Translations of LTL to ω-automata

Report for the "Embedded Systems and Robotics" Seminar (WS 2024/25)

Flavia Tego
Fachbereich Informatik, RPTU Kaiserslautern-Landau

Abstract. The development of more complex computational systems, which interact continuously with the environmental stimuli, has brought to our attention how specific, the system behaviour should be for the user. Considering the high level specifications for inputs, outputs and temporal behaviour of the systems, reactive synthesis procedure is taken into account. Temporal relationships can be specified through different specification logics, among which linear temporal logic (LTL) is in focus. Furthermore, centering on the verification aspect of reactive systems, it is important to expand on the translation from LTL to ω-automata. Significant progress has been made on this aspect with the latest research exploring optimization techniques, alternative representations, such as use of intermediate automata representations (e.g Limit-Deterministic Büchi Automata, LDBA) for deterministic parity automata (DPA) construction and even reverse translation problem from ω-automata to LTL formulas. Integrating the newest research insights [4, 9, 10, 24], we can represent a more structured framework for LTL to ω-automata translations, with direct implications for improving the scalability of formal verification and synthesis tools.

1. Introduction

Linear Temporal Logic LTL [22] and ω-automata [26], as tools to specify and analyze the system behaviour over infinite executions, play a central role in formal verification, model checking [2] and reactive synthesis. In order to achieve an automated reasoning about the system correctness, translations methods or processes have been proposed and developed (as in [3, 27, 14]). However, during this process, several issues related to state-space explosion, efficiency limitations, and the complexity of determinization, have arisen. Recent research has proposed new techniques to alleviate these issues. The proposed methods vary from normalization procedures [10, 24] to intermediate automata representations [9], and even the reverse problem of translating automata back into LTL [4].

As mentioned previously, model checking and formal verification rely heavily on the translation of LTL expressions into equivalent automata, be it alternating, deterministic or nondeterministic automata. Since the translation process gives us the chance to automatically verify systems for their complex temporal specifications, it is of importance to dive in the optimization of translations techniques with regard to expressiveness and efficiency. Traditional approaches usually involve intricate constructions among which Safra's determinization procedure [23] is used. Our focus, in this paper, will be on four different perspectives of the translation procedure based on the latest findings.

Firstly, in the translation process, an important aspect to address is the complexity of LTL formulas themselves. Often, LTL formulas contain redundancies or their structure is quite complex, making their translations to an automata inefficient. To improve on the automata construction, we refer to the normalization of the input LTL formulas. *Sickert and Esparza* [24] introduce a systematic method for normalizing LTL formulas before translation into automata. The procedure is based on the use of Very Weak Alternating Automata (VWAA) as an intermediate automaton. In the newest paper, *Esparza, Rubio, and Sickert* [10] go even further in the normalization process by introducing a rewrite system that formalizes and automates the simplification of LTL formulas.

On the other hand, looking at another step of the translation procedure, especially on the construction of deterministic automata, *Esparza et al.* [9] proposes a two step process. The traditional approach is to convert an LTL formula into a deterministic parity automata (DPA). Being computationally expensive, *Esparza et al.* [9] addresses this by simplifying the determinization process. The proposition is to translate LTL into limit-deterministic Büchi automata (LDBA), followed by a transformation into DPA. This intermediate transformation bypasses the state-space explosion problem.

Lastly, we take a look into a less explored aspect, the reverse translation from an automaton to an LTL formula. Normally, we look at the translation LTL-to-automata, however the reverse translation is equally important for debugging, specification inference, and synthesis. *Boker, Lehtinen, and Sickert* [4] offer insights to help bridge the gap between logic and automata, enabling better introspection into the structure of automata. Furthermore, this translation can be optimized to generate more readable LTL formulas, which is beneficial in formal verification for understanding system properties.

The remainder of this paper is structured as follows: Section 2 offers an overview of LTL and au-

tomata definitions, Section 3 will focus on two normalization techniques, Section 4 looks into constructing efficient automata by using as intermediate automata limit-deterministic Büchi automata, Section 5 diverges from the LTL-to-automata translation, to reverse translation. At the end, we summarize and offer suggentions for future work in Section 6 and 7, respectively.

2. Preliminaries

2.1 LTL Definitions

In this section, we will outline a part of the theoretical background needed with the focus on brief definitions for the linear temporal logic, hierarchies and types of automata.

Given a finite alphabet Σ, a word w over Σ is an infinite sequence of letters $a_0a_1a_2...$ with $a_i \in \Sigma$ for all $i \geq 0$, while a language is a set of words denoted as $\Sigma^{\varphi}(\Sigma^*)$. $w[i]$ where $i = 0$, depicts the i-th letter of a word w. We abbreviate the infinite infix $w[i]w[i+1]...w[j-1]$ to $w_{i,j}$ and the infinite infix $w[i]w[i+1]...$ to w_i. The infinite repetition of a finite word $\sigma_1...\sigma_n$ is shown as $(\sigma_1...\sigma_n)^{\varphi} = \sigma_1...\sigma_n\sigma_1...\sigma_n\sigma_1...$

Definition 1. LTL formulas over a set of atomic propositions *Ap* are established with the following syntax:

$$\varphi ::= tt \mid ff \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$
$$\mid \mathsf{X}\varphi \mid \varphi\ \mathsf{U}\ \varphi \mid \varphi\ \mathsf{W}\ \varphi \mid \varphi\ \mathsf{R}\ \varphi \mid \varphi\ \mathsf{M}\ \varphi$$

where $a \in Ap$ is an atomic proposition and the temporal operators $\mathsf{X}, \mathsf{U}, \mathsf{W}, \mathsf{R}, \mathsf{M}$ are respectively the operators for next, (strong) until, weak until, (weak) release, and strong release.

For the semantics, we define the usual sematics:

Definition 2. Given a word w over the alphabet 2^{Ap} and a given formula φ, we inductively define the satisfaction relation $w \vDash \varphi$ as the smallest relation satisfying:

$w \vDash tt$
$w \vDash a$ *iff* $a \in w[0]$
$w \vDash \neg a$ *iff* $a \notin w[0]$
$w \vDash \varphi \wedge \psi$ *iff* $w \vDash \varphi$ *and* $w \vDash \psi$
$w \vDash \varphi \vee \psi$ *iff* $w \vDash \varphi$ *or* $w \vDash \psi$
$w \vDash \mathsf{X}\varphi$ *iff* $w_1 \vDash \varphi$
$w \vDash \varphi\mathsf{U}\psi$ *iff* $\exists k.w_k \vDash \psi$ *and* $\forall j < k.w_j \vDash \varphi$
$w \vDash \varphi\mathsf{M}\psi$ *iff* $\exists k.w_k \vDash \varphi$ *and* $\forall j \leq k.w_j \vDash \psi$
$w \vDash \varphi\mathsf{R}\psi$ *iff* $\forall k.w_k \vDash \psi$ *or* $w \vDash \varphi\mathsf{M}\psi$
$w \vDash \varphi\mathsf{W}\psi$ *iff* $\forall k.w_k \vDash \varphi$ *or* $w \vDash \varphi\mathsf{U}\psi$

The language of φ is denoted by $\mathsf{L}(\varphi) \coloneqq \{\mathsf{w} \in (2^{\mathsf{Ap}})^{\omega} : \mathsf{w} \vDash \varphi\}$. By overloading the definition of $\vDash$, we can write $\varphi \vDash \psi$ as a shorthand for $\mathsf{L}(\varphi) \subseteq \mathsf{L}(\psi)$.

The standard abbreviations $\mathsf{F}\varphi \coloneqq tt\ \mathsf{U}\ \varphi$ (eventually) and $\mathsf{G}\varphi \coloneqq ff\ \mathsf{R}\ \varphi$ (always).

The equivalence of formulas and the equivalence within the languages is defined as follows:

Definition 3. Two given formulas φ and ψ are equivalent $\varphi \equiv \psi$, if $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$. Given a language $\mathcal{L} \subseteq (2^{Ap})^{\omega}$, two formulas φ and ψ are equivalent within the given language L, $\varphi \equiv^{\mathsf{L}} \psi$, if $\mathcal{L}(\varphi) \cap \mathsf{L} = \mathcal{L}(\psi) \cap \mathsf{L}$.

2.2 The Safety-Progress Hierarchy

To navigate the process of the research from [10, 24], in regard to the normalization precedure and later on the use of Very Weak Alternating Automata, it is necessary to be aware of the hierarchy of temporal properties observed by Manna and Pnueli [18] and formulated by Černá and Pelánek [6].

Definition 4. ([6, 10, 18, 24]) Let $\mathsf{P} \subseteq \Sigma^{\omega}$ be a property over Σ.

- **Safety property:** A property P is considered a safety property if there exists a set of finite words $\mathsf{L} \subseteq \Sigma^*$ such that every finite prefix of any word $w \in \mathsf{P}$ belongs to L.
- **Guarantee property:** A property P is a guarantee property if there exists a set of finite words $\mathsf{L} \subseteq \Sigma^*$ such that for every word $w \in \mathsf{P}$, at least one finite prefix of w belongs to L.
- **Obligation property:** A property P is a obligation property if it can be described as a positive boolean combination of safety and guarantee properties.
- **Recurrence property:** A property P is a recurrence property if there exists a set of finite words $\mathsf{L} \subseteq \Sigma^*$ such that for every word $w \in \mathsf{P}$, infinitely many prefixes of w belongs to L.
- **Persistence property:** A property P is a persistence property if there exists a set of finite words $\mathsf{L} \subseteq \Sigma^*$ such that for every word $w \in \mathsf{P}$, all but a finite number of its prefixes belong to L.
- **Reactivity property:** A property P is a reactivity property if it can be described as a positive boolean combination of recurrence and persistence properties.

The relationships between these classes are illustrated in Figure 1.a. Chang, Manna, and Pnueli provide a syntactic characterization of the safety-progress hierarchy classes using fragments of LTL in [18].

Definition 5. [24] The classes of LTL formulas are depicted as follows:

- $\Sigma_0 = \Pi_0 = \Delta_0$: This is the smallest set that includes all atomic propositions and their negations, and it is closed under conjunction and disjunction.
- Σ_{i+1}: This is the smallest set containing Π_i, closed under conjunction, disjunction, and the operators X (next), U (until), and M (strong until).

- Π_{i+1}: This is the smallest set containing Σ_i, closed under conjunction, disjunction, and the operators X (next), R (release), and W (weak until).

- Δ_{i+1}: This is the smallest set containing both Σ_{i+1} and Π_{i+1} and closed under conjunction and disjunction.

According to [18, 24], an LTL property is a guarantee, safety, obligation, persistence, recurrence, or reactivity property if and only if it can be specified by a formula from the classes $\Sigma_1, \Pi_1, \Delta_1, \Sigma_2, \Pi_2$ or Δ_2, respectively.

2.3 Automata Definitions

A *Büchi automaton* (specifically a nondeterministic word automaton with Büchi acceptance) is defined as a tuple $\mathcal{A} = (\mathcal{Q}, q_0, \Sigma, \delta, \alpha)$, where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, Σ represents a finite alphabet, $\delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is the transition relation defining valid state transitions, and $\alpha \in \delta$ is the set of accepting transitions. The automaton is called *deterministic* if, for every state $q \in \mathcal{Q}$ and every $\sigma \in \Sigma$, there is at most one state $q' \in \mathcal{Q}$ such that $(q, \sigma, q') \in \delta$. In other words, for each state-symbol pair, there is at most one valid transition or none at all. For a set of states $S \subseteq \mathcal{Q}$ and a given $\sigma \in \Sigma$, the post-transition set is defined as $post^{\sigma}_{\delta}(S) = \{q' \mid \exists q \in S.(q, \sigma, q') \in \delta\}$. If δ is clear from the context, as a shorthand annotation for $(q, \sigma, p) \in \delta$, $p \rightarrow^{\sigma} q$ is used.

A run of a Büchi automaton $\mathcal{A}$ on an ω-word w: $\mathbb{N} \rightarrow \Sigma$ is defined as an ω-sequence of states $\rho : \mathbb{N} \rightarrow \mathcal{Q}$, such that $\rho(0) = q_0$ and $\forall i \in \mathbb{N}$, the transition $(\rho(i), w(i), \rho(i+1))$ holds. A *run* ρ is *accepting* if there are infinitely many $i \in \mathbb{N}$ such that the transition is $(\rho(i), w(i), \rho(i+1)) \in \alpha$. The language defined by $\mathcal{A}$, L(A), consists of all ω-words w for which the automaton has an accepting run.

A *limit-deterministic Büchi automaton* (LBDA) is defined as a tuple $\mathcal{A} = (\mathcal{Q}, q_0, \Sigma, \delta, \alpha)$. The automaton behaves nondeterministically in the initial part of the run and then after a finite number of steps, the automaton enters a deterministic mode where it behaves like a deterministic Büchi automaton (DBA). A run is considered accepting if it eventually remains in the deterministic part and infinitely often visits the accepting states.

A *parity automaton* is a deterministic word automaton $\mathcal{A}$ defined, the same as a deterministic Büchi automaton, by a tuple $(\mathcal{Q}, q_0, \Sigma, \delta, p)$, with the difference that p is a function that assigns an integer from the set 1,2,...,d, called a colour, to each transition in the automaton. The colours are naturally ordered according to their integer values. The acceptance condition is defined based on the parity of the smallest colour that appears infinitely often during a run.

An *alternating Büchi word automaton (AWW)* is a tuple $\mathcal{A} = \langle \Sigma, Q, \theta_0, \delta, \alpha \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $\theta_0 \in B^+(Q)$ is an initial formula, $\delta : Q \times \Sigma \rightarrow B^+(Q)$ is a transition function, $\alpha \subseteq Q$ is the acceptance condition, indicating the set of accepting states. A run of $\mathcal{A}$ on a word w is represented as a directed acyclic graph $G = (V, E)$ satisfying:

1. $V \subseteq Q \times \mathbb{N}_0$ and $E \subseteq \bigcup_{l \geq 0}((Q \times \{l\}) \times (Q \times \{l+1\}))$
2. There exists a minimal model S of θ_0 such that $(q, 0) \in V$ if and only if $q \in S$
3. For each $(q, l) \in V$, the set $\{q' : ((q, l), (q', l+1)) \in E\}$ must be a minimal model of $\delta(q, w[l])$
4. For every $(q, l) \in V \setminus (Q \times \{0\})$, there exists $q' \in Q$ such that $((q', l-1), (q, l)) \in E$

A run is accepting if $\delta(q, w[l]) \not\equiv \text{ff}$ for every $(q, l) \in V$ and every infinite path in G visits α-nodes infinitely often. An automaton $\mathcal{A}$ accepts a word w if there exists an accepting run on w. The language $\mathcal{L}(\mathcal{A})$ recognized by $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$.

A *weak automaton* $\mathcal{A} = \langle \Sigma, Q, \theta_0, \delta, \alpha \rangle$ is an alternating (co-)Büchi automaton with the following properties:
There exists a partition $Q_1, \ldots, Q_m$ of Q such that:

1. For every $q, q' \in Q$, if $q \rightarrow q'$ (i.e., there exists a $a \in \Sigma$ such that q' belongs to a minimal model of $\delta(q, a)$), then there exist indices $i \leq j$ such that $q \in Q_i$ and $q' \in Q_j$
2. For each $1 \leq i \leq m$, $Q_i \subseteq \alpha$ or $Q_i \cap \alpha = \emptyset$

A *very weak automaton* is a weak automaton where every class Q_i in the partition is a singleton, i.e., $|Q_i| = 1$.

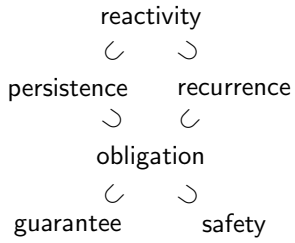
AWW (weak alternating automata) and A1W (very weak alternating automata) are the abbreviations for weak and very weak alternating automata, respectively. In the context of weak automata with co-Büchi acceptance conditions, it is possible to define a Büchi acceptance condition that recognizes the same language, so weak automata are assumed to have a Büchi acceptance condition for convenience.

A weak alternating automaton $\mathcal{A}$ is said to have height n if every path in $\mathcal{A}$ alternates at most $n-1$ times between α and $Q \setminus \alpha$. The height defines how "deep" the alternation between accepting and non-accepting states can be. The set $AWW[n]$ (or $A1W[n]$) consists of weak (or very weak) automata with height at most n.

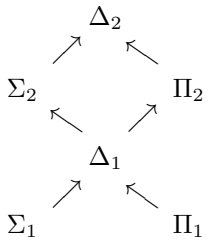
3. LTL Normalization: Preprocessing for Efficient Translation

3.1 Motivation for LTL Normalization

The efficient translation of LTL formulas to ω-automata is an important part in formal verification and reactive synthesis. However, direct translation can result in an exponential blow-up in the number



(a) Safety-progress hierarchy [18]



(b) Syntactic-future hierarchy

Figure 1: Side-by-side hierarchies highlighting the correspondence established by ([24], Thm. 6)

of states of the resulting automation [1], where in the worst case we may be faced with a double exponential complexity $O(2^{2^{|n|}})$ or even a non-elementary one. The complexity of nested temporal operators, redundant or logically equivalent subformulas or the combination of future and past operators may lead to a worst case complexity.

To alleviate this issue, latest findings have brought forward normalization techniques which simplify LTL formulas before the automata, construction leading to smaller resulting automata. Two key approaches in LTL normalization are:

- A rewrite system that systematically simplifies LTL formulas [10]
- An alternative method using Very Weak Alternating Automata (VWAA) for an intermediate representation [24]

3.2 An Efficient LTL Normalization Procedure

Esparza and Sickert in [24], develop a straightforward algorithm for converting LTL formulas into deterministic Rabin automata (DRW). Firstly, the formula is normalized, then it gets translated into a special very weak alternating automaton where a simple determinisation procedure is applied, which is valid only for these special automata. To reiterate, using the proposed normalization procedure, every formula Δ_2 is proved to be able to be translated into a very weak alternating Büchi automaton (A1W), where every path has at most one alternation between accepting and non-accepting states. Afterwards, a very simple determinisation procedure, based on breakpoint construction, is used. The whole process is described as the translation of LTL-to-DRW. Since, the desired automaton, at the end of the translation is a DRW, then A1W is considered the intermediate automaton.

3.2.1 LTL Normal Form

We take into consideration, as the first step towards the translation LTL-to-DRW, the normal form of LTL formulas. Hence, the following definitions are acknowledged:
Consider an LTL formula ϕ defined over a set of atomic propositions A_p, a new normal form is defined in terms of two key concepts:

- A **partition of the universe** $\mathcal{U} \coloneqq (2^{A_p})^\omega$ consists of equivalence classes of words that exhibit the same "limit behavior" with respect to the LTL formula φ. Specifically, words within the same equivalence class will satisfy similar conditions concerning the formula's future behavior.
- **Stable word** with respect to φ: A word w is considered stable with respect to φ if the subformulas of φ exhibit predictable, stable behavior over time. The use of stable words is necessary in simplifying and improving the structure of the LTL formula, making it more amenable to the DRW translation.

The behavior of an LTL formula, over time will be depended on the limit behavior of its subformulas, where $\mu(\phi)$ depicts a set of subformulas of the form $\psi_1 U \psi_2$ or $\psi_1 M \psi_2$ (future or until operators) and $\nu(\phi)$, a set of subformulas of the form $\psi_1 W \psi_2$ or $\psi_1 R \psi_2$ (weak or release operators).
For any word w, define:

$$\mathcal{GF}^{\varphi}_{w} \coloneqq \{\psi : \psi \in \mu(\varphi) \wedge w \models \mathrm{GF}\psi\}$$

$$\mathcal{FG}^{\varphi}_{w} \coloneqq \{\psi : \psi \in \nu(\varphi) \wedge w \models \mathrm{FG}\psi\}$$

Two words w, v have the *same limit behavior* w.r.t. φ if:

$$\mathcal{GF}_w = \mathcal{GF}_v \quad \text{and} \quad \mathcal{FG}_w = \mathcal{FG}_v$$

Hence, an equivalence relation is defined, which partitions the universe U into equivalence classes of words, having the same limit behavior:

$$\mathcal{P}_{M,N} \coloneqq \{w \in \mathcal{U} \mid M = \mathcal{GF}_w \wedge N = \mathcal{FG}_w\}$$

with $M \subseteq \mu(\varphi)$, $N \subseteq \nu(\varphi)$ and $\mathcal{P}_{M,N} \subseteq \mathcal{U}$

The second perception, mentioned at the beginning of this section is the concept of a *stable word*. A word w is regarded as stable w.r.t. φ if:

- Every formula in $\mu(\varphi)$ holds either infinitely often or never.
- Every formula in $\nu(\varphi)$ fails either infinitely often or never.

While not all words are stable, every word will eventually stabilize. Thus, we define the set of stable words $\mathcal{S}_\varphi$, as $\mathcal{S}_\varphi := \{w \in \mathcal{U} : \mathcal{F}^\varphi_w = \mathcal{G}\mathcal{F}^\varphi_w \wedge \mathcal{G}^\varphi_w = \mathcal{F}\mathcal{G}^\varphi_w\}$.

Since we want to derive a normal form for LTL formulas using concepts of partition and stable words, then it is necessary to introduce the subsequent formulas:

- $\varphi[M]^\Pi_1$ defines a formula from the set $M \subseteq \mu(\varphi)$, where we include the temporal operators U and M.
- $\varphi[N]^\Sigma_1$ defines a formula from the set $N \subseteq \nu(\varphi)$, where the temporal operators R and W

Then we can deduct the normal form for stable words as:

$$\phi \equiv^{S_\varphi} \bigvee_{\substack{M\subseteq\mu(\varphi)\\ N\subseteq\nu(\varphi)}} \left(\varphi[M]^\Pi_1 \wedge \bigwedge_{\psi\in M} \mathsf{GF}(\psi[N]^\Sigma_1) \wedge \bigwedge_{\psi\in N} \mathsf{FG}(\psi[M]^\Pi_1)\right)$$

However, as already stated, not all the words are stable, therefore a need for a generalized normal form arises, for which Esparza and Sickert [24] propose an extension of the above normal form:

$$\varphi \equiv \bigvee_{\substack{M\subseteq\mu(\varphi)\\ N\subseteq\nu(\varphi)}} \left(\varphi[M]^\Sigma_2 \wedge \bigwedge_{\psi\in M} \mathsf{GF}(\psi[N]^\Sigma_1) \wedge \bigwedge_{\psi\in N} \mathsf{FG}(\psi[M]^\Pi_1)\right)$$

Based on Section 6.1, in [24], Sickert and Esparza prove that the complexity of the normalization procedure has at most most single exponential blowup in the length of the formula.

3.2.2 Translation of LTL to DRW

The translation of Linear Temporal Logic (LTL) to Deterministic Rabin Automata (DRW) involves a structured sequence of intermediate translations and constructions. This process ensures that every LTL formula can be represented by a DRW of double exponential size.

Translation of LTL to A1W[2]

The process begins with the translation of LTL formulas into Alternating Automata of rank 1 with at most one alternation (A1W). In the standard translation, the states of the A1W for a formula φ correspond to its subformulas or their negations. However, to ensure that the resulting automaton belongs to the class A1W[2], the construction is slightly modified:

- *State definition*: States are defined as pairs of the form $\langle\psi\rangle_\Gamma$, where ψ is a proper subformula of ϕ (i.e., not a Boolean constant, conjunction, or disjunction) and Γ is the smallest class in the syntactic-future hierarchy containing ψ, excluding the zeroth level.
- *Transition structure*: The transition relation is designed to ensure that transitions are only allowed between formulas at the same or lower level in the syntactic-future hierarchy and accepting states correspond to Π_i subformulas, ensuring at most one alternation between Σ and Π states.
- *Disambiguation of ambiguous levels*: Some formulas belong to multiple levels of the hierarchy. For example, Xa belongs to both Σ_1 and Π_1. In such cases, multiple states are introduced to represent the ambiguity (e.g., both $\langle Xa\rangle_{\Sigma_1}$ and $\langle Xa\rangle_{\Pi_1 1}$ are valid states).

By this construction, the resulting A1W[2] automaton for a formula in Δ_i belongs to A1W[i]. The number of states in the automaton is bounded by $2^{|sf(\varphi)|}$, where $sf(\varphi)$ is the set of proper subformulas of φ.

Determinization of AWW[2]

The next step involves determinizing the A1W[2] automaton into a deterministic co-Büchi automaton (DCW). The determinization procedure is based on a breakpoint construction adapted from prior work on alternating automata.

- *Input automaton*: The input is an Alternating Weak Automaton of rank 2 (AWW[2]).
- *State definition*: A state in the resulting DCW is a pair $(Levels, Promising)$, where $Levels \subseteq 2^Q$ represents the current level in the run of the automaton, while $Promising \subseteq Levels \cap a$ tracks the accepting levels that can still produce an accepting run.
- *Transition function*: Upon reading an input symbol if $Promising$ is non-empty, the next state is computed by propagating the successors of the promising levels, however if $Promising$ is empty, the α-levels of the next state are added to Promising.
- *Acceptance condition*: The co-Büchi condition ensures that if the Promising set becomes empty infinitely often, the word is rejected; otherwise, it is accepted.

The deterministic automaton produced by this step has at most 3^{2^n} states, where n is the number of states in the input AWW[2].

Construction of DRW

To obtain a deterministic Rabin automaton (DRW), the deterministic co-Büchi automaton is further refined:

- For each initial model of the starting formula θ_0, a separate DRW is constructed.
- Each individual DRW has at most 2^{2n+2} states and a single Rabin pair.
- The final DRW is obtained by taking the union of the individual DRWs, resulting in at most $2^{2n+\log_2 m+2}$ states and m Rabin pairs, where m is the number of minimal models of the initial formula.

Final Translation
Finally, the LTL formula is normalised and converted to the DRW using the established translation steps:

$$\phi \equiv \bigvee_{M \subseteq \mu(\phi)} \bigwedge_{N \subseteq \nu(\phi)} \phi_{M,N}$$

where $\phi_{M,N}$ is translated into an A1W[2], determinised into a DCW, and converted into a DRW. The resulting DRW has a double-exponential number of states in the size of the input LTL formula.

3.3 A Rule-Based Rewrite System for LTL

Past research from Lichtenstein, Pnueli, and Zuck [17] introduced us to the classification of LTL properties, which was later extended from Manna and Pnueli [18, 19], in what is known as the *safety-progress* hierarchy. According to it, LTL formulas are extended with past temporal operators. The *progress-safety* hierarchy has a safety class of formulas, and five progress classes, from which, the largest class, known as the *reactivity* class, holds all properties expressible in LTL. Manna and Pnueli prove that every *reactivity* property can be expressed in the form of $\mathsf{GF}\varphi \vee \mathsf{FG}\psi$, where φ and ψ have only past temporal operators.

In 1992, Chang, Manna, and Pnueli [7] presented a similar normal form, using standard LTL (without past operators), with only future operators X (next), U (until) and W (weak until), proving the **Normalization Theorem**. According to the later, every reactivity formula is equivalent to an LTL formula, written in negation normal form, where every path would have at most one alternation of operators U and W in the syntax tree. Furthermore, based on the notation [6, 20, 24], which parallels the definition of Σ_i, Π_i and Δ_i classes in arithmetical and polynomial hierarchies, they demonstrated that every LTL formula corresponds to an Δ_2-formula.

Even though normal forms play an influencial part in model checking and verification, the complex and indirect normalization procedures show impracticality in implementation. Zuck's proof [29] for the *Normalization Theorem* [7], based on the 1985 theorem [17], involves automaton translation and the Krohn-Rhodes decomposition, leading to a non-elementary blow-up with little progress in simplifying it. In 2020, Sickert and Esparza offered a new proof for the *Normalization Theorem* [24]. The proof shows that every LTL formula φ can be rewritten as Δ_2 formula as $\bigvee_{M \subseteq \psi(\varphi), N \subseteq \nu(\varphi)} \varphi_{M,N}$, where *M* and *N* are sets of subformulas with top operators in U, M and W, R, respectively. This leads to a single-exponential normalization procedure, used in [24, 25] for translating LTL formulas into deterministic and limit-deterministic ω-automata. However, the algorithm still has exponential complexity in the best case since it considers all possible sets M and N. To solve the aforementioned problem, Esparza, Rubio, and Sickert[10] present a normalization procedure for LTL formulas, similar to converting a Boolean formula to conjunctive normal form (CNF). Just as CNF ensures no conjunctions are below disjunctions, the LTL procedure imposes constraints on the order of temporal operators. Rewritten rules are used to remove nodes that violate these constraints, analogous to distributing conjunctions over disjunctions in CNF.

The main idea of the proposed normalization form is to treat the temporal operators GF and FG as atomic limit operators GF and FG, focusing on their behavior "in the limit." Hence, the syntax of LTL formulas is extended as follows:

Definition 6. *Extended LTL formulas over a set Ap of atomic propositions are produced using the following syntax:* $\varphi ::= tt \mid ff \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \varphi \,\mathsf{W}\, \varphi \mid \mathsf{GF}\varphi \mid \mathsf{FG}\varphi$.

The proposed normal form, more strict than Δ_2 formulas, is defined in [10]:

Definition 7. *Let φ be an LTL formula. A node of T_φ is a limit node if it is either a GF-node or a FG-node. The formula φ is in normal form if T_φ satisfies the following properties:*

1. No **U-node** appears under a W-node.
2. No limit node is beneath another temporal node.
3. No **W-node** is under a GF-node, and no **U-node** is under a FG-node.

Referred as *temporal* formulas, the U-, W-, X-, GF-, FG-formulas are, respectively, forms of the formulas $\varphi \,\mathsf{U}\, \psi, \varphi \,\mathsf{W}\, \psi, \mathsf{X}\,\varphi, \mathsf{GF}\,\varphi, \mathsf{FG}\,\varphi$. The syntax tree T_φ of a formula φ is defined as usual, and $|\,\varphi\,|$ designates the number of nodes of T_φ. If the subformula rooted at a node of T_φ is a U-formula, then the node is known as U-node. Analogously, we defined the other temporal nodes of W, GF and FG.

The normalization algorithm will apply the rules from Table 1 to correct any violations of the defined properties:

1. **Removing U-nodes under W-nodes**
   Rules (1) and (2) are applied to eliminate U-nodes beneath W-nodes unless they are under limit nodes. Applying rule (2) only to the highest U-nodes of φ_1 ensures that the number of new GF-subformulas remains linear relative to the original formula size.
2. **Extracting limit nodes under other temporal nodes**
   Rules (3) and (4) pull out limit nodes from beneath other temporal nodes. Since the rules are applied only to the lowest limit nodes, each limit subformula needs to be processed just once.
3. **Handling W-nodes under GF-nodes and U-nodes under FG-nodes** Rule (5) removes W-nodes under GF-nodes, and rule (6) removes U-nodes under FG-nodes. This might create smaller limit nodes, which are handled recursively. By targeting the highest W- and U-nodes, the algorithm limits the blowup in formula size to a single exponential increase.

Stage 1:	(1)	$\varphi_1 \,\mathsf{W}\, \varphi_2\, [\psi_1 \,\mathsf{U}\, \psi_2] \equiv \varphi_1 \,\mathsf{U}\, \varphi_2 [\psi_1 \,\mathsf{U}\, \psi_2] \;\vee\; \mathsf{G}\varphi_1$
	(2)	$\varphi_1[\psi_1 \mathsf{U} \psi_2]\mathsf{W}\, \varphi_2 \equiv (\mathsf{GF}\psi_2 \wedge \varphi_1[\psi_1 \mathsf{W}\, \psi_2]\mathsf{W}\, \psi_2) \vee \varphi_1\, [\psi_1 \,\mathsf{U}\, \psi_2] \,\mathsf{U}\, (\varphi_2 \vee \mathsf{G}\varphi_1[\mathsf{ff}])$
Stage 2:	(3)	$\varphi[G\mathsf{F}\psi] \equiv (\mathsf{GF}\psi \wedge \varphi[\mathsf{tt}]) \vee \varphi[\mathsf{ff}]$
	(4)	$\varphi[F\mathsf{G}\psi] \equiv (\mathsf{FG}\psi \wedge \varphi[\mathsf{tt}]) \vee \varphi[\mathsf{ff}]$
Stage 3:	(5)	$G\mathsf{F}\varphi[\psi_1 \mathsf{W} \psi_2] \equiv \mathsf{GF}\varphi[\psi_1 \mathsf{U} \psi_2] \vee (\mathsf{FG}\psi_1 \wedge \mathsf{GF}\varphi[\mathsf{tt}])$
	(6)	$F\mathsf{G}\varphi[\psi_1 \mathsf{U} \psi_2] \equiv (\mathsf{GF}\psi_2 \wedge \mathsf{FG}\varphi[\psi_1 \mathsf{W} \psi_2]) \vee \mathsf{FG}\varphi[\mathsf{ff}])$

Table 1: Normalization rules [10]

After following the steps above, the formula is transformed into normal form, resulting in a single exponential increase in the number of nodes.

So far, only temporal operators U and W, have been taken into consideration, exluding operators R and M. The later can be expressed using other operators, hence they are not included in the proof and normalization procedure. Nonetheless, directly handling them improves efficiency, as their translation exponentially increases the number of nodes. Their roles during the procedure are similar to those of the U and W operators, with R treated like W and M like U.

A formula is in dual normal form if it satisfies conditions 2 and 3 of Definition 7, including that no W-node is under a U-node. For a given formula φ, its dual normal form can be obtained by negating it, converting it into negation normal form, and then pushing the negation into the formula to produce an equivalent formula in dual normal form.

Apart from the standard LTL, there is also Past LTL, an extension of LTL that includes past operators. Gabbay introduced a set of rewrite rules to transform past operators into future operators [11]. Combining these rules with the proposed normalization procedure from Esparza, Rubio, and Sickert[10] enables the transformation of a past LTL formula into a normalized LTL formula, where past operators are isolated into past-only subformulas and treated as atomic propositions.

4. Constructing Efficient Automata

We are faced with a significant challenge when it comes to synthesizing controllers for general Linear Temporal Logic (LTL) objectives. The conventional method requires converting the LTL objective into a deterministic parity automaton (DPA) using the Safra-Piterman construction [21]. A major difficulty lies in the size of the resulting DPA, which can grow very quickly, often reaching double-exponential complexity in the length of the LTL formula. Latest findings from Esparza et al. [9] introduce a more efficient alternative: a single-exponential translation from limit-deterministic Büchi automata (LDBA) to DPA. This translation can be combined with a recently developed efficient LTL-to-LDBA translation, resulting in a double-exponential "Safra-less" LTL-to-DPA construction.

4.1 From LDBA to DPA

The proposed construction a DPA from a LDBA is based on the use of Run Directed Acyclic Graphs (Run DGAs), originated from [15].

4.1.1 Structure of Run DGA

As observed from its name, a Run DAG is a graph structure that represents all possible runs of an automaton of a given word w. Each run will correspond to a sequence of states visited by the automaton as it runs through the word. The graph is built by layers of V_i, where each layer v_i contains vertices to represent the states at a time i. Layers are connected together via edges, where the later portrays a transition from one state to another, based on the transition relation of the given automaton. Given an LDBA $\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_d, q_0, \Sigma, \delta, \alpha)$, where $Q_d \subseteq Q$ stands for the set of deterministic states of the automaton, we can construct the following graph $G_w = (V, E)$:

- **Vertices V** represent pairs (q, i), where $q \in Q$ is a state and i depicts a time step.
- **Edges E** connect the vertices V if there exists a transition between states, i.e, $(q, w(i), q') \in \delta$, where $w(i)$ is the input at position i.

The function $\mathtt{Ord}$: $Q \to \{1, 2, \ldots, |Q_d|, +\infty\}$ is regarded an ordering of the states of A w.r.t Q_d if $\mathtt{Ord}$ specifies a strict total order on $q \in Q_d$ and maps each state $q \in \overline{Q_d}$, where $\overline{Q_d}$ denotes $Q \backslash Q_d$, to $+\infty$ as:

- $\forall q \in \overline{Q_d},\ \mathtt{Ord}(q) = +\infty$
- $\forall q \in Q_d,\ \mathtt{Ord}(q) \neq +\infty$
- $\forall q,\ q' \in Q_d,\ \mathtt{Ord}(q) = \mathtt{Ord}(q') \Rightarrow q = q'$

Extending $\mathtt{Ord}$ to the vertices in G_w, allows to define pre-order relation on the set of run prefixes of the run

DAG G_w.
We need to mention an important aspect of run DGA, which is the colouring of the transition between vertices. The colours assigned to each transition, are represented as "positive" (even colours) or "negative" (odd colours):

- **Even** colours are used for accepting transitions.
- **Odd** colours are used for transitions that abort or lead to smaller runs, marking this as a more negative event.

The specific colour assignment depends on the following:

- In case there is no abort run and we have an accepting transition, an even colour is assigned based on the smallest index of an accepting vertex.
- In case there are abort runs present and no accepting transitions, which is known as negative event, hence an odd colour is assigned based on the smallest index of an aborting vertex.
- If both events occur, meaning there is an accepting transition and an abort run, then the colour is decided by the minimum of the two events.

4.2 Construction of DPA

Given an LDBA $A = (\mathcal{Q}, \mathcal{Q}_d, q_0, \Sigma, \delta, \alpha)$ and an ordering function $\texttt{Ord}$: $\mathcal{Q} \to \{1, 2, \ldots, |\mathcal{Q}_d|, +\infty\}$ compatible with $\mathcal{Q}_d$, we define a deterministic parity automaton $B = (\mathcal{Q}^B, q_0^B, \Sigma, \delta^B, p)$, that simulates the previously described run DGA and the colouring. The construction should yield a DPA which accepts the same language as the original given LDBA. Subsequently, the DPA B is described as [9]:

1. *States and initial state.* The set of states $\mathcal{Q}^B = \mathcal{P}(\overline{\mathcal{Q}_d}) \times O\mathcal{P}(\mathcal{Q}_d)$, where $\mathcal{P}(\overline{\mathcal{Q}_d})$ is a set of subsets of $\overline{\mathcal{Q}_d}$ and $O\mathcal{P}(\mathcal{Q}_d)$, is a set of totally ordered subsets of $\mathcal{Q}_d$. A state in DPA is a pair $(s, (t, <))$ with s being a set of states outside $\mathcal{Q}_d$, t is an ordered subset of $\mathcal{Q}_d$ and $<$ represents the strict total order on the elements in t. The initial state is $q_0^B = (\{q_0\}, (\{\}, \{\}))$.
2. *Transition function.* Given $(s_1, (t_1, <_1)) \in \mathcal{Q}^B$ and $\sigma \in \Sigma$, let us presume that there is a state $q \in s_1 \cup t_1$ and a state $q' \in \mathcal{Q}$ so that $(q, \sigma, q') \in \delta$. Then $\delta_B((s_1, (t_1, <_1)), \sigma) = (s_2, (t_2, <_2)$ where:
   - $s_2 = post_\delta^\sigma(s_1) \cap \overline{\mathcal{Q}_d}$
   - $t_2 = post_\delta^\sigma(s_1 \cup t_1) \cap \overline{\mathcal{Q}_d}$
   - The new order $<_2$ on t_2 is defined based on the previous order $<_1$ and the order function $\texttt{Ord}$:
     - If neither state has a predecessor in $<_1$, order them using $\texttt{Ord}$
     - If one state has a predecessor in $<_1$ and the other does not, the one with a predecessor is ranked higher.
     - If both states have predecessors, they are ordered according to the order of their minimal predecessors in $<_1$.

Afterwards, we have to take care of the colouring definition. A colour is assigned to transitions depending on the properties fulfilled by the transitions taken into consideration. We start the colouring process by identifying two sets of states:

- $\texttt{Dec}(t_1)$, which includes states $q_1 \in t_1$ where either:
  - The indices associated with state q_1 decrease with the transition t_1 (with ordering $<_1$) to t_2 (with ordering $<_2$), i.e., $Ind(t_2, <_2)(\delta(q_i, \sigma)) < Ind(t_1, <_1)(q_i)$.
  - There exists no successor state for the transition with input σ, as indicated by $\neg\exists(q_1, \sigma, q)$.
- $\texttt{Acc}(t_1)$, consisting of accepting transitions from the subset of states in t_1 to a subset of states in t_2, i.e, $\texttt{Acc}\ (t_1) = \{q | \exists q' \in t_2 : (q, \sigma, q') \in \alpha\}$.

The colouring rules are then applied based on the following rules [9]:

1. If $\text{Dec}(t_1) = \emptyset$ and $\text{Acc}(t_1) \neq \emptyset$, the colour is $2 \cdot \min_{q \in \text{Acc}(t_1)} \text{Ind}_{(t_1, <_1)}(q)$
2. If $\text{Dec}(t_1) \neq \emptyset$ and $\text{Acc}(t_1) = \emptyset$, the colour is $2 \cdot \min_{q \in \text{Dec}(t_1)} \text{Ind}_{(t_1, <_1)}(q) - 1$
3. If $\text{Dec}(t_1) = \emptyset$ and $\text{Acc}(t_1) = \emptyset$, the colour is decided by taking the minimum among
   - $c_{\text{odd}} = 2 \cdot \min_{q \in \text{Dec}(t_1)} \text{Ind}_{(t_1, <_1)}(q) - 1$
   - $c_{\text{even}} = 2 \cdot \min_{q \in \text{Acc}(t_1)} \text{Ind}_{(t_1, <_1)}(q)$
4. If $\text{Dec}(t_1) = \text{Acc}(t_1) = \emptyset$, the colour is $2 \cdot |\mathcal{Q}_d| + 1$

Based on Theorem 2 in [9], the language recognized by both LDBA and DPA remains consistent. As per this statement, if we are able to achieve a correct translation, we can translate any LDBA into an equivalent DPA, where both types of automata recognize the same language.

4.3 Translating LTL to DPA

After going through the process of translating a LDBA in a DPA, we have to discuss its relevance to our main goal of translating an LTL formula to DPA. The approach from Esparza et al. [9] is the use of an intermediate step of translation to LDBA first, and then achieving the DPA automaton. The standard method of translating LTL to DPA is by using the Safra-Piterman construction[21], through the involvement of a non-deterministic Büchi automaton (NBA) as the intermediate step. However, this method can lead to a DPA size that is doubly exponential. To improve the translation, there are two recommended methods which utilize the translation to LDBA as an intermediate step and specific properties of the language structure of these automata. The methods include:

Pruning by Language Decomposition. This technique aims at identifying parts of the LDBA, to be later simplified based on their language characteristics, more specifically safety languages and suffix-closed languages. As a result, we can reduce the number of nodes while still preserving important information for acceptance decisions. We achieve reduced run DGAs with fewer vertices which ensues smaller DPAs, in comparison to standard constructions.

Pruning by Language Subsumption. Here, we take an oracle-based approach where the identification of redundant vertices is based on the inclusion relationships between the recognized languages at different states within the automaton. Hence, we allow further simplification without concurring information loss for accepted runs.

While the use of traditional translation methods can lead to a double exponential size blowup, the improved methods by [9] on the conversion of a given LTL formula to an efficient representation through LDBA, using techniques based on [25], shows that a single-exponential growth can be achieved.

5. Reverse translation: From automata to LTL

The several methods described until now, take a look at the different techniques of LTL formulas' translation to automata. However, the work from Boker, Lehtinen and Sickert [4] offer a view on the reverse translation, from general counter-free deterministic ω-regular automata into LTL formulas, using an intermediate Krohn-Rhodes cascade decomposition of the automaton. Until now, no other literature, has given insights on the non-elementary upper bound on the size blowup for the reverse translation.

Before, we dive into the reverse translation, the investigation of the succinctness of different automata models [13], when translating them into LTL over a unary alphabet, lays the groundwork for the later part of the reverse translation. A unary alphabet consists of only one symbol, such as a, which allows for a focused analysis on languages formed by repetitions of that single symbol. Hence, it is in our interest to compare automata's ability to recognize specific patterns and extends it to arbitrary languages over this simplified setting.

Based on Theorem 1 [4], built on Propositions 1-5 [28] and Lemmas 1-3 [16, 5], Boker, Lehtinen and Sickert establish that deterministic finite automata (DFAs), nondeterministic finite automata (NFAs), and alternating finite automata (AFAs) exhibit distinct size blow-ups when translated into LTL formulas: DFAs result in linear growth, NFAs yield quadratic growth, while AFAs lead to exponential growth in size relative to their state count. The results are established using an adaptation of Leiss's construction [16] and two-way deterministic automata for counting [8, 12]. Based on the previous results, we can generalize the complexity results about automata over general alphabets, demonstrating that the complexity increases to double-exponential depth and triple-exponential length when using a Krohn-Rhodes decomposition-based translation.

To this end, Boker, Lehtinen and Sickert expand on the translation of counter-free ω-regular automata into LTL formulas, with the main focus on reset cascade automata derived from Krohn-Rhodes-Holonomy decomposition. As our primary objective, we want to construct parameterized LTL formulas that capture reachability conditions within these cascades.

However, we are face with two challenges: firstly, how do we capture past states Two significant challenges arise in this context: first, capturing past states through cascading transitions while only being able to express future properties in LTL and secondly, how are we able to represent internal states of cascaded semiautomata without direct support in LTL syntax. To tackle these issues, auxiliary reachability formulas are defined inductively based on levels within the cascade structure and utilize both strong and weak temporal operators.

The depth and length analysis reveals that as one progresses through levels of configurations within reset cascades—where each level can have multiple states—the resulting nested structures lead to exponential growth rates concerning both temporal nesting depth and overall length when expressed as LTL formulas.

6. Applications and Experimental Results

In this paper, we focus on investigating the latest methods for translating LTL formulas to automata and the practical applications in model checking and formal verification. Each of the four papers, considered as the foundation of this work, offer their own applications and experimental results.

According to Esparza et al. [9], we have a new take on the translation of LTL to DPA with the intermediate translation to a limit deterministic Büchi automaton. The work is directly applicable to automata-based model checking, which is widely used in both hardware and software verification tasks. The paper provides a foundation for improving the efficiency of model checking by translating LTL formulas into deterministic automata that are easier to handle computationally. As shown by the experiments conducted, the proposed translation from LDBAs to deterministic parity automata reduces the complexity of model checking, in comparison to other translation methods. Furthermore, it performs better in both time and memory usage for large LTL formulas and automata.

Sickert and Esparza, in [24], propose an efficient normalization procedure for Linear Temporal Logic (LTL), by introducing the use of very weak alternating automata as an intermediate step in the normalization process. The experimental results show that using very weak alternating automata allows for faster processing of LTL formulas, particularly when dealing with large or complex formulas. The authors compare their method with existing normalization techniques and find that

it performs significantly better in terms of execution time, making it highly suitable for real-world verification tasks.

Expanding on their previous work, Esparza, Rubio and Sickert, in [10], propose normalization of LTL formulas to a canonical form, before the conversion to automata, through rewriting system. The experiments showed a significant reduction of formula size and the improvement of the performance for the subsequent steps, such as the translation into automata. Besides, this approach was shown to be time-efficient and scalable. The paper's approach is aimed at improving the scalability and efficiency of model checking by converting LTL formulas into automata that are easier to process and analyze.

Boker, Lehtinen and Sickert [4], address the challenge of reverse translation of converting automata models to LTL specifications. Until now no insights was given on the non-elementary upper bound on the size blowup for the reverse translation, hence this work brings an important understanding for the topic.

7. Conclusion and Future Work

To summarize, the latest normalization and translations techniques show an improvement on efficiency and scalability, as they outperform previous methods in terms of execution time and memory usage, when dealing with complex systems with numerous temporal properties.

Future work will focus on further exploring the use of the proposed techniques different classes of LTL formulas. Additionally, it would be interesting to see how these methods can be integrated into existing model-checking tools so they could be applied to industrial-scale software verification tasks.

As final remarks, we need to mention, that due to space constraints for this paper, only a general outline of the whole research of the referenced papers was taken into consideration. For further information, on the proofs, lemmas, theorems, experimental methods and used applications, it is advised to consult the original papers, like [4, 9, 10, 24] and other references to have a better understanding of the information presented on this work.

References

[1] Rajeev Alur and Salvatore La Torre. Deterministic generators and games for ltl fragments. *ACM Transactions on Computational Logic (TOCL)*, 5(1):1–25, 2004.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[3] U. Boker, O. Grumberg, and D. Harel. From linear temporal logic to automata on infinite words. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV 2004)*, pages 172–183. Springer, 2004.

[4] U. Boker, K. Lehtinen, and S. Sickert. On the translation of automata to linear temporal logic. In P. Bouyer and L. Schröder, editors, *Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 13242 of *LNCS*, pages 140–160, Munich, Germany, 2022. Springer.

[5] Uri Boker and Orna Kupferman. The quest for a tight translation of büchi to co-büchi automata. In *Fields of Logic and Computation*, pages 147–164. Springer, 2010.

[6] Ivana Černá and Radek Pelánek. Relating hierarchy of temporal properties to model checking. In Branislav Rovan and Peter Vojtáš, editors, *Mathematical Foundations of Computer Science 2003*, pages 318–327, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[7] Edward Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In W. Kuich, editor, *Automata, Languages and Programming*, pages 474–486, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[8] Michał Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.

[9] J. Esparza, J. Kretínský, J.-F. Raskin, and S. Sickert. From linear temporal logic and limit-deterministic Büchi automata to deterministic parity automata. *International Journal on Software Tools for Technology Transfer (STTT)*, 24(4):635–659, 2022.

[10] J. Esparza, R. Rubio, and S. Sickert. A simple rewrite system for the normalization of linear temporal logic. In J.-F. Raskin, K. Chatterjee, L. Doyen, and R. Majumdar, editors, *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, volume 13660 of *LNCS*, pages 208–227. Springer, 2022.

[11] Dov Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification: Altrincham, UK, April 8–10, 1987 Proceedings*, pages 409–448. Springer, 1989.

[12] Vincent Geert, Carlo Mereghetti, and Giuseppe Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007.

[13] Orna Kupferman, Amnon Ta-Shma, and Moshe Y. Vardi. Counting with automata. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 13–22. IEEE, 1999.

[14] Orna Kupferman and Mosab Vardi. From ltl to büchi automata. *Information and Computation*, 164(2):322–344, 2001.

[15] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Logic*, 2(3):408–429, July 2001.

[16] Eckehard Leiss. Succinct representation of regular languages by boolean automata. *Theoretical Computer Science*, 13(3):323–330, 1981.

[17] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. The glory of the past. In Rohit Parikh, editor, *Logics of Programs*, pages 196–218, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[18] Zohar Manna and Amir Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC '90, page 377–410, New York, NY, USA, 1990. Association for Computing Machinery.

[19] Zohar Manna and Amir Pnueli. Completing the temporal picture. In *Selected Papers of the 16th International Colloquium on Automata, Languages, and Programming*, page 97–130, NLD, 1991. Elsevier Science Publishers B. V.

[20] Radek Pelánek and Jan Strejček. Deeper connections between ltl and alternating automata. In Jacques Farré,

Igor Litovsky, and Sylvain Schmitz, editors, *Implementation and Application of Automata*, pages 238–249, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[21] Nir Piterman. From nondeterministic buchi and streett automata to deterministic parity automata. *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 255–264, 2006.

[22] Amir Pnueli. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.

[23] Saharon Safra. On the complexity of ω-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–327. IEEE Computer Society, 1988.

[24] S. Sickert and J. Esparza. An efficient normalisation procedure for linear temporal logic and very weak alternating automata. In *Logic in Computer Science (LICS)*, pages 831–844. IEEE Computer Society, 2020.

[25] Salomon Sickert. *A Unified Translation of Linear Temporal Logic to ω-Automata.* PhD thesis, Technische Universität München, 2019.

[26] Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 133–191. Elsevier and MIT Press, 1990.

[27] Mosab Vardi and Pierre Wolper. An automata-theoretic approach to linear temporal logic. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science (LICS 1995)*, pages 240–250. IEEE, 1995.

[28] Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

[29] Lenore Zuck. *Past temporal logic*. The Weizmann Institute of Science (Israel), 1986.