTE 2015, LNCS 9039, pp. 196–210, 2015.
DOI: 10.1007/978-3-319-19195-9_13

3. Synchronization of these automata. This constructs a synchronous product automaton $\mathcal{K}_M \otimes A_{\neg\varphi}$ whose language, $\mathcal{L}(\mathcal{K}_M \otimes A_{\neg\varphi}) = \mathcal{L}(\mathcal{K}_M) \cap \mathcal{L}(A_{\neg\varphi})$, is the set of executions of $M$ invalidating $\varphi$.
4. Emptiness check of this product. This operation tells whether $\mathcal{K}_M \otimes A_{\neg\varphi}$ accepts an infinite word, and can return such a word (a counterexample) if it does. The model $M$ verifies $\varphi$ *iff* $\mathcal{L}(\mathcal{K}_M \otimes A_{\neg\varphi}) = \emptyset$.

The main difficulty of the LTL model checking is the state-space explosion problem. In particular, in the automata-theoretic approach, the product automaton $\mathcal{K}_M \otimes A_{\neg\varphi}$ is often too large to be emptiness checked in a reasonable run time and memory. Indeed, the performance of the automata-theoretic approach mainly depends on the size of the explored part of $\mathcal{K}_M \otimes A_{\neg\varphi}$ during the emptiness check. This explored part itself depends on three parameters: the automaton $A_{\neg\varphi}$ obtained from the LTL property $\varphi$, the Kripke structure $\mathcal{K}_M$ representing the state-space of $M$, and the emptiness check algorithm: the fact that this algorithm is performed "on-the-fly" potentially avoids building the entire product automaton. The states of this product that are not visited by the emptiness check are not generated at all.

Different kinds of $\omega$-automata have been used to represent $A_{\neg\varphi}$. In the most common case, the negation of $\varphi$ is converted into a *Büchi automaton* (BA) with state-based accepting. *Transition-based Generalized Büchi Automata* (TGBA) represent the LTL properties using *generalized* (i.e., multiple) Büchi acceptance conditions *on transitions* rather than on states. TGBA allow to have a smaller [7, 4] property automaton than BA.

Unfortunately, having a smaller property automaton $A_{\neg\varphi}$ does not always imply a smaller product ($A_M \otimes A_{\neg\varphi}$). Thus, instead of targeting smaller property automata, some people have attempted to build automata that are *more deterministic* [14].

Hansen et al. [8, 6] introduced an alternative type of $\omega$-automata called *Testing Automata* (TA) that only observe changes on the atomic propositions. TA are often larger than their equivalent BA, but according to Geldenhuys and Hansen [6], thanks to their high degree of determinism [8], the TA allow to obtain a smaller product and thus improve the performance of model checking. As a back-side, TA have two different modes of acceptance (Büchi-accepting or livelock-accepting), and consequently their emptiness check requires two passes [6], mitigating the benefits of a having a smaller product.

In previous work [2], we propose an improvement of TA called *Transition-based Generalized Testing Automata* (TGTA) that combine the advantages of both TA and TGBA, and without the disadvantages of TA (without introducing a second mode of acceptance and without the second pass of the emptiness check).

Unfortunately, the two variants of testing automata TA and TGTA are less expressive than Büchi automata (BA and TGBA) since they are tailored to represent *stutter-invariant* properties.

The goal of this paper is to extend TGTA in order to obtain a new form of testing automata that represent any LTL property, and therefore extend the model checking approach based on this alternative kind of automata to check other kinds of properties and also other kinds of models.

In order to remove the constraint that TGTA only represent stutter-invariant properties, one solution would be to change the construction of TGTA to take into account the "sub-parts" of the automata corresponding to the "sub-formulas" that are not insensitive to stuttering. Indeed, during the transformation of a TGBA into a TGTA, only the second step exploits the fact that the LTL property is stutter-invariant (this step allows to remove the useless stuttering-transitions). The idea is to apply this second step only for the parts of the automata that are insensitive to stuttering and to apply only the first step of the construction for the other parts (that are sensitive to stuttering).

We have run benchmarks to compare the new form of TGTA against BA and TGBA. Experiments reported that, in most cases, TGTA produce the smallest products ($A_M \otimes A_{\neg\varphi}$) and TGTA outperform BA and TGBA when no counterexample is found (i.e., the property is satisfied), but they are comparable when the property is violated, because in this case the on-the-fly algorithm stops as soon as it finds a counterexample without exploring the entire product.

## 2 Preliminaries

Let $AP$ a finite set of atomic propositions, a valuation $\ell$ over $AP$ is represented by a function $\ell : AP \mapsto \{\bot, \top\}$. 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 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 1 (Kripke Structure).** *A* Kripke structure *over the 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 the set of initial states,*
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ *is the transition relation,*
- $l : \mathcal{S} \to \Sigma$ *is a labeling function that maps each state s to a valuation that represents the set of atomic propositions that are true in s.*

The automata-theoretic approach is based on the transformation of the negation of the LTL property to be checked into an $\omega$-automaton that accepts the same executions. Büchi Automata (BA) are $\omega$-automata with labels on transitions and acceptance conditions on states. Büchi Automata are commonly used for LTL model checking (we use the abbreviation BA for the standard variant of Büchi Automata). The following section present TGBA [7]: a generalized variant of BA that allow a more compact representation of LTL properties [4].

### 2.1 Transition-Based Generalized Büchi Automata (TGBA)

A Transition-based Generalized Büchi Automaton (TGBA) [7] is a variant of a Büchi automaton that has multiple acceptance conditions on transitions.

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

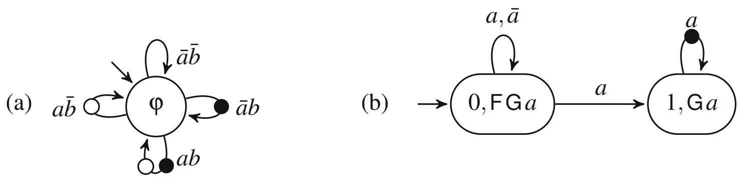


**Fig. 1.** (a) A TGBA recognizing the LTL property $\varphi = \mathsf{GF}a \wedge \mathsf{GF}b$ with acceptance conditions $\mathcal{F} = \{●, ○\}$ . (b) A TGBA recognizing the LTL property $\varphi = \mathsf{FG}a$ with $\mathcal{F} = \{●\}$ .

- *$Q$ is a finite set of states,*
- *$I \subseteq Q$ is a set of initial states,*
- *$\mathcal{F}$ is a finite set of acceptance conditions,*
- *$\delta \subseteq Q \times \Sigma \times 2^{\mathcal{F}} \times Q$ is the transition relation, where each element $(q, \ell, F, q') \in \delta$ represents a transition from state $q$ to state $q'$ labeled by a valuation $\ell \in 2^{AP}$, and a set of acceptance conditions $F \in 2^{\mathcal{F}}$.*

*An infinite word $\sigma = \ell_0 \ell_1 \ell_2 \ldots \in \Sigma^{\omega}$ is accepted by $\mathcal{G}$ if there exists an infinite run $r = (q_0, \ell_0, F_0, q_1)(q_1, \ell_1, F_1, q_2)(q_2, \ell_2, F_2, q_3) \ldots \in \delta^{\omega}$ where:*
- *$q_0 \in I$ (the infinite word is recognized by the run),*
- *$\forall f \in \mathcal{F}, \forall i \in \mathbb{N}, \exists j \geq i, f \in F_j$ (each acceptance condition is visited infinitely often).*

*The* language *of $\mathcal{G}$ is the set $\mathscr{L}(\mathcal{G}) \subseteq \Sigma^{\omega}$ of infinite words it accepts.*

Any LTL formula $\varphi$ can be converted into a TGBA whose language is the set of executions that satisfy $\varphi$ [4].

Figure 1 shows two examples of LTL properties expressed as TGBA. The Boolean expression over $AP = \{a, b\}$ that labels each transition represents the valuation of atomic propositions that hold in this transition. A run in these TGBA is accepted if it visits infinitely often all acceptance conditions (represented by colored dots ○ and ● on transitions). It is important to note that the LTL formulas labeling each state represent the property accepted starting from this state of the automaton. These labels are generated by our LTL-to-TGBA translator (Spot [12]), they are shown for the reader's convenience but not used for model checking.
Figure 1(a) is a TGBA recognizing the LTL formula $(\mathsf{GF}a \wedge \mathsf{GF}b)$, i.e., recognizing the runs where $a$ is true infinitely often and $b$ is true infinitely often. An accepting run in this TGBA has to visit infinitely often the two acceptance conditions indicated by ○ and ●. Therefore, it must explore infinitely often the transitions where $a$ is true (i.e., transitions labeled by $ab$ or $a\bar{b}$) and infinitely often the transitions where $b$ is true (i.e., transitions labeled by $ab$ or $\bar{a}b$).
Figure 1(b) shows a TGBA derived from the LTL formula $\mathsf{FG}a$. Any infinite run in this example is accepted if it visits infinitely often the only acceptance condition ● on transition $(1, a, 1)$. Therefore, an accepting run in this TGBA must stay on state 1 by executing infinitely $a$.

The product of a TGBA with a Kripke structure is a TGBA whose language is the intersection of both languages.

**Definition 3 (Product using TGBA).** *For a Kripke structure $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l \rangle$ and a TGBA $\mathcal{G} = \langle \mathcal{Q}, I, \delta, \mathcal{F} \rangle$ the product $\mathcal{K} \otimes \mathcal{G}$ is the TGBA $\langle \mathcal{S}_\otimes, I_\otimes, \delta_\otimes, \mathcal{F} \rangle$ where*

- $\mathcal{S}_\otimes = \mathcal{S} \times \mathcal{Q}$,
- $I_\otimes = \mathcal{S}_0 \times I$,
- $\delta_\otimes = \{((s,q), \ell, F, (s',q')) \mid (s,s') \in \mathcal{R}, (q, \ell, F, q') \in \delta, l(s) = \ell\}$

**Property 1.** *We have $\mathscr{L}(\mathcal{K} \otimes \mathcal{G}) = \mathscr{L}(\mathcal{K}) \cap \mathscr{L}(\mathcal{G})$ by construction.*

The goal of the emptiness check algorithm is to determine if the product automaton accepts an execution or not. In other words, it checks if the language of the product automaton is empty or not. Testing the TGBA (representing the product automaton) for emptiness amounts to the search of an accepting cycle that contains at least one occurrence of each acceptance condition. This can be done in different ways: either with a variation of Tarjan or Dijkstra algorithm [3] or using several Nested Depth-First Searches (NDFS) [15]. The product automaton that has to be explored during the emptiness check is generally very large, its size can reach the value obtained by multiplying the the sizes of the model and formula automata, which are synchronized to build this product. Therefore, building the entire product must be avoided. "On-the-fly" emptiness check algorithms allow the product automaton to be constructed lazily during its 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.

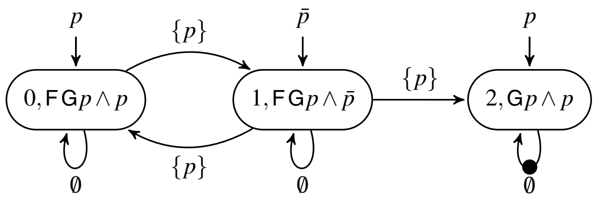
## 3 Transition-Based Generalized Testing Automata (TGTA)

Another kind of ω-automaton called *Testing Automaton (TA)* was introduced by Hansen et al. [8]. Instead of observing the valuations on states or transitions, the TA transitions only record the changes between these valuations. However, TA are less expressive than Büchi automata since they are able to represent only stutter-invariant LTL properties. Also they are often a lot larger than their equivalent Büchi automaton, but their high degree of determinism [8] often leads to a smaller product size [6].

In previous work [1], we evaluate the efficiency of LTL model checking approach using TA. We have shown that TA are better than Büchi automata (BA and TGBA) when the formula to be verified is violated (i.e., a counterexample is found), but this is not the case when the property is verified since the entire product have to be visited twice to check for each acceptance mode of a TA. Then, in order to improve the TA approach, we proposed in [2] a new ω-automata for stutter-invariant properties, called Transition-based Generalized Testing Automata (TGTA) [2], that mixes features from both TA and TGBA.

The basic idea of TGTA is to build an improved form of testing automata with generalized acceptance conditions on transitions, which allows us to modify the automata construction in order to remove the second pass of the emptiness check of the product.

Another advantage of TGTA compared to TA, is that the implementation of TGTA approach does not require a dedicated emptiness check, it reuses the same algorithm used for Büchi automata, and the counterexample constructed by this algorithm is also reported as a counterexample for the TGTA approach.



$p$
$\{p\}$
$\bar{p}$
$p$
$0, \mathsf{FG}p \wedge p$
$1, \mathsf{FG}p \wedge \bar{p}$
$\{p\}$
$2, \mathsf{G}p \wedge p$
$\{p\}$
$\emptyset$
$\emptyset$
$\emptyset$

**Fig. 2.** TGTA recognizing the LTL property $\varphi = \mathsf{FG}p$ with acceptance conditions $\mathcal{F} = \{\bullet\}$

In [2], we compared the LTL model checking approach using TGTA with the "traditional" approaches using BA, TGBA and TA. The results of these experimental comparisons show that TGTA compete well: the TGTA approach was statistically more efficient than the other evaluated approaches, especially when no counterexample is found (i.e., the property is verified) because it does not require a second pass. Unfortunately, the TGTA constructed in [2] only represent stutter-invariant properties (LTL$\setminus$X [13]).

This section shows how to adapt the TGTA construction to obtain a TGTA that can represent all LTL properties.

Before presenting this new construction of TGTA, we first recall the definition of this improved variant of testing automata.

While Büchi automata observe the values of the atomic propositions of $AP$, the basic idea of testing automata (TA and TGTA) is to only detect the *changes* in these values; if the values of the atomic propositions do not change between two consecutive valuations of an execution, this transition is called 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., $a\bar{b} \oplus ab = \{b\}$). Technically, this is implemented with an XOR operation (also denoted by the symbol $\oplus$).

**Definition 4 (TGTA).** *A TGTA over the alphabet* $\Sigma$ *is a tuple* $\mathcal{T} = \langle Q, I, U, \delta, \mathcal{F} \rangle$ *where:*

- $Q$ *is a finite set of states,*
- $I \subseteq Q$ *is a set of initial states,*
- $U : I \to 2^{\Sigma}$ *is a function mapping each initial state to a set of symbols of* $\Sigma$*,*
- $\mathcal{F}$ *is a finite set of acceptance conditions,*
- $\delta \subseteq Q \times \Sigma \times 2^{\mathcal{F}} \times Q$ *is the transition relation, where each element* $(q, k, F, q')$ *represents a transition from state* $q$ *to state* $q'$ *labeled by a* changeset $k$ *interpreted as a (possibly empty) set of atomic propositions whose values change between* $q$ *and* $q'$*, and the set of acceptance conditions* $F \in 2^{\mathcal{F}}$*,*

*An infinite word* $\sigma = \ell_0\ell_1\ell_2 \ldots \in \Sigma^{\omega}$ *is accepted by* $\mathcal{T}$ *if there exists an infinite run* $r = (q_0, \ell_0 \oplus \ell_1, F_0, q_1)(q_1, \ell_1 \oplus \ell_2, F_1, q_2)(q_2, \ell_2 \oplus \ell_3, F_2, q_3) \ldots \in \delta^{\omega}$ *where:*

- $q_0 \in I$ *with* $\ell_0 \in U(q_0)$ *(the infinite word is recognized by the run),*
- $\forall f \in \mathcal{F}, \forall i \in \mathbb{N}, \exists j \geq i, f \in F_j$ *(each acceptance condition is visited infinitely often).*

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

Figure 2 shows a TGTA recognizing the LTL formula $\mathsf{FG}p$. Acceptance conditions are represented using dots as in TGBAs. Transitions are labeled by changesets: e.g., the

transition $(0,\{p\},1)$ means that the value of $p$ changes between states 0 and 1. Initial valuations are shown above initial arrows: $U(0)=\{p\}$, $U(1)=\{\bar{p}\}$ and $U(2)=\{p\}$. Any infinite run in this example is accepted if it visits infinitely often, the acceptance transition indicated by the black dot ●: i.e., the stuttering self-loop $(2,\emptyset,\bullet,2)$.
As an illustration, the infinite word $\bar{p};p;p;p;\ldots$ is accepted by the run:
$① \xrightarrow{\{p\}} ② \xrightarrow{\overset{\emptyset}{\bullet}} ② \xrightarrow{\overset{\emptyset}{\bullet}} ② \cdots$ because the value $p$ only changes between the first two steps. Indeed, a run recognizing such an infinite word must start in state 1 (because only $U(1)=\{\bar{p}\}$), then it changes the value of $p$, so it has to take transitions labeled by $\{p\}$, i.e., $(1,\{p\},0)$ or $(1,\{p\},2)$. To be accepted, it must move to state 2 (rather than state 0), and finally stay on state 2 by executing infinitely the accepting stuttering self-loop $(2,\emptyset,\bullet,2)$.

In the next section, we present in detail the formalization of the different steps used to build a TGTA that represent any LTL property.

## 4 TGTA Construction

Let us now describe how to build a TGTA starting from a TGBA. The TGTA construction is inspired by the one presented in [2], with some changes introduced in the second step of this construction. Indeed, a TGTA is built in two steps as illustrated in Figure 3. The first step transforms a TGBA into an intermediate TGTA by labeling transitions with changesets. Then, the second step builds the final form of TGTA by removing the useless stuttering transitions. In this work, this simplification of stuttering transitions does not require the hypothesis that the LTL property is stutter-invariant (this represents a crucial difference compared to the TGTA construction presented in [2]). For example, Figure 4d shows a TGTA constructed for $\varphi = \mathsf{X}\, p \wedge \mathsf{F}\,\mathsf{G}\, p$ which is not stutter-invariant. In the following, we will detail the successive steps to build this TGTA.

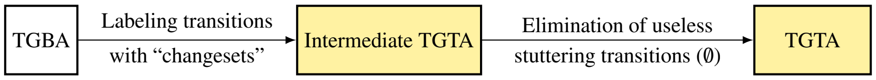
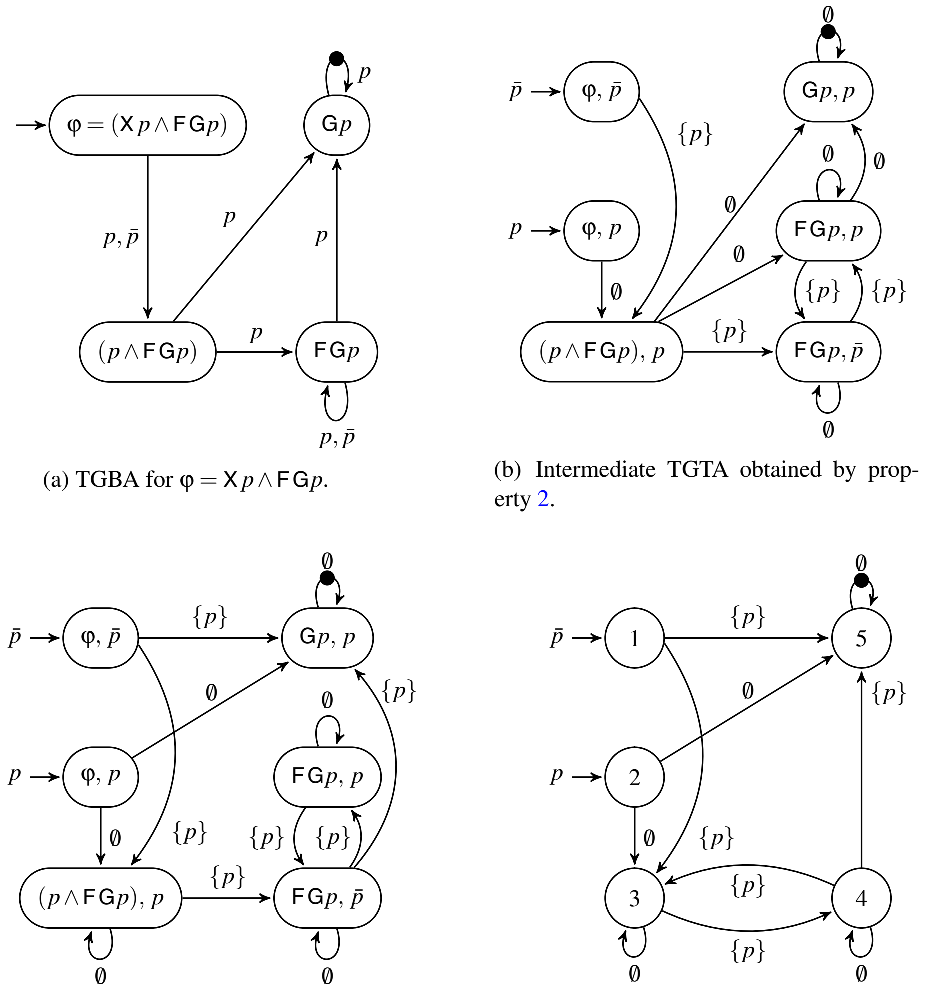


**Fig. 3.** The two steps of the construction of a TGTA from a TGBA

### 4.1 First Step: Construction of an Intermediate TGTA from a TGBA

Geldenhuys and Hansen [6] have shown how to convert a Büchi Automaton (BA) into a Testing Automtaton (TA) by first converting the BA into an automaton with valuations on the states (called State-Labeled Büchi Automaton (SLBA)), and then converting this SLBA into an intermediate form of TA by computing the difference between the labels of the source and destination states of each transition.

The first step of the TGTA construction is similar to the first step of the TA construction [6, 2]. We construct an intermediate TGTA from a TGBA by moving labels to states, and labeling each transition by the set difference between the labels of its source and destination states. While doing so, we keep the generalized acceptance conditions on the transitions. The next proposition implements these first steps.



(a) TGBA for $\varphi = \mathsf{X}p \wedge \mathsf{FG}p$.

(b) Intermediate TGTA obtained by property 2.

(c) TGTA after simplifications by property 3.

(d) TGTA after bisimulation.

**Fig. 4.** TGTA obtained after various steps while translating the TGBA representing $\varphi = \mathsf{X}p \wedge \mathsf{FG}p$, into a TGTA with $\mathcal{F} = \{\bullet\}$

**Property 2 (Converting TGBA into an intermediate TGTA).** *Given any TGBA* $\mathcal{G} = \langle Q_\mathcal{G}, I_\mathcal{G}, \delta_\mathcal{G}, \mathcal{F}\rangle$ *over the alphabet* $\Sigma$*, let us build the TGTA* $\mathcal{T} = \langle Q_\mathcal{T}, I_\mathcal{T}, U_\mathcal{T}, \delta_\mathcal{T}, \mathcal{F}\rangle$ *with* $Q_\mathcal{T} = Q_\mathcal{G} \times \Sigma$*,* $I_\mathcal{T} = I_\mathcal{G} \times \Sigma$ *and*
*(i)* $\forall (q,\ell) \in I_\mathcal{T}, U_\mathcal{T}((q,\ell)) = \{\ell\}$
*(ii)* $\forall (q,\ell) \in Q_\mathcal{T}, \forall (q',\ell') \in Q_\mathcal{T}, \big((q,\ell), \ell \oplus \ell', F, (q',\ell')\big) \in \delta_\mathcal{T} \iff ((q,\ell,F,q') \in \delta_\mathcal{G})$
*Then* $\mathscr{L}(\mathcal{G}) = \mathscr{L}(\mathcal{T})$. (The proof of this property. 2 is given in [2].)

An example of an intermediate TGTA is shown on Figure 4b. It is the result of applying the construction of property. 2 to the example of TGBA for $\varphi = \mathsf{X}p \wedge \mathsf{FG}p$ (shown in Figure 4a). The next property shows how to remove the useless stuttering-transitions (labeled by $\emptyset$) in TGTA.

## 4.2 Second Step: Elimination of Useless Stuttering-Transitions ($\emptyset$)

In the following, we say that a language $\mathscr{L}$ is *stutter-invariant* if the number of the successive repetitions of any letter of a word $\sigma \in \mathscr{L}$ does not affect the membership of $\sigma$ to $\mathscr{L}$ [5]. In other words, $\mathscr{L}$ is stutter-invariant *iff* for any finite sequence $u \in \Sigma^*$, any element $\ell \in \Sigma$, and any infinite sequence $v \in \Sigma^\omega$ we have $u\ell v \in \mathscr{L} \iff u\ell\ell v \in \mathscr{L}$.

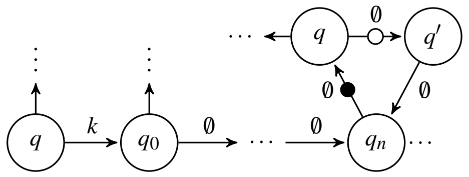
We begin by defining the concept of a *language recognized starting from a state* in a TGTA. This definition will be very useful for the formalization of the second step of the TGTA construction presented below.

**Definition 5** **($\mathscr{L}(\mathcal{T},q)$).** *Given a TGTA $\mathcal{T}$ and a state $q$ of $\mathcal{T}$, we say that an infinite word $\sigma = \ell_0\ell_1\ell_2\ldots \in \Sigma^\omega$ is accepted by $\mathcal{T}$* starting from *the state $q$ if there exists an infinite run $r = (q,\ell_0 \oplus \ell_1, F_0, q_1)(q_1, \ell_1 \oplus \ell_2, F_1, q_2)(q_2, \ell_2 \oplus \ell_3, F_2, q_3)\ldots \in \delta^\omega$ where: $\forall f \in \mathcal{F}, \forall i \in \mathbb{N}, \exists j \geq i, f \in F_j$ (each acceptance condition is visited infinitely often). The language $\mathscr{L}(\mathcal{T},q) \subseteq \Sigma^\omega$ is the set of infinite words accepted by $\mathcal{T}$* starting from *the state $q$.*

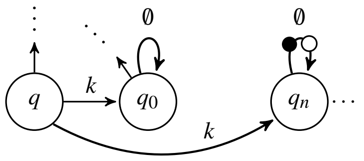
In the following, we will exploit the fact that in a TGTA, the language recognized starting from certain states of a TGTA can be stutter-invariant, although the overall TGTA language (recognized from the initial states) is not stutter-invariant. For example, in the intermediate TGTA shown on Figure 4b, the language recognized starting from the two initial states (labeled by the formula $\varphi = \mathsf{X}p \wedge \mathsf{FG}p$) is not stutter-invariant. However, the languages recognized starting form the other states (labeled by the formulas $(p \wedge \mathsf{FG}p)$, $(\mathsf{FG}p)$ and $(\mathsf{G}p)$) are stutter-invariants. Indeed, similar to the original TGBA $\mathcal{G}$, in an intermediate TGTA $\mathcal{T}$ obtained by property 2, if the LTL formula labeling a state $q$ is stutter-invariant, then the language recognized starting from this state $q$ is also stutter-invariant (Figure 4b). This can easily be deduced from the proof [2] of property 2.

The next property allow to simplify the intermediate TGTA by removing the useless stuttering transitions and thus obtain the final TGTA. The intuition behind this simplification is illustrated in Figure 5a: In a TGTA $\mathcal{T}$, we have that the language recognized starting from a state $q_0$ is stutter-invariant and $q_0$ can reach an accepting stuttering-cycle by following only stuttering transitions. In the context of TA we would have to declare $q_0$ as being a livelock-accepting state. For TGTA, we replace the accepting stuttering-cycle by adding a self-loop labeled by all acceptance conditions on $q_n$, then the predecessors of $q_0$ are connected to $q_n$ as in Figure 5b. In the last step of the following construction, for each state $q$ such that $\mathscr{L}(\mathcal{T},q)$ is stutter-invariant, we add a stuttering self-loop to $q$ and we remove all stuttering transitions from $q$ to other states. Figure 4c shows how the automaton from Figure 4b is simplified.

**Property 3** **(Elimination of useless stuttering transitions to build a TGTA).** *Given a TGTA $\mathcal{T} = \langle \mathcal{Q}, I, U, \delta, \mathcal{F} \rangle$. By combining the first three of the following operations,*



(a) TGTA $\mathcal{T}$ before reduction of stuttering transitions ($\emptyset$). $\mathscr{L}(\mathcal{T},q_0)$ is stutter-invariant.



(b) $\mathcal{T}$ after reduction of stuttering transitions ($\emptyset$).

**Fig. 5.** Elimination of useless stuttering transitions in TGTA

*we can remove the useless stuttering-transitions in this TGTA (Figure 5). The fourth operation can be performed along the way for further (classical) simplifications.*

1. *If $Q \subseteq \mathcal{Q}$ is a SCC such that for any state $q \in Q$ we have $\mathscr{L}(\mathcal{T},q)$ is stutter-invariant and any two states $q,q' \in Q$ can be connected using a sequence of stuttering transitions $(q,\emptyset,F_0,r_1)(r_1,\emptyset,F_1,r_2)\cdots(r_n,\emptyset,F_n,q') \in \delta^*$ with $F_0 \cup F_1 \cup \cdots \cup F_n = \mathcal{F}$, then we can add an accepting stuttering self-loop $(q,\emptyset,\mathcal{F},q)$ on each state $q \in Q$. I.e., the TGTA $\mathcal{T}' = \langle \mathcal{Q}, I, U, \delta \cup \{(q,\emptyset,\mathcal{F},q) \mid q \in Q\}, \mathcal{F}\rangle$ is such that $\mathscr{L}(\mathcal{T}') = \mathscr{L}(\mathcal{T})$. Let us call such a component $Q$ an* ***accepting Stuttering-SCC****.*
2. *Let $q_0$ is a state of $\mathcal{T}$ such that $\mathscr{L}(\mathcal{T},q_0)$ is stutter-invariant. If there exists an accepting Stuttering-SCC $Q$ and a sequence of stuttering-transitions:*
   *$(q_0,\emptyset,F_1,q_1)(q_1,\emptyset,F_2,q_2)\cdots(q_{n-1},\emptyset,F_n,q_n) \in \delta^*$ such that $q_n \in Q$ and $q_0$, $q_1,\ldots$ $q_{n-1} \notin Q$ (as shown in Figure 5a), then:*
   - *For any transition $(q,k,F,q_0) \in \delta$ going to $q_0$ (with $(q,k,F,q_n) \notin \delta$), the TGTA $\mathcal{T}'' = \langle \mathcal{Q}, I, U, \delta \cup \{(q,k,F,q_n)\}, \mathcal{F}\rangle$ is such that $\mathscr{L}(\mathcal{T}'') = \mathscr{L}(\mathcal{T})$ (Figure 5b).*
   - *If $q_0 \in I$, the TGTA $\mathcal{T}'' = \langle \mathcal{Q}, I \cup \{q_n\}, U'', \delta, \mathcal{F}\rangle$ with $\forall q \neq q_n$, $U''(q) = U(q)$ and $U''(q_n) = U(q_n) \cup U(q_0)$, is such that $\mathscr{L}(\mathcal{T}'') = \mathscr{L}(\mathcal{T})$.*
3. *Let $\mathcal{T}^\dagger = \langle \mathcal{Q}, I^\dagger, U^\dagger, \delta^\dagger, \mathcal{F}\rangle$ be the TGTA obtained after repeating the previous two operations as much as possible (i.e., $\mathcal{T}^\dagger$ contains all the transitions and initial states that can be added by the above two operations. Then, we add a non-accepting stuttering self-loop $(q,\emptyset,\emptyset,q)$ to any state $q$ that did not have an accepting stuttering self-loop and such that $\mathscr{L}(\mathcal{T},q)$ is stutter-invariant. Also we remove all stuttering transitions from $q$ that are not self-loops since stuttering can be captured by self-loops after the previous two operations. After this last reduction of stuttering transitions, we obtain the final TGTA (Figure 4c).*
   *More formally, the TGTA $\mathcal{T}''' = \langle \mathcal{Q}, I^\dagger, U^\dagger, \delta''', \mathcal{F}\rangle$ with $\delta''' = \{(q,k,F,q') \in \delta^\dagger \mid$ $\mathscr{L}(\mathcal{T},q)$ is* ***not stutter-invariant*** *$\} \cup \{(q,k,F,q') \in \delta^\dagger \mid k \neq \emptyset \vee (q = q' \wedge F = \mathcal{F})\} \cup$ $\{(q,\emptyset,\emptyset,q) \mid \mathscr{L}(\mathcal{T},q)$ is stutter-invariant $\wedge\, (q,\emptyset,\mathcal{F},q) \notin \delta^\dagger\}$ is such that:*
   *$\mathscr{L}(\mathcal{T}''') = \mathscr{L}(\mathcal{T}^\dagger) = \mathscr{L}(\mathcal{T})$.*
4. *Any state from which one cannot reach a Büchi-accepting cycle can be removed from the automaton without changing its language.*

The proof of Property 3 is similar to the proof of the second step of TGTA construction given in [2].

Figure 4c shows how the TGTA from Figure 2 is simplified by the above Property 3.

Similar to the TA construction [2], the resulting TGTA can be further simplified by merging bisimilar states (two states $q$ and $q'$ are bisimilar if the automaton $\mathcal{T}$ can accept the same infinite words starting from either of these states, i.e., $\mathscr{L}(\mathcal{T},q) = \mathscr{L}(\mathcal{T},q')$). This optimization can be achieved using any algorithm based on partition refinement, the same as for Büchi automata, taking $\{\mathcal{F} \cap \mathcal{G}, \mathcal{F} \setminus \mathcal{G}, \mathcal{G} \setminus \mathcal{F}, \mathcal{Q} \setminus (\mathcal{F} \cup \mathcal{G})\}$ as initial partition and taking into account the acceptance conditions of the outgoing transitions. The final TGTA obtained after all these steps is shown in Figure 4.

As for the other variants of ω-automata, the automata-theoretic approach using TGTA has two important operations: the construction of a TGTA $\mathcal{T}$ recognizing the negation of the LTL property φ and the emptiness check of the product $(\mathcal{K} \otimes \mathcal{T})$ of the Kripke structure $\mathcal{K}$ with $\mathcal{T}$.

**Definition 6 (Product using TGTA).** *For a Kripke structure* $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, l\rangle$ *and a TGTA* $\mathcal{T} = \langle \mathcal{Q}, I, U, \delta, \mathcal{F}\rangle$*, the product* $\mathcal{K} \otimes \mathcal{T}$ *is a TGTA* $\langle \mathcal{S}_\otimes, I_\otimes, U_\otimes, \delta_\otimes, \mathcal{F}_\otimes\rangle$ *where*

– $\mathcal{S}_\otimes = \mathcal{S} \times \mathcal{Q}$,
– $I_\otimes = \{(s,q) \in \mathcal{S}_0 \times I \mid l(s) \in U(q)\}$,
– $\forall (s,q) \in I_\otimes, U_\otimes((s,q)) = \{l(s)\}$,
– $\delta_\otimes = \{((s,q),k,F,(s',q')) \mid (s,s') \in \mathcal{R}, (q,k,F,q') \in \delta, k = (l(s) \oplus l(s'))\}$,
– $\mathcal{F}_\otimes = \mathcal{F}$.

**Property 4.** *We have* $\mathscr{L}(\mathcal{K} \otimes \mathcal{T}) = \mathscr{L}(\mathcal{K}) \cap \mathscr{L}(\mathcal{T})$ *by construction.*

Since a product of a TGTA with a Kripke structure is a TGTA, we only need an emptiness check algorithm for a TGTA automaton. A TGTA can be seen as a TGBA whose transitions are labeled by changesets instead of valuations of atomic propositions. When checking a TGBA for emptiness, we are looking for an accepting cycle that is reachable from an initial state. When checking a TGTA for emptiness, we are looking exactly for the same thing. Therefore, because emptiness check algorithms do not look at transitions labels, the same emptiness check algorithm used for the product using TGBA can also be used for the product using TGTA.

## 5 Experimental Evaluation of TGTA

In order to evaluate the TGTA approach against the TGBA and BA approaches, an experimentation was conducted under the same conditions as our previous work [1], i.e., within the same CheckPN tool on top of Spot [12] and using the same benchmark Inputs (formulas and models) used in the experimental comparison [1] of BA, TGBA and TA. The models are from the Petri net literature [11], we selected two instances of each of the following models: the Flexible Manufacturing System (4/5), the Kanban system (4/5), the Peterson algorithm (4/5), the slotted-ring system (6/5), the dining philosophers (9/10) and the Round-robin mutex (14/15). We also used two models from actual case studies: **PolyORB** [10] and **MAPK** [9]. For each selected model instance, we generated 200 verified formulas (no counterexample in the product) and 200 violated formulas (a counterexample exists): 100 random (length 15) and 100 weak-fairness [1] (length 30) of the two cases of formulas. Since generated formulas are very often trivial to verify (the emptiness check needs to explore only a handful of states), we selected only those formulas requiring more than one second of CPU for the emptiness check in all approaches.

## 5.1 Implementation

Figure 6 shows the building blocks we used to implement the three approaches. The automaton used to represent the property to check has to be synchronized with a Kripke structure representing the model. Depending on the kind of automaton, this synchronous product is implemented differently. The TGBA and BA approaches can share the same product implementation. The TGTA approach require a dedicated product computation. The TGBA, BA, and TGTA approaches share the same emptiness check.

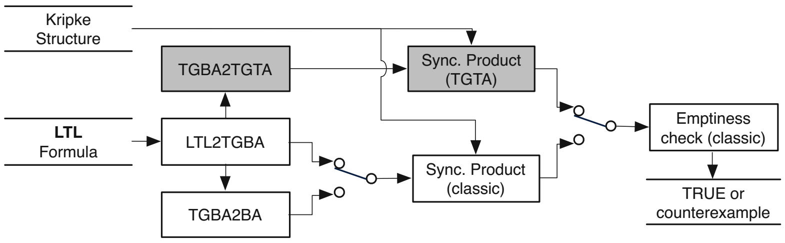


**Fig. 6.** The experiment's architecture in Spot. Three command-line switches control which one of the approaches is used to verify an LTL formula on a Kripke structure. The new components required by the TGTA approach are outlined in Gray.

## 5.2 Results

Figure 7 compares the sizes of the products automata (in terms of number of states) and Figure 8 compares the number of visited transitions when running the emptiness check; plotting TGTA against BA and TGBA. This gives an idea of their relative performance. Indeed, in order to protect the results against the influence of various optimizations, implementation tricks, and the central processor and memory architecture, Geldenhuys and Hansen [6] found that the number of states gives a reliable indication of the memory required, and, similarly, the number of transitions a reliable indication of the time consumption. Each point of the scatter plots corresponds to one of the 5600 evaluated formulas (2800 violated with counterexample as black circles, and 2800 verified having no counterexample as green crosses). Each point below the diagonal is in favor of TGTA while others are in favor of the other approach. Axes are displayed using a logarithmic scale. All these experiments were run on a 64bit Linux system running on an Intel(R) 64-bit Xeon(R) @2.00GHz, with 10GB of RAM.

## 5.3 Discussion

**On verified properties** (green crosses), the results are very straightforward to interpret. On the scatter plots of Figure 7, the cases where the TGTA approach is better than BA and TGBA approaches, appear as green crosses below the diagonal. In these cases,

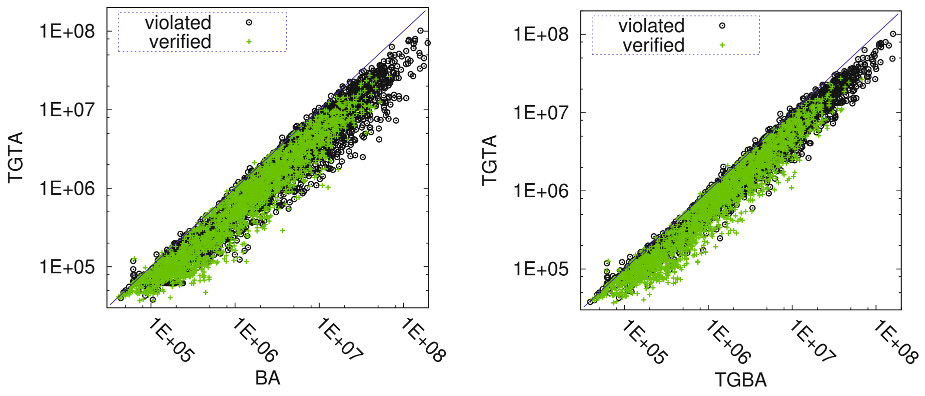


**Fig. 7.** Size of products (number of states) using TGTA against BA and TGBA

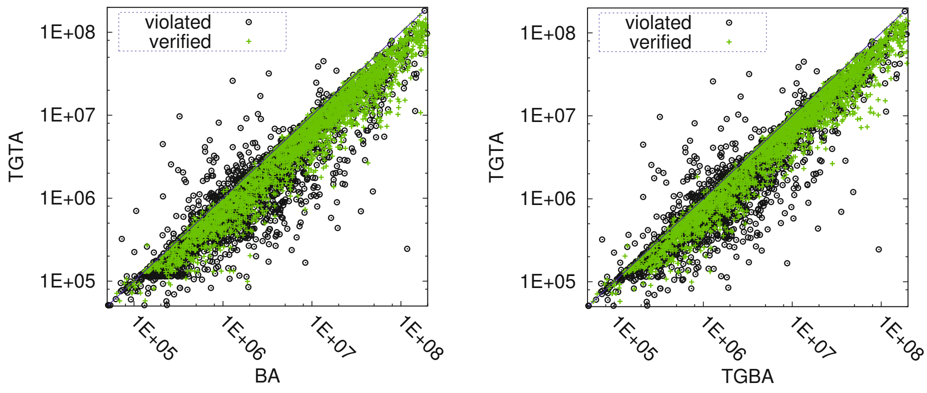


**Fig. 8.** Performance (number of transitions explored by the emptiness check) using TGTA against BA and TGBA

The TGTA approach is a clear improvement because the products automata are smaller using TGTA. The same result is observed for the scatter plots of Figure 8, when looking at the number of transitions explored by the emptiness check, the TGTA approach also outperforms TGBA and BA approaches for verified properties.

**On violated properties** (black circles), for the number of transitions explored by the emptiness check, it is difficult to interpret the scatter plots of Figure 8 because the emptiness check is an on-the-fly algorithm. It stops as soon as it finds a counterexample without exploring the entire product. Thus, for violated properties, the exploration order of non-deterministic transitions of TGBA, BA and TGTA changes the number of transitions explored in the product before a counterexample is found.

However, if we analyze the scatter plots of Figure 7, we observe that the TGTA approach produces the smallest products. This allows the TGTA approach to seek a counterexample in a smaller product and therefore have a better chance to find it faster.

## 6 Conclusion

In previous work [2], we have shown that *Transition-based Generalized Testing Automata* (TGTA) are a way to improve the model checking approach when verifying stutter-invariant properties. In this work, we propose a construction of a TGTA that allow to check any LTL property (stutter-invariant or not). This TGTA is constructed in two steps. The first one builds an intermediate TGTA from a TGBA (*Transition-based Generalized Büchi Automata*). The second step transforms an intermediate TGTA into a TGTA by removing the useless stuttering-transitions that are not self-loops (this reduction does not need the restriction to stutter-invariant properties as in our previous work [2]).

The constructed TGTA combines advantages observed on both Testing Automata (TA) and TGBA:

- From TA, it reuses the labeling of transitions with changesets, and the elimination of the useless stuttering-transitions, but without requiring a second pass in the emptiness check of the product.
- From TGBA, it inherits the use of generalized acceptance conditions on transitions.

TGTA have been implemented in Spot easily, because only two new algorithms are required: the conversion of a TGBA into a TGTA, and a new definition of a product between a TGTA and a Kripke structure.

We have run benchmarks to compare TGTA against BA and TGBA. Experiments reported that TGTA produce the smallest products automata and therefore TGTA outperform BA and TGBA when no counterexample is found in these products (i.e., the property is satisfied), but they are comparable when the property is violated, because in this case the on-the-fly algorithm stops as soon as it finds a counterexample without exploring the entire product.

We conclude that there is nothing to lose by using TGTA to verify any LTL property, since they are always at least as good as BA and TGBA and we believe that TGTA are better thanks to the elimination of the useless stuttering-transitions during the TGTA construction.

As a future work, an idea would be to provide a direct conversion of LTL to TGTA, without the intermediate TGBA step. We believe a tableau construction such as the one of Couvreur [3] could be easily adapted to produce TGTA. Another important optimization is to build on-the-fly the TGTA during the construction of the synchronous product. Especially when the number of atomic propositions (*AP*) is very large, because this may lead to build a TGTA with a large number of unnecessary initial states, that are not synchronized with the initial state(s) of the Kripke structure.

## References

1. Ben Salem, A.E., Duret-Lutz, A., Kordon, F.: Generalized Büchi automata versus testing automata for model checking. In: Proc. of SUMo 2011, vol. 726, pp. 65–79. CEUR (2011)
2. Ben Salem, A.-E., Duret-Lutz, A., Kordon, F.: Model Checking Using Generalized Testing Automata. In: Jensen, K., van der Aalst, W.M., Ajmone Marsan, M., Franceschinis, G., Kleijn, J., Kristensen, L.M. (eds.) ToPNoC VI. LNCS, vol. 7400, pp. 94–122. Springer, Heidelberg (2012)

3. Couvreur, J.-M.: On-the-fly verification of linear temporal logic. In: Wing, J.M., Woodcock, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 253–271. Springer, Heidelberg (1999)
4. Duret-Lutz, A., Poitrenaud, D.: SPOT: an extensible model checking library using transition-based generalized Büchi automata. In: Proc. of MASCOTS 2004, pp. 76–83. IEEE Computer Society Press (2004)
5. Etessami, K.: Stutter-invariant languages, ω-automata, and temporal logic. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 236–248. Springer, Heidelberg (1999)
6. Geldenhuys, J., Hansen, H.: Larger automata and less work for LTL model checking. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 53–70. Springer, Heidelberg (2006)
7. Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulæ to Büchi automata. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, pp. 308–326. Springer, Heidelberg (2002)
8. Hansen, H., Penczek, W., Valmari, A.: Stuttering-insensitive automata for on-the-fly detection of livelock properties. In: Proc. of FMICS 2002. ENTCS, vol. 66(2). Elsevier (2002)
9. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets for systems and synthetic biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)
10. Hugues, J., Thierry-Mieg, Y., Kordon, F., Pautet, L., Barrir, S., Vergnaud, T.: On the formal verification of middleware behavioral properties. In: Proc. of FMICS 2004. ENTCS, vol. 133, pp. 139–157. Elsevier (2004)
11. Kordon, F., et al.: Report on the model checking contest at petri nets 2011. In: Jensen, K., van der Aalst, W.M., Ajmone Marsan, M., Franceschinis, G., Kleijn, J., Kristensen, L.M. (eds.) ToPNoC VI. LNCS, vol. 7400, pp. 169–196. Springer, Heidelberg (2012)
12. MoVe/LRDE. The Spot home page: http://spot.lip6.fr (2014)
13. Peled, D., Wilke, T.: Stutter-invariant temporal properties are expressible without the next-time operator. Information Processing Letters 63(5), 243–246 (1995)
14. Sebastiani, R., Tonetta, S.: "More deterministic" vs. "Smaller" Büchi automata for efficient LTL model checking. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 126–140. Springer, Heidelberg (2003)
15. Tauriainen, H.: Automata and Linear Temporal Logic: Translation with Transition-based Acceptance. PhD thesis, Helsinki University of Technology, Espoo, Finland (September 2006)
16. Vardi, M.Y.: Automata-theoretic model checking revisited. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 137–150. Springer, Heidelberg (2007)