Exploiting the Temporal Logic Hierarchy and the Non-Confluence Property for Efficient LTL Synthesis

Andreas Morgenstern and Klaus Schneider
University of Kaiserslautern
P.O. Box 3049
67653 Kaiserslautern, Germany
email: {morgenstern,schneider}@cs.uni-kl.de

The classic approaches to synthesize a reactive system from a linear temporal logic (LTL) specification first translate the given LTL formula to an equivalent ω-automaton and then compute a winning strategy for the corresponding ω-regular game. To this end, the obtained ω-automata have to be (pseudo)-determinized where typically a variant of Safra's determinization procedure is used. In this paper, we show that this determinization step can be significantly improved for tool implementations by replacing Safra's determinization by simpler determinization procedures. In particular, we exploit (1) the temporal logic hierarchy that corresponds to the well-known automata hierarchy consisting of safety, liveness, Büchi, and co-Büchi automata as well as their boolean closures, (2) the non-confluence property of ω-automata that result from certain translations of LTL formulas, and (3) symbolic implementations of determinization procedures for the Rabin-Scott and the Miyano-Hayashi breakpoint construction. In particular, we present convincing experimental results that demonstrate the practical applicability of our new synthesis procedure.

1 Introduction

In formal verification, we have to check for a given implementation $\mathscr{M}$ and a given LTL property φ whether $\mathscr{M}$ satisfies φ in any environment, which is usually written as $\mathscr{M} \models \varphi$. In the more general synthesis problem, we have to check whether an (incomplete) implementation[1] $\mathscr{M}$ can be refined by another system (often called a controller) $\mathscr{C}$ such that a given property φ holds, i.e., whether there exists a system $\mathscr{C}$ such that $\mathscr{M} \parallel \mathscr{C} \models \varphi$ holds. This means that the combined behavior of $\mathscr{C}$ and $\mathscr{M}$ satisfies φ in *any* environment. This controller synthesis problem can be naturally viewed as an infinite game between the controller and an adversary environment.

Due to the enormous progress made during the past two decades, the tools used in formal verification can now be applied to real-world problems, and therefore are routinely used in industrial practice. In contrast, the so-far known tools for the synthesis of LTL specifications can only be applied to small examples which is due to the implementation of these tools: The currently available procedures used in these tools consist of two steps: Similar to verification, the first step consists of translating the LTL formula φ to an equivalent nondeterministic ω-automaton $\mathfrak{A}_\varphi$. While this automaton $\mathfrak{A}_\varphi$ can be directly used for symbolic model checking, it is favorable to have a deterministic automaton for constructing a ω-regular game from the obtained automaton $\mathfrak{A}_\varphi$ and the incomplete implementation $\mathscr{M}$. This determinization step is the main reason for the high complexity of the synthesis problem for LTL.

In particular, Safra's construction [23] is often used for this purpose. This construction generates for a given Büchi automaton with n states an equivalent deterministic Rabin automaton with $12^n \cdot n^{2n}$ states and n acceptance pairs. However, Safra's construction is extremely difficult to implement [11],

[1] It is allowed that $\mathscr{M}$ consists of only the environment.

Angelo Montanari, Margherita Napoli, Mimmo Parente (Eds.)
Proceedings of GandALF 2010
EPTCS 25, 2010, pp. 89–102, doi:10.4204/EPTCS.25.11

© A. Morgenstern & K. Schneider
This work is licensed under the
Creative Commons Attribution License.

and unfortunately not amenable to a symbolic implementation. Indeed, a major drawback of Safra's construction is that known implementations use an *explicit representation* of the automata, since the states of Safra's automaton consist of trees whose nodes are labeled with sets of states. This is probably the main reason why tools for synthesis lack behind (compared to model checking tools that achieved a significant breakthrough first by symbolic state space representations, and second by the use of efficient SAT solvers). As a consequence, tools for LTL synthesis are still limited to very small formulas with only few temporal operators.

In this paper, we show how we can always replace Safra's determinization procedure by simpler ones like the Rabin-Scott subset construction and the Miyano-Hayashi breakpoint construction [17, 25]. To this end, we exploit the membership of the subformulas of a given formula to the classes of the temporal logic hierarchy as well as a recently published determinization procedure for non-confluent ω-automata [18].

The *temporal logic hierarchy* [5, 25] has been developed in correspondence to the ω-automata hierarchy [14] that has been inspired by the Borel hierarchy known from descriptive set theory. As the ω-automata hierarchy, the temporal logic hierarchy distinguishes between six different classes (cf. Figure 1) of properties. It is moreover known that each class can be characterized by a deterministic class of ω-automata that differ in their acceptance conditions as explained in Section 4.1. While Manna and Pnueli's original definition of the hierarchy was a semantic one (meaning a LTL formula belongs to a class iff there is an equivalent ω-automaton in that class), Schneider [24, 25] presented syntactic characterizations of these classes by means of simple grammars for each of these logics (cf. Figure 2). Clearly, such syntactic characterizations are incomplete (in the sense that an equivalent formula may belong to a different class), but they are sufficient for practical use: In general, the membership test given by these grammars yields an upper bound of the temporal logic class, and in practice, it often yields the precise class, making therefore more expensive tests unnecessary.

In the following, we will therefore also make use of a syntactic, and therefore incomplete, but more efficiently testable definition to come up with a very efficient LTL synthesis procedure. The key idea of this procedure is thereby based on the following observation: It is well-known since [5] that for every LTL formula of each temporal logic class, there is an equivalent deterministic automaton with the corresponding acceptance condition. However, it was *previously not recognized* that we can avoid complex determinization procedures like Safra's one once we can express a given LTL formula as a boolean combination of lower class subformulas.

In fact, by the syntactic representation, each formula in the highest class is already a boolean combination of co-Büchi properties, so that we can obtain a Safra-less determinization procedure by first computing deterministic co-Büchi automata for these subformulas, and then computing the boolean closure of the obtained deterministic automata (which is simple as outlined in [25]). The translation of the subformulas to symbolically represented non-deterministic co-Büchi automata has been presented in detail in [24, 25], and their determinization can be performed using the Miyano-Hayashi breakpoint construction [17, 25]. We can even improve this procedure by determining tighter classes like safety and liveness properties so that even the Rabin-Scott subset construction is sufficient for the determinization.

The second major ingredient of our efficient synthesis procedure are *symbolic determinization constructions* we have developed in [19] for the Rabin-Scott subset construction [22] and the Miyano-Hayashi breakpoint construction: For a given symbolically represented nondeterministic automaton, we directly construct symbolic descriptions of the corresponding deterministic automata. Although we can not avoid one exponential step (namely the enumeration of the reachable states of the nondeterministic automaton), we achieved that the symbolic description of the deterministic automaton can be obtained without building it explicitly. Thus, we avoid the enumeration of the exponentially larger state space of

the *deterministic* automaton.

Thus, we are able to translate every formula of the temporal logic hierarchy to an equivalent deterministic ω-automaton. Due to results of [5], one can translate every LTL formula to a formula contained at least in the highest class of this hierarchy. However, all known such translations require a determinization step, and therefore, they are not useful for our purpose. Thus, we currently have the restriction that the given LTL formulas must already syntactically belong to one of the classes of the hierarchy. In practice, we found that this is almost always the case (in more than 95% in our benchmarks), and in some other cases, it was not too difficult to rewrite the formula to achieve this membership (checking the equivalence of the rewritten LTL formulas is no problem by verification tools).

To handle the remaining rare cases of formulas that can not be easily rewritten to classes of the temporal logic hierarchy, we use as a final ingredient of our synthesis procedure our recently published determinization procedure for non-confluent automata [18]. This procedure can be applied to any given LTL formula, provided we use a translation to ω-automata that generates non-confluent automata (which exists!).

In this paper, we build upon the mentioned results we developed in our previous work, i.e., we first try to decompose a given specification into subformulas that syntactically belong to a class of the temporal logic hierarchy, and apply symbolic implementations of the Rabin-Scott and the Miyano-Hayashi constructions to translate these subformulas to deterministic ω-automata. If this is not directly possible, we make use of a simple translation for full LTL *to non-confluent automata so that we can apply our recently published determinization procedure [18].*

Having obtained a deterministic automaton, we use standard automata translations to obtain (generalized) deterministic parity automata for each subformula which has to be combined in one conjunctive generalized parity condition. The obtained generalized parity automaton yields a generalized parity game which is solved in the final step of our algorithm using the symbolic algorithm given in [7].

The outline of this paper is at follows: after starting with more details on related work (Section 2) and some basic definitions (Section 3), we explain how we can make use of the temporal logic hierarchy (Section 4). The second ingredient is the exploitation of the non-confluence property that is explained in Section 5. The added value and the core of this paper is the combination of these results in Section 6 to obtain an efficient symbolic synthesis procedure for full LTL. We conclude the paper by experimental results in Section 7.

2 Related Work

There are already symbolic implementations of the subset and breakpoint construction [1, 2]. In [2], procedures are described to compute a symbolically represented nondeterministic automaton from a given alternating automaton, i.e., a non-determinization procedure. Although there are some similarities to our procedure, non-determinization of alternating automata and determinization of nondeterministic automata is different for ω-automata [29]. Closer to our determinization procedure is [1] 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 [1] could be generalized to more expressive classes like co-Büchi automata.

Since it became clear that the determinization step is the major hurdle in the synthesis of full LTL, a recent research trend aims at avoiding determinization by somehow integrating the synthesis procedure with lightweight 'pseudo'-determinization procedures. In [13], Kupferman and Vardi present an approach that avoids Safra's determinization and goes through universal co-Büchi word and weak alternating tree automata instead. This approach has been refined in [12] to allow compositional synthesis or to allow also controller synthesis. A major disadvantage of the approach presented in [13] regarding controller synthesis is that it suffers from an exponential blowup with respect to the size of the system under control, even if this system is deterministic. Our approach avoids this exponential blowup. Jobstmann and Bloem developed in [10] optimizations for this Safra-less approach and developed the tool 'Lily'. This tool was the first implementation that is able to synthesize designs that satisfy arbitrary LTL specifications. Although Kupferman and Vardi's approach is potentially amenable to a symbolic implementation, the tool Lily is implemented explicitly, so that is also limited to small LTL formulas only.

In practice, specifications are not given as a single large formula; instead they consist of several relatively small subformulas. In [27], an algorithm for LTL synthesis is presented that assumes that the overall specification is given as a conjunction of LTL formulas. Instead of performing determinization for the whole specification, the algorithm generates deterministic automata using the approach of [21] explicitly. Those explicitly represented automata are then encoded symbolically to obtain a generalized parity game which is then solved using the generalized parity algorithm given in [6]. Our algorithm assumes a similar setting: We also assume that the specification is a conjunction of LTL formulas. The determinization is also performed only on the nondeterministic automata obtained from these small subformulas, and the final automaton for the overall specification is obtained by combining the single deterministic automata. The main difference is however that we never represent the automata explicitly, so that we expect that our algorithm scales much better (unfortunately, the tool mentioned in [27] is not publicly available so that we can not perform comparisons).

3 Basic Definitions

3.1 Linear Temporal Logic

For a given set of Boolean variables (propositions) V_Σ, we define the set of LTL formulas by the following grammar[2]: $\varphi := V_\Sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid [\varphi \,\underline{\mathsf{U}}\, \varphi]$. Additionally, we define $\varphi \wedge \psi$, $\mathsf{F}\varphi$, $\mathsf{G}\varphi$, and $[\varphi \,\mathsf{U}\, \psi]$ as abbreviations for $\neg(\neg\varphi \vee \neg\psi)$, $[1 \,\underline{\mathsf{U}}\, \varphi]$, $\neg\mathsf{F}\neg\varphi$, and $[\varphi \,\underline{\mathsf{U}}\, \psi] \vee \mathsf{G}\varphi$, respectively.

To define the semantics of an LTL formula φ, we consider infinite sequences of truth assignments to its atomic propositions, i.e., words in $(2^{V_\Sigma})^\omega$. We denote the $(i+1)$-th element of w by w_i, i.e., $w = (w_0, w_1, \ldots)$. The semantics of LTL formulas is then defined as follows:

- for $p \in V_\Sigma$, we have $w, i \models p$ iff $p \in w_i$
- $w, i \models \neg\varphi$ iff $w, i \not\models \varphi$
- $w, i \models \varphi \vee \psi$ iff $w, i \models \varphi$ or $w, i \models \psi$
- $w, i \models \mathsf{X}\varphi$ iff $w, i+1 \models \varphi$
- $w, i \models [\varphi \,\underline{\mathsf{U}}\, \psi]$ iff there exists $k \geq i$ such that $w, k \models \psi$ and for all j with $i \leq j$ and $j < k$, we have $w, j \models \varphi$

[2] We neglect past temporal operators, although these are also available in our framework.

For a formula φ and a position $j \geq 0$ such that $w, j \models \varphi$ holds, we say that φ holds at position j of w. If $w, 0 \models \varphi$ holds, we say that φ holds on w, and write simply $w \models \varphi$.

3.2 ω-Automata on Infinite Words

Definition 1 (ω-Automata)**.** *Let Q be a finite set of state variables. Let V_Σ be a finite set of input variables disjoint from Q that defines an alphabet $\Sigma = 2^{V_\Sigma}$. Then, a ω-automaton $\mathfrak{A} = (\mathscr{S}, \mathscr{I}, \mathscr{R}, \lambda, \mathscr{A})$ over the alphabet Σ is given by a finite set of states $\mathscr{S}$, a set of initial states $\mathscr{I} \subseteq \mathscr{S}$, a transition relation $\mathscr{R} \subseteq \mathscr{S} \times \Sigma \times \mathscr{S}$, a labeling function $\lambda : \mathscr{S} \to 2^Q$ with $\lambda(s) \neq \lambda(s')$ for $s \neq s' \in \mathscr{S}$ and an acceptance component $\mathscr{A}$.*

Using standard terminology, we say that $\mathfrak{A}$ is *deterministic*, if exactly one initial state exists and for each $s \in \mathscr{S}$ and each input $\sigma \in \Sigma$ there exists exactly one $s' \in \mathscr{S}$ with $(s, \sigma, s') \in \mathscr{R}$.

The acceptance of a word is defined with respect to the set of runs: Given an automaton $\mathfrak{A} = (\mathscr{S}, \mathscr{I}, \mathscr{R}, \mathscr{L}, \alpha)$ and an infinite word $\alpha : \mathbb{N} \to \Sigma$ over Σ. Each infinite word $\beta : \mathbb{N} \to \mathscr{S}$ with $\beta^{(0)} \in \mathscr{I}$ and $\forall t. \left(\beta^{(t)}, \alpha^{(t)}, \beta^{(t+1)}\right) \in \mathscr{R}$ is called a run of α through $\mathfrak{A}$. The set of all runs of α through $\mathfrak{A}$ is defined as follows: We extend the labeling function to runs by defining $\lambda(\beta) = \lambda(\beta^{(0)})\lambda(\beta^{(1)})\ldots$. Note that $\lambda(\beta)$ is a path over Q, called the *trace* of β. Since every trace of a automaton $\mathfrak{A}$ over a word α is a path over Q, we can use LTL to specify the acceptance conditions for ω-automata. To simplify notation, we identify each run with its trace and write $\beta \models \Phi$ to mean that $\lambda(\beta) \models \Phi$. Another form of acceptance conditions are parity conditions that are conveniently defined by a priority function $\Omega : \mathscr{S} \to \{0, 1, \ldots d-1\}$ for some $d \in \mathbb{N}$. A run is accepted by a parity condition if the minimal priority seen infinitely often is even. A generalized (conjunctive) parity condition is given by k priority functions and the run is accepted iff it is accepted by each parity condition separately.

Since the states are uniquely identified by the state variables (we have $\lambda(s) \neq \lambda(s')$ for $s \neq s'$) we can represent each state (set) with a propositional formula. Hence, we can already represent the set of initial states by a propositional formula $\Phi_{\mathscr{I}}$ over $Q \cup V_\Sigma$. Introducing moreover for each variable $p \in Q$ a next-state variable p' allows us to represent also the transition relation by a propositional formula $\Phi_{\mathscr{R}}$ over $Q \cup V_\Sigma \cup \{p' \mid p \in Q\}$.

4 Exploiting the Temporal Logic Hierarchy

4.1 The Automata Hierarchy

In the past, several kinds of acceptance conditions have been proposed and their different expressivenesses have been studied in depth. In particular, the following acceptance conditions have been considered [30, 28, 25]:

- 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.

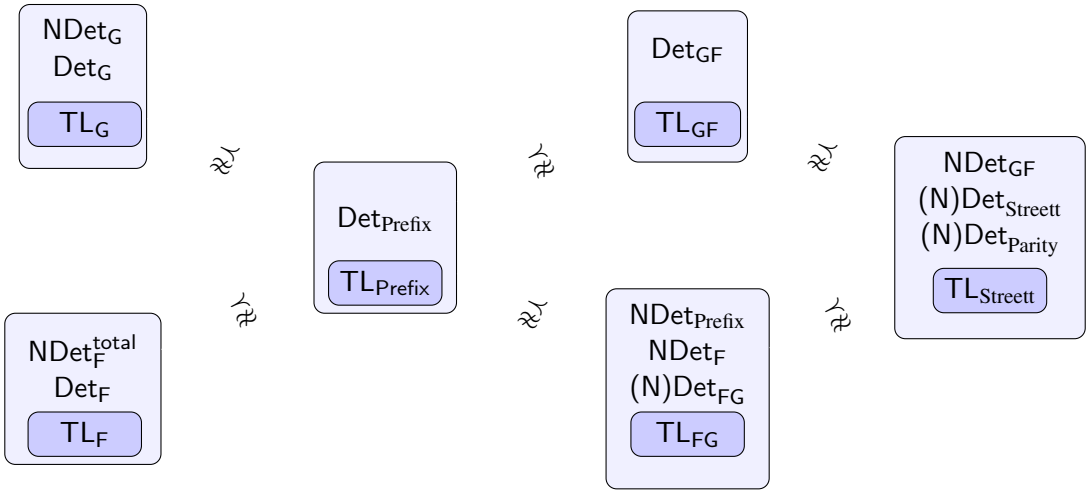


Figure 1: (Borel) Hierarchy of ω-Automata and Temporal Logic

The above 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, $(\mathsf{N})\mathsf{Det}_{\mathsf{Prefix}}$ and $(\mathsf{N})\mathsf{Det}_{\mathsf{Streett}}$ automata have acceptance conditions of the form $\bigwedge_{j=0}^{f} \mathsf{G}\varphi_j \vee \mathsf{F}\psi_j$ and $\bigwedge_{j=0}^{f} \mathsf{GF}\varphi_j \vee \mathsf{FG}\psi_j$, respectively.

The expressiveness of these classes is illustrated in Figure 1, where $\mathscr{C}_1 \precsim \mathscr{C}_2$ means that for any automaton in $\mathscr{C}_1$, there is an equivalent one in $\mathscr{C}_2$. Moreover, we define $\mathscr{C}_1 \approx \mathscr{C}_2 := \mathscr{C}_1 \precsim \mathscr{C}_2 \wedge \mathscr{C}_2 \precsim \mathscr{C}_1$ and $\mathscr{C}_1 \precnsim \mathscr{C}_2 := \mathscr{C}_1 \precsim \mathscr{C}_2 \wedge \neg(\mathscr{C}_1 \approx \mathscr{C}_2)$. As can be seen, the hierarchy consists of six different classes, and each class has a deterministic representative.

4.2 The Temporal Logic Hierarchy

In [5, 24, 25], corresponding hierarchies for temporal logics have been defined. Following [24, 25], we define the hierarchy of temporal logic formulas syntactically by the grammar rules of Fig. 2:

$P_{\mathsf{G}} ::= V_\Sigma \mid \neg P_{\mathsf{F}} \mid P_{\mathsf{G}} \wedge P_{\mathsf{G}} \mid P_{\mathsf{G}} \vee P_{\mathsf{G}} \mid \mathsf{X}P_{\mathsf{G}} \mid \mathsf{G}P_{\mathsf{G}} \mid [P_{\mathsf{G}} \mathsf{U} P_{\mathsf{G}}]$	$P_{\mathsf{F}} ::= V_\Sigma \mid \neg P_{\mathsf{G}} \mid P_{\mathsf{F}} \wedge P_{\mathsf{F}} \mid P_{\mathsf{F}} \vee P_{\mathsf{F}} \mid \mathsf{X}P_{\mathsf{F}} \mid \mathsf{F}P_{\mathsf{F}} \mid [P_{\mathsf{F}} \underline{\mathsf{U}} P_{\mathsf{F}}]$
$P_{\mathsf{Prefix}} ::= P_{\mathsf{G}} \mid P_{\mathsf{F}} \mid \neg P_{\mathsf{Prefix}} \mid P_{\mathsf{Prefix}} \wedge P_{\mathsf{Prefix}} \mid P_{\mathsf{Prefix}} \vee P_{\mathsf{Prefix}}$	
$P_{\mathsf{GF}} ::= P_{\mathsf{Prefix}} \mid \neg P_{\mathsf{FG}} \mid P_{\mathsf{GF}} \wedge P_{\mathsf{GF}} \mid P_{\mathsf{GF}} \vee P_{\mathsf{GF}} \mid \mathsf{X}P_{\mathsf{GF}} \mid \mathsf{G}P_{\mathsf{GF}} \mid [P_{\mathsf{GF}} \mathsf{U} P_{\mathsf{GF}}] \mid [P_{\mathsf{GF}} \underline{\mathsf{U}} P_{\mathsf{F}}]$	$P_{\mathsf{FG}} ::= P_{\mathsf{Prefix}} \mid \neg P_{\mathsf{GF}} \mid P_{\mathsf{FG}} \wedge P_{\mathsf{FG}} \mid P_{\mathsf{FG}} \vee P_{\mathsf{FG}} \mid \mathsf{X}P_{\mathsf{FG}} \mid \mathsf{F}P_{\mathsf{FG}} \mid [P_{\mathsf{FG}} \underline{\mathsf{U}} P_{\mathsf{FG}}] \mid [P_{\mathsf{G}} \mathsf{U} P_{\mathsf{FG}}]$
$P_{\mathsf{Streett}} ::= P_{\mathsf{GF}} \mid P_{\mathsf{FG}} \mid \neg P_{\mathsf{Streett}} \mid P_{\mathsf{Streett}} \wedge P_{\mathsf{Streett}} \mid P_{\mathsf{Streett}} \vee P_{\mathsf{Streett}}$	

Figure 2: Syntactic Characterizations of the Classes of the Temporal Logic Hierarchy

Definition 2 (Temporal Logic Classes)**.** *For* $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{FG}, \mathsf{GF}, \mathsf{Streett}\}$*, we define the logics*

TL_κ *by the grammars given in Fig. 2, where* TL_κ *is the set of formulas that can be derived from the nonterminal* P_κ *(*V_Σ *represents any variable* $v \in V_\Sigma$*).*

Typical safety conditions like $\mathsf{G}\varphi$ or $\mathsf{G}\,[a \,\mathsf{U}\, b]$ that state that something bad never happens, are contained in TL_G. Liveness conditions like $\mathsf{F}\varphi$ are contained in TL_F. Finally, fairness conditions like $\mathsf{GF}\varphi$ that demand that something good infinitely often happens, are contained in TL_GF while stabilization/persistence properties like $\mathsf{FG}\varphi$ that demand that after a finite interval, nothing bad happens are contained in TL_FG.

In our experience, almost all formulas that occur in practice belong to $\mathsf{TL}_\mathsf{Streett}$. If a given formula should not belong to $\mathsf{TL}_\mathsf{Streett}$, it is often straightforward to rewrite it to an equivalent $\mathsf{TL}_\mathsf{Streett}$ formula. For example, the formula $\mathsf{GF}\,[b \,\mathsf{U}\, a]$ that demands that infinitely often $[b \,\mathsf{U}\, a]$ holds is equivalent to the $\mathsf{TL}_\mathsf{Streett}$ formula $\mathsf{GF}\,[b \,\underline{\mathsf{U}}\, a] \vee \mathsf{FG}\,[b \,\mathsf{U}\, a]$. Clearly, there are many formulas outside $\mathsf{TL}_\mathsf{Streett}$, but we claim that these formulas seldom occur in practice.

4.3 Relating the Temporal Logic and the Automata Hierarchy

In [24, 25] several translation procedures are given to translate formulas from TL_κ to equivalent $(\mathsf{N})\mathsf{Det}_\kappa$ automata. In particular, the following is an important result:

Theorem 1 (Temporal Logic and Automaton Hierarchy)**.** *Given a formula* $\Phi \in \mathsf{TL}_\kappa$*, we can construct a deterministic* ω*-automaton* $\mathfrak{A} = (2^Q, \mathscr{I}, \mathscr{R}, \lambda, \mathscr{A})$ *of the class* Det_κ *in time* $O(2^{|\Phi|})$ *with* $|Q| \leq 2^{|\Phi|}$ *state variables. Therefore,* $\mathfrak{A} = (2^Q, \mathscr{I}, \mathscr{R}, \lambda, \mathscr{A})$ *is a symbolic representation of a deterministic automaton with* $O(2^{2^{|\Phi|}})$ *states.*

The above results are already proved in detail in [25], where translation procedures from TL_κ to NDet_κ have been constructed. Moreover, it has been shown in [25] that the subset construction can be used to determinize the automata that stem from the classes TL_G and TL_F and that the Miyano-Hayashi breakpoint construction is sufficient to determinize the automata that stem from the translation of formulas from TL_FG and TL_GF.

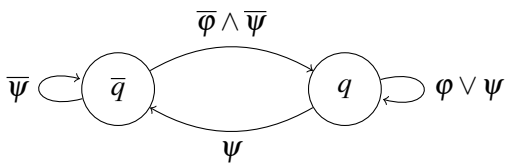
Since $\mathsf{TL}_\mathsf{Prefix}$ and $\mathsf{TL}_\mathsf{Streett}$ are the boolean closures of $\mathsf{TL}_\mathsf{G} \cup \mathsf{TL}_\mathsf{F}$ and $\mathsf{TL}_\mathsf{FG} \cup \mathsf{TL}_\mathsf{GF}$, respectively, the remaining results for $\mathsf{TL}_\mathsf{Prefix}$ and $\mathsf{TL}_\mathsf{Streett}$ follow from the boolean combinations of $\mathsf{Det}_\mathsf{G}/\mathsf{Det}_\mathsf{F}$ and $\mathsf{Det}_\mathsf{FG}/\mathsf{Det}_\mathsf{GF}$, respectively.

The final step consists of computing the boolean closure of the acceptance conditions. To this end, it is shown in [25] how arbitrary boolean combinations of $\mathsf{G}\varphi$ and $\mathsf{F}\varphi$ with propositional formulas φ are translated to equivalent $\mathsf{Det}_\mathsf{Prefix}$ automata, and analogously, how arbitrary boolean combinations of $\mathsf{GF}\varphi$ and $\mathsf{FG}\varphi$ with propositional formulas φ are translated to equivalent $\mathsf{Det}_\mathsf{Streett}$ automata.

5 Exploiting the Non-Confluence Property

It is well-known that the ω-automata that stem from LTL formulas are a special class that has already found several characterizations. Due to results of [15], the automata can be characterized as *non-counting* automata, and in terms of alternating automata, the class of *linear weak* or *very weak* automata has been defined [16, 20]. Moreover, many translation procedures from LTL generate *unambiguous automata* [4] where every accepted word has a unique accepting run [25] (although there may be additional non-accepting runs for the same word). The determinization procedure presented in this chapter makes use of the fact that the automata generated from LTL are unambiguous. Without useless states, the transition relation of an unambiguous automaton has a certain form that we call *non-confluence*:

> *An automaton is non-confluent if whenever two runs of the same infinite word meet at a state q, then they must share the entire finite prefix up to state q.*



ψ	φ	$q \leftrightarrow \psi \vee \varphi \wedge q'$
0	0	$q \leftrightarrow 0$
0	1	$q \leftrightarrow q'$
1	0	$q \leftrightarrow 1$
1	1	$q \leftrightarrow 1$

Figure 3: Nondeterministic Automaton for $[\varphi \mathsf{U} \psi]$

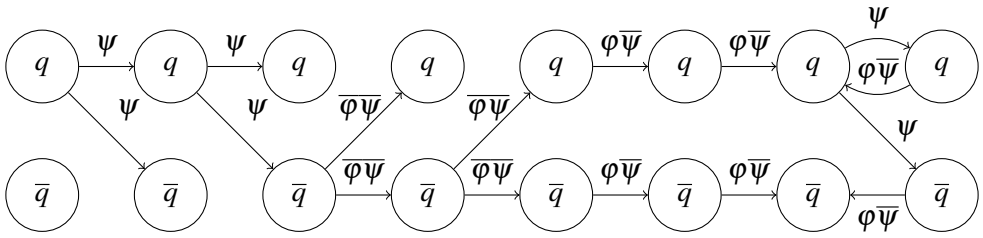


Figure 4: Run Tree with a Uniquely Determined Infinite Run

To give an intuitive idea why the automata constructed from LTL formulas by the 'standard translation' procedure are non-confluent, consider the automaton of Figure 3 that is obtained by translating the formula $[\varphi \mathsf{U} \psi]$ to a non-deterministic automaton. As explained in [24, 25], the 'standard' translation procedure from LTL to ω-automata traverses the syntax tree of the LTL formula in a bottom-up manner and abbreviates each subformula that starts with a temporal operator. The subformula $[\varphi \mathsf{U} \psi]$ is thereby abbreviated by a new state variable q, and the preliminary transition relation $\mathscr{R}$ is replaced with $\mathscr{R} \wedge (q \leftrightarrow \psi \vee \varphi \wedge q')$.

As can be seen by Figure 3, the input $\varphi \wedge \neg\psi$ demands that the current state is maintained, but allows the automaton to be in any of the two states. The other three classes of inputs uniquely determine the current state, but leave the successor state completely unspecified[3]. As a consequence, input words that infinitely often satisfy $\neg(\varphi \wedge \neg\psi)$, i.e., $\neg\varphi \vee \psi$, do only have one (infinite) run, while the remaining input words that satisfy $\varphi \wedge \neg\psi$ from a certain point of time on do have two infinite runs that are of the form ξq^ω and $\xi \overline{q}^\omega$ with the same finite prefix ξ. Hence, the automaton is non-confluent, since the two runs never merge after they have split.

An example run tree (that encodes all the runs of a given word) is shown in Fig. 4. It can be seen that there is a uniquely determined run, since all other nondeterministic choices lead to finite paths. Another example run tree that contains two infinite runs is shown in Figure 5.

As every automaton $\mathfrak{A}_\varphi$ obtained by the translation of a LTL formula φ is a product of non-confluent automata, and as the product automaton of two non-confluent automata is also non-confluent, it follows that the automata $\mathfrak{A}_\varphi$ obtained by the above 'standard' translation are non-confluent.

The above non-confluence property has been used in [18] to develop a determinization procedure that exploits symbolic set representations. In particular, it does not rely on Safra trees as used by Safra's

[3]According to the Krohn-Rhodes decomposition theorem, automata that stem from LTL properties do only have reset and identity inputs [25].

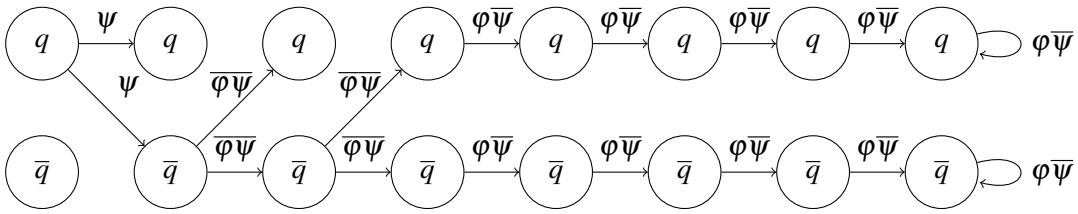


Figure 5: Run Tree with two Infinite Runs of the Automaton of Fig. 3

original procedure [23] or by the improved version of Piterman [21]. The states of the deterministic automata obtained by these procedures are trees of subsets of states of the original automaton. In contrast, our procedure generates deterministic automata whose states consist of n-tuples of subsets of states, where n is the number of states of the nondeterministic automaton.

6 A Symbolic Controller Synthesis Algorithm for Full LTL

6.1 The Averest Framework

Averest [26] is a set of tools for the specification, verification, and implementation of reactive systems. It includes a compiler and a simulator for synchronous programs, a symbolic model checker, and a tool for hardware-software synthesis. Averest can be used for modeling and verifying finite as well as infinite state systems at various levels of abstraction. In particular, Averest is not only well-suited for hardware design, but also for modeling communication protocols, concurrent programs, software in embedded systems, etc.

The design flow using Averest consists of the following steps: First, the system is described as a synchronous program in our language Quartz [26], a descendent of Esterel. Then, the program is translated to a transition system in the Averest Interchange Format (AIF) using the Quartz compiler. This intermediate description can be directly used for verification with the symbolic model checker to check whether the system satisfies its specifications. If this is the case, code can be generated for an implementation in hardware or software with output formats like VHDL, Verilog or C.

The compiler contained in Averest does not only compile a Quartz program to a corresponding transition system, but also provides procedures to translate LTL and other specification logics to symbolically represented ω-automata.

The tool implementing the features of this paper is called Opal and extends the current Averest compiler to deal with controller synthesis. In the following subsections, the different steps of Opal are described in more detail. Opal is implemented in Moscow ML and uses Moscow MLs foreign function interface to connect to the BDD-package CUDD.

6.2 Decomposing Specifications

Specifications often consist of several relatively simple components - for instance, a collection of LTL properties whose conjunction should be satisfied. Thus, we consider specifications of the form $\Phi = \bigwedge_{j=0}^{N} \Phi_j$.

Instead of translating the entire specification at once, we generate separate deterministic automata for every subformula Φ_j. Clearly, since we allow any LTL property, we may have to make use of the determinization procedure outlined in [18] to translate Φ_j. For example, this is the case if the top-level

operator of Φ_j is a temporal operator and Φ_j does not belong to one of the lower Borel classes $\mathsf{TL}_{\mathsf{GF}}$ or $\mathsf{TL}_{\mathsf{FG}}$.

In practice, this is however nearly never the case. Instead, most often, also the subformulas Φ_j are a boolean combination of even smaller formulas. Instead of handling them all at once, we break also these into smaller parts so that we obtain for every Φ_j a collection $\varphi_1 \dots \varphi_k$ of LTL properties that all start with a temporal operator, and that either belong to one of the classes TL_{G}, TL_{F}, $\mathsf{TL}_{\mathsf{GF}}$, $\mathsf{TL}_{\mathsf{FG}}$ or to none of these classes.

6.3 Handling Safety and Liveness formulas

Safety formulas φ_i are first translated to a nondeterministic safety automaton. Although safety automata can not be directly transformed to a parity automaton, it is possible to minimize them using direct simulation[4]. After minimization, we perform the ordinary subset construction and afterwards minimize the automaton again using the direct simulation relation. However, one important subclass of properties does not scale well using this approach. Since many specifications are of the form 'if something now happens, in the next step something else happens', we treat this subclass separately. This subclass can be formally described by Boolean combinations of formula of the form ψ or of the form $\mathsf{X}\xi$ where both ψ and ξ are Boolean formulas over the input variables.

In that case, every X operator doubles the state space of the non-deterministic automaton and thus leads to a blow-up in the number of BDD variables of the deterministic automaton. Even worse, the simulation relations can neither minimize the nondeterministic nor the deterministic automaton since the two states that occur due to an occurrence of $\mathsf{X}a$ can not be equivalent as one of the two will lead to a non-satisfying loop. Thus, the basic translation procedure really suffers from a double-exponential blowup. Instead, we translate those formulas by abbreviating each variable a that is not under the scope of a X operator by a previous variable a_p such that ${a_p}' \leftrightarrow a$ holds and replace any subformula $\mathsf{X}a$ with a.

Proposition 1. *Given a formula $\mathsf{G}\varphi$ where φ is a boolean combination of state variables $v \in V_\Sigma$ and formulas $\mathsf{X}v$ where $v \in V_\Sigma$. Then $\mathsf{G}\varphi$ is initially equivalent to the symbolically represented deterministic automaton $\mathfrak{A} = (2^Q, \Phi_{\mathscr{I}}, \Phi_{\mathscr{R}}, \mathscr{L}, \mathscr{A})$ where $Q = \{a_p \mid a \in V\} \cup \{p\}$, $\Phi_{\mathscr{I}} = p \wedge \bigwedge_{a \in V_\Sigma} \neg a_p$, $\mathscr{A} = \mathsf{G}(p)$ and the transition relation is defined by*

$$\Phi_{\mathscr{R}} = \bigwedge_{a \in V_\Sigma} {a_p}' \leftrightarrow a \wedge p' \leftrightarrow \varphi'$$

Here φ' is obtained from φ by replacing any occurrence of a' with a and any occurrence of a formula $a \in V$ with a_p. Moreover $\mathscr{L} : 2^Q \to 2^Q$ is the identity function.

Obviously, this leads to an automaton that has at most $|a| + 1$ state variables, thus the automaton is only exponential in the size of the specification. Nevertheless, if many subformulas a exist but only little subformulas $\mathsf{X}a$, the ordinary translation may give better results, so that this translation is optional in our algorithm.

For liveness formulas φ_i, we translate $\neg\varphi_i$ and dualize the corresponding deterministic safety automaton to obtain a deterministic liveness automaton.

[4]Indeed, the nondeterministic automata generated by the translation procedures given in [25] translates the safety fragment to an automaton with the trivial acceptance condition $\mathsf{G1}$, so that even the ordinary simulation relation as described in [8] could be applied.

6.4 Handling Co-Büchi and Büchi specifications

For co-Büchi specifications, we use the translation from $\mathsf{TL_{FG}}$ to nondeterministic co-Büchi automata, minimize this automaton using the minimization techniques of [9]. Afterwards this automaton is determinized using the breakpoint construction and again minimized. Büchi specifications are translated using the dual deterministic automaton of the formula obtained from negating the formula.

6.5 Handling LTL Formulas that do not belong to a lower Borel Class

If a subformula Φ_i does not belong to one of the lower Borel classes, we have to resort on the determinization procedure from [18]. Thus, we first translate Φ_i to a non-confluent Büchi automaton and minimize it. This nondeterministic automaton is determinized using the procedure described in [18]. However, we do not need to construct the whole automaton at once. Instead, we construct the automaton for a fixed bound k and check whether every marking of a state has been noticed by a state set. If so, we return this automaton, otherwise we do the same with an increased bound. Afterwards, we use the minimization techniques of [9] to minimize the obtained parity automaton.

Although this procedure gives back a parity automaton that is from a theoretical point of view more efficient than a Street automaton, the heavy complexity of determinization makes even this approach infeasible in practice. Thus, we break up also the formulas Φ_i into smaller parts until every subformula starts with a temporal operator. Those subformulas are then translated as explained before. It is well known that every parity automaton can also be interpreted as a Streett or a Rabin automaton. We thus interpret the obtained automaton as a Streett automaton and combine the deterministic Streett automata to obtain a Streett automaton for Φ_i that is afterwards translated to a generalized parity automaton.

6.6 Solving Generalized Parity Games

Instead of solving the whole generalized parity game at once, we first solve the subgames that are obtained by constructing the game for the subformula Φ_j. The set of states that are loosing for the controller need not be considered in the overall game. Afterwards, we solve the reduced overall game using the generalized parity game algorithm of [6].

6.7 Generating Circuits from BDDs

The output of the generalized parity algorithm is a BDD over the (current state) variables $V_u, V_c, \mathscr{S}$ and over newly introduced state variables V_M to encode counter variables that are used to switch between the sub-strategies calculated by the generalized parity algorithm. A slight modification of the algorithm given in Figures 2 and 3 of [3] allows us to generate for every controllable input variable c a BDD φ_c with the meaning that c should hold whenever φ_c holds. We then write those BDDs to a file in our Averest interchange format. The tool Topaz in our Averest toolset can be used to obtain either Verilog, VHDL or C code from the generated file.

7 Experiments and Conclusion

This section describes the experiments performed using the controller synthesis algorithm described in this paper. To this end, the 23 specifications that come with the Lily tool [10] are used as a benchmark set. Those 23 handwritten formulas are mostly traffic light examples or arbiters.

No.	T,B,AP	subf	O	NoncDetT	OpalDetT	NoncT	OpalT	LilyT
L1	12,5,4	3	1	0.59	0.12	0.62	0.14	0.13
L2	12,5,4	3	1	0.64	0.13	0.66	0.16	0.12
L3	12,5,4	1	1	0.71	0.13	0.79	0.16	0.65
L4	16,10,4	1	1	0.72	0.1	0.77	0.15	1.31
L5	20,11,4	1	1	0.72	0.12	0.84	0.2	0.98
L6	18,14,4	1	1	0.71	0.14	0.9	0.2	1.98
L7	16,15,4	1	1	0.74	0.13	0.81	0.19	0.87
L8	4,1,2	1	1	0.01	0	0.02	0.01	0.07
L9	8,6,2	1	1	0.12	0.03	0.16	0.06	0.22
L10	6,3,2	1	1	0.02	0.02	0.02	0.03	0.39
L11	4,3,2	1	1	0.17	0.02	0.18	0.03	0.66
L12	5,4,2	1	1	0.21	0.02	0.22	0.03	0.19
L13	4,3,2	1	2	0.01	0.02	0.04	0.04	0.03
L14	9,3,4	3	2	0.01	0	0.02	0.03	0.26
L15	9,5,4	5	4	0.21	0.03	0.29	0.1	0.18
L16	15,9,6	9	6	0.28	0.04	0.36	0.14	1
L17	13,5,5	6	3	0.01	0	0.05	0.04	0.37
L18	21,10,7	10	4	0	0.01	0.11	0.11	1.67
L19	10,7,4	1	1	0.25	0.04	0.33	0.14	2.46
L20	17,20,5	1	1	0.11	0.02	0.2	0.11	4.38
L21	40,38,8	1	1	0.29	0.04	1.48	0.32	8.11
L22	22,18,4	1	1	0.32	0.06	0.43	0.12	9.98
L23	8,5,2	1	1	0.09	0.01	0.11	0.03	0.28

Figure 6: Experiments performed with different Determinization Constructions

To analyze the effect of using the all-purpose determinization of [18] instead of the much simpler breakpoint or subset construction, we performed three different experiments on each of the formulas where the runtimes are summarized in Figure 6. The first column lists the identification number of the example. The second column gives some measures of the specification in terms of number of temporal operators, boolean operators, and the number of input variables. The third column gives the number of subformulas, i. e. the number of top-level conjunctions. The conjunction of those formula form the overall specification. Notice that the number of boolean operators do not include the top-level conjuncts. The next column indicates how many subformulas are not safety formulas[5]. The next two columns give the time for determinization followed by the overall time needed to synthesize the specification.

Algorithm Opal uses all optimizations. Every formula is translated with the lowest possible determinization procedure. In contrast, algorithm Nonc translates every formula that is not a safety formula with the all-purpose determinization construction of [18]. Finally, the last column gives the running time of the Lily algorithm.

As it can be seen, using "'easier"' determinization procedures like the Rabin-Scott subset construc-

[5]In the considered experiments, any subformula is at least contained in the Büchi or co-Büchi class so that we obtain a formula of the temporal logic hierarchy.

tion or the Miyano-Hayashi breakpoint construction significantly improves the determinization step itself. When the size of the specification grows as in examples L20-L23, the overall synthesis time is no longer dominated by the determinization, but instead by the time to solve the generalized parity game. In that case using the easier determinization procedures leads to smaller sized automata and thus also to an improvement on the overall synthesis time. With the exception of the small specifications L1 and L2, our tool is significantly faster compared to the explicitly implemented tool Lily.

References

[1] R. Armoni, S. Egorov, R. Fraer, D. Korchemny & M.Y. Vardi (2005): *Efficient LTL compilation for SAT-based model checking*. In: *International Conference on Computer-Aided Design (ICCAD)*, ACM/IEEE Computer Society, San Jose, California, USA, pp. 877–884.

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

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

[4] O. Carton & M. Michel (2003): *Unambiguous Büchi automata*. *Theoretical Computer Science (TCS)* 297(1-3), pp. 37–81.

[5] E.Y. Chang, Z. Manna & A. Pnueli (1992): *Characterization of Temporal Property Classes*. In: W. Kuich, editor: *International Colloquium on Automata, Languages and Programming (ICALP)*, *LNCS* 623, Springer, Vienna, Austria, pp. 474–486.

[6] K. Chatterjee, T.A. Henzinger & N. Piterman (2007): *Generalized Parity Games*. In: H. Seidl, editor: *Foundations of Software Science and Computational Structures (FOSSACS)*, *LNCS* 4423, Springer, Braga, Portugal, pp. 153–167.

[7] K. Chatterjee, T.A. Henzinger & N. Piterman (2007): *Strategy Logic*. In: L. Caires & V.T. Vasconcelos, editors: *Conference on Concurrency Theory (CONCUR)*, *LNCS* 4703, Springer, Lisbon, Portugal, pp. 59–73.

[8] K. Etessami, T. Wilke & R.A. Schuller (2005): *Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata*. *SIAM Journal on Computing* 34(5), pp. 1159–1175.

[9] C. Fritz (2005): *Simulation-Based Simplification of omega-Automata*. Ph.D. thesis, Technischen Fakultät der Christian-Albrechts-Universität zu Kiel, Germany.

[10] B. Jobstmann & R. Bloem (2006): *Optimizations for LTL Synthesis*. In: A. Gupta & P. Manolios, editors: *Formal Methods in Computer-Aided Design (FMCAD)*, IEEE Computer Society, San Jose, California, USA, pp. 117–124.

[11] J. Klein & C. Baier (2006): *Experiments with Deterministic ω-Automata for Formulas of Linear Temporal Logic*. *Theoretical Computer Science (TCS)* 363(2), pp. 182–195. Http://dx.doi.org/10.1016/j.tcs.2006.07.022.

[12] O. Kupferman, N. Piterman & M.Y. Vardi (2006): *Safraless Compositional Synthesis*. In: T. Ball & R.B. Jones, editors: *Computer Aided Verification (CAV)*, *LNCS* 4144, Springer, Seattle, Washington, USA, pp. 31–44.

[13] O. Kupferman & M.Y. Vardi (2005): *Safraless Decision Procedures*. In: *Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, pp. 531–540.

[14] L.H. Landweber (1969): *Decision problems for ω-automata*. *Mathematical Systems Theory* 3(4), pp. 376–384.

[15] R. McNaughton & S. Papert (1971): *Counter-free Automata*. MIT Press.

[16] S. Merz & A. Sezgin (2003): *Emptiness of Linear Weak Alternating Automata*. Technical Report, LORIA.

[17] S. Miyano & T. Hayashi (1984): *Alternating automata on ω-words*. Theoretical Computer Science (TCS) 32, pp. 321–330.

[18] A. Morgenstern & K. Schneider (2008): *From LTL to Symbolically Represented Deterministic Automata*. In: F. Logozzo, D.A. Peled & L.D. Zuck, editors: Verification, Model Checking, and Abstract Interpretation (VMCAI), *LNCS* 4905, Springer, San Francisco, California, USA, pp. 279–293.

[19] A. Morgenstern, K. Schneider & S. Lamberti (2008): *Generating Deterministic ω-Automata for most LTL Formulas by the Breakpoint Construction*. In: C. Scholl & S. Disch, editors: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV), Shaker, Freiburg, Germany, pp. 119–128.

[20] D.E. Muller, A. Saoudi & P.E. Schupp (1986): *Alternating automata, the weak monadic theory of the tree, and its complexity*. In: L. Kott, editor: International Colloquium on Automata, Languages and Programming (ICALP), *LNCS* 226, Springer, Rennes, France, pp. 275–283.

[21] N. Piterman (2006): *From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata*. In: Symposium on Logic in Computer Science (LICS), IEEE Computer Society, Seattle, Washington, USA, pp. 255–264.

[22] M.O. Rabin & D. Scott (1959): *Finite automata and their decision problems*. IBM Journal of Research and Development 3, pp. 115–125.

[23] S. Safra (1988): *On the complexity of ω-automata*. In: Symposium on Foundations of Computer Science (FOCS), pp. 319–327.

[24] K. Schneider (2001): *Improving Automata Generation for Linear Temporal Logic by Considering the Automata Hierarchy*. In: R. Nieuwenhuis & A. Voronkov, editors: International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), *LNAI* 2250, Springer, Havana, Cuba, pp. 39–54.

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

[26] K. Schneider (2009): *The Synchronous Programming Language Quartz*. Internal Report 375, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany.

[27] S. Sohail, F. Somenzi & K. Ravi (2008): *A Hybrid Algorithm for LTL Games*. In: F. Logozzo, D.A. Peled & L.D. Zuck, editors: Verification, Model Checking, and Abstract Interpretation (VMCAI), *LNCS* 4905, Springer, San Francisco, California, USA, pp. 309–323.

[28] W. Thomas (1990): *Automata on Infinite Objects*. In: J. van Leeuwen, editor: Handbook of Theoretical Computer Science, B: Formal Models and Semantics, chapter 4, Elsevier, pp. 133–191.

[29] T. Tuerk & K. Schneider (2005): *Relationship between Alternating omega-Automata and Symbolically Represented Nondeterministic omega-Automata*. Internal Report 340, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany.

[30] K. Wagner (1979): *On ω-regular sets*. Information and Control 43(2), pp. 123–177.