Generating Deterministic ω-Automata for most LTL Formulas by the Breakpoint Construction

Andreas Morgenstern, Klaus Schneider and Sven Lamberti
Department of Computer Science, University of Kaiserslautern
P.O. Box 3049
67653 Kaiserslautern, Germany
Email: {morgenstern,schneider,lamberti}@informatik.uni-kl.de

Abstract

Temporal logics like LTL are frequently used for the specification and verification of reactive systems. To this end, LTL formulas are typically translated to nondeterministic *Büchi automata so that the LTL verification problem is reduced to a nonemptiness problem of ω-automata. While nondeterministic automata are sufficient for this purpose, many other applications require* deterministic *ω-automata. Unfortunately, the known determinization procedures for Büchi automata like Safra's procedure are extremely difficult to implement, and the currently available implementations are only able to handle very small examples.*

In this paper, we present a new symbolic translation of a remarkably large fragment of LTL formulas to equivalent deterministic ω-automata. Our method is based on (1) a syntactically defined fragment of the temporal logic LTL together with a linear-time translation procedure to equivalent nondeterministic symbolic ω-automata, and (2) a (semi)-symbolic determinization procedure for this fragment. The fragment that we consider is complete in the sense that every LTL formula is equivalent to a formula in this fragment, and in practice, we found that most formulas occurring in real specifications already belong to this fragment.

1 Introduction

Automata on finite and infinite words have been intensively studied in computer science. Automata on infinite words (called ω-automata) are used for the specification and verification of all kinds of reactive systems [20]. In particular, the model checking problem of the temporal logic LTL [16] (that is the core of PSL) can be reduced to checking the nonemptiness of ω-automata: To check whether a system $\mathcal{M}$ satisfies a LTL property φ, the negation $\neg\varphi$ is first translated to an equivalent (nondeterministic) ω-automaton $\mathfrak{A}_{\neg\varphi}$ so that the emptiness of the product automaton $\mathcal{M} \times \mathfrak{A}_{\neg\varphi}$ can be checked in a second step. Algorithms that translate the LTL formulas to *symbolically* represented ω-automata[1] [9] have been developed to benefit from symbolic set representations [7]. Thus, nondeterministic automata are already well established and form the backbones of many verification tools.

[1] For an LTL formula φ, these procedures compute in time $O(|\varphi|)$ a symbolic description $\mathfrak{A}_\varphi$ of a nondeterministic ω-automaton, which is comparable, but not equivalent to an alternating ω-automaton [22]. The description of $\mathfrak{A}_\varphi$ has size $O(|\varphi|)$, makes use of $O(|\varphi|)$ state variables, and therefore encodes $2^{O(|\varphi|)}$ states.

In contrast, algorithms that require the determinization of ω-automata have not yet made their way to industrial practice, although there are many applications where deterministic automata would be more convenient or even necessary, as e.g. the translation of branching time temporal logics like CTL* to (nondeterministic) automata on infinite trees [11], the computation of winning strategies for possibly infinite games [17] whose winning conditions are LTL formulas, the quantitative analysis of Markov decision processes [23] and, of course, the construction of monitors that run in parallel to a given implementation to check desired safety properties at runtime. Hence, algorithms to compute deterministic automata for given LTL formulas have already found many applications, and are therefore of high interest. However, most state-of-the-art translations of *full* LTL to ω-automata yield nondeterministic Büchi automata. After this, the well-known determinization procedure of Safra [19] is often employed to compute a deterministic (Rabin) automaton. Unfortunately, Safra's algorithm is very difficult to implement [14], and the underlying data structures do not lend themselves well for the use of symbolic set representations. As a consequence, the related tools are limited to small LTL formulas which restricts their application in practice.

In this paper, we therefore propose a new approach to the translation of LTL formulas to equivalent deterministic automata. Our approach is based on the temporal logic hierarchy [8, 20]. Based on this hierarchy, it can be proved [8, 20] that every LTL formula Φ can be written as a boolean combination of LTL formulas $\varphi_1, \ldots, \varphi_n$ that can be translated to nondeterministic co-Büchi[2] automata. If we are given an LTL formula Φ as such a boolean combination, we can use the algorithms presented in [20] to first translate the subformulas φ_i in linear time to linear sized symbolic descriptions of equivalent co-Büchi automata $\mathfrak{A}_i$. After this step, we can use the well-known breakpoint construction [15] to determinize these automata and finally, we compute the boolean closure of the obtained deterministic automata, which is straightforward due to the previous determinization steps. As a result, we finally obtain a deterministic Rabin or Streett automaton.

Due to results of [8], every LTL formula Φ can be written as a boolean combination of 'Büchi/co-Büchi' LTL formulas φ_i. However, the currently known transformations from arbitrary LTL formulas into such that normal form require a determinization step, and therefore, they are not useful for our purpose. Thus, we currently have the requirement that the given LTL formulas must already be boolean combinations of 'Büchi/co-Büchi' LTL formulas φ_i. In practice, we found that this requirement is almost always given, and in some other cases, it was not too difficult to manually rewrite the formulas (checking the equivalence of LTL formulas is simple). Therefore, we believe that our procedure is a valuable tool in practice. We substantiate this conjecture in Section 4 by experimental results.

It is known [1] that there are LTL formulas φ such that the smallest equivalent deterministic ω-automaton has doubly-exponential size $O(2^{2^{|\varphi|}})$. Hence, the complexity of determinization algorithms is a critical issue. For this reason, we make use of symbolic set representations in our translations as far as possible, namely, (1) by translating suitable subformulas of a given LTL formula to symbolic descriptions of nondeterministic safety or co-Büchi automata, (2) by employing new symbolic algorithms to determinize the obtained safety and co-Büchi automata, (3) and, finally by computing the boolean combination of the obtained deterministic automata.

The added values of this paper are therefore (1) the development of new symbolic determinization procedures for safety and co-Büchi automata, and (2) the use of these procedures to determinize all formulas of all classes in the temporal logic hierarchy. Our implementation is

[2]While Büchi automata demand that a run of a word must infinitely often visit a set of designated states, co-Büchi automata impose the dual requirement that, except for finitely many points of time, only designated states are visited.

remarkably efficient and can produce deterministic automata with hundreds of state variables.

There are already symbolic versions of variants of the subset and the breakpoint construction [3, 4]. In [4], procedures are described to compute a symbolically represented nondeterministic automaton from an alternating automaton, i.e., a 'nondeterminization' procedure. Although there are some similarities to our procedure, nondeterminization of alternating automata and determinization of nondeterministic automata is different for ω-automata [22]. Closer to our determinization procedure is [3] which generates a deterministic automaton for the safety fragment and thus implements the subset construction. However, they also start with an alternating automaton which is then translated to an explicitly represented nondeterministic automaton. The nondeterministic automaton is generated on the fly, thus avoiding the construction of the whole explicit automaton. However, this step crucially relies on a translation from alternating automata to the corresponding nondeterministic automata while our procedure is independent of the previous translation from temporal logic to nondeterministic automata. In particular, it is not obvious how the work [3] could be generalized to more expressive classes like co-Büchi automata.

The outline of the paper is as follows: In the next section, we list basic definitions on ω-automata and our use of symbolic representations of automata. Moreover, we briefly consider the automata hierarchy presented in [8, 20]. The main part of the paper describes the semi-symbolic determinization procedures in Section 3 that allowed us to implement highly efficient tools to translate LTL formulas to deterministic ω-automata. In Section 4, we present experimental results we obtained with our tool and the implementation of Safra's procedure of [14].

2 Basic Definitions

2.1 Symbolic Representations of ω-Automata

In this section, we briefly describe how we symbolically represent ω-automata. We assume that the input alphabet is encoded using boolean variables V_Σ. To represent the state transition relation of the automata, we introduce two state sets of propositional variables, one for the current and an associated one v' for the next point of time. Using these propositional variables, we are able to encode the initial states, the transition relation and the acceptance condition using simple propositional formulas:

Definition 1 *Given a finite set of variables Q with $Q \cap V_\Sigma = \{\}$, a propositional formula $\mathcal{I}$ over $Q \cup V_\Sigma$, a propositional formula $\mathcal{R}$ over $Q \cup V_\Sigma \cup \{v' \mid v \in Q\}$, and a LTL- formula $\mathcal{F}$ over $Q \cup V_\Sigma$, then $\mathcal{A}_\exists(Q, \mathcal{I}, \mathcal{R}, \mathcal{F})$ is an (existential) automaton formula.*

It is easily seen [20] that for automaton formulas $\mathcal{A}_\exists(Q, \mathcal{I}, \mathcal{R}, \mathcal{F})$, we can demand that $\mathcal{I}$ and $\mathcal{F}$ contain only state variables Q (yielding state-based instead of edge-based automata). In these cases, it is clear that an automaton formula describes a nondeterministic ω-automaton in a symbolic way: Q is the set of state variables, so the set of states corresponds with the powerset of Q. We identify any set $\vartheta \subseteq Q \cup V_\Sigma \cup \{v' \mid v \in Q \cup V_\Sigma\}$ with a propositional interpretation that exactly assigns the variables of ϑ to true, and the remaining ones to false. Having this view, the formula $\mathcal{I}$ describes the set of the initial states, which are the sets $\vartheta \subseteq Q$ that satisfy $\mathcal{I}$. Similarly, $\mathcal{R}$ symbolically represents the set of transitions.

2.2 Acceptance Conditions

In addition to the states and state transitions, we have to define the acceptance condition that determines whether a word is accepted by the automaton. Several kinds of acceptance conditions $\mathcal{F}$ have been proposed [8, 20]:

- A run is accepted by a safety condition $\mathsf{G}\varphi$ with a state set φ if the run exclusively runs through the set φ.
- A run is accepted by a liveness condition $\mathsf{F}\varphi$ with a state set φ if the run visits at least one state of the set φ at least once.
- A run is accepted by a Büchi condition $\mathsf{GF}\varphi$ with a state set φ if the run visits at least one state of the set φ infinitely often.
- A run is accepted by a co-Büchi condition $\mathsf{FG}\varphi$ with a state set φ if the run visits only states of the set φ infinitely often, i.e., states outside φ are only finitely often visited.

A run $\pi(w) = \pi_0\pi_1\ldots$ of an automaton over a word w is an (infinite) sequence of truth assignments to the variables in $Q \cup V_\Sigma$ that coincides with w on the variables of V_Σ, i.e. $\pi = \pi_0\pi_1\ldots$ is a sequence such that $\pi_0 \models \mathcal{I}$, and for each $i > 0$, we have $\pi_i \models \mathcal{R}$ and $\pi_i \cap V_\Sigma = w_i \cap V_\Sigma$. Thus, a run $\pi(w)$ is a word in $(2^{Q \cup V_\Sigma})^\omega$. A run is accepted by the automaton, if $\pi(w) \models \mathcal{F}$. Using the notation $w[i,\infty)$ to denote the suffix of w starting at position i, we have $w, i \models \mathcal{A}_\exists\,(Q, \mathcal{I}, \mathcal{R}, \mathcal{F})$ iff $\pi(w[i,\infty)) \models \mathcal{F}$.

The considered automaton formulas are usually nondeterministic. As usual, we call an automaton $\mathfrak{A}$ deterministic, if every word $w \in (2^{V_\Sigma})^\omega$ has exactly one run $\pi(w)$ through $\mathfrak{A}$.

2.3 The Hierarchy of ω-Automata and the Temporal Logic Hierarchy

The above acceptance conditions define the corresponding automaton classes $(\mathsf{N})\mathsf{Det}_\mathsf{G}$, $(\mathsf{N})\mathsf{Det}_\mathsf{F}$, $(\mathsf{N})\mathsf{Det}_\mathsf{GF}$, and $(\mathsf{N})\mathsf{Det}_\mathsf{FG}$, respectively. Moreover, $(\mathsf{N})\mathsf{Det}_\mathsf{Prefix}$ automata have acceptance conditions of the form $\bigwedge_{j=0}^{f} \mathsf{G}\varphi_j \vee \mathsf{F}\psi_j$, and $(\mathsf{N})\mathsf{Det}_\mathsf{Streett}$ automata have acceptance conditions of the form $\bigwedge_{j=0}^{f} \mathsf{GF}\varphi_j \vee \mathsf{FG}\psi_j$. The expressiveness of these classes is illustrated in Figure 1, where $\mathcal{C}_1 \precapprox \mathcal{C}_2$ means that for any automaton in $\mathcal{C}_1$, there is an equivalent one in $\mathcal{C}_2$. Moreover, we define $\mathcal{C}_1 \approx \mathcal{C}_2 := \mathcal{C}_1 \precapprox \mathcal{C}_2 \wedge \mathcal{C}_2 \precapprox \mathcal{C}_1$ and $\mathcal{C}_1 \npreccurlyeq \mathcal{C}_2 := \mathcal{C}_1 \precapprox \mathcal{C}_2 \wedge \neg(\mathcal{C}_1 \approx \mathcal{C}_2)$. As can be seen, the hierarchy consists of six different classes, and each class has a deterministic representative.

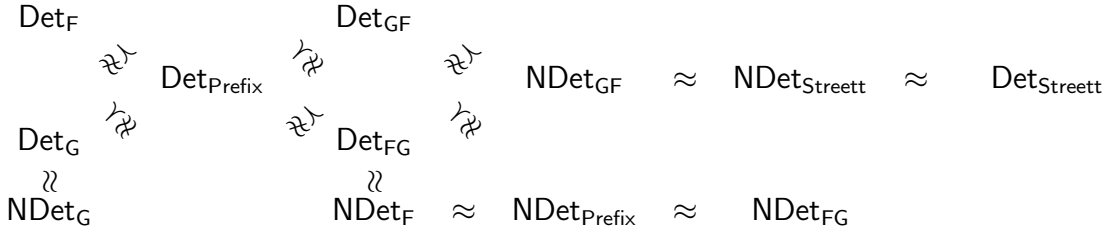

Figure 1: (Borel) Hierarchy of ω-Automata

In analogy, a temporal logic hierarchy has been defined [8, 20] that consists of six temporal logics TL_κ for $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Rabin}\}$ that correspond with the six automaton classes $(\mathsf{N})\mathsf{Det}_\kappa$. Translations of $\Phi \in \mathsf{TL}_\kappa$ with $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{GF}, \mathsf{FG}\}$ to symbolic representations of automata in NDet_κ require only time $O(|\Phi|)$. Moreover, for every formula $\Phi \in \mathsf{TL}_\kappa$, we can construct a symbolic description $\mathfrak{A}$ of a deterministic ω-automaton of the class Det_κ in time $O(2^{|\Phi|})$ with $|Q| \leq 2^{|\Phi|}$ state variables. Note that $\mathfrak{A}$ may have $O(2^{2^{|\Phi|}})$ states.

3 Symbolic Determinization Procedures

As reported in the previous section, it is already known that we can compute for every formula $\varphi \in \mathsf{TL}_\kappa$ an equivalent deterministic ω-automaton $\mathfrak{A}_\varphi \in \mathsf{Det}_\kappa$. Moreover, it is already known that the subset and the breakpoint constructions are sufficient for this purpose [20], so that there is no need for Safra's considerably more difficult determinization procedure for these classes.

However, a critical issue of the known translations from the classes TL_κ to Det_κ is still their complexity: It is well-known [1] that there exists formulas $\Phi \in \mathsf{TL}_\kappa$ such that all equivalent deterministic automata have at least $2^{2^{\Omega|\Phi|}}$ states. Symbolic translation procedures from TL_κ to NDet_κ as shown in [20] elegantly circumvent a first bottleneck. However, all determinization procedures including the subset and the breakpoint construction assume explicitly represented automata, so that this advantage can no longer be exploited. Even worse, the resulting deterministic automata are also given in an explicit representation, so that naive implementations really suffer from the double exponential complexity.

For this reason, a major improvement is obtained by the results of this section that show how the subset and the breakpoint construction can be implemented in a semi-symbolic way: For a given symbolically represented nondeterministic automaton $\mathcal{A}_\exists\,(Q, \mathcal{I}, \mathcal{R}, \mathcal{F})$ with reachable states $\{\vartheta_1, \ldots, \vartheta_n\}$, we directly construct symbolic descriptions of the deterministic automata that are constructed by the subset and the breakpoint constructions. Although we can not avoid one exponential step (namely the enumeration of the reachable states), we achieved that the symbolic description of the deterministic automaton can be obtained without enumerating its states. All steps except for the enumeration of the reachable states of the nondeterministic automaton are symbolically implemented. As a result, we obtained highly efficient determinization procedures that allow us to translate large LTL formulas to equivalent deterministic automata. In the following subsections, we describe these algorithms in detail, but we assume that the reader is familiar with the subset and the breakpoint construction.

3.1 Symbolic Implementation of the Subset Construction

As already explained, our algorithms expect a *symbolic* representation of a nondeterministic automaton, i.e. a formula $\mathcal{A}_\exists\,(Q, \mathcal{I}, \mathcal{R}, \mathcal{F}')$, where $\mathcal{I}$ is a propositional formula over the state variables Q, and where the acceptance condition $\mathcal{F}'$ is based on a propositional formula $\mathcal{F}$ over the state variables Q that is used to construct a safety, liveness, fairness or persistence property $\mathcal{F}'$. In the following, we assume that $Q = \{q_1, \ldots, q_m\}$ and that $V_\Sigma = \{x_1, \ldots, x_k\}$ holds.

The first step of our algorithms consists of *computing the reachable states* of the automaton. To this end, we first eliminate all variables that are not state variables, i.e., we define $\mathcal{R}_\exists :\equiv \exists x_1 \ldots x_k.\ \mathcal{R}$, and compute then the reachable states. The reachable states can be computed as the fixpoint[3] $\mu x.\ \mathcal{I} \vee \overleftarrow{\Diamond}\, x$, but since we additionally eliminate deadend states, we compute $\mathcal{S}_{\mathsf{reach}} :\equiv (\nu y.\Diamond y) \wedge (\mu x.\ \mathcal{I} \vee \overleftarrow{\Diamond}\, x)$ instead, i.e. the reachable states having at least one infinite path. Using symbolic methods, the result $\mathcal{S}_{\mathsf{reach}}$ is a propositional formula over the state variables Q.

Having computed the reachable states $\mathcal{S}_{\mathsf{reach}}$, we now perform a one-hot encoding of the states of the original nondeterministic automaton. To this end, we have to explicitly enumerate the reachable states which are the models of the formula $\mathcal{S}_{\mathsf{reach}}$. Recall that we identify a variable assignment with a set $\vartheta \subseteq Q$ such that exactly the variables contained in ϑ are true. Thus, assume the reachable states are $\{\vartheta_1, \ldots, \vartheta_n\}$. We now introduce new state variables $Q_{\mathsf{det}} =$

[3] For a set of states x, $\overleftarrow{\Diamond}\, x$ denotes the existential successor states of x.

$\{p_1, \ldots, p_n\}$ to identify a reachable state ϑ_i with a state variable p_i. Moreover, for $\vartheta \subseteq Q$, we define the corresponding minterm as:

$$\mathsf{mt}_Q(\vartheta) :\equiv \left(\bigwedge_{x \in \vartheta} x\right) \wedge \left(\bigwedge_{x \in Q \setminus \vartheta} \neg x\right)$$

Based on these definitions, the symbolic subset construction is described as follows:

Definition 2 (Symbolic Subset Construction) *Given $\mathfrak{A} = \mathcal{A}_\exists\,(Q, \mathcal{I}, \mathcal{R}, \mathcal{F})$ with the reachable states $\{\vartheta_1, \ldots, \vartheta_n\}$ and new state variables $Q_{\mathsf{det}} = \{p_1, \ldots, p_n\}$, we define the following automaton $\mathfrak{A}_{\mathsf{det}} = \mathcal{A}_\exists\,(Q_{\mathsf{det}}, \mathcal{I}_{\mathsf{det}}, \mathcal{R}_{\mathsf{det}}, \mathcal{F}_{\mathsf{det}})$:*

- $\mathcal{H} :\equiv \bigvee_{j=1}^{n} p_j \wedge \mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_j)$
- $\mathcal{I}_{\mathsf{det}} :\equiv \bigwedge_{i=1}^{n} (p_i \leftrightarrow \exists q_1 \ldots q_m.\ \mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i) \wedge \mathcal{I})$
- $\mathcal{F}_{\mathsf{det}} :\equiv \exists q_1 \ldots q_m.\ \mathcal{H} \wedge \mathcal{F}$
- $\mathcal{R}_{\mathsf{det}} :\equiv \bigwedge_{i=1}^{n} (p_i' \leftrightarrow \eta_i)$, *with* $\eta_i :\equiv \exists q_1 \ldots q_m q_1' \ldots q_m'.\ \mathcal{H} \wedge \mathcal{R} \wedge (\mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i))'$

As can be seen, the initial condition and the transition relation of $\mathfrak{A}_{\mathsf{det}}$ are given as equation systems, which is beneficial for many applications.

To see that our construction is correct, consider first the acceptance condition $\mathcal{F}_{\mathsf{det}}$: Note that $\mathcal{F}_{\mathsf{det}}$ can be rewritten as $\bigvee_{j=1}^{n} p_j \wedge \exists q_1 \ldots q_m.\ \mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_j) \wedge \mathcal{F}$. The subformula $\mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i) \wedge \mathcal{F}$ is false in case that ϑ_i is a variable assignment that does not satisfy $\mathcal{F}$. Otherwise, $\mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i) \wedge \mathcal{F}$ is equivalent to $\mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i)$. Thus, the existential quantification used in the definition of $\mathcal{F}_{\mathsf{det}}$ yields true if ϑ_i belongs to $\mathcal{F}$ (since every minterm $\mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i)$ is satisfiable), and yields false if ϑ_i does not belong to $\mathcal{F}$. Consequently, $\mathcal{F}_{\mathsf{det}}$ is equivalent to the disjunction of those p_j that correspond to states ϑ_j of $\mathcal{F}$, i.e. $\mathcal{F}_{\mathsf{det}} \Leftrightarrow \bigvee_{\vartheta_i \in \mathcal{F}} p_i$.

The initial superstate of the subset construction is the set of initial states $\mathcal{I}$, i.e., the initial condition is $\bigwedge_{i=1}^{n} p_i \leftrightarrow \alpha_i$, where $\alpha_i \in \{1, 0\}$, so that $\alpha_i = 1$ iff $\vartheta_i \in \mathcal{I}$. Now note that $\exists q_1 \ldots q_m.\ \mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i) \wedge \mathcal{I}$ evaluates to true iff $\vartheta_i \in \mathcal{I}$.

Finally, consider the transition relation $\mathcal{R}_{\mathsf{det}}$: The successor states of a superstate $\Theta \subseteq Q_{\mathsf{det}}$ under the input condition $\tau_{j,i}$ are given as the set of states p_j that have a transition under the input condition $\tau_{j,i}$ to a state $p_j \in \Theta$. Our definition of $\mathcal{R}_{\mathsf{det}}$ directly implements this: in the next step, all p_i are true (thus belong to the superstate Θ) whenever η_i holds. Now η_i is equivalent to

$$\bigvee_{j=1}^{n} p_j \wedge \underbrace{\exists q_1 \ldots q_m q_1' \ldots q_m'.\ \mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_j) \wedge \mathcal{R} \wedge (\mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i))'}_{\tau_{j,i}},$$

where $\tau_{j,i}$ is the condition on the input variables $V_\Sigma = \{x_1, \ldots, x_k\}$ that must hold to enable the transition from state p_j to state p_i. Thus, η_i is equivalent to $\bigvee_{j=1}^{n} p_j \wedge \tau_{j,i}$, and thus, η_i lists all possibilities to reach state p_i from any other reachable state.

Therefore, the above definition implements the subset construction in a symbolic way. We therefore can now state the following theorem:

Theorem 1 (Symbolic Subset Construction) *For every automaton $\mathfrak{A}$, the automaton $\mathfrak{A}_{\mathsf{det}}$ as constructed in Definition 2 is deterministic and is a symbolic description of the automaton obtained by the well-known subset construction [18, 20].*

Since the subset construction can not only be used to determinize automata on finite words, but also ω-automata that stem from the classes TL_{G}, TL_{F}, and $\mathsf{TL}_{\mathsf{Prefix}}$, we can already handle these classes[4] with the construction given in Definition 2.

3.2 Symbolic Implementation of the Breakpoint Construction

In order to handle the remaining classes of the temporal logic hierarchy, we show in the next definition how the breakpoint construction for the determinization of the class $\mathsf{NDet}_{\mathsf{FG}}$ can be implemented in a symbolic manner. Again, by using dualities between the classes, we are then able to determinize also the classes $\mathsf{TL}_{\mathsf{GF}}$, $\mathsf{TL}_{\mathsf{FG}}$ and $\mathsf{TL}_{\mathsf{Streett}}$.

Definition 3 (Symbolic Breakpoint Construction) *Given $\mathfrak{A} = \mathcal{A}_{\exists}\left(Q, \mathcal{I}, \mathcal{R}, \mathcal{F}\right)$ with the reachable states $\{\vartheta_1, \ldots, \vartheta_n\}$ so that $\mathcal{F}$ is the set $\{\vartheta_{n+1-\ell}, \ldots, \vartheta_n\}$. Using new state variables $Q_{\mathsf{bpt}} = \{p_1, \ldots, p_n, b_1, \ldots, b_\ell\}$ and the definitions of $\mathcal{I}_{\mathsf{det}}$ and $\mathcal{R}_{\mathsf{det}}$ of Definition 2, we define $\mathfrak{A}_{\mathsf{bpt}} = \mathcal{A}_{\exists}\left(Q_{\mathsf{bpt}}, \mathcal{I}_{\mathsf{det}} \wedge \mathcal{I}_{\mathsf{bpt}}, \mathcal{R}_{\mathsf{det}} \wedge \mathcal{R}_{\mathsf{bpt}}, \mathcal{F}_{\mathsf{bpt}}\right)$ as follows:*

- $\mathcal{H} :\equiv \bigvee_{j=1}^{n} p_j \wedge \mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_j)$
- $\mathcal{I}_{\mathsf{bpt}} :\equiv \bigwedge_{i=1}^{\ell} b_i \leftrightarrow 0$
- $\mathcal{F}_{\mathsf{bpt}} :\equiv \bigvee_{i=1}^{\ell} b_i$
- $\mathcal{R}_{\mathsf{bpt}} :\equiv \bigwedge_{i=1}^{\ell} b_i' \leftrightarrow \neg\mathcal{F}_{\mathsf{bpt}} \wedge \eta_i \vee \mathcal{F}_{\mathsf{bpt}} \wedge [\eta_i]_{\varrho}$, *with*
  $\eta_i :\equiv \exists q_1 \ldots q_m q_1' \ldots q_m'.\ \mathcal{H} \wedge \mathcal{R} \wedge (\mathsf{mt}_{Q_{\mathsf{det}}}(\vartheta_i))'$ *and ϱ is the substitution that maps each $p_1, \ldots, p_{n-\ell}$ to 0 and $p_{n-\ell}, \ldots, p_n$ to $b_1, \ldots, b_\ell$, respectively*

The idea behind the breakpoint construction is to maintain pairs of sets of states, where the first component is computed by the subset construction. The second component is the set of states that have never left the set of designated states since the last breakpoint, where a breakpoint is a state whose second component is empty.

States of $\mathfrak{A}_{\mathsf{bpt}}$ correspond with subsets of Q_{bpt} which may be considered as pairs of subsets of $\{p_1, \ldots, p_n\}$ and $\{b_1, \ldots, b_\ell\}$. A breakpoint is a pair (S_1, S_2) where S_2 represents an empty state set, i.e., all of the variables b_j are false and $\mathcal{F}_{\mathsf{bpt}}$ evaluates to false. Whenever a breakpoint is reached, the second set is filled with the designated successors of the first set. In this case, we evaluate the status of the b_j according to the variables p_i, thus all we have to do is to copy the formula representing the transition relation of the subset construction. Otherwise, the usual subset construction is performed on the second step in the explicit breakpoint construction. To calculate the transition relation in our symbolic setting, it is sufficient to eliminate transitions from non-accepting states (which is done by setting $p_i = 0$) and then replacing each occurrence of p_i for $\vartheta_i \in \mathcal{F}$ by the corresponding b_i.

Hence, also the breakpoint construction can be implemented in a symbolic manner. It is well-known that $\mathfrak{A}_{\mathsf{bpt}}$ may have at most $O(3^n)$, reachable states, while the automaton $\mathfrak{A}_{\mathsf{det}}$ obtained from the subset construction may have at most $O(2^n)$ reachable states.

[4] Note, however, that $\mathsf{NDet}_{\mathsf{F}}$ is not the dual class of $\mathsf{NDet}_{\mathsf{G}}$ (see Figure 1).

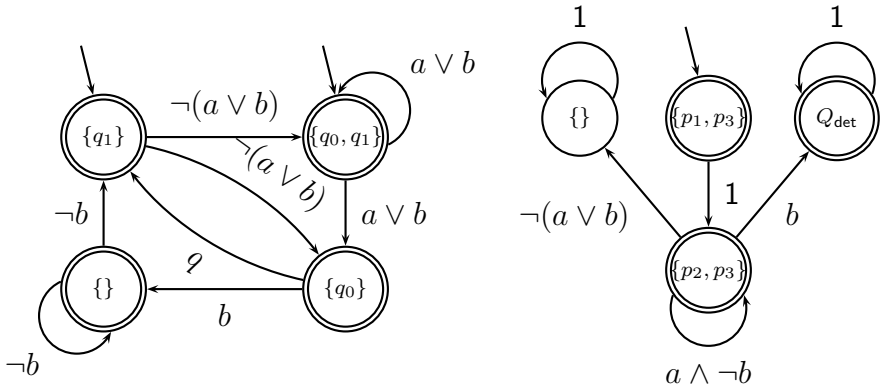

Figure 2: Nondeterministic ω-automaton obtained from $\varphi :\equiv \mathsf{X}\,[a\ \mathsf{U}\ b]$ (left hand side), and the corresponding deterministic ω-automaton obtained from the subset construction with the acceptance condition $\mathsf{G}(p_0 \vee p_1 \vee p_2 \vee p_3)$ and $Q_{\mathsf{det}} := \{p_0, p_1, p_2, p_3\}$.

Theorem 2 (Symbolic Breakpoint Construction) *For every automaton $\mathfrak{A}$, the automaton $\mathfrak{A}_{\mathsf{bpt}}$ as constructed in Definition 3 is deterministic and is a symbolic description of the automaton obtained by the well-known breakpoint construction [15, 20].*

3.3 Illustrating Example

As an example, consider the LTL formula $\varphi :\equiv \mathsf{X}\,[a\ \mathsf{U}\ b]$. Using the translation given in [20], we obtain the following equivalent nondeterministic safety automaton

$$\mathfrak{A}_\varphi = \mathcal{A}_\exists\left(\{q_0, q_1\}, q_1, (q_0 \leftrightarrow b \vee a \wedge q_0') \wedge (q_1 \leftrightarrow q_0'), 1\right).$$

Its state transition diagram is given on the left hand side of Fig. 2, and its acceptance condition simply demands that there must be an infinite run. Using the above algorithm, we obtain the following equation systems for $\mathfrak{A}_{\mathsf{det}}$ (where we encoded the reachable states as follows: $p_0 \sim \vartheta_0 = \{\}$, $p_1 \sim \vartheta_1 = \{q_1\}$, $p_2 \sim \vartheta_2 = \{q_0\}$, and $p_3 \sim \vartheta_3 = \{q_0, q_1\}$):

$$\mathcal{I}_{\mathsf{det}} = \begin{cases} p_0 \leftrightarrow 0 \\ p_1 \leftrightarrow 1 \\ p_2 \leftrightarrow 0 \\ p_3 \leftrightarrow 1 \end{cases} \qquad \mathcal{R}_{\mathsf{det}} = \begin{cases} p_0' \leftrightarrow p_0 \wedge \neg b \vee p_2 \wedge b \\ p_1' \leftrightarrow p_0 \wedge \neg b \vee p_2 \wedge b \\ p_2' \leftrightarrow p_1 \wedge \neg(a \vee b) \vee p_3 \wedge (a \vee b) \\ p_3' \leftrightarrow p_1 \wedge \neg(a \vee b) \vee p_3 \wedge (a \vee b) \end{cases}$$

The state transition diagram of this deterministic automaton is shown in Figure 2.

4 Experimental Results

We have implemented the presented algorithms in our Averest framework and made some experiments that we compared with the tool ltl2dstar [14] that is an implementation of Safra's construction. The output of our tool is a symbolic representation of the computed deterministic automaton. The output of ltl2dstar is a textual representation of the explicit deterministic automaton. All experiments have been performed on a dual Opteron workstation with 2Ghz running SUSE Linux 9.3.

As a starting point, we examined the examples from [12, 21, 10]. From the 94 formulas contained in the sample set, only 4 do not belong to $\mathsf{TL}_{\mathsf{Streett}}$. However, after a manual rewriting

AMBA Protocol				
n	Time[sec]	Mem[MB]	#state variables	
			det. auto.	ndet. auto.
2	0.18	14.4	401	54
4	0.27	14.9	758	88
8	0.53	15.7	1486	156
16	1.4	17.9	2942	292

GenBuf				
n	Time[sec]	Mem[MB]	#state variables	
			det. auto.	ndet. auto.
2	0.14	13.6	233	58
4	0.16	13.6	341	89
8	0.22	13.6	581	163
16	0.41	15.3	1157	359
32	1.44	18.3	2693	943
64	10.00	104.0	7301	2879

Figure 3: Experimental results of our case studies

step, all of the formulas can be translated to deterministic automata with our procedure[5]. All example formulas of [12, 21, 10] where translated by both Averest and ltl2dstar in a couple of seconds using at most 30 MB memory. None of the algorithms typically outperformed the other on these rather small examples, but we can already see that *almost all LTL formulas used in specifications already belong to* $\mathsf{TL}_{\mathsf{Streett}}$.

To evaluate the performance on real world examples, we first considered the AMBA bus protocol [2] specification given in [5]. The specification consists of 11 safety specifications, 3 fairness constraints, and one constraint on the initial state. The entire specification, i.e., the conjunction of these formulas, belongs to $\mathsf{TL}_{\mathsf{Streett}}$. In the original formulation of the specification, Averest required more than 4 hours and used more than 129 MB memory for 6 masters. We quickly identified a problematic subformula and after manually rewriting this formula[6], Averest was able to generate deterministic automata for all possible instances of the AMBA protocol with the runtime requirements shown in Figure 3. Averest would even be able to handle larger instances, which are however not specified in the AMBA protocol (only up to 16 masters and slaves are considered).

The tool ltl2dstar was not able to finish even the rewritten specification with 2 masters: After 1 minute, we terminated the process that already claimed an enormous amount of 3.5 GB of main memory. Until that point of time, ltl2dstar was still busy with the translation to a nondeterministic automaton using the translator ltl2ba [13].

As a second example, we considered the generalized buffer that has been developed by IBM as a tutorial for the Rulebase verification tool [7]. GenBuf comes with a relatively complete specification in PSL for a family of buffers parameterized by a number n. Data is offered by the senders in an arbitrary order, and is received by the receivers in round-robin order. We used the modified specifications given in [6]. One problem that we encountered was that the rose and fell operator of PSL is not an LTL operator. However, by using a deterministic monitor, we were able to translate the specification to a LTL specification and finally to a deterministic automaton using Averest. Unfortunately, ltl2dstar does not support past operators, so that we can not give a comparison with ltl2dstar on this example.

[5] As an example for this rewriting process, consider the formula $\mathsf{G}\,(p \rightarrow [q\ \underline{\mathsf{U}}\ (\mathsf{G}r \vee \mathsf{G}s)])$ from [12] and the equivalent $\mathsf{TL}_{\mathsf{Streett}}$ formula $\mathsf{G}\neg p \vee \mathsf{G}\,(p \rightarrow [q\ \mathsf{U}\ (\mathsf{G}r \vee \mathsf{G}s)]) \wedge \mathsf{F}(\mathsf{G}r \vee \mathsf{G}r)$

[6] The problematic formula was $\mathsf{G}(decide \rightarrow \bigwedge_i hgrant[i] \leftrightarrow \mathsf{X}hgrant[i])$ which we rewrote to $\bigwedge_i \mathsf{G}(decide \rightarrow hgrant[i] \leftrightarrow \mathsf{X}hgrant[i])$.

[7] http://www.haifa.ibm.com/projects/verification/RB_Homepage/tutorial3/GenBuf_english_spec.htm

5 Summary

We exploited the temporal logic hierarchy [8, 20] so that the subset and the breakpoint constructions suffice to translate a large fragment of LTL to equivalent deterministic ω-automata. While this already follows from results given in [20], it has not yet found the way into efficient tools so far. The main added value of this paper is therefore to put those theoretical results into practice by the development of new algorithms that directly compute a symbolic representation of the deterministic automata. Although our translation to deterministic ω-automata can not handle all LTL formulas, it seems to be sufficient in practice: Almost all specifications we found in our experiments can be translated with our algorithm. The experimental results we have presented show clearly that our tool outperforms existing tools in that area.

References

[1] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Transactions on Computational Logic*, 5(1):1–15, 2004.

[2] ARM Ltd. AMBA specification (rev.2), 1999. http://www.arm.com.

[3] R. Armoni, S. Egorov, R. Fraer, D. Korchemny, and M. Vardi. Efficient LTL compilation for SAT-based model checking. In *Conf. on Computer Aided Design (ICCAD)*, pp. 877–884. IEEE Computer Society, 2005.

[4] R. Bloem, A. Cimatti, I. Pill, M. Roveri, and S. Semprini. Symbolic implementation of alternating automata. In O. Ibarra and H.-C. Yen, editors, *Conference on Implementation and Application of Automata (CIAA)*, *LNCS* 4094, pp. 208–218, Taipei, Taiwan, 2006. Springer.

[5] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: A case study. In *Design, Automation and Test in Europe (DATE)*, 2007.

[6] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from PSL. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 190:3–16, 2007.

[7] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Logic in Computer Science (LICS)*, pp. 1–33, Washington, DC, USA, 1990. IEEE Computer Society.

[8] E. Chang, Z. Manna, and A. Pnueli. Characterization of temporal property classes. In *International Colloquium on Automata, Languages and Programming (ICALP)*, *LNCS* 623, pp. 474–486. Springer, 1992.

[9] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design (FMSD)*, 10(1):47–71, February 1997.

[10] M. Dwyer, G. Avrunin, and J. Corbett. Property specification patterns for finite-state verification. In *Formal Methods in Software Practice (FMSP)*, pp. 7–15, Clearwater Beach, Florida, United States, 1998. ACM.

[11] E. Emerson and A. Sistla. Deciding branching time logic. In *Symposium on Theory of Computing (STOC)*, pp. 14–24, 1984.

[12] K. Etessami and G. Holzmann. Optimizing Büchi automata. In C. Palamidessi, editor, *Conference on Concurrency Theory (CONCUR)*, *LNCS* 1877, pp. 153–167, University Park, PA, USA, 2000. Springer.

[13] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Computer Aided Verification (CAV)*, *LNCS* 2102, pp. 53–65, Paris, France, 2001. Springer.

[14] J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of temporal logic. *Theoretical Computer Science*, 363(2):182–195, 2006.

[15] S. Miyano and T. Hayashi. Alternating automata on ω-words. *Theoret. Comp. Science*, 32:321–330, 1984.

[16] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science (FOCS)*, pp. 46–57, Providence, RI, USA, 1977. IEEE Computer Society.

[17] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Symposium on Principles of Programming Languages (POPL)*, pp. 179–190, Austin, Texas, 1989. ACM.

[18] M. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[19] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Symposium on Theory of Computing (STOC)*, pp. 275–282, 1992.

[20] K. Schneider. *Verification of Reactive Systems - Formal Methods and Algorithms*. Texts in Theoretical Computer Science (EATCS Series). Springer, 2003.

[21] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Computer Aided Verification (CAV)*, *LNCS* 1855, pp. 248–263, Chicago, IL, USA, 2000. Springer.

[22] T. Tuerk and K. Schneider. Relationship between alternating ω-automata and symbolically represented nondeterministic ω-automata. Technical Report 340, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, 2005.

[23] M. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In J.-P. Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems (ARTS)*, *LNCS* 1601, pp. 265–276, Bamberg, Germany, 1999. Springer.