Mechanizing the Minimization of Deterministic Generalized Büchi Automata

Souheib Baarir*[1,2] and Alexandre Duret-Lutz[3]

[1] Université Paris Ouest Nanterre la Défense, Nanterre, France
[2] Sorbonne Universités, UPMC Univ. Paris 6, UMR 7606, LIP6, Paris, France
souheib.baarir@lip6.fr
[3] LRDE, EPITA, Le Kremlin-Bicêtre, France
adl@lrde.epita.fr

Abstract. Deterministic Büchi automata (DBA) are useful to (probabilistic) model checking and synthesis. We survey techniques used to obtain and minimize DBAs for different classes of properties. We extend these techniques to support DBA that have generalized and transition-based acceptance (DTGBA) as they can be even smaller. Our minimization technique—a reduction to a SAT problem—synthesizes a DTGBA equivalent to the input DTGBA for any given number of states and number of acceptance sets (assuming such automaton exists). We present benchmarks using a framework that implements all these techniques.

1 Introduction

Deterministic Büchi automata (DBA) are used in several domains like probabilistic model checking [3] or synthesis of distributed systems [13]. DBAs are commonly obtained from Büchi automata (BA) by applying Safra's construction [20] or, when it works, some powerset-based construction [9]. Since applications are usually more efficient when dealing with small automata, it makes sense to try to minimize these DBA (i.e., find an equivalent DBA such that no smaller equivalent DBA exist). Two techniques can be used depending on the class of the DBA at hand. Weak DBAs (where accepting cycles cannot mix with rejecting cycles) are minimizable almost like finite automata [16]. For the general case, which is NP-complete [21], Ehlers [11] gives a reduction to a SAT problem.

For model checking, several efficient algorithms have been developed to check the emptiness of BAs, but also generalized Büchi automata (GBA) [19], and even transition-based generalized Büchi automata (TGBA) [8]. GBAs have multiple sets of accepting states, while TGBAs have multiple sets of accepting transitions. All these automata types are expressively equivalent, but GBAs and TGBAs are more concise than BAs. Furthermore, TGBAs are naturally obtained from linear-time temporal logic (LTL) formulae [10].

In this paper, we discuss the construction of minimal deterministic transition-based generalized Büchi automata (DTGBA) for which we do not know any

* This work has been supported by the project ImpRo/ANR-2010-BLAN-0317.

general minimization technique. To handle (non-deterministic) TGBA as input, we propose (and implement) two processing chains that convert TGBA into minimal DTGBA. One attempts to determinize transition-based Büchi automata via a powerset construction, the other uses Safra's construction. Although these two determinizations do not work with generalized acceptance, we interpret the resulting automaton as a DTGBA with a single acceptance set and attempt to minimize it using *more* acceptance sets. The minimization technique we propose is a generalization of Ehlers' SAT-based procedure [11] to deal with DTGBA instead of DBA. This way we effectively obtain a minimal DTGBA, even though we use determinization procedures that do not support generalized acceptance.

The paper is organized as follows. Section 2 defines the various types of automata, some conversions between them, and key concepts. Section 3 reviews the hierarchy of recurrence properties and algorithms that exist to determinize or minimize automata in that class. Section 4 shows how to encode the minimization of a DTGBA as a SAT-problem, and then discusses its use. Finally, Section 5 presents results on the minimization of DTGBA obtained from LTL formulae.

2 Preliminaries

Let AP be a finite set of (atomic) propositions, and let $\mathbb{B} = \{\bot, \top\}$ represent Boolean values. An assignment is a function $\ell : \mathrm{AP} \to \mathbb{B}$ that valuates each proposition. $\Sigma = \mathbb{B}^{\mathrm{AP}}$ is the set of all assignments of AP. X^* (resp. X^ω) denotes the set of finite (resp. infinite) sequences over a set X. The empty sequence is denoted ε. A *word* $w \in (\mathbb{B}^{\mathrm{AP}})^\omega$ is an infinite sequence of assignments. A *language* (also called *property*) is a set of words. For any infinite sequence π we denote by $\inf(\pi)$ the set of elements that appear infinitely often in π.

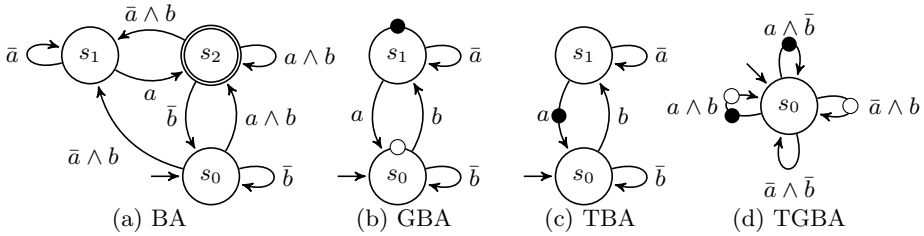
Definition 1 (Labeled Transition System). *An* LTS *is a tuple* $\mathcal{S} = \langle \mathrm{AP}, Q, \iota, \delta \rangle$ *where* AP *is a finite set of atomic propositions,* Q *is a finite set of states,* $\iota \in Q$ *is the initial state,* $\delta \subseteq Q \times \mathbb{B}^{\mathrm{AP}} \times Q$ *is the transition relation, labeling each transition by an assignment.*

A run of an LTS is an infinite sequence $\pi = (q_0, \ell_0, q_1)(q_1, \ell_1, q_2)(q_2, \ell_2, q_3) \cdots \in \delta^\omega$ *of transitions such that* $q_0 = \iota$*. For any such run, we denote by* $\pi_{|Q} = q_0 q_1 q_2 \cdots \in Q^\omega$ *the sequence of states it visits, and we say that* π *recognizes the word* $\ell_0 \ell_1 \ell_2 \cdots \in (B^{\mathrm{AP}})^\omega$*.*

A cycle $C = (q_1, \ell_1, q_2)(q_2, \ell_2, q_3) \ldots (q_n, \ell_n, q_1)$ *is a finite sequence of transitions that forms a loop and visits the finite sequence* $C_{|Q} = q_1 \ldots q_n$ *of states. For notational convenience, we sometime interpret* $C_{|Q}$ *as a set of states.*

As an implementation optimization, and to simplify illustrations, it is practical to use *edges* labeled by Boolean formulae to group *transitions* with same sources and destinations: for instance two *transitions* $(s_1, a\bar{b}, s_2)$ and (s_1, ab, s_2) will be represented by an *edge* from s_1 to s_2 and labeled by the Boolean formula a.

For convenience of notation, we may view the relation δ as a one-argument function $\delta : Q \to 2^{(\mathbb{B}^{\mathrm{AP}} \times Q)}$ with $\delta(q) = \{(\ell, d) \mid (q, \ell, d) \in \delta\}$ or as a two-argument function $\delta : Q \times \mathbb{B}^{\mathrm{AP}} \to 2^Q$ with $\delta(q, \ell) = \{d \mid (q, \ell, d) \in \delta\}$.



(a) BA (b) GBA (c) TBA (d) TGBA

Fig. 1: Minimal deterministic automata recognizing the LTL formula $\mathsf{GF}a \wedge \mathsf{GF}b$.

Definition 2 (Deterministic and Complete). *An LTS $\mathcal{S} = \langle \mathrm{AP}, Q, \iota, \delta \rangle$ is deterministic iff $|\delta(q, \ell)| \leq 1$ for all $q \in Q$ and $\ell \in \mathbb{B}^{\mathrm{AP}}$, and it is complete iff $|\delta(q, \ell)| \geq 1$ for all q and ℓ.*

To characterize the infinite runs of an LTS that should be accepted, we consider several variants of Büchi automata that differ by their acceptance condition.

Definition 3 (Büchi-like Automata). *A Büchi-like automaton is a pair $\mathcal{A} = \langle S, \mathcal{F} \rangle$ where S is an LTS, and $\mathcal{F}$ is a finite set whose semantic is defined differently for each type of acceptance. Given a run π and a cycle C:*

BA: *for Büchi Automata, $\mathcal{F} \subseteq Q$, π is accepted iff $\inf(\pi_{|Q}) \cap \mathcal{F} \neq \emptyset$, and C is accepting iff $C_{|Q} \cap \mathcal{F} \neq \emptyset$;*

GBA: *for Generalized Büchi Automata, $\mathcal{F} \subseteq 2^Q$, π is accepted iff $\forall F \in \mathcal{F}$, $\inf(\pi_{|Q}) \cap F \neq \emptyset$, and C is accepting iff $\forall F \in \mathcal{F}$, $C_{|Q} \cap F \neq \emptyset$;*

TBA: *for Transition-based Büchi Automata, $\mathcal{F} \subseteq \delta$, π is accepted iff $\inf(\pi) \cap \mathcal{F} \neq \emptyset$, and C is accepting iff $C \cap \mathcal{F} \neq \emptyset$;*

TGBA: *for Transition-based Generalized Büchi Automata, $\mathcal{F} \subseteq 2^\delta$, π is accepted iff $\forall F \in \mathcal{F}$, $\inf(\pi) \cap F \neq \emptyset$, and C is accepting iff $\forall F \in \mathcal{F}$, $C \cap F \neq \emptyset$.*

In all cases the language $\mathscr{L}(\mathcal{A})$ of an automaton is the set of words recognized by accepting runs of $\mathcal{A}$. A cycle is rejecting iff it is not accepting.

***DBA**, **DGBA**, **DTBA**, and **DTGBA**, denote the restrictions of the above types to automata where the associated LTS S is deterministic.*

As an example, Fig. 1 shows one deterministic and complete automaton of each type. For BA, the states in $\mathcal{F}$ (called accepting states), are traditionally denoted by double circles. For GBA, $\mathcal{F} = \{F_1, F_2, \ldots\}$ is a set of acceptance sets $F_i \subseteq Q$, and each of these F_i is pictured using colored dots on the side of the states it contains (a state may have multiple such dots if it belongs to multiple acceptance sets). For transition-based acceptance (TBA or TGBA), we put those colored dots on the transitions (of course, only one color is used for TBA).

Definition 4 (SCC). *A* strongly connected component *(SCC) of an automaton $\langle S, \mathcal{F} \rangle$ is a set of states that are pairwise connected in S, and that is maximal with respect to inclusion. An SCC is* accepting *if it contains an accepting cycle, otherwise, it is* rejecting*. An SCC with a single state and no cycle is called* trivial*.*

Definition 5 (Automaton Typeness). *An automaton $\mathcal{A} = \langle S, \mathcal{F} \rangle$ is said to be β-type if there exists an automaton $\mathcal{B} = \langle S, \mathcal{F}' \rangle$ of acceptance type β such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{B})$. In other words, $\mathcal{A}$ is β-type if we can change its acceptance set so that it accepts the same language with the same transition structure but with a different type of acceptance condition (the β one).*

For instance any BA $\langle S, \mathcal{F} \rangle$ is GBA-type, because it can be trivially represented as the GBA $\langle S, \{\mathcal{F}\} \rangle$. Any GBA $\langle S, \{F_1, \ldots, F_n\} \rangle$ is TGBA-type, since it is equivalent to the TGBA $\langle S, \{F'_1, \ldots, F'_n\} \rangle$ with $F_i = \{(s, \ell, d) \mid (s, \ell, d) \in \delta \wedge s \in F_i\}$ (i.e., the membership of any state to some accepting set F_i is transferred to all its outgoing transitions). Similarly all BA are TBA-type, and all TBA are TGBA-type. These trivial conversions can sometime be used in the opposite direction. For instance a TGBA is "trivially" GBA-type if all the outgoing transitions of each state belong to the same acceptance sets.

Since a TGBA can represent any considered type (BA, GBA, TBA) using the same transition structure, it is often practical to consider only TGBAs and treat the other types as particular cases. For algorithms that cannot deal with generalized acceptance, a TGBA can be *degeneralized* into a TBA or a BA [2]. For instance the TBA of Fig. 1(c) was degeneralized from the TGBA of Fig. 1(d).

Definition 6 (Realizability). *A property (or an automaton) is β-realizable if there exists an automaton with β acceptance that accepts the same language.*

Any Büchi-like automaton is β-realizable for $\beta \in \{\mathrm{BA}, \mathrm{GBA}, \mathrm{TBA}, \mathrm{TGBA}\}$. Furthermore any deterministic Büchi-like automaton is $D\beta$-realizable. However it is well known than not all BAs are DBA-realizable, and this fact holds for the three other types of Büchi acceptance as well.

Definition 7 (Weakness). *An automaton is* inherently weak *if it contains no accepting cycle that has a state in common with a rejecting cycle.*

This notion of *inherent weakness* [5] generalizes the more common notion of weak automaton [4, 6] where states of an SCC should be either all accepting or all non-accepting. Defining $\mathcal{F}$ as the set of states that belong to some accepting cycle will turn any inherently weak automaton into a Weak BA (WBA) [5].

Definition 8 (Minimal-state automaton [17]). *For a language $L \subseteq \Sigma^\omega$, let $\approx_L \subseteq \Sigma^* \times \Sigma^*$ be the right-congruence defined by $u \approx_L v$ iff $u\alpha \in L \iff v\alpha \in L$ for all $\alpha \in \Sigma^\omega$. Let $[\cdot]$ denote the equivalence classes of L, i.e., $[u] = \{v \in \Sigma^* \mid u \approx_L v\}$. The minimal-state automaton of L, denoted $MSA(L)$ is the (complete and deterministic) LTS $\langle \mathrm{AP}, \{[u] \mid u \in \Sigma^*\}, [\varepsilon], \delta \rangle$ where $\delta([u], \ell) = [u\ell]$.*

3 Existing Determinization and Minimization Procedures

We now discuss several existing procedures to determinize and minimize Büchi automata. Some of these apply only to specific subclasses of temporal properties.

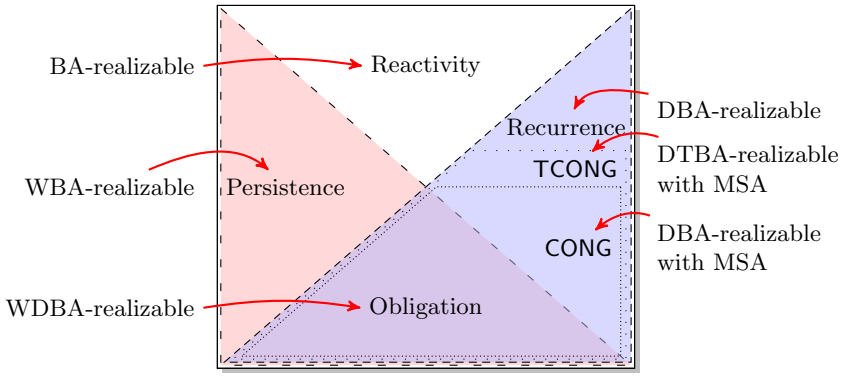


Fig. 2: Hierarchy of temporal properties in relation with subclasses of Büchi BA.

Figure 2 shows a classification of temporal properties, using the names from Manna and Pnueli [18], and with an additional couple of interesting subclasses. The whole square represents the entire set of BA-realizable properties, also called *reactivity* properties. This set includes *persistence* properties, which can be realized by weak automata, and *recurrence* properties ,which can be realized by deterministic automata. The intersection of these last two sets defines the *obligation* properties, which can be realized by weak and deterministic automata. The *obligation* class includes the well-known subclasses of *safety* and *guarantee* properties (not depicted here), as well as all their Boolean combinations.

Determinization. Assume that a recurrence property φ is represented by a possibly non-deterministic BA $A_\varphi = \langle T, \mathcal{F} \rangle$, and for which we wish to obtain a DBA $D_\varphi = \langle T', \mathcal{F}' \rangle$. We can consider different procedures depending on the class to which φ belongs.

If φ is known to be an obligation (for most LTL formulae, this can be checked syntactically [6]), then a weak DBA (WDBA) D_φ can be obtained from A_φ in a quite efficient way [9]: T should be determinized by the classical powerset construction to create $T' = \mathcal{P}(T)$, and the set $\mathcal{F}'$ can be computed by selecting one word of each SCC of T' and checking whether it is accepted by A_φ. Dax et al. [9] additionally show that if it is not known whether a property φ is an obligation, this procedure may still be applied but its result D_φ must then be checked for equivalence with A_φ.

The class CONG is defined as the set of languages L which are realizable by a BA automaton whose LTS is $MSA(L)$, this includes all obligations as well as some properties that are not expressible by weak automata. For instance the property specified by the formula $\mathsf{G}(a \rightarrow \mathsf{X}\neg a) \wedge \mathsf{GF}(a)$ is in CONG: it has three equivalence classes $[\varepsilon]$, $[a]$, and $[aa]$, and the corresponding MSA can be used to define the DBA shown in Fig. 3(a).

The formula $\mathsf{GF}a$ is not in CONG because its MSA has a single state (the language has no distinguishing prefixes), yet there is no single-state DBA that can recognize $\mathsf{GF}a$. However there exists a 1-state DTBA for this property, shown in Fig. 3(b). We could therefore define the class TCONG of properties that are

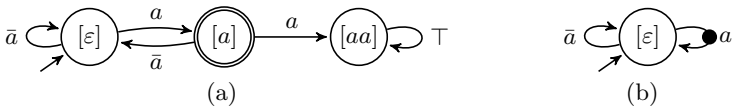


Fig. 3: (a) $\mathsf{G}(a \to \mathsf{X}\neg a) \wedge \mathsf{GF}a$ specifies a property in CONG: it is realizable by a DBA whose LTS is the MSA of its language. (b) Although $\mathsf{GF}a$ is not in CONG, its MSA can serve as a support for an equivalent DTBA.

DTBA-realizable with their MSA as LTS. Note that $\mathsf{GF}a \wedge \mathsf{GF}b$ whose MSA also has a single state, is not in TCONG: its minimal DTBA is shown in Fig. 1(c).

Dax et al. [9] show that you can apply the classical powerset construction to determinize the LTS of any BA $A_\varphi = \langle T, \mathcal{F}\rangle$ recognizing a property in CONG, and the resulting deterministic LTS $T' = \mathcal{P}(T)$ can be labeled as a DBA $D_\varphi = \langle T', \mathcal{F}'\rangle$ recognizing $\mathscr{L}(A_\varphi)$. The algorithm used to obtain the necessary $\mathcal{F}'$ is more complex than the one used for weak automata as it requires an enumeration of all the elementary cycles in T'. Futhermore, the DBAs obtained this way are not necessary minimal. This construction can be extended straightforwardly to transition-based acceptance; this way we may obtain DTBAs for properties (such as $\mathsf{GF}a$) where the original state-based procedure would not deliver any DBA.

Another determinization procedure, which applies to all recurrence properties, is to use Safra's construction [20] to convert a BA into a deterministic Rabin automaton (DRA). More generally, Schewe and Varghese [22] have recently proposed a determinization algorithm that inputs a TGBA and outputs a DRA. DRA that express recurrence properties are all DBA-type, and Krishnan et al. [15] show how to compute the associated DBAs.

Unfortunately, all these determinization procedures produce only DBA (or DTBA). To our knowledge, there is no determinization procedure that would produce a deterministic automaton with generalized acceptance.

Minimization. Minimizing a DBA, i.e., building a DBA such that no smaller equivalent DBA exist, is in general, an NP-complete problem [21].

However, weak DBAs (i.e., obligation properties) can be minimized using an algorithm identical to those performed on deterministic finite automata: the only requirement is to choose the acceptance of trivial SCCs correctly [16].

For the more general recurrence properties, Ehlers [11] gives an encoding of an equivalent minimal DBA as the solution of a SAT problem. His technique can also be extended straightforwardly to minimize a DTBA.

Until now there were no minimization procedure for generalized Büchi automata. In the next section we show an encoding inspired by Ehlers' [11] for minimizing DTGBAs. For any DTGBA A with n states and m acceptance sets, our encoding attempts to synthesize an equivalent DTGBA with n' states and m' acceptance sets for some given n' or m' (and possibly $m' > m$). By combining this procedure with existing (non-generalized) determinizations, we effectively obtain a minimal DTGBA with multiple acceptance sets, even though we do not know of any determinization procedure that supports generalized acceptance.

4 Synthesis of Equivalent DTGBA

Given a complete DTGBA $R = \langle\langle AP, Q_R, \iota_R, \delta_R\rangle, \mathcal{F}_R\rangle$, and two integers n and m, we would like to construct (when it exists) a complete DTGBA $C = \langle\langle AP, Q_C, \iota_C, \delta_C\rangle, \mathcal{F}_C\rangle$ such that $\mathscr{L}(R) = \mathscr{L}(C)$, $|Q_C| = n$ and $|\mathcal{F}_C| = m$. We call R the *reference* automaton, and C, the *candidate* automaton.

Since C and R are complete and deterministic, any word of Σ^ω has a unique run in R and C, and testing $\mathscr{L}(R) = \mathscr{L}(C)$ can be done by ensuring that each word is accepted by R iff it is accepted by C. In practice, this is checked by ensuring that any cycle of the synchronous product $C \otimes R$ corresponds to cycles that are either accepting in C and R, or rejecting in both.

This observation is used by Ehlers [11] for his SAT encoding. In his case, a cycle in a DBA can be detected as accepting as soon as it visits an accepting state. We use a similar encoding for our setup. However, because of the generalized acceptance condition of DTGBAs, we have to keep track of all the acceptance sets visited by paths of the product to ensure that cycles have the same acceptance status in both C and R.

4.1 Encoding as a SAT Problem

Let $SCC_R \subseteq 2^{Q_R}$ denote the set of non-trivial strongly connected components of R, and let $Acc(q'_1, \ell, q'_2) = \{A \in \mathcal{F}_R \mid (q'_1, \ell, q'_2) \in A\}$ be the set of acceptance sets of R that contain the transition $(q'_1, \ell, q'_2) \in \delta_R$.

We encode C with two sets of variables:

- The variables $\{\langle q_1, \ell, q_2\rangle_{\delta_C} \mid (q_1, q_2) \in Q_C^2, \ell \in \Sigma\}$ encode the existence of transitions $(q_1, \ell, q_2) \in \delta_C$ in the candidate automaton.
- Variables $\{\langle q_1, \ell, A_i, q_2\rangle_{\mathcal{F}_C} \mid (q_1, q_2) \in Q_C^2, \ell \in \Sigma, A_i \in \mathcal{F}_C\}$ encode the membership of these transitions to the acceptance set $A_i \in \mathcal{F}_C$ of C.

For the product $C \otimes R$, we retain the reachable states, and parts of paths that might eventually be completed to become cycles.

- A variable in $\{\langle q, q', q, q', \emptyset, \emptyset\rangle \mid q \in Q_C, q' \in Q_R\}$ encodes the existence of a reachable state (q, q') in $C \otimes R$. The reason we use a sextuplet to encode such a pair is that each (q, q') will serve as a starting point for possible paths.
- A variable in $\{\langle q_1, q'_1, q_2, q'_2, F, F'\rangle \mid (q_1, q_2) \in Q_C^2, S \in SCC_R, (q'_1, q'_2) \in S^2, F \subseteq \mathcal{F}_C, F' \subseteq \mathcal{F}_R\}$ denotes that there is a path between (q_1, q'_1) and (q_2, q'_2) in the product, such that its projection on the candidate automaton visits the acceptance sets $F \subseteq \mathcal{F}_C$, and its projection on the reference automaton visits the acceptance sets $F' \subseteq \mathcal{F}_R$. This set of variables is used to implement the cycle equivalence check, so the only q'_1 and q'_2 that need to be considered should belong to the same non-trivial SCC of R.

With these variables, the automaton C can be obtained as a solution of the following SAT problem. First, C should be complete (i.e., δ_C is total):

$$\bigwedge_{q_1 \in Q_C,\, \ell \in \Sigma} \bigvee_{q_2 \in Q_C} \langle q_1, \ell, q_2\rangle_{\delta_C} \tag{1}$$

The initial state of the product exists. Furthermore, if (q_1, q'_1) is a state of the product, $(q'_1, \ell, q'_2) \in \delta_R$ is a transition in the reference automaton, and

$(q_1, \ell, q_2) \in \delta_C$ is a transition in the candidate automaton, then (q_2, q'_2) is a state of the product too:

$$\wedge\langle \iota_C, \iota_R, \iota_C, \iota_R, \emptyset, \emptyset\rangle \wedge \bigwedge_{\substack{(q_1,q_2)\in Q_C^2,\, q'_1 \in Q_R,\\ (\ell,q'_2)\in\delta_R(q'_1)}} \langle q_1, q'_1, q_1, q'_1, \emptyset, \emptyset\rangle \wedge \langle q_1, \ell, q_2\rangle_{\delta_C} \rightarrow \langle q_2, q'_2, q_2, q'_2, \emptyset, \emptyset\rangle \quad (2)$$

Any transition of the product augments an existing path, updating the sets F and F' of acceptance sets visited in each automata. Unfortunately, we have to consider all possible subsets $G \in 2^{\mathcal{F}_C}$ of acceptances sets to which the candidate transition (q_2, ℓ, q_3) could belong, and emit a different rule for each possible G.

$$\wedge \bigwedge_{\substack{(q_1,q_2,q_3)\in Q_C^3,\\ (F,G)\in(2^{\mathcal{F}_C})^2,\\ S\in SCC_R,\,(q'_1,q'_2)\in S^2,\\ F'\in 2^{\mathcal{F}_R},\,(\ell,q'_3)\in\delta_R(q'_2)}} \left(\begin{array}{l} \langle q_1, q'_1, q_2, q'_2, F, F'\rangle \\ \wedge\langle q_2, \ell, q_3\rangle_{\delta_C} \\ \wedge \bigwedge_{A\in G} \langle q_2, \ell, A, q_3\rangle_{\mathcal{F}_C} \\ \wedge \bigwedge_{A\notin G} \neg\langle q_2, \ell, A, q_3\rangle_{\mathcal{F}_C} \end{array} \right) \rightarrow \begin{array}{l} \langle q_1, q'_1, q_3, q'_3, F \cup G, \\ F' \cup Acc(q'_2, \ell, q'_3)\rangle \end{array} \quad (3)$$

If a path of the product is followed by a transition $(q'_2, \ell, q'_3) \in \delta_R$ and a transition $(q_2, \ell, q_3) \in \delta_C$ that both closes the cycle ($q_3 = q_1 \wedge q'_3 = q'_1$), but such that this cycle is non-accepting with respect to the reference automaton ($Acc(q'_2, \ell, q'_3) \cup F' \neq \mathcal{F}_R$), then the cycle formed in the candidate automaton by (q_2, ℓ, q_1) should not be accepting (at least one acceptance set of $\mathcal{F}_C \setminus F$ is missing):

$$\wedge \bigwedge_{\substack{(q_1,q_2)\in Q_C^2,\, F\in 2^{\mathcal{F}_C},\\ S\in SCC_R,\,(q'_1,q'_2)\in S^2,\, F'\in 2^{\mathcal{F}_R},\\ (\ell,q'_3)\in\delta_R(q'_2),\, q'_3=q'_1,\\ Acc(q'_2,\ell,q'_3)\cup F'\neq\mathcal{F}_R}} \langle q_1, q'_1, q_2, q'_2, F, F'\rangle \wedge \langle q_2, \ell, q_1\rangle_{\delta_C} \rightarrow \neg \bigwedge_{A\in\mathcal{F}_C\setminus F} \langle q_2, \ell, A, q_1\rangle_{\mathcal{F}_C} \quad (4)$$

Conversely, if a path of the product is followed by a transition $(q'_2, \ell, q'_3) \in \delta_R$ and a transition $(q_2, \ell, q_3) \in \delta_C$ that both closes the cycle ($q_3 = q_1 \wedge q'_3 = q'_1$), but such that this cycle is accepting with respect to the reference automaton ($Acc(q'_2, \ell, q'_3) \cup F' = \mathcal{F}_R$), then the cycle formed in the candidate automaton by (q_2, ℓ, q_1) should also be accepting ((q_2, ℓ, q_1) should belong at least to all missing acceptance sets $\mathcal{F}_C \setminus F$):

$$\wedge \bigwedge_{\substack{(q_1,q_2)\in Q_C^2,\, F\in 2^{\mathcal{F}_C},\\ S\in SCC_R,\,(q'_1,q'_2)\in S^2,\, F'\in 2^{\mathcal{F}_R},\\ (\ell,q'_3)\in\delta_R(q'_2),\, q'_3=q'_1\\ Acc(q'_2,\ell,q'_3)\cup F'=\mathcal{F}_R}} \langle q_1, q'_1, q_2, q'_2, F, F'\rangle \wedge \langle q_2, \ell, q_1\rangle_{\delta_C} \rightarrow \bigwedge_{A\in\mathcal{F}_C\setminus F} \langle q_2, \ell, A, q_1\rangle_{\mathcal{F}_C} \quad (5)$$

Optimizations. As suggested by Ehlers [11], the set of possible solutions can be optionally constrained with some partial symmetry breaking clauses. Assuming $Q_C = \{q_1, ..., q_n\}$ and $\Sigma = \{\ell_0, \ldots, \ell_{m-1}\}$ we can order the transitions of the solution lexicographically with respect to their source and label, as (q_1, ℓ_0, d_1), (q_1, ℓ_1, d_2), $\ldots$, (q_1, ℓ_{m-1}, d_m), (q_2, ℓ_0, d_{m+1}), etc., and constrain the solution so that the destination of the first transition should be chosen between $d_1 \in \{q_1, q_2\}$, the destination of the second between $d_2 \in \{q_1, q_2, q_3\}$, etc.

$$\wedge \bigwedge_{1\leq i\leq|Q|,\, 0\leq j<|\Sigma|,\, (i-1)|\Sigma|+j+3\leq k\leq n} \neg\langle q_i, \ell_j, q_k\rangle_{\delta_C} \quad (S)$$

ReduceStatesDTGBA($R, m = R$.nb_acc_sets()):
  repeat:
    $n \leftarrow R$.nb_states()
    $C \leftarrow$ SynthesizeDTGBA$(R, n-1, m)$
    if C **does not exists**: **return** R
    $R \leftarrow C$

Fig. 4: Given a complete DTGBA A, attempt to build an equivalent smaller one with the same number of acceptance sets.

Equations (3)–(5) can be encoded more efficiently when q'_1 and q'_2 belong to a weak SCC. In that case it is not necessary to remember the history F' of the acceptance sets seen by paths in that SCC, since all paths are either accepting or rejecting. Variables of the form $\langle q_1, q'_1, q_2, q'_2, F, F' \rangle$ where $F' \in 2^{\mathcal{F}_R}$ can hence be restricted to just $F' = \emptyset$, limiting the number of variables and clauses emitted.

State-based Output. To produce automata with state-based acceptance, it suffices to consider all variables $\langle q, \ell, A_i, q' \rangle_{\mathcal{F}_C}$ that share the same q and A_i as aliases. This way, the DTGBA output is DGBA-type.

4.2 Usage

We call SynthesizeDTGBA(R, n, m) the procedure that:

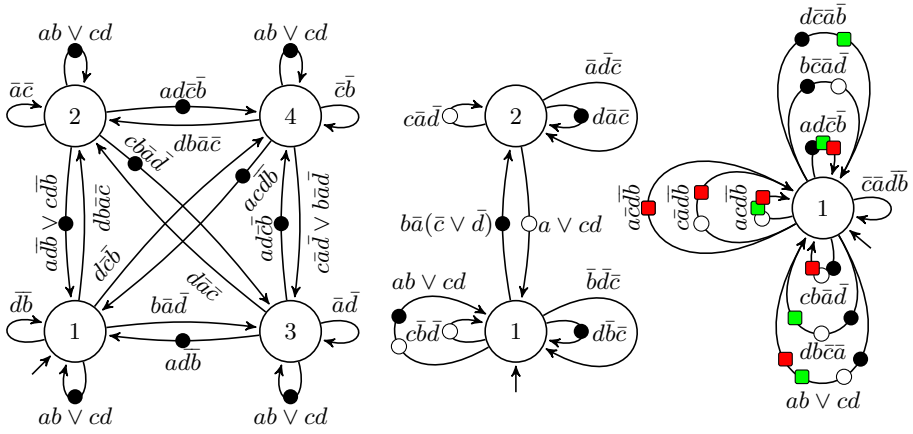
1. inputs a complete DTGBA R, two integers $n = |Q_C|$ and $m = |\mathcal{F}_C|$,
2. produces a DIMACS file with all the above clauses,
3. calls a SAT solver to solve this problem,
4. builds the resulting DTGBA C if it exists. The construction of this automaton depends only on the value of variables $\{\langle q, \ell, q' \rangle_{\delta_C} \mid (q, q') \in Q_C^2, \ell \in \Sigma\} \cup \{\langle q, \ell, A_i, q' \rangle_{\mathcal{F}_C} \mid (q, q') \in Q_C^2, \ell \in \Sigma, A_i \in \mathcal{F}_C\}$.

Although equation (1) constraints C to be complete, there is no explicit constraint for C to be deterministic. By construction, any word w is accepted in R iff it is accepted by any run that recognizes w in C. In the presence of two transitions $\langle q, \ell, d_1 \rangle_{\delta_C}$ and $\langle q, \ell, d_2 \rangle_{\delta_C}$ when building C, we can safely ignore one of them to ensure determinism.

Minimization: $|\mathcal{F}_C| = |\mathcal{F}_R|$. Given a complete DTGBA $R = \langle \langle AP, Q_R, \iota_R, \delta_R \rangle, \mathcal{F}_R \rangle$, we may use SynthesizeDTGBA in a loop such as the one shown in Fig. 4 to reduce the number of states of the automaton. By default, we keep the same number $m = |\mathcal{F}_R|$ of acceptance sets for the candidate automaton.

Minimization with Generalization: $|\mathcal{F}_C| > |\mathcal{F}_R|$. A greater reduction might be obtained by increasing the number $m = |\mathcal{F}_C|$ of acceptance sets. This can be interpreted as the converse of a degeneralization: instead of augmenting the number of states to reduce the number of acceptance sets, we augment the number of acceptance sets in an attempt to reduce the number of states.

It is however not clear how this increase of m should be done. Figure 5 shows an example of a property that can be expressed by a 4-state DTGBA if $m = 1$, a 2-state DTGBA if $m = 2$ and a 1-state DTGBA if $m = 4$. Interestingly, the



(a) $|\mathcal{F}| = 1$ (b) $|\mathcal{F}| = 2$ (c) $|\mathcal{F}| = 4$

Fig. 5: Examples of minimal DTGBA recognizing $(\mathsf{GF}a \wedge \mathsf{GF}b) \vee (\mathsf{GF}c \wedge \mathsf{GF}d)$.

smallest DTGBA for $m = 3$ also has two states. Consequently an algorithm that increments m as long as it reduces the number of states would never reach $m = 4$.

We leave open the problem of finding the smallest m such that no smaller equivalent DTGBA with a larger m can be found. Instead, we use the following heuristic: let m be the number of acceptance set that were "naturally" used to translate the formula φ into a TGBA A_φ (before any degeneralization or determinization). In the case of $(\mathsf{GF}a \wedge \mathsf{GF}b) \vee (\mathsf{GF}c \wedge \mathsf{GF}d)$, each F operator will require one acceptance set during the translation [10], so we would use $m = 4$.

5 Implementation and Experiments

5.1 Implemented Tools

Figure 6 gives an overview of the processing chains we implemented to produce a minimal DTGBA or DTBA. With the exception of `ltl2dstar` 0.5.1, a tool written by Joachim Klein [14] to convert LTL formulae into deterministic Rabin or Streett automata, all the other white boxes correspond to algorithms that have been implemented in Spot 1.2, and have been integrated in two command-line tools: `ltl2tgba` at the top of the picture, and `dstar2tgba` at the bottom.

`ltl2tgba` takes as input an LTL formula and translates it into a TGBA $A = \langle S, \mathcal{F} \rangle$ [10]. If the desired number of acceptance sets m was not supplied on the command-line, we set $m \leftarrow |\mathcal{F}|$ right after this step. We now attempt WDBA minimization [9], if that succeeded, we output a minimal weak DBA (looking for transition-based or generalized acceptance will not reduce it further). Otherwise, we simplify the TGBA using simulation-based reductions [2]. If the resulting TGBA has more than one acceptance set but we plan to build a DTBA (i.e. $m = 1$), we degeneralize it. If the resulting TBA is nondeterministic, we attempt

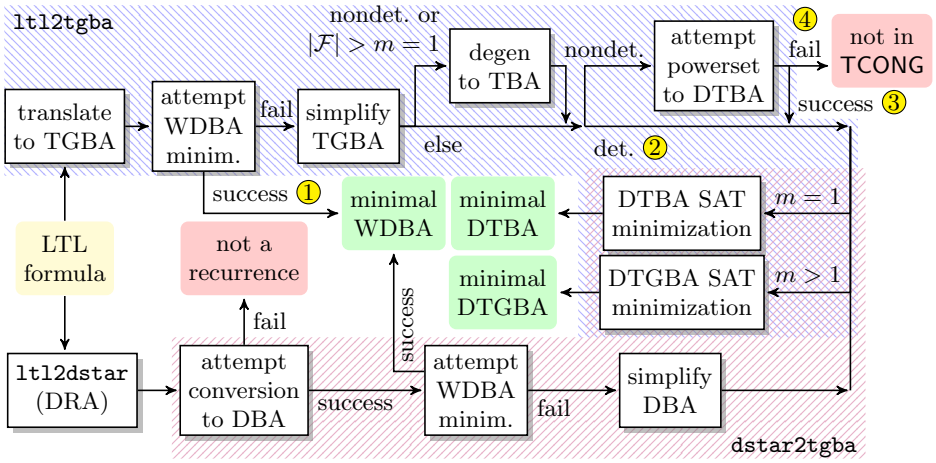


Fig. 6: The two tool chains we use to convert an LTL formula into a DTBA or a DTGBA. $|\mathcal{F}|$ is the number of acceptance condition of the current automaton, while m is the desired number of acceptance sets.

the transition-based variant of Dax et al. [9]'s powerset construction followed by a cycle-enumeration to decide the acceptance of each transition. This procedure works only on a TBA, hence the previous degeneralization is also performed on any nondeterministic TGBA. If this powerset construction fails to produce an equivalent DTBA, it means the input property is not in TCONG. Otherwise the deterministic automaton is sent for SAT-based minimization. We have two procedures implemented: "DTGBA SAT" is the encoding described in Sec. 4.1, while "DTBA SAT" is an adaptation of Ehlers' encoding [11] to transition-based Büchi acceptance and is more-efficient to use when $m = 1$ (see Annex A). Not apparent on this picture, is that these two algorithms can be configured to output automata that are DBA-type or DGBA-type, as discussed in the last paragraph of Sec. 4.1. To solve SAT problems, we use `glucose` 2.3 [1], which is ranked first in the hard-combinatorial benchmark of the 2013 SAT Competition.

If the automaton could not be determinized by `ltl2tgba`, the second approach with `dstar2tgba` is used instead. This approach starts with `ltl2dstar` to convert the input LTL formula into a deterministic Rabin automaton (DRA). The produced DRA is then converted to a DBA [15] if such a DBA exists, otherwise it means that the formula is not a recurrence. The resulting DBA is turned into a minimal WDBA if it expresses an obligation [9], otherwise it is simplified using simulation-based reduction, before finally proceeding to SAT-based minimization.

5.2 Experiments

To assert the effectiveness of our minimization procedures, we took LTL formulas expressing recurrence properties, and attempted to build a minimal DBA, DTBA, and DTGBA for each of them. The LTL formulae are all those listed by Ehlers

[11], and those of Esparza et al. [12] that are DBA-realizable. We also added a few formulas of our own.

For comparison, we also ran Ehlers' DBAminimizer (dated November 2011) [11]: a tool that takes deterministic Rabin automata, as produced by `ltl2dstar`, and converts them into their minimal DBA by solving a SAT problem with PicoSat (version 957). This tool is named "DBAm..zer" in our tables.

We executed our benchmark on a machine with four Intel Xeon E7-2860 processors and 512GB of memory. We limited each minimization procedure (the entire process of taking an LTL formula or a Rabin automaton into a minimal DBA, including writing the problem to disk, not just the SAT-solving procedure) to 2 hours, and each process to 40GB of memory.

For presentation, we partitioned our set of formulae into four classes, based on which path they would take in `ltl2tgba`, denoted by the ⓧ marks in Fig. 6.

① Formulae that correspond to obligations properties. On these formulas, the powerset construction can be used to obtain a DBA [9], and this DBA can be minimized in polynomial time [16]. Using a SAT-based minimization in this case would be unreasonable.

② Formulae for which `ltl2tgba` translator naturally produces deterministic automata, without requiring a powerset construction.

③ Formulae that `ltl2tgba` can successfully convert into a DTBA by applying a powerset construction.

④ Formulae that `ltl2tgba` would fail to convert into a DTBA, and for which `dstar2tgba` is needed.

For size reason, we only present an excerpt of the results for the most interesting classes: ④ (Table 1) and ③ (Table 2).[4]

For each formula we show the size of the DRA constructed by `ltl2dstar`, the size of the DTBA or DBA built by `dstar2tgba` before SAT-minimization, and the size of the minimal DBA, DTBA or DTGBA after SAT-minimization. In the case of DTGBA the desired number of acceptance sets m is shown in the second column (this m was computed by running only the translation algorithm of `ltl2tgba`, and picking $m \leftarrow |\mathcal{F}|$ on the result).

Column "C." indicates whether the produced automata are complete: rejecting sink states are always omitted from the sizes we display, and this explains differences when comparing our tables with other papers that measure complete automata. When $C = 0$, a complete automaton would have one more state.

Cases that took more than 2 hours or 40GB are shown as "(killed)"; an additional "$\leq n$" gives the number of states of the smallest automaton successfully computed. SAT problems requiring more than 2^{31} clauses, the maximum supported by state-of-the-art solvers, are shown as "(intmax)". Cases where the output of the SAT minimization is smaller than the input are shown with a light gray background. Bold numbers show cases where the minimal DTBA is smaller than the minimal DBA (transition-based acceptance was useful) and cases where the minimal DTGBA is smaller than the minimal DTBA (generalized acceptance

[4] Detailed results and instructions to reproduce the benchmark can be found at `http://www.lrde.epita.fr/~adl/forte14/`.

Table 1: Formulae that we do not know how to determinize other than via DRA.

	m	C.	DRA $\|Q\|$	DBA $\|Q\|$	DBAm..zer $\|Q\|$	DBAm..zer time	minDBA $\|Q\|$	minDBA time	minDTBA $\|Q\|$	minDTBA time	minDTGBA $\|Q\|$	minDTGBA $\|\mathcal{F}\|$	minDTGBA time
$(\bar{a} \wedge \mathsf{F}a)\,\mathsf{R}(\bar{b}\,\mathsf{R}\,\mathsf{XF}c)$	2	1	60	24	(killed)		11	1585	**7**	14	7	1	815
$\bar{a} \wedge ((\mathsf{F}b\,\mathsf{U}\,a)\,\mathsf{W}\,\mathsf{X}c)$	2	0	31	20	(killed)		(killed≤15)		(killed≤14)		(killed≤13)		
$(a\,\mathsf{R}(b\,\mathsf{R}\,\mathsf{F}c))\,\mathsf{W}\,\mathsf{XG}b$	1	1	31	24	(killed)		(killed≤11)		**8**	188	8	1	190
$(a\,\mathsf{R}\,\mathsf{F}b)\,\mathsf{U}\,\mathsf{X}\bar{c}$	2	1	19	13	11	1356	11	2405	**10**	55	10	1	182
$\mathsf{F}(a \wedge \mathsf{F}b)\,\mathsf{W}\,\mathsf{X}c$	2	1	25	16	(killed)		(killed≤12)		**11**	1445	**10**	**2**	364
$(\mathsf{F}\bar{a}\,\mathsf{R}\,\mathsf{F}\bar{b})\,\mathsf{W}\,\mathsf{G}\bar{c}$	2	1	18	15	5	59	5	14	**4**	12	4	1	444
$((\mathsf{F}a\,\mathsf{U}\,b)\,\mathsf{R}\,\mathsf{F}c)\,\mathsf{W}\,\mathsf{X}\bar{c}$	3	1	47	32	11	1216	11	1792	**9**	55	(intmax)		
$\mathsf{G}((a \wedge b) \vee \mathsf{G}a \vee \mathsf{F}(\bar{c} \wedge \mathsf{XX}c))$	1	1	49	27	(killed)		9	577	**7**	307	7	1	307
$\mathsf{G}a\,\mathsf{R}(\mathsf{F}\bar{b} \wedge (c\,\mathsf{U}\,b))$	2	0	10	9	8	13	8	27	**6**	1	**5**	**2**	36
$\mathsf{G}(\mathsf{F}(\bar{a} \wedge \mathsf{F}a)\,\mathsf{U}(b\,\mathsf{U}\,\mathsf{X}c))$	3	1	52	35	(killed)		10	1304	**7**	981	(intmax)		
$\mathsf{GF}(a \wedge \mathsf{F}(b \wedge \mathsf{F}c))$	3	1	10	10	4	1	4	1	**3**	1	**1**	**3**	156
$\mathsf{G}(\mathsf{F}\bar{a} \vee (\mathsf{F}b\,\mathsf{U}\,c))$	3	1	29	23	4	1368	4	77	**3**	76	(intmax)		
$\mathsf{G}(\mathsf{F}\bar{a}\,\mathsf{U}\,\mathsf{X}(\mathsf{F}\bar{b} \wedge \mathsf{X}b))$	3	1	58	37	(killed)		4	480	**3**	512	(intmax)		
$\mathsf{GF}(a \wedge \mathsf{XXXF}b)$	2	1	66	6	66	4927	3	0	**2**	0	**1**	**2**	1
$\mathsf{G}(\mathsf{G}\bar{a} \vee ((\mathsf{F}b\,\mathsf{U}\,c)\,\mathsf{U}\,a))$	3	1	20	18	10	174	10	191	**9**	59	18	1	3183
$\mathsf{X}(\bar{a} \wedge \mathsf{F}a)\,\mathsf{R}(a\,\mathsf{M}\,\mathsf{F}b)$	2	1	11	11	9	3	9	2	9	2	9	1	10
$\mathsf{X}(\bar{a} \vee \mathsf{G}(a \wedge \bar{b}))\,\mathsf{R}\,\mathsf{F}(c \wedge \mathsf{F}b)$	2	1	29	18	11	1467	11	1905	**10**	46	**9**	**2**	667
$\mathsf{XF}\bar{a}\,\mathsf{R}\,\mathsf{F}(b \vee (\bar{a} \wedge \mathsf{F}\bar{c}))$	3	1	17	12	7	0	7	1	**6**	0	6	1	101
$\mathsf{X}((\mathsf{F}a \wedge \mathsf{XF}b)\,\mathsf{R}\,\mathsf{XF}c)$	3	1	54	34	(killed)		(killed≤13)		**10**	149	(intmax)		

Table 2: Formulae determinized via the TBA-variant of the procedure of Dax et al. [9].

	m	C.	DRA $\|Q\|$	DTBA $\|Q\|$	DBA $\|Q\|$	DBAm..zer $\|Q\|$	DBAm..zer time	minDBA $\|Q\|$	minDBA time	minDTBA $\|Q\|$	minDTBA time	minDTGBA $\|Q\|$	minDTGBA $\|\mathcal{F}\|$	minDTGBA time
$(a\,\mathsf{U}\,\mathsf{X}\bar{a}) \vee$ $\mathsf{XG}(\bar{b} \wedge \mathsf{XF}c)$	2	0	10	10	12			8	3.9	**6**	0.5	6	1	3.6
				8	8	8	4.2	8	2.3	**6**	0.1	6	1	1.7
$\mathsf{F}(a \wedge \mathsf{GF}b) \vee (\mathsf{F}c \wedge$ $\mathsf{F}a \wedge \mathsf{F}(c \wedge \mathsf{GF}\bar{b}))$	5	1	4	5				5	0.0	**4**	0.0	4	1	1.9
			7	6	6	5	0.1	5	0.0	**4**	0.0	4	1	68.9
$\mathsf{GF}(a \leftrightarrow \mathsf{XX}b)$	1	1	7	7	11			6	2.7	**4**	0.1	4	1	0.1
			9	9	9	6	1.2	6	1.8	**4**	0.2	4	1	0.2
$\mathsf{GF}(a \leftrightarrow \mathsf{XXX}b)$	1	1	17	15	23			(killed≤11)		(killed≤8)		(killed≤8)		
				17	17	(killed)		(killed≤11)		(killed≤8)		(killed≤8)		
$(\mathsf{GF}b \wedge \mathsf{GF}a) \vee$ $(\mathsf{GF}c \wedge \mathsf{GF}d)$	4	1	13	5	9			5	2.0	**4**	0.1	**1**	**4**	25.3
				8	8	5	2.2	5	1.5	**4**	1.3	**1**	**4**	406.0
$\mathsf{X}((a\,\mathsf{M}\,\mathsf{F}((b \wedge c) \vee$ $(\bar{b} \wedge \bar{c})))\,\mathsf{W}(\mathsf{G}\bar{c}\,\mathsf{U}\,b))$	3	1	29	14	18			(killed≤12)		**11**	3934.0	**10**	**3**	2010.5
				22	22	(killed)		(killed≤12)		**11**	4628.4	22	1	1617.2
$\mathsf{X}(a\,\mathsf{R}((\bar{b} \wedge \mathsf{F}\bar{c})\,\mathsf{M}\,\mathsf{X}\bar{a}))$	2	0	13	10	12			10	4188.5	**9**	52.7	**8**	**2**	13.0
				13	13	10	1449.4	10	3424.5	**9**	52.3	**8**	**2**	37.1
$\mathsf{X}(\mathsf{G}(\bar{a}\,\mathsf{M}\,\bar{b}) \vee$ $\mathsf{G}(a \vee \mathsf{G}\bar{a}))$	1	0	14	7	8			7	0.2	**6**	0.1	6	1	0.1
				13	13	7	1.3	7	1.1	**6**	0.4	6	1	0.4
$\mathsf{XXG}(\mathsf{F}a\,\mathsf{U}\,\mathsf{X}b)$	2	1	21	10	14			8	2.9	**6**	1.1	**5**	**2**	17.5
				18	18	8	80.6	8	11.3	**6**	12.2	**5**	**2**	337.0

was useful). Dark gray cells show cases where the tool returned an automaton that was not minimal. For `dstar2tgba`, it is because `glucose` answers `INDETERMINATE` when it is not able to solve the problem: `dstar2tgba` then pessimistically assumes that the problem is unsatisfiable and returns the input automaton.

The couple of cases where we are able to produce a minimal DBA, but DBAminimizer failed, are because we apply more efficient simplification routines on the DBA before it is passed to the SAT-minimization, our encoding takes advantage of SCCs in the reference automaton, and we use a different SAT solver. The cases where we fail to output minimal DBA, but successfully output a DTBA (or even a DTGBA) are due to the fact that DTBA and DTGBA will produce smaller automata, so the last iteration of the algorithm of Fig. 4, the one where the problem is UNSAT (often the more time-consuming problem), is applied to a smaller automaton. Finally bold numbers of the minDTGBA column confirm that using generalized acceptance can actually reduce the size of a DBA or DTBA. Note that useless acceptance sets have been removed from all DTGBA produced [2], so $|\mathcal{F}|$ might be smaller than m.

Table 2 shows an excerpt of class ③: formulae that can be determinized by the powerset construction of Dax et al. [9]. As this construction enumerates all cycles of the determinized automaton to fix its acceptance sets, it is potentially very long. We therefore compare our two approaches: each formula of the table has two result lines, the upper line corresponds to `ltl2tgba` (with the powerset construction), while the lower line shows results via `ltl2dstar` (with Safra's construction). In this table, the former approach is almost always the fastest.

These tables show that even if powerset and Safra's construction are only able to deal with a single acceptance set, so force us to degeneralize automata, we can successfully reconstruct minimal DTGBAs with generalized acceptance.

6 Conclusion and Future Work

Deterministic Büchi automata are mandatory in some applications like verification of Markov decision processes, probabilistic model checking, or synthesis of distributed systems from LTL specifications. In these contexts small automata are more than welcome. Furthermore, it is well known that transition-based and generalized acceptance contributes to the conciseness of automata. However, to our knowledge, there did not exist any algorithm to produce minimal and generalized deterministic Büchi automata. Furthermore, we do not know of any determinization algorithm that would build a generalized automaton.

In this paper, we have presented a complete framework with two complete processing chains for determinizing and minimizing transition-based generalized Büchi automata. Even though we construct non-generalized deterministic automata before minimizing them, our SAT-based minimization can be used to produce minimal DTGBA for a given number of acceptance sets m.

Our results show that this SAT-based technique is effective for medium-sized automata. For large automata, SAT-solving is expensive, but it could still be run with a time constraint to produce a smaller (but not necessarily minimal) DTGBA.

In the future we plan to improve our encoding by using more structural information about the reference automaton to reduce the number clauses and variables used, and, hopefully, deal with larger automata. We also need to investigate the problem, mentioned in Section 4.2, of selecting the "best" m for a given automaton, and the possibility to extend Dax et al.'s technique to a class of properties that are realizable by a DTGBA whose LTS is a MSA (this class would contain TCONG).

Other recent algorithms from the literature should also be considered for integration in our setup. For instance our TGBA simplification step of Fig. 6 could be improved using other simulation techniques [21, 7], and an implementation of determinization technique of [22] could also contribute to reducing the size of the automaton that has to be minimized.

References

1. G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI'09*, pp. 399–404, July 2009.
2. T. Babiak, T. Badie, A. Duret-Lutz, M. Křetínský, and J. Strejček. Compositional approach to suspension and other improvements to LTL translation. In *SPIN'13*, vol. 7976 of *LNCS*, pp. 81–98. Springer, July 2013.
3. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *CAV'99*, vol. 1633 of *LNCS*, pp. 222–235. Springer, 1999.
5. B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *IJCAR'01*, vol. 2083 of *LNCS*, pp. 611–625. Springer, 2001.
6. I. Černá and R. Pelánek. Relating hierarchy of temporal properties to model checking. In *MFCS'03*, vol. 2747 of *LNCS*, pp. 318–327, Aug. 2003. Springer.
7. L. Clemente and R. Mayr. Advanced automata minimization. In *POPL'13*, pp. 63–74. ACM, 2013.
8. J.-M. Couvreur, A. Duret-Lutz, and D. Poitrenaud. On-the-fly emptiness checks for generalized Büchi automata. In *SPIN'05*, vol. 3639 of *LNCS*, pp. 143–158. Springer, Aug. 2005.
9. C. Dax, J. Eisinger, and F. Klaedtke. Mechanizing the powerset construction for restricted classes of ω-automata. In *ATVA'07*, vol. 4762 of *LNCS*. Springer, 2007.
10. A. Duret-Lutz. LTL translation improvements in Spot. In *VECoS'11*, Sept. 2011. British Computer Society.
11. R. Ehlers. Minimising deterministic Büchi automata precisely using SAT solving. In *SAT'10*, vol. 6175 of *LNCS*, pp. 326–332. Springer, 2010.
12. J. Esparza, A. Gaiser, and J. Kretinsky. Rabinizer: Small deterministic automata for LTL(F,G). In *ATVA'12*, vol. 7561 of *LNCS*, pp. 95–109. Springer, 2012.
13. B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *LICS'05*, pp. 321–330, June 2005.
14. J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363:182–195, 2005.
15. S. C. Krishnan, A. Puri, and R. K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *ISAAC'94*, vol. 834 of *LNCS*, pp. 378–386. Springer, 1994.

16. C. Löding. Efficient minimization of deterministic weak ω-automata. *Information Processing Letters*, 79(3):105–109, 2001.
17. O. Maler and L. Staiger. On syntactic congruences for ω-languages. In *STACS'93*, vol. 665 of *LNCS*, pp. 586–594. Springer, 1993.
18. Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *PODC'90*, pp. 377–410, 1990. ACM.
19. K. Y. Rozier and M. Y. Vardi. A multi-encoding approach for LTL symbolic satisfiability checking. In *FM'11*, pp. 417–431. Springer, 2011.
20. S. Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, The Weizmann Institute of Science, Rehovot, Israel, Mar. 1989.
21. S. Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In *FSTTCS'10*, vol. 8 of *LIPIcs*, pp. 400–411, 2010. Schloss Dagstuhl LZI.
22. S. Schewe and T. Varghese. Tight bounds for the determinisation and complementation of generalised Büchi automata. In *ATVA'12*, vol. 7561 of *LNCS*, pp. 42–56. Springer, 2012.

A Synthesis of Equivalent DTBA

This section gives our transition-based adaptation of Ehlers' encoding [11]. After giving a straightforward adaptation of his encoding to deal with DTBA instead of DBA, we list some additional optimizations we applied in our implementation.

From a reference DTBA $R = \langle\langle \Sigma, Q_R, \iota_R, \delta_R\rangle, \mathcal{F}_R\rangle$, we give a set of constraints that ensures a candidate DTBA $C = \langle\langle \Sigma, Q_C, \iota_C, \delta_C\rangle, F_C\rangle$ is equivalent to R.

We encode C with two sets of variables:

- The variables $\{\langle q_1, \ell, q_2\rangle_{\delta_C} \mid (q_1, q_2) \in Q_C^2,\ \ell \in \Sigma\}$ encode the existence of transitions $(q_1, \ell, q_2) \in \delta_C$ in the candidate automaton.
- Variables $\{\langle q_1, \ell, q_2\rangle_{\mathcal{F}_C} \mid (q_1, q_2) \in Q_C^2,\ \ell \in \Sigma\}$ encode the membership of these transitions to the acceptance set $\mathcal{F}_C$.

For the product $C \otimes R$, we retain the reachable states, and parts of paths that might eventually be completed to become cycles.

- A variable in $\{\langle q, q'\rangle_G \mid q \in Q_C, q' \in Q_R\}$ encodes the existence of a reachable state (q, q') in $C \otimes R$.
- A variable in $\{\langle q_1, q_1', q_2, q_2'\rangle_C \mid (q_1, q_2) \in Q_C^2,\ (q_1', q_2') \in Q_R^2\}$ denotes that there is an acyclic path between (q_1, q_1') and (q_2, q_2') in the product, such that its projection on the candidate automaton C does not visit $\mathcal{F}_C$.
- A variable in $\{\langle q_1, q_1', q_2, q_2'\rangle_R \mid (q_1, q_2) \in Q_C^2,\ (q_1', q_2') \in Q_R^2\}$ denotes that there is an acyclic path between (q_1, q_1') and (q_2, q_2') in the product, such that its projection on the reference automaton R does not visit $\mathcal{F}_R$.

The problem is encoded as follows. The candidate automaton is complete:

$$\bigwedge_{q_1 \in Q_C,\, \ell \in \Sigma} \bigvee_{q_2 \in Q_C} \langle q_1, \ell, q_2\rangle_{\delta_C} \tag{6}$$

The initial state of the product exists. Furthermore if (q_1, q_1') is a state of the product, $(q_1', \ell, q_2') \in \delta_R$ is a transition in the reference automaton, and

$(q_1, \ell, q_2) \in \delta_C$ is transition in the candidate automaton, then (q_2, q_2') is a state of the product too:

$$\wedge \quad \langle \iota_C, \iota_R \rangle_G \quad \wedge \bigwedge_{\substack{(q_1,q_2)\in Q_C^2,\, q_1'\in Q_R \\ (\ell,q_2')\in\delta_R(q_1')}} \langle q_1, q_1' \rangle_G \wedge \langle q_1, \ell, q_2 \rangle_{\delta_C} \rightarrow \langle q_2, q_2' \rangle_G \tag{7}$$

Each state of the product corresponds to an empty path that is non-accepting in the reference and in the candidate automata:

$$\wedge \bigwedge_{q_1\in Q_C,\, q_1'\in Q_R} \langle q_1, q_1' \rangle_G \rightarrow (\langle q_1, q_1', q_1, q_1' \rangle_R \wedge \langle q_1, q_1', q_1, q_1' \rangle_C) \tag{8}$$

Otherwise when one of the two transitions does not close the cycle ($q_3 \neq q_1 \vee q_3' \neq q_1'$), then the non-accepting path is prolonged:

$$\wedge \bigwedge_{\substack{(q_1,q_2,q_3)\in Q_C^3,\, (q_1',q_2')\in Q_R^2 \\ (\ell,q_3')\in\delta_R(q_2'),\, (q_2',\ell,q_3')\notin F_R \\ (q_3'\neq q_1')\vee(q_3\neq q_1)}} \langle q_1, q_1', q_2, q_2' \rangle_R \wedge \langle q_2, \ell, q_3 \rangle_{\delta_C} \rightarrow \langle q_1, q_1', q_3, q_3' \rangle_R \tag{9}$$

If a path of the product that is non-accepting with respect to the **reference** automaton, is completed by a **non-accepting** transition (q_2', ℓ, q_3') and a transition (q_2, ℓ, q_3) that both closes the cycle ($q_3 = q_1 \wedge q_3' = q_1'$), then (q_2, ℓ, q_1) is also non-accepting:

$$\wedge \bigwedge_{\substack{(q_1,q_2)\in Q_C^2,\, (q_1',q_2')\in Q_R^2, \\ (\ell,q_3')\in\delta_R(q_2'),\, (q_2',\ell,q_3')\notin F_R \\ q_3=q_1'}} \langle q_1, q_1', q_2, q_2' \rangle_R \wedge \langle q_2, \ell, q_1 \rangle_{\delta_C} \rightarrow \neg\langle q_2, \ell, q_1 \rangle_{\mathcal{F}_C} \tag{10}$$

Otherwise when one of the two transitions does not close the cycle ($q_3 \neq q_1 \vee q_3' \neq q_1'$) and the candidate transition (q_2, ℓ, q_3) is non-accepting, then the non-accepting path is prolonged (note that we don't care whether $(q_2', \ell, q_3') \in F_R$ or not):

$$\wedge \bigwedge_{\substack{(q_1,q_2,q_3)\in Q_C^3,\, (q_1',q_2')\in Q_R^2 \\ (\ell,q_3')\in\delta_R(q_2'), \\ (q_3'\neq q_1')\vee(q_3\neq q_1)}} \langle q_1, q_1', q_2, q_2' \rangle_C \wedge \langle q_2, \ell, q_3 \rangle_{\delta_C} \wedge \neg\langle q_2, \ell, q_3 \rangle_{\mathcal{F}_C} \rightarrow \langle q_1, q_1', q_3, q_3' \rangle_C \tag{11}$$

Conversely, if a path of the product that is non-accepting with respect to the **candidate** automaton, is completed by an **accepting** transition (q_2', ℓ, q_3') and a transition (q_2, ℓ, q_3) that both closes the cycle ($q_3 = q_1 \wedge q_3' = q_1'$), then (q_2, ℓ, q_1) is also accepting:

$$\wedge \bigwedge_{\substack{(q_1,q_2)\in Q_C^2,\, (q_1',q_2')\in Q_R^2, \\ (\ell,q_3')\in\delta_R(q_2'),\, (q_2',\ell,q_3')\in F_R \\ q_3'=q_1'}} \langle q_1, q_1', q_2, q_2' \rangle_C \wedge \langle q_2, \ell, q_1 \rangle_{\delta_C} \rightarrow \langle q_2, \ell, q_1 \rangle_{\mathcal{F}_C} \tag{12}$$

The symmetry-breaking equation (S) of Section 4 applies here as well.

As we did in our DTGBA encoding of Sec. 4.1, variables of the form $\langle q_1, q_1', q_2, q_2' \rangle_C$ or $\langle q_1, q_1', q_2, q_2' \rangle_R$ should be limited to cases where q_1' and q_2' belong to the same SCC of R, since it is not possible to build a cycle outside an SCC. Furthermore, variables of the form $\langle q_1, q_1', q_1, q_1' \rangle_R$ or $\langle q_1, q_1', q_2, q_2' \rangle_C$ are actually superfluous. Each time we use such a variable in equations (9)–(12), we could replace it by $\langle q_1, q_1' \rangle_G$. Doing so will avoid generating $|Q_C| \times |Q_R|$ variables, and all the clauses generated by equation (8) which is no longer needed.