A Direct Translation from LTL with Past to Deterministic Rabin Automata

Shaun Azzopardi ✉ 🆔
University of Malta, Msida, Malta

David Lidell ✉ 🆔
University of Gothenburg, Sweden

Nir Piterman ✉ 🆔
University of Gothenburg, Sweden

Abstract

We present a translation from linear temporal logic *with past* to deterministic Rabin automata. The translation is direct in the sense that it does not rely on intermediate non-deterministic automata, and asymptotically optimal, resulting in Rabin automata of doubly exponential size. It is based on two main notions. One is that it is possible to encode the history contained in the prefix of a word, as relevant for the formula under consideration, by performing simple rewrites of the formula itself. As a consequence, a formula involving past operators can (through such rewrites, which involve alternating between weak and strong versions of past operators in the formula's syntax tree) be correctly evaluated at an arbitrary point in the future without requiring backtracking through the word. The other is that this allows us to generalize to linear temporal logic with past the result that the language of a pure-future formula can be decomposed into a Boolean combination of simpler languages, for which deterministic automata with simple acceptance conditions are easily constructed.

1 Introduction

Finite-state automata over infinite words, commonly referred to as ω-automata, have been studied as models of computation since their introduction in the 1960s. Their introduction was followed by extensive investigations into their expressive power, closure properties, and the decidability and complexity of related decision problems, such as non-emptiness and language containment. In particular, it was soon established that determinization constructions of such automata are considerably more complex than they are for their finite word counterparts, for which a simple subset construction is sufficient.

In the 1970s, Pnueli [15] proposed linear temporal logic (*LTL*) as a specification language for the analysis and verification of programs, based on the idea that a program can be viewed in the context of a stream of interactions between it and its environment. It has since become ubiquitous in both academia and the industry due to the perceived balance of its expressive power and the computational complexity of its related decision procedures. The

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 13; pp. 13:1–13:15

Leibniz International Proceedings in Informatics
LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

intimate semantic connection between *LTL* and ω-automata further motivated research into the properties of both, also from a practical point of view. In addition to being of theoretical interest, the fact that multiple automata-based applications of *LTL* as a specification language – such as reactive synthesis and probabilistic model checking – require deterministic automata, has drawn additional attention to the study of efficient translation procedures from *LTL* to deterministic ω-automata.

In the classic approach to determinization, the given *LTL* formula is first translated to a non-deterministic Büchi automaton. This automaton is subsequently determinized, for example using Safra's procedure [16], or the more modern Safra/Piterman variant [13]. Such constructions are both conceptually complex and difficult to implement. Moreover, information about the structure of the given formula is lost in the initial translation to Büchi automata; in particular, because of the generality of such determinization procedures, they cannot take advantage of the fact that *LTL* is less expressive than ω-automata.

In 2020, Esparza et al. [5] presented a novel translation from *LTL* to various automata, that is asymptotically optimal in both the deterministic and non-deterministic cases. The translation is direct in the sense that it avoids the intermediate steps of the classic approach, which involve employing a variety of separate translation procedures. In particular, for deterministic automata, it forgoes Safra-based constructions. Instead, the language of the formula under consideration is decomposed into a Boolean combination of simpler languages, for which deterministic automata with simple acceptance conditions can easily be constructed using what the authors have dubbed the "after-function"; that such a decomposition exists is a fundamental result named the "Master Theorem". These simpler automata are then combined into the desired final automaton using basic product or union operations according to the structure of the decomposition.

In this paper, we consider past linear temporal logic (*pLTL*); the extension of *LTL* that includes the past operators "Yesterday" and "Since", analogous to the standard operators "Next" and "Until", respectively. We adapt the Master Theorem and generalize the derived *LTL*-to-deterministic-Rabin-automata translation to *pLTL*, while maintaining its optimal asymptotic complexity. The merits of *LTL* extended with past operators were argued by Lichtenstein et al. [9], who also showed that their addition does not result in a more expressive logic with respect to initial equivalence of formulae. At the same time, the complexity of satisfiability/validity- and model checking remains PSPACE-complete for *pLTL* [17]. When it comes to determinization, as Esparza's approach [5] applies only to (future) *LTL*, the only option is the two-step approach: translate *pLTL* to nondeterministic Büchi automata [14] and convert to deterministic parity/Rabin automata [16, 13]. Our generalization to *pLTL* is of both theoretical and practical interest, for two main reasons. First, certain properties are more naturally and elegantly expressed with the help of past operators. Secondly, there exist formulae in *pLTL* such that all (initially) equivalent *LTL* formulae are exponentially larger [11]. Both of these properties can be exemplified by considering the natural language specification *"At any point in time, p should occur if and only if q and r have occurred at least once in the past"*. Expressing this in *LTL* requires explicitly describing the possible desired orders of occurrences of p, q, and r:

$$\big((\neg p \wedge \neg q)\,\mathbf{W}\,(r \wedge ((\neg p \wedge \neg q)\,\mathbf{W}\,(p \wedge q))) \big) \vee (\neg p \wedge \neg r)\,\mathbf{W}\,(q \wedge ((\neg p \wedge \neg r)\,\mathbf{W}\,(p \wedge r)))\big)$$
$$\wedge \mathbf{G}(p \Rightarrow \mathbf{XG}p).$$

The same specification is very intuitively and succinctly expressed in *pLTL* by the formula $\mathbf{G}(p \Leftrightarrow \mathbf{O}q \wedge \mathbf{O}r)$.

The main contributions of this paper are an adaptation of the Master Theorem for *pLTL* and a utilization of this in the form of an asymptotically optimal direct translation from *pLTL* to deterministic Rabin automata. The paper is structured in the following manner: In Section 2, we define the syntax and semantics of linear temporal logic with past, infinite words and ω-automata, and the notion of propositional equivalence. As the automata we aim to construct are one-way, we need a way to encode information about the input history directly into the formula being translated; the machinery required to accomplish this is introduced in Section 3. The foundation of the translation from *pLTL* to Rabin automata is the after-function of Section 4. The subsequent Sections 5 and 6 are adaptations of the corresponding sections in Esparza et al. [5]. The decomposition of the language of the formula to be translated into a Boolean combination of simpler languages requires considering the limit-behavior of the formula. This notion is made precise in Section 5, which finishes with a presentation of the Master Theorem for *pLTL*. Section 6 describes how to create deterministic automata from the simpler languages of the decomposition, and how to combine them into a deterministic Rabin automaton. Finally, we conclude with a brief discussion in Section 7.

2 Preliminaries

2.1 Infinite Words and ω-automata

An infinite word w over a non-empty finite alphabet Σ is an infinite sequence $\sigma_0, \sigma_1, \dots$ of letters from Σ. Given an infinite word w, we denote the finite infix $\sigma_t, \sigma_{t+1}, \dots, \sigma_{t+s-1}$ of w by w_{ts}. If $t = s$, then w_{ts} is defined as representing the empty word ϵ. Note that no ambiguity will arise as we use parentheses whenever required; for example, $w_{(st)(en)}$ rather than w_{sten} We denote the infinite suffix $\sigma_t, \sigma_{t+1}, \dots$ of w by w_t. We will also consider finite words, using the same infix- and suffix notation.

An ω-*automaton* over an alphabet Σ is a quadruple (Q, Q_0, δ, α), where Q is a finite set of states, $Q_0 \subseteq Q$ a non-empty set of initial states, $\delta \in Q \times \Sigma \to 2^Q$ a (partial) transition function, and α a set constituting its acceptance condition. In the case where $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all $q \in Q$ and all $\sigma \in \Sigma$, the automaton is called *deterministic*. For deterministic automata we write $\delta : Q \times \Sigma \to Q$. Given an ω-automaton $\mathcal{A} = (Q, Q_0, \delta, \alpha)$ and an infinite word $w = \sigma_0, \sigma_1, \dots$, both over the same alphabet, a *run* of $\mathcal{A}$ on w is a sequence of states $r = r_0, r_1, \dots$ of Q such that $r_0 \in Q_0$ and $r_{i+1} \in \delta(r_i, \sigma_i)$ for all $i \geq 0$. Given such a run r, we write $Inf(r)$ to denote the set of states appearing infinitely often in r.

In this paper, we consider three particular classes of ω-automata, which differ only by their acceptance condition α. *Büchi*- and *co-Büchi* automata are ω-automata with α as a set $Q' \subseteq Q$. A Büchi automaton *accepts* the infinite word w iff there exists a run r on w such that $Inf(r) \cap Q' \neq \varnothing$, while a co-Büchi automaton accepts w iff there exists a run r on w such that $Inf(r) \cap Q' = \varnothing$. We will also consider Rabin automata. A Rabin automaton has a set of subsets $R \subseteq 2^Q \times 2^Q$ as acceptance condition, and accepts w iff there exists a run r on w and a pair $(A, B) \in R$ such that $Inf(r) \cap A = \varnothing$ and $Inf(r) \cap B \neq \varnothing$. We write DBA, DCA, and DRA to refer to deterministic Büchi-, co-Büchi-, and Rabin automata, respectively.

We use a specialized form of cascade composition of two automata: bed automaton and runner automaton. Bed automata are automata without an acceptance condition. Runner automata read input letters and (next) states of bed automata: given a bed automaton $\mathcal{A}$ with S as set of states, a runner automaton is $\mathcal{B} = (S, Q, Q_0, \delta, \alpha)$, where Q, Q_0, and α are as before and $\delta : Q \times S \times \Sigma \to 2^Q$ is the transition function. The composition $\mathcal{A} \ltimes \mathcal{B}$ is the automaton $(Q \times S, Q_0 \times S_0, \overline{\delta}, \overline{\alpha})$, where $\overline{\delta}(q, s, \sigma) = \{(q', s') \mid q' \in \delta_B(q, s', \sigma), s' \in \delta_{\mathcal{A}}(s, \sigma)\}$ and either

$\overline{\alpha} = \alpha \times S$ if $\mathcal{B}$ is a Büchi- or co-Büchi automaton, or $\overline{\alpha} = \bigcup\{(A \times S) \times (B \times S) \mid (A, B) \in \alpha\}$ if $\mathcal{B}$ is a Rabin automaton. Note that $\mathcal{A} \ltimes \mathcal{B}$ is an ω-automaton with $\mathcal{B}$'s acceptance, e.g., if $\mathcal{A}$ is deterministic and $\mathcal{B}$ is a deterministic co-Büchi runner automaton, then $\mathcal{A} \ltimes \mathcal{B}$ is a DCA.

2.2   Linear Temporal Logic with Past

Given a non-empty finite set of propositional variables AP, the well-formed formulae φ of *pLTL* are generated by the following grammar:

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \, \mathbf{U} \, \varphi \mid \mathbf{Y}\varphi \mid \varphi \, \mathbf{S} \, \varphi,$$

where $p \in AP$. Given a formula φ, we write $Var(\varphi)$ to denote the set of all atomic propositions appearing in φ. Given a formula φ, natural number t, and infinite word $w = \sigma_0, \sigma_1, \ldots$ over $2^{Var(\varphi)}$, we write $(w, t) \models \varphi$ to denote that w satisfies φ at index t. The meaning of this is made precise by the following inductive definition:

$(w, t) \models \top$ $(w, t) \not\models \bot$

$(w, t) \models p$ iff $p \in \sigma_t$ $(w, t) \models \neg\varphi$ iff $(w, t) \not\models \varphi$

$(w, t) \models \varphi \wedge \psi$ iff $(w, t) \models \varphi$ and $(w, t) \models \psi$ $(w, t) \models \varphi \vee \psi$ iff $(w, t) \models \varphi$ or $(w, t) \models \psi$

$(w, t) \models \mathbf{X}\varphi$ iff $(w, t+1) \models \varphi$ $(w, t) \models \mathbf{Y}\varphi$ iff $t > 0$ and $(w, t-1) \models \varphi$

$(w, t) \models \varphi \, \mathbf{U} \, \psi$ iff $\exists r \geq t \,.\, ((w, r) \models \psi$ and $\forall s \in [t, r) \,.\, (w, s) \models \varphi)$

$(w, t) \models \varphi \, \mathbf{S} \, \psi$ iff $\exists r \leq t \,.\, ((w, r) \models \psi$ and $\forall s \in (r, t] \,.\, (w, s) \models \varphi)\,.$

When $t = 0$ we omit the index and simply write $w \models \varphi$. Observe that $\mathbf{Y}$ is almost exactly the past analog of $\mathbf{X}$, with a similar relationship between $\mathbf{S}$ and $\mathbf{U}$. They differ in that the past is bounded; in particular, a formula $\mathbf{Y}\varphi$ is never satisfied at $t = 0$. The *language* of a formula φ, denoted $\mathcal{L}(\varphi)$, is the set of all infinite words w such that $w \models \varphi$. Two *pLTL* formulae φ and ψ are semantically equivalent, denoted $\varphi \equiv \psi$, iff $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$.

In addition to the above, we will also consider the following derived operators:

$$\mathbf{F}\varphi := \top \, \mathbf{U} \, \varphi \qquad \mathbf{O}\varphi := \top \, \mathbf{S} \, \varphi \qquad \mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$$

$$\mathbf{H}\varphi := \neg\mathbf{O}\neg\varphi \qquad \varphi \, \mathbf{W} \, \psi := \varphi \, \mathbf{U} \, \psi \vee \mathbf{G}\varphi \qquad \varphi \, \tilde{\mathbf{S}} \, \psi := \varphi \, \mathbf{S} \, \psi \vee \mathbf{H}\varphi$$

$$\varphi \, \mathbf{M} \, \psi := \psi \, \mathbf{U} \, (\varphi \wedge \psi) \qquad \varphi \, \mathbf{B} \, \psi := \psi \, \mathbf{S} \, (\varphi \wedge \psi) \qquad \varphi \, \mathbf{R} \, \psi := \psi \, \mathbf{W} \, (\varphi \wedge \psi)$$

$$\varphi \, \tilde{\mathbf{B}} \, \psi := \psi \, \tilde{\mathbf{S}} \, (\varphi \wedge \psi) \qquad \tilde{\mathbf{Y}}\varphi := \mathbf{Y}\varphi \vee \neg\mathbf{Y}\top.$$

The past operators above are defined in analogy with their standard future counterparts. An important exception is the *weak yesterday* operator $\tilde{\mathbf{Y}}$, which is similar to $\mathbf{Y}$. However, a formula $\tilde{\mathbf{Y}}\varphi$ is always satisfied at $t = 0$.

A *pLTL* formula that is neither atomic nor whose syntax tree is rooted with a Boolean operator is called a *temporal* formula, and a *pLTL* formula whose syntax tree is rooted with an element of $\{\mathbf{Y}, \tilde{\mathbf{Y}}, \mathbf{S}, \tilde{\mathbf{S}}, \mathbf{B}, \tilde{\mathbf{B}}\}$ a *past* formula. Finally, a *pLTL* formula that is a proposition or the negation thereof is *propositional*. We write $sf(\varphi)$ to denote the set of propositional and temporal subformulae of φ, and $psf(\varphi)$ the set of past subformulae of φ. The *size* of a *pLTL* formula φ, denoted $|\varphi|$, is defined as the number of nodes of its syntax tree that are either temporal or propositional.

A *pLTL* formula where negations only appear before atomic propositions is in *negation normal form*. Observe that with the derived operators above, an arbitrary *pLTL* formula can be rewritten in negation normal form with a linear increase in size. For the remainder of the paper, when we write "formula" we implicitly refer to *pLTL* formulae in negation normal

form, with no occurrences of $\mathbf{F}, \mathbf{G}, \mathbf{O}$ or $\mathbf{H}$. Subformulae rooted with either of these four operators can be replaced by equivalent formulae of the same size: every subformula of the form $\mathbf{F}\psi$ can be replaced with $\top \, \mathbf{U} \, \psi$ and every subformula of the form $\mathbf{G}\psi$ with $\psi \, \mathbf{W} \perp$, and analogously for $\mathbf{O}\psi$ and $\mathbf{H}\psi$. While these four derived operators are not part of the syntax under consideration – for the purpose of keeping the presentation more concise – we will occasionally use them as convenient shorthand. When we write "word" we implicitly refer to infinite words, unless otherwise stated.

We conclude this section by defining the notion of *propositional equivalence*, which is a stronger notion of equivalence than that given by the semantics of *pLTL*. It is relatively simple to determine whether two formulae are propositionally equivalent, which makes the notion useful in defining the state spaces of automata as equivalence classes of formulae. We also state a lemma that allows us to lift functions defined on formulae to the propositional equivalence classes they belong to. Both are due to Esparza et al. [5].

▶ **Definition 1** (Propositional Semantics of *pLTL*)**.** *Let $\mathcal{I}$ be a set of formulae and φ a formula. The propositional satisfaction relation $\mathcal{I} \models_p \varphi$ is inductively defined as*

$$\mathcal{I} \models_p \top \qquad \mathcal{I} \models_p \psi \wedge \xi \text{ iff } \mathcal{I} \models_p \psi \text{ and } \mathcal{I} \models_p \xi$$
$$\mathcal{I} \not\models_p \perp \qquad \mathcal{I} \models_p \psi \vee \xi \text{ iff } \mathcal{I} \models_p \psi \text{ or } \mathcal{I} \models_p \xi,$$

with $\mathcal{I} \models_p \varphi$ iff $\varphi \in \mathcal{I}$ for all other cases. Two formulae φ and ψ are propositionally equivalent, denoted $\varphi \sim \psi$, if $\mathcal{I} \models_p \varphi \Leftrightarrow \mathcal{I} \models_p \psi$ for all sets of formulae $\mathcal{I}$. The (propositional) equivalence class of a formula φ is denoted $[\varphi]_\sim$. The (propositional) quotient set of a set of formulae Ψ is denoted $\Psi_{/\sim}$.

▶ **Lemma 2.** *Let f be a function on formulae such that $f(\top) = \top$, $f(\perp) = \perp$, and for all formulae φ and ψ, $f(\varphi \wedge \psi) = f(\varphi) \wedge f(\psi)$ and $f(\varphi \vee \psi) = f(\varphi) \vee f(\psi)$. Then, for all pairs of formulae φ and ψ, if $\varphi \sim \psi$ then $f(\varphi) \sim f(\psi)$.*

3 Encoding the Past

Informally, given a formula φ and word w, our aim is to define a function that consumes a given finite prefix of w, of arbitrary length t, and produces a new formula φ', such that the suffix w_t satisfies φ' iff w satisfies φ. This function will serve as the foundation for defining the state spaces and transition relations of the automata that we are to construct. For standard LTL, defining such a function is straightforward using the local semantics of LTL. With the introduction of past operators the situation becomes more complicated. As a prefix of w is consumed we lose the information about the past therein, and must instead encode this information in the rewritten formula. The key insight is that this can be accomplished by rewriting strong past operators of φ – the operators $\mathbf{Y}$, $\mathbf{S}$, and $\mathbf{B}$ – into their weak counterparts $\tilde{\mathbf{Y}}$, $\tilde{\mathbf{S}}$, and $\tilde{\mathbf{B}}$, respectively, and vice versa, based on the consumed input. This section makes this idea precise.

▶ **Definition 3** (Weakening and strengthening formulae)**.** *The weakening $\varphi_\mathcal{W}$ and strengthening $\varphi_\mathcal{S}$ of a formula φ is defined by case distinction on φ as*

$$(\mathbf{Y}\psi)_\mathcal{W} \coloneqq \tilde{\mathbf{Y}}\psi \quad (\psi \, \mathbf{S} \, \xi)_\mathcal{W} \coloneqq \psi \, \tilde{\mathbf{S}} \, \xi \quad (\psi \, \mathbf{B} \, \xi)_\mathcal{W} \coloneqq \psi \, \tilde{\mathbf{B}} \, \xi$$
$$(\tilde{\mathbf{Y}}\psi)_\mathcal{S} \coloneqq \mathbf{Y}\psi \quad (\psi \, \tilde{\mathbf{S}} \, \xi)_\mathcal{S} \coloneqq \psi \, \mathbf{S} \, \xi \quad (\psi \, \tilde{\mathbf{B}} \, \xi)_\mathcal{S} \coloneqq \psi \, \mathbf{B} \, \xi.$$

For all other cases we have $\varphi_\mathcal{W} \coloneqq \varphi$ and $\varphi_\mathcal{S} \coloneqq \varphi$.

▶ **Definition 4** (Rewriting past operators under sets). *Given a formula φ and set of past formulae C, we write $\varphi\langle C\rangle$ to denote the result of weakening or strengthening the past operators in the syntax tree of φ according to C while otherwise maintaining its structure, as per the following inductive definition:*

$$a\langle C\rangle := a \qquad\qquad\qquad\qquad\qquad\qquad (a\ atomic)$$

$$(op\,\psi)\langle C\rangle := \begin{cases} (op\,\psi\langle C\rangle)_{\mathcal{W}} & (op\,\psi \in C) \\ (op\,\psi\langle C\rangle)_{\mathcal{S}} & (otherwise) \end{cases} \qquad (op\ unary)$$

$$(\psi\,op\,\xi)\langle C\rangle := \begin{cases} (\psi\langle C\rangle\,op\,\xi\langle C\rangle)_{\mathcal{W}} & (\psi\,op\,\xi \in C) \\ (\psi\langle C\rangle\,op\,\xi\langle C\rangle)_{\mathcal{S}} & (otherwise). \end{cases} \quad (op\ binary).$$

We overload this definition to sets: given a set of formulae S, we define $S\langle C\rangle := \{s\langle C\rangle \mid s \in S\}$.

▶ **Example 5.** Consider the formula $\varphi = \mathbf{Y}(p\,\tilde{\mathbf{S}}\,q)$ and set $C = \{\varphi\}$. We then have $\varphi\langle C\rangle = \tilde{\mathbf{Y}}(p\,\mathbf{S}\,q)$. For the same formula φ and set $C = \{p\,\tilde{\mathbf{S}}\,q\}$, we instead have $\varphi\langle C\rangle = \varphi$.

▶ **Definition 6** (Weakening conditions). *Given a past formula φ, we define the weakening condition function $wc(\varphi)$:*

$$wc(\mathbf{Y}\psi) := \psi \quad wc(\psi\,\mathbf{S}\,\xi) := \xi \qquad wc(\psi\,\mathbf{B}\,\xi) := \psi \wedge \xi$$
$$wc(\tilde{\mathbf{Y}}\psi) := \psi \quad wc(\psi\,\tilde{\mathbf{S}}\,\xi) := \psi \vee \xi \quad wc(\psi\,\tilde{\mathbf{B}}\,\xi) := \xi.$$

The weakening condition for a past formula serves as a requirement that must hold immediately in order to justify weakening the formula for the next time step. More precisely, if $w \models wc(\varphi)$ then it is enough to check that $w_1 \models \varphi_{\mathcal{W}}$ to conclude that $(w, 1) \models \varphi$.

▶ **Example 7.** Let $\varphi = \mathbf{X}(p\,\mathbf{S}\,q)$ and w be a word such that $w_{02} = \{q\}\{p\}$. By establishing that $w \models wc(p\,\mathbf{S}\,q) = q$, we can "forget" about the initial letter $\{q\}$; it is enough to check that $w_1 \models p\,\tilde{\mathbf{S}}\,q$ to conclude that $(w, 1) \models p\,\mathbf{S}\,q$, and hence that $w \models \varphi$.

This suggests that there exists a set of past subformulae of φ that precisely captures the information contained in the initial letter of w required to evaluate all past subformulae of φ at every point in time. This is the main result of this section, which we now summarize.

▶ **Definition 8** (Sets of entailed subformulae). *Let φ be a formula, w a word, and $t \in \mathbb{N}$. The set of past subformulae of φ entailed by w at t is inductively defined as*

$$C_{\varphi,0}^w := \{\psi \in psf(\varphi) \mid \psi = \psi_{\mathcal{W}}\} \qquad C_{\varphi,t}^w := \{\psi \in psf(\varphi\langle C_{\varphi,t-1}^w\rangle) \mid w_t \models wc(\psi)\} \quad (t > 0).$$

When the word w above is clear from the context, we simply write $C_{\varphi,t}$. Given $t \in \mathbb{N}$ we denote the sequence $C_{\varphi,0}, \ldots, C_{\varphi,t}$ of length $t + 1$ by $\vec{C}_{\varphi,t}$. Given a sequence $\vec{C} = C_0, C_1, \ldots, C_t$ of sets of past formulae, there exists a set $C \subseteq psf(\varphi)$ that has the same effect in a rewrite as the sequential application of rewrites of $\vec{C}$, i.e., such that $\varphi\langle C\rangle = \varphi\langle C_0\rangle\langle C_1\rangle \ldots \langle C_t\rangle$. This is the set $\{\psi \in psf(\varphi) \mid \psi\langle C_0\rangle\langle C_1\rangle \ldots \langle C_t\rangle = (\psi\langle C_0\rangle\langle C_1\rangle \ldots \langle C_t\rangle)_{\mathcal{W}}\}$, which we denote by $\circ\,\vec{C}$. In particular, there exists a set that captures the sequence of sets of entailed subformulae, denoted $\circ\,\vec{C}_{\varphi,t}$.

▶ **Lemma 9.** *Given a formula φ, word w, and $t \in \mathbb{N}$, we have $(w, t) \models \varphi$ iff $w_t \models \varphi\langle\circ\,\vec{C}_{\varphi,t}\rangle$.*

With these definitions in place, we are ready to define the after-function for *pLTL*.

4 The after-function for $pLTL$

The after-function is the foundation for defining the states and transition relations of all automata used in our $pLTL$-to-DRA translation. We begin by defining the *local* after-function:

▶ **Definition 10** (The local after-function)**.** *Given a formula φ, a letter $\sigma \in 2^{Var(\varphi)}$, and a set of past formulae C, we inductively define the local after-function af_ℓ mutually with the local past update-function pu_ℓ as follows:*

$$af_\ell(\top, \sigma, C) := \top$$
$$af_\ell(\bot, \sigma, C) := \bot$$
$$af_\ell(p, \sigma, C) := \textit{if } (p \in \sigma) \textit{ then } \top \textit{ else } \bot$$
$$af_\ell(\neg p, \sigma, C) := \textit{if } (p \in \sigma) \textit{ then } \bot \textit{ else } \top$$
$$af_\ell(\mathbf{X}\psi, \sigma, C) := pu_\ell(\psi, \sigma, C)$$
$$af_\ell(\mathbf{Y}\psi, \sigma, C) := \bot$$
$$af_\ell(\tilde{\mathbf{Y}}\psi, \sigma, C) := \top$$
$$af_\ell(\psi\ op\ \xi, \sigma, C) := af_\ell(\psi, \sigma, C)\ op\ af_\ell(\xi, \sigma, C) \qquad\qquad (op \in \{\wedge, \vee\})$$
$$af_\ell(\psi\ op\ \xi, \sigma, C) := af_\ell(\xi, \sigma, C) \vee af_\ell(\psi, \sigma, C) \wedge pu_\ell(\psi\ op\ \xi, \sigma, C) \qquad (op \in \{\mathbf{U}, \mathbf{W}\})$$
$$af_\ell(\psi\ op\ \xi, \sigma, C) := af_\ell(\xi, \sigma, C) \wedge (af_\ell(\psi, \sigma, C) \vee pu_\ell(\psi\ op\ \xi, \sigma, C)) \qquad (op \in \{\mathbf{R}, \mathbf{M}\})$$
$$af_\ell(\psi\ op\ \xi, \sigma, C) := af_\ell(wc(\psi\ op\ \xi), \sigma, C)) \qquad\qquad (op \in \{\mathbf{S}, \tilde{\mathbf{S}}, \mathbf{B}, \tilde{\mathbf{B}}\}),$$

where

$$pu_\ell(\varphi, \sigma, C) := \varphi\langle C\rangle \wedge \bigwedge_{\psi \in psf(\varphi) \cap C} af_\ell(wc(\psi), \sigma, C).$$

Intuitively, in the context of reading the initial letter σ of the word w, the local after-function decomposes φ into parts that can be fully evaluated using σ and immediately be replaced with $\top$ or $\bot$, and parts that can only be partially evaluated using σ. The resulting formula is then left to be further evaluated in the future; in the automaton, it corresponds to the state reached upon reading σ from the state corresponding to φ. Crucially, the past subformulae of the partially evaluated part are updated by pu_ℓ, using the information in C. Here, C is to be thought of as a guess of the past subformulae of φ whose weakening conditions hold upon reading σ. Finally, the weakening conditions that must hold to justify the guess are added conjunctively. Observe that, as these weakening conditions may contain subformulae referring to the future, it may not be possible to fully evaluate them immediately; this motivates the recursive application of af_ℓ in pu_ℓ.

▶ **Definition 11** (The extended local after-function)**.** *Given a (possibly empty) finite word w of length n and sequence of sets of past formulae $\vec{C}$ of length $n + 1$, we extend af_ℓ to w and $\vec{C}$ as follows:*

$$af_\ell(\varphi, \epsilon, \vec{C}_{01}) := \varphi \qquad\qquad (n = 0)$$
$$af_\ell(\varphi, w_{0n}, \vec{C}_{0(n+1)}) := af_\ell\left(\varphi_{n-1}, w_{(n-1)n}, \vec{C}_{n(n+1)}\right) \quad (n > 0),$$

where $\varphi_n = af_\ell\left(\varphi, w_{0n}, \vec{C}_{0(n+1)}\right)$.

Observe that the initial set of $\vec{C}$ in the above definition is discarded; this is to match the sequence of Definition 8. For formulae where no future operators are nested inside past operators, the set of entailed subformulae is completely determined by the prefix w_{0n}. This is not the case in general, however, and so the (global) after-function is defined to consider all possible subsets of past subformulae of φ as a disjunction.

▶ **Definition 12** (The after-function). *Let φ be a formula and σ a letter. The after-function af is defined as:*

$$af(\varphi, \sigma) := \bigvee_{C \in 2^{psf(\varphi)}} af_\ell(\varphi, \sigma, C).$$

The extension of af to finite words is done in the natural way: given a formula φ and word w of length n, we define

$$af(\varphi, \epsilon) := \varphi \qquad\qquad\qquad (n = 0)$$
$$af(\varphi, w_{0n}) := af(af(\varphi, w_{0(n-1)}), w_{(n-1)n}) \quad (n > 0).$$

▶ **Example 13.** Let $\varphi = \mathbf{X}(p \, \mathbf{S} \, \mathbf{X}q)$. Observe that $\varphi \equiv \mathbf{X}(p \wedge q \vee \mathbf{X}q)$. Upon reading a letter σ we can guess that the "since" started holding at the current point, corresponding to the set $C = \{p \, \mathbf{S} \, \mathbf{X}q\}$ and formula $p \, \tilde{\mathbf{S}} \, \mathbf{X}q \wedge q$. Alternatively, we may guess that the "since" did *not* start holding upon reading σ, corresponding to the set $C = \varnothing$ and formula $p \, \mathbf{S} \, \mathbf{X}q$. Hence $af(\varphi, \sigma) = p \, \tilde{\mathbf{S}} \, \mathbf{X}q \wedge q \vee p \, \mathbf{S} \, \mathbf{X}q$. This is equivalent (at $t = 0$) to $p \wedge q \vee \mathbf{X}q$, which is what must be satisfied by w_1, as desired.

The correctness of af is established by the following theorem:

▶ **Theorem 14.** *For every formula φ, word w, and $t \in \mathbb{N}$ we have $w \models \varphi$ iff $w_t \models af(\varphi, w_{0t})$.*

5 Stability and the Master Theorem

We consider two fragments of *pLTL*: μ-*pLTL*, the set of formulae whose future operators are members of $\{\mathbf{X}, \mathbf{U}, \mathbf{M}\}$, and ν-*pLTL*, the set of formulae whose future operators are members of $\{\mathbf{X}, \mathbf{W}, \mathbf{R}\}$. Given a formula φ, we define the set $\mu(\varphi)$ of subformulae of φ whose syntax trees are rooted with $\mathbf{U}$ or $\mathbf{M}$. Similarly, we define the set $\nu(\varphi)$ of subformulae of φ whose syntax trees are rooted with $\mathbf{W}$ or $\mathbf{R}$.

The Master Theorem for *pLTL* establishes that the language of a *pLTL* formula can be decomposed into a Boolean combination of simple languages. It is motivated by two ideas:

i) Assume that φ is a formula and w is a word such that all subformulae in $\mu(\varphi)$ that are eventually satisfied by w are infinitely often satisfied by w, and all subformulae in $\nu(\varphi)$ that are almost always satisfied by w never fail to be satisfied by w. In this case, we say that w is a *stable* word of φ, as will be properly defined shortly. Under these circumstances, a subformula of φ of the form $\psi \, \mathbf{U} \, \xi$ is satisfied by w iff both $\psi \, \mathbf{W} \, \xi$ and $\mathbf{GF}(\psi \, \mathbf{U} \, \xi)$ are. Dually, a subformula of φ of the form $\psi \, \mathbf{W} \, \xi$ is satisfied by w iff either $\psi \, \mathbf{U} \, \xi$ or $\mathbf{FG}(\psi \, \mathbf{W} \, \xi)$ are. Hence, we can partition all words over $\mathrm{Var}(\varphi)$ into partitions of the form $P_{M,N}$, where $w \in P_{M,N}$ iff M is the set of μ-*pLTL*-subformulae of φ satisfied infinitely often by w, and N the set ν-*pLTL*-subformulae of φ that are almost always satisfied by w. Given two such sets M and N, the above implies that φ can be rewritten into a formula that belongs to either the fragment μ-*pLTL* or ν-*pLTL*, as desired.

ii) Given a formula φ and word w, there exists a point in the future from which the above holds. Indeed, if we look far ahead into the future, all subformulae of φ that are satisfied by w only finitely often will have been satisfied for the last time. In particular, this is

true for the μ-*pLTL*-subformulae of φ. Similarly, there exists a point in the future at which all subformulae of φ that are almost always satisfied by w will have failed to be satisfied by w for the last time. In particular, this is true for the ν-*pLTL*-subformulae of φ.

These two notions suggest that, given a formula φ and word $w \in P_{M,N}$, it is possible to transform φ using the after-function until w becomes stable, and then rewrite it according to either M or N. Since these sets are unknown, we need to consider all possible combinations of such subsets, which ultimately manifests as a number of Rabin pairs exponential in the size of the formula. For more details and further examples we refer the reader to Section 5 of Esparza et al. [5]. The exposition of this section follows the similar exposition therein. However, the Master Theorem and the lemmata that imply it require considerable "pastification".

We now make precise the idea expressed in ii). Given a formula φ, word w, and $t \in \mathbb{N}$, we define the set of subformulae in $\mu(\varphi)$ that are satisfied by w at least once at t and the set of subformulae in $\mu(\varphi)$ that are satisfied by w infinitely often at t. Similarly, we define the set of subformulae in $\nu(\varphi)$ that are always satisfied by w at t and the set of subformulae in $\nu(\varphi)$ that are almost always satisfied by w at t:

$$\mathcal{F}_{w,t}^{\varphi} := \{\psi \in \mu(\varphi) \mid (w,t) \models \mathbf{F}\psi\} \quad \mathcal{GF}_{w,t}^{\varphi} := \{\psi \in \mu(\varphi) \mid (w,t) \models \mathbf{GF}\psi\}$$

$$\mathcal{G}_{w,t}^{\varphi} := \{\psi \in \nu(\varphi) \mid (w,t) \models \mathbf{G}\psi\} \quad \mathcal{FG}_{w,t}^{\varphi} := \{\psi \in \nu(\varphi) \mid (w,t) \models \mathbf{FG}\psi\}.$$

As mentioned, we are in particular interested in the point at which the two sets in each row coincide. We express this as the word being *stable* at that point.

▶ **Definition 15** (Stable words). *A word w is μ-stable (ν-stable) with respect to a formula φ at index t if $\mathcal{F}_{w,t}^{\varphi} = \mathcal{GF}_{w,t}^{\varphi}$ ($\mathcal{G}_{w,t}^{\varphi} = \mathcal{FG}_{w,t}^{\varphi}$). If w is both μ-stable and ν-stable with respect to φ at index t, then it is stable with respect to φ at index t.*

▶ **Lemma 16.** *Let φ be a formula and w a word. Then there exists an index $r \in \mathbb{N}$ such that w is stable with respect to φ at all indices $t \geq r$.*

The following two definitions specify how to rewrite a formula according to sets M and N, as indicated in i):

▶ **Definition 17.** *Let φ be a formula and M a set of μ-pLTL-formulae. The formula $\varphi[M]_\nu$ is inductively defined as*

$$a[M]_\nu := a \qquad \qquad (a \text{ atomic})$$

$$(op\,\psi)[M]_\nu := op\,(\psi[M]_\nu) \qquad (op \text{ unary})$$

$$(\psi\,op\,\xi)[M]_\nu := (\psi[M]_\nu)\,op\,(\xi[M]_\nu) \quad (op \in \{\mathbf{W},\mathbf{R},\mathbf{S},\tilde{\mathbf{S}},\mathbf{B},\tilde{\mathbf{B}}\})$$

$$(\psi\,\mathbf{U}\,\xi)[M]_\nu := \begin{cases} (\psi[M]_\nu)\,\mathbf{W}\,(\xi[M]_\nu) & (\psi\,\mathbf{U}\,\xi \in M) \\ \bot & (\text{otherwise}) \end{cases}$$

$$(\psi\,\mathbf{M}\,\xi)[M]_\nu := \begin{cases} (\psi[M]_\nu)\,\mathbf{R}\,(\xi[M]_\nu) & (\psi\,\mathbf{M}\,\xi \in M) \\ \bot & (\text{otherwise}), \end{cases}$$

▶ **Definition 18.** *Let φ be a formula and N a set of ν-pLTL-formulae. The formula $\varphi[N]_\mu$ is inductively defined as*

$$(\psi\,\mathbf{W}\,\xi)[N]_\mu := \begin{cases} \top & (\psi\,\mathbf{W}\,\xi \in N) \\ (\psi[N]_\mu)\,\mathbf{U}\,(\xi[N]_\mu) & (\text{otherwise}) \end{cases}$$

$$(\psi\,\mathbf{R}\,\xi)[N]_\mu := \begin{cases} \top & (\psi\,\mathbf{R}\,\xi \in N) \\ (\psi[N]_\mu)\,\mathbf{M}\,(\xi[N]_\mu) & (\text{otherwise}). \end{cases}$$

The other cases are defined by recursive descent similarly to Definition 17.

Notice that once a formula has been rewritten by M, it becomes a ν-formula. Dually, once a formula has been rewritten by N, it becomes a μ-formula. In terms of the hierarchy of Manna and Pnueli [10], these are safety and guarantee formulae, respectively. It is relatively simple to construct deterministic automata for such formulae as they do not require complicated acceptance conditions.

▶ **Theorem 19** (The Master Theorem for $pLTL$). *Let φ be a formula and w a word stable with respect to φ at index r. Then $w \models \varphi$ iff there exist $M \subseteq \mu(\varphi)$ and $N \subseteq \nu(\varphi)$ such that*

1) $w_r \models af_\ell(\varphi, w_{0r}, \vec{C}_{\varphi,r})[M\langle \circ \vec{C}_{\varphi,r}\rangle]_\nu$.

2) $\forall \psi \in M \,.\, \forall s \,.\, \exists t \geq s \,.\, w_t \models \mathbf{F}(\psi\langle \circ \vec{C}_{\varphi,t}\rangle[N\langle \circ \vec{C}_{\varphi,t}\rangle]_\mu)$.

3) $\forall \psi \in N \,.\, \exists t \geq 0 \,.\, w_t \models \mathbf{G}(\psi\langle \circ \vec{C}_{\varphi,t}\rangle[M\langle \circ \vec{C}_{\varphi,t}\rangle]_\nu)$.

The statements of the Master Theorem in the only if-direction can be significantly strengthened. The existential quantification over t in premise 2 can be made universal. The statement $w_t \models \ldots$ in premise 3 holds for all $t' \geq r$. Given the semantics of $\mathbf{F}$ and $\mathbf{G}$, this is not a surprise. However, the ability to do the rewrites at every given moment is technically involved due to the incorporation of the past. In the next section we show how to use the Master Theorem in the construction of a DRA.

6 From $pLTL$ to DRA

We are now ready, based on the Master Theorem, to construct a DRA for the language of a formula φ. We decompose the language of φ into a Boolean combination of languages, each of which is recognized by a deterministic automaton with the relatively simple acceptance condition of Büchi or co-Büchi. For every possible pair of sets M and N of μ- and ν-subformulae we try to establish the premises of the Master Theorem. Premise 1 can be checked by trying to identify a stability point r from which the safety automaton for $af_\ell(\varphi, w_{0r}, \vec{C}_{\varphi,r})[M\langle \circ \vec{C}_{\varphi,t}\rangle]_\nu$ continues forever. Whenever this safety check fails, simply try again. Overall, this corresponds to a co-Büchi condition and a DCA. Premise 2 can be checked for every $\psi \in M$ by identifying infinitely many points from which the guarantee automaton for $\mathbf{F}(\psi\langle \circ \vec{C}_{\varphi,t}\rangle[N\langle \circ \vec{C}_{\varphi,t}\rangle]_\mu)$ finishes its check. Overall, this corresponds to a Büchi condition and a DBA. Premise 3 is dual to premise 2 and leads to a DCA. The three together are combined to a DRA with one pair. Overall, we get a DRA with exponentially many Rabin pairs; one for each choice of M and N. We will as shorthand make use of the operators $\mathbf{F}$ and $\mathbf{G}$ in the construction of these automata, as described in Section 2.2.

The major difficulty of incorporating the past into this part, is that the rewriting using the set M and N needs to be done with the past subformulae correctly weakened. To facilitate this, we begin by defining an auxiliary automaton in Section 6.1 that serves to track the weakening conditions that must hold in order to justify rewrites by $\cdot\langle\cdot\rangle$.

The state spaces of the automata we construct are defined in terms of the following notions. Given a formula φ we denote by $\mathbb{B}(\varphi)$ the set of formulae ψ satisfying $sf(\psi) \subseteq sf(\varphi) \cup \{\top, \bot\}$. Similarly, consider a formula that is a disjunction of formulae with the property that for each such disjunct ψ there exists a $C \in 2^{psf(\varphi)}$ such that $\psi \in \mathbb{B}(\varphi\langle C\rangle)$. We denote by $\mathbb{B}^\vee(\varphi)$ the set of all such formulae. A key observation is that the sizes of the quotient sets $\mathbb{B}(\varphi)_{/\sim}$ and $\mathbb{B}^\vee(\varphi)_{/\sim}$ are doubly exponential in the size of φ.

For the remainder of this section we consider a fixed formula φ that is to be translated into a Rabin automaton and a fixed ordering $C_1, C_2, \ldots, C_k$ of the elements of $2^{psf(\varphi)}$. For simplicity, we assume $C_1 = \{\psi \in psf(\varphi) \mid \psi = \psi_\mathcal{W}\} = C_{\varphi,0}$. We freely make use of af, $\cdot[\cdot]_\nu$, and $\cdot[\cdot]_\mu$ lifted to equivalence classes of formulae. This is justified by Lemma 2.

6.1 The Weakening Conditions Automaton

The weakening conditions automaton (WC automaton) is a bed automaton that tracks the development of weakening conditions under rewrites of the local after-function. Its states are k-tuples of formulae, each of which describes the requirements that remain to be verified in order to justify a sequence of rewrites. To facilitate the construction of the WC automaton, we define the following function:

▶ **Definition 20.** *Given a k-tuple of formulae $\psi^\times = \langle \psi_1, \psi_2, \ldots, \psi_k \rangle$ and a letter σ, the rewrite condition function is defined as $rc(\psi^\times, \sigma) \coloneqq \langle \psi'_1, \psi'_2, \ldots, \psi'_k \rangle$, where*

$$\psi'_i = \bigvee_{j \in J_i} \left(af_\ell(\psi_j, \sigma, C_i \langle C_j \rangle) \wedge \bigwedge_{\xi \in C_i} af_\ell(wc(\xi \langle C_j \rangle), \sigma, C_i \langle C_j \rangle) \right),$$

and where $J_i \coloneqq \left\{ j \in [1..k] \mid \forall \xi, \xi' \in psf(\varphi) \, . \, \xi \langle C_j \rangle = \xi' \langle C_j \rangle \Rightarrow \xi \langle C_i \rangle = \xi' \langle C_i \rangle \right\}$.

The definition of J_i ensures that $\psi'_i \in \mathbb{B}(\varphi\langle C_i \rangle)$. The rewrite condition function takes a k-tuple of formulae and a letter, and returns an updated k-tuple. In the resulting tuple, the ith item ψ'_i is a formula that encodes the updated requirements for further applying the rewrite $\cdot \langle C_i \rangle$. We remark that, for every $t > 0$, there exist indices $i \leq k$ and $j \in J_i$ such that $C_i = \circ \vec{C}_{\varphi,t}$, $C_j = \circ \vec{C}_{\varphi,t-1}$, and $C_i \langle C_j \rangle = C_{\varphi,t}$.

We now define the WC automaton $\mathcal{H}_\varphi \coloneqq (S, S_0, \delta_\mathcal{H})$ over $2^{Var(\varphi)}$. Its set of states S is $\Pi_{i=1}^k \mathbb{B}(\varphi\langle C_i \rangle)_{/\sim}$. The initial state S_0 is the k-tuple $\langle [\top]_\sim, [\bot]_\sim, \ldots, [\bot]_\sim \rangle$, which represents that the set used to rewrite φ into its initial form is known to be C_1. Finally, the transition relation δ is defined by $\delta_\mathcal{H}(\psi^\times, \sigma) = rc(\psi^\times, \sigma)$, with rc lifted to propositional equivalence classes of formulae in the natural way.

6.2 Verifying the Premises of the Master Theorem

We now describe the automata that are capable of verifying the premises of the Master Theorem. They are all runner automata with bed automaton $\mathcal{H}_\varphi \coloneqq (S, S_0, \delta_\mathcal{H})$.

Given a formula $\psi \in \mu(\varphi)$ and set $N \subseteq \nu(\varphi)$ we define the runner automaton $\mathcal{B}_{2,N}^\psi \coloneqq (S, Q, Q_0, \delta, \alpha)$ over $2^{Var(\varphi)}$ with set of states $Q \coloneqq \mathbb{B}^\vee(\varphi[N]_\mu \wedge \mathbf{F}(\psi[N]_\mu))_{/\sim}$, initial state $Q_0 \coloneqq [\mathbf{F}(\psi[N]_\mu)]_\sim$, and Büchi acceptance condition $\alpha \coloneqq [\top]_\sim$. Finally, its transition relation is defined as

$$\delta(\zeta, \xi^\times, \sigma) \coloneqq \begin{cases} \displaystyle\bigvee_{i \in [1..k]} \mathbf{F}(\psi\langle C_i \rangle [N\langle C_i \rangle]_\mu) \wedge \xi_i[N\langle C_i \rangle]_\mu & (\zeta \sim \top) \\ af(\zeta, \sigma) & (\text{otherwise}), \end{cases}$$

where $\xi^\times = \langle \xi_1, \xi_2, \ldots, \xi_k \rangle$. For readability, we express δ in terms of formulae, but ask the reader to note that they represent their corresponding propositional equivalence classes.

Informally, the automaton $\mathcal{B}_{2,N}^\psi$ begins by checking that the input word w satisfies the formula $\mathbf{F}(\psi[N]_\mu)$. Because this formula is in the fragment μ-$pLTL$, it is satisfied by w iff the after-function eventually rewrites it into a propositionally true formula. At this point, the automaton restarts, checking the formula again. Because the subset of $psf(\varphi)$ that puts $\psi[N]_\mu$ in the correct form at this point – with respect to its past subformulae – is unknown, all possible such sets are considered in the form of a disjunction. To each disjunct the corresponding weakening conditions, as tracked by the WC automaton, are added. It follows that $\mathcal{H}_\varphi \ltimes \mathcal{B}_{2,N}^\psi$ is able to verify premise 2) of the Master Theorem for the considered formula ψ and sets M and N.

Given a formula $\psi \in \nu(\varphi)$ and set $M \subseteq \mu(\varphi)$ we define the runner automaton $\mathcal{C}_{3,M}^{\psi} :=$ $(S, Q, Q_0, \delta, \alpha)$ over $2^{Var(\varphi)}$ with set of states $Q := \mathbb{B}^{\vee}(\varphi[M]_{\nu} \wedge \mathbf{G}(\psi[M]_{\nu}))_{/\sim}$, initial state $Q_0 := [\mathbf{G}(\psi[M]_{\nu})]_{\sim}$, and co-Büchi acceptance condition $\alpha := [\bot]_{\sim}$. Finally, its transition relation is defined as

$$\delta(\zeta, \xi^{\times}, \sigma) := \begin{cases} \bigvee_{i \in [1..k]} \mathbf{G}(\psi\langle C_i\rangle[M\langle C_i\rangle]_{\nu}) \wedge \xi_i[M\langle C_i\rangle]_{\nu} & (\zeta \sim \bot) \\ af(\zeta, \sigma) & \text{(otherwise)}, \end{cases}$$

where $\xi^{\times} = \langle \xi_1, \xi_2, \ldots, \xi_k \rangle$. As before, the formulae of δ represent their corresponding equivalence classes. By a similar argument as before, the automaton $\mathcal{H} \ltimes \mathcal{C}_{3,M}^{\psi}$ is able to verify premise 3) of the Master Theorem for the formula ψ and sets M and N.

We now turn to constructing automata for verifying the first premise of the Master Theorem. Given a set $M \subseteq \mu(\varphi)$, we define the runner automaton $\mathcal{C}_{\varphi,M}^1 := (S, Q, Q_0, \delta, \alpha)$ over $2^{Var(\varphi)}$ with set of states $Q := \mathbb{B}^{\vee}(\varphi)_{/\sim} \times \mathbb{B}^{\vee}(\varphi[M]_{\nu})_{/\sim}$, initial state $Q_0 := \langle [\varphi]_{\sim}, [\varphi[M]_{\nu}]_{\sim} \rangle$, and co-Büchi acceptance condition $\alpha := \mathbb{B}^{\vee}(\varphi)_{/\sim} \times \{[\bot]_{\sim}\}$. Its transition relation is defined as

$$\delta(\langle \psi, \zeta \rangle, \xi^{\times}, \sigma) := \begin{cases} \left\langle af(\psi, \sigma), \bigvee_{i \in [1..k]} af(\psi, \sigma)[M\langle C_i\rangle]_{\nu} \wedge \xi_i[M\langle C_i\rangle]_{\nu} \right\rangle & (\zeta \sim \bot) \\ \left\langle af(\psi, \sigma), af(\zeta, \sigma) \right\rangle & \text{(otherwise)}, \end{cases}$$

where $\xi^{\times} = \langle \xi_1, \xi_2, \ldots, \xi_k \rangle$. Again, we remind the reader that the formulae in the above definition represent equivalence classes of formulae. The purpose of the above automaton is to "guess" an index at which w is stable with respect to φ, starting with the guess that it is initially stable. Both φ and $\varphi[M]$ are evaluated in tandem. If the current guess of point of stability is incorrect, the second component will eventually collapse into a formula propositionally equivalent to $\bot$. At this point, the automaton proceeds with a new guess by reapplying $\cdot[M]_{\nu}$ to φ as it has currently been transformed by the after-function. As with the other automata, the formulae that make up the states of the WC automaton are used to justify the rewrite by each set C_i. The automaton $\mathcal{H}_{\varphi} \ltimes \mathcal{C}_{\varphi,M}^1$ is able to verify premise 1) of the Master Theorem for the formula φ and the set M.

6.3   The Rabin Automaton

We now construct the final deterministic Rabin automaton. The automaton is the disjunction of up to 2^n simpler Rabin automata; one for every possible choice of $M \subseteq \mu(\varphi)$ and $N \subseteq \mu(\varphi)$. Taking the disjunction of deterministic Rabin automata is possible by running automata for all disjuncts in parallel (via a product construction) and taking the acceptance condition that checks that at least one of them is accepting. Each simpler Rabin automaton checks the three premises of the Master Theorem for its specific M and N: (a) $M \subseteq \mathcal{GF}_{w,0}^{\varphi}$, (b) $N \subseteq \mathcal{FG}_{w,0}^{\varphi}$, and (c) w satisfies a version of φ simplified by M. Each of the three is checked by a Büchi or co-Büchi automaton. In order to check all three we have to consider their conjunction. For a Rabin automaton (with one pair) it is possible to check the conjunction of Büchi and co-Büchi by running automata for all conjuncts in parallel (product construction) and using the Rabin acceptance condition (one pair) to ensure that all co-Büchi automata and all Büchi automata are accepting.

▶ **Theorem 21.** *Let φ be a formula. For each $M \subseteq \mu(\varphi)$ and $N \subseteq \nu(\varphi)$, define*

$$\mathcal{B}_{M,N}^2 := \bigcap_{\psi \in M} \mathcal{H}_\varphi \ltimes \mathcal{B}_{2,N}^\psi \qquad \mathcal{C}_{M,N}^3 := \bigcap_{\psi \in N} \mathcal{H}_\varphi \ltimes \mathcal{C}_{3,M}^\psi$$

$$\mathcal{R}_{\varphi,M,N} := (\mathcal{H}_\varphi \ltimes \mathcal{C}_{\varphi,M}^1) \cap \mathcal{B}_{M,N}^2 \cap \mathcal{C}_{M,N}^3,$$

where $\mathcal{R}_{\varphi,M,N}$ has one Rabin pair. Then the following DRA over $2^{Var(\varphi)}$ recognizes φ:

$$\mathcal{A}_{DRA}(\varphi) := \bigcup_{\substack{M \subseteq \mu(\varphi) \\ N \subseteq \nu(\varphi)}} \mathcal{R}_{\varphi,M,N}.$$

▶ **Corollary 22.** *Let φ be a formula of size $n + m$, where n is the number of future- and propositional subformulae of φ, and m is the number of past subformulae of φ. There exists a deterministic Rabin automaton recognizing φ with doubly exponentially many states in the size of the formula and at most 2^n Rabin pairs.*

7   Discussion

We presented a direct translation from *pLTL* to deterministic Rabin automata. Starting from a formula with n future subformulae and m past subformulae, we produce an automaton with an optimal $2^{2^{O(n+m)}}$ states and $2^{O(n)}$ acceptance pairs. Our translation relies on extending the classical "after"-function of *LTL* to *pLTL* by encoding memory about the past through the weakening and strengthening of embedded past operators. We extended the Master Theorem about decomposition of languages expressed for *LTL* to *pLTL*.

The only applicable approach (prior to our work) to obtain deterministic automata from *pLTL* formulae was to convert the formula to a nondeterministic automaton [14] and then determinize this automaton [16, 13]. The first can be done either directly [9, 14] or through two-way very-weak alternating automata [7]. In any case, the first translates a formula with n future operators and m past operators to an automaton with $2^{O(n+m)}$ states and the second translates an automaton with k states to a parity automaton with $O(k!^2)$ states and $O(k)$ priorities. It follows that the overall complexity of this construction is $2^{2^{O((n+m)\cdot\log(n+m))}}$ states and $2^{O(n+m)}$ priorities. Our approach improves this upper bound to $2^{2^{O(m+n)}}$. It is well known that *pLTL* does not extend the expressive power of *LTL*. However, conversion from *pLTL* to *LTL* is not viable algorithmically. The best known translation is worst-case non-elementary [6], and the conversion is provably exponential [11]. So using a conversion to *LTL* as a preliminary step to determinization could result in a triple exponential construction. We note that in the case where there are no future operators nested within past operators, it is possible to convert the past subformulae directly to deterministic automata. Then, the remaining future can be determinized independently. This approach has been advocated for usage of the past in reactive synthesis [3, 4] and implemented recently in a bespoke tool [2].

As future work, we note that the approach of Esparza et al. [5] additionally led to translations from *LTL* to nondeterministic automata, limit-deterministic automata, and deterministic automata. The same should be done for *pLTL*. Their work also led to a normal form for *LTL* formulae, which we believe could be generalized to work for *pLTL*. The latter could have interesting relations to the temporal hierarchy of Manna and Pnueli [10]. Such a normal form could also be related to more efficient translations from *pLTL* to *LTL*. Finally, this approach has led to a competitive implementation of determinization [8] and reactive synthesis [12]. Extending these implementations to handle past is of high interest.

References

1 Shaun Azzopardi, David Lidell, and Nir Piterman. A Direct Translation from LTL with Past to Deterministic Rabin Automata, 2024. `arXiv:2405.01178`.

2 Shaun Azzopardi, David Lidell, Nir Piterman, and Gerardo Schneider. ppLTLTT : Temporal Testing for Pure-Past Linear Temporal Logic Formulae. In Étienne André and Jun Sun, editors, *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II*, volume 14216 of *Lecture Notes in Computer Science*, pages 276–287. Springer, 2023. `doi:10.1007/978-3-031-45332-8_15`.

3 Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012. `doi:10.1016/J.JCSS.2011.08.007`.

4 Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. Reactive Synthesis from Extended Bounded Response LTL Specifications. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 83–92. IEEE, 2020. `doi:10.34727/2020/ISBN.978-3-85448-042-6_15`.

5 Javier Esparza, Jan Křetínský, and Salomon Sickert. A Unified Translation of Linear Temporal Logic to ω-Automata. *J. ACM*, 67(6), October 2020. `doi:10.1145/3417995`.

6 Dov M. Gabbay. The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In Behnam Banieqbal, Howard Barringer, and Amir Pnueli, editors, *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448, New York, NY, USA, 1987. Springer. `doi:10.1007/3-540-51803-7_36`.

7 Paul Gastin and Denis Oddoux. LTL with Past and Two-Way Very-Weak Alternating Automata. In Branislav Rovan and Peter Vojtás, editors, *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, volume 2747 of *Lecture Notes in Computer Science*, pages 439–448, New York, NY, USA, 2003. Springer. `doi:10.1007/978-3-540-45138-9_38`.

8 Jan Kretínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A Library for ω-Words, Automata, and LTL. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550. Springer, 2018. `doi:10.1007/978-3-030-01090-4_34`.

9 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.

10 Zohar Manna and Amir Pnueli. A Hierarchy of Temporal Properties. In Cynthia Dwork, editor, *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, August 22-24, 1990*, pages 377–410. ACM, 1990. `doi:10.1145/93385.93442`.

11 Nicolas Markey. Temporal Logic with Past is Exponentially More Succinct. *Bulletin- European Association for Theoretical Computer Science*, 79:122–128, 2003. URL: `https://hal.science/hal-01194627`.

12 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit Reactive Synthesis Strikes Back! In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018. `doi:10.1007/978-3-319-96145-3_31`.

13 Nir Piterman. From Nondeterministic Buchi and Streett Automata to Deterministic Parity Automata. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, LICS '06, pages 255–264, USA, 2006. IEEE Computer Society. `doi:10.1109/LICS.2006.28`.

14 Nir Piterman and Amir Pnueli. Temporal Logic and Fair Discrete Systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 27–73. Springer, New York, NY, USA, 2018. `doi:10.1007/978-3-319-10575-8_2`.

15 Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, USA, 1977. IEEE Computer Society. `doi:10.1109/SFCS.1977.32`.

16 S. Safra. On the Complexity of Omega-Automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, pages 319–327, USA, 1988. IEEE Computer Society. `doi:10.1109/SFCS.1988.21948`.

17 A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, July 1985. `doi:10.1145/3828.3837`.