Symbolic Automata: Omega-Regularity Modulo Theories

MARGUS VEANES, Microsoft Research, USA
THOMAS BALL, Microsoft Research, USA
GABRIEL EBNER, Microsoft Research, USA
EKATERINA ZHUCHKO, Tallinn University of Technology, Estonia

Symbolic automata are finite state automata that support potentially infinite alphabets, such as the set of rational numbers, generally applied to regular expressions and languages over finite words. In symbolic automata (or automata modulo $\mathcal{A}$), an alphabet is represented by an effective Boolean algebra $\mathcal{A}$, supported by a decision procedure for satisfiability. Regular languages over infinite words (so called ω-regular languages) have a rich history paralleling that of regular languages over finite words, with well known applications to model checking via Büchi automata and temporal logics.

We generalize symbolic automata to support ω-regular languages via *transition terms* and *symbolic derivatives*, bringing together a variety of classic automata and logics in a unified framework that provides all the necessary ingredients to support symbolic model checking modulo $\mathcal{A}$. In particular, we define: (1) alternating Büchi automata modulo $\mathcal{A}$ ($ABW_{\mathcal{A}}$) as well (non-alternating) nondeterministic Büchi automata modulo $\mathcal{A}$ ($NBW_{\mathcal{A}}$); (2) an alternation elimination algorithm Æ that incrementally constructs an $NBW_{\mathcal{A}}$ from an $ABW_{\mathcal{A}}$, and can also be used for constructing the product of two $NBW_{\mathcal{A}}$; (3) a definition of linear temporal logic modulo $\mathcal{A}$, $\mathbf{LTL}\langle\mathcal{A}\rangle$, that generalizes Vardi's construction of alternating Büchi automata from LTL, using (2) to go from LTL modulo $\mathcal{A}$ to $NBW_{\mathcal{A}}$ via $ABW_{\mathcal{A}}$.

Finally, we present $\mathbf{RLTL}\langle\mathcal{A}\rangle$, a combination of $\mathbf{LTL}\langle\mathcal{A}\rangle$ with extended regular expressions modulo $\mathcal{A}$ that generalizes the Property Specification Language (PSL). Our combination allows regex *complement*, that is not supported in PSL but can be supported naturally by using transition terms. We formalize the semantics of $\mathbf{RLTL}\langle\mathcal{A}\rangle$ using the *Lean* proof assistant and formally establish correctness of the main derivation theorem.

CCS Concepts: • **Theory of computation → Automata over infinite objects**; **Regular languages**; **Modal and temporal logics**; **Verification by model checking**; • **Computing methodologies → Symbolic and algebraic manipulation**.

Additional Key Words and Phrases: regular expression, temporal logic, derivative, satisfiability modulo theories, automata

ACM Reference Format:
Margus Veanes, Thomas Ball, Gabriel Ebner, and Ekaterina Zhuchko. 2025. Symbolic Automata: Omega-Regularity Modulo Theories. *Proc. ACM Program. Lang.* 9, POPL, Article 2 (January 2025), 34 pages. https://doi.org/10.1145/3704838

1 Introduction

Classical finite automata, which correspond to regular expressions and regular languages, are finite in *three orthogonal dimensions*:

✓ the number of *states* in the automata, which make it a *finite* (or *finite-state*) automata;

Authors' Contact Information: Margus Veanes, Microsoft Research, Redmond, USA, margus@microsoft.com; Thomas Ball, Microsoft Research, Redmond, USA, tball@microsoft.com; Gabriel Ebner, Microsoft Research, Redmond, USA, gabrielebner@microsoft.com; Ekaterina Zhuchko, Tallinn University of Technology, Tallinn, Estonia, ekzhuc@taltech.ee.

ACM 2475-1421/2025/1-ART2
https://doi.org/10.1145/3704838

- ✓ the number of *alphabet elements* that induce state transitions;
- ✓ the length of the *words* (a sequence of alphabet elements) recognized by the automata.

Symbolic automata [D'Antoni and Veanes 2021] are finite state automata that support potentially infinite alphabets, such as the set of rational numbers. The transitions of symbolic automata are labeled with predicates from an effective Boolean algebra (EBA) $\mathcal{A}$. Each atomic predicate and their Boolean combination in $\mathcal{A}$ can compactly represent a (possibly infinite) subset of elements.

The power of symbolic automata is to make use of decision procedures for $\mathcal{A}$ [de Moura and Bjørner 2011] to avoid the need for reduction to the propositional (classic) case, which can incur an exponential blowup. For example, a propositional encoding of a Boolean expression of size N in $\mathcal{A}$ would require $O(2^N)$ conjunctions of atomic predicates in a propositional setting (so called "mintermization"). In some cases, a classical algorithm for finite automata can be lifted straightforwardly to symbolic automata; in other cases, entirely new algorithms are required [D'Antoni and Veanes 2021].

Regular languages over infinite words (so called ω-regular languages) have a rich history paralleling that of regular languages over finite words; In fact, the two are intimately connected: [McNaughton 1966] showed that ω-regular languages are equivalently described by Büchi automata and the union of a finite set of ω-regular expressions, where each expression has the form $\alpha_i\beta_i^\omega$, where α_i and β_i are regular expressions for finite words and β_i is infinitely repeated using the *ω-closure* operator. Alternating Büchi automata have equivalent expressive power as (nondeterministic) Büchi automata, but can be exponentially more compact [Miyano and Hayashi 1984].

Furthermore, ω-regular languages are an important part of the theoretical foundation for model checking of reactive systems [Clarke et al. 1999; Pnueli 1985], where both the system and (many of) the properties to be verified of the system are described by sets of infinite traces. Various forms of automata, including Büchi automata, can be used to describe the system while temporal logics describe the properties to be checked. In particular, Linear temporal logic (LTL) [Pnueli 1977] is used to describe the expected behavior of reactive systems and can be lowered to Büchi automata via a variety of algorithms. As LTL does not fully capture ω-regular languages [Emerson 1991; Wolper 1983], various extensions of LTL have been proposed [Armoni et al. 2002; Banieqbal and Barringer 1989; Duret-Lutz 2024; Eisner and Fisman 2006; Sistla et al. 1987; Vardi and Wolper 1994] to increase its expressivity. The LTL-based automata-theoretic approach to model checking constructs the product of the system automaton with the Büchi automaton representing the negation of the LTL formula and then analyzes whether the resulting automaton has any accepting runs [Vardi and Wolper 1986]. If there are no accepting runs then the system automaton is verified with respect to the LTL formula.

Extensions of LTL based program analysis, synthesis, and realizability, that incorporate *modulo theories* reasoning is an active research area. *Fairness modulo theory* [Dietsch et al. 2015] introduces *Büchi programs* whose fair and feasible traces represent feasible traces of a given program that violate a classical LTL property, where feasibility uses SMT solving for satisfiability checking of constraints involving linear artithmetic. *Temporal Stream Logic* [Finkbeiner et al. 2019] (TSL) is an extension of LTL intended for *synthesis* by introducing updates and expressions that may involve arbitrary *uninterpreted* function and predicate symbols. TSL provides increased scalability at the cost of decidability – synthesis for TSL is undecidable in general, while both realizability and synthesis are 2ExpTime-complete for LTL [Pnueli and Rosner 1989]. Further extension of TSL with modulo theories [Finkbeiner et al. 2022] investigates use of *interpreted* functions and predicate symbols, where satisfiability of TSL modulo several standard decidable theories is shown to be neither semidecidable nor co-semidecidable. Recent work on *LTL modulo theories* for *realizability* [Rodríguez

and Sánchez 2023] uses Boolean propositions in place of theory literals in combination with additional theory constraints, that are checked by an SMT solver in order to guarantee consistent interpretations with the theory literals they stand for. This is a form of *mintermization* through *bitblasting* (see Section 1.3) that we believe can be avoided by a closer integration of reasoning modulo theories into the foundations of linear temporal logic through the theory and algorithms of *symbolic derivatives* that utilize *transition terms*.

1.1 Transition Terms

We generalize symbolic automata to ω-regular languages by first lifting the concept of *transition regexes* from [Stanford et al. 2021] to *transition terms*, which are parametric over two domains:

- ✓ *alphabet*, with an EBA $\mathcal{A}$ over universe Σ, over it;
- ✓ *leaves*, which represent the language of the automata, with its own EBA $\mathcal{B}$.

Transition terms for $\mathcal{B}$ (modulo $\mathcal{A}$) are nested if-then-else (ITE) expressions $(\alpha\,?\,f\,:\,g)$ whose conditions α come from $\mathcal{A}$ and leaves come from $\mathcal{B}$, with the intuition "if α then f else g". In the case of nested ITE expressions such as $(\alpha\,?\,(\beta\,?\,f\,:\,g)\,:\,h)$, the primary role of $\mathcal{A}$ is to maintain their *cleanness* by removing unreachable subterms, e.g., if α implies β then g is unreachable.

The EBA $\mathcal{B}$ is typically accompanied with *weak equivalence* laws that are used as rewrite rules to simplify leaves. In particular, conjunction and disjunction are treated as associative, commutative and idempotent (ACI) operations, the law of excluded middle is applied, and the bottom and top elements of $\mathcal{B}$ act as units/zeros of the corresponding Boolean operations. The algebra of transition terms thus becomes an EBA with its own set of rewrite rules derived from $\mathcal{A}$ and $\mathcal{B}$.

Transition terms can be used as the basis of defining the semantics of languages/logics and their corresponding automata using *symbolic derivatives*, providing a unifying framework for automated reasoning and rewriting that operates incrementally over a symbolic representation.

1.2 Symbolic Derivatives

Let $\mathcal{B}$ be an EBA of formulas ϕ modulo an alphabet EBA $\mathcal{A}$ such that $\mathcal{B}$ is associated with a formal semantics $w \models \phi$ where w is a word over Σ, such that $\models$ respects the Boolean operators of $\mathcal{B}$. (I.e., $w \models \neg\phi \Leftrightarrow w \not\models \phi$, $w \models \phi \wedge \psi \Leftrightarrow w \models \phi \wedge w \models \psi$, and $w \models \phi \vee \psi \Leftrightarrow w \models \phi \vee w \models \psi$.) We say that $\mathcal{B}$ has a *derivative function* ρ when

$$\textit{for all } a \in \Sigma,\ \rho(\phi, a) \textit{ is a formula in } \mathcal{B} \textit{ such that } aw \models \phi \Leftrightarrow w \models \rho(\phi, a).$$

Two classical examples of such $\mathcal{B}$ over a *finite* alphabet Σ are *extended regular expressions* (ERE) with $\rho(\phi, a)$ as the derivative $D_a\phi$ in [Brzozowski 1964], and *linear temporal logic* (LTL) with ρ as the transition function ρ in [Vardi 1995]. In the former case $w \in \Sigma^*$ and in the latter case $w \in \Sigma^\omega$.

The *symbolic derivative function* of $\mathcal{B}$ is a function ϱ that maps ϕ into a transition term $\varrho(\phi)$ for $\mathcal{B}$ modulo $\mathcal{A}$. The transition term $\varrho(\phi)$, called the *symbolic derivative of* ϕ, is such that,

$$\textit{for all } a \in \Sigma,\ \varrho(\phi)[a] = \rho(\phi, a).$$

where $\varrho(\phi)[a]$ returns the leaf that the transition term $\varrho(\phi)$ evaluates to for a. Intuitively, $\varrho(\phi)$ represents a *partial evaluation* of ρ with respect to ϕ. Observe that the size of $\varrho(\phi)$ is independent of the size of Σ (Σ can be *infinite*). Existence of $\varrho(\phi)$ depends on ϕ distinguishing only finitely many conditions of $\mathcal{A}$, which holds both for ERE modulo $\mathcal{A}$ as well as LTL modulo $\mathcal{A}$.

An example of a symbolic derivative of a regex `\d+` is $(\texttt{\textbackslash d}\,?\,\texttt{\textbackslash d}{*}\,:\,\bot)$ whose condition `\d` is the character class of all digits. In this case Σ is the set of all characters, that may be a very large set such as Unicode, and the predicates in $\mathcal{A}$ are character classes. E.g., $(\texttt{\textbackslash d}\,?\,\texttt{\textbackslash d}{*}\,:\,\bot)[\text{`7'}] = \texttt{\textbackslash d}{*}$.

An example of an LTL (modulo SMT theory $\mathcal{A}$) or $\mathbf{LTL}\langle\mathcal{A}\rangle$ formula ϕ and $\varrho(\phi)$ are as follows

$$\phi = \mathbf{G}(0{<}x) \wedge ((2|x)\ \mathbf{U}\ (3|x))$$
$$\varrho(\phi) = \mathbf{(}(0{<}x)\ \mathbf{?}\ \mathbf{(}(3|x)\ \mathbf{?}\ \mathbf{G}(0{<}x)\ \mathbf{:}\ \mathbf{(}(2|x)\ \mathbf{?}\ \phi\ \mathbf{:}\ \bot\mathbf{)}\mathbf{)}\ \mathbf{:}\ \bot\mathbf{)}$$

where x has integer type and ϕ states that x is always positive and remains even until divisible by 3. For example, $\varrho(\phi)[x{\mapsto}8] = \phi$ because 8 is a positive even integer that is not divisible by 3. Observe that in this case Σ contains the *infinite* set of all possible interpretations for x.

Integration of symbolic derivatives for ERE into the *sequence theory solver* of Z3 contributed to a significant performance improvement of Z3 [Stanford et al. 2021]. A variant of symbolic derivatives for *regex matching* is also integrated into the new *nonbacktracking* regex matcher of .NET [Moseley et al. 2023] where algebraic rewrites of regexes play a key role as simplification rules during incremental exploration of the state space of a DFA that arises during match search.

1.3 Why Modulo Theories?

The benefits of deeply integrating SMT into the core algorithms are substantial [de Moura and Bjørner 2011] by providing *incremental theory specific reasoning*. At the same time, it has been widely recognized that full ω-regularity is necessary for general applications of linear temporal properties [Pnueli 1985]. Here we achieve both goals modulo $\mathcal{A}$. Our motivating application is *trace analysis of cloud services* (see Section 7.4). We highlight both aspects next.

1.3.1 Avoiding Mintermization. Consider any $\mathbf{LTL}\langle\mathcal{A}\rangle$ formula ψ that contains the $\mathcal{A}$ predicates $\{\alpha_i\}_{i=1}^{n}$. A standard technique to represent ψ in classical LTL, is to replace each α_i by a proposition p_i and to add the following theory constraint as a conjunction: $\mathbf{G}(\bigvee\{p_I \mid I \subseteq \{1,\ldots,n\}, \mathbf{SAT}(\alpha_I)\})$ where $\alpha_I = (\bigwedge_{i\in I}\alpha_i) \wedge (\bigwedge_{i\notin I}\neg\alpha_i)$ and $p_I = (\bigwedge_{i\in I} p_i) \wedge (\bigwedge_{i\notin I}\neg p_i)$. This is a form of *bitblasting*, with each p_i acting as an independent bit, and where $\mathbf{SAT}(\beta)$ decides satisfiability of the formula β in $\mathcal{A}$. Irrespective of how this is done, in general, the *cost is bound to be exponential* in n.

1.3.2 ω-Regularity Modulo $\mathcal{A}$. We introduce a powerful extension $\mathbf{RLTL}\langle\mathcal{A}\rangle$ (**RLTL**) of $\mathbf{LTL}\langle\mathcal{A}\rangle$ with regexes in ERE modulo $\mathcal{A}$. Besides increasing the expressive power to be *ω-regular modulo $\mathcal{A}$*, it also enables a variety of transformations to utilize regexes. In particular, **RLTL** includes formulas of the form $R \boxdot\!\!\rightarrow \phi$ called *suffix implications* where R is a regex and ϕ any formula in **RLTL**, with the semantics $w \models R \boxdot\!\!\rightarrow \phi$ iff $\forall i : w_{..i} \in \mathcal{L}(R) \Rightarrow w_{i..} \models \phi$, where $w_{..i}$ is the finite prefix of w up to position i and $w_{i..}$ is the infinite suffix of w starting from position i. **RLTL** also includes the dual construct $R \diamond\!\!\rightarrow \phi$ called *existential suffix implication* that is equivalent to $\neg(R \boxdot\!\!\rightarrow \neg\phi)$.

New Superpower. **RLTL** supports *complement* $\sim R$ of regexes R, such that $v \in \mathcal{L}(\sim R) \Leftrightarrow v \notin \mathcal{L}(R)$. Thus, expressing an ω-regular property such as *all proper prefixes of an infinite word must be in* $\mathcal{L}(R)$ becomes elegantly expressible by the formula $\sim R \boxdot\!\!\rightarrow \bot$. Since $\bot$ (false) is unsatisfiable, it follows that $w \models \sim R \boxdot\!\!\rightarrow \bot$ iff $\nexists i : w_{..i} \notin \mathcal{L}(R)$ iff $\forall i : w_{..i} \in \mathcal{L}(R)$.

Combined with intersection and union, this lifts the role of regexes in $\mathbf{RLTL}\langle\mathcal{A}\rangle$ into a self-contained sub-logic whose derivative rules are seamlessly integrated with the derivative rules of the temporal formulas (see Section 7.3). We can therefore leverage algorithms in SMT [Stanford et al. 2021] by lifting them to reason over infinite sequences in temporal logic, while ensuring decidability modulo any core element theory $\mathcal{A}$ that is supported by SMT (e.g., linear arithmetic).

Regex Based Rewrites. Rewriting **LTL** formulas by introducing regexes can be beneficial by leveraging regex derivatives that can avoid subsequent alternation elimination from an initial Büchi automaton. E.g., the formula $\mathbf{F}\alpha \wedge \mathbf{F}\beta$ where both α and β belong to $\mathcal{A}$ states that eventually α holds and eventually β holds. In **RLTL** it is equivalent to $\phi = (\top{*}{\cdot}\alpha{\cdot}\top{*}) \Cap (\top{*}{\cdot}\beta{\cdot}\top{*}) \diamond\!\!\rightarrow \top$ where $\Cap$ is regex intersection. Then $w \models \phi$ iff some finite prefix of w is in $\mathcal{L}(\top{*}{\cdot}\alpha{\cdot}\top{*}) \cap \mathcal{L}(\top{*}{\cdot}\beta{\cdot}\top{*})$.

1.4 Contributions and Overview

We show how transition terms and derivatives can be used to generalize symbolic automata to work with ω-regular languages modulo an infinite alphabet represented by $\mathcal{A}$, bringing together a variety of classic automata and logics in a unified framework. In particular, we give:

- ✓ a definition of alternating Büchi automata modulo $\mathcal{A}$ ($ABW_{\mathcal{A}}$) and their relationship to classical alternating Büchi automata as well as to (non-alternating) nondeterministic Büchi automata modulo $\mathcal{A}$ ($NBW_{\mathcal{A}}$) (Section 4);
- ✓ an alternation elimination algorithm, a symbolic generalization of [Miyano and Hayashi 1984] that incrementally constructs an $NBW_{\mathcal{A}}$ from an $ABW_{\mathcal{A}}$ (Section 5);
- ✓ a definition of linear temporal logic (LTL) [Pnueli 1977] modulo $\mathcal{A}$ that generalizes Vardi's construction of alternating Büchi automata from LTL [Vardi 1995] – with the previous results, this gives a way to go from LTL modulo $\mathcal{A}$ to $NBW_{\mathcal{A}}$ (Section 6);
- ✓ a combination of LTL modulo $\mathcal{A}$ with extended regular expressions modulo $\mathcal{A}$ ($\mathbf{RLTL}\langle\mathcal{A}\rangle$) that generalizes SPOT [Duret-Lutz et al. 2022] and PSL [Eisner and Fisman 2006] (Section 7);
- ✓ a formalized proof of correctness in *Lean* of the main derivation theorem of $\mathbf{RLTL}\langle\mathcal{A}\rangle$ (Theorem 4) that links the incremental derivative based unfolding with the formal semantics.

Our combination allows regex *complement*, that is not supported in SPOT/PSL but can be supported naturally by using transition terms. We also lift the classical concept of ω-regular languages [Büchi 1960; McNaughton 1966] as the languages accepted by *ABW* so as to be modulo $\mathcal{A}$, and show that $\mathbf{RLTL}\langle\mathcal{A}\rangle$ precisely captures ω-regularity modulo $\mathcal{A}$. Section 8 reviews related work and Section 9 discusses future work. Section 10 concludes the paper.

2 Preliminaries

As a meta-notation throughout the paper we write $lhs \stackrel{\text{DEF}}{=} rhs$ to let *lhs* be *equal by definition* to *rhs*. In the following let Σ be a nonempty, possibly infinite, *universe* of *core elements*. The universe of natural numbers is denoted by $\mathbb{N}$.

2.1 Effective Boolean Algebras

An *Effective Boolean Algebra (EBA)* over an element universe Σ [D'Antoni and Veanes 2021] is a tuple $(\Sigma, \mathcal{A}, \vDash, \bot, \top, \sqcup, \sqcap, ^c)$ where $\mathcal{A}$ is a set of *predicates* that is closed under the Boolean connectives and contains $\bot$ and $\top$. For $a \in \Sigma$ and $\alpha \in \mathcal{A}$ the *models* relation $a \vDash \alpha$ with $[\![\alpha]\!] \stackrel{\text{DEF}}{=} \{a \in \Sigma \mid a \vDash \alpha\}$ obeys classical Tarski laws such that $[\![\bot]\!] = \emptyset$ and $[\![\top]\!] = \Sigma$. For $\alpha, \beta \in \mathcal{A}$ let $\alpha \equiv \beta \stackrel{\text{DEF}}{=} [\![\alpha]\!] = [\![\beta]\!]$. If $\alpha \equiv \bot$ then α is *unsatisfiable* ($\mathbf{UNSAT}(\alpha)$) else *satisfiable* ($\mathbf{SAT}(\alpha)$). We require all the connectives to be *computable* and $\vDash$ to be *decidable*.

Let $\Gamma \subseteq \mathcal{A}$ be finite. A *minterm* of Γ is a *satisfiable* predicate $(\bigsqcap_{\alpha \in S} \alpha) \sqcap (\bigsqcap_{\alpha \in \Gamma \setminus S} \alpha^c)$ for some subset S of Γ, where $(\bigsqcap_{\alpha \in \emptyset} \cdots) \stackrel{\text{DEF}}{=} \top$. Thus, all predicates in $Minterms(\Gamma)$ are satisfiable, mutually disjoint, and each satisfiable predicate in Γ is equivalent to a disjunction of some minterms. So $|Minterms(\Gamma)| \leq 2^{|\Gamma|}$. For example, $Minterms(\{\alpha, \beta\}) \subseteq \{\alpha \sqcap \beta, \alpha^c \sqcap \beta, \alpha \sqcap \beta^c, \alpha^c \sqcap \beta^c\}$.

We let $\mathcal{A}$ also stand for the EBA itself and say that $\mathcal{A}$ is *decidable* if $\mathbf{SAT}(\alpha)$ is decidable. In most of the paper we use $(\Sigma, \mathcal{A}, \vDash, \bot, \top, \sqcup, \sqcap, ^c)$ as a given *alphabet* EBA.

2.2 Boolean Closures

Given a nonempty set S, we define the *Boolean closure* $\mathbb{B}(S)$ of S such that $S \subseteq \mathbb{B}(S)$ and $\mathbb{B}(S)$ is closed under $\vee$, $\wedge$, and $\neg$ (as some given Boolean operators). We write $\mathbb{B}^+(S)$ for the *positive* Boolean closure where $\neg$ is omitted. We associate a *weak equivalence* relation $\cong$ over $\mathbb{B}(S)$ defining $\wedge$ and $\vee$ both as *associative*, *commutative*, and *idempotent* (ACI) operations. Weak equivalence is lifted to

transition terms with leaves in $\mathbb{B}(S)$ in Section 3 and plays a key role in maintaining *finiteness* of induced state space in later automata constructions.

The disjunctive normal form of $\phi \in \mathbb{B}^+(S)$ is a disjunction $\bigvee_{i=0}^{n} \psi_i$ where each ψ_i is a conjunction $\bigwedge X_i$ for some $X_i \subseteq S$. We adopt the *set-of-sets* representation of $\mathrm{DNF}(\phi)$ as $\{X_i\}_{i=0}^{n}$. For example, $\mathrm{DNF}((s_1 \vee s_2) \wedge s_3) = \{\{s_1, s_3\}, \{s_2, s_3\}\}$ that stands for $(s_1 \wedge s_3) \vee (s_2 \wedge s_3)$ and $\mathrm{DNF}(s_1) = \{\{s_1\}\}$.

Let $\boldsymbol{\varphi} = \mathrm{DNF}(\varphi)$ and $\boldsymbol{\psi} = \mathrm{DNF}(\psi)$. Then we can define $\vee$ and $\wedge$ directly over the set-of-sets representation by $\boldsymbol{\varphi} \vee \boldsymbol{\psi} \stackrel{\text{DEF}}{=} \boldsymbol{\varphi} \cup \boldsymbol{\psi}$ and $\boldsymbol{\varphi} \wedge \boldsymbol{\psi} \stackrel{\text{DEF}}{=} \{X \cup Y \mid X \in \boldsymbol{\varphi}, Y \in \boldsymbol{\psi}\}$. Observe that $\mathrm{DNF}(\phi) \subseteq 2^S$ and in DNF, $\emptyset$ acts as the unit element ($\bot$) of $\vee$, and $\{\emptyset\}$ acts as the unit element ($\top$) of $\wedge$. Intuitively, DNF is going to be used locally on small formulas. A key advantage here is that ACI of the *intended semantics* of the operations is built-in into the set-of-sets representation.

2.3 Words and Streams

A *stream* or *infinite word* $w \in \Sigma^\omega$ is a function from $\mathbb{N}$ to Σ with $w_i \stackrel{\text{DEF}}{=} w(i)$. For *finite* words $v \in \Sigma^*$ the i'th element v_i of v is defined for $0 \leq i < |v|$ where $|v|$ is the length of v. Concatenation of $v \in \Sigma^*$ with $w \in \Sigma^\omega$ (or $w \in \Sigma^*$) is the word or stream $v{\cdot}w$ (or vw) such that if $i < |v|$ then $(vw)_i$ is v_i else $w_{i-|v|}$. Let $w \in \Sigma^\omega$ and $n \in \mathbb{N}$. Let $\mathit{head}(w) \stackrel{\text{DEF}}{=} w_0$. We use the following stream operations that are adopted from the standard `Stream` library in Lean in order to align with the terminology used in the Lean formalization in Section 7.6.

$\mathit{drop}(n, w) \in \Sigma^\omega$ is the suffix of w s.t., for all i, $\mathit{drop}(n, w)_i = w_{n+i}$; $\mathit{tail}(w) \stackrel{\text{DEF}}{=} \mathit{drop}(1, w)$.

$\mathit{take}(n, w) \in \Sigma^*$ is the prefix of w of length n such that $\mathit{take}(n, w)_i = w_i$ for $i < n$.

If $a \in \Sigma$ then $a{::}w$ is the stream with head a and tail w, and a^ω is the stream $(a^\omega)_i = a$ for all i.

If $L \subseteq \Sigma^*$ then L^ω is the set of all $w \in \Sigma^\omega$ for which there exists $\Delta \in \mathbb{N}^\omega$ such that for all i, $\mathit{take}(1{+}\Delta_i, \mathit{drop}(\sum_{j<i}(1{+}\Delta_j), w)) \in L$ where $\sum_{j<0}(\ldots) \stackrel{\text{DEF}}{=} 0$. Note that $L^\omega = (L \setminus \{\epsilon\})^\omega$. For example, $\{a\}^\omega = \{a^\omega\}$ where $\Delta = 0^\omega$.

We also use the following, more light-weight, math notations for stream operations. For $w \in \Sigma^\omega$ and $i \geq 0$, let $w_{..i} \stackrel{\text{DEF}}{=} \mathit{take}(i+1, w)$ and $w_{i..} \stackrel{\text{DEF}}{=} \mathit{drop}(i, w)$. Note that $w_{..0} = w_0$ and $w_{0..} = w$.

2.4 Languages and Derivatives

Let $\mathfrak{D}$ be either Σ^ω or Σ^*, with the associated definition of complement $\complement(L) \stackrel{\text{DEF}}{=} \mathfrak{D} \setminus L$ for $L \subseteq \mathfrak{D}$. The *derivative of L for $a \in \Sigma$* is $\mathbf{D}_a(L) \stackrel{\text{DEF}}{=} \{v \mid av \in L\}$. Irrespective of choice of $\mathfrak{D}$, one can show that $\mathbf{D}_a(\complement(L)) = \complement(\mathbf{D}_a(L))$ and $\mathbf{D}_a(L \cap L') = \mathbf{D}_a(L) \cap \mathbf{D}_a(L')$.

We use the language denotation function $\mathcal{L}$ in relation to Σ^* and $\mathscr{L}$ in relation to Σ^ω.

2.5 Extended Regular Expressions

Consider an EBA $(\Sigma, \mathcal{A}, \vDash, \bot, \top, \sqcup, \sqcap, {}^c)$. $\mathbf{ERE}\langle\mathcal{A}\rangle$ or **ERE** for short is defined by the following abstract grammar, where $\alpha \in \mathcal{A}$. We let $R \in \mathbf{ERE}$, R is called a *regex*. All operators are in order of precedence where $\Cup$ binds weakest. Let $R{+} \stackrel{\text{DEF}}{=} R \cdot R{*}$.

$$R ::= \alpha \mid \varepsilon \mid R_1 \Cup R_2 \mid R_1 \Cap R_2 \mid R_1 \cdot R_2 \mid R{*} \mid {\sim}R$$

Union ($\Cup$), *intersection* ($\Cap$), and *complement* ($\sim$), give rise to the EBA $(\Sigma^*, \mathbf{ERE}, \Vdash, \bot, \top{*}, \Cup, \Cap, \sim)$ where $v \Vdash R \Leftrightarrow v \in \mathcal{L}(R)$ with $\mathcal{L}(R)$ having the standard language semantics for **ERE** where, for all $a \in \Sigma$: $a \vDash \alpha \Leftrightarrow a \Vdash \alpha$. **RE** denotes the *standard* fragment of **ERE** without $\Cap$ and $\sim$. For **ERE** we use *weak equivalence* $\cong$ to simplify regexes in a light-weight manner so that language semantics is preserved: $\Cup$ and $\Cap$ are ACI, ε is the unit of $\cdot$, ${\sim}\bot \cong \top{*}$, and further rules such as ${\sim}\varepsilon \cong \top{+}$.

The standard notation for the regex union operator is |, and & is typically used as the regex intersection operator. Here we use $\Cup$ and $\Cap$ to better align with the Lean formalization in Section 7.6 where the symbols | and & are unavailable to this end.

3 Transition Terms

Let $(\Sigma, \mathcal{A}, \vDash, \bot, \top, \sqcup, \sqcap, ^c)$ be a fixed EBA and let $\mathcal{B}$ be a set or type called *leaves*. Here we provide a generalized view of *transition regexes* from [Stanford et al. 2021] that can be applied both to represent derivatives over Σ^* as well as Σ^ω.The set of *transition terms* $\mathbf{TTerm}\langle\mathcal{A},\mathcal{B}\rangle$ ($\mathbf{TTerm}$ for brevity) is defined by the following abstract grammar where $f \in \mathbf{TTerm}$, $\alpha \in \mathcal{A}$, and $\ell \in \mathcal{B}$:

$$f \quad ::= \quad \ell \quad | \quad (\alpha\,?\,f_1 : f_2)$$

where α is called the *condition*, f_1 the *then-case* and f_2 the *else-case* of $(\alpha\,?\,f_1 : f_2)$. While the formalization of $\mathbf{TTerm}$ as an *inductive type* in Lean in Section 7.6 requires a constructor for leaves, we omit it here to avoid leaf conversion rules (much like $\Sigma \subseteq \Sigma^*$ while Σ^* is `List Σ` in Lean).

We work in a similar but more general setting than if-then-else terms in SMT and primarily want to avoid *lift rules* [Stanford et al. 2021, Section 4.1] that we consider as an implementation aspect of $\mathbf{TTerm}\langle\mathcal{A},\mathcal{B}\rangle$ that depends on $\mathcal{B} = \mathbf{ERE}\langle\mathcal{A}\rangle$. All binary operations $\diamond : \mathcal{B}\times\mathcal{B} \to \mathcal{B}'$, and unary operations $\blacklozenge : \mathcal{B} \to \mathcal{B}'$ are, by definition, always *lifted* to $\mathbf{TTerm}$:

$$(\alpha\,?\,f : g) \diamond h \overset{\text{DEF}}{=} (\alpha\,?\,f \diamond h : g \diamond h) \qquad \ell \diamond (\alpha\,?\,f : g) \overset{\text{DEF}}{=} (\alpha\,?\,\ell \diamond f : \ell \diamond g) \tag{1}$$

$$\blacklozenge(\alpha\,?\,f : g) \overset{\text{DEF}}{=} (\alpha\,?\,\blacklozenge f : \blacklozenge g) \tag{2}$$

When $\bot \in \mathcal{B}$ we use $(\alpha\,?\,f) \overset{\text{DEF}}{=} (\alpha\,?\,f : \bot)$ with $\bot$ as the implicit else-case. Observe therefore that $(\alpha\,?\,f) \diamond g = (\alpha\,?\,f \diamond g : \bot \diamond g)$ and $\blacklozenge(\alpha\,?\,f) = (\alpha\,?\,\blacklozenge f : \blacklozenge\bot)$.

The case when $\mathcal{B}$ and $\mathcal{B}'$ are different types is needed in Sections 5 and 7.3. Given $a \in \Sigma$ and $f \in \mathbf{TTerm}\langle\mathcal{A},\mathcal{B}\rangle$, the *evaluation (leaf) of f for a*, denoted by $f[a]$, is defined as follows.

$$\ell[a] \overset{\text{DEF}}{=} \ell, \qquad (\alpha\,?\,f_1 : f_2)[a] \overset{\text{DEF}}{=} \textbf{if } a \vDash \alpha \textbf{ then } f_1[a] \textbf{ else } f_2[a]$$

Therefore, the following facts hold for all the lifted operations:

$$(f \diamond g)[a] = f[a] \diamond g[a] \wedge (\blacklozenge f)[a] = \blacklozenge(f[a]) \tag{3}$$

Weak equivalence of $f, g \in \mathbf{TTerm}\langle\mathcal{A},\mathcal{B}\rangle$ is defined by $f \cong g \overset{\text{DEF}}{=} \forall a \in \Sigma : f[a] \cong g[a]$.

Leaves of f, $lvs(f)$, and conditions of f, $conds(f)$, are defined by induction over $\mathbf{TTerm}$.

A transition term f is *clean* when all of its branches are feasible, in particular $\forall \ell \in lvs(f) : \exists\, a : f[a] = \ell$. Many rewrites that preserve $\cong$ can be applied incrementally during operations on $\mathbf{TTerm}$ by incorporating *satisfiability* checking of $\mathcal{A}$ to eliminate unreachable subterms. Some rewrites, such as $(\_\,?\,f : f) \cong f$, $(\top\,?\,f : \_) \cong f$, and $(\bot\,?\,\_ : f) \cong f$ are always applied implicitly as trivial condition elminations.

Example 3.1. Let φ belong to $\mathcal{B}$ and let α and β belong to $\mathcal{A}$ such that α implies β. The rewrites below illustrate a typical scenario where the initial transition term (the one in the first row below) is computed and simplified. Let $\mathcal{B}$ be an EBA with disjunction $\vee$, complement $\neg$, bottom element $\bot$, and top element Π (to distinguish it from $\top$ in $\mathcal{A}$). The law of excluded middle in $\mathcal{B}$ is thus $\neg\varphi\vee\varphi \cong \Pi$, and further weak equivalence laws include $\neg\bot \cong \Pi$ and $\Pi \vee \varphi \cong \Pi$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg(\alpha\,?\,\varphi : \bot) \vee (\beta\,?\,\varphi : \bot)}{(\alpha\,?\,(\beta\,?\,\neg\varphi\vee\varphi : \neg\varphi\vee\bot) : (\beta\,?\,\neg\bot\vee\varphi : \neg\bot\vee\bot))}\,(\textit{lift } \neg \text{ and } \vee \text{ to } \mathbf{TTerm}\langle\mathcal{A},\mathcal{B}\rangle)}{(\alpha\,?\,(\beta\,?\,\Pi : \neg\varphi) : (\beta\,?\,\Pi : \Pi))}\,(\textit{rewrites} \text{ of leaves based on } \cong \text{ in } \mathcal{B})}{(\alpha\,?\,(\beta\,?\,\Pi : \neg\varphi) : \Pi)}\,(\textit{trivial condition elimination} \text{ of } \beta)}{(\alpha\,?\,\Pi : \Pi)}\,(\mathbf{UNSAT}(\alpha\sqcap\beta^c) \text{ in } \mathcal{A} \text{ so } (\beta\,?\,\Pi : \neg\varphi) \text{ rewrites to } \Pi)}{\Pi}\,(\textit{trivial condition elimination} \text{ of } \alpha)$$

More generally, $\mathbf{TTerm}\langle\mathcal{A},\mathcal{B}\rangle$ becomes a derived EBA with it own set of rewrite rules. Elimination of unreachable subterms, as with $\neg\varphi$ above, is called *cleaning* in [Stanford et al. 2021]. ⊠

When $\mathcal{B}$ is an EBA modulo $\mathcal{A}$, such as $\mathbf{ERE}\langle\mathcal{A}\rangle$, the *implementation* of $\mathbf{TTerm}\langle\mathcal{A},\mathcal{B}\rangle$ can maintain a symbolic representation of the lifted Boolean operators of $\mathcal{B}$. In particular, it can be beneficial not to always propagate disjunctions into leaves by (1), but to maintain top-level disjunctions as operators between transition terms.

Example 3.2. Consider $\mathbf{TTerm}\langle\mathcal{A},\mathbf{ERE}\langle\mathcal{A}\rangle\rangle$. A typical case is that the symbolic derivative of a regex $\alpha{\cdot}L \cup \beta{\cdot}R$, where α and β are predicates in $\mathcal{A}$, has the form $(\alpha\,?\,L) \cup (\beta\,?\,R)$. Rather than always computing $(\alpha\,?\,(\beta\,?\,L \cup R : L) : (\beta\,?\,R))$ the union operator $\cup$ is maintained as a symbolic operator between the transition regexes as $(\alpha\,?\,L) \cup (\beta\,?\,R)$. We can call it the *Antimirov normal form* in this case, in analogy with [Antimirov 1996]. Similar representation can be maintained for $\mathbf{RLTL}\langle\mathcal{A}\rangle$. One key advatage is to avoid unnecessary cleaning when it is irrelevant how the conditions in the separate disjuncts relate to each other (e.g., whether α and β overlap or not). ⊠

4 Alternating Büchi Word Automata Modulo Theories

We now turn our attention to alternating Büchi automata over words or *ABW* (which admit a linear time translation from LTL [Tsay and Vardi 2021]) and show how they can be generalized using transition terms so as to be modulo $(\Sigma, \mathcal{A}, \vDash, \bot, \top, \sqcup, \sqcap, {}^c)$. The key aspect of the following definition is that transitions in the automata are represented symbolically by transition terms.

An *alternating Büchi automaton modulo* $\mathcal{A}$ ($ABW_{\mathcal{A}}$) is a tuple $M = (\mathcal{A}, Q, \boldsymbol{q}^0, \varrho, F)$ where Q is a *finite* set of *states*, $\boldsymbol{q}^0 \in \mathbb{B}^+(Q)$ is an *initial state combination*, $F \subseteq Q$ is a set of *accepting states*, and $\varrho : Q \to \mathbf{TTerm}\langle\mathcal{A}, \mathbb{B}^+(Q)\rangle$ is a *transition function*.

Two $ABW_{\mathcal{A}}$'s M and N are *compatible* if all shared states of M and N have identical behaviors:

$$\forall q \in Q_M \cap Q_N : \varrho_M(q) = \varrho_N(q) \wedge (q \in F_M \Leftrightarrow q \in F_N)$$

Boolean operations $\diamond \in \{\wedge, \vee\}$ over compatible $M, N \in ABW_{\mathcal{A}}$, are:

$$M \diamond N \stackrel{\text{DEF}}{=} (\mathcal{A},\ Q_M \cup Q_N,\ \boldsymbol{q}^0_M \diamond \boldsymbol{q}^0_N,\ \varrho_M \cup \varrho_N,\ F_M \cup F_N)$$

In figures we show transition terms $\varrho(q)$ by *symbolic transitions* $q \xrightarrow{\beta} \phi$ where $\phi \in \mathbb{B}^+(Q)$ is a leaf of $\varrho(q)$ *guarded* by the branch condition $\beta \in \mathcal{A}$ from the root of $\varrho(q)$ to ϕ. The automaton M (or ϱ_M) is *clean* when all the transition terms in ϱ_M are clean. For example, the automaton in Figure 1 is clean when both α and α^c are satisfiable.

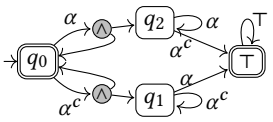


Fig. 1. $ABW_{\mathcal{A}}$.

The *DNF normal form* of ϱ, $\mathrm{DNF}(\varrho)$, maps $q \in Q$ to $\mathrm{DNF}(\varrho(q))$. Let $\boldsymbol{\varrho} = \mathrm{DNF}(\varrho)$. We let also $\mathit{conds}(M) \stackrel{\text{DEF}}{=} \bigcup_{q\in Q} \mathit{conds}(\varrho(q))$ denote the set of all conditions in ϱ. E.g., in Figure 1, $\mathit{conds}(M) = \{\alpha\}$ and $\boldsymbol{\varrho}(q_0) = (\alpha\,?\,\{\{q_2, q_0\}\} : \{\{q_1, q_0\}\})$.

4.1 Runs and Languages

Let $M = (\mathcal{A}, Q, \boldsymbol{q}^0, \varrho, F)$ be an $ABW_{\mathcal{A}}$. The standard definition of runs [Vardi 1995] uses $X \subseteq Q$ to specify *satisfying assignments* for $\phi \in \mathbb{B}^+(Q)$ as follows where $\vDash$ is standard (Tarski) semantics:

$$X \mathit{\ satisfies\ } \phi \stackrel{\text{DEF}}{=} \bigcup_{q\in X}\{q \mapsto \mathbf{true}\} \cup \bigcup_{q\in Q\setminus X}\{q \mapsto \mathbf{false}\} \vDash \phi$$

Lemma 1. $\forall \phi \in \mathbb{B}^+(Q), X \subseteq Q : X \mathit{\ satisfies\ } \phi \Leftrightarrow \exists Y \subseteq X : Y \in \mathrm{DNF}(\phi)$.

Proof. We use that X satisfies ϕ iff some $X_{\min} \subseteq X$ *satisfies* ϕ *in a minimal manner* [Kupferman 2018, 4.5.1], that we can formally define here by using $\mathrm{DNF}(\phi)$ (recall that $\mathrm{DNF}(\phi) \subseteq 2^Q$):

$$X_{\min} \in \mathit{antichain}_{\subseteq}(\mathrm{DNF}(\phi)) \qquad \mathit{antichain}_{\subseteq}(X) \stackrel{\text{DEF}}{=} \{x \in X \mid \nexists\, y \in X : y \subsetneq x\} \tag{4}$$

Note that $\mathit{antichain}_{\subseteq}(\mathrm{DNF}(\phi)) \subseteq \mathrm{DNF}(\phi)$ and if $Y \in \mathrm{DNF}(\phi)$ then Y *satisfies* ϕ since $\phi \in \mathbb{B}^+(Q)$. □

Let $aw \in \Sigma^\omega$ and $q \in Q$. A *run for aw from q* is a (potentially *infinite*) Q-labelled tree τ whose root node has label q and, by using Lemma 1, there exists $X \in \mathrm{DNF}(\varrho(q)[a])$ such that, for each $p \in X$, there is an immediate subtree τ_p of τ such that τ_p is a run for w from p.

A run τ is *accepting* if *all* of the infinite branches of τ visit F infinitely often, where an *infinite branch* visits the root of τ and then continues as an infinite branch from some immediate subtree of τ. For $w \in \Sigma^\omega$ and $q \in Q$: *w is accepted from q* iff there exists an accepting run for w from q; the *language of q in M* is defined and lifted to $\mathbb{B}^+(Q)$ as follows:

$$\begin{array}{c} \mathscr{L}_M(q) \stackrel{\text{DEF}}{=} \{w \in \Sigma^\omega \mid w \text{ is accepted from } q\} \\ \mathscr{L}_M(\mathbf{q} \vee \mathbf{p}) \stackrel{\text{DEF}}{=} \mathscr{L}_M(\mathbf{q}) \cup \mathscr{L}_M(\mathbf{p}) \quad \mathscr{L}_M(\mathbf{q} \wedge \mathbf{p}) \stackrel{\text{DEF}}{=} \mathscr{L}_M(\mathbf{q}) \cap \mathscr{L}_M(\mathbf{p}) \quad \mathscr{L}(M) \stackrel{\text{DEF}}{=} \mathscr{L}_M(\boldsymbol{q}^0) \end{array} \tag{5}$$

For the positive Boolean operations it follows that $\mathscr{L}(M \wedge N) = \mathscr{L}(M) \cap \mathscr{L}(N)$ and $\mathscr{L}(M \vee N) = \mathscr{L}(M) \cup \mathscr{L}(N)$, as in the classical case [Kupferman 2018, Theorem 20]. Complementation of an $ABW_{\mathcal{A}}$ M is theoretically possible via alternation elimination (Section 5) and mintermization (Section 4.1.1) followed by [Kupferman 2018, Theorem 4], but the existence of a *symbolic* algorithm that would construct an $ABW_{\mathcal{A}}$ $\neg M$ from M such that $\mathscr{L}(\neg M) = \complement(\mathscr{L}(M))$ is an open problem.

We adopt the same notation also for classical ABW M that $\mathscr{L}_M(\mathbf{q})$ denotes the language of M accepted starting from the state combination $\mathbf{q} \in \mathbb{B}^+(Q_M)$.

4.1.1 From $ABW_{\mathcal{A}}$ to Classical ABW. A variety of decidability results for $ABW_{\mathcal{A}}$ follow directly by reduction to classical ABW (this does not imply a practical implementation scheme, as it requires mintermization). Let $A = \mathit{Minterms}(\mathit{conds}(M))$ be a finite alphabet and for $a \in \Sigma$ let $\widehat{a} \in A$ denote the minterm such that $a \vDash \widehat{a}$. Let $\widehat{M}$ denote the classical ABW $(A, Q, \boldsymbol{q}^0, \widehat{\varrho}, F)$ where $\widehat{\varrho}(q, \widehat{a}) \stackrel{\text{DEF}}{=} \varrho(q)[a]$ and lift $\widehat{a} \in A$ to $\widehat{w} \in A^\omega$. Then

Lemma 2. $\forall M \in ABW_{\mathcal{A}}, \phi \in \mathbb{B}^+(Q_M), w \in \Sigma^\omega : \widehat{w} \in \mathscr{L}_{\widehat{M}}(\phi) \Leftrightarrow w \in \mathscr{L}_M(\phi)$.

In particular, *decidability* of $\mathscr{L}(M) \neq \emptyset$, provided that $\mathcal{A}$ is decidable, follows from decidability of $\mathscr{L}(\widehat{M}) \neq \emptyset$ [Miyano and Hayashi 1984; Rabin 1970].

4.1.2 From Classical ABW to $ABW_{\mathcal{A}}$. In the opposite direction, consider a classical ABW $M = (2^P, Q, \boldsymbol{q}^0, \rho, F)$ where P is a finite set of propositions. For all $a \subseteq P$ let $\alpha_a \stackrel{\text{DEF}}{=} (\sqcap_{p \in a} p) \sqcap (\sqcap_{p \in P \setminus a} p^c)$. M is lifted to $M_{\mathcal{A}} = (\mathcal{A}, Q, \boldsymbol{q}^0, \varrho, F)$ where the EBA for $\mathcal{A}$ is essentially a SAT solver for $\mathbb{B}(P)$ and ϱ maps $q \in Q$ to $\bigvee_{a \subseteq P} (\alpha_a \,?\, \rho(q, a))$. Then $\varrho(q)[a] = \rho(q, a)$ and so $\mathscr{L}(M) = \mathscr{L}(M_{\mathcal{A}})$.

4.2 Nondeterministic Büchi Word Automata Modulo Theories

An $ABW_{\mathcal{A}}$ M is *nondeterministic*, an $NBW_{\mathcal{A}}$, if all leaves of all transition terms in ϱ_M and $\boldsymbol{q}^0_M$ are disjunctions of states. One can view an $NBW_{\mathcal{A}}$ equivalently as a finite automaton modulo $\mathcal{A}$ [D'Antoni and Veanes 2021] but with the Büchi acceptance condition, i.e., that each transition term $\varrho_M(q)$ for $q \in Q_M$ has the equivalent representation as $\bigvee_{i=1}^n (\alpha_i \,?\, q_i)$ where $q_i \in Q_M$.

Decidability of nonemptiness of a clean $NBW_{\mathcal{A}}$ has *linear time* complexity, that follows by adapting the corresponding result from [Emerson and Lei 1985, 1987], where cleaning, as a separate preprocessing step, has computational complexity that depends on that of satisfiability in $\mathcal{A}$ but is otherwise linear in the size of M. In particular, mintermization, i.e., conversion to the classical setting, is avoided.

*NBW*s play a prominent role in model-checking. Of particular interest is *product* of *NBW*s that is used to determine if a specification of bad behavior, translated into an *NBW*, intersects with a system model that is also given as an *NBW*. The ability to define and work with such models *modulo background theories*, i.e., with $NBW_{\mathcal{A}}$ directly, can increase their succinctness by an *exponential* factor without later on incurring that cost in algorithms, as shown in Section 5.3.

The alternation elimination algorithm Æ in Section 5 is used to define a product algorithm for $NBW_{\mathcal{A}}$'s in Section 5.3 that is *quadratic* in the size of the automata provided that satisfiability checking in $\mathcal{A}$ has constant cost. Most importantly, mintermization is avoided.

5 Alternation Elimination

Although alternating Büchi automata are an attractive target for temporal logic, incurring no space blowup, the problem of testing for nonemptiness is harder for alternating automata (due to the presence of both $\wedge$ and $\vee$ in the state formula) than non-alternating automata [Boker et al. 2010].

Let $M = (\mathcal{A}, Q, \boldsymbol{q}^0, \varrho, F)$ be an $ABW_{\mathcal{A}}$ modulo $(\Sigma, \mathcal{A}, \models, \bot, \top, \sqcup, \sqcap, ^c)$. We introduce an alternation elimination algorithm for M that is a derivative-based generalization of the algorithm in [Miyano and Hayashi 1984] (see also [Vardi 1995, Proposition 20]). The algorithm constructs an $NBW_{\mathcal{A}}$ $N = (\mathcal{A}, S, S_0, \sigma, F_N)$ accepting the same language as M. As shown by [Boker et al. 2010], the concept behind [Miyano and Hayashi 1984] accurately captures the prime concern in alternation removal, which is the need to associate the states of the equivalent $NBW_{\mathcal{A}}$ N with two sets of states from M, which means that a $\Omega(3^{|Q|})$ space blowup cannot be avoided. Let $\boldsymbol{\varrho} = \mathrm{DNF}(\varrho)$.

In the rest of this section let $\boldsymbol{Q} = 2^{2^Q}$, $P = 2^Q \times 2^Q$, and $\boldsymbol{\mathcal{P}} = 2^P$. $\boldsymbol{Q}$ is the leaf type of transition terms of M in DNF, i.e., $\boldsymbol{\varrho} : Q \to \mathbf{TTerm}\langle \mathcal{A}, \boldsymbol{Q}\rangle$. $\boldsymbol{\mathcal{P}}$ is the leaf type of transition terms of N where P is the *state type* of N, i.e., $\sigma : S \to \mathbf{TTerm}\langle \mathcal{A}, \boldsymbol{\mathcal{P}}\rangle$. $F_N = \{\langle U, V\rangle \in S \mid U = \emptyset\}$.

5.1 Algorithm Æ

We start by defining the key operation $@$ called *alternation product* or *alternation times* of $f, g \in \mathbf{TTerm}\langle \mathcal{A}, \boldsymbol{Q}\rangle$ as a term $f \mathbin{@} g \in \mathbf{TTerm}\langle \mathcal{A}, \boldsymbol{\mathcal{P}}\rangle$. Let $\boldsymbol{\varphi}, \boldsymbol{\psi} \in \boldsymbol{Q}$ and first define $@ : \boldsymbol{Q} \times \boldsymbol{Q} \to \boldsymbol{\mathcal{P}}$:

$$\boldsymbol{\varphi} \mathbin{@} \boldsymbol{\psi} \stackrel{\text{DEF}}{=} \{\langle X \setminus F, Y \cup (X \cap F)\rangle \mid X \in \boldsymbol{\varphi}, Y \in \boldsymbol{\psi}\} \tag{6}$$

The purpose of $@$ is to partition the states into those that have not yet visited a final state ($X \setminus F$) and those that have ($Y \cup (X \cap F)$). Observe that $@$ is not symmetric and $\{\emptyset\}$ intuitively denotes $\top$ in $\boldsymbol{Q}$. For example, if $q \notin F$ then $\{\{q\}\} \mathbin{@} \{\emptyset\} = \langle \{q\}, \emptyset\rangle$ is not a final state of N while $\{\emptyset\} \mathbin{@} \{\{q\}\} = \langle \emptyset, \{q\}\rangle$ is a final state of N. Recall that (1) lifts $@$ to $\mathbf{TTerm}\langle \mathcal{A}, \boldsymbol{Q}\rangle \times \mathbf{TTerm}\langle \mathcal{A}, \boldsymbol{Q}\rangle \to \mathbf{TTerm}\langle \mathcal{A}, \boldsymbol{\mathcal{P}}\rangle$.

For $h \in \mathbf{TTerm}\langle \mathcal{A}, \boldsymbol{\mathcal{P}}\rangle$ let $\mathit{states}(h) \stackrel{\text{DEF}}{=} \bigcup \mathit{lvs}(h)$. So $\mathit{states}(h) \subseteq P$. The algorithm Æ constructs an equivalent $NBW_{\mathcal{A}}$ from M. For $U, V \subseteq Q$ let

$$\boldsymbol{\varrho}(U) \stackrel{\text{DEF}}{=} \begin{cases} \{\emptyset\}, & \textbf{if } U = \emptyset; \\ \bigwedge_{q \in U} \boldsymbol{\varrho}(q), & \textbf{otherwise}. \end{cases}$$

$$@_{\boldsymbol{\varrho}}(\langle U, V\rangle) \stackrel{\text{DEF}}{=} \begin{cases} \boldsymbol{\varrho}(V) \mathbin{@} \{\emptyset\}, & \textbf{if } U = \emptyset; \\ \boldsymbol{\varrho}(U) \mathbin{@} \boldsymbol{\varrho}(V), & \textbf{otherwise}. \end{cases}$$

and let $\boldsymbol{dom}(\sigma)$ denote the domain of the partially defined transition function σ in the while loop of the Æ algorithm that computes the fixpoint of S:

$$Æ(M) \stackrel{\text{DEF}}{=} \begin{cases} S_0 = \mathrm{DNF}(\boldsymbol{q}^0) \mathbin{@} \{\emptyset\}; S \leftarrow S_0; \sigma \leftarrow \emptyset; \\ \textbf{while } \exists s \in S \setminus \boldsymbol{dom}(\sigma) \textbf{ let } f = @_{\boldsymbol{\varrho}}(s) \textbf{ in } \ \sigma(s) \leftarrow f;\ S \leftarrow S \cup \mathit{states}(f) \\ \textbf{return } (\mathcal{A}, S, S_0, \sigma, \{\langle U, V\rangle \in S \mid U = \emptyset\}) \end{cases}$$

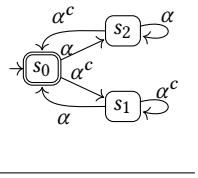
The key power of the algorithm lies in its modular use of **TTerm** that enables eager simplification of transition terms $f \mathbin{@} g$ by utilizing satisfiability checking in $\mathcal{A}$, both by cleaning and by rules like Lemma 3 below. That said, simplifications in **TTerm**, while essential for implementing Æ, require satisfiability in $\mathcal{A}$ to be *decidable*, which is not necessary for the formal correctness (Theorem 1).

We make use of the following *state reduction* lemma in the following example, where the number of states would otherwise *double*.

Lemma 3 (State Reduction). *If $U' \subseteq U$, $V' \subseteq V$, and $U = \emptyset \Leftrightarrow U' = \emptyset$ then if $\boldsymbol{\varrho}(U') = \boldsymbol{\varrho}(U)$ and $\boldsymbol{\varrho}(V') = \boldsymbol{\varrho}(V)$ then $\langle U, V\rangle$ can be replaced by $\langle U', V'\rangle$ in the Æ algorithm.*

Proof. The computation of f preserves language equivalence if we identify states with identical transition terms and acceptance conditions. □

Example 5.1. Let M be as in Figure 1, where $\boldsymbol{q}^0 = q_0$, $F = \{q_0, \top\}$, and
$\varrho = \{q_0 \mapsto (\alpha\,?\,q_2 \wedge q_0 : q_1 \wedge q_0),\ q_1 \mapsto (\alpha\,?\,\top : q_1),\ q_2 \mapsto (\alpha\,?\,q_2 : \top),\ \top \mapsto \top\}$
Each row of the following table represents one iteration of the while-loop in the run of $Æ(M)$. Since $q_0 \in F$, observe that $\mathrm{DNF}(q_0) \mathbin{@} \{\emptyset\} = \{\{q_0\}\} \mathbin{@} \{\emptyset\} = \{\langle\emptyset, \{q_0\}\rangle\}$.

s	$\sigma(s)$	Resulting $NBW_{\mathcal{A}}$
$s_0{=}\langle\emptyset, \{q_0\}\rangle$	$(\alpha\,?\,\{\{q_2, q_0\}\} : \{\{q_1, q_0\}\}) \mathbin{@} \{\emptyset\}$ $\cong (\alpha\,?\,\{\langle\{q_2\}, \{q_0\}\rangle\} : \{\langle\{q_1\}, \{q_0\}\rangle\})$	
$s_1{=}\langle\{q_1\}, \{q_0\}\rangle$	$\boldsymbol{\varrho}(q_1) \mathbin{@} \boldsymbol{\varrho}(q_0)$ $\cong (\alpha\,?\,\{\langle\emptyset, \{q_2, q_0\}\rangle\} : \{\langle\{q_1\}, \{q_1, q_0\}\rangle\})$ (Lma 3) $\cong (\alpha\,?\,\{\langle\emptyset, \{q_0\}\rangle\} : \{\langle\{q_1\}, \{q_0\}\rangle\})$	
$s_2{=}\langle\{q_2\}, \{q_0\}\rangle$	(Lma 3) $\cong (\alpha\,?\,\{\langle\{q_2\}, \{q_0\}\rangle\} : \{\langle\emptyset, \{q_0\}\rangle\})$	

The constructions of $\sigma(s_1)$ and $\sigma(s_2)$ applied Lemma 3 whereby both $\{q_1, q_0\}$ and $\{q_2, q_0\}$ simplify to $\{q_0\}$. Cleaning was used to eliminate nested ITEs with condition α. ⊠

5.2 Correctness of Æ

The correctness proof of the algorithm is by reduction to [Miyano and Hayashi 1984], using generic properties of transition terms and mintermization. Given a pair $\langle U, V\rangle$ of finite sets U, V of states let $\wedge\langle U, V\rangle \stackrel{\text{DEF}}{=} \bigwedge(U \cup V)$.

Theorem 1 (Æ). *$\forall q \in Q_{Æ(M)}$: $\mathscr{L}_{Æ(M)}(q) = \mathscr{L}_M(\wedge q)$ and $\mathscr{L}(Æ(M)) = \mathscr{L}(M)$.*

Proof. Let $N = Æ(M)$ as above. The algorithm terminates because P is finite and $S \subseteq P$. First let $q = \langle U, V\rangle$ be such that $U \neq \emptyset$. Note that $V = \emptyset$ implies that $\boldsymbol{\varrho}(V) = \{\emptyset\}$.

$$\sigma(\langle U, V\rangle)[a] = (\boldsymbol{\varrho}(U) \mathbin{@} \boldsymbol{\varrho}(V))[a]$$

$$= (\mathrm{DNF}(\textstyle\bigwedge_{t\in U} \varrho(t)) \mathbin{@} \mathrm{DNF}(\bigwedge_{t\in V} \varrho(t)))[a] \overset{(3)}{=} \mathrm{DNF}(\textstyle\bigwedge_{t\in U}(\varrho(t)[a])) \mathbin{@} \mathrm{DNF}(\bigwedge_{t\in V}(\varrho(t)[a]))$$

$$\overset{(6)}{=} \{\langle X\backslash F, Y \cup (X \cap F)\rangle \mid X \in \mathrm{DNF}(\textstyle\bigwedge_{t\in U}(\varrho(t)[a])), Y \in \mathrm{DNF}(\bigwedge_{t\in V}(\varrho(t)[a]))\} \tag{7}$$

Now let $q = \langle U, V\rangle$ be such that $U = \emptyset$. We get that

$$\sigma(\langle U, V\rangle)[a] = (\boldsymbol{\varrho}(V) \mathbin{@} \{\emptyset\})[a] = (\mathrm{DNF}(\textstyle\bigwedge_{t\in V} \varrho(t)) \mathbin{@} \{\emptyset\})[a] \overset{(3)}{=} \mathrm{DNF}(\textstyle\bigwedge_{t\in V}(\varrho(t)[a])) \mathbin{@} \{\emptyset\}$$

$$\overset{(6)}{=} \{\langle X\backslash F, X \cap F\rangle \mid X \in \mathrm{DNF}(\textstyle\bigwedge_{t\in V}(\varrho(t)[a]))\} \tag{8}$$

Next, we relate (7) and (8) with $\widehat{M}$ (recall Section 4.1), where for $W \subseteq Q$, $\bigwedge_{t\in W}(\varrho(t)[a]) = \bigwedge_{t\in W} \widehat{\varrho}(t, \widehat{a})$ – (7) and (8) represent the construction used in [Miyano and Hayashi 1984] (MH84) on top of which further analysis over accepting runs of $\widehat{M}$ is built, and where choosing all satisfiers of ϕ from $\mathrm{DNF}(\phi)$ is sufficient by Lemma 1. It follows, for all $s \in S$ and $w \in \Sigma^\omega$, that $w \in \mathscr{L}_N(s)$ $\Leftrightarrow$(by Lemma 2) $\widehat{w} \in \mathscr{L}_{\widehat{N}}(s)$ $\Leftrightarrow$(by MH84) $\widehat{w} \in \mathscr{L}_{\widehat{M}}(\wedge s)$ $\Leftrightarrow$(by Lemma 2) $w \in \mathscr{L}_M(\wedge s)$.
So $\mathscr{L}(M) = \mathscr{L}_M(\boldsymbol{q}^0) = \bigcup_{X\in\mathrm{DNF}(\boldsymbol{q}^0)} \mathscr{L}_M(\bigwedge X) = \bigcup_{s\in S_0} \mathscr{L}_M(\wedge s) = \bigcup_{s\in S_0} \mathscr{L}_N(s) = \mathscr{L}(N)$. □

5.3 Product of NBW Modulo Theories

A direct application of Æ is the *product* of two compatible $NBW_{\mathcal{A}}$'s $N_i = (\mathcal{A}, Q_i, Q_i^0, \varrho_i, F_i)$, for $i \in \{1,2\}$ as the $NBW_{\mathcal{A}}$:

$$N_1 \times N_2 \stackrel{\text{DEF}}{=} Æ(N_1 \wedge N_2)$$

Product provides an essential step in the automata-based approach to model checking, as discussed in the introduction. We get the following corollary of Theorem 1. Assume, w.l.o.g., that $Q_1 \cap Q_2 = \emptyset$.

Corollary 1 (Product). $|Q_{N_1 \times N_2}| \leq 4|Q_1||Q_2| \wedge \forall \langle U, V\rangle \in Q_{N_1 \times N_2}$:

(1) $U \cap V = \emptyset \wedge U \cap (F_1 \cup F_2) = \emptyset$
(2) $\exists\, q \in Q_1, p \in Q_2 : U \cup V = \{q, p\} \wedge \mathscr{L}_{N_1 \times N_2}(\langle U, V\rangle) = \mathscr{L}_{N_1}(q) \cap \mathscr{L}_{N_2}(p)$.

Recall that, in an $NBW_{\mathcal{A}}$, $\varrho(q)$ is maintained as a disjunction of symbolic transitions representing the transition term $\bigvee_{i\in I} (\alpha_i \,?\, \mathbf{q}_i)$ where $\mathbf{q}_i = \{\{q_i\}\}$ for some $q_i \in Q$. (Recall that $(\alpha_i \,?\, \mathbf{q}_i)$ stands for $(\alpha_i \,?\, \mathbf{q}_i : \emptyset)$ here, where $\emptyset$ acts as $\bot$ in DNF.) The alternation product $\boldsymbol{\varrho}(q) \mathbin{@} \boldsymbol{\varrho}(p)$ can then be restated more compactly as:

$$\bigvee_{i\in I} (\alpha_i \,?\, \mathbf{q}_i) \mathbin{@} \bigvee_{j\in J} (\beta_j \,?\, \mathbf{p}_j) \stackrel{\text{DEF}}{=} \bigvee_{i\in I, j\in J, \mathbf{SAT}(\alpha_i \sqcap \beta_j)} (\alpha_i \sqcap \beta_j \,?\, \mathbf{q}_i \mathbin{@} \mathbf{p}_j)$$

where *cleaning* ($\mathbf{SAT}(\alpha_i \sqcap \beta_j)$) is embedded. Observe that $\emptyset \mathbin{@} \mathbf{p} = \mathbf{q} \mathbin{@} \emptyset = \emptyset$, so

$$(\alpha \,?\, \mathbf{q}) \mathbin{@} (\beta \,?\, \mathbf{p}) = (\alpha \,?\, (\beta \,?\, \mathbf{q} \mathbin{@} \mathbf{p} : \mathbf{q} \mathbin{@} \emptyset) : (\beta \,?\, \emptyset \mathbin{@} \mathbf{p} : \emptyset \mathbin{@} \emptyset)) \cong (\alpha \,?\, (\beta \,?\, \mathbf{q} \mathbin{@} \mathbf{p})) \cong (\alpha \sqcap \beta \,?\, \mathbf{q} \mathbin{@} \mathbf{p})$$

This means that, also the number of satisfiability tests is $O(|N_1||N_2|)$, that eliminate unreachable states. *In stark contrast, a translation to classical NBWs in $O(2^{|N_1|+|N_2|})$ due to the mintermization or bitblasting needed to establish the finite alphabet.*

6 LTL Modulo Theories

Here we lift classical linear temporal logic (LTL) to be modulo $(\Sigma, \mathcal{A}, \vDash, \bot, \top, \sqcup, \sqcap, ^c)$, or $\mathbf{LTL}\langle\mathcal{A}\rangle$, via symbolic derivatives that translate $\mathbf{LTL}\langle\mathcal{A}\rangle$ formulas into transition terms. We then utilize the algorithms developed for $ABW_{\mathcal{A}}$ for decision procedures of $\mathbf{LTL}\langle\mathcal{A}\rangle$. Let $\alpha \in \mathcal{A}$ and $\varphi \in \mathbf{LTL}\langle\mathcal{A}\rangle$.

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \,\mathbf{U}\, \varphi_2 \mid \varphi_1 \,\mathbf{R}\, \varphi_2$$

where $\mathbf{X}$ (*Next*), $\mathbf{U}$ (*Until*), and $\mathbf{R}$ (*Release*), are *modal* operators. The *true* formula is $\top$ and the *false* formula is $\bot$ from $\mathcal{A}$. The following standard abbreviations are also used: $\varphi \rightarrow \psi \stackrel{\text{DEF}}{=} \neg\varphi \vee \psi$, $\mathbf{F}\psi \stackrel{\text{DEF}}{=} \top \,\mathbf{U}\, \psi$, and $\mathbf{G}\psi \stackrel{\text{DEF}}{=} \bot \,\mathbf{R}\, \psi$, where $\mathbf{F}$ is *Finally (Eventually)* and $\mathbf{G}$ is *Globally (Always)*.

6.1 Semantics

A stream $w \in \Sigma^\omega$ is a *model of* $\varphi \in \mathbf{LTL}\langle\mathcal{A}\rangle$, $w \vDash \varphi$, when the following holds, where $\alpha \in \mathcal{A}$:

$$w \vDash \alpha \stackrel{\text{DEF}}{=} w_0 \vDash \alpha \tag{9}$$

$$w \vDash \varphi \wedge \psi \stackrel{\text{DEF}}{=} w \vDash \varphi \;\mathbf{\wedge}\; w \vDash \psi \qquad w \vDash \varphi \vee \psi \stackrel{\text{DEF}}{=} w \vDash \varphi \;\mathbf{\vee}\; w \vDash \psi \qquad w \vDash \neg\varphi \stackrel{\text{DEF}}{=} w \nvDash \varphi \tag{10}$$

$$w \vDash \mathbf{X}\psi \stackrel{\text{DEF}}{=} tail(w) \vDash \psi \tag{11}$$

$$w \vDash \varphi \,\mathbf{U}\, \psi \stackrel{\text{DEF}}{=} \exists j : drop(j, w) \vDash \psi \wedge \forall i{<}j : drop(i, w) \vDash \varphi \tag{12}$$

$$w \vDash \varphi \,\mathbf{R}\, \psi \stackrel{\text{DEF}}{=} \forall j : drop(j, w) \vDash \psi \;\;\mathbf{\vee}\;\; \exists j : drop(j, w) \vDash \varphi \wedge \forall i{\leq}j : drop(i, w) \vDash \psi \tag{13}$$

$$\mathscr{L}(\varphi) \stackrel{\text{DEF}}{=} \{w \in \Sigma^\omega \mid w \vDash \varphi\} \qquad \varphi \equiv \psi \stackrel{\text{DEF}}{=} \mathscr{L}(\varphi) = \mathscr{L}(\psi) \tag{14}$$

The rules (12) and (13) are duals of each other, either one suffices as the main definition. Observe that if $\alpha \in \mathcal{A}$ then $w \vDash \neg\alpha \Leftrightarrow w \nvDash \alpha \Leftrightarrow w_0 \nvDash \alpha \Leftrightarrow w_0 \vDash \alpha^c \Leftrightarrow w \vDash \alpha^c$.

The following examples illustrate some cases of $\mathbf{LTL}\langle\mathcal{A}\rangle$ modulo various $\mathcal{A}$. Example 6.1 illustrates – *at a very abstract level* – the well-known connection of integrating SAT solving into *symbolic LTL*, e.g., by using BDDs [Wulf et al. 2008].

Example 6.1. Classical LTL over a set of atomic propositions P is $\mathbf{LTL}\langle\mathcal{A}\rangle$ where $\mathcal{A}$ is an algebra of Boolean combinations over P, where $\Sigma = 2^P$ and $\mathcal{A} = \mathbb{B}(P)$, and where satisfiability can be implemented using a SAT solver. An element $d \in \Sigma$ such that $d \models \alpha$ defines a *truth assignment* to P that makes α true. For example, if $P = \{p_i\}_{i<7}$ and $\alpha = p_6 \sqcap p_5 \sqcap p_4 \sqcap (p_3 \sqcup ((p_2^c) \sqcap p_1))$ then if $w \in \Sigma^\omega$ is such that $w_0 = \{p_1, p_4, p_5, p_6\}$ and $w_1 = \{p_1, p_2, p_4, p_5, p_6\}$ then $w \models \alpha$ but $\mathit{tail}(w) \not\models \alpha$. Thus, for example, $w \not\models \mathbf{G}\alpha$. ⊠

Unlike in Example 6.1, in Example 6.2 Σ is *infinite*.

Example 6.2. Consider LTL modulo the EBA $(\Sigma, \mathcal{A}, \models, \bot, \top, \vee, \wedge, \neg)$ of SMT formulas $\mathcal{A}$ that can be implemented using an SMT solver. Here Σ is the set of interpretations to *uninterpreted constants* or *variables*. Let x be a variable of type *real*. Then $(x < 1)\ \mathbf{R}\ (0 < x)$ states that x must remain positive until $x < 1$. E.g., $(x{\mapsto}1)(x{\mapsto}\frac{1}{2})(x{\mapsto}0)^\omega \models (x < 1)\ \mathbf{R}\ (0 < x)$. ⊠

6.2 Vardi Derivatives of LTL Modulo Theories

Here we show how the semantics of $\mathbf{LTL}\langle\mathcal{A}\rangle$, or **LTL** for short, can be realized via transition terms. The key observation here is that a concrete derivative (i.e., for a given symbol $a \in \Sigma$) is not actually constructed but maintained in a symbolic form as a transition term. Let $\alpha \in \mathcal{A}$, and $\varphi, \psi \in \mathbf{LTL}$. The *(symbolic) derivative* of an **LTL** formula is defined as a term in $\mathbf{TTerm}\langle\mathcal{A}, \mathbf{LTL}\rangle$:

$$\partial(\alpha) \overset{\text{DEF}}{=} (\alpha\ \mathbf{?}\ \top\ \mathbf{:}\ \bot) \tag{15}$$
$$\partial(\varphi \wedge \psi) \overset{\text{DEF}}{=} \partial(\varphi) \wedge \partial(\psi) \tag{16}$$
$$\partial(\varphi \vee \psi) \overset{\text{DEF}}{=} \partial(\varphi) \vee \partial(\psi) \tag{17}$$
$$\partial(\neg\varphi) \overset{\text{DEF}}{=} \neg\partial(\varphi) \tag{18}$$
$$\partial(\mathbf{X}\psi) \overset{\text{DEF}}{=} \psi \tag{19}$$
$$\partial(\varphi\ \mathbf{U}\ \psi) \overset{\text{DEF}}{=} \partial(\psi) \vee (\partial(\varphi) \wedge (\varphi\ \mathbf{U}\ \psi)) \tag{20}$$
$$\partial(\varphi\ \mathbf{R}\ \psi) \overset{\text{DEF}}{=} \partial(\psi) \wedge (\partial(\varphi) \vee (\varphi\ \mathbf{R}\ \psi)) \tag{21}$$

Symbolic derivatives, as given above, lift the construction in [Vardi 1995] (see also [Kupferman 2018, Theorem 24]) as to be modulo $\mathcal{A}$. We use the following simplified rules for $\mathbf{G}$ and $\mathbf{F}$ (where $\bot \vee \phi \cong \phi$ and $\top \wedge \phi \cong \phi$ are applied implicitly): $\partial(\mathbf{G}\psi) \overset{\text{DEF}}{=} \partial(\psi) \wedge \mathbf{G}\psi$ and $\partial(\mathbf{F}\psi) \overset{\text{DEF}}{=} \partial(\psi) \vee \mathbf{F}\psi$. Correctness of the derivation rules will follow as a special case of Theorem 4 in Section 7.

We now link derivatives formally with *ABW*s. We write $\mathbf{LTL}^+$ for the *positive* formulas in **LTL** where $\neg$ does not occur. This is a standard normal form assumption and every formula in **LTL** can be translated into $\mathbf{LTL}^+$ of the same size if members of $\mathcal{A}$ are treated as units. The particular aspect with modulo $\mathcal{A}$ is that *complement is propagated into* $\mathcal{A}$, i.e., for $\alpha \in \mathcal{A}$, the **LTL** formula $\neg\alpha$ becomes the predicate α^c in $\mathcal{A}$.

Definition 6.3 ($\mathbf{LTL}^+\langle\mathcal{A}\rangle$). The *positive fragment of* $\mathbf{LTL}\langle\mathcal{A}\rangle$ is obtained by eliminating any explicit use of $\neg$ through de Morgan's laws and the standard equivalence preserving rules where $\alpha \in \mathcal{A}$ and $\varphi, \psi \in \mathbf{LTL}$: $\neg\alpha \equiv \alpha^c$, $\neg\mathbf{X}\varphi \equiv \mathbf{X}\neg\varphi$, $\neg(\varphi\ \mathbf{U}\ \psi) \equiv \neg\varphi\ \mathbf{R}\ \neg\psi$, and $\neg(\varphi\ \mathbf{R}\ \psi) \equiv \neg\varphi\ \mathbf{U}\ \neg\psi$.

Definition 6.4 (M_ϕ for $\phi \in \mathbf{LTL}^+\langle\mathcal{A}\rangle$). For $\phi \in \mathbf{LTL}^+\langle\mathcal{A}\rangle$ let Q_ϕ denote the set containing $\bot$ and $\top$ and all *non-Boolean* subformulas of ϕ, i.e., predicates in $\mathcal{A}$, and all *modal* formulas. Let ϱ_ϕ denote $\lambda q.\partial(q)$ for $q \in Q_\phi$. Let F_ϕ contain all the $\mathbf{R}$-formulas and $\top$ in Q_ϕ. Then $M_\phi \overset{\text{DEF}}{=} (\mathcal{A}, Q_\phi, \phi, \varrho_\phi, F_\phi)$. Observe that $\varrho_\phi : Q_\phi \to \mathbf{TTerm}\langle\mathcal{A}, \mathbb{B}^+(Q_\phi)\rangle$.

The key result from [Vardi 1995] ([Kupferman 2018, Theorem 24]) is lifted as to be modulo $\mathcal{A}$.

Theorem 2 (Vardi derivatives modulo theories). $\forall \phi \in \mathbf{LTL}^+\langle\mathcal{A}\rangle : \mathscr{L}(M_\phi) = \mathscr{L}(\phi)$

Thus, the following invariant is preserved by *all* states ψ of M_ϕ because $\mathscr{L}_{M_\phi}(\psi) = \mathscr{L}(M_\psi)$.

Corollary 2 (LTL invariance). $\forall \phi \in \mathbf{LTL}^+\langle\mathcal{A}\rangle : \forall \psi \in Q_{M_\phi} : \mathscr{L}_{M_\phi}(\psi) = \mathscr{L}(\psi)$

Some states may become unreachable from ϕ through rewrites of ϱ_ϕ, that the definition of M_ϕ does not directly reflect, e.g., α and α^c are not relevant as states in Example 6.5.

Example 6.5. Let $\phi = \mathbf{G}(\mathbf{F}\alpha \wedge \mathbf{F}\alpha^c)$, where $\alpha \in \mathcal{A}$ is such that $\alpha \not\equiv \bot$ and $\alpha \not\equiv \top$. Then the transition function $\varrho = \varrho_\phi$ has the following reachable states and terms:

$$\varrho(\mathbf{F}\alpha) = \mathbf{(}\alpha\,\mathbf{?}\,\top\,\mathbf{:}\,\mathbf{F}\alpha\mathbf{)}, \quad \varrho(\mathbf{F}\alpha^c) = \mathbf{(}\alpha^c\,\mathbf{?}\,\top\,\mathbf{:}\,\mathbf{F}\alpha^c\mathbf{)}, \quad \varrho(\phi) = \mathbf{(}\alpha\,\mathbf{?}\,\mathbf{F}\alpha^c \wedge \phi\,\mathbf{:}\,\mathbf{F}\alpha \wedge \phi\mathbf{)}, \quad \varrho(\top) = \top$$

where $\partial(\mathbf{F}\alpha) \cong (\partial(\alpha) \vee \mathbf{F}\alpha) = \mathbf{(}\alpha\,\mathbf{?}\,\top \vee \mathbf{F}\alpha\,\mathbf{:}\,\bot \vee \mathbf{F}\alpha\mathbf{)} \cong \mathbf{(}\alpha\,\mathbf{?}\,\top\,\mathbf{:}\,\mathbf{F}\alpha\mathbf{)}$ (similarly for $\partial(\mathbf{F}\alpha^c)$), and

$$\begin{aligned}\partial(\phi) &\cong (\partial(\mathbf{F}\alpha \wedge \mathbf{F}\alpha^c) \wedge \phi) = (\partial(\mathbf{F}\alpha) \wedge \partial(\mathbf{F}\alpha^c) \wedge \phi)\\ &\cong (\mathbf{(}\alpha\,\mathbf{?}\,\top\,\mathbf{:}\,\mathbf{F}\alpha\mathbf{)} \wedge \mathbf{(}\alpha^c\,\mathbf{?}\,\top\,\mathbf{:}\,\mathbf{F}\alpha^c\mathbf{)} \wedge \phi)\\ &\cong \mathbf{(}\alpha\,\mathbf{?}\,\mathbf{(}\alpha^c\,\mathbf{?}\,\phi\,\mathbf{:}\,\mathbf{F}\alpha^c \wedge \phi\mathbf{)}\,\mathbf{:}\,\mathbf{(}\alpha^c\,\mathbf{?}\,\mathbf{F}\alpha \wedge \phi\,\mathbf{:}\,\mathbf{F}\alpha \wedge \mathbf{F}\alpha^c \wedge \phi\mathbf{)}\mathbf{)}\\ &\cong \mathbf{(}\alpha\,\mathbf{?}\,\mathbf{F}\alpha^c \wedge \phi\,\mathbf{:}\,\mathbf{F}\alpha \wedge \phi\mathbf{)}\end{aligned}$$

The resulting M_ϕ is shown in Figure 1 with $q_0 = \phi$, $q_1 = \mathbf{F}\alpha$ and $q_2 = \mathbf{F}\alpha^c$. ⊠

The following example illustrates the impact of cleaning modulo different element theories $\mathcal{A}$ and how this is reflected in the computation of M_ϕ.

Example 6.6. Consider $\mathcal{A}$ with an SMT solver, let x be of type *integer*, and let $\phi = (x{<}1)\ \mathbf{R}\ (0{<}x)$, stating that the condition that $x > 0$ is released when $x < 1$ becomes true. Then

$$\begin{aligned}\partial(\phi) &\cong \partial(0{<}x) \wedge (\partial(x{<}1) \vee \phi) = \mathbf{(}0{<}x\,\mathbf{?}\,\top\,\mathbf{:}\,\bot\mathbf{)} \wedge (\mathbf{(}x{<}1\,\mathbf{?}\,\top\,\mathbf{:}\,\bot\mathbf{)} \vee \phi)\\ &\cong \mathbf{(}0{<}x\,\mathbf{?}\,\top\,\mathbf{:}\,\bot\mathbf{)} \wedge \mathbf{(}x{<}1\,\mathbf{?}\,\top\,\mathbf{:}\,\phi\mathbf{)} \cong \mathbf{(}0{<}x\,\mathbf{?}\,\mathbf{(}x{<}1\,\mathbf{?}\,\top\,\mathbf{:}\,\phi\mathbf{)}\,\mathbf{:}\,\bot\mathbf{)} \cong \mathbf{(}0{<}x\,\mathbf{?}\,\phi\,\mathbf{:}\,\bot\mathbf{)}\end{aligned}$$

where $\mathbf{UNSAT}((0{<}x) \sqcap (x{<}1))$ is used for cleaning. I.e., in this case $(x{<}1)\ \mathbf{R}\ (0{<}x) \equiv \mathbf{G}(0{<}x)$. ⊠

6.3 Alternation Elimination for LTL Modulo Theories

Combined together, Theorem 1 and Corollary 2 show the decidability of $\mathbf{LTL}\langle\mathcal{A}\rangle$ for decidable $\mathcal{A}$. They allow us to directly use the Æ algorithm on the transition terms resulting from the symbolic derivatives of $\mathbf{LTL}\langle\mathcal{A}\rangle$. This will produce a nondeterministic Büchi automata from an LTL formula in a lazy manner, while safely applying many LTL-based rewrites.

Theorem 3 (LTL invariance of Æ). $\forall \phi \in \mathbf{LTL}^+\langle\mathcal{A}\rangle : \forall q \in Q_{Æ(M_\phi)} : \mathscr{L}_{Æ(M_\phi)}(q) = \mathscr{L}(\wedge q)$

Proof. Let $M = M_\phi$ and $N = Æ(M)$. Fix $q = \langle U, V\rangle \in Q_N$. We show that $\mathscr{L}_N(q) = \mathscr{L}(\wedge q)$. We have that $\mathscr{L}_N(q) = \mathscr{L}_M(\wedge q) = \bigcap_{\psi \in U \cup V} \mathscr{L}_M(\psi) = \bigcap_{\psi \in U \cup V} \mathscr{L}(\psi) = \mathscr{L}(\wedge q)$ where the first equality holds by Theorem 1 and the third equality holds by Corollary 2 because $U \cup V \subseteq Q_M$. □

States $\langle U, V\rangle$ in $Q_{Æ(M_\phi)}$ can be reduced by Lemma 3. E.g., for any φ, if $\varphi, \mathbf{G}(\varphi \wedge \_) \in V$ then φ can be eliminated from V because $\partial(\varphi \wedge \mathbf{G}(\varphi \wedge \_)) \cong \partial(\mathbf{G}(\varphi \wedge \_))$, which means that the derivative (transition term) does not even need to be computed in this case, in order to know that Lemma 3 can be applied. In fact, this exact situation appeared twice implicitly in Example 5.1, where $M = M_{\mathbf{G}(\mathbf{F}\alpha \wedge \mathbf{F}\alpha^c)}$. Many similar rules can be used, where the one illustrated here is an instance of the subsumption rule in [Somenzi and Bloem 2000, ≤]. Such rules would clearly be out of reach if the LTL invariance property had been lost in translation. A full analysis of which rules in [Somenzi and Bloem 2000] preserve LTL invariance of Æ is beyond the scope of this work.

7 LTL Extended with Regexes Modulo Theories

We now show the power of symbol derivatives by using them to combine the following two languages into one:

- ✓ $\mathbf{ERE}\langle\mathcal{A}\rangle$, with derivatives summarized in Section 7.1;
- ✓ $\mathbf{LTL}\langle\mathcal{A}\rangle$, as given in the previous section.

We first work with *finite* words in Σ^*, and later lift the semantics to *infinite* words in Σ^ω when we extend $\mathbf{LTL}\langle\mathcal{A}\rangle$ with $\mathbf{ERE}\langle\mathcal{A}\rangle$ in the combined language $\mathbf{RLTL}\langle\mathcal{A}\rangle$ in Section 7.3.

Our focus here is on the *classical subset* of the extended regular expression operators supported in SPOT [Duret-Lutz 2024] and PSL [Eisner and Fisman 2006]. The language $\mathbf{RLTL}\langle\mathcal{A}\rangle$ also has regex *complement* and *ω-closure*, which are not in SPOT/PSL but can be supported naturally with transition terms. In Section 7.7 we lift ω-regular languages to be modulo $\mathcal{A}$ and show that $\mathbf{RLTL}^+\langle\mathcal{A}\rangle$ captures them precisely.

7.1 Derivatives of ERE as Transition Regexes

Recall the definition of $\mathbf{ERE}\langle\mathcal{A}\rangle$ from Section 2.5 and let $R \in \mathbf{ERE}\langle\mathcal{A}\rangle$. The property $\mathit{nullable}(R)$ means that $\epsilon \Vdash R$ and is maintained as flag of each regex. Recall that $v \Vdash R \Leftrightarrow v \in \mathcal{L}(R)$.

$$\mathit{nullable}(\alpha) \overset{\text{DEF}}{=} \mathbf{false} \quad \mathit{nullable}(\varepsilon) \overset{\text{DEF}}{=} \mathbf{true} \quad \mathit{nullable}(R*) \overset{\text{DEF}}{=} \mathbf{true} \quad \mathit{nullable}(\sim R) \overset{\text{DEF}}{=} \neg\, \mathit{nullable}(R)$$
$$\mathit{nullable}(R_1 \cup R_2) \overset{\text{DEF}}{=} \mathit{nullable}(R_1) \vee \mathit{nullable}(R_2) \quad \mathit{nullable}(R_1 \cap R_2) \overset{\text{DEF}}{=} \mathit{nullable}(R_1) \wedge \mathit{nullable}(R_2)$$

The *derivative of* $R \in \mathbf{ERE}$ is denoted by $\delta(R) \in \mathbf{TTerm}\langle\mathcal{A}, \mathbf{ERE}\rangle$ [Stanford et al. 2021]. Recall that (1) and (2) automatically lift all the regex operators to $\mathbf{TTerm}\langle\mathcal{A}, \mathbf{ERE}\rangle$, which means that the algebra of lift rules in [Stanford et al. 2021, Section 4.1] (dealing with incremental propagation of the operators in transition regexes) is irrelevant here.

$$\begin{array}{ll} \delta(\varepsilon) \overset{\text{DEF}}{=} \bot & \delta(R_1 \cup R_2) \overset{\text{DEF}}{=} \delta(R_1) \cup \delta(R_2) \\ \delta(\alpha) \overset{\text{DEF}}{=} (\alpha\,?\,\varepsilon : \bot) & \delta(R_1 \cap R_2) \overset{\text{DEF}}{=} \delta(R_1) \cap \delta(R_2) \\ \delta(\sim R) \overset{\text{DEF}}{=} \sim\delta(R) & \multirow{2}{*}{$\delta(R_1 \cdot R_2) \overset{\text{DEF}}{=} \begin{cases} \delta(R_1)\cdot R_2 \cup \delta(R_2), & \textbf{if } \mathit{nullable}(R_1); \\ \delta(R_1)\cdot R_2, & \textbf{otherwise}. \end{cases}$} \\ \delta(R*) \overset{\text{DEF}}{=} \delta(R)\cdot R* & \end{array} \tag{22}$$

In particular, the definition of $\delta(\alpha) = (\alpha\,?\,\varepsilon : \bot)$ differs crucially from $\partial(\alpha) = (\alpha\,?\,\top : \bot)$ in $\mathbf{LTL}\langle\mathcal{A}\rangle$. Moreover, $\neg(\alpha\,?\,\top : \bot) \cong (\alpha\,?\,\bot : \top)$ while $\sim(\alpha\,?\,\top : \bot) \cong (\alpha\,?\,\varepsilon \cup \top\top+ : \top*)$. In other words, while structurally similar, the derivatives (and their semantics) are fundamentally different.

The union operation $\cup$ over $\mathbf{ERE}$ is treated as an ACI operation in [Stanford et al. 2021], as the critical part of $\cong$ over $\mathbf{ERE}$, which implies that M_R, as defined next, is well-defined as a symbolic DFA modulo $\mathcal{A}$ ($\mathit{DFA}_{\mathcal{A}}$) by having a *finite* state space.

Definition 7.1 (M_R for $R \in \mathbf{ERE}\langle\mathcal{A}\rangle$). $M_R = (\mathcal{A}, Q, R, \varrho, F)$ where Q is the least set of states containing R as the initial state and $\varrho : Q \to \mathbf{TTerm}\langle\mathcal{A}, Q\rangle$ is the transition function such that $\varrho(q) \cong \delta(q)$ and $\mathit{lvs}(\varrho(q)) \subseteq Q$. $F = \{q \in Q \mid \mathit{nullable}(q)\}$.

Let $M = M_R$. For all $a \in \Sigma, u \in \Sigma^*$ and $q \in Q$ let $\hat{\delta}(\epsilon, q) \overset{\text{DEF}}{=} q$ and $\hat{\delta}(a{::}u, q) \overset{\text{DEF}}{=} \hat{\delta}(u, \delta(q)[a])$. For all $q \in Q$ let $\mathcal{L}_M(q) \overset{\text{DEF}}{=} \{u \in \Sigma^* \mid \hat{\delta}(u, q) \in F\}$. Lemma 4 is implied by [Stanford et al. 2021, Theorem 4.3] and [Brzozowski 1964, Theorem 3.1].

Lemma 4 ([Stanford et al. 2021]). $\forall q \in Q : \mathcal{L}_M(q) = \mathcal{L}(q) \wedge \mathcal{L}(\varrho(q)[a]) = \mathbf{D}_a(\mathcal{L}(q))$.

A state q is *alive* if $\mathcal{L}_M(q) \neq \emptyset$ else *dead*. Observe that a state $\hat{\delta}(u, R)$ is alive $\Leftrightarrow \exists x : ux \Vdash R$.

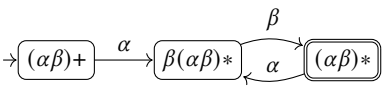
7.2 One Step Lookahead

For $R \in \mathbf{ERE}$ let $\mathit{OneStep}(R)$ be a predicate in $\mathcal{A}$ that denotes $\mathcal{L}(R) \cap \Sigma$. The predicate $\mathit{OneStep}(R)$ plays a key role in the definition of derivatives for $\mathbf{RLTL}\langle\mathcal{A}\rangle$ below, as a *one-step lookahead* such that, for all $a \in \Sigma$, $a \vDash \mathit{OneStep}(R) \Leftrightarrow a \Vdash R$.

$$\begin{aligned}\mathit{OneStep}(R) &\stackrel{\text{DEF}}{=} \mathit{OneStep}'(\delta(R))\\ \mathit{OneStep}'(R) &\stackrel{\text{DEF}}{=} \textbf{if } \mathit{nullable}(R) \textbf{ then } \top \textbf{ else } \bot\\ \mathit{OneStep}'(\textbf{(}\alpha \textbf{ ? } f \textbf{ : } g\textbf{)}) &\stackrel{\text{DEF}}{=} (\alpha \sqcap \mathit{OneStep}'(f)) \sqcup (\alpha^c \sqcap \mathit{OneStep}'(g))\end{aligned}$$

In particular, $\mathit{OneStep}(R) \equiv \bot \Leftrightarrow \mathcal{L}(R) \cap \Sigma = \emptyset$. The predicate $\mathit{OneStep}(R)$ is often used when $\delta(R)$ is used. Therefore, the predicate $\mathit{OneStep}'(f)$ is ideally cached (and simplifed to $\bot$ when unsat) with each transition regex f when f is constructed.

Example 7.2. We construct the transition terms of $M_{(\alpha\beta)+}$, where $\alpha, \beta \in \mathcal{A}$, for all $r \in Q_{M_{(\alpha\beta)+}}$.

r	$\delta(r)$	$\mathit{OneStep}(r)$	$\mathit{nullable}(r)$	$\mathit{DFA}_{\mathcal{A}}\ M_{(\alpha\beta)+}$
$(\alpha\beta)+$	$\textbf{(}\alpha \textbf{ ? } \beta(\alpha\beta)*\textbf{)}$	$\bot$	**false**	
$\beta(\alpha\beta)*$	$\textbf{(}\beta \textbf{ ? } (\alpha\beta)*\textbf{)}$	β	**false**	
$(\alpha\beta)*$	$\textbf{(}\alpha \textbf{ ? } \beta(\alpha\beta)*\textbf{)}$	$\bot$	**true**	

$\mathit{OneStep}(r)$ is shown for each state above. (We typically omit the state $\bot$ to avoid clutter.) ⊠

7.3 RLTL

$\mathbf{RLTL}\langle\mathcal{A}\rangle$ extends $\mathbf{LTL}\langle\mathcal{A}\rangle$ with $\mathbf{ERE}\langle\mathcal{A}\rangle$. Let $\phi \in \mathbf{RLTL}\langle\mathcal{A}\rangle$, $R \in \mathbf{ERE}\langle\mathcal{A}\rangle$, and $\alpha \in \mathcal{A}$.

$$\phi \quad ::= \quad \alpha \mid \neg\phi \mid \phi \vee \phi' \mid \phi \wedge \phi' \mid \mathbf{X}\phi \mid \phi\ \mathbf{U}\ \phi' \mid \phi\ \mathbf{R}\ \phi' \mid R \diamond\!\!\rightarrow \phi \mid R \,\square\!\!\rightarrow\, \phi \mid \textbf{\{}R\textbf{\}} \mid \textbf{!\{}R\textbf{\}} \mid R^{\omega}$$

The operator $\diamond\!\!\rightarrow$ is *existential suffix implication*, $\square\!\!\rightarrow$ is (*universal*) *suffix implication*, $\textbf{\{}R\textbf{\}}$ is *weak closure*, $\textbf{!\{}R\textbf{\}}$ is *negated weak closure*, and R^{ω} is *ω-closure*. The semantics of $\phi \in \mathbf{RLTL}$ is a conservative extension of **LTL** and consistent with the semantics in SPOT [Duret-Lutz 2024], except that if R is *nullable* then in **RLTL** the formula $\textbf{\{}R\textbf{\}}$ (resp. $\textbf{!\{}R\textbf{\}}$) is equivalent to $\top$ (resp. $\bot$) and R^{ω},~R are not supported in PSL/SPOT. The SPOT formula $\textbf{\{}R\textbf{\}}$ ($\textbf{!\{}R\textbf{\}}$) maps to $\textbf{\{}R \Cap \top\textbf{+\}}$ ($\textbf{!\{}R \Cap \top\textbf{+\}}$) in **RLTL**. Definitions (9–14) carry over to **RLTL** and we add the new definitions:

$$w \models R \diamond\!\!\rightarrow \phi \stackrel{\text{DEF}}{=} \exists i : \mathit{take}(i+1, w) \Vdash R \wedge \mathit{drop}(i, w) \models \phi \tag{23}$$

$$w \models R \,\square\!\!\rightarrow\, \phi \stackrel{\text{DEF}}{=} \forall i : \mathit{take}(i+1, w) \Vdash R \Rightarrow \mathit{drop}(i, w) \models \phi \tag{24}$$

$$w \models \textbf{\{}R\textbf{\}} \stackrel{\text{DEF}}{=} \exists i : \mathit{take}(i, w) \Vdash R \ \vee\ \forall i > 0 : \exists x : \mathit{take}(i, w) \cdot x \Vdash R \tag{25}$$

$$w \models \textbf{!\{}R\textbf{\}} \stackrel{\text{DEF}}{=} w \not\models \textbf{\{}R\textbf{\}} \tag{26}$$

$$w \models R^{\omega} \stackrel{\text{DEF}}{=} w \in \mathcal{L}(R)^{\omega} \tag{27}$$

We use the relations $v \prec_{\epsilon} w \stackrel{\text{DEF}}{=} \exists i : v = \mathit{take}(i, w)$ and $v \prec w \stackrel{\text{DEF}}{=} \exists i > 0 : v = \mathit{take}(i, w)$ below.

Example 7.3. Let $R = \alpha*\beta$ where $[\![\alpha]\!] = \{a\}$ and $[\![\beta]\!] = \{b\}$. Then $\nexists u \prec_{\epsilon} a^{\omega} : u \in \mathcal{L}(R)$, however $\forall u \prec a^{\omega} : \exists x : ux \in \mathcal{L}(R)$ (namely for $x = b$) – i.e., for all $u \prec a^{\omega}$, $\hat{\delta}(u, R)$ is alive. So $a^{\omega} \models \textbf{\{}R\textbf{\}}$. ⊠

It follows from the definitions that $R \,\square\!\!\rightarrow\, \phi \equiv \neg(R \diamond\!\!\rightarrow \neg\phi)$ and trivially that $\textbf{!\{}R\textbf{\}} \equiv \neg\textbf{\{}R\textbf{\}}$, i.e., (24) and (26) are duals of (23) and (25).

*Definition 7.4 (**RLTL**$^{+}$).* The *positive fragment of* **RLTL** is obtained by using Definition 6.3, and for $R \in \mathbf{ERE}$ and $\varphi, \psi \in \mathbf{RLTL}$: $\neg(R \diamond\!\!\rightarrow \varphi) \equiv R \,\square\!\!\rightarrow\, \neg\varphi$, $\neg(R \,\square\!\!\rightarrow\, \varphi) \equiv R \diamond\!\!\rightarrow \neg\varphi$, $\neg\textbf{\{}R\textbf{\}} \equiv \textbf{!\{}R\textbf{\}}$, and $\neg\textbf{!\{}R\textbf{\}} \equiv \textbf{\{}R\textbf{\}}$. **RLTL**$^{+}$ ***disallows*** $\neg(R^{\omega})$.

Example 7.5. Recall $\sim R \boxdot\!\!\rightarrow \bot$ from Section 1.3 that belongs to $\mathbf{RLTL}^+$. So

$$w \models \sim R \boxdot\!\!\rightarrow \bot \Leftrightarrow \forall\, i : take(i+1, w) \not\models \sim R \vee drop(i, w) \models \bot \Leftrightarrow \forall\, i > 0 : take(i, w) \not\models \sim R$$
$$\Leftrightarrow \forall\, i > 0 : take(i, w) \models R$$

and thus $w \models \sim R \boxdot\!\!\rightarrow \bot \Leftrightarrow \forall\, v \prec w : v \models R$. E.g., if a regex r is *bounded* (only accepts words of *bounded length*, any regex that is both $*$-free and $\sim$-free is bounded) and $\lfloor r \rfloor$ is the *prefix closure regex* of r, then $\sim(r*\cdot\lfloor r \rfloor) \boxdot\!\!\rightarrow \bot \equiv r^\omega$. Since $\neg(r^\omega)$ is not supported in $\mathbf{RLTL}^+\langle\mathcal{A}\rangle$, $\sim(r*\cdot\lfloor r \rfloor) \diamond\!\!\rightarrow \top$ can be used instead of $\neg(r^\omega)$ if r is bounded. ⊠

For $M \in ABW_{\mathcal{A}}$ and $\phi \in \mathbf{RLTL}\langle\mathcal{A}\rangle$ let $M \models \phi \stackrel{\text{DEF}}{=} \mathscr{L}(M) \subseteq \mathscr{L}(\phi)$.

7.4 Trace Analysis of Cloud Services

Our primary application of $\mathbf{RLTL}\langle\mathcal{A}\rangle$ is to analyze traces of *cloud services*. The elements in Σ are called *actions* in this context, that are pairs (req, res) of *JSON values* where *req* represents an `http` *request* and *res* represents an `http` *response*. SMT solvers like Z3 [de Moura and Bjørner 2008] can natively reason about JSON values as they can be represented by nested data types.

Note that a JSON value here is a structured representation of an `http` request or response that is itself just flat text. The basic fields of a JSON value are usually *strings*, that are sequences of characters of type *Unicode* in Z3. JSON fields are often associated with additional type constraints in form of regular expressions, e.g., that a *key* field must be a nonempty string of digits, so it must match the regex `\d+`. Thus, *regexes modulo Unicode* are part of $\mathcal{A}$, while *regexes modulo* $\mathcal{A}$ are used to specify action patterns in $\mathbf{RLTL}\langle\mathcal{A}\rangle$, as illustrated below.

Since cloud services run continuosly their trace semantics is naturally described in terms of *action streams*. (In reality, there can also be several requests in-flight simultaneously, but here we let each request *req* be paired with an immediate response *res*.)

A model of the cloud service is given or is being learned as an $NBW_{\mathcal{A}}$ M. If we have some **RLTL** property ψ that is expected to hold, i.e., $M \models \psi$ is expected, but $\mathscr{L}(M \times Æ(M_{\neg\psi})) \neq \emptyset$ we can extract a counterexample from $M \times Æ(M_{\neg\psi})$. If the counterexample turns out to be spurious, by evaluating it against the actual service, it can be used to refine M (during learning).

For example, in some services requests and responses must have equal IDs in the same action, in other words $M \models \mathbf{G}(ID(req) = ID(res))$ must hold. Let FAIL denote failed actions and let $\text{SUCC} = \text{FAIL}^c$. A failure typically means that *res* contains an `http` status code in the range ≥ 400, but it may also be a request specific error message, e.g., that a null reference exception was thrown, or that the *etag* values in *req* and *res* do not match when expected to match for certain operations.

We consider a subset of the Azure *App Configuration* service [Microsoft 2023] for basic key operations. For a *fixed* key, we use the following predicates on *req*: D = the operation is *DeleteKey*; L = the operation is *LockKey*; U = the operation is *UnlockKey*. Then the following formula, say ψ,

$$\top*\cdot(\text{L} \sqcap \text{SUCC})\cdot(\text{U}^c \sqcup \text{FAIL})*\cdot\text{D} \boxdot\!\!\rightarrow \text{FAIL}$$

states that a locked key cannot be deleted. It describes the following action pattern: if at some point the key is successfully locked (L ⊓ SUCC) and there are no subsequent successful unlock actions ($\text{U}^c \sqcup \text{FAIL}$)$*$ before a delete request D then D must fail. Observe that D and FAIL in ψ apply to the same action w_i in any action stream w – the last action in $take(i+1, w)$ is also the first action in $drop(i, w)$ in the semantics of $\boxdot\!\!\rightarrow$ (24). $M_{\neg\psi}$ is illustrated in Figure 2.

Patterns such as the one above are often most naturally expressed by using regexes and suffix implications, that also illustrates a core advatage over the $\mathbf{LTL}\langle\mathcal{A}\rangle$ fragment. It is often unclear and difficult to decide if a property can or cannot be expresses in $\mathbf{LTL}\langle\mathcal{A}\rangle$. Even when a property can be expressed in $\mathbf{LTL}\langle\mathcal{A}\rangle$ it can be difficult to understand the resulting formula. As it happens,

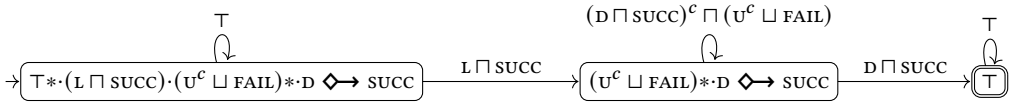


Fig. 2. $M_{\neg\psi}$ where $\psi = \top{*}\cdot(\mathrm{L} \sqcap \mathrm{SUCC})\cdot(\mathrm{U}^c \sqcup \mathrm{FAIL}){*}\cdot\mathrm{D}\ \square\!\!\rightarrow\ \mathrm{FAIL}$ as in Section 7.4.

the formula ψ above can be written as the following equivalent formula in $\mathbf{LTL}\langle\mathcal{A}\rangle$, say ϕ,

$$\mathbf{G}((\mathrm{L} \wedge \mathrm{SUCC}) \rightarrow \mathbf{X}((\mathrm{U} \wedge \mathrm{SUCC})\ \mathbf{R}\ (\mathrm{D} \rightarrow \mathrm{FAIL})))$$

stating that, whenever the key has been successfully locked then, consequently, successfull unlock releases the condition that delete fails. The $\mathbf{LTL}\langle\mathcal{A}\rangle$ symbolic derivative of $\neg\phi$ is as follows:

$$\partial(\neg\phi) \cong \mathbf{(}\mathrm{L} \sqcap \mathrm{SUCC}\ \mathbf{?}\ ((\mathrm{U} \rightarrow \mathrm{FAIL})\ \mathbf{U}\ (\mathrm{D} \wedge \mathrm{SUCC})) \vee \neg\phi\ \mathbf{:}\ \neg\phi\mathbf{)}$$
$$\partial((\mathrm{U} \rightarrow \mathrm{FAIL})\ \mathbf{U}\ (\mathrm{D} \wedge \mathrm{SUCC})) \cong \mathbf{(}\mathrm{D} \sqcap \mathrm{SUCC}\ \mathbf{?}\ \top\ \mathbf{:}\ \mathbf{(}\mathrm{U}^c \sqcup \mathrm{FAIL}\ \mathbf{?}\ (\mathrm{U} \rightarrow \mathrm{FAIL})\ \mathbf{U}\ (\mathrm{D} \wedge \mathrm{SUCC})\mathbf{)}\mathbf{)}$$

The automaton $M_{\neg\phi}$ is illustrated in Figure 3 and is clearly equivalent to $M_{\neg\psi}$ in Figure 2.

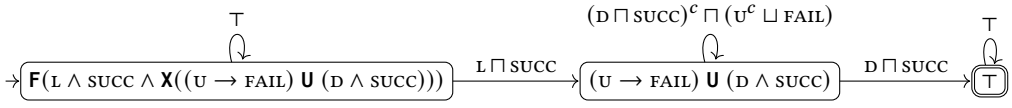


Fig. 3. $M_{\neg\phi}$ where $\phi = \mathbf{G}((\mathrm{L} \wedge \mathrm{SUCC}) \rightarrow \mathbf{X}((\mathrm{U} \wedge \mathrm{SUCC})\ \mathbf{R}\ (\mathrm{D} \rightarrow \mathrm{FAIL})))$ as in Section 7.4.

Observe that the predicates here are not just propositions but predicates over the underlying JSON values. For example, a request cannot simultaneously be a delete operation and an unlock operation, i.e., $\mathbf{UNSAT}(\mathrm{D} \wedge \mathrm{U})$. Therefore, any bitblasting to propositions, say p_{D} and p_{U} in SPOT, would require the theory constraint $\neg(p_{\mathrm{D}} \wedge p_{\mathrm{U}})$, among all the other required theory constraints.

7.5 Derivatives of RLTL

Derivatives for **RLTL** are defined as terms in $\mathbf{TTerm}\langle\mathcal{A}, \mathbf{RLTL}\rangle$ reusing (15–21) and the rules:

$$\partial(R \diamondarrow \phi) \overset{\text{DEF}}{=} \mathbf{(}\mathit{OneStep}(R)\ \mathbf{?}\ \partial(\phi)\ \mathbf{:}\ \bot\mathbf{)} \vee (\delta(R) \diamondarrow \phi) \tag{28}$$
$$\partial(R \boxarrow \phi) \overset{\text{DEF}}{=} \mathbf{(}\mathit{OneStep}(R)\ \mathbf{?}\ \partial(\phi)\ \mathbf{:}\ \top\mathbf{)} \wedge (\delta(R) \boxarrow \phi) \tag{29}$$
$$\partial(\mathbf{\{}R\mathbf{\}}) \overset{\text{DEF}}{=} \mathbf{if}\ \mathit{nullable}(R)\ \mathbf{then}\ \top\ \mathbf{else}\ \mathbf{\{}\delta(R)\mathbf{\}} \tag{30}$$
$$\partial(\mathbf{!\{}R\mathbf{\}}) \overset{\text{DEF}}{=} \mathbf{if}\ \mathit{nullable}(R)\ \mathbf{then}\ \bot\ \mathbf{else}\ \mathbf{!\{}\delta(R)\mathbf{\}} \tag{31}$$
$$\partial(R^{\omega}) \overset{\text{DEF}}{=} \partial(R \diamondarrow \mathbf{X}R^{\omega}) \tag{32}$$

where $r \diamondarrow \phi$, $r \boxarrow \phi$, $\mathbf{\{}r\mathbf{\}}$, and $\mathbf{!\{}r\mathbf{\}}$ are lifted to $\mathbf{TTerm}\langle\mathcal{A}, \mathbf{RLTL}\rangle$ by (2). Rules (28–31) seamlessly combine the derivatives of **LTL** (15–21) with derivatives of **ERE** (22). Correctness is proved in Theorem 4. Several derived laws can be inlined as immediate rewrites into the definition of ∂, such as $(\bot \diamondarrow \phi) \cong (\varepsilon \diamondarrow \phi) \cong \mathbf{\{}\bot\mathbf{\}} \cong \bot$, and if R is nullable then $\mathbf{\{}R\mathbf{\}} \cong \top$. Analogous rewrites apply to their duals $\boxarrow$ and $\mathbf{!\{}R\mathbf{\}}$. Note that if $\mathit{OneStep}(R) \equiv \bot$ then $\mathbf{(}\mathit{OneStep}(R)\ \mathbf{?}\ \partial(\phi)\ \mathbf{:}\ \bot\mathbf{)} \cong \bot$ and $\mathbf{(}\mathit{OneStep}(R)\ \mathbf{?}\ \partial(\phi)\ \mathbf{:}\ \top\mathbf{)} \cong \top$ whereby the definitions (28) and (29) simplify accordingly.

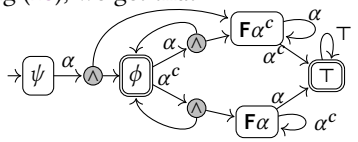
Let $\mathit{Acc}(\mathbf{RLTL}^+)$ denote the subset of all the following formulas of $\mathbf{RLTL}^+$: $\top$, all $\mathbf{R}$ formulas, all $\boxarrow$ formulas, all $\mathbf{\{}R\mathbf{\}}$ formulas where R is *alive*, all $\mathbf{!\{}R\mathbf{\}}$ formulas where R is *dead*, and all R^{ω} formulas.

Definition 7.6 (M_{ϕ} for $\phi \in \mathbf{RLTL}^+\langle\mathcal{A}\rangle$). $M_{\phi} = (\mathcal{A}, Q, \phi, \varrho, Q \cap \mathit{Acc}(\mathbf{RLTL}^+))$ where $Q \subseteq \mathbf{RLTL}^+$ is the least set such that $\phi \in Q$ and if $q \in Q$ then $\varrho(q) \cong \partial(q)$ and if p is a non-Boolean subformula of a leaf of $\varrho(q)$ then $p \in Q$.

M_ϕ is well-defined (Q above is finite) because all the M_R are finite by Lemma 4. In contrast to **LTL** derivatives, where all the original subformulas suffice, for $\phi \in \mathbf{RLTL}$, M_ϕ also gets new formulas in suffix implications and closures, e.g., in every leaf $r \diamond\!\!\rightarrow \psi$ of $\delta(R) \diamond\!\!\rightarrow \psi$, r is some leaf of $\delta(R)$ and $r \diamond\!\!\rightarrow \psi$ is typically not a subformula of ϕ. (See Figure 2.)

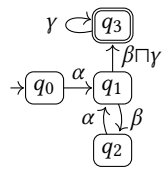
Example 7.7. Consider the **RLTL** formula $\psi = \alpha \diamond\!\!\rightarrow \phi$ with $\phi = \mathbf{G}(\mathbf{F}\alpha \wedge \mathbf{F}\alpha^c)$ and recall Example 6.5. Here $\delta(\alpha) = (\alpha\,?\,\varepsilon)$. Since $\mathit{OneStep}(\alpha) = \alpha$, by using (28), we get that

$$\begin{aligned}\partial(\psi) &= (\alpha\,?\,\partial(\phi)) \vee (\delta(\alpha) \diamond\!\!\rightarrow \phi) \cong (\alpha\,?\,\partial(\phi)) \vee (\alpha\,?\,(\varepsilon \diamond\!\!\rightarrow \phi))\\ &\cong (\alpha\,?\,\partial(\phi)) \vee (\alpha\,?\,\bot) \cong (\alpha\,?\,\partial(\phi))\\ &\cong (\alpha\,?\,(\alpha\,?\,\mathbf{F}\alpha^c \wedge \phi : \mathbf{F}\alpha \wedge \phi)) \cong (\alpha\,?\,\mathbf{F}\alpha^c \wedge \phi)\end{aligned}$$



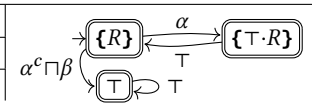
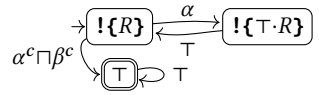
The complete $ABW_{\mathcal{A}}$ is depicted on the right. Æ can be used to obtain an $NBW_{\mathcal{A}}$ in this case. ⊠

Example 7.8. Let $\alpha, \beta, \gamma \in \mathcal{A}$ and consider $\phi = (\alpha\beta)+ \diamond\!\!\rightarrow \mathbf{G}\gamma$. The transition terms that arise from ϕ give rise here to the ABW M_ϕ as shown below, where we reused the regex derivatives from the earlier Example 7.2. Observe that M_ϕ also happens to be nondeterministic because conjunctions do not arise. Simplifications are directly inlined for $\bot$ as the zero of $\wedge$ and the unit of $\vee$.

q	$\partial(q)$	$NBW_{\mathcal{A}}$ M_ϕ
$q_0 = (\alpha\beta)+ \diamond\!\!\rightarrow \mathbf{G}\gamma$	$(\bot\,?\,\partial(\mathbf{G}\gamma)) \vee ((\alpha\,?\,\beta(\alpha\beta)*) \diamond\!\!\rightarrow \mathbf{G}\gamma)$ $\cong (\alpha\,?\,\beta(\alpha\beta)* \diamond\!\!\rightarrow \mathbf{G}\gamma)$	
$q_1 = \beta(\alpha\beta)* \diamond\!\!\rightarrow \mathbf{G}\gamma$	$(\beta\,?\,\partial(\mathbf{G}\gamma)) \vee (\beta\,?\,(\alpha\beta)* \diamond\!\!\rightarrow \mathbf{G}\gamma)$ $\cong (\beta\sqcap\gamma\,?\,\mathbf{G}\gamma) \vee (\beta\,?\,(\alpha\beta)* \diamond\!\!\rightarrow \mathbf{G}\gamma)$	
$q_2 = (\alpha\beta)* \diamond\!\!\rightarrow \mathbf{G}\gamma$	$(\alpha\,?\,\beta(\alpha\beta)* \diamond\!\!\rightarrow \mathbf{G}\gamma)$	
$q_3 = \mathbf{G}\gamma$	$(\gamma\,?\,\mathbf{G}\gamma)$	

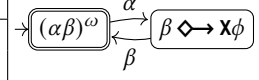
If $\beta \sqcap \gamma \not\equiv \bot$ then M_ϕ is actually nondeterministic because the outgoing transition guards from q_1 overlap, i.e., the disjunction $q_2 \vee q_3$ is the leaf $\partial(q_1)[a]$ for any $a \vDash \beta \sqcap \gamma$. ⊠

Example 7.9. Let α and β be two predicates in $\mathcal{A}$ such that $[\![\alpha]\!] = \{a\}$ and $[\![\beta]\!] = \{b\}$ where $a \neq b$. Let $R = (\alpha{\cdot}\top)*{\cdot}\beta$. The following table shows the transitions that arise in the NBW $M_{\{R\}}$, where $\mathbf{UNSAT}(\alpha\sqcap\beta)$ is used to clean $\partial(\{R\})$. Observe that $\alpha^c \sqcap \beta \equiv \beta$. The table also shows $M_{!\{R\}}$ where $!\{(\beta\,?\,\varepsilon)\} = (\beta\,?\,!\{\varepsilon\} : !\{\bot\}) \cong (\beta\,?\,\bot : \top) \cong (\beta^c\,?\,\top)$.

q	$\partial(q)$	NBW $M_{\{R\}}$	NBW $M_{!\{R\}}$
$\{R\}$	$(\alpha\,?\,\{\top{\cdot}R\} : (\beta\,?\,\{\varepsilon\}))$		
$\{\top{\cdot}R\}$	$(\top\,?\,\{R\}) \cong \{R\}$		
$\top (\cong \{\varepsilon\})$	$\top$		

Then $a^\omega \models \{R\}$ because $\forall u \prec a^\omega : \exists x : u{\cdot}x \Vdash R$. Note also that $\mathscr{L}(\mathbf{G}(\beta^c) \wedge \{R\}) = \mathscr{L}((\alpha\beta^c)^\omega)$ where α holds in all even positions – a classical example not expressible in LTL [Wolper 1983]. ⊠

Example 7.10. With $\alpha, \beta \in \mathcal{A}$ let $\phi = (\alpha\beta)^\omega$. We get the following NBW:

q	$\partial(q) = \partial(r \diamond\!\!\rightarrow \mathbf{X}\phi)$	$\mathit{OneStep}(r)$	NBW M_ϕ
ϕ	$\partial(\alpha\beta \diamond\!\!\rightarrow \mathbf{X}\phi) \cong (\delta(\alpha\beta) \diamond\!\!\rightarrow \mathbf{X}\phi)$ $\cong ((\alpha\,?\,\beta) \diamond\!\!\rightarrow \mathbf{X}\phi) = (\alpha\,?\,\beta \diamond\!\!\rightarrow \mathbf{X}\phi : \bot \diamond\!\!\rightarrow \mathbf{X}\phi)$ $\cong (\alpha\,?\,\beta \diamond\!\!\rightarrow \mathbf{X}\phi)$	$\bot$	
$\beta \diamond\!\!\rightarrow \mathbf{X}\phi$	$(\beta\,?\,\partial(\mathbf{X}\phi)) \vee (\delta(\beta) \diamond\!\!\rightarrow \mathbf{X}\phi)$ $= (\beta\,?\,\phi) \vee (\beta\,?\,\varepsilon \diamond\!\!\rightarrow \mathbf{X}\phi : \bot \diamond\!\!\rightarrow \mathbf{X}\phi)$ $\cong (\beta\,?\,\phi) \vee (\beta\,?\,\bot : \bot) \cong (\beta\,?\,\phi) \vee \bot \cong (\beta\,?\,\phi)$	β	

where many rewrite rules for $\cong$ are being used. ⊠

Recall that $w_{..i} = take(i+1, w)$ and $w_{i..} = drop(i, w)$. A distributivity law that can be very practical is that $(R_1 \Cup R_2) \diamond\!\!\rightarrow \phi \equiv (R_1 \diamond\!\!\rightarrow \phi) \vee (R_2 \diamond\!\!\rightarrow \phi)$, because for all $w \in \Sigma^\omega$ it holds that

$$\begin{aligned} \exists\, i : w_{..i} \Vdash R_1 \Cup R_2 \wedge w_{i..} \models \phi \quad &\Leftrightarrow \quad \exists\, i : (w_{..i} \Vdash R_1 \wedge w_{i..} \models \phi) \vee (w_{..i} \Vdash R_2 \wedge w_{i..} \models \phi) \\ &\Leftrightarrow \quad (\exists\, i : (w_{..i} \Vdash R_1 \wedge w_{i..} \models \phi)) \vee (\exists\, i : (w_{..i} \Vdash R_2 \wedge w_{i..} \models \phi)) \end{aligned}$$

Theorem 4 describes the semantics of **RLTL** in terms of languages and derivatives. It shows that the definition of derivatives correctly captures the intended semantics and is proved formally in Lean in Section 7.6 along with all required background theory. Theorem 4 does not yet imply any formal relationship with the corresponding *ABW* semantics, which is discussed in Section 7.7.

Theorem 4 (Derivation). $\forall \phi \in \mathbf{RLTL}\langle \mathcal{A} \rangle, a \in \Sigma, w \in \Sigma^\omega : a{::}w \models \phi \Leftrightarrow w \models \partial(\phi)[a]$

The distributivity law of $\diamond\!\!\rightarrow$ over union can be lifted and applied to derivatives through Theorem 4. This is related to adapting the construction in [Antimirov 1996] for **RLTL** that can avoid exponential blowup of the state space during incremental application of derivatives. Intuitively this amounts to unfolding a regex into an NFA instead of a DFA while distributing $\diamond\!\!\rightarrow$ over union. A further derived law that follows is that

$$(R_1 \Cup R_2) \,\Box\!\!\rightarrow \phi \equiv \neg((R_1 \Cup R_2) \diamond\!\!\rightarrow \neg\phi) \equiv \neg(R_1 \diamond\!\!\rightarrow \neg\phi) \wedge \neg(R_2 \diamond\!\!\rightarrow \neg\phi) \equiv (R_1 \,\Box\!\!\rightarrow \phi) \wedge (R_2 \,\Box\!\!\rightarrow \phi)$$

Such laws can also be formalized and proved correct in Lean.

7.6 Formalization in Lean

In this section we describe some of the key features of the Lean formalization, which focuses on the correctness of the derivative-based algorithm for RLTL given in Theorem 4.

Throughout the formalization we indicate the type of predicates by $\mathcal{A}$ (`A` in Lean) and the type of alphabet as Σ. The two are connected by the notion of EBA $(\Sigma, \mathcal{A}, \vDash, \bot, \top, \sqcup, \sqcap, ^c)$ as formally presented in [Zhuchko et al. 2024] using type classes, where Σ is marked as an out-param and is thus automatically inferred from $\mathcal{A}$. The inductive type `TTerm` represents $\mathbf{TTerm}\langle \mathcal{A}, \mathcal{B} \rangle$ where $\mathcal{A}$ is the condition type and $\mathcal{B}$ (`B` in Lean) the leaf type.

```
inductive TTerm (A B : Type) : Type where
  | Leaf : B → TTerm A B
  | Node (condition : A) (_then : TTerm A B) (_else : TTerm A B) : TTerm A B
```

The evaluation of a transition term f for an element a of type Σ is defined as follows in Lean and the notation `f[a]` in Lean corresponds precisely to the math notation $f[a]$.

```
def evaluation (a : Σ) (f : TTerm A B) : B :=
  match f with
  | Leaf b      => b
  | Node p f g => if a ⊨ p then evaluation a f else evaluation a g
notation f "[" a "]" => evaluation a f
```

The type `Stream' Σ` represents Σ^ω as functions from $\mathbb{N}$ to Σ, and is provided by the Lean standard library along with `drop`, `take`, `head`, `tail`, and `get` functions where (`get w i`) returns w_i.

We lift unary and binary operations to `TTerm` as presented in Section 3. The two crucial lemmas used throughout the formalization show the correctness of the lifting operations:

```
theorem liftU (op : B → B') {f : TTerm A B} (x : Σ) :
  (lift_unary op f)[x] = op (f [x])
theorem liftB (op : B → B → B') (f g : TTerm A B) (x : Σ) :
  (lift_binary op f g)[x] = op (f [x]) (g [x])
```

For example, if $f, g \in \mathbf{TTerm}\langle \mathcal{A}, \mathcal{B}\rangle$ and o is a binary operation over $\mathcal{B}$ then the math expression $(f\ o\ g)$ corresponds in Lean to the expression `(lift_binary o f g)`. The `liftU` and `liftB` lemmas are distribution laws for `op` and `evaluation` that correspond to (3) in Section 3.

7.6.1 Regular Languages. Before we can introduce semantics for infinite words, we have to define the semantics for finite words. We base the general structure of our approach on the formalization presented in [Zhuchko et al. 2024], which defines match semantics for regular expressions with lookarounds. In our context, we simplify this by excluding the cases for lookarounds in the type `ERE`. For a word `v` and a regular expression `r`, the match semantics `v ⊨ r` corresponds to the classical semantics $v \in \mathcal{L}(r)$. The theorems below establish the equivalence between derivatives and the classical language-based semantics, proved by induction on the match length:

```
theorem equivalenceNull {r : ERE A} : [] ⊨ r ↔ nullable r
theorem equivalenceDer {r : ERE A} : a::v ⊨ r ↔ v ⊨ (δ r)[a]
```

The type of finite words Σ^* is in Lean represented by `List Σ` where ϵ is the empty list `[]` in Lean.

7.6.2 ω-Regular Languages. We now show how we formalize the notion of ω-regular language presented by a regular expression R. Since a language is a set of infinite words, we define a stream w to be in the ω-closure of R if it can be partitioned into subwords of non-zero lengths such that each subword is in the language of R. We formalize this idea by requiring the existence of a stream of natural numbers, called `deltas`, which indicates for any i the length of the i-th subword minus one. Each subword must then be accepted by R. A graphical description is given as follows:

```
deltas =   2,      4,     1,   ...
w =      a, b, c, d, e, f, g, h, i, l, m, ...
```

We start by defining a function `getWordStart` which, given $i \in \mathbb{N}$, returns the starting position of the i-th subword in the stream w by summing up the values of the stream of deltas up to i.

```
def getWordStart (deltas : Stream'ℕ) (i : ℕ) : ℕ :=
    match i with
    | 0 => 0
    | .succ j => (getWordStart (tail deltas) j) + (head deltas) + 1
```

Note that in order to enforce the invariant that each subword has non-zero length, and since the stream `deltas` represents lengths minus 1, each individual delta is incremented by 1 – the calculation for $i > 0$ is precisely $\sum_{j<i} 1+\Delta_j$ in Section 2.3 where Δ corresponds to `deltas`. This explains the `(head deltas) + 1` in the inductive case.

The central definition is `IsDeltasOmegaLanguage`, which expresses the idea that each subword in `w` indicated by a given `delta` is recognized by `r`.

```
def IsDeltasOmegaLanguage (w : Stream'Σ) (r : ERE A) (deltas : Stream'ℕ) : Prop :=
  ∀ (i : ℕ),
    let start := getWordStart deltas i
    let len := (get deltas i) + 1
    take len (drop start w) ⊨ r
```

that for $i > 0$ corresponds to $take(1+\Delta_i, drop(\sum_{j<i} 1+\Delta_j, w)) \in \mathcal{L}(r)$. Finally, `InOmegaLanguage` checks for the existence of a `deltas` that correctly partitions `w` into recognized subwords.

```
def InOmegaLanguage (w : Stream'Σ) (r : ERE A) : Prop :=
  ∃ (deltas : Stream'ℕ), IsDeltasOmegaLanguage w r deltas
infixr:40 " ∈* " => InOmegaLanguage
```

In summary, we have that $w \in* r \Leftrightarrow w \in \mathcal{L}(r)^\omega$. We use the following theorem.

```
theorem regexOmegaClosure {r : ERE A} : w ∈* r ↔ ∃ i > 0, take i w ⊨ r ∧ drop i w ∈* r
```

7.6.3 RLTL Semantics. In Lean, we represent the models relation $\models$ as a binary predicate `models` between streams and `RLTL` formulas. The relation represents $\models$ as defined by the rules (9–13) and (23–27), and uses the models relation $\vDash$ of `A` as well as the models relation $\Vdash$ of `ERE A`.

```
def models (w : Stream'Σ) : RLTL A → Prop
  | RLTL.Pred p => head w ⊨ p
  | ¬ₗ ϕ        => ¬ models w ϕ
  | ϕ ∧ₗ ψ      => models w ϕ ∧ models w ψ
  | ϕ ∨ₗ ψ      => models w ϕ ∨ models w ψ
  | ϕ →ₗ ψ      => models w ϕ → models w ψ
  | X ϕ         => models (tail w) ϕ
  | ϕ U ψ       => ∃ j, models (drop j w) ψ ∧ ∀ i < j, models (drop i w) ϕ
  | ϕ R ψ       => ∃ j, models (drop j w) ϕ ∧ ∀ k ≤ j, models (drop k w) ψ
                   ∨ ∀ i, models (drop i w) ψ
  | r ◇→ ϕ      => ∃ i, take (i + 1) w ⊩ r ∧ models (drop i w) ϕ
  | r □→ ϕ      => ∀ i, take (i + 1) w ⊩ r → models (drop i w) ϕ
  | { r }       => (∃ i, take i w ⊩ r) ∨ (∀ i > 0, ∃ x, (take i w) ++ x ⊩ r)
  | r^ω         => w ∈* r
notation:52 lhs:53 " |= " rhs:53 => models lhs rhs
```

Similarly to the case of match semantics for regular languages [Zhuchko et al. 2024], we define `models` as a function rather than an inductive predicate to avoid the negative occurrence of the relation for the $\neg_l\ \phi$ constructor. Note that in Lean, the negation operator `¬ p` is defined as `p → ⊥`. In order to keep the definitions constructive, we introduce implication $\to_l$ as a built-in operator in Lean. The negative weak closure operator is defined in Lean as just syntactic sugar for $\neg_l${`r`}.

The definition of derivatives of RLTL formulas closely follows the definitions (15–21) and definitions (28–32) in Section 7.3. To illustrate this, we present the definition of the derivative for the existential suffix implication and weak closure operators:

```
def derivative (r : RLTL A) : TTerm A (RLTL A) :=
  | RLTL.Pred p => Node p (Leaf (Pred ⊤)) (Leaf (Pred ⊥))
  ...
  | r ◇→ ϕ     => let lhs := Node (OneStep r) (derivative ϕ) (Leaf (Pred ⊥))
                  lift_binary (· ∨ₗ ·) lhs (lift_unary (fun x => x ◇→ ϕ) (δ r))
  | { r }      => if nullable r then Leaf (Pred ⊤) else (lift_unary ({ . }) (δ r))
prefix:max " ∂ " => RLTL.derivative
```

Another key definition is the `OneStep` predicate, which is essential for defining the derivative of an RLTL formula. This is defined in terms of a helper function `OneStep'` that computes the match semantics of a derivative of an ERE regex. `OneStep` is defined as `OneStep'` of the derivative of `r`.

```
def OneStep' (tr : TTerm A (ERE A)) : A :=
  match tr with
  | Leaf r     => if nullable r then ⊤ else ⊥
  | Node p f g => (p ⊓ OneStep' f) ⊔ (p^c ⊓ OneStep' g)
def OneStep (r : ERE A) := OneStep' (δ r)
```

The main correctness and completeness property of `OneStep`, as well as the invariants needed for `OneStep'`, can be identified as follows:

```
theorem denoteOneStep' {f : TTerm A (ERE A)} : [] ⊩ f[a] ↔ a ⊨ OneStep' f
theorem denoteOneStep  {r : ERE A} :           [a] ⊩ r    ↔ a ⊨ OneStep r
```

When handling the dual temporal operators **U** and **R**, the following expansion laws prove to be particularly useful:

```
theorem expansion_release {ϕ ψ : RLTL A} : w |= ϕ R ψ ↔ w |= ψ ∧ₗ (ϕ ∨ₗ X(ϕ R ψ))
theorem expansion_until   {ϕ ψ : RLTL A} : w |= ϕ U ψ ↔ w |= ψ ∨ₗ (ϕ ∧ₗ X(ϕ U ψ))
```

The main correctness theorem for derivatives, given in Theorem 4 states that taking the derivative of an RLTL formula and then evaluating it on a single element of the domain preserves the match semantics.

```
theorem derivation {ϕ : RLTL A} : a::w |= ϕ ↔ w |= (∂ ϕ)[a]
```

The proof follows by induction on ϕ, using the correctness and completeness of `OneStep` and of the standard match semantics of regular expressions on finite words. We highlight some key cases of the proof and use the math formulation when presenting these for conciseness.

The proof of the case of existential suffix implication can be outlined as follows, although in Lean the two directions of the equivalence are proved as separate implications and use also secondary induction over i. Recall that $w_{..i} = take(i+1, w)$ and $w_{i..} = drop(i, w)$.

$$\begin{aligned}
a{::}w \models r \diamond\!\!\rightarrow \psi &\Leftrightarrow \exists\, i : (a{::}w)_{..i} \Vdash r \wedge (a{::}w)_{i..} \models \psi \\
&\Leftrightarrow (a{::}w)_{..0} \Vdash r \wedge a{::}w \models \psi \ \vee\ \exists\, i{>}0 : (a{::}w)_{..i} \Vdash r \wedge (a{::}w)_{i..} \models \psi \\
&\overset{\dagger}{\Leftrightarrow} a \vDash \mathit{OneStep}(r) \wedge a{::}w \models \psi \ \vee\ \exists\, i : w_{..i} \Vdash \delta(r)[a] \wedge w_{i..} \models \psi \\
&\Leftrightarrow a \vDash \mathit{OneStep}(r) \wedge a{::}w \models \psi \ \vee\ w \models \delta(r)[a] \diamond\!\!\rightarrow \psi \\
&\overset{\text{IH}}{\Leftrightarrow} a \vDash \mathit{OneStep}(r) \wedge w \models \partial(\psi)[a] \ \vee\ w \models \delta(r)[a] \diamond\!\!\rightarrow \psi \\
&\Leftrightarrow w \models (\mathit{OneStep}(r)\,?\,\top)[a] \wedge w \models \partial(\psi)[a] \ \vee\ w \models \delta(r)[a] \diamond\!\!\rightarrow \psi \\
&\Leftrightarrow w \models (\mathit{OneStep}(r)\,?\,\partial(\psi))[a] \ \vee\ w \models \delta(r)[a] \diamond\!\!\rightarrow \psi \\
&\Leftrightarrow w \models (\mathit{OneStep}(r)\,?\,\partial(\psi))[a] \vee \delta(r)[a] \diamond\!\!\rightarrow \psi \\
&\Leftrightarrow w \models ((\mathit{OneStep}(r)\,?\,\partial(\psi)) \vee \delta(r) \diamond\!\!\rightarrow \psi)[a] \\
&\Leftrightarrow w \models \partial(r \diamond\!\!\rightarrow \psi)[a]
\end{aligned}$$

In step † the proof uses the theorem `equivalenceDer` as well as the theorem `denoteOneStep`. Step IH indicates use of the induction hypothesis over RLTL formulas. The remaining proof steps make, on multiple occasions, use of `liftU`, `liftB` for lifting operations over transition terms that is implicit in the math formulation. It also uses the core properties of streams.

The proof of the case of weak closure `{r}` is largely based on theorems `equivalenceDer` and `equivalenceNull` and does not use the induction hypothesis over RLTL formulas. Here we present it as further two separate subcases when r is nullable and when r is not nullable and use the more relaxed math notation to avoid `liftU` and `liftB` that are implicit.

If $\mathit{nullable}(r) = \textbf{true}$ then

$$a{::}w \models \{r\} \ \Leftrightarrow\ a{::}w \models \top \ \Leftrightarrow\ w \models \top[a] \ \Leftrightarrow\ w \models \partial(\{r\})[a].$$

If $\mathit{nullable}(r) = \textbf{false}$ then

$$\begin{aligned}
a{::}w \models \{r\} &\Leftrightarrow \exists\, i : \mathit{take}(i, a{::}w) \Vdash r \ \vee\ \forall\, i{>}0 : \exists\, x : \mathit{take}(i, a{::}w){\cdot}x \Vdash r \\
&\Leftrightarrow \exists\, i{>}0 : \mathit{take}(i, a{::}w) \Vdash r \ \vee\ \forall\, i{>}0 : \exists\, x : \mathit{take}(i, a{::}w){\cdot}x \Vdash r \\
&\Leftrightarrow \exists\, i : \mathit{take}(i, w) \Vdash \delta(r)[a] \ \vee\ \forall\, i : \exists\, y : \mathit{take}(i, w){\cdot}y \Vdash \delta(r)[a] \\
&\overset{\ddagger}{\Leftrightarrow} \exists\, i : \mathit{take}(i, w) \Vdash \delta(r)[a] \ \vee\ \forall\, i{>}\,0 : \exists\, x : \mathit{take}(i, w){\cdot}x \Vdash \delta(r)[a] \\
&\Leftrightarrow w \models \{\delta(r)[a]\} \Leftrightarrow w \models \{\delta(r)\}[a] \Leftrightarrow w \models \partial(\{r\})[a]
\end{aligned}$$

While $\overset{\ddagger}{\Rightarrow}$ holds trivially, $\overset{\ddagger}{\Leftarrow}$ uses that $\mathit{take}(1, w){\cdot}x \Vdash \delta(r)[a]$ implies that $\mathit{take}(0, w){\cdot}y \Vdash \delta(r)[a]$ holds with $y = \mathit{take}(1, w){\cdot}x$ as a witness.

The final case that we illustrate here is ω-closure `r^ω`. In this case it is more intuitive to start with the statement $w \models \partial(r^\omega)[a]$. The proof in Lean uses the theorem `equivalenceDer` as well as the theorem `denoteOneStep` (and also `liftU` and `liftB` that are implicit here). Here the induction hypothesis over RLTL formulas is not applicable. Recall also that $\partial(\mathbf{X}\varphi) = \varphi$ and that $v \models \mathbf{X}\varphi \Leftrightarrow$

$drop(1, v) \models \varphi$. The step $\star$ uses the `regexOmegaClosure` theorem in Section 7.6.2.

$$
\begin{aligned}
w \models \partial(r^\omega)[a] &\Leftrightarrow w \models \partial(r \diamond\!\!\rightarrow \mathbf{X}r^\omega)[a] \\
&\Leftrightarrow w \models ((OneStep(r)\,?\,r^\omega) \vee \delta(r) \diamond\!\!\rightarrow \mathbf{X}r^\omega)[a] \\
&\Leftrightarrow a \vDash OneStep(r) \wedge w \models r^\omega \vee w \models \delta(r)[a] \diamond\!\!\rightarrow \mathbf{X}r^\omega \\
&\Leftrightarrow a \vDash OneStep(r) \wedge w \models r^\omega \vee \exists\, i : take(i{+}1, w) \Vdash \delta(r)[a] \wedge drop(i, w) \models \mathbf{X}r^\omega \\
&\Leftrightarrow a \vDash OneStep(r) \wedge w \models r^\omega \vee \exists\, i : take(i{+}2, a{::}w) \Vdash r \wedge drop(i{+}2, a{::}w) \models r^\omega \\
&\Leftrightarrow take(1, a{::}w) \Vdash r \wedge drop(1, a{::}w) \in\!* r \vee \\
&\quad\ \exists\, i : take(i{+}2, a{::}w) \Vdash r \wedge drop(i{+}2, a{::}w) \in\!* r \\
&\Leftrightarrow \exists\, i > 0 : take(i, a{::}w) \Vdash r \wedge drop(i, a{::}w) \in\!* r \\
&\overset{\star}{\Leftrightarrow} a{::}w \in\!* r \Leftrightarrow a{::}w \models r^\omega
\end{aligned}
$$

The Lean formalization establishes full confidence in the derivative based semantics of **RLTL**. It also provided feedback that helped to streamline the theory. One particular example is the definition of $OneStep(r)$ that in its original form implicitly assumed $\mathcal{A}$ to be decidable by depending on satisfiability checking in $\mathcal{A}$, this assumption turned out to be unnecessary and lead to, both a simpler definition of $OneStep(r)$ in Section 7.2 as well as a more general theory that applies also to the case when $\mathcal{A}$ is undecidable.

7.7 Omega-Regularity Modulo Theories

Here we lift the classical concept of ω-regular languages [Büchi 1960; McNaughton 1966] as the languages accepted by ABW so as to be modulo $\mathcal{A}$, and show that $\mathbf{RLTL}^+\langle\mathcal{A}\rangle$ captures ω-regularity modulo $\mathcal{A}$. We say that $L \subseteq \Sigma^\omega$ is *$\omega_{\mathcal{A}}$-regular* if $L = \mathscr{L}(M)$ for some M that is an $NBW_{\mathcal{A}}$. We lift the following result from [Büchi 1960; McNaughton 1966] (see also [Thomas 1990, 1.1. Theorem]) as to be modulo $\mathcal{A}$. Recall that $\mathscr{L}(R^\omega) = \mathcal{L}(R)^\omega$.

Theorem 5 ($\omega_{\mathcal{A}}$-regularity). *A language $L \subseteq \Sigma^\omega$ is $\omega_{\mathcal{A}}$-regular $\Leftrightarrow$ there exist $n > 0$ and regexes $\{R_i\}_{i=1}^n$ and $\{S_i\}_{i=1}^n$ in $\mathbf{RE}\langle\mathcal{A}\rangle$ such that $L = \bigcup_{i=1}^n \mathcal{L}(R_i)\cdot\mathscr{L}(S_i^\omega)$.*

Proof. By using Lemma 2 and [McNaughton 1966]. □

Next we show that $\mathbf{RLTL}^+\langle\mathcal{A}\rangle$ captures $\omega_{\mathcal{A}}$-regularity. We use the following *stepping* lemma of the transition function of any ABW, the proof of which is based on the definition of accepting runs. For all $\phi \in \mathbb{B}^+(Q)$ lift the transition function ϱ to ϕ as usual, and for all $a \in \Sigma, u \in \Sigma^*$ let

$$\hat{\varrho}(\epsilon, \phi) \overset{\text{DEF}}{=} \phi \qquad \hat{\varrho}(a{::}u, \phi) \overset{\text{DEF}}{=} \hat{\varrho}(u, \varrho(\phi)[a]).$$

Lemma 5. $\forall M \in ABW_{\mathcal{A}}, \phi \in \mathbb{B}^+(Q_M), u \in \Sigma^*, w \in \Sigma^\omega : uw \in \mathscr{L}_M(\phi) \Leftrightarrow w \in \mathscr{L}_M(\hat{\varrho}_M(u, \phi))$.

Recall the construction of the $ABW_{\mathcal{A}}$ M_ϕ in Definition 7.6.

Theorem 6 ($\omega_{\mathcal{A}}$-Regularity of $\mathbf{RLTL}^+\langle\mathcal{A}\rangle$).
(1) *$L \subseteq \Sigma^\omega$ is $\omega_{\mathcal{A}}$-regular $\Leftrightarrow \exists \phi \in \mathbf{RLTL}^+\langle\mathcal{A}\rangle : L = \mathscr{L}(\phi)$*
(2) *$\forall \phi \in \mathbf{RLTL}^+\langle\mathcal{A}\rangle : \mathscr{L}(\phi) = \mathscr{L}(M_\phi)$*

Proof outline. Statement (1)'$\Rightarrow$' is proved by using Theorem 5. Statement (1)'$\Leftarrow$' follows from statement (2). Statement (2) is proved by extending the proof in [Vardi 1995] to handle suffix implications. The proof is by induction over ϕ and makes use of Lemma 5. Regex closure formulas are additional base cases that are based on derivative laws of **ERE**.

In order to improve readability we make use of more light-weigth notations. In particular, we write $w \Vdash \varphi$ for $w \in \mathscr{L}_M(\varphi)$ and recall that $w_{..i} = take(i{+}1, w)$ and $w_{i..} = drop(i, w)$. The statement (2) is for any $\phi \in \mathbf{RLTL}^+\langle\mathcal{A}\rangle$ equivalently formulated as

$$\forall w \in \Sigma^\omega : w \Vdash \phi \Leftrightarrow w \models \phi$$

We illustrate the key induction case of $\diamond\!\!\rightarrow$ and use of Lemma 5. By construction of $M = M_\phi$ we have that $\partial(q) \cong \varrho_M(q)$ and we let $\varrho = \varrho_M$ below. We also implicitly use the following property:

$$\forall R \in \mathbf{ERE}\langle\mathcal{A}\rangle, a \in \Sigma, u \in \Sigma^* : a \vDash \mathit{OneStep}(\hat{\delta}(u, R)) \Leftrightarrow ua \Vdash R$$

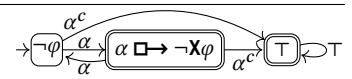
Let $w \in \Sigma^\omega$ and $\phi = R \diamond\!\!\rightarrow \psi$. The induction case is to prove that $w \Vdash \phi \Leftrightarrow w \models \phi$ and by the induction hypothesis it holds that $\forall v \in \Sigma^\omega : v \Vdash \psi \Leftrightarrow v \models \psi$. Let $\boldsymbol{k} \in \mathbb{N}$ be some large number.

$$\begin{aligned}
w \Vdash R \diamond\!\!\rightarrow \psi &\overset{\text{Lma 5}}{\Leftrightarrow} w_{1..} \Vdash \varrho(R \diamond\!\!\rightarrow \psi)[w_0] \\
&\overset{(28)}{\Leftrightarrow} w_{1..} \Vdash ((\mathit{OneStep}(R)\ ?\ \partial(\psi)) \vee (\delta(R) \diamond\!\!\rightarrow \psi))\ [w_0] \\
&\overset{(3)}{\Leftrightarrow} (w_0 \vDash R \wedge w_{1..} \Vdash \partial(\psi)[w_0]) \vee w_{1..} \Vdash \hat{\delta}(w_0, R) \diamond\!\!\rightarrow \psi \\
&\overset{\text{Lma 5}}{\Leftrightarrow} (w_{..0} \vDash R \wedge w_{0..} \Vdash \psi) \vee w_{1..} \Vdash \hat{\delta}(w_{..0}, R) \diamond\!\!\rightarrow \psi \\
&\overset{\dagger}{\Leftrightarrow} \bigvee_{i<\boldsymbol{k}} (w_{..i} \vDash R \wedge w_{i..} \Vdash \psi) \vee w_{\boldsymbol{k}..} \Vdash \hat{\delta}(w_{..\boldsymbol{k}-1}, R) \diamond\!\!\rightarrow \psi \\
&\overset{\ddagger}{\Leftrightarrow} \exists i : w_{..i} \vDash R \wedge w_{i..} \Vdash \psi \overset{\text{IH}}{\Leftrightarrow} \exists i : w_{..i} \vDash R \wedge w_{i..} \models \psi \overset{(23)}{\Leftrightarrow} w \models R \diamond\!\!\rightarrow \psi
\end{aligned}$$

In † we repeated the unfolding $\boldsymbol{k}$ times similarly to the previous steps, by using Lemma 5, for some large enough $\boldsymbol{k}$. In ‡ we used the key property that all states $r \diamond\!\!\rightarrow \psi$ in M are *nonaccepting* and would cause an infinite branch $(\hat{\delta}(w_{..\boldsymbol{k}+i}, R) \diamond\!\!\rightarrow \psi)_{i\geq 0}$ containing no accepting states if visited infinitely often, i.e., any state $\hat{\delta}(w_{..n}, R) \diamond\!\!\rightarrow \psi$ acts as $\bot$ because it is not visited for any $n \geq \boldsymbol{k}$. □

Derivatives of R^ω work correctly in Theorem 4, and therefore also when considering $\neg(R^\omega)$ because the semantics is based on **RLTL**. However, *the complemented derivatives arising from* $\neg\partial(R^\omega)$ *would in general have incorrect semantics as transitions of an ABW*, as illustrated next.

Example 7.11. Let $\alpha \in \mathcal{A}$, $a \in [\![\alpha]\!]$, and $\varphi = (\alpha{\cdot}\alpha)^\omega$, so $a^\omega \models \varphi$. The following derivatives arise from φ by (32) and (28), and where $\mathit{OneStep}(r)$ shows the predicate used in (28). If we were to start from $\neg\varphi$, by just negating the transition terms, we end up with an erroneous automaton M^{err}:

q	$\partial(q)$	$\mathit{OneStep}(r)$	$\partial(\neg q)$	M^{err}
φ	$(\alpha\ ?\ \alpha \diamond\!\!\rightarrow \mathbf{X}\varphi)$	$\bot$	$(\alpha\ ?\ \neg(\alpha \diamond\!\!\rightarrow \mathbf{X}\varphi) : \top)$	
$\alpha \diamond\!\!\rightarrow \mathbf{X}\varphi$	$(\alpha\ ?\ \varphi)$	α	$(\alpha\ ?\ \neg\varphi : \top)$	

where the state $\alpha \boxdot\!\!\rightarrow \neg\mathbf{X}\varphi$ is *accepting*, so $a^\omega \in \mathscr{L}(M^{\text{err}})$ while $a^\omega \not\models \neg\varphi$. ⊠

8 Related Work

Derivatives. Transition regexes for extended regular expressions [Stanford et al. 2021], a symbolic generalization of [Brzozowski 1964], have been implemented in Z3 [de Moura and Bjørner 2008]. We view $\mathbf{TTerm}\langle\mathcal{A}, \mathbf{LTL}\rangle$ as a symbolic generalization of *Vardi's derivatives for LTL* [Vardi 1995]. LTL based *monitoring* also uses classical derivatives [Havelund and Roşu 2001; Sen et al. 2003] combined with the decision procedure from [Hsiang 1985]. The relationship between using the tableaux method for LTL [Wolper 1985] vs. linear factors from [Antimirov 1996] were studied in [Sulzmann and Thiemann 2018]. The work in [Antimirov and Mosses 1995] studies Horn-equational reasoning as an alternative to derivatives.

Alternation Elimination. The original alternation elimination algorithm for *ABW* [Miyano and Hayashi 1984] was studied in-depth by [Boker et al. 2010] with an improved lower bound $O(n2^n)$ for *ordered ABW*.

Non-emptiness of *ABW* resulting from LTL is studied in [Bloem et al. 1999] through classification into *weak* and *terminal* cases where the problem is easier than in the general case.

The LTL to *NBW* procedure in [Fritz 2003] starts with an *ABW* and then computes delayed simulation relations on-the-fly using ϵ-transitions. In the final phase the *ABW* is translated into an *NBW* via [Miyano and Hayashi 1984].

The widely used algorithm of [Gerth et al. 1995] uses *tableau* to translate LTL into *NBW*. It first constructs a *Generalized Büchi automaton* (GBA) that is then, via a well-known encoding, translated into an *NBW*. Tableau based techniques for LTL were initially studied by Wolper [Wolper 1981, 1983, 1985]. A further extension of the tableau based technique for LTL is introduced in [Couvreur 1999] using on-the-fly expansion of *transition Büchi automata*.

Vardi's [Vardi 1994, Theorem 14, Proof] is the first LTL to *ABW* construction defined in terms of a step-wise unwinding essentially as derivatives. This construction is not symbolic, as it uses the next element to directly compute a Boolean combination of successor states. The work in [Tsay and Vardi 2021] gives a full construction of LTL to *symbolic co-ABW*. While aspects of the construction are similar to the one in our work, by being based on a symbolic representation of $\rho(q, a)$ – a key difference is in the representation of $\varrho(q)$ as a transition term providing a natural separation of concerns between evaluation of transitions from their target state formulas, that moreover works with *any EBA* $\mathcal{A}$. The work in [Gastin and Oddoux 2001] modifies Vardi's original construction to produce *very weak* co-ABW instead, prior to the GBA transformation.

The algorithm in [Wulf et al. 2008] defines *symbolic ABW* (sABW) where transition relations are Boolean combinations of literals and successor states, where incremental satisfiability and model checking methods use BDDs [Bryant 1986] both as the alphabet theory and to represent sets of states. For the construction from LTL to sABW the work refes to [Gastin and Oddoux 2001; Vardi 1995].

The first proof that LTL can be translated into Büchi automata appears in [Muller et al. 1988]. It uses weak alternating automata over trees and applies the reduction in [Muller et al. 1986].

The ITE aspect of transition terms is not preserved in the above works and the close relation between states as formulas, even if maintainable in some form in a GBA, gets then lost in the translation to *NBW*.

Extensions of LTL with Background Theories. In addition to related work discussed in Section 1, *LTL with constraints* [Demri and D'Souza 2007] is a fragment of first-order LTL, where LTL with *Presburger arithmetic* [Demri 2006] is a typical extension of LTL. While many decidable fragments exist, such as CLTL($\mathcal{D}$) [Demri and D'Souza 2007], constraint extensions in general lead to undecidability. Further extensions of LTL over infinite domains are introduced in [Grumberg et al. 2012] and subsequently formalized with *generalized register automata* in [Grumberg et al. 2013]. In [Faran and Kupferman 2018], *LTL with arithmetic or LTLA* is studied further with a focus on its decidable and undecidable fragments, where the model-checking problem of the existential fragment is shown to be in PSPACE and the problem is studied also for hierarchical systems.

Recent work in [Geatti et al. 2022] extends LTL with modulo theories over *finite* strings, the theory is generally *undecidable*. The work in [D'Antoni and Veanes 2017b] extends M2L-STR [Henriksen et al. 1995; Klarlund et al. 2002] (WS1S [Büchi 1960] for finite strings) as to be modulo theories. Recently LTL modulo theories has also been studied in [Rodríguez and Sánchez 2023] for *realizability* of temporal specifications where the core algorithms in connection to modulo theories are based on a variant of bitblasting combined with an interpretation back to SMT models. The most recent work in [Heim and Dimitrova 2025] further extends $\mathbf{LTL}\langle\mathcal{A}\rangle$ realizability by allowing *primed* theory variables, which greatly increases scalability for practical applications. Let w be a stream of interpretations over a set $X \cup \{x' \mid x \in X\}$ of theory variables. In terms of $\mathbf{LTL}\langle\mathcal{A}\rangle$ or $\mathbf{RLTL}\langle\mathcal{A}\rangle$ semantics, all *valid* streams $w \in \Sigma^\omega$ must then satisfy the additional semantic condition $\forall i \in \mathbb{N}, x \in X : w_i(x') = w_{i+1}(x)$. For example, $\mathbf{G}(x \leq x')$ implies that x is never decreasing.

In the above works derivatives are not being used and the techniques are in general orthogonal to what we propose here and further extend into $\mathbf{RLTL}\langle\mathcal{A}\rangle$.

Extensions of LTL to ω-Regularity. It has been widely recognized that full ω-regularity is fundamental for general applications of linear temporal properties [Pnueli 1985]. Classical LTL expresses precisely the star-free fragment of ω-regular events [Emerson 1991], e.g., it cannot express ω-regular properties such as $(p\cdot\top)^\omega$ (p holds in all even positions) [Wolper 1983]. Several ω-regular extension of LTL have been proposed: *ETL* [Vardi and Wolper 1994], *LTL with fixpoint operators* [Banieqbal and Barringer 1989], *QPTL* [Sistla et al. 1987] using quantifiers, and *ForSpec* [Armoni et al. 2002] using regular expressions over propositions. *MONA* [Henriksen et al. 1995] also supports LTL and is ω-regular. [Bustan et al. 2005, Proposition 3.14] notes that ω-regularity of ForSpec also carries over to PSL [Eisner and Fisman 2006].

Our extension $\mathbf{RLTL}\langle\mathcal{A}\rangle$ of $\mathbf{LTL}\langle\mathcal{A}\rangle$ with $\diamond\!\!\rightarrow$ and $\mathbf{\{\cdot\}}$ was inspired by SPOT [Duret-Lutz 2024; Duret-Lutz et al. 2022] and PSL because both of those operators elegantly admit incremental derivative based unfolding. Originally $\diamond\!\!\rightarrow$ appears in ForSpec as *follows_by* and is also related to the $\diamond$-modality in PDL [Fischer and Ladner 1979]. The primary purpose of PSL is for formal specification of concurrent systems. SPOT is a state-of-the-art verification tool that supports a large subset of PSL, including many optimizations that originate from [Cimatti et al. 2008]. **RLTL** covers a core subset of PSL that is consistent with the semantics of suffix implications in SPOT, but *differs* in one important aspect in regard to *weak closure* (and its negation), namely that, for any nullable regex R, $\mathbf{\{}R\mathbf{\}} \equiv \top$. For a PSL formula $\mathbf{\{}R\mathbf{\}}$ the corresponding weak closure in **RLTL** is $\mathbf{\{}R \Cap \top\mathbf{+\}}$. This difference is driven by the semantics of derivatives that effectively *enforces* the law $\mathbf{\{}R\mathbf{\}} \equiv \top$ whenever R is nullable, in order to maintain an algebraically well-behaved system of rules. Thus, some rewrite rules in SPOT, like $\mathbf{\{}r*\mathbf{\}} \equiv \mathbf{\{}r\mathbf{\}}$, are *invalid* in **RLTL**.

RLTL also supports regex *complement* that comes naturally because of the built-in duality law of transition regexes: $\mathtt{\sim}\mathbf{(}\alpha\,\mathbf{?}\,f\,\mathbf{:}\,g\mathbf{)} \equiv \mathbf{(}\alpha\,\mathbf{?}\,\mathtt{\sim}f\,\mathbf{:}\,\mathtt{\sim}g\mathbf{)}$ [Stanford et al. 2021, Lemma 4.2]. It is difficult and highly impractical to support ~ by other means, because it would in general require *determinization* of SFAs [Veanes et al. 2010] that is avoided by the duality law that propagates ~ *lazily*.

A fundamental difference is that **RLTL** lifts (a core subset of) PSL as to be *modulo theories*, while in PSL the atomic regexes α are Boolean combinations of *propositions*.

9 Future Work and Open Problems

The paper presents a self-contained framework with both theory and algorithms to implement symbolic model checking of $\mathbf{RLTL}\langle\mathcal{A}\rangle$. We implemented a prototype in F# where for nonemptiness we used the meticulously presented *nested depth-first cycle detection algorithm* in [Courcoubetis et al. 1992, Algorithm B], that required essentially no modifications. However, it is premature and out of scope to discuss that implementation here. For proper evaluation, one needs a standard that, e.g., combines PSL with SMT-LIB [SMT-LIB 2021], in order to express modulo theories, which, as of today, does not exist. That said, there are many interesting open problems we mention below.

Several kinds of optimizations can be applied. All leaves in DNF can be maintained as *antichains* based on (4) and leverage techniques from [Abdulla et al. 2010; Filiot et al. 2009; Wulf et al. 2008]. At the automata level, we believe that *(bi)simulation* algorithms [Abdulla et al. 2010; Bonchi and Pous 2013; Bustan and Grumberg 2003; Etessami et al. 2005; Fritz 2003; Fritz and Wilke 2002, 2005; Gurumurthy et al. 2002; Mayr and Clemente 2013] can be lifted to $NBW_{\mathcal{A}}$ similarly to the case of SFAs [D'Antoni and Veanes 2014, 2016, 2017a; Holík et al. 2018] that lift algorithms from [Bustan and Grumberg 2003; Hopcroft 1971; Paige and Tarjan 1987]. Symbolic derivatives can also be relevant for $DBW_{\mathcal{A}}$ – $NBW_{\mathcal{A}}$ *that omits* $\vee$ – and revisit results in [Baarir and Duret-Lutz 2014; Babiak et al. 2012; Baier and Katoen 2008; Diekert et al. 2015; Finkbeiner and Schewe 2005; Kupferman and

Rosenberg 2010; Kurshan 1987; Tourneur and Duret-Lutz 2017]. Implementation of **TTerm** can potentially also use *generic BDDs* [D'Antoni and Veanes 2017b] or *Shannon expansions* [Shannon 1949] for structure sharing. *Dead state detection* in M_R can use [Stanford and Veanes 2023].

To reduce the constant 4 in *product* of $NBW_{\mathcal{A}}$, we believe one can lift the algorithm in [Choueka 1974] ([Kupferman 2018, Theorem 2]) to be modulo $\mathcal{A}$. For *NBW complementation* [Büchi 1960] ([Kupferman 2018, Theorem 4]), an open question is if derivatives can be used to lift complementation to be modulo $\mathcal{A}$ to support $\neg(R^\omega)$ incrementally.

Counterfactual reasoning in form of causality analysis of ω-regular properties has recently been proposed in [Coenen et al. 2022]. In this context QPTL [Sistla et al. 1987] and HyperQPTL [Beutner and Finkbeiner 2023] are the target formalisms used in the tool developed in [Beutner et al. 2023]. Could derivatives be developed for HyperQPTL and be useful to extend causality analysis as to be modulo background theories?

The *past* operator does not increase expressivity of LTL [Gabbay et al. 1980] but is practically very useful [Armoni et al. 2002; Kupferman and Pnueli 1995; Kupferman et al. 2012; Laroussinie et al. 2002; Lichtenstein et al. 1985]. *Fair correctness* [Varacca and Völzer 2006] also relates to *past*. Derivatives are inherently forward looking, so understanding how to support *past* is intriguing. *Counting* [Laroussinie et al. 2010] with derivatives is another interesting topic. Could derivatives be developed for *CTL* [Piterman and Pnueli 2018] where *trees* rather than words are the semantic foundation? Derivatives for *finite* trees have been studied in [Attou et al. 2021].

A future challenge is to also formalize the other parts of the theory in *Lean*, including Theorem 1 and Theorem 6, for which the formalization developed here provides a necessary basis. Due to the intricate semantics of $ABW_{\mathcal{A}}$, such a formalization is an invaluable aid, not only for establishing correctness, but also to serve as a platform for formally validating correctness of new rewrite rules and other algorithmic optimizations. Formalizing Theorem 1 in Lean is a very difficult task, as it would require formalizing a generalization of [Miyano and Hayashi 1984] that is one of the most technically involved proofs in the classical literature of ω-regularity. A difficult aspect regarding a Lean formalization of either Theorem 1 or Theorem 6 is connected to the fact that automata algorithms are cumbersome to express in a proof assistant because, unlike trees or regexes or formulas, graphs are typically not defined inductively.

10 Conclusion

We have shown how transition terms and symbolic derivatives can be used to define a realizable symbolic semantics for (alternating) Büchi automata and linear temporal logic (LTL). The semantics is parameterized by an EBA for the base alphabetic domain, which enables it to apply to ω-languages and infinite alphabets in an algebraically well-defined and precise manner. This framework allows syntactic rewrite rules for LTL extended with regular expressions (RLTL) to be applied *on-the-fly* during alternation elimination, where they simultaneously respect the semantics of RLTL formulas and their alternating Büchi automata. Similarly to the new algorithm Æ, there is a rich landscape of further optimizations and algorithms yet to be discovered.

Acknowledgments

We thank *Olli Saarikivi* for the collaboration that lead to numerous breakthroughs during initial development of the theory. We also thank the anonymous reviewers for their helpful suggestions.

Artifact Availability

The complete Lean formalization of Theorem 4, i.e., `theorem derivation` in Lean, including all the required background theory, as outlined in Section 7.6, is part of the active github project [Zhuchko 2024]. The artifact [Zhuchko and Ebner 2024] is a snapshot of that project.

References

Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr, and Tomáš Vojnar. 2010. When Simulation Meets Antichains. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (Paphos, Cyprus) *(LNCS, Vol. 6015)*, Javier Esparza and Rupak Majumdar (Eds.). Springer, Berlin, Heidelberg, 158–174. https://doi.org/10.1007/978-3-642-12002-2_14

Valentin Antimirov. 1996. Partial Derivatives of Regular Expressions and Finite Automata Constructions. *Theoretical Computer Science* 155, 2 (1996), 291–319. https://doi.org/10.1007/3-540-59042-0_96

Valentin Antimirov and Peter Mosses. 1995. Rewriting extended regular expressions. *Theoretical Computer Science* 143 (1995), 51–72. https://doi.org/10.1016/0304-3975(95)80010-7

Roy Armoni, Limor Fix, Alon Flaisher, Rob Gerth, Boris Ginsburg, Tomer Kanza, Avner Landver, Sela Mador-Haim, Eli Singerman, Andreas Tiemeyer, Moshe Y. Vardi, and Yael Zbar. 2002. The ForSpec Temporal Logic: A New Temporal Property-Specification Language. In *TACAS* (Grenoble, France) *(LNCS, Vol. 2280)*, Joost-Pieter Katoen and Perdita Stevens (Eds.). Springer, Berlin, Heidelberg, 296–311. https://doi.org/10.1007/3-540-46002-0_21

Samira Attou, Ludovic Mignot, and Djelloul Ziadi. 2021. Bottom-Up derivatives of tree expressions. *RAIRO-Theor. Inf. Appl.* 55, 4 (2021), 21 pages. https://doi.org/10.1051/ita/2021008

Souheib Baarir and Alexandre Duret-Lutz. 2014. Mechanizing the Minimization of Deterministic Generalized Büchi Automata. In *Proceedings of the 34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE'14)* (Berlin, Germany) *(LNCS, Vol. 8461)*. Springer, Berlin, Heidelberg, 266–283. https://doi.org/10.1007/978-3-662-43613-4_17

Tomás Babiak, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. 2012. LTL to Büchi Automata Translation: Fast and More Deterministic. In *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012* (Tallinn, Estonia) *(LNCS, Vol. 7214)*, Cormac Flanagan and Barbara König (Eds.). Springer, Berlin, Heidelberg, 95–109. https://doi.org/10.1007/978-3-642-28756-5_8

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press, Cambridge, MA, USA. https://mitpress.mit.edu/9780262026499/principles-of-model-checking/

Behnam Banieqbal and Howard Barringer. 1989. Temporal logic with fixed points. In *Temporal Logic in Specification* (Altrincham, UK) *(LNCS, Vol. 398)*, B. Banieqbal, H. Barringer, and A. Pnueli (Eds.). Springer, Berlin, Heidelberg, 62–74. https://doi.org/10.1007/3-540-51803-7_22

Raven Beutner and Bernd Finkbeiner. 2023. Model Checking Omega-Regular Hyperproperties with AutoHyperQ. In *Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR* (Manizales, Colombia) *(EPiC Series in Computing, Vol. 94)*, Ruzica Piskac and Andrei Voronkov (Eds.). EasyChair, online, 23–35. https://doi.org/10.29007/1xjt

Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Julian Siber. 2023. Checking and Sketching Causes on Temporal Sequences. In *Automated Technology for Verification and Analysis, ATVA* (Singapore) *(LNCS, Vol. 14216)*, Étienne André and Jun Sun (Eds.). Springer, Cham, 314–327. https://doi.org/10.1007/978-3-031-45332-8_18

Roderick Bloem, Kavita Ravi, and Fabio Somenzi. 1999. Efficient Decision Procedures for Model Checking of Linear Time Logic Properties. In *Computer Aided Verification, CAV'99* (Trento, Italy) *(LNCS, Vol. 1633)*, Nicolas Halbwachs and Doron Peled (Eds.). Springer, Berlin, Heidelberg, 222–235. https://doi.org/10.1007/3-540-48683-6_21

Udi Boker, Orna Kupferman, and Adin Rosenberg. 2010. Alternation Removal in Büchi Automata. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Proceedings, Part II* (Bordeaux, France) *(LNCS, Vol. 6199)*, Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis (Eds.). Springer, Berlin, Heidelberg, 76–87. https://doi.org/10.1007/978-3-642-14162-1_7

Filippo Bonchi and Damien Pous. 2013. Checking NFA Equivalence with Bisimulations up to Congruence. *ACM SIGPLAN Notices (POPL'13)* 48, 1 (2013), 457–468. https://doi.org/10.1145/2480359.2429124

Randal Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.* 35, 8 (1986), 677–691. https://doi.org/10.1109/TC.1986.1676819

Janusz A. Brzozowski. 1964. Derivatives of regular expressions. *JACM* 11 (1964), 481–494. https://doi.org/10.1145/321239.321249

J. Richard Büchi. 1960. On a decision method in restricted second order arithmetic. In *Logic, methodology and philosophy of science, Proceedings of the 1960 International Congress*, Ernest Nagel, Patrick Suppes, and Alfred Tarski (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 44. Elsevier, Amsterdam, 1–11. https://doi.org/10.1016/S0049-237X(09)70564-6

Doron Bustan, Dana Fisman, and John Havlicek. 2005. *Automata construction for PSL*. Technical Report Report MCS05-04. The Weizmann Institute of Science. https://api.semanticscholar.org/CorpusID:14807945

Doron Bustan and Orna Grumberg. 2003. Simulation-Based Minimization. *ACM Trans. Comput. Logic* 4, 2 (2003), 181–206. https://doi.org/10.1145/635499.635502

Yaacov Choueka. 1974. Theories of automata on ω-tapes: A simplified approach. *J. Computer and System Sciences* 8 (1974), 117–141. https://doi.org/10.1016/S0022-0000(74)80051-6

Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. 2008. Symbolic compilation of PSL. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems* 27, 10 (2008), 1737–1750. https://doi.org/10.1109/TCAD.2008.2003303

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 1999. *Model Checking*. MIT Press, Cambridge, MA, USA. https://mitpress.mit.edu/9780262032704/model-checking/

Norine Coenen, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Niklas Metzger, and Julian Siber. 2022. Temporal Causality in Reactive Systems. In *Automated Technology for Verification and Analysis* (Virtual Event) *(LNCS, Vol. 13505)*, Ahmed Bouajjani, Lukáš Holík, and Zhilin Wu (Eds.). Springer, Cham, 208–224. https://doi.org/10.1007/978-3-031-19992-9_13

Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper, and Mihalis Yannakakis. 1992. Memory-Effcient Algorithms for the Verification of Temporal Properties. *Formal Methods in System Design* 1 (1992), 275–288. https://doi.org/10.1007/BF00121128

Jean-Michel Couvreur. 1999. On-the-Fly Verification of Linear Temporal Logic. In *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems* (Toulouse, France) *(LNCS, Vol. 1708)*, Jeannette M. Wing, Jim Woodcock, and Jim Davies (Eds.). Springer, Berlin, Heidelberg, 253–271. https://doi.org/10.1007/3-540-48119-2_16

Loris D'Antoni and Margus Veanes. 2014. Minimization of Symbolic Automata. *ACM SIGPLAN Notices – POPL'14* 49, 1 (2014), 541–553. https://doi.org/10.1145/2535838.2535849

Loris D'Antoni and Margus Veanes. 2016. Minimization of Symbolic Tree Automata. In *LICS '16* (New York, NY, USA). IEEE, Piscataway, NJ, USA, 873–882. https://doi.org/10.1145/2933575.2933578

Loris D'Antoni and Margus Veanes. 2017a. Forward Bisimulations for Nondeterministic Symbolic Finite Automata. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017* (Uppsala, Sweden) *(LNCS, Vol. 10205)*. Springer, Berlin, Heidelberg, 518–534. https://doi.org/10.1007/978-3-662-54577-5_30

Loris D'Antoni and Margus Veanes. 2017b. Monadic second-order logic on finite sequences. *ACM SIGPLAN Notices – POPL'17* 52, 1 (2017), 232–245. https://doi.org/10.1145/3093333.3009844

Loris D'Antoni and Margus Veanes. 2021. Automata Modulo Theories. *Commun. ACM* 64, 5 (May 2021), 86–95. https://doi.org/10.1145/3419404

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS'08* (Budapest, Hungary) *(LNCS, Vol. 4963)*. Springer, Berlin, Heidelberg, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

Leonardo de Moura and Nikolaj Bjørner. 2011. Satisfiability Modulo Theories: Introduction & Applications. *Commun. ACM* 54, 9 (Sept. 2011), 69–77. https://doi.org/10.1145/1995376.1995394

Stéphane Demri. 2006. Linear-time temporal logics with Presburger constraints: an overview. *Journal of Applied Non-Classical Logics* 16, 3–4 (2006), 311–347. https://doi.org/10.3166/jancl.16.311-347

Stéphane Demri and Deepak D'Souza. 2007. An automata-theoretic approach to constraint LTL. *Information and Computation* 205, 3 (2007), 380–415. https://doi.org/10.1016/j.ic.2006.09.006

Volker Diekert, Anca Muscholl, and Igor Walukiewicz. 2015. A Note on Monitors and Büchi Automata. In *Theoretical Aspects of Computing - ICTAC 2015* (Cali, Colombia) *(LNCS, Vol. 9399)*. Springer, Cham, 39–57. http://dx.doi.org/10.1007/978-3-319-25150-9_3

Daniel Dietsch, Matthias Heizmann, Vincent Langenfeld, and Andreas Podelski. 2015. Fairness Modulo Theory: A New Approach to LTL Software Model Checking. In *Computer Aided Verification (CAV'15)* (San Francisco, CA, USA) *(LNCS, Vol. 9206)*, Daniel Kroening and Corina S. Păsăreanu (Eds.). Springer, Cham, 49–66. https://doi.org/10.1007/978-3-319-21690-4_4

Alexandre Duret-Lutz. 2024. Spot: a platform for LTL and ω-automata manipulation. https://spot.lre.epita.fr

Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. 2022. From Spot 2.0 to Spot 2.10: What's New?. In *34th International Conference on Computer Aided Verification (CAV'22)* (Haifa, Israel) *(LNCS, Vol. 13372)*. Springer, Cham, 174–187. https://doi.org/10.1007/978-3-031-13188-2_9

Cindy Eisner and Dana Fisman. 2006. *A Practical Introduction to PSL*. Springer, New York, NY, USA. https://doi.org/10.1007/978-0-387-36123-9

E. Allen Emerson. 1991. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science (Vol. B): Formal Models and Semantics*. MIT Press, Cambridge, MA, USA, 995–1072. https://dl.acm.org/doi/book/10.5555/114891

E. Allen Emerson and Chin-Laung Lei. 1985. Temporal reasoning under generalized fairness constraints. In *STACS 86* (Orsay, France) *(LNCS, Vol. 210)*, B. Monien and G. Vidal-Naquet (Eds.). Springer, Berlin, Heidelberg, 21–36. https://doi.org/10.1007/3-540-16078-7_62

E. Allen Emerson and Chin-Laung Lei. 1987. Modalities for model checking: branching time logic strikes back. *Science of Computer Programming* 8, 3 (1987), 275–306. https://doi.org/10.1016/0167-6423(87)90036-0

Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. 2005. Fair simulation relations, parity games, and state space reduction for Büchi automata. *SIAM J. Comput.* 34, 5 (May 2005), 1159–1175. https://doi.org/10.1137/S0097539703420675

Rachel Faran and Orna Kupferman. 2018. LTL with Arithmetic and its Applications in Reasoning about Hierarchical Systems. In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning* (Montevideo, Uruguay) *(EPiC Series in Computing, Vol. 57)*. EasyChair, online, 343–362. https://doi.org/10.29007/wpg3

Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. 2009. An antichain algorithm for LTL realizability. In *Computer Aided Verification (CAV'09)* (Grenoble, France) *(LNCS, Vol. 5643)*, Ahmed Bouajjani and Oded Maler (Eds.). Springer, Berlin, Heidelberg, 263–277. https://doi.org/10.1007/978-3-642-02658-4_22

Bernd Finkbeiner, Philippe Heim, and Noemi Passing. 2022. Temporal Stream Logic modulo Theories. In *Foundations of Software Science and Computation Structures (FoSSaCS'22)* (Munich, Germany) *(LNCS, Vol. 13242)*, Patricia Bouyer and Lutz Schröder (Eds.). Springer, Cham, 325–346. https://doi.org/10.1007/978-3-030-99253-8_17

Bernd Finkbeiner, Felix Klein, Ruzica Piskac, and Mark Santolucito. 2019. Temporal Stream Logic: Synthesis Beyond the Bools. In *Computer Aided Verification (CAV'19)* (New York, NY, USA) *(LNCS, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, Cham, 609–629. https://doi.org/10.1007/978-3-030-25540-4_35

Bernd Finkbeiner and Sven Schewe. 2005. Uniform distributed synthesis. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)* (Chicago, IL, USA). IEEE, Piscataway, NJ, USA, 321–330. https://doi.org/10.1109/LICS.2005.53

Michael J. Fischer and Richard E. Ladner. 1979. Propositional dynamic logic of regular programs. *J. Comput. System Sci.* 18, 2 (1979), 194–211. https://doi.org/10.1016/0022-0000(79)90046-1

Carsten Fritz. 2003. Constructing Büchi Automata from Linear Temporal Logic Using Simulation Relations for Alternating Büchi Automata. In *Implementation and Application of Automata* (Santa Barbara, CA, USA) *(LNCS, Vol. 2759)*, Oscar H. Ibarra and Zhe Dang (Eds.). Springer, Berlin, Heidelberg, 35–48. https://doi.org/10.1007/3-540-45089-0_5

Carsten Fritz and Thomas Wilke. 2002. State space reductions for alternating Büchi automata: Quotienting by simulation equivalences. In *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science* (Kanpur, India) *(LNCS, Vol. 2556)*. Springer, Berlin, Heidelberg, 157–169. https://doi.org/10.1007/3-540-36206-1_15

Carsten Fritz and Thomas Wilke. 2005. Simulation Relations for Alternating Büchi Automata. *Theor. Comput. Sci.* 338, 1–3 (June 2005), 275–314. https://doi.org/10.1016/j.tcs.2005.01.016

Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. 1980. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'80)* (Las Vegas, NV, USA). ACM, New York, NY, USA, 163–173. https://doi.org/10.1145/567446.567462

Paul Gastin and Denis Oddoux. 2001. Fast LTL to Büchi Automata Translation. In *Computer Aided Verification, 13th International Conference, CAV* (Paris, France) *(LNCS, Vol. 2102)*, Gérard Berry, Hubert Comon, and Alain Finkel (Eds.). Springer, Berlin, Heidelberg, 53–65. https://doi.org/10.1007/3-540-44585-4_6

Luca Geatti, Alessandro Gianola, and Nicola Gigante. 2022. Linear Temporal Logic Modulo Theories over Finite Traces (Extended Version). https://doi.org/10.48550/ARXIV.2204.13693

Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification* (Sitges, Spain), Piotr Dembinski and Marek Sredniawa (Eds.). Chapman & Hall, GBR, 3–18. https://dl.acm.org/doi/10.5555/645837.670574

Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. 2012. Model Checking Systems and Specifications with Parameterized Atomic Propositions. In *Automated Technology for Verification and Analysis (ATVA'12)* (Thiruvananthapuram, India) *(LNCS, Vol. 7561)*. Springer, Berlin, Heidelberg, 122–136. https://doi.org/10.1007/978-3-642-33386-6_11

Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. 2013. An Automata-Theoretic Approach to Reasoning about Parameterized Systems and Specifications. In *Automated Technology for Verification and Analysis (ATVA'13)* (Hanoi, Vietnam) *(LNCS, Vol. 8172)*. Springer, Cham, 397–411. https://doi.org/10.1007/978-3-319-02444-8_28

Sankar Gurumurthy, Roderick Bloem, and Fabio Somenzi. 2002. Fair Simulation Minimization. In *Computer Aided Verification, CAV 2002* (Copenhagen, Denmark) *(LNCS, Vol. 2404)*, Ed Brinksma and Kim Guldstrand Larsen (Eds.). Springer, Berlin, Heidelberg, 610–623. https://doi.org/10.1007/3-540-45657-0_51

Klaus Havelund and Grigore Roşu. 2001. Monitoring Programs Using Rewriting. In *16th IEEE International Conference on Automated Software Engineering (ASE'01)* (San Diego, CA, USA). IEEE, Piscataway, NJ, USA, 135–143. https://dl.acm.org/doi/10.5555/872023.872572

Philippe Heim and Rayna Dimitrova. 2025. Translation of Temporal Logic for Efficient Infinite-State Reactive Synthesis. *Proc. ACM Program. Lang.* 9, POPL, Article 52 (January 2025), 32 pages. https://doi.org/10.1145/3704888

Jesper G. Henriksen, Jakob Jensen, Michael Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. 1995. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)* (Aarhus, Denmark) *(LNCS, Vol. 1019)*, E. Brinksma, W. R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen (Eds.). Springer, Berlin, Heidelberg, 89–110. https://doi.org/10.1007/3-540-60630-0_5

Lukáš Holík, Ondřej Lengál, Juraj Síč, Margus Veanes, and Tomáš Vojnar. 2018. Simulation Algorithms for Symbolic Automata. In *Automated Technology for Verification and Analysis (ATVA'18)* (Los Angeles, CA, USA) *(LNCS, Vol. 11138)*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, Cham, 109–125. https://doi.org/10.1007/978-3-030-01090-4_7

John E. Hopcroft. 1971. *An n log n algorithm for minimizing states in a finite automaton.* Technical Report. Stanford University, Stanford, CA, USA. https://dl.acm.org/doi/10.5555/891883

Jieh Hsiang. 1985. Refutational theorem proving using term-rewriting systems. *Artificial Intelligence* 25, 3 (1985), 255–300. https://doi.org/10.1016/0004-3702(85)90074-8

Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. 2002. MONA Implementation Secrets. *International Journal of Foundations of Computer Science* 13, 4 (2002), 571–586. https://doi.org/10.1142/S012905410200128X

Orna Kupferman. 2018. Automata Theory and Model Checking. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer, Cham, Chapter 4, 107–151. https://doi.org/10.1007/978-3-319-10575-8_4

Orna Kupferman and Amir Pnueli. 1995. Once and for all. In *Tenth Annual IEEE Symposium on Logic in Computer Science (LICS'95)* (San Diego, CA, USA). IEEE, Piscataway, NJ, USA, 25–35. https://dl.acm.org/doi/10.5555/788017.788753

Orna Kupferman, Amir Pnueli, and Moshe Y. Vardi. 2012. Once and for All. *J. Comput. Syst. Sci.* 78, 3 (May 2012), 981–996. https://doi.org/10.1016/j.jcss.2011.08.006

Orna Kupferman and Adin Rosenberg. 2010. The Blowup in Translating LTL to Deterministic Automata. In *Model Checking and Artificial Intelligence - 6th International Workshop, MoChArt 2010* (Atlanta, GA, USA) *(LNCS/LNAI, Vol. 6572)*, Ron van der Meyden and Jan-Georg Smaus (Eds.). Springer, Berlin, Heidelberg, 85–94. https://doi.org/10.1007/978-3-642-20674-0_6

Robert P. Kurshan. 1987. Complementing Deterministic Büchi Automata in Polynomial Time. *J. Comput. System Sci.* 35 (1987), 59–71. https://doi.org/10.1016/0022-0000(87)90036-5

François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. 2002. Temporal Logic with Forgettable Past. In *17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)* (Vienna, Austria). IEEE, Piscataway, NJ, USA, 383–392. https://dl.acm.org/doi/10.5555/645683.664573

François Laroussinie, Antoine Meyer, and Eudes Petonnet. 2010. Counting LTL. In *17th International Symposium on Temporal Representation and Reasoning* (Paris, France). IEEE, Piscataway, NJ, USA, 51–58. https://doi.org/10.1109/TIME.2010.20

Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. 1985. The glory of the past. In *Logics of Programs* (Brooklyn, NY, USA) *(LNCS, Vol. 193)*, Rohit Parikh (Ed.). Springer, Berlin, Heidelberg, 196–218. https://doi.org/10.1007/3-540-15648-8_16

Richard Mayr and Lorenzo Clemente. 2013. Advanced Automata Minimization. *SIGPLAN Not. POPL'13* 48, 1 (Jan 2013), 63–74. https://doi.org/10.1145/2480359.2429079

Robert McNaughton. 1966. Testing and Generating Infinite Sequences by a Finite Automaton. *Inf. Control.* 9, 5 (1966), 521–530. https://doi.org/10.1016/S0019-9958(66)80013-X

Microsoft. 2023. App Configuration. https://azure.microsoft.com/en-us/products/app-configuration/

Satoru Miyano and Takeshi Hayashi. 1984. Alternating finite automata on ω-words. *Theoretical Computer Science* 32, 3 (1984), 321–330. https://doi.org/10.1016/0304-3975(84)90049-5

Dan Moseley, Mario Nishio, Jose Perez Rodriguez, Olli Saarikivi, Stephen Toub, Margus Veanes, Tiki Wan, and Eric Xu. 2023. Derivative Based Nonbacktracking Real-World Regex Matching with Backtracking Semantics. In *PLDI '23: 44th ACM SIGPLAN International Conference on Programming Language Design and Implementation*, Nate Foster et al. (Ed.). ACM, New York, NY, USA, 1026–1049. https://doi.org/10.1145/3591262

David Muller, Ahmed Saoudi, and Paul Schupp. 1986. Alternating automata, the weak monadic theory of the tree, and its complexity. In *International Colloquium on Automata, Languages and Programming on Automata, Languages and Programming (ICALP'86)* (Rennes, France). Springer, Berlin, Heidelberg, 275—-283. https://dl.acm.org/doi/10.5555/16046.16075

David Muller, Ahmed Saoudi, and Paul Schupp. 1988. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Third Annual Symposium on Logic in Computer Science (LICS'88)* (Edinburgh, Scotland). IEEE, Piscataway, NJ, USA, 422–427. https://doi.org/10.1109/LICS.1988.5139

Robert Paige and Robert E. Tarjan. 1987. Three Partition Refinement Algorithms. *SIAM J. Comput.* 16, 6 (1987), 973–989. https://doi.org/10.1137/0216062

Nir Piterman and Amir Pnueli. 2018. Temporal Logic and Fair Discrete Systems. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer, Cham, 27–73. https://doi.org/10.1007/978-3-319-10575-8_2

Amir Pnueli. 1977. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science* (Providence, RI, USA). IEEE, Piscataway, NJ, USA, 46–57. https://doi.org/10.1109/SFCS.1977.32

Amir Pnueli. 1985. In Transition From Global to Modular Temporal Reasoning about Programs. In *Logics and Models of Concurrent Systems* (Louvain-la-Neuve, Belgium), Krzysztof R. Apt (Ed.). Springer, Berlin, Heidelberg, 123–144. https://doi.org/10.1007/978-3-642-82453-1_5

Amir Pnueli and Roni Rosner. 1989. On the synthesis of an asynchronous reactive module. In *Automata, Languages and Programming (ICALP'89)* (Stresa, Italy) *(LNCS, Vol. 372)*, Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Della Rocca (Eds.). Springer, Berlin, Heidelberg, 652–671. https://doi.org/10.1007/BFb0035790

Michael O. Rabin. 1970. Weakly definable relations and special automata. In *Mathematical Logic and Foundations of Set Theory*, Yehoshua Bar-Hillel (Ed.). Studies in Logic and the Foundations of Mathematics, Vol. 59. Elsevier, Amsterdam, 1–23. https://doi.org/10.1016/S0049-237X(08)71929-3

Andoni Rodríguez and César Sánchez. 2023. Boolean Abstractions for Realizability Modulo Theories. In *Computer Aided Verification (CAV'23)* (Paris, France) *(LNCS, Vol. 13966)*, Constantin Enea and Akash Lal (Eds.). Springer, Cham, 305–328. https://doi.org/10.1007/978-3-031-37709-9_15

Koushik Sen, Grigore Roşu, and Gul Agha. 2003. Generating Optimal Linear Temporal Logic Monitors by Coinduction. In *Advances in Computing Science – ASIAN 2003* (Mumbai, India) *(LNCS, Vol. 2896)*, Vijay A. Saraswat (Ed.). Springer, Berlin, Heidelberg, 260–275. https://doi.org/10.1007/978-3-540-40965-6_17

Claude. E. Shannon. 1949. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal* 28, 1 (1949), 59–98. https://doi.org/10.1002/j.1538-7305.1949.tb03624.x

A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. 1987. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science* 49, 2 (1987), 217–237. https://doi.org/10.1016/0304-3975(87)90008-9

SMT-LIB. 2021. https://smtlib.cs.uiowa.edu/

Fabio Somenzi and Roderick Bloem. 2000. Efficient Büchi Automata from LTL Formulae. In *Computer Aided Verification, 12th International Conference, CAV* (Chicago, IL, USA) *(LNCS, Vol. 1855)*, E. Allen Emerson and A. Prasad Sistla (Eds.). Springer, Berlin, Heidelberg, 248–263. https://doi.org/10.1007/10722167_21

Caleb Stanford and Margus Veanes. 2023. Incremental Dead State Detection in Logarithmic Time. In *Computer Aided Verification, CAV 2023* (Paris, France) *(LNCS, Vol. 13965)*, Constantin Enea and Akash Lal (Eds.). Springer, Cham, 241–264. https://doi.org/10.1007/978-3-031-37703-7_12

Caleb Stanford, Margus Veanes, and Nikolaj S. Bjørner. 2021. Symbolic Boolean derivatives for efficiently solving extended regular expression constraints. In *PLDI'21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada), Stephen N. Freund and Eran Yahav (Eds.). ACM, New York, NY, USA, 620–635. https://doi.org/10.1145/3453483.3454066

Martin Sulzmann and Peter Thiemann. 2018. LTL Semantic Tableaux and Alternating ω-automata via Linear Factors. In *Theoretical Aspects of Computing – ICTAC 2018* (Stellenbosch, South Africa) *(LNCS, Vol. 11187)*, Bernd Fischer and Tarmo Uustalu (Eds.). Springer, Cham, 11–34. https://doi.org/10.1007/978-3-030-02508-3_2

Wolfgang Thomas. 1990. Automata on Infinite Objects. In *Formal Models and Semantics*, Jan Van Leeuwen (Ed.). Elsevier, Amsterdam, Chapter 4, 133–191. https://doi.org/10.1016/B978-0-444-88074-1.50009-3

Vincent Tourneur and Alexandre Duret-Lutz. 2017. Fixes for two equations in *The Blow-Up in Translating LTL to Deterministic Automata* by Kupferman and Rosenberg. The original authors have reviewed the fixes and agreed with them.

Yih-Kuen Tsay and Moshe Y. Vardi. 2021. From Linear Temporal Logics to Büchi Automata: The Early and Simple Principle. In *Model Checking, Synthesis, and Learning - Essays Dedicated to Bengt Jonsson on The Occasion of His 60th Birthday (LNCS, Vol. 13030)*, Ernst-Rüdiger Olderog, Bernhard Steffen, and Wang Yi (Eds.). Springer, Cham, 8–40. https://doi.org/10.1007/978-3-030-91384-7_2

Daniele Varacca and Hagen Völzer. 2006. Temporal Logics and Model Checking for Fairly Correct Systems. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)* (Seattle, WA, USA). IEEE, Piscataway, NJ, USA, 389–398. https://doi.org/10.1109/LICS.2006.49

Moshe Y. Vardi. 1994. Nontraditional applications of automata theory. In *Theoretical Aspects of Computer Software* (Sendai, Japan) *(LNCS, Vol. 789)*, Masami Hagiya and John C. Mitchell (Eds.). Springer, Berlin, Heidelberg, 575–597. https://doi.org/10.1007/3-540-57887-0_116

Moshe Y. Vardi. 1995. An Automata-Theoretic Approach to Linear Temporal Logic. In *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop)* (Banff, Alberta, Canada) *(LNCS, Vol. 1043)*, Faron Moller and Graham M. Birtwistle (Eds.). Springer, Berlin, Heidelberg, 238–266. https://doi.org/10.1007/3-540-60915-6_6

Moshe Y. Vardi and Pierre Wolper. 1986. An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report). In *Proceedings of the Symposium on Logic in Computer Science (LICS'86)* (Cambridge, MA, USA). IEEE, Piscataway, NJ, USA, 332–344. https://api.semanticscholar.org/CorpusID:38567081

Moshe Y. Vardi and Pierre Wolper. 1994. Reasoning about infinite computations. *Information and Computation* 115, 1 (1994), 1–37. https://doi.org/10.1006/inco.1994.1092

Margus Veanes, Peli de Halleux, and Nikolai Tillmann. 2010. Rex: Symbolic Regular Expression Explorer. In *Third International Conference on Software Testing, Verification and Validation (ICST'10)* (Paris, France). IEEE, Piscataway, NJ, USA, 498–507. https://doi.org/10.1109/ICST.2010.15

Pierre Wolper. 1981. Temporal Logic Can Be More Expressive. In *22nd Annual Symposium on Foundations of Computer Science* (Nashville, TN, USA). IEEE, Piscataway, NJ, USA, 340–348. https://doi.org/10.1109/SFCS.1981.44

Pierre Wolper. 1983. Temporal logic can be more expressive. *Information and Control* 56, 1 (1983), 72–99. https://doi.org/10.1016/S0019-9958(83)80051-5

Pierre Wolper. 1985. The tableau method for temporal logic: an overview. *Logique Et Analyse* 28 (1985), 119–136. https://api.semanticscholar.org/CorpusID:118632087

Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. 2008. Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking. In *TACAS* (Budapest, Hungary) *(LNCS, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, Berlin, Heidelberg, 63–77. https://doi.org/10.1007/978-3-540-78800-3_6

Ekaterina Zhuchko. 2024. Lean4 Formalization of RLTL. https://github.com/ezhuchko/RLTL-derivatives

Ekaterina Zhuchko and Gabriel Ebner. 2024. Artifact for this paper. https://doi.org/10.5281/zenodo.14092718

Ekaterina Zhuchko, Margus Veanes, and Gabriel Ebner. 2024. Lean Formalization of Extended Regular Expression Matching with Lookarounds. In *13th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP'24)* (London, UK). ACM, New York, NY, USA, 118–131. https://doi.org/10.1145/3636501.3636959

Received 2024-07-08; accepted 2024-11-07