

FP6-IST-507219

PROSYD:

Property-Based System Design

Instrument: Specific Targeted Research Project

Thematic Priority: Information Society Technologies

Automata Construction Algorithms Optimized for PSL (Deliverable 3.2/4)

Due date of deliverable: July 1, 2005
Actual submission date: June 30, 2005

Start date of project: January 1, 2004

Duration: Three years

Organisation name of lead contractor for this deliverable: TU-Graz

Revision 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	<input checked="" type="checkbox"/>
PP	Restricted to other programme participants (including the Commission Services)	<input type="checkbox"/>
RE	Restricted to a group specified by the consortium (including the Commission Services)	<input type="checkbox"/>
CO	Confidential, only for members of the consortium (including the Commission Services)	<input type="checkbox"/>

Notices

For information, contact Roderick Bloem rbloem@ist.tugraz.at.

This document is intended to fulfil the contractual obligations of the PROSYD project concerning deliverable 3.2/4 described in contract number 507219.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

© Copyright PROSYD 2005. All rights reserved.

Table of Revisions

Version	Date	Description and reason	By	Affected sections
0.1	March, 2005	Creation	Pill	
0.2	March-April, 2005	Draft document (MS-Word)	Pill	
0.3	April 13, 2005	Transition to \LaTeX format	Griesmayer, Pill	
0.4	April, 2005	Extend basic content, TUG research results	Pill, Griesmayer	
0.5	May 2005	Incorporate work on finite words	Fisman, Pill, Griesmayer	
0.6	May 2005	Extend approach	Pill, Fisman, Griesmayer	
0.7	May 2005	Proof of weakness added	Fisman	
0.8	May 31, 2005	Assorted changes	Pill, Fisman, Griesmayer, Bloem	
0.9	June 7, 2005	Removed need for generalized automata	Griesmayer, Bloem	
0.95	June 9, 2005	Release candidate sent to partners	Bloem, Pill, Griesmayer, Fisman	
0.99	June 24, 2005	Incorporated comments, minor improvements	Pill, Bloem, Fisman	
1.0	June 27, 2005	Final approval by project coordinator	Eisner	

Authors

Shoham Ben-David
Roderick Bloem
Dana Fisman*
Andreas Griesmayer
Ingo Pill
Sitvanit Ruah

*Part of the work of this author was done in joint research with Doron Bustan and John Havlicek

Executive Summary

Property verification and synthesis tools work with automata to represent both the model to examine and the property to check. Subject of this work is to show the steps necessary for creation of automata for properties given in PSL [2].

Automata construction for the temporal logic LTL is a well known problem [11]. We extend this to automata for the core of PSL, denoted here LTL_{WR}. The extensions over LTL take two directions: First, the logic is interpreted over finite, possibly truncated as well as infinite words. In another direction, the new basic formulas for weak and strong regular expressions and operators to combine them with other formulas are added to the language. The regular expressions used in Sugar extend the usual one by the intersection operator (“length-matching sequence conjunction”).

We show that for every LTL_{WR} formula ϕ there exists a Büchi automaton whose size is doubly exponential in the size of ϕ . We discuss how the suggested constructions can be used in the process of model checking PSL properties using the automata-theoretic approach. We further show an efficient algorithm to derive a symbolic representation for a subset of the language.

The work presented here is easily extended to the full foundation language of PSL, which extends LTL_{WR} by a set of abbreviations.

The technical material presented in this deliverable extends the work presented in [9]. Other research performed in the course of this deliverable is presented in [25] and [6].

Purpose

This document describes research of the PROSYD project into efficient construction algorithms for PSL. It forms the basis on which PSL translation software will be written. The translation from PSL to automata is a basic technique needed for most algorithms based on PSL.

Intended Audience

This guide is intended for researchers working with automata-theory related approaches like model checking and PSL, who intend to implement a translator from PSL to automata. It is assumed that readers are familiar with the notions and terms related to PSL and have a basic understanding of automata-theory.

Background

A construction for a logic extending LTL with suffix conjunction/implication operators, interpreted over infinite words appears in [16]. A construction for reasoning over truncated words, via reset operators appears in [3]. The contribution of this document in respect of automata construction is threefold.

1. We are the first to show a construction for weak regular expressions.
2. We are the first to show a complete construction for the PSL core denoted `LTL_WR` synthesizing previous results into a cohesive whole.
3. We show an improved subset algorithm for a class of properties, and show how it can be used to construct a symbolic representation directly.

Contents

Table of Revisions	iii
Authors	iii
Executive Summary	iv
Purpose	iv
Intended Audience.....	iv
Background	v
Contents	vii
Table of Figures	viii
Glossary	ix
1 Introduction	1
2 Basics and Preliminaries	3
Notation.....	3
PSL Core Logic	3
Syntax	4
Semantics.....	4
Non-deterministic Automata.....	6
Alternating Automata	7
3 Constructions and Proofs.....	11
Automata construction: LTL over finite and infinite words	11
Automata Construction for LTL_WR	14
Classification of Automata for RE-based LTL_WR-formulas	25
4 Translating Alternating into Nondeterministic Automata	31
Removing Universality	32
Removing Universality from Terminal Alternating Automata	33
Removing Universality from One-Weak Automata.....	36
Removing Universality from Alternating Automata.....	37
Symbolic Implementation	39
5 Conclusions	43
6 References.....	45

Table of Figures

Figure 1 - Alternating automaton B_{SC1}	34
Figure 2 - Subset Construction on B_{SC1}	35
Figure 3 - Weak alternating automaton B_{SC2}	35
Figure 4 - Subset Construction on B_{SC2}	35

Glossary

AFA, Alternating Finite Automaton

We define these automata as a special subclass of alternating Büchi automata (See Definition 6 (p. 7)).

Alternation

Alternating automata combine the features of existential and universal choice (see p. 7).

BDD, Binary Decision Diagram

(reduced ordered) Binary decision diagrams (BDDs, for short) are a special data structure for characteristic descriptions of state sets, which many model-checkers use for enhanced efficiency. Specific information can be found in [8].

CTL, Computation Tree Logic

Computation tree logic is a temporal logic for property specification in formal verification.

DAG

Acronym for directed acyclic graph.

EDA

Acronym for electronic design automation.

FILO, First In Last Out

A strategy to process the elements in a queue. The first element stored in the queue is the last one to be processed.

IEEE, Institute of Electrical and Electronics Engineers

The IEEE is a non-profit, technical professional association of more than 360,000 individual members in approximately 175 countries.

Kleene Closure

The Kleene closure for regular expressions is an unary operator on sets of or single syntactical elements of a language. S^* describes the set of words built by concatenating zero or more (finite) elements of S .

Language Emptiness

Language emptiness of an automaton equals to the fact that there is no accepting run for this automaton.

LTL, Linear Temporal Logic, Linear-Time Temporal Logic

Linear temporal logic is a temporal logic for property specification in formal verification [23].

LTL_{WR}

The core logic of PSL, which is an extension of the linear temporal logic LTL. Refer to section 2 (p. 3) for a detailed specification.

NBA, Non-deterministic Büchi Automaton

See Definition 5 (p. 6). Note that our definition might differ from other ones, as we define the automaton for both finite and infinite words.

NFA, Non-deterministic Finite Automaton

We define these automata as a special subset of NBAs (See Definition 5 (p. 6)).

Non-determinism

Non-determinism or existential choice for automata describes the property to choose one of several possible successors, defining a different run for every possible successor (see p. 6).

One-weak

One-weak automata are a special subset of weak automata (See Definition 7 (p. 8)).

Property

A collection of logical and temporal relationships between and among subordinate Boolean expressions, regular expressions, and other properties that in aggregate represent a set of behaviors.

PSL

Property specification language. This is the specification language used throughout and building the basis for the PROSYD project. Refer to the Language Reference Manual [1] for more details on the language.

Regular Expression

A regular expression is a pattern or expression that describes a set of finite words.

Tableau

In model-checking the automaton built from the negated property is often referred to as tableau.

Terminal

Terminal automata are a special subset of weak automata (See Definition 7 (p. 8)).

Universalism, Universality

Universalism or universal choice for automata describes the property that all possible successors have to be chosen simultaneously, splitting a run into several branches.

Weak, Weakness

The term "weak" is used in this document in two different contexts:

- Weak automata represent a special subclass of automata offering distinctive features. Refer to Definition 7 (p. 8) for more information.
- Weak operators or regular expressions in PSL (see [13]).

1 Introduction

The language PSL [2] is a temporal logic standardized by the Accellera standards organization and currently undergoing the process of becoming an IEEE standard. The core of PSL, denoted here `LTL_WR`, is an extension of the linear temporal logic LTL. The extension takes two orthogonal directions. In one direction the logic is interpreted over finite, possibly truncated, as well as infinite words. Truncated words are words that are finite, but not necessarily maximal. Reasoning over truncated words (as well as maximal words) is important for incomplete verification methods such as simulation and bounded model checking, as well as for supporting abort/reset operators [14]. In another direction, new basic formulas and operators are added to the language. The new basic formulas are weak and strong regular expressions [13], and the new operators are suffix conjunction/implication that combine regular expressions with other formulas. The regular expressions used in PSL extend the usual regular expressions by the intersection operator (“length-matching sequence conjunction”).

In this document we provide automata construction for `LTL_WR`. Nondeterministic Büchi automata are at the basis of many algorithms based on PSL: they are needed in verification and synthesis and in the requirements engineering techniques proposed in PROSYD. We show that for every `LTL_WR` formula φ there exists a Büchi automaton whose size is doubly exponential in the size of φ . One exponential blowup is needed to translate LTL to automata, and is sufficient to translate all of PSL with the exception of intersection in the regular expressions. The second exponential blowup is needed for the intersection operator.

The construction shown here is the first full construction for the PSL core language. In particular, it shows how to handle weak regular expressions, which has not been done before.

Our construction is through weak alternating Büchi automata on finite and infinite words. We show that Miyano and Hayashi’s classical algorithm to convert alternating to non-deterministic automata can be extended to Büchi automata on finite and infinite words. Then, we show that for the frequent class of one-weak alternating automata we can construct a smaller automaton than the one produced by Miyano and Hayashi’s method. In this case, the non-deterministic automaton has $n \cdot 2^n$ rather than 2^{2n} states (where n is the number of states of the alternating automaton). We also show how, in this case, the alternating automaton can easily be converted to a symbolic form, without passing through a nondeterministic automaton and suffering an expensive re-encoding step.

Since the foundation language of PSL consists of the core language and a set of abbreviations, the approach presented here is therefore easily extended to the full foundation language.

A construction for a logic extending LTL with suffix conjunction/implication operators, interpreted over infinite words appears in [16]. A construction for reasoning

over truncated words, via reset operators appears in [3].

The document is organized as follows. In the following section we provide the basics of the document regarding notation, automata-theory, subset construction as well as a definition of LTL_{WR} . Section three covers the proposed automaton construction, whose integration into existing workflows is considered in section four, which also provides a special algorithm for symbolic representation retrieval for a subset of LTL_{WR} . Section five concludes research results.

2 Basics and Preliminaries

This section covers the definitions used in this document. It starts with an introduction on notation and the definition of the PSL core logic LTL_WR addressed by this document. The necessary definitions on automata theory are introduced, including a classification of Büchi automata into weak, one-weak and terminal automata.

Notation

We denote a letter from a given alphabet Σ by ℓ and an empty, finite, or infinite word from Σ by u , v , or w (possibly with subscripts). The *concatenation* of u and v is denoted by uv . If u is infinite, then $uv = u$. The empty word is denoted by ε , so that $w\varepsilon = \varepsilon w = w$. If $w = uv$ we say that u is a *prefix* of w , denoted $u \preceq w$, that v is a *suffix* of w , and that w is an *extension* of u , denoted $w \succeq u$.

We denote the *length* of word v as $|v|$. The empty word ε has length 0, a finite word $v = (\ell_0\ell_1\ell_2\cdots\ell_n)$ has length $n + 1$, and an infinite word has length ω . We use i , j , and k to denote non-negative integers. For $i < |v|$ we use v^i to denote the $(i + 1)^{st}$ letter of v (since counting of letters starts at zero), and we denote by $v^{i\cdots}$ the suffix of v starting at v^i . When $i \leq j \leq |v|$, we denote by $v^{i\cdots j}$ the finite sequence of letters starting from v^i and ending in v^j . That is, $v^{i\cdots j} = v^i v^{i+1} \dots v^j$.

We denote by ℓ^k the word of length k , each letter of which is ℓ , and by ℓ^ω the infinite-length word, each letter of which is ℓ . We use Σ^* to denote the set of all finite words over Σ , and Σ^ω to denote the set of all infinite words over Σ . We use Σ^∞ to denote the set of all finite and infinite words over Σ .

An *unlabeled tree* is a prefix-closed subset of \mathbb{N}^* . Elements of the tree are referred to as *nodes*. For a node $t \in \mathbb{N}^*$ we refer to $|t|$ as the *depth* of t . The node ε is called the *root*. For nodes t_1 and t_2 such that $t_1 \prec t_2$ we say that t_2 is a *descendant* of t_1 . If t_2 is a descendant of t_1 and $|t_2| = |t_1| + 1$ we say that t_2 is a *child* of t_1 . A Σ -*labeled tree* τ is a pair $\langle T, \tau \rangle$ where T is an unlabeled tree and $\tau : T \rightarrow \Sigma$ maps nodes of T to symbols in Σ .

PSL Core Logic

Below we provide a formal definition of the PSL core logic LTL_WR , an extension of LTL with weak and strong regular expressions and a suffix conjunc-

tion/implication operator, interpreted over finite (possibly truncated) as well as infinite words.

Syntax

The logic LTL_{WR} is defined over boolean expressions as well as regular expressions. We assume a given set \mathbb{B} of boolean expressions defined over a set \mathbb{P} of atomic propositions. Below we define Regular Expressions (REs) using \cdot , \cup , \cap and $*$ for concatenation, union, intersection and Kleene-closure, respectively.

Definition 1 (REs).

- Every boolean expression $b \in \mathbb{B}$ is an RE.
- If r , r_1 and r_2 are REs, then the following are REs:
 - $r_1 \cdot r_2$ • $r_1 \cup r_2$ • $r_1 \cap r_2$ • r^*

Definition 2 (LTL_{WR} formulas).

- If b is a boolean expression then $b!$ is an LTL_{WR} formula.
- If φ and ψ are LTL_{WR} formulas, and r is an RE, then the following are LTL_{WR} formulas:
 - $\neg\varphi$ • $\varphi \wedge \psi$ • $X!\varphi$ • $[\varphi U \psi]$ • $r \diamondrightarrow \varphi$ • r

Additional operators are defined as syntactic sugaring of the above operators:

- $\varphi \vee \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \wedge \neg\psi)$ • $[\varphi W \psi] \stackrel{\text{def}}{=} \neg[\neg\psi U (\neg\varphi \wedge \neg\psi)]$ • $X\varphi \stackrel{\text{def}}{=} \neg(X!\neg\varphi)$
- $r! \stackrel{\text{def}}{=} r \diamondrightarrow \text{true}$ • $r \mapsto \varphi \stackrel{\text{def}}{=} \neg(r \diamondrightarrow \neg\varphi)$ • $b \stackrel{\text{def}}{=} \neg(\neg b)!$

We refer to the operators $r!$ and r as *strong* and *weak* regular expressions, respectively. We refer to the \diamondrightarrow operator as the *suffix conjunction operator*, since $r \diamondrightarrow \varphi$ (read r “suffix-and” φ) demands that there exist a non-empty prefix of the path tightly satisfying r and that the suffix starting at the last letter of the prefix satisfy φ . We refer to its dual, the \mapsto operator, as the *suffix implication operator*, since $r \mapsto \varphi$ (read r “suffix-implies” φ) requires that *if* there exists a non-empty prefix of the path tightly satisfying r *then* the suffix starting at the last letter of the prefix should satisfy φ . The operator \diamondrightarrow exists in the temporal logic ForSpec [4] under the names *follows_by* and *seq*. The operator \mapsto exists in the temporal logic Sugar [5] under the notation $\{r\}(\varphi)$. These operators are essentially the “diamond” and “box” modalities of PDL [15], respectively.

Semantics

The semantics of LTL_{WR} is defined with respect to a non-empty set of atomic propositions \mathbb{P} , a set of boolean expressions \mathbb{B} over \mathbb{P} , which we identify with $2^{2^{\mathbb{P}}}$, and the set of regular expressions over \mathbb{B} .

Although we are interested in the set of words over $2^{\mathbb{P}}$ which satisfy a given LTL_WR formula, it is convenient to define the semantics of LTL_WR over an enhanced set of words. Let Σ denote the alphabet $2^{\mathbb{P}} \cup \{\top, \perp\}$. The semantics of LTL_WR is defined with respect to words over Σ . Note, however, that since computations of systems are words over $2^{\mathbb{P}}$, our interest is still focused on such words. We refer to words over $2^{\mathbb{P}}$ as *natural* words. We use \bar{w} to denote the word obtained by replacing every \top with a \perp and vice versa. We call \bar{w} the *dual* of w . For a set of words W we use \bar{W} to denote the set $\{\bar{w} \mid w \in W\}$.

The semantics of LTL_WR is defined inductively with respect to finite/infinite (possibly empty) words over Σ , using as the base case the semantics of boolean expressions over letters in Σ and of regular expressions over finite words over Σ . To denote satisfaction we introduce several operators defined in the following:

- \models_{PSL} : Boolean Satisfaction
- \vDash_{PSL} : RE Tight Satisfaction
- \vDash_{PSL} : Formula Satisfaction

For a boolean expression $b \in \mathbb{B}$ and a letter $\ell \in \Sigma$ we define the boolean satisfaction relation \models_{PSL} as follows. Let $b \in \mathbb{B}$. For $\ell \in 2^{\mathbb{P}}$, we define $\ell \models_{\text{PSL}} b \iff \ell \in b$. We define $\top \models_{\text{PSL}} b$ and $\perp \not\models_{\text{PSL}} b$. Note that in particular we get $\top \models_{\text{PSL}} \text{false}$ and $\perp \not\models_{\text{PSL}} \text{true}$.

Definition 3 (RE Tight Satisfaction). Let v denote a finite (possibly empty) word over Σ ; b denote a boolean expression; and r, r_1 , and r_2 denote REs. The notation $v \vDash_{\text{PSL}} r$ means that v tightly satisfies r . The relation \vDash_{PSL} is defined as follows:

- $v \vDash_{\text{PSL}} b \iff |v| = 1 \text{ and } v^0 \models_{\text{PSL}} b$
- $v \vDash_{\text{PSL}} r_1 \cdot r_2 \iff \exists v_1, v_2 \text{ s.t. } v = v_1 v_2 \text{ and } v_1 \vDash_{\text{PSL}} r_1 \text{ and } v_2 \vDash_{\text{PSL}} r_2$
- $v \vDash_{\text{PSL}} r_1 \cup r_2 \iff v \vDash_{\text{PSL}} r_1 \text{ or } v \vDash_{\text{PSL}} r_2$
- $v \vDash_{\text{PSL}} r_1 \cap r_2 \iff v \vDash_{\text{PSL}} r_1 \text{ and } v \vDash_{\text{PSL}} r_2$
- $v \vDash_{\text{PSL}} r^* \iff \text{either } v = \varepsilon \text{ or } \exists v_1, v_2 \text{ s.t. } v_1 \neq \varepsilon, v = v_1 v_2, v_1 \vDash_{\text{PSL}} r \text{ and } v_2 \vDash_{\text{PSL}} r^*$

Definition 4 (Formula Satisfaction). Let v denote a word over Σ ; b a boolean expression; r an RE; and φ and ψ LTL_WR formulas. The notation $v \models_{\text{PSL}} \varphi$ means that v satisfies φ . The relation \models_{PSL} is defined as follows:¹

1. $v \models_{\text{PSL}} b! \iff |v| > 0 \text{ and } v^0 \models_{\text{PSL}} b$
2. $v \models_{\text{PSL}} \neg \varphi \iff \bar{v} \not\models_{\text{PSL}} \varphi$
3. $v \models_{\text{PSL}} \varphi \wedge \psi \iff v \models_{\text{PSL}} \varphi \text{ and } v \models_{\text{PSL}} \psi$
4. $v \models_{\text{PSL}} X! \varphi \iff |v| > 1 \text{ and } v^{1..} \models_{\text{PSL}} \varphi$
5. $v \models_{\text{PSL}} [\varphi U \psi] \iff \exists 0 \leq k < |v| \text{ s.t. } v^{k..} \models_{\text{PSL}} \psi \text{ and } \forall 0 \leq j < k, v^{j..} \models_{\text{PSL}} \varphi$
6. $v \models_{\text{PSL}} r \diamond \rightarrow \psi \iff \exists 0 \leq j < |v| \text{ s.t. } v^{0..j} \vDash_{\text{PSL}} r \text{ and } v^{j..} \models_{\text{PSL}} \psi$

¹The semantics of the LTL operators are the standard ones. The operators of LTL are all the operators in definition 2 except for $\diamond \rightarrow$ and r .

$$7. v \models_{\text{PSL}} r \iff \forall \text{finite } u \preceq v, u \top^\omega \models_{\text{PSL}} (r \diamond \rightarrow \text{true})$$

For an LTL_{WR} formula ϕ , we use $\llbracket \phi \rrbracket$ to denote the set $\{w \in \Sigma^\infty \mid w \models_{\text{PSL}} \phi\}$.

Between *strong* and *weak* regular expressions there is a significant difference. Strong operators require that the described sequence is exactly fulfilled, which means that for a strong regular expression r_1 and a word v there has to exist a $j < |v|$ such that $v^{0..j} \models_{\text{PSL}} r_1$. In contrary a weak regular expression r_2 requires for accepting an input word v only that it is not contradicting the described sequence, thus accepts truncated words which can be a prefix for \top^ω to create a word satisfying r_2 : $v \models_{\text{PSL}} r_2$ iff for all $j < |v|$: $v^{0..j} \top^\omega \models_{\text{PSL}} r_2$. The following examples illustrate the difference:

- $a^\omega \models_{\text{PSL}} a^*b$
- $a^\omega \not\models_{\text{PSL}} a^*b!$

Non-deterministic Automata

Non-deterministic automata feature existential branching, which offers the choice among several possible successor states. This leads to multiple runs for each word, and a word is accepted if it features at least one accepting run.

Non-deterministic automata may be exponentially smaller than their deterministic counterparts, and they can be handled directly by the search strategies used in model checkers. The nondeterministic automaton may also be more intuitive than a deterministic one.

An automaton on finite words accepts an input word if there is a run for this word, that ends in a final state. For infinite words, a different acceptance condition is required, as infinite words have infinite runs. We will use Büchi acceptance condition. A non-deterministic Büchi automaton accepts an infinite input word, if it features a run that passes accepting states infinitely often. To be able to cope with both finite and infinite words we use two separate acceptance conditions, one for finite and one for infinite runs.

Definition 5 (Büchi automaton (NBA)). A non-deterministic Büchi automaton over finite/infinite words may be described as a tuple $B = (\Sigma, S, I, \rho, F, A)$ where

- Σ is a finite nonempty alphabet
- S is a finite nonempty set of states
- $I \subseteq S$ is a nonempty set of initial states
- $\rho : S \times \Sigma \rightarrow 2^S$ is a transition function
- $F \subseteq S$ is a set of final states
- $A \subseteq S$ is a set of accepting states

We often regard ρ as a relation $\rho \subseteq S \times \Sigma \times S$. A *run* σ of B on an infinite word $w = \ell_0 \ell_1 \dots$ is an infinite sequence s_0, s_1, \dots where $s_0 \in I$ and $s_{i+1} \in \rho(s_i, \ell_i)$ for all $i \geq 0$. A *run* σ of B on a finite word $w = \ell_0 \ell_1 \dots \ell_n$ is a finite sequence s_0, s_1, \dots, s_{n+1} where $s_0 \in I$ and either $w = \varepsilon$ or $s_{i+1} \in \rho(s_i, \ell_i)$ for all $0 \leq i \leq n$. We define by $\text{lim}(\sigma)$ the set $\{s \mid s = s_i \text{ for infinitely many } i\}$. An infinite run is *accepting* if there is some accepting state that repeats infinitely often, i.e. if $\text{lim}(\sigma) \cap A \neq \emptyset$. A finite run is *accepting* if the last state is a final state, i.e. if $s_{|w|} \in F$. The word w is *accepted* by B if there is an accepting run of B on w . We denote by $\mathcal{L}(B)$ the set of words w such that there exists an accepting run of B on w . For $s \in S$ we use B^s to denote the automaton $(\Sigma, S, \{s\}, \rho, F, A)$.

Non-deterministic finite automata (NFA) represent a special subset of non-deterministic Büchi automata, where the accepting set A is empty, and thus acceptance is only determined by the set of final states F .

The more traditional definition of a non-deterministic Büchi automaton, where the automaton is defined only for infinite input words, is a special case of our NBA where the set of final states F is empty, and thus acceptance is only determined by the accepting set A .

A *reduced NFA* is an NFA which does not contain any states which do not feature paths to final states. These states do not feature accepted behaviors and thus can be removed from the automaton without changing its language.

Alternating Automata

Alternating automata combine semantic elements of both universalism and non-determinism by offering both universal and existential branching [10]. Nondeterminism, as usual, results in multiple runs. Universalism results in branches of the run, which thus becomes a tree. In order for a word to be accepted, there has to be a run for the word such that all branches in the run are accepting.

Definition 6 (Alternating Büchi automaton (ABA)). *An alternating Büchi automaton over finite/infinite words is a tuple $B = (\Sigma, S, \{s_0\}, \rho, F, A)$ where*

- Σ is a finite nonempty alphabet
- S is a finite nonempty set of states
- $s^0 \in S$ is the initial state.
- $\rho : S \times \Sigma \rightarrow \mathcal{B}^+(S)$ is a transition function where $\mathcal{B}^+(S)$ is the set of boolean formulas obtained by application of \wedge and \vee to element in S .²
- $F \subseteq S$ is a set of final states
- $A \subseteq S$ is a set of accepting states

²We assume true and false are in $\mathcal{B}^+(S)$.

A *run* of B on an infinite word $w = \ell_0\ell_1\dots$ is a (possibly infinite) S -labeled tree $\tau = \langle T, \mathfrak{t} \rangle$ such that $\mathfrak{t}(\epsilon) = s^0$ and for every node $t \in T$, t has at most $|S|$ children and, if $|t| = i$ and $\mathfrak{t}(t) = s$, then the children of t satisfy $\rho(s, \ell_i)$ (i.e. if t_1, \dots, t_k are t 's children then $\{\mathfrak{t}(t_1), \dots, \mathfrak{t}(t_k)\}$ satisfies $\rho(s, \ell_i)$). If w is infinite, then a run tree τ is *accepting* if every branch of infinite depth has infinitely many labels in A . If w is finite, then a run tree τ is *accepting* if all nodes at depth $|w|$ are labeled by states in F . The word w is *accepted* by B if there is an accepting run tree of B on w . We denote by $\mathcal{L}(B)$ the set of words w such that there exists an accepting run tree of B on w .

Alternating finite automata (AFA) represent a special subclass of alternating Büchi automata, where the accepting set $A \subseteq S$ is empty, and thus acceptance is only determined by the set of final states $F \subseteq S$.

Note also that non-deterministic Büchi automata are a subset of the alternating automata.

As a measure of efficiency we use the classification of Büchi automata as suggested by Muller et al. [22] and adopted by [7]. The classification uses the notion of *weak*, *one-weak*, and *terminal* Büchi automata.³

Definition 7. [*weak, one weak, terminal (see e.g [7])*] Let $B = (\Sigma, S, I, \rho, F, A)$ be an alternating Büchi automaton on finite and infinite words.

- B is a *weak* Büchi automaton on finite and infinite words if there exists a partition of the set of states S into components S_i and a partial ordering \leq on these sets, such that
 1. for each $s_i \in S_i$, $s_j \in S_j$, if $\exists \ell \in \Sigma$ s.t. $s_j \in \rho(s_i, \ell)$ then $S_j \leq S_i$, and
 2. for each S_i , either $S_i \cap A = \emptyset$, in which case S_i is a *rejecting component*, or $S_i \subseteq A$, in which case S_i is an *accepting component*.
- A *weak* Büchi automaton is *one-weak* if the partition can be chosen in such a way that all rejecting components have size one.
- B is a *terminal* Büchi automaton on finite and infinite words if it is a *weak* Büchi automaton such that the components of the partition contained in A are minimal elements of the partial order. Furthermore for every state $s \in A$ and every letter $\sigma \in \Sigma$ we have that $\rho(s, \sigma) \neq \text{false}$.

For weak automata the state space is partitioned into disjoint components. These components underly a partial order such that there is no edge from a state in component A to a state in component B if $A < B$. The acceptance conditions are based on these subsets, such that in every subset either all states are accepting or none are. Because the state space is finite, every branch of a run of the automaton gets trapped in one of the partitioned subsets at some point, and acceptance is decided according to this subset's classification. One-weak automata are a special subclass of weak automata in which the rejecting components are of size one. Weak alternating automata are as expressive as alternating Büchi automata define, but non-deterministic weak automata are less expressive. Terminal automata are automata

³Note that the term *weak* used for a Büchi automaton has nothing to do with the term *weak* used for REs. We apologize for the potential confusion.

in which the accepting states are universally accepting. They actually define the finitary languages described by co-safety properties.

Note that our definition of one-weak is weaker than the usual definition. (The usual definition requires that accepting components also have size one.) The current definition allows us to apply Proposition 4.3 to a larger class of automata.

In the non-deterministic case, weak and terminal automata are easier to check than general automata.

Claim 2.1 (see e.g [7]). *Let $B = (\Sigma, S, I, \rho, F, A)$ be a non-deterministic Büchi automaton on finite and infinite words.*

- *The language of a weak Büchi automaton on finite and infinite words is empty iff $AG AF \neg A$*
- *The language of a terminal Büchi automaton on finite and infinite words is empty iff $AG \neg A$*

3 Constructions and Proofs

In this section we show how to construct efficient automata for LTL_{WR}. The construction to translate the PSL core works for both finite and infinite words.

First, we consider automata for linear temporal logic interpreted over finite and infinite words, and then we generalize the approach to an efficient algorithm supporting LTL_{WR}. Finally, we prove that the derived automata are weak.

Automata construction: LTL over finite and infinite words

Since we are interested in finite as well as infinite words, we check first that the well-known singly-exponential complexity for constructing a Büchi automaton for the ω -language of an LTL formula still applies when the automaton runs on both finite and infinite words.

We follow the construction of an alternating Büchi automaton for an LTL formula φ as given in [28, Theorem 22]. Our construction differs from that of [28] in several ways.

1. Our automaton is designed to run on both finite and infinite words, while that of [28] is only for infinite words. Some care is needed to handle the nexttime operators on finite words since the semantics looks ahead in order to know whether the condition $|w| > 1$ is satisfied. Our construction uses a simple subautomaton to distinguish nonempty from empty words.
2. The construction of [28] introduces an automaton state for each subformula and its negation and relies on a dualization operator in the definition of the transition relation. We achieve a similar effect by including only subformulas and using a negation parity bit in the state. The negation parity of a subformula indicates the number of enclosing negations, modulo 2. Our construction gives the dual forms of the transition relation explicitly rather than relying on the dualizing operator.
3. Our construction works over the extended alphabet $\Sigma = 2^{\mathbb{P}} \cup \{\top, \perp\}$. This is accomplished simply by using the dual $\bar{\ell}$ of the input letter ℓ when testing boolean satisfaction for the transition relation on a boolean subformula with negation parity 1.

Proposition 3.1. *Given an LTL formula φ , one can build an alternating Büchi automaton $B_\varphi = (\Sigma, S, I, \rho, F, A)$ where $|S|$ is in $O(|\varphi|)$ and $\mathcal{L}(B_\varphi)$ is exactly the set of (finite and infinite) words satisfying the formula φ .*

Proof. Let ϕ be an LTL formula. Let S be the set of pairs (p, ψ) where either ψ is a subformula of ϕ or $\psi \in \{\text{TRUE}, \text{FALSE}, \text{NONEMPTY}\}$, and $p \in \{0, 1\}$ denotes the negation parity, with $p = 1$ indicating an odd number of enclosing negations. The initial state is $(0, \phi)$.

The set A of accepting states is produced by the following rules:

1. A contains all states of the form:
 $(1, \text{NONEMPTY}), (0, \text{TRUE}), (1, \text{FALSE}), (1, b!), (1, X!\psi), (1, [\psi U \vartheta])$.
2. $(p, \neg\psi) \in A$ iff $(1 - p, \psi) \in A$.
3. $(0, \psi \wedge \vartheta) \in A$ iff both $(0, \psi) \in A$ and $(0, \vartheta) \in A$.
 $(1, \psi \wedge \vartheta) \in A$ iff either $(1, \psi) \in A$ or $(1, \vartheta) \in A$.

The set F of final states is equal to A .

The transition relation ρ is defined as follows:

- $\rho((p, \text{NONEMPTY}), \ell) = (p, \text{TRUE})$
- $\rho((p, \text{TRUE}), \ell) = (p, \text{TRUE})$
 $\rho((p, \text{FALSE}), \ell) = (p, \text{FALSE})$
- $\rho((0, b!), \ell) = (0, \ell \models_{\text{PSL}} b)$
 $\rho((1, b!), \ell) = (1, \bar{\ell} \models_{\text{PSL}} b)$
- $\rho((0, \psi \wedge \vartheta), \ell) = \rho((0, \psi), \ell) \wedge \rho((0, \vartheta), \ell)$
 $\rho((1, \psi \wedge \vartheta), \ell) = \rho((1, \psi), \ell) \vee \rho((1, \vartheta), \ell)$
- $\rho((0, \neg\psi), \ell) = \rho((1, \psi), \ell)$
 $\rho((1, \neg\psi), \ell) = \rho((0, \psi), \ell)$
- $\rho((0, X!\psi), \ell) = (0, \text{NONEMPTY}) \wedge (0, \psi)$
 $\rho((1, X!\psi), \ell) = (1, \text{NONEMPTY}) \vee (1, \psi)$
- $\rho((0, [\psi U \vartheta]), \ell) = \rho((0, \vartheta), \ell) \vee (\rho((0, \psi), \ell) \wedge (0, [\psi U \vartheta]))$
 $\rho((1, [\psi U \vartheta]), \ell) = \rho((1, \vartheta), \ell) \wedge (\rho((1, \psi), \ell) \vee (1, [\psi U \vartheta]))$

Lemma 3.2. *If ψ is a subformula of ϕ , let $A^{(p, \psi)}$ denote the subautomaton of this construction obtained by taking (p, ψ) as initial state.*

1. $\mathcal{L}(A^{(0, \psi)}) = \llbracket \psi \rrbracket$.
2. $\mathcal{L}(A^{(1, \psi)}) = \Sigma^\infty - \overline{\llbracket \psi \rrbracket} = \llbracket \neg\psi \rrbracket$.

Proof. By induction over the subformulas of ϕ . Note that for any $L \subseteq \Sigma^\infty$,

$$\Sigma^\infty - \bar{L} = \overline{\Sigma^\infty - L}$$

Note also that for any ψ ,

$$\Sigma^\infty - \overline{\llbracket \psi \rrbracket} = \llbracket \neg\psi \rrbracket$$

Let v, w denote elements of Σ^∞ .

- $\mathcal{L}(A^{(0, \text{NONEMPTY})}) = \{v : |v| > 0\}$.
- $\mathcal{L}(A^{(1, \text{NONEMPTY})}) = \{\epsilon\}$.

$$\begin{aligned}
& - \mathcal{L}(A^{(0,b!)}) \\
& \quad = [(0, b!) \text{ is not accepting}] \\
& \quad \quad \{\ell w : \ell \models_{\text{PSL}} b\} \\
& \quad = \llbracket b! \rrbracket \\
& \mathcal{L}(A^{(1,b!)}) \\
& \quad = [(1, b!) \text{ is accepting}] \\
& \quad \quad \{\varepsilon\} \cup \{\ell w : \bar{\ell} \not\models_{\text{PSL}} b\} \\
& \quad = \Sigma^\infty - \overline{\llbracket b! \rrbracket} \\
& - \llbracket \neg \psi \rrbracket \\
& \quad = \Sigma^\infty - \overline{\llbracket \psi \rrbracket} \\
& \quad = [\text{induction}] \\
& \quad \quad \mathcal{L}(A^{(1,\psi)}) \\
& \quad = \mathcal{L}(A^{(0,\neg\psi)}) \\
& \mathcal{L}(A^{(1,\neg\psi)}) \\
& \quad = \mathcal{L}(A^{(0,\psi)}) \\
& \quad = [\text{induction}] \\
& \quad \quad \llbracket \psi \rrbracket \\
& \quad = \Sigma^\infty - \overline{\llbracket \neg \psi \rrbracket} \\
& - \llbracket \psi \wedge \vartheta \rrbracket \\
& \quad = \llbracket \psi \rrbracket \cap \llbracket \vartheta \rrbracket \\
& \quad = [\text{induction}] \\
& \quad \quad \mathcal{L}(A^{(0,\psi)}) \cap \mathcal{L}(A^{(0,\vartheta)}) \\
& \quad = \mathcal{L}(A^{(0,\psi \wedge \vartheta)}) \\
& \mathcal{L}(A^{(1,\psi \wedge \vartheta)}) \\
& \quad = \mathcal{L}(A^{(1,\psi)}) \cup \mathcal{L}(A^{(1,\vartheta)}) \\
& \quad = [\text{induction}] \\
& \quad \quad (\Sigma^\infty - \overline{\llbracket \psi \rrbracket}) \cup (\Sigma^\infty - \overline{\llbracket \vartheta \rrbracket}) \\
& \quad = \Sigma^\infty - (\overline{\llbracket \psi \rrbracket} \cap \overline{\llbracket \vartheta \rrbracket}) \\
& \quad = \Sigma^\infty - (\overline{\llbracket \psi \wedge \vartheta \rrbracket}) \\
& - \mathcal{L}(A^{(0,X!\psi)}) \\
& \quad = [(0, X!\psi) \text{ is not accepting; } \mathcal{L}(A^{(0,\text{NONEMPTY})}) = \{v : |v| > 0\}] \\
& \quad \quad \{\ell w : |w| > 0 \text{ and } w \in \mathcal{L}(A^{(0,\psi)})\} \\
& \quad = [\text{induction}] \\
& \quad \quad \{\ell w : |w| > 0 \text{ and } w \in \llbracket \psi \rrbracket\} \\
& \quad = \{v : |v| > 1 \text{ and } v^{1..} \models_{\text{PSL}} \psi\} \\
& \quad = \llbracket X!\psi \rrbracket \\
& \mathcal{L}(A^{(1,X!\psi)}) \\
& \quad = [(1, X!\psi) \text{ is accepting; } \mathcal{L}(A^{(1,\text{NONEMPTY})}) = \{\varepsilon\}] \\
& \quad \quad \{v : |v| \leq 1 \text{ or } v^{1..} \in \mathcal{L}(A^{(1,\psi)})\} \\
& \quad = [\text{induction}] \\
& \quad \quad \{v : |v| \leq 1 \text{ or } v^{1..} \in \llbracket \neg \psi \rrbracket\} \\
& \quad = \{v : |v| \leq 1 \text{ or } \bar{v}^{1..} \not\models_{\text{PSL}} \psi\} \\
& \quad = \Sigma^\infty - \overline{\{v : |v| > 1 \text{ and } v^{1..} \models_{\text{PSL}} \psi\}}
\end{aligned}$$

$$= \Sigma^\infty - \overline{[\chi \upharpoonright \psi]}$$

– Let $v \in \mathcal{L}(A^{(0, [\psi \cup \vartheta])})$. Suppose $v \notin \mathcal{L}(A^{(0, \vartheta)})$. Then $v \in \mathcal{L}(A^{(0, \psi)})$ and, since $(0, [\psi \cup \vartheta])$ is not accepting, $|v| > 0$ and $v^{1..} \in \mathcal{L}(A^{(0, [\psi \cup \vartheta])})$. Thus, $v \in \mathcal{L}(A^{(0, [\psi \cup \vartheta])})$

$$\iff v \in \mathcal{L}(A^{(0, \vartheta)}) \text{ or } (v \in \mathcal{L}(A^{(0, \psi)}) \text{ and } |v| > 0 \text{ and } v^{1..} \in \mathcal{L}(A^{(0, [\psi \cup \vartheta])}))$$

$$\iff \text{[induction]}$$

$$v \models_{\text{PSL}} \vartheta \text{ or } (v \models_{\text{PSL}} \psi \text{ and } |v| > 0 \text{ and } v^{1..} \in \mathcal{L}(A^{(0, [\psi \cup \vartheta])}))$$

If there does not exist $0 \leq k < |v|$ such that $v^{k..} \models_{\text{PSL}} \vartheta$, then, since $(0, [\psi \cup \vartheta])$ is not accepting, there is no accepting run of $A^{(0, [\psi \cup \vartheta])}$ on v . Therefore,

$$v \in \mathcal{L}(A^{(0, [\psi \cup \vartheta])})$$

$$\iff \text{there exists } 0 \leq k < |v| \text{ such that } v^{k..} \models_{\text{PSL}} \vartheta \text{ and for all } 0 \leq j < k,$$

$$v^{j..} \models_{\text{PSL}} \psi$$

$$\iff v \models_{\text{PSL}} [\psi \cup \vartheta]$$

$$v \in \mathcal{L}(A^{(1, [\psi \cup \vartheta])})$$

$$\iff v \in \mathcal{L}(A^{(1, \vartheta)}) \text{ and } (v \in \mathcal{L}(A^{(1, \psi)}) \text{ or } v \in \mathcal{L}(A^{(1, [\psi \cup \vartheta])}))$$

$$\iff \text{[induction]}$$

$$v \models_{\text{PSL}} \neg \vartheta \text{ and } (v \models_{\text{PSL}} \neg \psi \text{ or } v \in \mathcal{L}(A^{(1, [\psi \cup \vartheta])}))$$

Since $(1, [\psi \cup \vartheta])$ is accepting, there need not exist $0 \leq k < |v|$ such that $v^{k..} \in \overline{[\neg \psi]}$. Thus,

$$v \in \mathcal{L}(A^{(1, [\psi \cup \vartheta])})$$

$$\iff v \models_{\text{PSL}} \neg \vartheta \wedge (\neg \psi \wedge \neg \vartheta)$$

$$\iff \text{[duality of U, W over } \Sigma]$$

$$v \models_{\text{PSL}} \neg [\psi \cup \vartheta]$$

□

This completes the proof of the proposition.

□

Corollary 3.3. *Given an LTL formula φ , one can build a Büchi automaton $B_\varphi = (\Sigma, S, I, \rho, F, A)$ where $|S|$ is in $2^{O(|\varphi|)}$ and $\mathcal{L}(B_\varphi)$ is exactly the set of (finite and infinite) words satisfying the formula φ .*

Proof. Follows directly from Propositions 3.1 and 4.4. By Proposition 3.1 one can build an alternating automaton from a given LTL formula φ . Proposition 4.4, which is given and proved independently on page 38, shows that there is a subsequent translation into a Büchi automaton. □

Automata Construction for LTL_WR

In this section we extend the construction of automata for LTL (over finite as well as infinite words) given in the previous section to all of LTL_WR. Recall from

Definition 2 that the LTL_WR formulas are generated from the LTL formulas by adding the basic form r , where r is a regular expression, and the inductive form $r \diamond \rightarrow \varphi$, where r is a regular expression and φ is an LTL_WR formula. Thus, the construction of Proposition 3.1 needs to be extended to handle each of these forms under both negation parities.

First we construct a non-deterministic finite automata on finite words (NFA) that recognizes tight satisfaction (\models_{PSL}) and a Büchi automata that recognizes weak RE satisfaction. Then we provide the overall construction for LTL_WR.

Proposition 3.4. *Let r be an RE. There exists a reduced NFA N_r such that*

$$w \models_{\text{PSL}} r \iff w \in \mathcal{L}(N_r)$$

and N_r has $O(|r|)$ states if r contains no intersection operator and $2^{O(|r|)}$ otherwise.

Proof. We inductively define an NFA N_r for r as follows:

- Base: For a Boolean expression b we construct the automaton $N_b = (\Sigma, \{0, 1\}, \{0\}, \rho_b, \{1\})$ such that for $l \in \Sigma$ we have $\rho(0, l) = \{1\}$ iff $l \models b$.
- Induction: assume that for the regular expressions r_1 and r_2 we constructed the NFAs $N_1 = (\Sigma, S_1, I_1, \rho_1, F_1)$ and $N_2 = (\Sigma, S_2, I_2, \rho_2, F_2)$ respectively.
 - The NFA N_{\cdot} for $r_1 \cdot r_2$ is $N_{\cdot} = (\Sigma, S_1 \cup S_2, I_1, \rho_1 \cup \rho_2 \cup \{(s_1, l, s_2) \mid s_1 \in F_1 \wedge s_2 \in \rho_2(q, l) \text{ for } q \in I_2\}, \hat{F})$. Where $\hat{F} = F_2$ if $I_2 \cap F_2 = \emptyset$ and $F_1 \cup F_2$ otherwise.
 - The NFA N_{\cup} for $r_1 \cup r_2$ is $N_{\cup} = (\Sigma, S_1 \cup S_2 \cup \{s_0\}, \{s_0\}, \rho_1 \cup \rho_2 \cup \{(s_0, l, s_1) \mid s_1 \in \rho_1(q, l) \text{ for } q \in I_1\} \cup \{(s_0, l, s_2) \mid s_2 \in \rho_2(q, l) \text{ for } q \in I_2\}, \hat{F})$. Where $\hat{F} = F_1 \cup F_2$ if $I_2 \cap F_2 = \emptyset$ and $I_1 \cap F_1 = \emptyset$ and $F_1 \cup F_2 \cup \{s_0\}$ otherwise.
 - The NFA N_{\cap} for $r_1 \cap r_2$ is obtained by building the AFA $A_{\cap} = (\Sigma, S_1 \cup S_2 \cup \{s_0\}, \{s_0\}, \rho_1 \cup \rho_2 \cup \{(s_0, l, s_1 \wedge s_2) \mid s_1 \in \rho_1(q, l) \text{ for } q \in I_1, s_2 \in \rho_2(q, l) \text{ for } q \in I_2\}, \hat{F})$, and remove universal edges by using subset construction afterwards (as described in Section 4 on page 33). Edges which can not lead to accepting states and non accepting states are removed. The accepting set is defined as $\hat{F} = F_1 \cup F_2$ if $I_2 \cap F_2 = \emptyset$ or $I_1 \cap F_1 = \emptyset$ and $F_1 \cup F_2 \cup \{s_0\}$ otherwise.
Note that the subset construction induces an exponential blowup. Removal of edges which can not lead to accepting states is done in linear time (reachability). All other operators take place in linear space.
 - The NFA N_* for r_1^* is $N_* = (\Sigma, S_1 \cup \{s_0\}, \{s_0\}, \rho_1 \cup \{(s_0, l, s_1) \mid s_1 \in \rho_1(q, l) \text{ and } q \in I_1\} \cup \{(s_1, l, s_0) \mid \rho_1(s_1, l) \cap F_1 \neq \emptyset\}, F_1 \cup \{s_0\})$.

Let w be a finite word. We show that

$$w \in \mathcal{L}(N_r) \quad \text{iff} \quad w \models_{\text{PSL}} r$$

by induction on the structure of r .

- Base: Let $r = b$. Note that 0 is not accepting and that there are no outgoing transitions from 1, thus if N_b accepts v , then $|v| = 1$. Then, N_b accepts a word v iff $|v| = 1$ and $1 \in \rho(0, v^0)$ iff $|v| = 1$ and $v^0 \models b$ iff $v \models_{\text{PSL}} b$.

- Induction: Assume that for REs r_1 and r_2 we proved that $w \in \mathcal{L}(N_1) \Leftrightarrow w \vDash_{\text{PSL}} r_1$ and $w \in \mathcal{L}(N_2) \Leftrightarrow w \vDash_{\text{PSL}} r_2$.

– For $r_1 \cdot r_2$ we distinguish between two cases:

1. The case where $\varepsilon \vDash_{\text{PSL}} r_2$. In this case, N for $r_1 \cdot r_2$ is $N = (\Sigma, S_1 \cup S_2, I_1, \rho_1 \cup \rho_2 \cup \{(s_1, l, s_2) \mid s_1 \in F_1 \wedge s_2 \in \rho_2(q, l) \text{ for } q \in I_2\}, F_1 \cup F_2)$. Then, N accepts a word w iff

it has a running trace s_0, s_1, \dots, s_k ($k = |w|$) over w such that $s_k \in F_1 \cup F_2$ iff

either N_1 accepts w or there exists $j < k$ such that $s_j \in F_1$ and $s_{j+1} \in \rho_2(q, w^j)$ for $q \in I_2$ iff

either N_1 accepts w or there exists $j < k$ such that N_1 accepts $w^{0..j}$ and N_2 accepts $w^{j+1..k}$ iff

(induction) either $w \vDash_{\text{PSL}} r_1$ or there exists $j < k$ such that $w^{0..j} \vDash_{\text{PSL}} r_1$ and $w^{j+1..k} \vDash_{\text{PSL}} r_2$ iff

$w \vDash_{\text{PSL}} r_1 \cdot r_2$.

2. The case where $\varepsilon \not\vDash_{\text{PSL}} r_2$. In this case, N for $r_1 \cdot r_2$ is $N = (\Sigma, S_1 \cup S_2, I_1, \rho_1 \cup \rho_2 \cup \{(s_1, l, s_2) \mid s_1 \in F_1 \wedge s_2 \in \rho_2(q, l) \text{ for } q \in I_2\}, F_2)$. Then, N accepts a word w iff

it has a running trace s_0, s_1, \dots, s_k ($k = |w|$) over w such that $s_k \in F_2$ iff

there exists $j < k$ such that $s_j \in F_1$ and $s_{j+1} \in \rho_2(q, w^j)$ for $q \in I_2$ iff

there exists $j < k$ such that N_1 accepts $w^{0..j}$ and N_2 accepts $w^{j+1..k}$ iff (induction)

there exists $j < k$ such that $w^{0..j} \vDash_{\text{PSL}} r_1$ and $w^{j+1..k} \vDash_{\text{PSL}} r_2$ iff

$w \vDash_{\text{PSL}} r_1 \cdot r_2$.

- For $r_1 \cup r_2$ we have that the NFA N_U for $r_1 \cup r_2$ is $N_U = (\Sigma, S_1 \cup S_2 \cup \{s_0\}, \{s_0\}, \rho_1 \cup \rho_2 \cup \{(s_0, l, s_1) \mid s_1 \in \rho_1(q, l) \text{ for } q \in I_1\} \cup \{(s_0, l, s_2) \mid s_2 \in \rho_2(q, l) \text{ for } q \in I_2\}, \hat{F})$. Where $\hat{F} = F_1 \cup F_2 \cup \{s_0\}$ if $I_2 \cap F_2 \neq \emptyset$ or $I_1 \cap F_1 \neq \emptyset$ and $\hat{F} = F_1 \cup F_2$ otherwise.

N_U accepts ε iff

$s_0 \in \hat{F}$ iff

$I_1 \cap F_1 \neq \emptyset$ or $I_2 \cap F_2 \neq \emptyset$ iff

N_1 accepts ε or N_2 accepts ε iff (induction) $\varepsilon \vDash_{\text{PSL}} r_1$ or $\varepsilon \vDash_{\text{PSL}} r_2$ iff

$\varepsilon \vDash_{\text{PSL}} r_1 \cup r_2$.

Let w be a non empty word. N_U accepts w iff

N_U has a running trace that s_0, s_1, \dots, s_k (for $k = |w|$) such that s_0 is the initial state and $s_k \in F_1 \cup F_2$ iff

either $s_1 \in \rho(q, w^0)$ for $q \in I_1$ and s_1, s_2, \dots, s_k is a running trace of N_1 or $s_1 \in \rho(q, w^0)$ for $q \in I_2$ and s_1, s_2, \dots, s_k is a running trace of N_2 iff

N_1 accepts w or N_2 accepts w iff (induction)

$w \vDash_{\text{PSL}} r_1$ or $w \vDash_{\text{PSL}} r_2$ iff $w \vDash_{\text{PSL}} r_1 \cup r_2$.

- For $r_1 \cap r_2$ we have that the NFA N_\cap is equivalent to the AFA $A_\cap = (\Sigma, S_1 \cup S_2 \cup \{s_0\}, \{s_0\}, \rho_1 \cup \rho_2 \cup \{(s_0, l, \{s_1, s_2\}) \mid s_1 \in \rho_1(q, l) \text{ for } q \in I_1, s_2 \in \rho_2(q, l) \text{ for } q \in I_2\}, \hat{F})$. Where $\hat{F} = F_1 \cup F_2$ if $I_2 \cap F_2 = \emptyset$ or $I_1 \cap F_1 = \emptyset$ and $F_1 \cup F_2 \cup \{s_0\}$ otherwise.

A_\cap accepts ε iff

$s_0 \in \hat{F}$ iff

$I_1 \cap F_1 \neq \emptyset$ and $I_2 \cap F_2 \neq \emptyset$ iff

N_1 accepts ε and N_2 accepts ε iff (induction) $\varepsilon \models_{\text{PSL}} r_1$ and $\varepsilon \models_{\text{PSL}} r_2$ iff $\varepsilon \models_{\text{PSL}} r_1 \cap r_2$.

Let w be a non empty word. A_{\cap} accepts w iff

A_{\cap} has a run with two branches of the form $s_0, s_{1,1}, \dots, s_{1,k}$ with $s_{1,k} \in F_1$ and $s_0, s_{2,1}, \dots, s_{2,k}$ with $s_{2,k} \in F_2$. (for s_0 is the initial state and $k = |w|$) iff

$(s_{1,1}, s_{2,1}) \in \rho(s_0, w^0)$ and $s_{1,1}, s_{1,2}, \dots, s_{1,k}$ is a running trace of N_1 and $s_{2,1}, s_{2,2}, \dots, s_{2,k}$ is a running trace of N_2 iff

N_1 accepts w and N_2 accepts w iff (induction)

$w \models_{\text{PSL}} r_1$ and $w \models_{\text{PSL}} r_2$ iff $w \models_{\text{PSL}} r_1 \cap r_2$.

Removal of edges to non accepting states does not change the language of an NFA because a missing edge for a input rejects the words as well as a path to a non accepting state.

- For r_1^* we have that the NFA N_* for r_1^* is $N_* = (\Sigma, S_1 \cup \{s_0\}, \{s_0\}, \rho_1 \cup \{(s_0, l, s_1) \mid s_1 \in \rho_1(q, l) \text{ and } q \in I_1\} \cup \{(s_1, l, s_0) \mid \rho_1(s_1, l) \cap F_1 \neq \emptyset\}, F_1 \cup \{s_0\})$.

First note that $s_0 \in F_1$, thus N_* accepts ε . This comply with $\varepsilon \models_{\text{PSL}} r_1^*$.

Let w be a non empty word. First direction, assume that N_* accepts w .

Let s_0, s_1, \dots, s_k be an accepting run trace of N_* on w . Let j_0, j_1, \dots be the sequence of all indices such that $s_{j_i} = s_0$. Then for every j_i in the set we have that $s_{j_i+1} \in \rho_1(q, w^{j_i})$ for $q \in I_1$ and that $p \in \rho_1(s_{j_i-1}, w^{j_i-1})$ for $p \in F_1$. Thus, for every j_i excepts for the last, we have that $q, s_{j_i+1}, \dots, s_{j_{i+1}-1}, p$ is accepting run of N_1 on $w^{j_i \cdot j_{i+1} - 1}$, and that for the last j_i , we have that $q, s_{j_i+1}, \dots, s_{k+1}$ is accepting run of N_1 on $w^{j_i \cdot k}$. This implies that (induction) for every j_i excepts for the last, we have that $w^{j_i \cdot j_{i+1} - 1} \models_{\text{PSL}} r_1$, and that for the last j_i , we have that $w^{j_i \cdot k} \models_{\text{PSL}} r_1$. Thus $w \models_{\text{PSL}} r_1^*$.

Second direction: Note that for every accepting run $q, s_1, \dots, s_{k-1}, p$ of N_1 on a non empty word w , there exists an accepting run $s_0, s_1, \dots, s_{k-1}, s_0$ of N_* on w . Assume that $w \models_{\text{PSL}} r_1^*$. Then, there are w_0, w_1, \dots, w_k such that $w_0 \cdot w_1 \cdot \dots \cdot w_k = w$ and for every $0 \leq j \leq k$ we have $w_j \models_{\text{PSL}} r_1$. Thus (induction), for every $0 \leq j \leq k$ we have that N_1 accepts w_j . Then, for every $0 \leq j \leq k$ we have N_* has a running trace on w_j that starts and ends at s_0 . This implies that N_* accepts w .

□

Proposition 3.5. *Let r be an RE. There exists a Büchi automaton B_r with $O(|r|)$ states if r contains no intersection operator and $2^{O(|r|)}$ states otherwise such that B_r has a trapping non-accepting state q_{bad} and for every word w over Σ ,*

1. *there exists an accepting run of B_r on w iff $w \models_{\text{PSL}} r$*
2. *every run of B_r on w reaches q_{bad} iff $w \not\models_{\text{PSL}} r$*

Proof. We build the Büchi automaton from the NFA for r constructed in Proposition 3.4. Before we provide the actual construction, we prove the following Lemma.

Lemma 3.6. *Let r be an RE and let N_r be the automaton constructed for r in Proposition 3.4. Then, for every $s \in S$ there exists $k \geq 0$ and a running trace s, s_1, \dots, s_k of N_r on \top^k such that $s_k \in F$.*

Proof. By induction.

- Base: $r = b$. The state 1 is in F , and $(0, \top, 1) \in \rho$.
- Induction: Assume that the NFAs N_1 and N_2 for r_1 and r_2 , respectively, satisfy the lemma.
 - * $r = r_1 \cdot r_2$. If $s \in S_2$, then by induction there exists $k \geq 0$ and a running trace s to F_2 on \top^k . Otherwise, $s \in S_1$. By induction, there is a running trace on \top^j from s to F_1 . From the transition relation, there exists $j \geq 0$ and a running trace on \top^j from s to a state in S_2 . By induction, there exists $k \geq 0$ and a running trace on \top^k from that state of S_2 to F_2 .
 - * $r = r_1 \cup r_2$. If $s \in S_i$, then by induction there exists $k \geq 0$ and a running trace on \top^k from s to F_i . Otherwise, $s = s_0$, and there is a transition on \top from s_0 to each of S_1 and S_2 .
 - * $r = r_1 \cap r_2$. Because of the explicit removal of edges and states from which we can not reach accepting states, this step is obvious.
 - * $r = r_1^*$. If $s \in S_1$, then by induction there exists $k \geq 0$ and a running trace on \top^k from s to F_1 . Otherwise, $s = s_0$, which is itself accepting.

□

Let $N_r = (\Sigma, S, I, \rho, F)$ be the NFA for RE r , constructed as in Proposition 3.4. Let $B_r = (\Sigma, S \cup \{q_{bad}\}, I, \rho', S, S)$ where

$$\rho' = \rho \cup \{(s, \ell, s) \mid \ell \in \Sigma, s \in F\} \cup \{(q_{bad}, \ell, q_{bad}) \mid \ell \in \Sigma\} \cup \{(s, \ell, q_{bad}) \mid \ell \in \Sigma, s \notin F \text{ and } \forall s' \in S: (s, \ell, s') \notin \rho\}$$

Observation 3.7. *Let w be a word over Σ . If there exists a running trace of B_r on w that contains a state in F , then B_r accepts w .*

Lemma 3.8. *A (finite or infinite) word w is accepted by B_r iff for every finite $v \preceq w$ there exists a prefix $u \preceq v \cdot \top^\omega$, such that $u \models_{\text{PSL}} r$.*

Proof. First direction: Let w be a word that is accepted by B_r . We prove that for every finite $v \preceq w$ there exists a prefix $u \preceq v \cdot \top^\omega$ such that $u \models_{\text{PSL}} r$. Let s_0, s_1, \dots be an accepting running trace of B_r on w . We distinguish between two cases:

1. The trace s_0, s_1, \dots contains a state in F (in N_r). In this case let k be the minimal number such that $s_k \in F$. The definition of ρ implies that s_0, s_1, \dots, s_k is a trace in N_r , thus $w^{0..k-1} \in \mathcal{L}(N_r)$. Proposition 3.4 implies that $w^{0..k-1} \models_{\text{PSL}} r$. Let $v \preceq w$ be a prefix of w . We distinguish between two cases:

- If $w^{0..k-1} \preceq v$, then $w^{0..k-1} \preceq v \cdot \top^\omega$ and $w^{0..k-1} \models_{\text{PSL}} r$.
- $v \prec w^{0..k-1}$. Let $j = |v|$, then s_0, s_1, \dots, s_j is a trace of N_r over v . Lemma 3.6 implies that there exists a trace s_j, s_{j+1}, \dots, s_i of N_r on \top^{i-j} such that $s_i \in F$, thus, $v \cdot \top^{i-j} \in \mathcal{L}(N_r)$. Then $v \cdot \top^{i-j} \models_{\text{PSL}} r$ and $v \cdot \top^{i-j} \preceq v \cdot \top^\omega$.

2. The trace s_0, s_1, \dots does not contain a state in F (in N_r). Then the definition of ρ implies that s_0, s_1, \dots is a trace in N_r . Let $v \preceq w$ be a finite prefix of w of length k . Then, s_0, s_1, \dots, s_{k+1} is a run of N_r on v . Lemma 3.6 implies that there exists a trace s_k, s_{k+1}, \dots, s_i of N_r on \top^{i-k} such that $s_i \in F$, thus, $v \cdot \top^{i-k} \in \mathcal{L}(N_r)$. Then $v \cdot \top^{i-k} \models_{\text{PSL}} r$ and $v \cdot \top^{i-k} \preceq v \cdot \top^\omega$.

Second direction: Let w be a word that is not accepted by B_r . We prove that there exists a finite $v \preceq w$ such that for every prefix $u \preceq v \cdot \top^\omega$, we have that $u \not\models_{\text{PSL}} r$. Since w is not accepted by B_r , and since q_{bad} is the only non accepting and non final state, there exists a prefix $v \preceq w$ such that all running trace of B_r on v are trapped in q_{bad} . In addition, by Observation 3.7 above there is no running trace of B_r on $u \preceq v$ that contains a state in F . The definition of ρ implies that there is no running trace of N_r on v , and that there is no running trace of N_r on $u \preceq v$ that contains a state in F . Since there is no running trace of N_r on v , for every $v \preceq u \preceq v \cdot \top^\omega$ we have that u is not accepted by N_r and thus $u \not\models_{\text{PSL}} r$. Since there is no running trace of N_r on $u \preceq v$ that contains a state in F , we have that for every $u \preceq v$, u is not accepted by N_r , and thus $u \not\models_{\text{PSL}} r$. \square

Lemma 3.8 shows that $w \models r$ iff there exists an accepting run of B_r on w . That is, $w \not\models_{\text{PSL}} r$ iff every run of B_r on w is not accepting.

Assume w is infinite. Then $w \not\models_{\text{PSL}} r$

- iff** every run of B_r on w does not visit S infinitely often
- iff** [since $S_r \setminus S = \{q_{bad}\}$] every run of B_r on w visits q_{bad} infinitely often
- iff** [since q_{bad} is trapping] every run of B_r on w visits q_{bad} .

Assume w is finite. Then $w \not\models_{\text{PSL}} r$

- iff** every run of B_r on w does not terminate in a state in S
- iff** [since $S_r \setminus S = \{q_{bad}\}$] every run of B_r on w terminates in S

Thus $w \not\models_{\text{PSL}} r$ iff every run of B_r on w reaches state q_{bad} . This completes the proof of Proposition 3.5. \square

Below we extend the construction of the alternating automaton given in the previous section for LTL (over finite as well as infinite words) to LTL_{WR}. The idea is that for RE-based formulas $r \diamond \rightarrow \psi$ and r the automaton mimics the automaton for r . Recall that $r \diamond \rightarrow \psi$ states that *there exists* a prefix tightly satisfying r and the suffix beginning at the end of this prefix satisfies ψ , while $r \mapsto \varphi$ states that *for each* prefix tightly satisfying r , the suffix beginning at the end of the prefix satisfies φ . Thus, for “ $r \diamond \rightarrow \varphi$ ” states when the parity bit is 0 the automaton mimics a move to some next state in the NFA of r (so as to require one match for r) and when the parity bit is 1 the automaton mimics moves to all next states in the NFA of r (in

order to traverse all matches for r). Then, when the automaton reaches a final state in N_r , it continues to state “ ψ ” in order to require satisfaction of ψ . For a weak RE r the automaton mimics the Büchi automaton B_r of Proposition 3.5. When the parity bit is 0, the automaton traverses all states, to check that all prefixes are “potentially accepting” (i.e., have an extension with \top s that is accepting). When the parity bit is 1, the automaton traverses some state to check for some bad prefix.

To make this idea work we need to consider a new type of formula involving automata. For these we introduce the following logic.

Definition 8 (LTL-WRA formulas).

- If b is a boolean expression then $b!$ is an LTL-WRA formula.
- If ϕ and ψ are LTL-WRA formulas, then the following are LTL-WRA formulas:
 - $\neg\phi$ • $\phi \wedge \psi$ • $X!\phi$ • $[\phi U \psi]$
- If ϕ is an LTL-WRA formula $N = (\Sigma, S, I, \rho, F)$ is an NFA such that I is a singleton, then $\langle N, \phi \rangle$ is an LTL-WRA formula.
- If $B = (\Sigma, S, I, \rho, F, A)$ is a Büchi automaton such that I is a singleton then $\langle B \rangle$ is an LTL-WRA formula.

The semantics of LTL-WRA formulas is defined as follows:

Definition 9. Let v denote a word over Σ ; b a boolean expression; r an RE; and ϕ and ψ LTL-WRA formulas. The notation $v \models_{\text{AUT}} \phi$ means that v satisfies ϕ . The relation \models_{AUT} is defined as follows:⁴

1. $v \models_{\text{AUT}} b! \iff |v| > 0$ and $v^0 \models_{\text{PSL}} b$
2. $v \models_{\text{AUT}} \neg\phi \iff \bar{v} \not\models_{\text{AUT}} \phi$
3. $v \models_{\text{AUT}} \phi \wedge \psi \iff v \models_{\text{AUT}} \phi$ and $v \models_{\text{AUT}} \psi$
4. $v \models_{\text{AUT}} X!\phi \iff |v| > 1$ and $v^{1..} \models_{\text{AUT}} \phi$
5. $v \models_{\text{AUT}} [\phi U \psi] \iff \exists 0 \leq k < |v|$ s.t. $v^{k..} \models_{\text{AUT}} \psi$ and $\forall 0 \leq j < k$, $v^{j..} \models_{\text{AUT}} \phi$
6. $v \models_{\text{AUT}} \langle N, \psi \rangle \iff \exists 0 \leq j < |v|$ s.t. $v^{0..j} \in \mathcal{L}(N)$ and $v^{j..} \models_{\text{AUT}} \psi$
7. $v \models_{\text{AUT}} \langle B \rangle \iff v \in \mathcal{L}(B)$

Proposition 3.9. Let ϕ be a formula of LTL-WR. Let Φ be the formula of LTL-WRA obtained from ϕ by replacing sub-formulas of the form $r \diamond \rightarrow \psi$ with $\langle N_r, \psi \rangle$ where $N_r = (\Sigma, S_r, I_r, \rho_r, F_r)$ is the NFA from Proposition 3.4; and formulas of the form r with $\langle B_r \rangle$ where $B_r = (\Sigma, S_r \cup \{q_{\text{bad}}\}, I_r, \rho_r', S_r, S_r)$ is the Büchi automaton defined in Proposition 3.5 and q_{bad} is the trapping non-accepting state. Then

$$\llbracket \phi \rrbracket = \llbracket \Phi \rrbracket$$

Proof. The proof is by induction on the structure of the formula. The cases for booleans and the LTL operators are immediate since the semantics for those are the same.

⁴The semantics of the LTL operators are the standard ones.

- $v \models_{\text{PSL}} r \diamond \rightarrow \psi$
 - $\iff \exists 0 \leq j < |v|$ s.t. $v^{0..j} \models_{\text{PSL}} r$ and $v^{j..} \models_{\text{PSL}} \psi$
 - \iff [by Proposition 3.4]
 - $\exists 0 \leq j < |v|$ s.t. $v^{0..j} \in \mathcal{L}(N_r)$ and $v^{j..} \models_{\text{PSL}} \psi$
 - $\iff v \models_{\text{AUT}} \langle N_r, \psi \rangle$
- $v \models_{\text{PSL}} r$
 - \iff [by Proposition 3.5]
 - $v \in \mathcal{L}(B_r)$
 - $\iff v \models_{\text{AUT}} \langle B_r \rangle$

□

Proposition 3.9 allows to use the construction of a Büchi automaton for an LTL_{WRA} formula Φ to obtain a construction for an LTL_{WR} formula ϕ for which $\llbracket \phi \rrbracket = \llbracket \Phi \rrbracket$.

Proposition 3.10. *Given an LTL_{WRA} formula ϕ , there exists an alternating Büchi automaton $B_\phi = (\Sigma, S, I, \rho, F, A)$ where $|S|$ is in $O(|\phi|)$ and $\mathcal{L}(B_\phi)$ is exactly the set of (finite and infinite) words satisfying the formula ϕ .*

Proof. Let ϕ be a formula of LTL_{WRA}. Let S be the set of pairs (p, ψ) where $p \in \{0, 1\}$ and ψ satisfies at least one of the following:

- ψ is a subformula of ϕ .
- $\psi \in \{\text{TRUE}, \text{FALSE}, \text{NONEMPTY}\}$.
- ψ is of the form $\langle N^q, \vartheta \rangle$, where $\langle N, \vartheta \rangle$ is a subformula of ϕ and q is a state of the NFA N .
- ψ is of the form $\langle B^q \rangle$, where $\langle B \rangle$ is a subformula of ϕ and q is a state of the Büchi automaton B .

Note that if q is the initial state of N , then $N^q = N$, and similarly with B .

The initial state is $(0, \phi)$.

The set A of accepting states is produced by the following rules:

1. A contains all states of the form: $(1, \text{NONEMPTY})$, $(0, \text{TRUE})$, $(1, \text{FALSE})$, $(1, b!)$, $(1, X!\psi)$, $(1, [\psi \cup \vartheta])$, $(1, \langle N^q, \psi \rangle)$, $(0, \langle B^q \rangle)$, $(1, \langle B^{q_{bad}} \rangle)$.
In $(0, \langle B^q \rangle)$ it is understood that q is not the trapping state of B , while in $(1, \langle B^{q_{bad}} \rangle)$ it is understood that q_{bad} is the trapping state of B .
2. $(p, \neg\psi) \in A$ iff $(1 - p, \psi) \in A$.
3. $(0, \psi \wedge \vartheta) \in A$ iff both $(0, \psi) \in A$ and $(0, \vartheta) \in A$.
 $(1, \psi \wedge \vartheta) \in A$ iff either $(1, \psi) \in A$ or $(1, \vartheta) \in A$.

The set F of final states is equal to A .

The transition relation ρ is defined as follows, where $N^q = (\Sigma, S_N, \{q\}, \rho_N, F_N)$ and $B^q = (\Sigma, S_B \cup q_{bad}, \{q\}, \rho_B, S_B)$:

- $\rho((p, \text{NONEMPTY}), \ell) = (p, \text{TRUE})$

- $\rho((p, \text{TRUE}), \ell) = (p, \text{TRUE})$
 $\rho((p, \text{FALSE}), \ell) = (p, \text{FALSE})$
- $\rho((0, b!), \ell) = (0, \ell \Vdash_{\text{PSL}} b)$
 $\rho((1, b!), \ell) = (1, \bar{\ell} \Vdash_{\text{PSL}} b)$
- $\rho((0, \psi \wedge \vartheta), \ell) = \rho((0, \psi), \ell) \wedge \rho((0, \vartheta), \ell)$
 $\rho((1, \psi \wedge \vartheta), \ell) = \rho((1, \psi), \ell) \vee \rho((1, \vartheta), \ell)$
- $\rho((0, \neg\psi), \ell) = \rho((1, \psi), \ell)$
 $\rho((1, \neg\psi), \ell) = \rho((0, \psi), \ell)$
- $\rho((0, X!\psi), \ell) = (0, \text{NONEMPTY}) \wedge (0, \psi)$
 $\rho((1, X!\psi), \ell) = (1, \text{NONEMPTY}) \vee (1, \psi)$
- $\rho((0, [\psi \cup \vartheta]), \ell) = \rho((0, \vartheta), \ell) \vee (\rho((0, \psi), \ell) \wedge (0, [\psi \cup \vartheta]))$
 $\rho((1, [\psi \cup \vartheta]), \ell) = \rho((1, \vartheta), \ell) \wedge (\rho((1, \psi), \ell) \vee (1, [\psi \cup \vartheta]))$
- If $\rho_N(q, \ell) \cap F_N$ is non-empty, then

$$\rho((0, \langle N^q, \psi \rangle), \ell) = \bigvee_{q' \in \rho_N(q, \ell)} (0, \langle N^{q'}, \psi \rangle) \vee \rho((0, \psi), \ell)$$

Otherwise

$$\rho((0, \langle N^q, \psi \rangle), \ell) = \bigvee_{q' \in \rho_N(q, \ell)} (0, \langle N^{q'}, \psi \rangle)$$

If $\rho_N(q, \bar{\ell}) \cap F_N$ is non-empty, then

$$\rho((1, \langle N^q, \psi \rangle), \ell) = \bigwedge_{q' \in \rho_N(q, \bar{\ell})} (1, \langle N^{q'}, \psi \rangle) \wedge \rho((1, \psi), \ell)$$

Otherwise

$$\rho((1, \langle N^q, \psi \rangle), \ell) = \bigwedge_{q' \in \rho_N(q, \bar{\ell})} (1, \langle N^{q'}, \psi \rangle)$$

- $\rho((0, \langle B^q \rangle), \ell) = \bigvee_{q' \in \rho_B(q, \ell)} (0, \langle B^{q'} \rangle)$
 $\rho((1, \langle B^q \rangle), \ell) = \bigwedge_{q' \in \rho_B(q, \bar{\ell})} (1, \langle B^{q'} \rangle)$

Lemma 3.11. *Let ϕ be an LTL-WRA formula. If ψ is a subformula of ϕ , let $A^{(p, \psi)}$ denote the subautomaton of this construction obtained by taking (p, ψ) as initial state. Then*

$$1. \mathcal{L}(A^{(0, \phi)}) = \llbracket \phi \rrbracket.$$

$$2. \mathcal{L}(A^{(1, \phi)}) = \Sigma^\infty - \overline{\llbracket \phi \rrbracket} = \llbracket \neg \phi \rrbracket.$$

Proof. The proof is by induction on the structure of the formula. The cases for booleans and the LTL operators are the same as in the proof of Lemma 3.2.

- We show that $\mathcal{L}(A^{(0, \langle N^q, \psi \rangle)}) = \llbracket \langle N^q, \psi \rangle \rrbracket$ and $\mathcal{L}(A^{(1, \langle N^q, \psi \rangle)}) = \llbracket \neg \langle N^q, \psi \rangle \rrbracket$
 - We show that $v \in \llbracket \langle N^q, \psi \rangle \rrbracket$ iff $v \in \mathcal{L}(A^{(0, \langle N^q, \psi \rangle)})$.

Suppose that $v \in \llbracket \langle N^q, \psi \rangle \rrbracket$. Then $\exists 0 \leq j < |v|$ s.t. $v^{0..j} \in \mathcal{L}(N^q)$ and $v^{j..} \in \llbracket \psi \rrbracket$. Since $v^{0..j} \in \mathcal{L}(N^q)$, there exists an accepting run

$$q, q_0, \dots, q_j \in F_N$$

of N^q on $v^{0..j}$. By induction, $\mathcal{L}(A^{(0,\Psi)}) = \llbracket \Psi \rrbracket$, so there exists an accepting run tree t of $A^{(0,\Psi)}$ on $v^{j..}$.

From the transition relation,

$$\langle N^q, \Psi \rangle, \langle N^{q_0}, \Psi \rangle, \dots, \langle N^{q_{j-1}}, \Psi \rangle$$

is a run of $A^{(0,\langle N^q, \Psi \rangle)}$ on $v^{0..j-1}$. Also, since $q_j \in F_N$, it follows from the transition relation that each of the successors of the root of the run tree t on letter v^j is also a successor of $\langle N^{q_{j-1}}, \Psi \rangle$ on the letter v^j . Let t' be obtained from t by replacing the root with $\langle N^{q_{j-1}}, \Psi \rangle$. Since t, t' differ only on the root node and since $v^{j..}$ is non-empty, it follows that t' is an accepting run tree of $A^{(0,\langle N^{q_{j-1}}, \Psi \rangle)}$ on $v^{j..}$. This shows that $v \in \mathcal{L}(A^{(0,\langle N^q, \Psi \rangle)})$.

Suppose now that $v \in \mathcal{L}(A^{(0,\langle N^q, \Psi \rangle)})$. Then there exists an accepting run tree t of $A^{(0,\langle N^q, \Psi \rangle)}$ on v . Since $(0, \langle N^q, \Psi \rangle)$ is not accepting, v is non-empty. From the transition relation, t has a branch labeled

$$(0, \langle N^q, \Psi \rangle), (0, \langle N^{q_0}, \Psi \rangle), \dots$$

Since none of the states $(0, \langle N^{q_0}, \Psi \rangle)$ is accepting, there must exist $0 \leq j < |v|$ such that

$$(0, \langle N^q, \Psi \rangle), (0, \langle N^{q_0}, \Psi \rangle), \dots, (0, \langle N^{q_{j-1}}, \Psi \rangle)$$

labels a branch t, t_0, \dots, t_{j-1} of t , and q, q_0, \dots, q_{j-1} is a run of N^q on $v^{0..j-1}$, and $\rho_N(q_{j-1}, v^j) \cap F_N \neq \emptyset$, and the set of children of t_{j-1} is labeled by the set $\rho((0, \Psi), v^j)$. Then q, q_0, \dots, q_j is a run of N^q on $v^{0..j}$, and so $v^{0..j} \in \mathcal{L}(N^q)$. Let $t' = t_{j,1}$.

Since t is accepting, t' must be an accepting run of $A^{(0,\langle N^{q_{j-1}}, \Psi \rangle)}$ on $v^{j..}$. Let t'' be the tree obtained from t' by switching the root to $(0, \Psi)$. From the transition relation, t'' is a run tree of $A^{(0,\Psi)}$ on $v^{j..}$. Since $v^{j..}$ is non-empty, it follows that t'' is accepting. Therefore, $v^{j..} \in \mathcal{L}(A^{(0,\Psi)})$. By induction, $v^{j..} \in \llbracket \Psi \rrbracket$. Thus, $v \in \llbracket \langle N^q, \Psi \rangle \rrbracket$.

- We show that $v \in \llbracket \neg \langle N^q, \Psi \rangle \rrbracket$ iff $v \in \mathcal{L}(A^{(1,\langle N^q, \Psi \rangle)})$.

$v \in \llbracket \neg \langle N^q, \Psi \rangle \rrbracket$ iff $\bar{v} \notin \llbracket \langle N^q, \Psi \rangle \rrbracket$ iff $\forall 0 \leq j < |v|$ s.t. $\bar{v}^{0..j} \in \mathcal{L}(N^q)$, $\bar{v}^{j..} \notin \llbracket \Psi \rrbracket$ iff [induction]

$$(\star) : \quad \forall 0 \leq j < |v| \text{ s.t. } \bar{v}^{0..j} \in \mathcal{L}(N^q), \\ v^{j..} \in \mathcal{L}(A^{(1,\Psi)})$$

Suppose that (\star) holds. We construct an accepting run tree t of $A^{(1,\langle N^q, \Psi \rangle)}$ on v . Let t_0 be a run tree of $A^{(1,\langle N^q, \Psi \rangle)}$ on v . Let t_N be the rooted subtree of t_0 consisting of all states of the form $(1, \langle N^{q'}, \Psi \rangle)$ (that is, trim every branch of t_0 at the first node which is not of the form $(1, \langle N^{q'}, \Psi \rangle)$). Since the states of this form are all accepting, every branch of this subtree is accepting. Consider a node $n = (1, \langle N^{q'}, \Psi \rangle)$ of t_N such that n has a nonempty set M_n of successors

in $t_0 - t_N$. From the transition relation, it follows that there exists $0 \leq j < |v|$ and a run $q, q_0, \dots, q_{j-1} = q'$ of N^q on $\bar{v}^{0..j-1}$ and there exists $q_j \in \rho_N(q', \bar{v}^j) \cap F_N$. Then q, q_0, \dots, q_j is an accepting run of N^q on $\bar{v}^{0..j}$, so by (\star) , $v^{j..} \in \mathcal{L}(A^{(1, \Psi)})$. Let t_n be an accepting run tree of $A^{(1, \Psi)}$ on $v^{j..}$. Replace M_n in t_0 by the set of successors of the root of t_n . As a result, any branch of t_0 passing through a successor of n in $t_0 - t_N$ is now converted to an accepting branch. The procedure is repeated for all such nodes n to yield the accepting run tree t .

Suppose that $v \in \mathcal{L}(A^{(1, \langle N^q, \Psi \rangle)})$. Let t be an accepting run tree of $A^{(1, \langle N^q, \Psi \rangle)}$ on v . Let t_N be the rooted subtree of t consisting of all states of the form $(1, \langle N^{q'}, \Psi \rangle)$ (that is, trim every branch of t at the first node which is not of the form $(1, \langle N^{q'}, \Psi \rangle)$). Suppose that there exists $0 \leq j < |v|$ s.t. $\bar{v}^{0..j} \in \mathcal{L}(N^q)$. Let q, q_0, \dots, q_j be an accepting run of N^q on $\bar{v}^{0..j}$. From the transition relation,

$$(1, \langle N^q, \Psi \rangle), (1, \langle N^{q_0}, \Psi \rangle), \dots, (1, \langle N^{q_{j-1}}, \Psi \rangle)$$

is a prefix of a branch of t_N , and the successors of $(1, \langle N^{q_{j-1}}, \Psi \rangle)$ in $t - t_N$ are successors of $(1, \Psi)$ on v^j .

Let t' be the tree obtained from the subtree of t rooted at $(1, \langle N^{q_{j-1}}, \Psi \rangle)$ by eliminating the successors of $(1, \langle N^{q_{j-1}}, \Psi \rangle)$ that are in t_N and replacing the root by $(1, \Psi)$. It follows that t' is an accepting run tree of $A^{(1, \Psi)}$ on $v^{j..}$, and so $v^{j..} \in \mathcal{L}(A^{(1, \Psi)})$. This proves that (\star) holds.

- We show that $\mathcal{L}(A^{(0, \langle B^q \rangle)}) = \llbracket \langle B^q \rangle \rrbracket$ and $\mathcal{L}(A^{(1, \langle B^q \rangle)}) = \llbracket \neg \langle B^q \rangle \rrbracket$.

Let w be a finite/infinite word.

Recall that q' is an accepting state of $\langle B^q \rangle$

iff $q' \neq q_{bad}$. Also, $(0, \langle B^{q'} \rangle)$ is an accepting state of $A^{(0, \langle B^q \rangle)}$

iff $q' \neq q_{bad}$.

Thus, by the construction,

$$w \in \mathcal{L}(A^{(0, \langle B^q \rangle)})$$

iff there exists a run of B^q on w that does not visit q_{bad}

iff $w \in \mathcal{L}(B^q)$

iff $w \in \llbracket \langle B^q \rangle \rrbracket$

Similarly, $(1, \langle B^{q'} \rangle)$ is an accepting state of $A^{(1, \langle B^q \rangle)}$ **iff** $q' = q_{bad}$. Thus,

$$w \in \mathcal{L}(A^{(1, \langle B^q \rangle)})$$

iff every run of B^q on \bar{w} reaches q_{bad}

iff $\bar{w} \notin \mathcal{L}(B^q)$

iff $\bar{w} \notin \llbracket \langle B^q \rangle \rrbracket$

iff $w \in \llbracket \neg \langle B^q \rangle \rrbracket$

□

This completes the proof of the Proposition. □

Corollary 3.12. *Given an LTL_WRA formula φ , there exists a Büchi automaton $B_\varphi = (\Sigma, S, I, \rho, F, A)$ where $|S|$ is in $2^{2^{O(|\varphi|)}}$ and $\mathcal{L}(B_\varphi)$ is exactly the set of (finite and infinite) words satisfying the formula φ .*

Proof. Follows directly from Propositions 3.10 and 4.4. By Proposition 3.10 one can build an alternating automaton from a given LTL_WRA formula φ . Proposition 4.4, which is given and proved independently on page 38, shows that there is a subsequent translation into a Büchi automaton. \square

Corollary 3.13. *Given an LTL_WR formula φ , there exists a Büchi automaton $B_\varphi = (\Sigma, S, I, \rho, F, A)$ where $|S|$ is in $2^{2^{O(|\varphi|)}}$ and $\mathcal{L}(B_\varphi)$ is exactly the set of (finite and infinite) words satisfying the formula φ .*

Proof. Follows directly from Corollary 3.12 and Proposition 3.9. \square

Definition 10. *We say that a language is ∞ -regular iff it is a union of a regular language and an ω -regular language.*

Proposition 3.14. *LTL_WR is as expressive as ∞ -regular languages.*

Proof. Clearly a language accepted by a Büchi automaton on finite/infinite words is an ∞ -regular language. Thus, Corollary 3.13 implies that LTL_WR formulas are not more expressive than ∞ -regular expressions (since for every formula φ in LTL_WR there exists a Büchi automaton B_φ on finite/infinite words such that $\llbracket \varphi \rrbracket = \mathcal{L}(B_\varphi)$).

A proof that any ω -regular language, can be expressed as an LTL_WR formula appears in [4], where the suffix implication operator (\vdash) is denoted `triggers`, and the suffix conjunction operator ($\diamond\rightarrow$) is denoted `follows_by`. Note, that the language of an LTL_WR formula interpreted over finite and infinite words may not be the same as its language when interpreted over infinite words alone. For example, when the semantics is interpreted over infinite words alone, we have that $\llbracket b \text{ W false} \rrbracket = \{\ell \mid \ell \Vdash_{\text{PSL}} b\}^\omega$, but when the semantics is interpreted over finite words as well, we have $\llbracket b \text{ W false} \rrbracket = \{\ell \mid \ell \Vdash_{\text{PSL}} b\}^\omega \cup \{\ell \mid \ell \Vdash_{\text{PSL}} b\}^*$.

A regular language (over finite words) L can be expressed in LTL_WR by means of the formula $(r_L \diamond\rightarrow \text{X false})$ where r_L is a regular expression accepting the same language as L . An omega regular language (over infinite words) L can be expressed in LTL_WR by means of the formula $\psi_L \wedge ((\text{X!true}) \text{ W false})$ where ψ_L is an LTL_WR formula accepting the language L when interpreted over infinite words only.

Thus, for every ∞ -regular language L we can construct an LTL_WR formula ψ_{fin} such that $\llbracket \psi_{fin} \rrbracket = \{w \mid w \in L \text{ and } w \text{ is finite}\}$ and a formula ψ_{inf} such that $\llbracket \psi_{inf} \rrbracket = \{w \mid w \in L \text{ and } w \text{ is infinite}\}$. Thus, the formula $\psi_{fin} \vee \psi_{inf}$ accepts exactly the set of (finite and infinite) words in L . \square

Classification of Automata for RE-based LTL_{WR}-formulas

Proposition 3.15. *Given an LTL_{WR} formula ϕ , one can build a weak alternating Büchi automaton $B_\phi = (\Sigma, S, I, \rho, F, A)$ where $|S|$ is in $O(|\phi|)$ and $\mathcal{L}(B_\phi)$ is exactly the set of (finite and infinite) words satisfying the formula ϕ .*

Proof. We claim that the construction given in Proposition 3.10 yields a weak alternating automaton. Thus, it is left to show that the constructed automaton is weak. We show this by attaching a ranking function v that satisfies the following two properties: (1) if $s' \in \rho(s, \ell)$ for some ℓ then $v(s') \leq v(s)$ and (2) a state is accepting iff $v(s)$ is even. The definition of v is given as follows, where $\lceil n \rceil_e$ and $\lceil n \rceil_o$ denotes the least even (resp. odd) number that is at least as n .

- $v(0, \text{TRUE}) = 0, \quad v(1, \text{TRUE}) = 1, \quad v(0, \text{FALSE}) = 1, \quad v(1, \text{FALSE}) = 0$
- $v(0, \text{NONEMPTY}) = 1, \quad v(1, \text{NONEMPTY}) = 2$
- $v(0, b!) = 1, \quad v(1, b!) = 2, \quad v(0, \neg b!) = 2, \quad v(1, \neg b!) = 1$
- $v(0, \langle B^q \rangle) = 2, \quad v(1, \langle B^q \rangle) = 3, \quad v(0, \langle B^{q_{bad}} \rangle) = 1, \quad v(1, \langle B^{q_{bad}} \rangle) = 2$
where in $(p, \langle B^q \rangle)$ it is understood that q is not the trapping state of B , while in $(p, \langle B^{q_{bad}} \rangle)$ it is understood that q_{bad} is the trapping state of B .
- $$v(p, \psi \wedge \vartheta) = \begin{cases} \lceil \max\{v(p, \psi), v(p, \vartheta)\} \rceil_e & \text{if } (p, \psi \wedge \vartheta) \text{ is accepting} \\ \lceil \max\{v(p, \psi), v(p, \vartheta)\} \rceil_o & \text{otherwise} \end{cases},$$

$$v(p, \neg(\psi \wedge \vartheta)) = \begin{cases} \lceil \max\{v(p, \psi), v(p, \vartheta)\} \rceil_e & \text{if } (p, \neg(\psi \wedge \vartheta)) \text{ is accepting} \\ \lceil \max\{v(p, \psi), v(p, \vartheta)\} \rceil_o & \text{otherwise} \end{cases}$$
- $v(0, X!\psi) = \lceil \max\{v(0, \text{NONEMPTY}), v(0, \psi)\} \rceil_o,$
 $v(1, X!\psi) = \lceil \max\{v(1, \text{NONEMPTY}), v(1, \psi)\} \rceil_e,$
 $v(0, \neg X!\psi) = \lceil \max\{v(1, \text{NONEMPTY}), v(1, \psi)\} \rceil_e,$
 $v(1, \neg X!\psi) = \lceil \max\{v(0, \text{NONEMPTY}), v(0, \psi)\} \rceil_o$
- $v(0, [\psi \cup \vartheta]) = \lceil \max\{v(0, \psi), v(0, \vartheta)\} \rceil_o,$
 $v(1, [\psi \cup \vartheta]) = \lceil \max\{v(1, \psi), v(1, \vartheta)\} \rceil_e,$
 $v(0, \neg[\psi \cup \vartheta]) = \lceil \max\{v(1, \psi), v(1, \vartheta)\} \rceil_e,$
 $v(1, \neg[\psi \cup \vartheta]) = \lceil \max\{v(0, \psi), v(0, \vartheta)\} \rceil_o$
- $v(0, \langle N^q, \psi \rangle) = \lceil v(0, \psi) \rceil_o, \quad v(1, \langle N^q, \psi \rangle) = \lceil v(1, \psi) \rceil_e,$
 $v(0, \neg \langle N^q, \psi \rangle) = \lceil v(1, \psi) \rceil_e, \quad v(1, \neg \langle N^q, \psi \rangle) = \lceil v(0, \psi) \rceil_o$

□

Proposition 3.16. *Let r be an RE, and ϕ an LTL_{WR} formula. If there exists a weak (terminal) Büchi automaton for $\neg\phi$ with state set S , then there exists a weak (terminal) Büchi automaton for $\neg(r \mapsto \phi)$ with at most $O(|r| + |S|)$ states.*

Proof. Let r be an RE and ϕ be an LTL_{WR} formula. We want to build a Büchi automaton for the negation of $r \mapsto \phi$. Note that by the semantics of suffix implication $\llbracket r \mapsto \phi \rrbracket = \llbracket r' \mapsto \phi' \rrbracket$ where $\mathbb{S}(r') = \mathbb{S}(r) \setminus \{\varepsilon\}$ and $\llbracket \phi' \rrbracket = \llbracket \phi \rrbracket \setminus \{\varepsilon\}$. We therefore assume without loss of generality that $\varepsilon \notin \mathbb{S}(r)$ and that $\varepsilon \notin \llbracket \phi \rrbracket$

Let $N_r = (\Sigma, S_r, I_r, \rho_r, F_r)$ be the NFA constructed for r as in Proposition 3.4. We build from it the NFA $N'_r = (\Sigma, S'_r, I_r, \rho'_r, F'_r)$ where $S'_r = S_r \cup \{q_{fin}\}$, $F'_r = \{q_{fin}\}$ and $\rho'_r = \rho_r \cup \{(q, \ell, q_{fin}) \mid \exists q_f \in F_r \text{ s.t. } (q, \ell, q_f) \in \rho_r\}$. Clearly, N'_r is a linear NFA in $|r|$ accepting the words tightly satisfying r such that there are no states in $I_r \cap F'_r$.

Let $B_\varphi = (\Sigma, S_\varphi, I_\varphi, \rho_\varphi, F_\varphi, A_\varphi)$ be the Büchi automaton accepting language $\llbracket \neg\varphi \rrbracket$. We can assume the construction of B_φ is such that I_φ is disjoint from F_φ and that $(s, \ell, s') \in \rho_\varphi$ implies $s' \notin I_\varphi$.

We build the Büchi automaton $B = (\Sigma, S, I, \rho, F, A)$ for $\neg(r \mapsto \varphi)$ as follows: $S = (S'_r \setminus F'_r) \cup (S_\varphi \setminus I_\varphi)$; $I = I_r$; $F = F_\varphi$; $A = A_\varphi$; and

$$\rho = (\rho_r \setminus \{(s, \ell, t) \mid t \in F'_r\}) \cup (\rho_\varphi \setminus \{(s^0, \ell, s') \mid s^0 \in I_\varphi\}) \cup \{(s, \ell, s') \mid \exists t \in F'_r \text{ s.t. } (s, \ell, t) \in \rho_r \text{ and } \exists s^0 \in I_\varphi \text{ s.t. } (s^0, \ell, s') \in \rho_\varphi\}$$

First we show that $\mathcal{L}(B) = \llbracket \neg(r \mapsto \varphi) \rrbracket$. Then we argue that if B_φ is weak (terminal) then so is B .

1. $v \in \llbracket \neg(r \mapsto \varphi) \rrbracket$

$$\iff v \models_{\text{PSL}} \neg(r \mapsto \varphi)$$

$$\iff v \models_{\text{PSL}} r \diamond \neg\varphi$$

$$\iff \exists j < |v| \text{ s.t. } v^{0..j} \models_{\text{PSL}} r \text{ and } v^{j..} \models_{\text{PSL}} \neg\varphi$$

$$\iff \text{there exists a run } s_0, s_1, \dots, s_{j+1} \text{ of } N'_r \text{ on } v^{0..j} \text{ such that } s_{j+1} \in F'_r \text{ and there exists an accepting (finite or infinite) run } s'_0, s'_1, s'_2 \dots \text{ of } B_\varphi \text{ on } v^{j..} \text{ such that } s'_0 \in I_\varphi$$

$$\iff [s_j \xrightarrow{v^j} s'_1 \in \rho \text{ iff } \exists s_{j+1} \in F'_r \text{ s.t. } s_j \xrightarrow{v^j} s_{j+1} \in \rho_r \text{ and } \exists s'_0 \in I_\varphi \text{ s.t. } s'_0 \xrightarrow{v^j} s'_1 \in \rho_\varphi] \text{ there exists an accepting (finite or infinite) run } s_0, s_1, \dots, s_j, s'_1, s'_2 \dots \text{ of } B \text{ on } v$$

$$\iff v \in \mathcal{L}(B)$$

2. Let S_1, \dots, S_k be the partition of states and \leq the partial order that prove B_φ is weak (terminal). Let S'_i be $S_i \setminus I_\varphi$ for $1 \leq i \leq k$. Let $S'_r = S_r \setminus F'_r$. Then B is proved weak (terminal) by the partition S'_r, S'_1, \dots, S'_k together with the partial order $S'_r \leq S'_i$, and $S'_i \leq S'_j$ iff $S_i \leq S_j$.

□

Proposition 3.17. *Let r be an RE. Then,*

- *there exists an universal alternating Büchi automaton for $\neg r$ with state complexity in $O(|r|)$ if r contains no intersection operator and with state complexity in $2^{O(|r|)}$ otherwise, and*
- *there exists a terminal existential Büchi automaton for $\neg r$ with state complexity in $2^{O(|r|)}$ if r contains no intersection operator and with state complexity in $2^{2^{O(|r|)}}$ otherwise.*

Proof. Let $B_r = (\Sigma, S \cup \{q_{bad}\}, I, \rho, S, S)$ be the existential Büchi automaton from Proposition 3.5. Recall that B_r has $2^{O(|r|)}$ states and for every word w over Σ ,

1. there exists an accepting run of B_r on w iff $w \models_{\text{PSL}} r$

2. every run of B_r on w reaches q_{bad} iff $w \not\models_{\text{PSL}} r$

Let B be the universal Büchi automaton $(\Sigma, S \cup \{q_{bad}\}, I, \rho, \{q_{bad}\}, \{q_{bad}\})$. Clearly B accepts a word w iff $w \not\models_{\text{PSL}} r$. Let \bar{B} be the universal Büchi automaton $(\Sigma, S \cup \{q_{bad}\}, I, \bar{\rho}, \{q_{bad}\}, \{q_{bad}\})$ where

$$\bar{\rho} = \{(s_1, \ell, s_2) \mid \ell \in \Sigma, (s_1, \bar{\ell}, s_2) \in \rho\}.$$

Clearly, $w \in \mathcal{L}(B)$ iff $\bar{w} \in \mathcal{L}(\bar{B})$. Therefore $w \in \mathcal{L}(B)$ iff $\bar{w} \not\models_{\text{PSL}} r$ iff $w \models_{\text{PSL}} \neg r$.

The universal Büchi automaton \bar{B} can be translated into an ordinary existential Büchi automaton B' with state set S' such that $|S'|$ is in $2^{2^{O(|r|)}}$ [21]. This translation can be done in a way that preserves q_{bad} as the unique trapping accepting state of B' . Then it is clear that B' is terminal by partitioning according to $S \setminus \{q_{bad}\} < \{q_{bad}\}$

□

Proposition 3.18. *Let r be an RE. Then, there exists a weak Büchi automaton for $\neg r!$ whose state complexity is in $O(|r|)$ if r contains no intersection operator and in $2^{O(|r|)}$ otherwise.*

Proof. Let r be an RE. Let $N = (\Sigma, S, I, \rho, F)$ be an NFA accepting the non-empty words tightly satisfying r , with state complexity linear in $|r|$ (for example, N is the NFA of Proposition 3.4). We build from it the NFA $N' = (\Sigma, S', I, \rho', F')$ where $S' = S \cup \{q_{fin}\}$, $F' = \{q_{fin}\}$ and $\rho' = \rho \cup \{(q, \ell, q_{fin}) \mid \exists q_f \in F \text{ s.t. } (q, \ell, q_f) \in \rho\}$. Clearly, N' is a linear NFA in $|r|$ accepting the non-empty words tightly satisfying r such that there are no transitions out of F' . It follows that in any accepting run, F' cannot be reached prior to the end of the run (i.e., if state $s_f \in F'$ is reached during an accepting run, then s_f is the last state of the run).

We build a Büchi automaton $B'' = (\Sigma, S', I, \rho'', F'', A'')$ as follows: $A'' = F'' = S' \setminus F'$; and $\rho'' = \rho' \cup \{(s, \ell, s) \mid s \in F', \ell \in \Sigma\}$. Note that the states of F' in B are non-accepting and trapping. Finally, we build a Büchi automaton $B = (\Sigma, S', I, \bar{\rho}, F'', A'')$ for $\neg r!$ as follows:

$$\bar{\rho} = \{(s_1, \ell, s_2) \mid \ell \in \Sigma, (s_1, \bar{\ell}, s_2) \in \rho''\}.$$

Clearly $v \in \mathcal{L}(B)$ if and only if $\bar{v} \in \mathcal{L}(B'')$.

First we show that $\mathcal{L}(B) = \llbracket \neg(r!) \rrbracket$. Then we argue that B is weak.

1. $v \in \llbracket \neg(r!) \rrbracket$

$$\iff v \models_{\text{PSL}} \neg r!$$

$$\iff \bar{v} \not\models_{\text{PSL}} r!$$

$$\iff \nexists k < |v| \text{ s.t. } \bar{v}^{0..k} \models_{\text{PSL}} r$$

$$\iff \forall k < |v|, \bar{v}^{0..k} \not\models_{\text{PSL}} r$$

$$\iff \text{[since } N' \text{ recognizes non-empty matches of } r]$$

$$\forall k < |v| \text{ there exists no run of } N' \text{ on } \bar{v}^{0..k} \text{ terminating in a state in } F'$$

$$\iff \text{[by the properties of } N']$$

$$\forall k < |v| \text{ there exists no run of } N' \text{ on } \bar{v}^{0..k} \text{ visiting a state in } F'$$

\Leftrightarrow [by the definition of B'']

$\forall k < |v|$ there exists no run of B'' on $\bar{v}^{0..k}$ visiting a state in F'

\Leftrightarrow [there are two options]

Either v is finite and any run of B'' on \bar{v} does not terminate in a state in F'

Or v is infinite and any run of B'' on \bar{v} never visits a state in F'

$\Leftrightarrow \bar{v} \in \mathcal{L}(B'')$

$\Leftrightarrow v \in \mathcal{L}(B)$

2. Let the partition and partial order of the states of B be $S' \setminus F' < F'$. Since $F'' = A'' = S' \setminus F'$ and F' is trapping, it follows that B is weak.

□

4 Translating Alternating into Nondeterministic Automata

This section discusses the translation of alternating Büchi automata to nondeterministic Büchi automata. Nondeterministic automata are necessary in most applications, including model checking.

The usual strategy for linear temporal logic (LTL) model checking is the following

1. Convert the negation of the desired property into a nondeterministic Büchi automaton (a *tableau*),
2. construct the product of the tableau and the finite-state machine representing the model under test, and
3. check for language emptiness on the resulting product automaton. For language emptiness checking a fair CTL model checker can be used [19].

Various methods have been suggested for the construction of the tableau for LTL [19, 18, 11, 12, 26, 17].

From an automata-theoretic point of view, the difference of the PSL core LTL_{WR} and linear temporal logic is essential. Linear temporal logic is equivalent to one-weak alternating automata which in turn correspond to star-free ω -regular languages [24][27]. The logic LTL_{WR}, on the other hand, is as expressive as ω -regular languages (or more accurately ∞ -regular languages) as stated in Proposition 3.14.

The fact that PSL is more expressive implies that the automata generation procedures for LTL do not generalize to PSL. Nevertheless, there are large classes of properties which generate one-weak automata or even terminal automata (e.g. repetition of single letters or star-free expressions).

To take advantage of this fact, we propose a flexible workflow using various algorithms to maximize efficiency. We present a general algorithm, based on Miyano and Hayashi's approach, which can handle all alternating automata. For the subclasses of one-weak and terminal automata we propose the use of specialized, more efficient algorithms. Therefore, the first step of model checking is to examine the alternating automaton and to choose the right algorithm to derive a nondeterministic tableau. The constructions are described in the following subsections.

- **Terminal Automata:** For terminal automata (which represent languages over finite words) we can use the subset construction to remove universal choice, yielding in automata of size 2^n .
- **One-weak Automata:** For these automata we present an algorithm to create a non-deterministic automaton of size $n \cdot 2^n$

- **Weak Alternating Automata:** For the general case of weak alternating automata, we use a variant of Miyano and Hayashi's approach. The resulting non-deterministic Büchi automaton has size 2^{2n} .

Many algorithms need a symbolic (BDD-based) representation of the automaton. The standard approach is to first translate the alternating automaton to a non-deterministic one, followed by a symbolic encoding of the non-deterministic automaton. The reencoding introduces inefficiency. At the end of this section, we show how to avoid this inefficiency by introducing a direct transformation from alternating automata to a symbolic representation in the form of a discrete transition structure. This transformation is inspired by the construction of [19, 11] for LTL. We extend it to alternating automata.

The following two subsections introduce the proposed algorithms in detail. The first one considers the three variations of universal choice removal, and the second one covers the implementation of efficient symbolic model checking.

Removing Universality

This section covers three solutions to translate alternating to non-deterministic automata by removing universal choice. In the first subsection, the subset construction is considered. This construction works for automata on finite words, and hence also for terminal automata. We also show why this construction does not work for automata on infinite words.

In the second subsection, an efficient algorithm for one-weak automata is proposed, followed in the third subsection by the discussion of a general algorithm in the style of Miyano and Hayashi.

The subset construction is widely known as a means to create deterministic finite-word automata from non-deterministic finite-word automata by eliminating existential choice [20]. For automata on finite words this method can also be used to eliminate universal choice [10], thus translating an alternating to a non-deterministic automaton. When removing non-determinism, a state in the resulting automaton represents the set of states that can be reached for a given input in the original automaton. When removing universality, a state in the resulting automaton represents the set of states that are reached simultaneously, on different branches of the run tree.

For finite words and a corresponding acceptance condition, the subset construction is perfectly suitable. Acceptance for the new states can be determined easily: if all simultaneously visited states are accepting in the original automaton, then the state representing this set is accepting as well. The same holds for terminal automata, which in effect recognize languages over finite words.

Infinite words and corresponding acceptance conditions pose a challenge. As we will show by example, later in this section, it is not easy to reduce the acceptance

conditions for separate branches of a run tree to an acceptance condition expressed in terms of the set of states reached at any particular level of the tree. Miyano and Hayashi [21] have shown how this problem can be avoided by using a modified version of the subset construction. The modified construction has the drawback of requiring 2^{2n} instead of 2^n states.

For one-weak automata we have a somewhat easier job. For such automata, the question is whether a branch in the run gets stuck in a bad state. This is something that can again be checked locally, by seeing if a self loop on a bad state is being used. This allows us to give a more efficient algorithm than the one of Miyano and Hayashi, using $n \cdot 2^n$ states.

Removing Universality from Terminal Alternating Automata

An algorithm for the proposed idea of subset construction for removal of alternation is described next. It can be applied to alternating automata on finite words and terminal alternating automata [10].

1. The algorithm takes an alternating automaton with labels on the edges as input.
2. The output of the algorithm is a non-deterministic automaton. A state of the non-deterministic automaton is identified by a set of states of the alternating automaton.
3. Add the initial states to an empty stack S_{proc} containing states of the constructed automaton yet to be processed.
4. Initialize an empty set S_{new} that will contain the states of the non-deterministic automaton.
5. Initialize an empty set of edges E .
6. Consecutively perform the following steps on Queue S_{proc} , and stop if the queue gets empty:
 - (a) Fetch the first element of S_{proc} . We will refer to this state as the actual state s_a .
 - (b) Perform the following steps for each label l and for each set of states S' in the alternating automaton, such that S' satisfies the transition relation of all states in the alternating automaton identifying the actual state.
 - i. Add the actual state s_a to the set S_{new} .
 - ii. If the new state determined by S is not in set S_{new} make sure it is in the queue S_{proc} .
 - iii. Add the corresponding edge to E .
7. Create an accepting set S_{acc} including all states of S_{new} , for which all corresponding states in the original automaton are accepting.
8. The resulting automaton is determined as follows:
 - The state space equals S_{new} .

- The transition relation equals the edge list E .
- The initial states remain the same as with the original automaton.
- The accepting states equal the set S_{acc} .

The proposed algorithm implements the following mathematical concept of subset construction.

Proposition 4.1. *Given an alternating finite automaton $B_o = (\Sigma, S, \{s_0\}, \rho, F, \emptyset)$, one can build a non-deterministic finite automaton $B_n = (\Sigma, S', \{s_0\}, \rho', F', \emptyset)$ with $\mathcal{L}(B_n) = \mathcal{L}(B_o)$ according to the following construction:*

- $S' = 2^S$
- $\rho'(q, \sigma) = \{q' \in S' \mid q' \models \bigwedge_{s \in q} \rho(s, \sigma), \text{ and } q' \text{ is minimal}\}$
- $F' = \{q \in S' \mid q \subseteq F\}$

The following two examples shall illustrate why subset construction as proposed does work for instance for some weak alternating automata and does not for general alternating finite automata.

Example SC1: Subset construction performed on an alternating Büchi automaton (for infinite words).

Consider the automaton B_{SC1} in Figure 1 (p.34) and a Büchi acceptance condition with two accepting states $A = \{4, 5\}$. The automaton obviously accepts the language of all words beginning with a . Subset construction performed on this automaton leads to an automaton illustrated in Figure 2 (p.35) without any accepting state, thus an incorrect result recognizing not any single word.

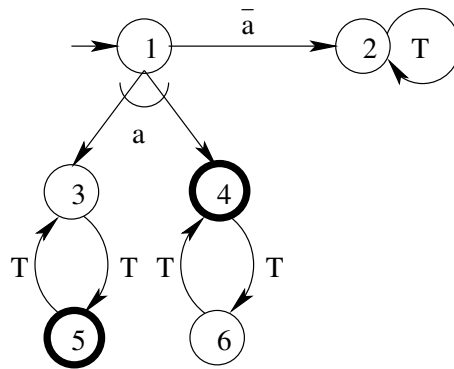


Figure 1: Alternating automaton B_{SC1}

Example SC2: Subset construction for a terminal weak alternating automaton for infinite words.

Consider the weak alternating automaton B_{SC2} in Figure 3 (p.35) featuring an acceptance set $A = \{5, 6\}$. The automaton obviously accepts all words beginning with a , as does the automaton in Example SC1. The subset construction performed on automaton B_{SC2} results in an automaton illustrated in Figure 4 (p.35) recognizing exactly the same language, thus providing a correct result.

Comparing the examples unveils the origin of the problem. Alternation in general may lead to automata runs in which each branch is accepting, while at the

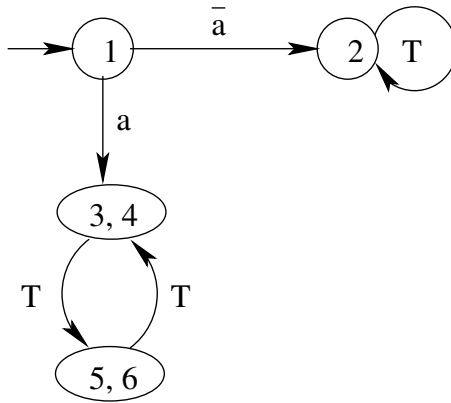


Figure 2: Subset Construction on B_{SC1}

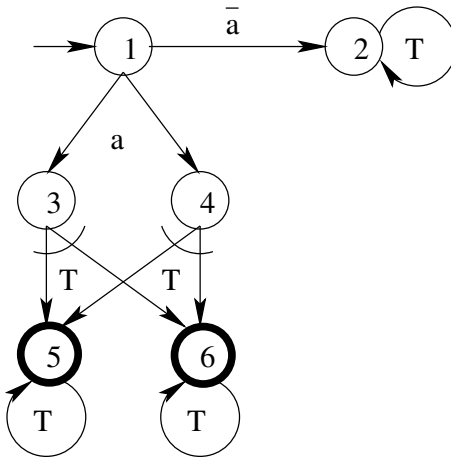


Figure 3: Weak alternating automaton B_{SC2}

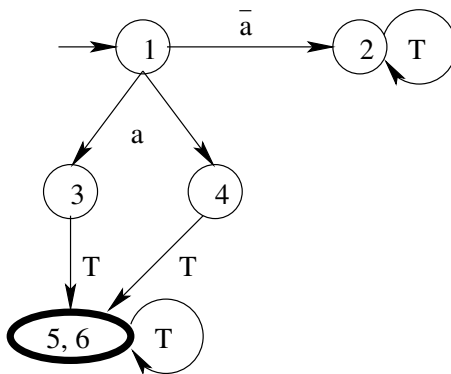


Figure 4: Subset Construction on B_{SC2}

same time, the simultaneously visited states may include accepting and some non-accepting states at all times. This does not suit the subset construction as proposed. For terminal alternating automata, however, we have that if the word is accepted, all runs eventually end up in a terminal accepting component. If it is not accepted, at least one branch will never visit an accepting component.

Concluding, the proposed subset construction for the elimination of universality introduces new states corresponding to sets of states which are visited simultaneously on a run of the original automaton. If all parallel visited states of the original

automaton are accepting, so is the combined state in the constructed one. If there is one single non-accepting state among the parallel visited ones, the combined one is non-accepting.

Removing Universality from One-Weak Automata

In this section we extend the concept of subset construction to work on one-weak automata by an enhanced acceptance management. To remove universal choice, we again use subset construction. Thus we get a nondeterministic automaton whose states represent the set of states the original, alternating automaton would be in. For the acceptance condition however, it no longer is enough to look at the set of states we visit concurrently as it was for AFAs. In opposite to finite words, we now might have accepting runs where the different branches are never in accepting states at the same time, but nevertheless each branch visits them infinitely often. Therefore we carefully have to track the success of each branch.

A run on our automaton is non-accepting, if one or more branches loops forever in a non accepting state. To capture this, we do not define the new accepting states in terms of the states they represent in the alternating automaton but by the transitions which were used to reach them as suggested in [17]. If a state is created by avoiding a self loop on s_i , we say it is good for s_i . An accepting SCC is good for all s_i , thus it is not necessary to limit its size to one element to calculate the corresponding predicate.

Because there are a number of states to avoid, we add a state variable that increases from i to $i + 1 \bmod n$ whenever we are not stuck in s_i . Thus, if a run does not get stuck in any s_i , this counter becomes zero infinitely often, and if the counter gets stuck at i , that means that the run has gotten stuck at s_i .

Definition 11. Let $B = (\Sigma, S, \{s_0\}, \rho, F, A)$ be a one-weak alternating Büchi automaton. Let $A^C = S \setminus A = \{s_1, \dots, s_n\}$ be the set of bad states, let a tuple $(s, t) \in S \times 2^S$ be good for a bad state b if $b \neq s$ or $b \notin t$, and let

$$\text{next}(c, ((s_1, S_1), \dots, (s_k, S_k))) = \begin{cases} c + 1 \bmod (k + 1) & \text{if } c = 0 \text{ or } \forall j, (s_j, S_j) \text{ is good for } b_c, \text{ and} \\ c & \text{otherwise.} \end{cases}$$

Define a NBA $B' = (\Sigma, T, \{s_0, 0\}, \rho', F', A')$

with

$$\begin{aligned} T &= 2^S \times [0, |A^C|] \\ \rho'((t, c), l) &= \{(t', c') \mid t = \{s_1, \dots, s_k\}, \exists S_1, \dots, S_k : \bigcup_i S_i = t', \forall i, S_i \models \rho(s_i, l), \\ &\quad \text{and } c' = \text{next}(c, ((s_1, S_1), \dots, (s_k, S_k)))\}, \\ A' &= 2^S \times \{0\} \\ F' &= 2^F \times [0, \dots, n] \end{aligned}$$

A run on the nondeterministic automaton B' only captures the set of states the alternating automaton would visit, but loses the structure of the original tree run. To capture this behaviour, we define the run on a directed acyclic graph.

Definition 12 (Run-DAG). A run on a directed acyclic graph for a run $\tau = \langle T, t \rangle$ is a structure $\tau' = \langle V, E, t' \rangle$ where

- $V = \{t \in T \mid \nexists t' \in T : |t| = |t'|, t' < t, t(t) = t(t')\}$
- $E = \{(t, t') \mid t, t' \in V, t \prec t', |t| + 1 = |t'|, \text{ or } \exists t'' \in T \setminus V : t(t') = t(t''), |t| + 1 = |t'| = |t''|, t \prec t''\}$
- $\forall t \in V : t(t)' = t(t)$

As a run-DAG only keeps the successors of one state per layer, we lose information in comparison to the tree run. Nevertheless, a recall of the fact that all branches of an accepting run are accepting leads us to the following observation.

We call τ a run-DAG for w if it is a run-DAG for some run for w . The acceptance criteria for a run-DAG are exactly the same as those for a run, but branches may now overlap.

Lemma 4.2. *If alternating automaton has a run for a word w iff it has a run-dag for w .*

Proposition 4.3. *The language of B is the same as the language of B' .*

Proof. \Rightarrow Let τ be a dag run of \mathcal{A} for a word w . Let S_l be the set of nodes on level l of the run

We have:

- $S_0 = \{s_0\}$
- $S_l = \{s_1, \dots, s_k\}$ and because of the definition of a run, there are S_{l1}, \dots, S_{ln} such that $S_l = \bigcup_i S_{li}$ and $\forall i, S_{li} \models \rho(s_i, w_l)$

Thus, there are i_j such that $\tau' = ((S_0, i_0), (S_1, i_1), \dots)$ is a run of A' for w

Now suppose that τ is accepting. If the $|w|$ is infinite, then there is no branch of τ that eventually gets stuck in a bad state, and for every bad state b there are infinitely levels l such that for all j , (s_j, S_j) is good. Thus, i infinitely often increases, infinitely often passes 0, and τ' is accepting. If $|w|$ is finite, then $S_{|w|}$ consists of only accepting states, so it is in F' , and τ' is accepting.

\Leftarrow Let $\tau = (S_0, \dots)$ be an accepting run of \mathcal{A}' for a word w . A dag run τ for A can be built by using the transitions (s_i, S_i) from the definition of A' . If τ' is accepting and infinite, on every level, then for all j , the counter c increases from j to $j + 1$, which means that the self loop on b_j is not used. Since all self loops on bad states are unused infinitely often, τ does not get stuck in a bad state, and is accepting. For finite runs, if τ' ends in a state $S_k \in F'$, then τ end in only states in F . \square

Removing Universality from Alternating Automata

For general automata, we have to remove alternation explicitly. The resulting automaton is converted to a symbolic representation in an extra step.

Proposition 4.4. *Let B be an alternating Büchi automaton on finite/infinite words with n states. There exists a Büchi automaton B_n on finite/infinite words with $2^{O(n)}$ states that accepts the same language.*

Proof. The proof follows the construction of Miyano and Hayashi [21] for the same proposition restricted to infinite words. Let $B = \langle \Sigma, S, \{s_0\}, \delta, F, A \rangle$ be an alternating Büchi automaton on finite/infinite words. We define the Büchi automaton (on finite/infinite words) $B_n = \langle \Sigma, S_n, \{(\{s_0\}, \{s_0\} \setminus A)\}, \delta_n, F_n, A_n \rangle$ as follows:

- $S_n \subseteq 2^S \times 2^S$ is the set of consistent pairs with respect to B , where a pair $(Q, P) \in 2^S \times 2^S$ is said to be consistent with respect to B if $P \subseteq Q \setminus A$.
- (Q', P') is in $\delta_n((Q, P), \sigma)$ iff $Q' \models \bigwedge_{s \in Q} \delta(s, \sigma)$ and either:
 - $P = \emptyset$ and $P' = Q' \setminus A$ or
 - $P \neq \emptyset$ and there exists a set $Y \subseteq Q'$ such that $Y \models \bigwedge_{s \in P} \delta(s, \sigma)$, and $P' = Y \setminus A$.
- $F_n = \{(Q, P) \in S_n \mid Q \subseteq F\}$.
- $A_n = \{(Q, P) \in S_n \mid P = \emptyset\}$.

Following [21], we have that for every infinite word w , $w \in \mathcal{L}(B)$ iff $w \in \mathcal{L}(B_n)$. Thus, we need to prove the following lemma:

Lemma 4.5. *Let w be a finite word. Then $w \in \mathcal{L}(B)$ iff $w \in \mathcal{L}(B_n)$.*

Proof. First direction: Let w be a finite word in $\mathcal{L}(B)$. We prove that $w \in \mathcal{L}(B_n)$. Let $\tau = \langle T, \mathfrak{t} \rangle$ be an accepting running tree of B on w . We define a running trace $(Q_0, P_0), (Q_1, P_1), \dots, (Q_{|w|}, P_{|w|})$ such that $Q_i = \{s \mid \mathfrak{t}(t) = s \text{ for some } t \in \tau \text{ with } |t| = i\}$, and $P_i = Q_i \setminus A$. We prove that this trace is an accepting running trace of B_n on w .

- Since $\mathfrak{t}(\varepsilon) = s_0$, we have that $(Q_0, P_0) = (\{s_0\}, \{s_0\} \setminus A)$ which is the initial state of B_n .
- Let $s \in Q_i$ for $i < |w|$. Then, there exists a node t of τ such that $\mathfrak{t}(t) = s$ and $|t| = i$. Let $X = \{\mathfrak{t}(t') \mid t' \text{ is a child of } t\}$. Then, $X \models \delta(s, w^i)$ and $X \subseteq Q_{i+1}$, thus $Q_{i+1} \models \delta(s, w^i)$. This implies that for every $s \in Q_i$ we have that $Q_{i+1} \models \delta(s, w^i)$. Since $P_i \subseteq Q_i$, we can always select $Y = Q_{i+1}$ and $P_{i+1} = Y \setminus A$. Then for every $s \in P_i$, $Y \models \delta(s, w^i)$. Thus, we have that $(Q_{i+1}, P_{i+1}) \in \delta_n((Q_i, P_i))$.
- Since τ is accepting, every branch of τ of depth $|w|$ ends in a node t such that $\mathfrak{t}(t) \in F$. This implies that every state $s \in Q_{|w|}$ is in F . Thus, $(Q_{|w|}, P_{|w|}) \in F_n$.

Second direction: Let $(Q_0, P_0), (Q_1, P_1), \dots, (Q_{|w|}, P_{|w|})$ be an accepting running trace of B_n on w . We define a tree $\tau = \langle T, \mathfrak{t} \rangle$ such that $\mathfrak{t}(\varepsilon) = s_0$ and every node t such that $|t| = i < |w|$ has $|Q_{i+1}|$ children such that for every $s \in Q_{i+1}$ the node t has a child t' with $\mathfrak{t}(t') = s$. We prove that τ is an accepting tree of B on w .

- s_0 is the initial state of B .
- Let t be a node of τ such that $|t| = i$ for $i < |w|$. Let $\mathfrak{t}(t) = s$. Then, $s \in Q_i$, and the definition of δ_n implies that $Q_{i+1} \models \delta(s, w^i)$.
Since $Q_{i+1} = \{\mathfrak{t}(t') \mid t' \text{ is a child of } t\}$,
we have that $\{\mathfrak{t}(t') \mid t' \text{ is a child of } t\} \models \delta(\mathfrak{t}(t), w^i)$.
- Since $Q_{|w|} \subseteq F_n$, we have that every branch of τ of depth $|w|$ ends in a node t such that $\mathfrak{t}(t) \in F$.

□

This completes the proof of the proposition. □

Symbolic Implementation

For model checking, we need the symbolic representation of a deterministic automaton. We show how to present the automata from Definition 11 and Proposition 4.4 as discrete transition system by introducing state variables which correspond directly to the states in the definition. We omit the description for the subset construction because it easily can be extracted from the definitions below (eg. by taking only the part for Q in Definition 15).

Definition 13 (DTS). *A discrete transition system is a symbolic representation of a finite automaton on finite or infinite words. A DTS $\mathcal{D} : \langle V, \Theta, \rho, A, J \rangle$ consists of the following components.*

- $V = \{u_1, \dots, u_n\}$: *A finite set of typed state-variables. We define a state s to be a type consistent interpretation of V , assigning to each variable $u \in V$ a value $s[u]$ in its domain.*
- Θ : *The initial condition. This is an assertion characterizing all the initial states of the DTS.*
- τ : *The transition relation. This is an assertion $\tau(V, V')$ relating a state s to its successor s' by relating to both unprimed and primed versions of the state variables.*
- A : *The accepting condition for finite words. This is an assertion characterizing all the accepting states for runs of the DTS satisfying finite words.*
- J : *The Büchi acceptance condition for infinite words as assertion characterizing the set of accepting states. For an infinite run to be accepting, it is required that the computation contains infinitely many states satisfying J .*

It is straightforward to encode the automaton \mathcal{A} from Definition 11 as DTS by using one binary variable for each state and input signal, and a counter for the bad states of size $ld(|S|)$.

Definition 14 (DTS for one weak automata).

- $V = \{s_0, \dots, s_n, a_0, \dots, a_m, c\}$ where there is one variable s_i with the domain $[0,1]$ for every state in the alternating automaton, a variable a_i with $[0,1]$ per possible input signal and a variable c with domain $[0, \dots, |A^C|]$ for the bad state counter.
- $\Theta = (s_0 \wedge \overline{s_1} \wedge \dots \wedge \overline{s_n} \wedge (c = 0))$
- We define τ in terms of the transition relation of the alternating automaton ρ . We use ρ' to emphasize that we write the result of the transition relation in terms of the primed state variables.

$$\tau = \bigwedge_{s \in S} \left(s \rightarrow \bigvee_{\sigma \in \Sigma} (\sigma \wedge \rho'(s, \sigma) \wedge \gamma_{s,c,\sigma}) \right) \wedge \varphi(c)$$

The outermost \wedge states that for a transition to take place, we have to follow the transitions of all involved states. The inner \vee demonstrates that we can choose which edge to take if it matches the input σ . Next states are given by τ . The selected edge also sets a bit $g_{s,c}$ which is used to set the counter c accordingly.

$$\gamma_{s,c,\sigma} = (g_{s,c} \leftrightarrow (s \neq b_c \vee \rho'(s, \sigma)[b_c \rightarrow \perp]))$$

The bit $g_{s,c}$ becomes true if our start state is not equal to the bad state $b_c \in A^C$ or we have the choice to avoid b_c in the next state. $\rho(s, \sigma)[b_c \rightarrow \perp]$ states that we set the value of b_c within the boolean expression returned by ρ to false. If we still can satisfy it, we also can avoid b_c . We control the value of c by:

$$\varphi(c) = \left(\bigwedge_{s \in S} g_{s,c} \wedge c' = c + 1 \right) \vee c' = c$$

- $A = (c = 0)$
- $J = \forall s \notin F : (s = 0)$

For alternating automata in general, we need two variables for each state resulting in an automaton of size $2^{O(2n)}$.

Definition 15 (DTS for alternating automaton).

- $V = \{s_{Q0}, \dots, s_{Qn}, s_{P0}, \dots, s_{Pn}, a_0, \dots, a_m\}$ with two variables s_{iQ} and s_{iP} with the domain $[0,1]$ for each state in the alternating automaton, and a variable a_i with $[0,1]$ for each possible input signal.
- $\Theta = s_{Q0} \wedge \overline{s_{Q1}} \wedge \dots \wedge \overline{s_{Qn}} \wedge \overline{s_{P0}} \wedge \dots \wedge \overline{s_{Pn}}$

- We have to maintain two sets: Q represents the states the alternating automaton would be in and is calculated in the obvious way by selecting the destinations of the old transition function. P represents states of branches which have not seen an accepting state in the alternating automaton since the last accepting state in the new automaton. The corresponding definition is split in two parts: If $P \neq \emptyset$ we demand the set of states which satisfy the transition relation without taking care of the accepting states ($\forall s \in A : s \rightarrow \top$ — all accepting states in the boolean expression are set to true). In case P is the empty set, all branches have seen an accepting state, thus we start over with the full set of successors of Q . Again, we use ρ' to emphasize that the result of the transition relation is written in terms of the primed state variables. (P' or Q')

$$\tau = \bigwedge_{s \in Q} \left(s \rightarrow \bigvee_{\sigma \in \Sigma} (\sigma \wedge \rho'(s, \sigma)) \right) \wedge \left(\bigvee_{s \in P} s \rightarrow \bigwedge_{s \in P} \left(s \rightarrow \bigvee_{\sigma \in \Sigma} (\sigma \wedge \rho'(s, \sigma) [\forall s \in A : s \rightarrow \top]) \right) \right) \wedge \left(\overline{\bigvee_{s \in P} s} \rightarrow \bigwedge_{s \in Q} \left(s \rightarrow \bigvee_{\sigma \in \Sigma} (\sigma \wedge \rho'(s, \sigma) [\forall s \in A : s \rightarrow \top]) \right) \right)$$

- We set a state accepting, if all branches of the run have seen an accepting state, thus P is the empty set.

$$A = \overline{s_{P0}} \wedge \dots \wedge \overline{s_{Pn}}$$

- $J = \forall s \notin F : (s = 0)$

5 Conclusions

In this paper we have provided a method to construct efficient automata for the PSL core language LTL_WR. We have shown that a specification written in LTL_WR can be translated into a nondeterministic Büchi automaton with a doubly-exponential blowup. One exponential blowup is necessary to handle the intersection operator “length-matching sequence conjunction” in the regular expressions, and affects only the parts of the formula that are sub-formulae of an intersection. In particular, in the absence of an intersection, the blowup is singly exponential.

Our construction goes through weak alternating automata. We have shown two methods to convert such automata to nondeterministic ones. The first method is an adaptation of Miyano and Hayashi’s approach to automata on finite and infinite words. This approach causes a 2^{2^n} blowup, and for the class of one-weak automata, we show that an $n \cdot 2^n$ blowup suffices.

We have also shown that the approaches to convert alternating to nondeterministic automata can be implemented symbolically. This means that the nondeterministic automata never needs to be built explicitly, and the inefficient step of re-encoding the state space is avoided.

6 References

- [1] Accellera. *Property Specification Language: Reference Manual*, 1.1 edition, June 2004.
- [2] Accellera Organization, Inc. Formal semantics of Accellera property specification language. In *Appendix B of <http://www.eda.org/vfv/docs/PSL-v1.1.pdf>*, pages 109–119, January 2004.
- [3] R. Armoni, D. Bustan, O. Kupferman, and M. Y. Vardi. Aborts vs resets in linear temporal logic. In *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2003.
- [4] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification language. In *TACAS*, pages 296–211, 2002.
- [5] I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In G. Berry, H. Comon, and A. Finkel, editors, *Proc. 13th International Conference on Computer Aided Verification (CAV)*, LNCS 2102, pages 363–367. Springer-Verlag, 2001.
- [6] S. Ben-David, D. Fisman, and S. Ruah. Embedding finite automata within regular expressions. In *1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*, Paphos, Cyprus, 2005.
- [7] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Proc. 11th International Conference on Computer Aided Verification (CAV)*, LNCS 1633, pages 222–235. Springer-Verlag, 1999.
- [8] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
- [9] D. Bustan, D. Fisman, and J. Havlicek. Automata construction for psl. Technical Report MCS05-04, The Weizmann Institute of Science, May 2005.
- [10] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
- [11] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design*, 10(1), February 1997.
- [12] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*, pages 249–260, London, UK, 1999. Springer-Verlag.
- [13] C. Eisner, D. Fisman, and J. Havlicek. A topological characterization of weakness. In *Proc 24th Annual ACM SIGACT-SIGOPS Symposium on Principles Of Distributed Computing (PODC05)*, Las-Vegas, Nevada, USA, July 2005. Springer Verlag.
- [14] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. Van Campenhout. Reasoning with temporal logic on truncated paths. In *The 15th international conference on computer aided verification (CAV)*, LNCS 2725, pages 27–40. Springer-Verlag, July 2003.
- [15] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. In *J. Comput. Syst. Sci.*, pages 18(2), 194–211, 1979.
- [16] A. Flaisher. Enhanced vacuity detection in linear temporal logic. Master's thesis, Israel Institute of Technology (The Technion), Haifa, Israel, August 2004.
- [17] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of the 13th Conference on Computer Aided Verification (CAV'01)*, number 2102 in Lecture Notes in Computer Science, pages 53–65. Springer Verlag, 2001.

- [18] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 3–18, London, UK, UK, 1996. Chapman & Hall, Ltd.
- [19] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.
- [20] H. R. Lewis, C. H. Papadimitriou, and C. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.
- [21] S. Miyano and T. Hayashi. Alternating finite automata on ω -words. *Theoretical Computer Science*, 32:321–330, 1984.
- [22] D. E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings of 3rd IEEE Symposium on Logic in Computer Science*, pages 422–427, 1988.
- [23] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
- [24] G. S. Rohde. *Alternating automata and the temporal logic of ordinals*. PhD thesis, University of Illinois, Urbana-Champaign, USA, 1997. Adviser-Paul E. Schupp.
- [25] S. Ruah, D. Fisman, and S. Ben-David. Automata construction for on-the-fly model checking psl safety simple subset, June 2005. IBM research report H-0234.
- [26] F. Somenzi and R. Bloem. Efficient Büchi automata from ltl formulae. In *CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification*, pages 248–263, London, UK, 2000. Springer-Verlag.
- [27] W. Thomas. Languages, automata, and logic. In *Handbook of formal languages, vol. 3: beyond words*, pages 389–455. Springer-Verlag New York, Inc., 1997.
- [28] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.