Symbolic Compilation of PSL

Alessandro Cimatti Marco Roveri Stefano Tonetta
Fondazione Bruno Kessler (FBK)
Institute for Scientific and Technological Research (IRST)
Via Sommarive 18, I-38050, Trento, Italy
{cimatti,roveri,tonettas}@fbk.eu

***Abstract*—**
The IEEE standard Property Specification Language (PSL) is increasingly used in many phases of the hardware design cycle, from specification to verification. PSL combines Linear Temporal Logic (LTL) with Sequential Extended Regular Expressions (SEREs), and thus provides a natural formalism to express all ω-regular properties.

In this paper, we propose a new method for efficiently converting PSL formulas into symbolically represented Nondeterministic (Generalized) Büchi Automata (NGBA), that are typically used in many verification and analysis tools. The construction is based on a normal form that separates the LTL and the SERE components, and allows for a modular and specialized encoding. The compilation is enhanced by a set of syntactic transformations that aim at reducing the state space of the resulting NGBA. These rules enable to achieve, at low cost, the simplification that can be achieved with expensive, semantic techniques based on minimization.

A thorough experimental analysis over large sets of paradigmatic properties (from patterns of properties commonly used in practice) shows that our approach drastically reduces the compilation time, and positively affects the overall search time.

***Index Terms*—PSL, SERE, Büchi Automata, Symbolic Compilation, Formal Verification, Formal Methods, BDD, SAT.**

I. Introduction

The IEEE standard Property Specification Language PSL [17], [28] is increasingly used as means to capture requirements on the behavior of a design, such as assumptions about the environment in which the design is expected to operate, internal behavioral requirements, and further constraints that arise during the design process from specification to verification.

The most important fragment of PSL combines Linear Temporal Logic (LTL) [36] with Sequential Extended Regular Expressions (SERE) [28], a variant of classical regular expressions [27]. This fragment, referred to as SERELTL in the rest of this paper, constitutes the core of many specification languages such as ForSpec [2] and SVA [47]. SERELTL enables encoding many properties of practical interest in compact and readable formulas, capturing the well known class of ω-regular properties. Most verification engines are in principle able to deal with ω-regular properties, by manipulating Nondeterministic (Generalized) Büchi Automata (NGBAs). However, only few model checkers are able to deal with SERELTL, and they usually cover only very limited fragments of the language, such as LTL or the PSL simple subset [17], [28]. Thus, the ability to convert SERELTL into NGBA would be an important enabling factor for the reuse of a large wealth of verification tools.

The main problem is that the translation of SERELTL into NGBA may become a bottleneck. In fact, the translation of LTL augmented with regular expressions into NGBA is exponential in the size of the input formula [3], [8]. In addition, SEREs extend regular expressions with the intersection operation, at the cost of another exponential blow-up [3].

In this paper, we propose a modular direct encoding of SERELTL into a symbolically represented NGBA, motivated by two main objectives: the first is to alleviate the theoretical blow-up of the translation by decomposing the formula into smaller components and exploiting the symbolic representation of their composition; the second is to apply syntactic simplification to the specifications as to avoid the explicit manipulation of automata.

The core of the algorithm is a normal form for SERELTL formulas that we named *Suffix Operator Normal Form* (SONF). This normal form separates the SERE components and the LTL components. This makes the approach modular, i.e., rather than constructing a monolithic automaton we generate it in the form of an implicit product, thus delaying composition until search time. The resulting overall automaton is the implicit symbolic composition of possibly smaller symbolic automata. In addition, the two kinds of components can be encoded separately. The encoding of the LTL components can rely on mature techniques (e.g., [23], [24], [39], [41]). Although the SERELTL components can be encoded by any standard conversion to NGBA, we exploit the fact that SONF admits only specific SERELTL patterns, called *Suffix Operator Subformulas*. For each pattern, we define a specific and optimized encoding into symbolic NGBAs.

In order to reduce the explicit manipulation of automata, we propose a number of syntactic rewriting rules, based on the following ideas. First, we try to minimize the size of the arguments to the SERE language-intersection operators, given that they are associated with an exponential blow up. Second, whenever possible we convert the SEREs into LTL, in order to limit as much as possible suffix operators, and to enable the use of specialized algorithms for LTL. Finally, some Suffix Operator Subformulas resulting from the conversion into SONF can be further simplified by taking into account their structure.

We prove that our transformation is correct, and evaluate our approach in the NuSMV model checker [9], with a thorough

Some of the material presented in this paper appears in preliminary version in the conference papers [11] and [13]. The current manuscript presents a new and more extensive experimental evaluation, contains a uniform presentation of the proposed techniques, and proves their correctness.

experimental analysis over large sets of paradigmatic PSL properties [4]. The analysis shows that our approach dramatically reduces the construction time of the symbolic NGBA. In addition, an evaluation on language-emptiness problems, using both `BDD`s and `SAT`, shows that our approach can positively affect the overall verification time. The experiments also show that the simplifications are computationally cheap, and substantially pay off in terms of verification time. The result is that, overall, the new method is vastly superior to the approach described in [3], [6].

The paper is structured as follows. In Section II, we present the syntax and semantics of SERELTL, some background on automata and on state-of-the-art symbolic techniques for compiling SERELTL formulas. In Section III, we outline our approach to SERELTL symbolic compilation. In Section IV, we define the Suffix Operator Normal form, and, in Section V, we discuss the encoding, and we show how to exploit the structure of the SONF to generate symbolically represented NGBAs. In Section VI, we present the set of syntactic optimizations. In Section VII, we discuss some related work. In Section VIII, we experimentally evaluate our approach. Finally, in Section IX, we draw the conclusions.

II. Technical Background

A. PSL *and* SERELTL

The fragment of PSL we deal with is based on a combination of operators from Linear Temporal Logic [36] (LTL), and Sequential Extended Regular Expressions (SEREs), a variant of classical regular expressions [27]. This combination has been studied in the literature under the name of SERELTL, and provides ω-regular expressiveness [31]. In the following, we will identify the relevant fragment of PSL with SERELTL.We will not deal with "clocked" expressions, since any clocked expression can be rewritten into an equivalent un-clocked one [28]. The same applies for the "abort" operator, which can be efficiently rewritten into an abort-free equivalent formula [1], [37].

1) Syntax: Given a generic set V, we denote with $\mathcal{B}^{+}(V)$ the set of Boolean formulas obtained by applying only disjunction ($\vee$) and conjunction ($\wedge$) to elements in $V \cup \{true, false\}$; with $\mathcal{B}^{\vee}(V)$ the set of Boolean formulas obtained by applying only disjunction $\vee$ to elements in $V \cup \{true, false\}$; and with $\mathcal{B}^{\neg}(V)$ the set of Boolean formulas obtained by applying disjunction, conjunction and negation ($\neg$) to elements in $V \cup \{true, false\}$. In the following, we assume as given a set $\mathcal{A}$ of atomic propositions.

*Definition 1 (*SERE*s syntax):*

- if $b \in \mathcal{B}^{\neg}(\mathcal{A})$, then b is a SERE;
- if r is a SERE, then r**[*]** is a SERE;
- if r_1 and r_2 are SEREs, then r_1 **;** r_2, r_1 **:** r_2, r_1 **|** r_2, r_1 **&** r_2, and r_1 **&&** r_2 are SEREs.

; and **:** are concatenation operators: the former is the classical regular expression concatenation [27]; the latter is a variant with one-letter overlapping. **&** and **&&** are intersection operators, the former without synchronization, the latter with synchronization. **|** and **[*]** are respectively the union and Kleene-closure standard regular expression operators. We use r**[***n**]** as an abbreviation for r **;** r **;** $\ldots$ **;** r repeated n times.

*Definition 2 (*SERELTL *syntax):* We define the SERELTL formulas over $\mathcal{A}$, as follows:

- if $p \in \mathcal{A}$, p is a SERELTL formula;
- if ϕ_1 and ϕ_2 are SERELTL formulas, then $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ are SERELTL formulas;
- if ϕ_1 and ϕ_2 are SERELTL formulas, then **X** ϕ_1, ϕ_1 **U** ϕ_2, ϕ_1 **R** ϕ_2 are SERELTL formulas;
- if r is a SERE and ϕ is a SERELTL formulas, then $r \diamond\!\!\rightarrow \phi$ and $r \mapsto \phi$ are SERELTL formulas;

The **X** ("next-time"), the **U** ("until"), and the **R** ("releases") operators are the standard *temporal operators* from Linear Temporal Logic. We call the $\diamond\!\!\rightarrow$ ("suffix conjunction"), and the $\mapsto$ ("suffix implication"), *suffix operators*. We also use **G** ϕ as an abbreviation for $false$ **R** ϕ and a SERE r not occurring in a suffix operator as an abbreviation for $r \diamond\!\!\rightarrow true$. LTL can be seen as a subset of SERELTL in which the suffix operators and the SEREs are suppressed.

In the following we write $\phi(\psi)$ to state that formula ψ is a subformula of ϕ. We also denote with $\phi[P/\psi]$ the formula resulting from the substitution of every occurrence of ψ in ϕ with P.

Definition 3: Let ϕ be a SERELTL formula and ψ a SERELTL subformula of ϕ. We define the positive [resp., negative] occurrences of ψ in ϕ recursively as follows:

- ϕ is a positive occurrence of ϕ in ϕ
- every positive [resp., negative] occurrence of ψ in ϕ is a negative [resp., positive] occurrence in $\neg\phi$
- every positive [resp., negative] occurrence of ψ in ϕ is a positive [resp., negative] occurrence in **X** ϕ, $\phi' \wedge \phi$, $\phi \wedge \phi'$, $\phi' \vee \phi$, $\phi \vee \phi'$, ϕ' **U** ϕ, ϕ **R** ϕ', $r \mapsto \phi$, and $r \diamond\!\!\rightarrow \phi$

Note that the definition does not consider subformulas of SEREs.

2) Semantics: The languages we are considering are defined as sets of finite or infinite words over the alphabet $\Sigma_{\mathcal{A}} := 2^{\mathcal{A}}$ (total assignments to $\mathcal{A}$). In the following, when clear from the context, we simply write Σ, thus omitting the set $\mathcal{A}$ of atomic propositions. We denote a letter from Σ by ℓ, a word from Σ by v or w, and the concatenation of v and w by vw. We denote with $|w|$ the length of word w. We denote with ϵ the word with length 0. A finite word $w = \ell_0\ell_1 \ldots \ell_{n-1}$ has length n, an infinite word has length ω. Σ^* denotes the set of finite words, while Σ^ω denotes the set of infinite words. For all i, $0 \leq i < |w|$, we use w^i to denote the $(i+1)^{th}$ letter of w, and we denote with $w^{i..}$ the suffix of w starting at w^i. When $0 \leq i \leq j < |w|$, we denote with $w^{i..j}$ the finite sequence of letters starting from w^i and ending in w^j, i.e., $w^{i..j} := w^i w^{i+1} \ldots w^j$.

Boolean expressions are interpreted over letters in Σ: a propositional atom p is true in ℓ iff $p \in \ell$, false otherwise; Boolean connectives are interpreted in the standard way. If ℓ is a letter and b a Boolean expression, we denote with $\ell \models_B b$ the fact that ℓ is a model of b.

The semantics of SEREs is defined over *finite* words:

*Definition 4 (*SERE*s semantics):* Let $w \in \Sigma^*$.

- $w \models b$ iff $|w| = 1$ and $w^0 \models_B b$

- $w \models r_1 \,\mathbf{;}\, r_2$ iff $\exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r_1$, $w_2 \models r_2$
- $w \models r_1 \,\mathbf{:}\, r_2$ iff $\exists w_1, w_2, \ell$ s.t. $w = w_1 \ell w_2$, $w_1 \ell \models r_1$, $\ell w_2 \models r_2$
- $w \models r_1 \,\mathbf{|}\, r_2$ iff $w \models r_1$ or $w \models r_2$
- $w \models r_1 \,\mathbf{\&}\, r_2$ iff
  - $w \models r_1$ and $\exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r_2$, or
  - $w \models r_2$ and $\exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r_1$
- $w \models r_1 \,\mathbf{\&\&}\, r_2$ iff $\exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r_1$, $w_1 \models r_2$
- $w \models r\mathbf{[*]}$ iff $|w| = 0$ or $\exists w_1, w_2$ s.t. $|w_1| \neq 0$, $w = w_1 w_2$, $w_1 \models r$, $w_2 \models r\mathbf{[*]}$

Definition 5 (Language of SEREs*):* The language of a SERE r is defined as $\mathcal{L}(r) := \{w \in \Sigma^* \mid w \models r\}$.

We interpret SERELTL expressions over *infinite* words and we assume a strong semantics for all operators (though our approach can be easily extended to deal with the weak semantics and with finite words, see [18] for a discussion).

*Definition 6 (*SERELTL *semantics):* Let $w \in \Sigma^\omega$.

- $w \models p$ iff $w^0 \models_B p$;
- $w \models \neg\phi$ iff $w \not\models \phi$;
- $w \models \phi \wedge \psi$ iff $w \models \phi$ and $w \models \psi$;
- $w \models \phi \vee \psi$ iff either $w \models \phi$ or $w \models \psi$;
- $w \models \mathbf{X}\,\phi$ iff $w^{1..} \models \phi$;
- $w \models \phi\,\mathbf{U}\,\psi$ iff, for some $j \geq 0$, $w^{j..} \models \psi$ and, for all $0 \leq k < j$, $w^{k..} \models \phi$;
- $w \models \phi\,\mathbf{R}\,\psi$ iff, for all $j \geq 0$, either $w^{j..} \models \psi$ or, for some $0 \leq k < j$, $w^{k..} \models \phi$;
- $w \models r \diamond\!\!\rightarrow \phi$ iff, for some $j \geq 0$, $w^{0..j} \models r$ and $w^{j..} \models \phi$;
- $w \models r \mapsto \phi$ iff, for all $j \geq 0$, if $w^{0..j} \models r$, then $w^{j..} \models \phi$.

Example 1: Consider the SERELTL formula $\mathbf{G}\,(\{\{a \,\mathbf{;}\, b\mathbf{[*]} \,\mathbf{;}\, c\} \,\mathbf{\&\&}\, \{d\mathbf{[*]} \,\mathbf{;}\, e\}\} \mapsto \{f \,\mathbf{;}\, g\})$. It encodes the property for which every sequence that matches both regular expressions $\{a \,\mathbf{;}\, b\mathbf{[*]} \,\mathbf{;}\, c\}$ and $\{d\mathbf{[*]} \,\mathbf{;}\, e\}$ must be followed by $\{f \,\mathbf{;}\, g\}$ with f overlapping with c and e.

Definition 7 (Language of SERELTL *formulas):* The language of a SERELTL formula ϕ over the alphabet Σ is defined as follows:

$$\mathcal{L}(\phi) := \{w \in \Sigma^\omega \mid w \models \phi\}$$

For any subset $\mathcal{A}' \subseteq \mathcal{A}$ of propositions, we define the language projected on $\mathcal{A}'$ as:

$$\begin{aligned}\mathcal{L}_{\mathcal{A}'}(\phi) \quad := \quad & \{w \in \Sigma^\omega_{\mathcal{A}'} \mid \text{for some } v \in \mathcal{L}(\phi), \\ & v^i \cap \mathcal{A}' = w^i \text{ for all } i \geq 0\}\end{aligned}$$

B. Automata

Definition 8 (NFA): A Non-deterministic Finite-state Automaton (NFA) is a tuple $A = \langle \Sigma, Q, q_0, \rho, F\rangle$, where

- Σ is the alphabet;
- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $\rho : Q \times \Sigma \rightarrow 2^Q$ is the transition function;
- $F \subseteq Q$ is the set of final states.

A *run* of an NFA A over the finite word $w = \ell_0, \ell_1, ..., \ell_{n-1} \in \Sigma^*$ is a finite sequence of states $\pi = q_0, q_1, ..., q_n$ such that, for $0 \leq i < n$, $q_{i+1} \in \rho(q_i, \ell_i)$, and $q_n \in F$.

Definition 9 (Language of NFAs): The language $\mathcal{L}(A)$ of an NFA A is the set of words w such that there exists a run of A over w.

Definition 10 (DFA): A Deterministic Finite-state Automaton (DFA) is an NFA $A = \langle \Sigma, Q, q_0, \rho, F\rangle$ such that, for all $q \in Q$ and $\ell \in \Sigma$, $|\rho(q, \ell)| \leq 1$.

Definition 11: A (Non-)Deterministic Finite-state Automaton is *complete* iff for all $q \in Q$, for all $\ell \in \Sigma$, $|\rho(q, \ell)| \geq 1$.

Definition 12 (NGBA): A Non-deterministic Generalized Büchi Automaton (NGBA) is a tuple $A = \langle \Sigma, Q, Q_0, \rho, \mathcal{F}\rangle$, where

- Σ is the alphabet;
- Q is a finite set of states;
- $Q_0 \subseteq Q$ is the set of initial states;
- $\rho : Q \times \Sigma \rightarrow 2^Q$ is the transition function;
- $\mathcal{F} \subseteq 2^Q$ is the set of fairness conditions.

A *run* of an NGBA A over the infinite word $w = \ell_0, \ell_1, ... \in \Sigma^\omega$ is an infinite sequence of states $\pi = q_0, q_1, ...$ starting from some $q_0 \in Q_0$ such that, for all $i \geq 0$, $q_{i+1} \in \rho(q_i, \ell_i)$, and for all $F \in \mathcal{F}$, for infinitely many $i \geq 0$, $q_i \in F$.

Definition 13 (ABA): An Alternating Büchi Automaton (ABA) over infinite words is a tuple $A = \langle \Sigma, Q, Q_0, \rho, F\rangle$, where

- Σ is the alphabet;
- Q is a finite set of states;
- $Q_0 \subseteq Q$ is the set of initial states;
- $\rho : Q \times \Sigma \rightarrow \mathcal{B}^+(Q)$ is the transition function;
- $F \subseteq Q$ is the fairness condition.

A *run* of an ABA on an infinite word w is a (possibly infinite) Q-labeled tree $\tau = (\mathcal{T}, L)$ such that $L(\epsilon) \in Q_0$ and for every node $t \in \mathcal{T}$, t has at most $|Q|$ children and, if t is at the i-th level of τ, $L(t) = q$, and the children of t are $t_1, ..., t_k$, then $L(t_1), ..., L(t_k)$ satisfy $\rho(q, w^i)$, and every branch has infinite depth and features infinitely many labels in F.

Definition 14 (ABA and NGBA language): The language $\mathcal{L}(A)$ of an ABA or NGBA A is the set of words w such that there exists a run of A over w.

Fair Transition Systems [32] are a symbolic representation of automata.

Definition 15 (FTS): A *Fair Transition System (FTS)* is a tuple $S = \langle V, W, T, I, \mathcal{F}\rangle$, where V is a finite set of *state variables*, W is a finite set of *input variables*, $T \in \mathcal{B}^\neg(V \cup W \cup V')$ is the *transition relation* (V' is the set of primed versions of variables in V), $I \in \mathcal{B}^\neg(V)$ specifies the set of *initial states*, and $F \in \mathcal{B}^\neg(V)$ specifies a *fairness condition* for each $F \in \mathcal{F}$. The size $|S|$ of S is given by the number of variables ($|V|$).

An FTS $S = \langle V_s, W_s, T_s, I_s, \mathcal{F}_s\rangle$ defines an NGBA A as follows:$A = \langle 2^{W_s}, 2^{V_s}, Q_0, \rho, \mathcal{F}\rangle$, where $Q_0 = \{q \mid q \models I_s\}$, $\rho(q, \ell) = \{q' \mid (q, \ell, q') \models T\}$, and $\mathcal{F}$ contains a fairness condition $F = \{q \mid q \models F_s\}$ for each $F_s \in \mathcal{F}_s$. Thus, we can speak of a run of an FTS and the language of an FTS as if it were an NGBA.

Let $S_1 = \langle V_1, W, T_1, I_1, \mathcal{F}_1\rangle$ and $S_2 = \langle V_2, W, T_2, I_2, \mathcal{F}_2\rangle$ be two FTSs over the same alphabet, such that V_1 and V_2 are disjoint. The *Synchronous Product* of S_1 and S_2, $P = \langle V_P, W, T_P, I_P, \mathcal{F}_P\rangle$, is defined as follows:

- $V_P = V_1 \cup V_2$,
- $T_P = T_1 \wedge T_2$,
- $I_P = I_1 \wedge I_2$,
- $\mathcal{F}_P = \mathcal{F}_1 \cup \mathcal{F}_2$.

The synchronous product corresponds to language intersection, i.e. $\mathcal{L}(P) = \mathcal{L}(S_1) \cap \mathcal{L}(S_2)$.

C. From SERELTL *to FTS*

The process of translating a formula into an FTS is called *Symbolic Compilation*. In this section, we list a series of known theorems that allow for the symbolic compilation of SERELTL in different ways.

Regular expressions, NFAs and DFAs have the same expressive power [27]. Since SERE extends the regular expressions with intersection operations, the following theorem holds:

Theorem 1 (From SERE *to NFA):* For every SERE r, there exists a complete NFA A_r such that $\mathcal{L}(A_r) = \mathcal{L}(r)$ and $|A_r| = |r|$ if r does not contain the **&&** operator, $|A_r| = 2^{O(|r|)}$ otherwise.

The determinization of an NFA results in an exponential blow up [27].

Theorem 2 (From NFA to DFA): Given an NFA A, we can build a DFA A' such that $\mathcal{L}(A') = \mathcal{L}(A)$ and $|A'| = 2^{O(|A|)}$.

SERELTL formulas can be translated both to NGBA and ABA. In [8], it is shown how to translate LTL extended with regular expressions into NGBA. Given Theorem 1, the following holds:

Theorem 3 (From SERELTL *to NGBA):* For every SERELTL formula ϕ, there exists an NGBA A_ϕ such that $\mathcal{L}(A_\phi) = \mathcal{L}(\phi)$ and $|A_\phi| = 2^{2^{O(|\phi|)}}$.

In [3], it is shown that:

Theorem 4 (From SERELTL *to ABA):* For every SERELTL formula ϕ, there exists an ABA A_ϕ such that $\mathcal{L}(A_\phi) = \mathcal{L}(\phi)$ and $|A_\phi| = 2^{O(|\phi|)}$.

Given an ABA A, the algorithm by Miyano and Hayashi [33] produces an NGBA B accepting the same language. In the following, we rely on a simplified version [29]:

Theorem 5 (From ABA to NGBA): For any ABA A there exists an NGBA B such that $\mathcal{L}(B) = \mathcal{L}(A)$. Given $A = \langle \Sigma, Q, Q_0, \rho, F \rangle$, B is defined as $B = \langle \Sigma, Q_B, Q_{0B}, \rho_B, \{F_B\} \rangle$, where:

- $Q_B = \{(L, R) | L \in 2^Q, R \in 2^{Q \setminus F}\}$
- $Q_{0B} = Q_0 \times \{\emptyset\}$
- if $R \neq \emptyset$, then $\rho_B((L,R),\ell) = \{(L', R' \setminus F) \mid L' \models \bigcap_{q \in L} \rho(q,\ell), R' \subseteq L', R' \models \bigcap_{q \in R} \rho(q,\ell)\}$
  if $R = \emptyset$, then $\rho_B((L,R),\ell) = \{(L', L' \setminus F) \mid L' \models \bigcap_{q \in L} \rho(q,\ell)\}$
- $F_B = 2^Q \times \{\emptyset\}$

The following is usually called *logarithmic encoding*:

Theorem 6 (From NGBA to FTS): For every NGBA A, there exists an FTS B such that $\mathcal{L}(A) = \mathcal{L}(B)$ and $|B| = log(O(|A|))$.

A symbolic variant of [33] is presented in [6]:

Theorem 7 (From ABA to FTS): For every ABA A, there exists an FTS B such that $\mathcal{L}(A) = \mathcal{L}(B)$ and $|B| = O(|A|)$.

Finally,

Theorem 8 (From SERELTL *to FTS):* For every SERELTL formula ϕ, there exists an FTS B_ϕ such that $\mathcal{L}(B_\phi) = \mathcal{L}(\phi)$ and $|B_\phi| = 2^{O(|\phi|)}$.

D. State-of-the-art Symbolic Compilers for LTL *and* SERELTL *into FTS*

We can identify two main approaches to LTL compilation into FTS: on one hand, *syntactic compilers* (such as `ltl2smv` [14]) translate the LTL formula directly into a linear size FTS by introducing one variable for each sub-formula; on the other hand, *semantic compilers* (such as `Wring` [41]) translate the formula into an intermediate explicit representation, which is optimized and then symbolically represented by means of a logarithmic encoding. Semantic compilers have been highly optimized and focus most of their efforts in the minimization of the explicit-state automata. LTL formulas can be translated linearly into a fragment of ABA, called linear weak ABA [24], [43]. The ABA can be exploited for a further intermediate minimization [23]. Also the classic syntactic compilation ([14]) can be seen as a linear encoding of the corresponding ABA, but typically the ABA is not explicitly built and minimized. In [39], it is shown that semantic compilers usually performs better in terms of verification time; though, their main drawback is that the automata optimization are often so expensive that the compilation time may result in a bottleneck.

The standard approach to the compilation of SERELTL extends the semantic compilation of LTL to handle SERE and their combination with temporal connectives. The translation proposed in [3] proceeds bottom-up: it builds the automaton corresponding to the SEREs, and combines them with the temporal and the suffix operators. The resulting automaton is an ABA which is then translated into an NGBA by the Miyano-Hayashi (MH) algorithm [33]. Finally, the NGBA can be converted into an FTS. In [6], this compilation is improved by providing a symbolic version of MH, so that the potential explosion is delayed at run time.

The main obstacles to an efficient symbolic compilation of SERELTL are the following:

1) The first bottleneck is in the translation of SEREs into NFAs. While the standard translation for concatenation (**;**, **:**) and union (**|**) is linear [27], the intersection operator **&&** allows for a greater succinctness, but implies a potential blow-up. The complexity stems from the difficulty of codifying the fact that two automata must synchronize on the end of the accepted words [30], [49].
2) A second bottleneck is in the suffix implication $r \mapsto \phi$, which requires that every finite prefix satisfying the SERE r is followed by a suffix satisfying the SERELTL formula ϕ. This can be guaranteed by determinizing and completing the NFA, thus forcing the search to consider exactly one path per prefix. Theoretical translations such as [8] avoid this determinization, but in order to define the transition relation they explicitly enumerate the letters of the alphabet, which is exponential in the number of atomic propositions.

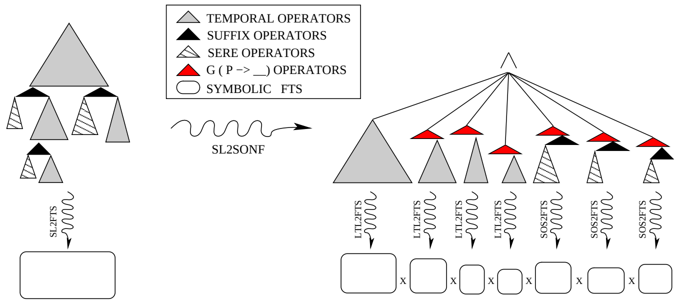


Fig. 1. The core of the approach.

III. The Approach: Overview

In this section, we give an overview the approach proposed in this paper. The main underlying ideas are the following. First, the original formula is transformed into a normal form that separates out the LTL and SERE components and enables for a modular encoding (see Figure 1). The final automaton is the (implicit) synchronous composition of the encoding of smaller components. Second, the SERE components are rewritten and simplified in order to maximize the parts that can be expressed in LTL, as to exploit off-the-shelf techniques for LTL and thus reducing the explicit manipulation of automata required for the SERE components.

The algorithm is based on the following three main steps: 1) formula simplification, 2) formula normalization (and further simplification), 3) final encoding.

The core of the translation is the reduction of SERELTL to a normal form, called *Suffix Operator Normal Form* (described in Section IV). The original SERELTL formula, where LTL and SERE components are arbitrarily mixed according to the SERELTL syntax, is decomposed into the conjunction of a set of pure LTL subformulas, and of a set of *Suffix Operator Subformulas*, i.e., invariant formulas of a precise structure where only one suffix operator and one SERE occur. In order to achieve this normal form, new variables are introduced as place-holders for some subformulas. The same variables are used at top-level to trigger the satisfaction of the corresponding formulas.

The translation takes advantage of this form in order to create a modular encoding (described in Section V). Each of the conjuncts resulting from the translation is encoded separately, and the results are recombined by means of an implicit, symbolic synchronous composition. The encoding of each component is also optimized based on the structure. LTL components are translated by means of LTL to FTS techniques. For Suffix Operator Subformulas, the encoding into automata can use any existing SERELTL to automata technique or can be further specialized (as described in Section V-A and in Section V-B).

The rewritings and the simplifications (described in Section VI) that are carried out on the original formula and on the result of the normalization aim at reducing the scope of the **&&** operator in the SEREs, and at translating most of the SERE components into LTL.

IV. Suffix Operator Normal Form for SERELTL

In this section, we define the *Suffix Operator Normal Form* for SERELTL. The first step is to extend the Negative Normal Form (standard for LTL) to the case of SERELTL.

Definition 16 (NNF): A SERELTL formula is in *negated normal form* (NNF) iff all the negations occur only in front of propositions.

The following two rules need to be added to the standard rules $\mathcal{N}(\cdot)$ for obtaining an NNF from an LTL formula:

- $\mathcal{N}(\neg(r \Diamond\!\!\!\rightarrow \phi_1)) := r \mapsto \mathcal{N}(\neg\phi_1)$
- $\mathcal{N}(\neg(r \mapsto \phi_1)) := r \Diamond\!\!\!\rightarrow \mathcal{N}(\neg\phi_1)$

Lemma 1 (NNF-ization): A SERELTL formula ϕ can always be translated to an NNF SERELTL formula $\mathcal{N}(\phi)$ such that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{N}(\phi))$ and $|\mathcal{N}(\phi)| = |\phi|$.

We now provide a set of rewriting rules to convert the formula into a normal form named *Suffix Operator Normal Form* (SONF). Intuitively, a formula in SONF is structured as follows

$$\overbrace{\bigwedge_i \phi_i}^{\Psi_{LTL}} \wedge \overbrace{\bigwedge_j \mathbf{G}\,(p_I^j \rightarrow (r_j \star\!\!\!\rightarrow p_F^j))}^{\Psi_{SOS}}$$

where ϕ_i are arbitrary LTL formulas, r_j are SEREs, p_I^j and p_F^j are atoms, and $\star\!\!\!\rightarrow \in \{\mapsto, \Diamond\!\!\!\rightarrow\}$.

Given a formula ϕ, the rewriting rules build ϕ', adding new atoms while preserving the language so that a model of ϕ' restricted to the original set of atomic propositions is a model of ϕ. We define the following transformation function:

Definition 17:

$$\mathcal{S}(\phi) := \begin{cases} \mathcal{S}(\phi[P_{r \Diamond\!\!\rightarrow \psi}/r \Diamond\!\!\!\rightarrow \psi]\wedge & \text{if } r \Diamond\!\!\!\rightarrow \psi \text{ occurs} \\ \quad \mathbf{G}\,(P_{r \Diamond\!\!\rightarrow \psi} \rightarrow (r \Diamond\!\!\!\rightarrow P_\psi))\wedge & \text{in } \phi \text{ and } \psi \text{ is not} \\ \quad \mathbf{G}\,(P_\psi \rightarrow \psi) \qquad) & \text{atomic;} \\ \mathcal{S}(\phi[P_{r \mapsto \psi}/r \mapsto \psi]\wedge & \text{if } r \mapsto \psi \text{ occurs} \\ \quad \mathbf{G}\,(P_{r \mapsto \psi} \rightarrow (r \mapsto P_\psi))\wedge & \text{in } \phi \text{ and } \psi \text{ is not} \\ \quad \mathbf{G}\,(P_\psi \rightarrow \psi) \qquad) & \text{atomic;} \\ \phi & \text{otherwise.} \end{cases}$$

Intuitively, for every subformula of ϕ of the form $r \star\!\!\!\rightarrow \psi$ with ψ not atomic, we introduce two new atoms: $P_{r \star\!\!\rightarrow \psi}$ and P_ψ. We substitute the suffix operator $r \star\!\!\!\rightarrow \psi$ with the corresponding activation predicate $P_{r \star\!\!\rightarrow \psi}$, and we add two global formulas at top-level: the first states that $P_{r \star\!\!\rightarrow \psi}$ always triggers $r \star\!\!\!\rightarrow P_\psi$; the second states that P_ψ always triggers ψ. We call formulas of the form $\mathbf{G}\,(P_{r \star\!\!\rightarrow \psi} \rightarrow (r \star\!\!\!\rightarrow P_\psi))$ *Suffix Operator Subformulas* (*SOS*).

Notice that $|\phi'| = O(|\phi|)$. The function $\mathcal{S}$ is well defined because the recursive definition always decreases the number of subformula of ϕ of the form $r \star\!\!\!\rightarrow \psi$ with ψ not atomic. Thus, the rewriting procedure terminates and it always produces the same formula (modulo renaming). The following theorem guarantees that the rewriting rules preserve the language with respect to the original alphabet.

Theorem 9: Let ϕ be a SERELTL formula over $\mathcal{A}$ in NNF, let P be a variable not occurring in ϕ, and let ψ be a SERELTL subformula of ϕ that occurs only positively in ϕ. If $\phi' := \phi[P/\psi] \wedge \mathbf{G}\,(P \rightarrow \psi)$ then $\mathcal{L}_{\mathcal{A}}(\phi) = \mathcal{L}_{\mathcal{A}}(\phi')$.

Example 2: The SONF of the formula of Example 1 $\mathbf{G}\,(\{\{a\ ;\ b[^*]\ ;\ c\}\ \&\&\ \{d[^*]\ ;\ e\}\} \mapsto \{f\ ;\ g\})$ is $\mathbf{G}\,p_1 \wedge$

$\mathbf{G}\,(p_1 \rightarrow \{\{a \,\mathbf{;}\, b\mathbf{[*]} \,\mathbf{;}\, c\} \,\mathbf{\&\&}\, \{d\mathbf{[*]} \,\mathbf{;}\, e\}\} \mapsto p_2) \wedge \mathbf{G}\,(p_2 \rightarrow \{f \,\mathbf{;}\, g\} \Diamond\!\!\rightarrow p_3)$.

Example 3: The SONF of $\mathbf{F}\,\mathbf{G}\,(\{(a \wedge b)\mathbf{[*]} \,\mathbf{;}\, (c \vee d)\} \mapsto \mathbf{F}\,c)$ is $\mathbf{F}\,\mathbf{G}\,(p_1) \wedge \mathbf{G}\,(p_1 \rightarrow \{(a \wedge b)\mathbf{[*]} \,\mathbf{;}\, (c \vee d)\} \mapsto p_2) \wedge \mathbf{G}\,(p_2 \rightarrow \mathbf{F}\,c)$.

V. A Modular Translation from SERELTL to FTS

Algorithm 1 depicts the pseudo code of the `ModSL2Fts` procedure for translating SERELTL to FTS in a modular manner. `ModSL2Fts` relies on two building blocks, `Ltl2Fts` and `Sos2Fts`. `Ltl2Fts` is a procedure that builds an FTS from an LTL formula (for instance [14]). `Sos2Fts` can be implemented by any procedure `SL2Fts` that builds an FTS from a SERELTL formula (like e.g. the procedure described in Section II-D).

The `ModSL2Fts` procedure improves over `SL2Fts` by transforming the SERELTL formula into an equivalent one in SONF. The `Sonf` procedure generates the SONF of a formula ϕ, thus decomposing it into subformulas according to their nature. This normalization is then exploited in two ways: first, we keep the resulting FTS partitioned (rather than monolithically); second, we call the tableau constructor `Ltl2Fts`, which is optimized for LTL, on the LTL part.

```
ModSL2Fts($\phi$)
input  : $\phi$, the SERELTL input formula
output : $S$, a set of FTSs;
         the final FTS is the product of all FTSs in $S$
begin
    $\phi'$ := Sonf($\phi$);
    $S := \emptyset$;
    /* $\phi'$ is in the form $\Psi_{LTL} \wedge \Psi_{SOS}$          */
    for $\psi \in \Psi_{LTL}$ do
        $A$ := Ltl2Fts($\psi$);
        $S := S \cup \{A\}$;
    end
    for $\psi \in \Psi_{SOS}$ do
        $A$ := Sos2Fts($\psi$);
        $S := S \cup \{A\}$;
    end
    return $S$
end
```

Algorithm 1: Modular translation.

Intuitively, the approach can be seen as a way to decompose the property in small pieces, apply to each piece the most effective encoding, and then gluing together the results by synchronous composition.

We remark that the result of the translation is a set of implicitly synchronously composed FTSs, while the approaches described in Section II-D return a single FTS. This enables a greater efficiency since we can exploit in the search standard techniques of conjunctive partitioning [15] widely used in model checking. It also allows for multiple fairness conditions, thus possibly resulting in a more compact encoding.

Algorithm 1 can be further optimized. First, we notice that each formula $\phi \in \Psi_{SOS}$ matches exactly one of two specific patterns, i.e., $\mathbf{G}\,(P_I \rightarrow (r \Diamond\!\!\rightarrow P_F))$ and $\mathbf{G}\,(P_I \rightarrow (r \mapsto P_F))$. In the following two sections we present optimized versions of `Sos2Fts`, for each one of the SOS

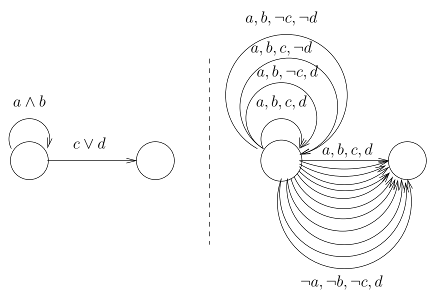


Fig. 2. Symbolic (on the left) vs. explicit (on the right) transitions.

patterns. In particular, we directly build the FTS S_ϕ from the NFA $A_r = \langle \Sigma, Q, q_0, \rho, F\rangle$ of r. As a consequence of the optimizations, the overall algorithm `ModSL2Fts` completely avoids the use of an intermediate ABA or NGBA. We remark that the symbolic encodings presented below are linear in the size of A_r and we do not add new variables either for the top-level $\mathbf{G}$ operator or for the propositions P_I and P_F.

A second optimization is to consider a symbolic representation of the labels in the NFA. The classical representation of transitions is a function $\rho : Q \times \Sigma \rightarrow 2^Q$ (see Def. 8). We call these transitions *explicit* because they depend on complete assignments to propositions. Instead, we use *symbolic* transitions, which are represented by a function $\rho : Q \rightarrow 2^{\mathcal{B}^{\neg}(\mathcal{A}) \times Q}$, where Boolean combinations of atomic formulas are used to represent sets of assignments. This way, we can preserve the succinctness of SEREs, which use Boolean expressions as atomic formulas.

Example 4: Consider the suffix implication subformula $\mathbf{G}\,(p_1 \rightarrow \{(a \wedge b)\mathbf{[*]} \,\mathbf{;}\, (c \vee d)\} \mapsto p_2)$ of Example 3. In order to represent $r = (a \wedge b)\mathbf{[*]} \,\mathbf{;}\, (c \vee d)$, we use the following NFA with symbolic transitions:

$$A_r = \langle \{q_1, q_2\}, q_1, \{(q_1, a \wedge b, q_1), (q_1, c \vee d, q_2)\}, \{q_2\}\rangle.$$

In Figure 2, we show A_r both with the symbolic and with the explicit transitions.

A. Encoding $\phi := \mathbf{G}\,(P_I \rightarrow (r \mapsto P_F))$

We recall that the semantics of $r \mapsto P_F$ says that every time we read a word accepted by r we must set P_F to true. In order to monitor the acceptance of r, we use the automaton A_r. We introduce a set of variables $V := \{v_q\}_{q \in Q}$, with the intuition that v_q is true whenever we start monitoring the acceptance of a suffix of r associated with q. Thus, if v_q is true and we read $l \in \Sigma$, we need to activate every $v_{q'}$ with $q' \in \rho(q, l)$. A simple way to encode this would be to enumerate all possible explicit transitions, i.e., all the assignments to the atomic propositions. Instead, we choose to enumerate all possible subsets of the symbolic transitions outgoing from q, since typical SEREs often exhibit a limited branching rate, and have Boolean expressions as atomic formulas.

Thus, the activation of the next variables is defined by the following condition:

$$T_r := \bigwedge_{q \in Q} (v_q \rightarrow (\bigvee_{C \subseteq \rho(q)} (\bigwedge_{(\gamma, q') \in C} (\gamma \wedge v'_{q'}) \wedge \bigwedge_{(\gamma, q') \in \rho(q) \setminus C} \neg\gamma)))$$

Given a state q, $\rho(q)$ contains a set of pairs (γ, q'), each of which symbolically represents a set of explicit transitions, i.e. the pairs (l, q') such that $l \models \gamma$. Given a subset C of $\rho(q)$, $v'_{q'}$ holds if q' can be reached from q by means of a transition in C, and if the propositions satisfy all the labels occurring in C and violate all the labels not in C.

The formula T_r can be seen as a *sloppy* symbolic encoding [42] of the deterministic and completed version of A_r. We note in fact that, for every $l \in \Sigma$, there exists one and only one subset of transitions in $\rho(q)$ whose labels accept l.

The FTS S_ϕ is defined as $\langle V_\phi, \mathcal{A}, T_\phi, I_\phi, \{F_\phi\}\rangle$, where $V_\phi = V$, $I_\phi := true$, $F_\phi := true$, and

$$T_\phi := (P_I \to v_{q_0}) \wedge T_r[v'_q \wedge P_F / v'_q]_{q \in F}.$$

In the substitution, P_F is used rather than P'_F, in order to take into account the overlapping required by $\mapsto$.

Theorem 10: Suppose, the NBA A_ϕ is built from S_ϕ as described in Definition 15, so that $\mathcal{L}(A_\phi) = \mathcal{L}(S_\phi)$, then $\mathcal{L}(A_\phi) = \mathcal{L}(\phi)$, and thus $\mathcal{L}(S_\phi) = \mathcal{L}(\phi)$.

Example 5: Consider the suffix implication subformula $\mathbf{G}\,(p_1 \to \{(a \wedge b)\mathbf{[*]}\ \mathbf{;}\ (c \vee d)\} \mapsto p_2)$ of Example 3 and the NFA for $r = (a \wedge b)\mathbf{[*]}\ \mathbf{;}\ (c \vee d)$ of Example 4. Then the transition relation of the final FTS is:

$$\begin{aligned}(p_1 \to v_{q_1}) \wedge (v_{q_1} \to\ & (((a \wedge b) \wedge (c \vee d) \wedge v'_{q_1} \wedge v'_{q_2} \wedge p_2) \vee \\ & ((a \wedge b) \wedge \neg(c \vee d) \wedge v'_{q_1}) \vee \\ & (\neg(a \wedge b) \wedge (c \vee d) \wedge v'_{q_2} \wedge p_2) \vee \\ & (\neg(a \wedge b) \wedge \neg(c \vee d)))).\end{aligned}$$

B. Encoding $\phi := \mathbf{G}\,(P_I \to (r \diamond\!\!\to P_F))$

In the case of the suffix conjunction $r \diamond\!\!\to P_F$, the semantics requires that at least one word accepted by r is read. As before, we introduce a linear number of symbolic variables $V_r := \{v_q\}_{q \in Q}$. The condition on the next variables is simpler, because if v_q is true and we read $l \in \Sigma$, it is sufficient to activate at least one $v_{q'}$ with $q' \in \rho(q, l)$.

$$T_r := \bigwedge_{q \in Q} (v_q \to (\bigvee_{(\gamma, q') \in \rho(q)} (\gamma \wedge v'_{q'})))$$

The FTS S_ϕ is defined as $\langle V_\phi, \mathcal{A}, T_\phi, I_\phi, \{F_\phi\}\rangle$, where $V_\phi = V_L \cup V_R$, $V_L := \{v_{qL}\}_{v_q \in V_r}$, $V_R := \{v_{qR}\}_{v_q \in V_r}$, $I_\phi := true$, and

- $T_\phi := P_I \to v_{q_0 L} \wedge$
  $T_r[v_{qL}/v_q]_{q \in Q}[v'_{qL}/v'_q]_{q \in Q \setminus F}[v'_{qL} \vee P_F/v'_q]_{q \in F} \wedge$
  $(\bigwedge_{q \in Q} \neg v_{qR}) \to (\bigwedge_{q \in Q}(v'_{qL} \to v'_{qR})) \wedge$
  $T_r[v_{qR}/v_q]_{q \in Q}[v'_{qR}/v'_q]_{q \in Q \setminus F}[v'_{qR} \vee P_F/v'_q]_{q \in F} \wedge$
  $\bigwedge_{q \in Q}(v_{qR} \to v_{qL})$,
- $F_\phi := \bigwedge_{v_q \in V_r} \neg v_{qR}$.

This encoding corresponds to the MH construction (see Theorem 5). Intuitively, the FTS triggers and monitors the acceptance of multiple runs of A_r that start and finish at different points in time. In particular, the variables V_L monitor the simulation of A_r, while the variables V_R track if every simulation eventually terminates with an accepting state.

Theorem 11: Suppose, the NBA A_ϕ is built from S_ϕ as described in Definition 15, so that $\mathcal{L}(A_\phi) = \mathcal{L}(S_\phi)$, then $\mathcal{L}(A_\phi) = \mathcal{L}(\phi)$, and thus $\mathcal{L}(S_\phi) = \mathcal{L}(\phi)$.

This solution produces a fairness condition for each Suffix Conjunction Subformula. Despite the resulting encoding is more compact, it may result in inefficiencies in the search. A single global condition FC, shared by all the FTS corresponding to Suffix Conjunction Subformula may alleviate this problem. In this case, the transition relation of each FTS must be changed so that the variables that track the fulfillment of the fairness condition are set to false only when FC becomes true. Formally, let Scs be the set of Suffix Conjunction Subformulas,

- $T_\phi := P_I \to v_{q_0 L} \wedge$
  $T_r[v_{qL}/v_q]_{q \in Q}[v'_{qL}/v'_q]_{q \in Q \setminus F}[v'_{qL} \vee P_F/v'_q]_{q \in F} \wedge$
  $FC \to (\bigwedge_{q \in Q}(v'_{qL} \to v'_{qR})) \wedge$
  $T_r[v_{qR}/v_q]_{q \in Q}[v'_{qR}/v'_q]_{q \in Q \setminus F}[v'_{qR} \vee P_F/v'_q]_{q \in F} \wedge$
  $\bigwedge_{q \in Q}(v_{qR} \to v_{qL})$,
- $FC := \bigwedge_{\phi \in Scs} F_\phi$.

VI. Syntactic optimizations for PSL

In this section, we describe an optimized approach, which extends the SONF-based conversion with the integration of the following simplifications. Before the SONF conversion, we apply two steps: (i) we simplify the SEREs in order to reduce the subformulas in the scope of SERE conjunction operators; (ii) we simplify occurrences of suffix operators by converting as much as possible the SEREs into LTL. Then, after the conversion in SONF, we apply two other steps: (iii) we simplify the Suffix Operator Subformulas by means of rules that strengthen the ones in (ii) by exploiting the specific structure of SOSs; (iv) the LTL component is rewritten in order to minimize the overall automaton and to reduce the number of resulting fairness constraints. In the rest of this section we describe the first three sets of rewriting rules $(i - iii)$, which regard SEREs and SERELTL formulas. For lack of space, we do not report a detailed description of the LTL simplification rules (iv), which follow the techniques described in [20], [41].

In the following, we write $b, b_1, b_2, \ldots$ for Boolean formulas, and $r, r_1, r_2, \ldots$ for SEREs. We notice that we can check if the empty word ϵ belongs to the language of r, written $\epsilon \in \mathcal{L}(r)$, by parsing: if $r = b$, then $False$; if $r = r_1\ \mathbf{;}\ r_2$, then $True$ if both r_1 and r_2 accept ϵ, $False$ otherwise; if $r = r_1\ \mathbf{:}\ r_2$, then $False$; if $r = r_1\mathbf{[*]}$, then $True$; if $r = r_1\ \mathbf{\&\&}\ r_2$ or $r = r_1\ \mathbf{\&}\ r_2$, then $True$ if both r_1 and r_2 accept ϵ, $False$ otherwise; if $r = r_1\ \mathbf{|}\ r_2$, then $True$ if either r_1 or r_2 accepts ϵ, $False$ otherwise. We also define a SERE to be "of fixed length" as follows. b has fixed length 1; $r = r_1\ \mathbf{;}\ r_2$ has fixed length $n_1 + n_2$ iff r_1 and r_2 have fixed length n_1 and n_2. $r = r_1\ \mathbf{:}\ r_2$ has fixed length $n_1 + n_2 - 1$ iff r_1 and r_2 have fixed length n_1 and n_2. $r = r_1\ \mathbf{\&\&}\ r_2$ has fixed length n iff both r_1 and r_2 have fixed length n. $r = r_1\ \mathbf{\&}\ r_2$ has fixed length n iff r_1 and r_2 have fixed length n_1 and n_2, and n is the maximum between n_1 and n_2. $r = r_1\ \mathbf{|}\ r_2$ has fixed length n iff both r_1 and r_2 have fixed length n. $r = r_1\mathbf{[*]}$ has fixed length n iff $n = 0$ and r_1 has fixed length 0.

(i) Simplifying SEREs*:* Step (i) of our simplification flow is implemented by the following rules for $\mathbf{\&\&}$:

$$
\begin{array}{rcl}
r \mathbin{\mathbf{\&\&}} \{r_1 \mid r_2\} & \Rightarrow & \{r \mathbin{\mathbf{\&\&}} r_1\} \mid \{r \mathbin{\mathbf{\&\&}} r_2\} \\
b_1 \mathbin{\mathbf{\&\&}} b_2 & \Rightarrow & b_1 \wedge b_2 \\
b \mathbin{\mathbf{\&\&}} \{r_1 \mathbin{\mathbf{\&\&}} r_2\} & \Rightarrow & \{b \mathbin{\mathbf{\&\&}} r_1\} \mathbin{\mathbf{\&\&}} r_2 \\
b \mathbin{\mathbf{\&\&}} \{r_1 \mathbin{\mathbf{;}} r_2\} & \Rightarrow & \left\{ \begin{array}{l} False \text{ if } \epsilon \notin \mathcal{L}(r_1), \epsilon \notin \mathcal{L}(r_2) \\ b \mathbin{\mathbf{\&\&}} r_1 \text{ if } \epsilon \notin \mathcal{L}(r_1), \epsilon \in \mathcal{L}(r_2) \\ b \mathbin{\mathbf{\&\&}} r_2 \text{ if } \epsilon \in \mathcal{L}(r_1), \epsilon \notin \mathcal{L}(r_2) \\ b \mathbin{\mathbf{\&\&}} r_1 \mid b \mathbin{\mathbf{\&\&}} r_2 \text{ otherwise} \end{array} \right. \\
b \mathbin{\mathbf{\&\&}} \{r_1 \mathbin{\mathbf{:}} r_2\} & \Rightarrow & \{b \mathbin{\mathbf{\&\&}} r_1\} \mathbin{\mathbf{\&\&}} r_2 \\
b \mathbin{\mathbf{\&\&}} r\mathbf{[*]} & \Rightarrow & b \mathbin{\mathbf{\&\&}} r \\
b\mathbf{[*]} \mathbin{\mathbf{\&\&}} \{r_1 \mathbin{\mathbf{;}} r_2\} & \Rightarrow & \{b\mathbf{[*]} \mathbin{\mathbf{\&\&}} r_1\} \mathbin{\mathbf{;}} \{b\mathbf{[*]} \mathbin{\mathbf{\&\&}} r_2\} \\
b\mathbf{[*]} \mathbin{\mathbf{\&\&}} \{r_1 \mathbin{\mathbf{:}} r_2\} & \Rightarrow & \{b\mathbf{[*]} \mathbin{\mathbf{\&\&}} r_1\} \mathbin{\mathbf{:}} \{b\mathbf{[*]} \mathbin{\mathbf{\&\&}} r_2\} \\
b\mathbf{[*]} \mathbin{\mathbf{\&\&}} r\mathbf{[*]} & \Rightarrow & \{b\mathbf{[*]} \mathbin{\mathbf{\&\&}} r\}\mathbf{[*]} \\
\{b_1 \mathbin{\mathbf{;}} r_1\} \mathbin{\mathbf{\&\&}} \{b_2 \mathbin{\mathbf{;}} r_2\} & \Rightarrow & \{b_1 \wedge b_2\} \mathbin{\mathbf{;}} \{r_1 \mathbin{\mathbf{\&\&}} r_2\} \\
\{b_1 \mathbin{\mathbf{:}} r_1\} \mathbin{\mathbf{\&\&}} \{b_2 \mathbin{\mathbf{:}} r_2\} & \Rightarrow & \{b_1 \wedge b_2\} \mathbin{\mathbf{:}} \{r_1 \mathbin{\mathbf{\&\&}} r_2\} \\
\{r_1 \mathbin{\mathbf{;}} b_1\} \mathbin{\mathbf{\&\&}} \{r_2 \mathbin{\mathbf{;}} b_2\} & \Rightarrow & \{r_1 \mathbin{\mathbf{\&\&}} r_2\} \mathbin{\mathbf{;}} \{b_1 \wedge b_2\} \\
\{r_1 \mathbin{\mathbf{:}} b_1\} \mathbin{\mathbf{\&\&}} \{r_2 \mathbin{\mathbf{:}} b_2\} & \Rightarrow & \{r_1 \mathbin{\mathbf{\&\&}} r_2\} \mathbin{\mathbf{:}} \{b_1 \wedge b_2\} \\
\{b_1\mathbf{[*]} \mathbin{\mathbf{;}} r_1\} \mathbin{\mathbf{\&\&}} \{b_2 \mathbin{\mathbf{;}} r_2\} & \Rightarrow & \{r_1 \mathbin{\mathbf{\&\&}} \{b_2 \mathbin{\mathbf{;}} r_2\}\} \mid \\
 & & \quad \{\{b_1 \wedge b_2\} \mathbin{\mathbf{;}} \{\{b_1\mathbf{[*]} \mathbin{\mathbf{;}} r_1\} \mathbin{\mathbf{\&\&}} r_2\}\} \\
\{b_1\mathbf{[*]} \mathbin{\mathbf{;}} r_1\} \mathbin{\mathbf{\&\&}} \{b_2\mathbf{[*]} \mathbin{\mathbf{;}} r_2\} & \Rightarrow & \{b_1 \wedge b_2\}\mathbf{[*]}\mathbf{;}\{\{\{r_1 \mathbin{\mathbf{\&\&}} \{b_2\mathbf{[*]} \mathbin{\mathbf{;}} r_2\}\} \mid \\
 & & \qquad \{\{b_1\mathbf{[*]} \mathbin{\mathbf{;}} r_1\} \mathbin{\mathbf{\&\&}} r_2\}\}\} \\
r_1\mathbf{[*]} \mathbin{\mathbf{\&\&}} r_2\mathbf{[*]} & \Rightarrow^{\dagger} & \{r_1\mathbf{[*}n_2\mathbf{]} \mathbin{\mathbf{\&\&}} r_2\mathbf{[*}n_1\mathbf{]}\}\mathbf{[*]}
\end{array}
$$

In the rule (†), r_1 and r_2 must be of fixed length, and n_1 and n_2 are the least integers such that $n = (|r_1| \cdot n_2) = (|r_2| \cdot n_1)$. For lack of space, we only report some of the rules for **&**; other rules based on the commutativity and associativity of the operators are also omitted.

$$
\begin{array}{rcl}
r \mathbin{\mathbf{\&}} \{r_1 \mid r_2\} & \Rightarrow & \{r \mathbin{\mathbf{\&}} r_1\} \mid \{r \mathbin{\mathbf{\&}} r_2\} \\
b_1 \mathbin{\mathbf{\&}} b_2 & \Rightarrow & b_1 \wedge b_2 \\
b \mathbin{\mathbf{\&}} \{r_1 \mathbin{\mathbf{\&}} r_2\} & \Rightarrow & \{b \mathbin{\mathbf{\&}} r_1\} \mathbin{\mathbf{\&}} r_2 \\
b \mathbin{\mathbf{\&}} \{r_1 \mathbin{\mathbf{;}} r_2\} & \Rightarrow & \left\{ \begin{array}{l} b \mathbin{\mathbf{:}} \{r_1 \mathbin{\mathbf{;}} r_2\} \text{ if } \epsilon \notin \mathcal{L}(r_1) \text{, or } \epsilon \notin \mathcal{L}(r_2) \\ b \mathbin{\mathbf{:}} \{r_1 \mathbin{\mathbf{;}} r_2\} \mid b \quad \text{otherwise} \end{array} \right. \\
b \mathbin{\mathbf{\&}} \{r_1 \mathbin{\mathbf{:}} r_2\} & \Rightarrow & b \mathbin{\mathbf{:}} \{r_1 \mathbin{\mathbf{:}} r_2\} \\
b \mathbin{\mathbf{\&}} r\mathbf{[*]} & \Rightarrow & b \mid \{b \mathbin{\mathbf{:}} r\mathbf{[*]}\} \\
b\mathbf{[*]} \mathbin{\mathbf{\&}} r & \Rightarrow & \{r\} \mid \{\{b\mathbf{[*]} \mathbin{\mathbf{\&\&}} r\} \mathbin{\mathbf{;}} b\mathbf{[*]}\} \\
r_1\mathbf{[*]} \mathbin{\mathbf{\&}} r_2 & \Rightarrow & r_2 \mid r_1\mathbf{[*]} \mathbin{\mathbf{\&\&}} \{r_2 \mathbin{\mathbf{;}} true\mathbf{[*]}\} \\
\{b_1 \mathbin{\mathbf{;}} r_1\} \mathbin{\mathbf{\&}} \{b_2 \mathbin{\mathbf{;}} r_2\} & \Rightarrow & \{b_1 \wedge b_2\} \mathbin{\mathbf{;}} \{r_1 \mathbin{\mathbf{\&}} r_2\} \\
\{b_1 \mathbin{\mathbf{:}} r_1\} \mathbin{\mathbf{\&}} \{b_2 \mathbin{\mathbf{:}} r_2\} & \Rightarrow & \{b_1 \wedge b_2\} \mathbin{\mathbf{:}} \{r_1 \mathbin{\mathbf{\&}} r_2\} \\
\{r_1 \mathbin{\mathbf{;}} b_1\} \mathbin{\mathbf{\&}} \{r_2 \mathbin{\mathbf{;}} b_2\} & \Rightarrow & \{r_1 \mathbin{\mathbf{\&}} r_2\} \mathbin{\mathbf{;}} \{b_1 \wedge b_2\} \\
\{r_1 \mathbin{\mathbf{:}} b_1\} \mathbin{\mathbf{\&}} \{r_2 \mathbin{\mathbf{:}} b_2\} & \Rightarrow & \{r_1 \mathbin{\mathbf{\&}} r_2\} \mathbin{\mathbf{:}} \{b_1 \wedge b_2\} \\
r_1\mathbf{[*]} \mathbin{\mathbf{\&}} r_2\mathbf{[*]} & \Rightarrow & r_1\mathbf{[*]} \mid r_2\mathbf{[*]}
\end{array}
$$

Example 6: The above rewriting rules apply to the SERE in the SERELTL formula of Example 1, as follows:

$$
\begin{array}{c}
\{a \mathbin{\mathbf{;}} b\mathbf{[*]} \mathbin{\mathbf{;}} c\} \mathbin{\mathbf{\&\&}} \{d\mathbf{[*]} \mathbin{\mathbf{;}} e\} \Rightarrow \{\{a \mathbin{\mathbf{;}} b\mathbf{[*]}\} \mathbin{\mathbf{\&\&}} d\mathbf{[*]}\} \mathbin{\mathbf{;}} c \wedge e \Rightarrow \\
\{a \mathbin{\mathbf{\&\&}} d\mathbf{[*]}\} \mathbin{\mathbf{;}} \{b\mathbf{[*]} \mathbin{\mathbf{\&\&}} d\mathbf{[*]}\} \mathbin{\mathbf{;}} c \wedge e \Rightarrow \\
\{a \mathbin{\mathbf{\&\&}} d\} \mathbin{\mathbf{;}} \{b\mathbf{[*]} \mathbin{\mathbf{\&\&}} d\}\mathbf{[*]} \mathbin{\mathbf{;}} c \wedge e \Rightarrow \\
a \wedge d \mathbin{\mathbf{;}} \{b \mathbin{\mathbf{\&\&}} d\}\mathbf{[*]} \mathbin{\mathbf{;}} c \wedge e \Rightarrow \\
a \wedge d \mathbin{\mathbf{;}} \{b \wedge d\}\mathbf{[*]} \mathbin{\mathbf{;}} c \wedge e.
\end{array}
$$

(ii) Simplifying Suffix Operations: In order to reduce a SERELTL formula to LTL "as much as possible", we define the following rules, where (†) requires $\epsilon \notin \mathcal{L}(r_1)$ and $\epsilon \notin \mathcal{L}(r_2)$, while (‡) requires $\epsilon \notin \mathcal{L}(r)$. The conversion aims at minimizing the size of the SEREs and heuristically applies the rewriting till no rule is applicable anymore.

$$
\begin{array}{rcl}
\{\mathbf{[0*]}\} \Diamond\!\!\rightarrow \phi & \Rightarrow & False \\
\{b\} \Diamond\!\!\rightarrow \phi & \Rightarrow & b \wedge \phi \\
\{r_1 \mathbin{\mathbf{:}} r_2\} \Diamond\!\!\rightarrow \phi & \Rightarrow & \{r_1\} \Diamond\!\!\rightarrow (\{r_2\} \Diamond\!\!\rightarrow \phi) \\
\{r_1 \mathbin{\mathbf{;}} r_2\} \Diamond\!\!\rightarrow \phi & \Rightarrow^{\dagger} & \{r_1\} \Diamond\!\!\rightarrow \mathbf{X}\,(\{r_2\} \Diamond\!\!\rightarrow \phi) \\
\{r_1 \mid r_2\} \Diamond\!\!\rightarrow \phi & \Rightarrow & (\{r_1\} \Diamond\!\!\rightarrow \phi) \vee (\{r_2\} \Diamond\!\!\rightarrow \phi) \\
\{r \mathbin{\mathbf{;}} b\mathbf{[*]}\} \Diamond\!\!\rightarrow \phi & \Rightarrow^{\ddagger} & \{r\} \Diamond\!\!\rightarrow ((\mathbf{X}\, b) \mathbin{\mathbf{U}} \phi) \\
\{b\mathbf{[*]} \mathbin{\mathbf{;}} r\} \Diamond\!\!\rightarrow \phi & \Rightarrow^{\ddagger} & b \mathbin{\mathbf{U}} (\{r\} \Diamond\!\!\rightarrow \phi) \\
 & & \\
\{\mathbf{[0*]}\} \mapsto \phi & \Rightarrow & True \\
\{b\} \mapsto \phi & \Rightarrow & b \rightarrow \phi \\
\{r_1 \mathbin{\mathbf{:}} r_2\} \mapsto \phi & \Rightarrow & \{r_1\} \mapsto (\{r_2\} \mapsto \phi) \\
\{r_1 \mathbin{\mathbf{;}} r_2\} \mapsto \phi & \Rightarrow^{\dagger} & \{r_1\} \mapsto \mathbf{X}\,(\{r_2\} \mapsto \phi) \\
\{r_1 \mid r_2\} \mapsto \phi & \Rightarrow & (\{r_1\} \mapsto \phi) \wedge (\{r_2\} \mapsto \phi) \\
\{r \mathbin{\mathbf{;}} b\mathbf{[*]}\} \mapsto \phi & \Rightarrow^{\ddagger} & \{r\} \mapsto ((\mathbf{X}\, \neg b) \mathbin{\mathbf{R}} \phi) \\
\{b\mathbf{[*]} \mathbin{\mathbf{;}} r\} \mapsto \phi & \Rightarrow^{\ddagger} & \neg b \mathbin{\mathbf{R}} (\{r\} \mapsto \phi)
\end{array}
$$

The rewritings are mostly effective on those expressions where the Kleene-closure is applied to Boolean expressions, as shown in the following example.

Example 7: Consider the formula of Example 1. After rewriting the SERE as in the Example 6, the formula becomes $\mathbf{G}\,(\{a \wedge d \mathbin{\mathbf{;}} \{b \wedge d\}\mathbf{[*]} \mathbin{\mathbf{;}} c \wedge e\} \mapsto \{f \mathbin{\mathbf{;}} g\})$. The above rewriting rules apply as follows:

$$
\begin{array}{l}
\mathbf{G}\,(\{a \wedge d \mathbin{\mathbf{;}} \{b \wedge d\}\mathbf{[*]} \mathbin{\mathbf{;}} c \wedge e\} \mapsto \{f \mathbin{\mathbf{;}} g\}) \Rightarrow \\
\quad \mathbf{G}\,((a \wedge d) \rightarrow \mathbf{X}\,\{\{b \wedge d\}\mathbf{[*]} \mathbin{\mathbf{;}} c \wedge e\} \mapsto (f \wedge \mathbf{X}\,\{g\}) \Rightarrow \\
\qquad \mathbf{G}\,((a \wedge d) \rightarrow \mathbf{X}\,(\neg(b \wedge d) \mathbin{\mathbf{R}} (\{c \wedge e\} \mapsto (f \wedge \mathbf{X}\, g)) \Rightarrow \\
\qquad\quad \mathbf{G}\,((a \wedge d) \rightarrow \mathbf{X}\,(\neg(b \wedge d) \mathbin{\mathbf{R}} ((c \wedge e) \rightarrow (f \wedge \mathbf{X}\, g)).
\end{array}
$$

(iii) Rewriting Suffix Implication Subformulas: After the simplifications described in previous sections, the SONF conversion is carried out, so that the occurrences of suffix operators have the fixed structure of SOS, and can be further rewritten. The aim is to apply the suffix operators to smaller SERE. This way, we partition further the automaton representation, and we enable the sharing of subformulas representations. The following rule requires $\epsilon \notin \mathcal{L}(r)$, pushes the occurrences of suffix implication inside the SEREs, while keeping the overall formula in SONF.

$$
\mathbf{G}\,(P \rightarrow (\{r\mathbf{[*]}\} \mapsto P')) \quad \Rightarrow \quad \mathbf{G}\,(P \rightarrow (\{r\} \mapsto (P' \wedge \mathbf{X}\, P)))
$$

Note that the transformation preserves the language only if the global formula is the result of the SONF conversion, so that there is a fixed structure for SOS. Unfortunately, no similar transformation is possible for suffix conjunction. In order to simplify the suffix conjunction in an analog way, one would need to add a fairness condition in order to guarantee that every P is eventually followed by the corresponding P'.

All the simplifications described above are correct.

Theorem 12: Let ϕ be a SERELTL property and ϕ' be the SERELTL formula resulting from the application of the above rewrite rules. Then $\mathcal{L}(\phi) = \mathcal{L}(\phi')$.

VII. Related Work

Since the seminal paper on LTL [36], the quest of a richer specification language that kept the simplicity of the linear temporal operators has been long (see [44] for an overview). Many formalisms have been conceived to reach the omega-regular expressiveness extending LTL either with grammar operators [48], or with automata [45], or with quantification [40], or with fixpoint [46]. Nevertheless, this kind of formalisms was not practical and not easy for designers to use. PSL, ForSpec [2] and SVA [47] instead combine the simplicity of regular expressions with LTL in order to reach the same expressive power. The core of the formalism common to all these languages is also called RELTL [8], and can be viewed as an extension of LTL with the dynamic modalities of Propositional Dynamic Logic [21], interpreted linearly [44].

In this section, we basically use PSL to refer to logics that, similarly to SERELTL, combine LTL with regular expressions. As discussed in Section II, an explicit translation from PSL into NGBA is described in [8], while [3] encodes PSL formulas into ABA. The authors of [25] exploit the fact that the ABAs obtained from PSL are "weak" (see [34]) to directly define a symbolic encoding for incremental `SAT`-based Bounded Model Checking. Therefore, their encoding cannot be reused for different verification engines, or different analysis tools.

The work in [37] proposes a different direct encoding of PSL into symbolically represented NGBA. It is based on the notion of tester, a finite-state machine that monitors if the suffix of the processed word satisfies the formula. The translation is bottom-up and compositional: the SEREs are first translated into grammars; then each subformula is associated with a symbolic variable that monitors its satisfiability. The translation is similar to our normalization step. There are at least two key differences. First, the symbolic variables we introduce trigger some subformulas (with an implication), while in a tester the value of the symbolic variables become true *if and only if* the associated subformula is satisfied (double implication). This difference may have substantial impact in the search, in particular in the case of SAT solvers, where implications may enable the use of don't cares. No implementation is however available for the approach in [37]. The second difference is our use of syntactic transformations, that are carried out also at the level of formulas.

Motivated by the needs of dynamic verification, the work in [7] studied how to convert PSL formulas into dynamic checkers, also known as monitors. The goal and the assumptions on the input formula are different from the static verification case. Only the simple subset of PSL is considered, which syntactically restricts the formulas to safety properties. The resulting circuit checks at run time if the property is satisfied or alert to the first failure. The corresponding automata might be not equivalent to the input formula, and therefore are not comparable with our results. Remarkably, some subroutines such as the construction of NFAs from the SEREs may be shared by the two approaches, and the determinization procedure presented in [7] exploits the symbolic labels in a way similar to our approach.

Finally, rewriting the formulas into a normal form is also used in two other different approaches: temporal resolution [22] exploits a normal form in order to determine if a formula is satisfiable; in [38], properties are normalized in order to synthesize a fragment of the simple subset of PSL.

VIII. Experimental evaluation

A. Evaluation Methodology

1) Measures: The proposed algorithms aims at an efficient compilation of SERELTL formulas into symbolically-represented automata. The efficiency is measured in terms of how fast the compilation is performed (*construction time*), and how fast a search of the resulting automata state space can be (*search time*). The efficiency of the search strongly depends on the adopted verification engine. Thus, in order to better evaluate the encoding, we use both a BDD-based and a SAT-based engines.

Construction and search times are typically antithetic parameters since a slow compilation may spend most of time in reducing the state space, thus resulting in a fast search procedure. Similarly, a fast compilation that represents implicitly the product of some components may pay the construction of such products at search time. Therefore, we evaluate the algorithm also on the sum of construction and search times (*total time*).

2) Examples Suite: For the comparison, we use the test suite of 1076 properties proposed in [11]. The set of properties has been obtained by filling in, with randomly generated SEREs, typical PSL patterns extracted from industrial case studies [4], which include both safety and liveness properties and fall in the SERELTL language. Then, we used both Boolean combinations and single and double implications between big conjunctions of typical properties. The latter cases model problems arising in requirements engineering setting, i.e. refinement and equivalence among specifications. More in detail, we obtained four families of properties: a) 297 formulas consist of conjunctions of random-filled PSL patterns [4]; b) 198 formulas consist of an implication between large conjunctions of random-filled PSL patterns; c) 198 formulas consist of a double implication between large conjunctions of random-filled PSL patterns; d) 383 formulas consists of entirely random PSL formulas.

Finally, for each formula, we also consider the negation, reaching a total number of 2152 formulas.

3) Procedures: We implemented and evaluated our algorithm within the NuSMV model checker [9]. We compared it against the symbolic encoding [6] of the monolithic approach [3] (called Mono), implemented as part of the PROSYD project and integrated into NuSMV. We consider the proposed approach without (referred to as ModNoSO) and with (referred to as Mod) the syntactic optimizations.

We compared the approaches with respect to automaton generation, and fair cycle detection (i.e. language emptiness), using both a BDD-based approach [19], and the Simple Bounded Model Checking (SBMC) SAT-based approach [26]. The BDD-based algorithm for fair cycle detection is restricted to the set of reachable states, which are preliminarily computed. The BDD algorithms were run with dynamic variable ordering activated. For SBMC, we used MiniSAT [16] as SAT engine. These settings provided good performance for all the approaches.

All experiments were run on a 3GHz Intel CPU equipped with 4GB of memory running Linux; for each run, we used a timeout of 120 seconds for construction time, 120 seconds for search time, and a memory limit of 768MB.

We also tested the new approach without the optimizations described in Section V-A and in Section V-B in order to evaluate their impact. It turned out that such optimizations are crucial and thus we always enabled them in the reported tests. Further evaluation has been performed on model checking instances. The results were similar in trend to the ones for language emptiness, and are not reported here (see [12] for additional details).

4) Relevance of the experimental results: As for any experimental evaluation, the test-bench is critical. Unfortunately, there are no available benchmarks neither for full PSL nor for the temporal layer we considered. There are benchmarks on LTL and on some subset of PSL, but none of them is suitable for evaluating our algorithm, since either they miss the SERE components or they do not cover fairness and general infinite behaviors. The benchmarks we created are based on realistic

PSL patterns, but have a random component that may bias the evaluation.

Another weakness is the lack of competitors, since other tools either produce monitors which are not comparable (such as [7]), or they verify the PSL property skipping the symbolic compilation such as [25]. We compared our algorithm with the only known non-commercial compiler for PSL.

Finally, the results may be biased by the choice of evaluating the search time by means of language emptiness. A thorough comparison on model checking would be more valuable, but would require to evaluate the different PSL properties on different real systems. This type of benchmarks are not currently available.

B. Experimental Results

Figures 3-6 show the results of the experimental evaluation[1]. Figure 3 reports the plot of the number of problems generated in a given amount of time (the samples are ordered by increasing computation time). The first row refers to construction time, while the second and the third refer respectively to search and total time. The left column shows the SAT-based evaluation, while the right column shows the BDD-based counterpart.

Figures 4-6 show the same results as scatter plots by comparing the methods pairwise. Each figure reports on the performance of a measure parameter, as discussed in Section VIII-A1. As before, the left column reports on the results obtained by using a SAT-based verification engine, while the right column refers to the results obtained with a BDD-based engine. For every figure the first row shows the comparison between MOD and MONO, the second row shows the comparison between MOD and MODNOSO, and the last row shows the comparison between MODNOSO and MONO. The lines labeled with “to” and “mo” represent respectively the timeout and the memory limit has been reached.

C. Discussion of the results.

The optimized modular approach MOD clearly outperforms MONO. The comparison between MODNOSO and MONO shows that the monolithic approach has a much harder time than the modular counterpart in completing the generation. The plots also show that the MOD rewriting, in addition to causing negligible overhead in the simple cases, seems to pay off in the harder cases. Overall, MOD is able to complete the generation for more properties than MONO: 116 more properties in the case of BDD, 184 as for SBMC. In the cases where MONO terminated, MOD is vastly more efficient: as for the BDD case, the monolithic approach is able to build the NBA for 1685 properties in about 14000 seconds, the modular approach dealt with 1801 properties in 5904 seconds; the improvement is even more dramatic for SBMC, for which MONO builds 1768 automata in 6491 seconds, while MOD builds almost all automata in 56 seconds. Note that, in the case of BDD, the improvement is obtained thanks to the syntactic optimizations. There are indeed several samples where the construction time is substantially reduced by these optimizations. We see in the BDD case that MOD completes the 1763 samples that MODNOSO can solve one order of magnitude faster; the same holds for the three most difficult instances in SAT. In addition, we see that MOD can solve harder problems where MODNOSO times out. The speed up typically occurs in examples where SERE automata have to be determinized both in MONO and MODNOSO, while for MOD the rules manage to generate smaller SERE.

The plots on search time show that the encoding of MONO is more efficient in terms of state space than MODNOSO because it allows for a fast search. This is more evident for a BDD-based search, while for the SAT-based case, the results are comparable. The reasons of such difference are twofold. First, the modular encoding tends to generate a higher number of fairness conditions. Second, MONO performs a more advanced minimization of the automata state space based on simulation. The syntactic optimization we proposed solve this problem and allow for a comparable search time, even superior when considering a BDD-based search. We remark that, MOD plot is always under the MONO one. The proposed rewriting is therefore as effective as the semantic ones of MONO; the improvement with respect to MODNOSO in terms of search time is also evident.

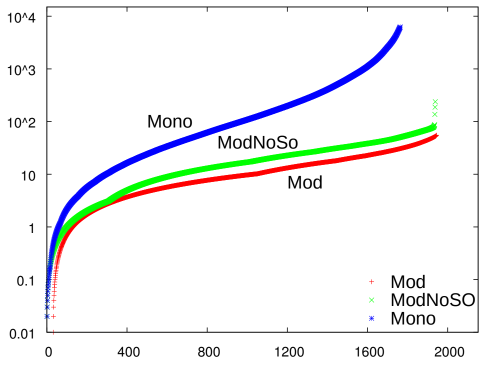
When considering the total time, we notice that these advantages come without paying the price of the semantic simplification. In fact, this price is often so high that also MODNOSO is superior to MONO. These claims are also confirmed by the scatter plots reported in Figures 4-6, where it is clear that MOD is almost uniformly superior to MODNOSO. It is also interesting to notice that while MONO and MOD have overall similar performance, they are not simplifying in the same way, and sometimes the semantic simplifications are unable to achieve as much reduction as rewriting.

IX. CONCLUSION

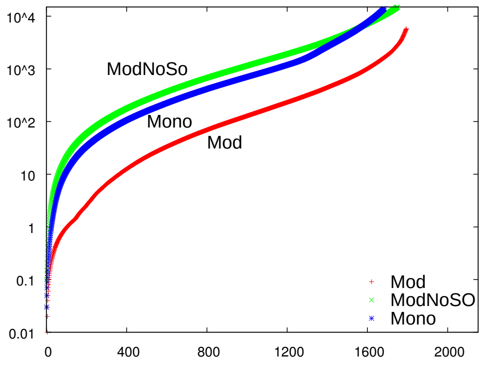
In this paper we have presented a new algorithm for the conversion of SERELTL into a symbolically represented NGBA. The approach is based on the decomposition of the SERELTL specification into a normal form that separates out LTL and SERE parts. The various components can be independently generated, and are implicitly conjuncted. Additional optimizations are possible by exploiting the specific structure of subformulas involving suffix operators. The approach is proved to be correct. A thorough experimental evaluation shows that the construction is extremely efficient, consuming many fewer resources than required by the monolithic construction. This makes it possible to tackle problems that were previously out of reach. Moreover, the resulting encoding enables an efficient search. The rewriting rules we proposed greatly reduce the redundancies of the generated automata. While the optimizations have negligible run-times, the benefit in search and overall time is substantial.

In the future, we plan to integrate the techniques described in this paper within the RAT tool [5], [35] for the formal analysis of requirements, to investigate the impact of different

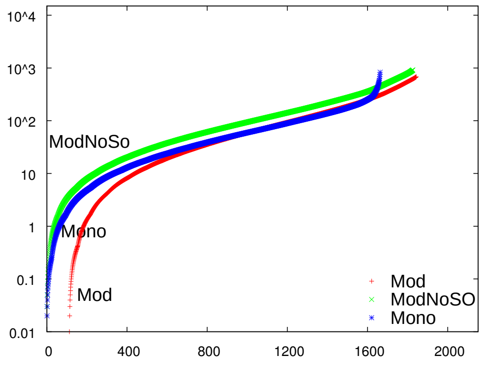
[1]All the files to reproduce this experimental analysis can be found at http://es.fbk.eu/people/tonetta/tests/tcad07/tcad.tar.gz.



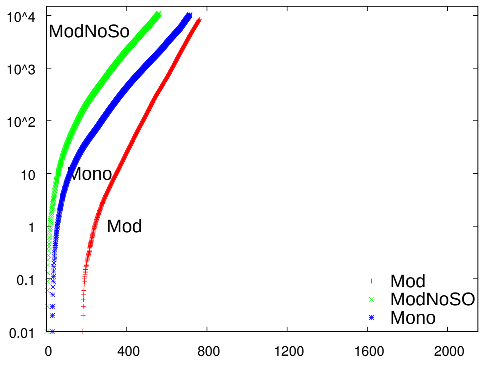
(a) Construction time per number of solved problems with `SBMC`.



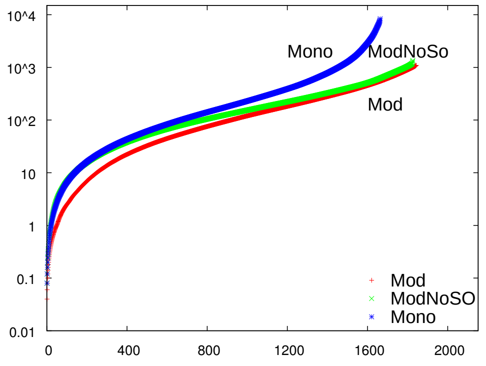
(b) Construction time per number of solved problems with `BDD`.



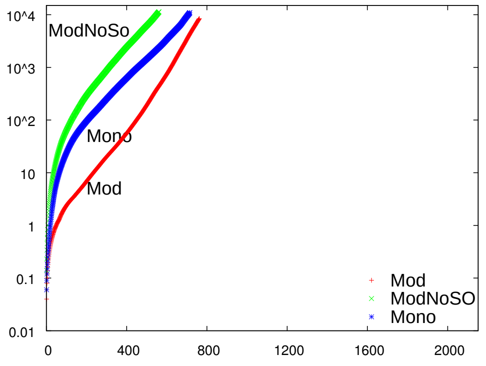
(c) Search time per number of solved problems with `SBMC`.



(d) Search time per number of solved problems with `BDD`.



(e) Total time per number of solved problems with `SBMC`.



(f) Total time per number of solved problems with `BDD`.

Fig. 3. Cumulative plots. X axis: number of solved problems; Y axis: time in seconds.

search techniques, in particular [10], and to apply them to practical case studies.

Acknowledgments

The authors would like to thank R. Bloem, I. Pill, V. Schuppan and M. Vardi for fruitful discussions. The authors would also thank S. Semprini for his contribution on a preliminary version of this paper.

References

[1] R. Armoni, D. Bustan, O. Kupferman, and M. Y. Vardi. Resets vs. aborts in linear temporal logic. In *TACAS*, pages 65–80, 2003.

[2] 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.

[3] S. Ben-David, R. Bloem, D. Fisman, A. Griesmayer, I. Pill, and S. Ruah. Automata Construction Algorithms Optimized for PSL. http://www.prosyd.org, 2005. Deliverable D 3.2/4.

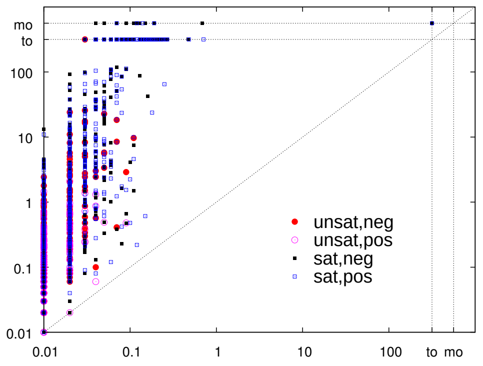
[4] S. Ben-David and A. Orni. Property-by-Example guide: a handbook of PSL/Sugar examples. http://www.prosyd.org, 2005. Deliverable D 1.1/3.

[5] R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. Rat: A tool for the formal analysis of requirements. In *CAV*, pages 263–267, 2007.

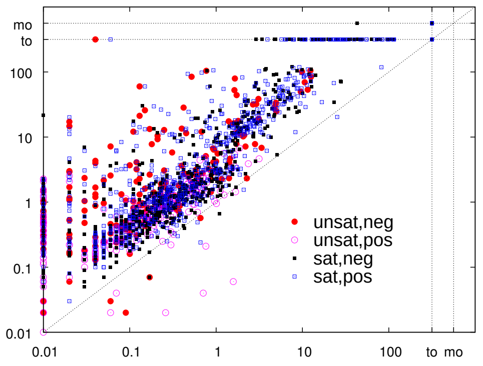
[6] R. Bloem, A. Cimatti, I. Pill, and M. Roveri. Symbolic Implementation of Alternating Automata. *International Journal of Foundations of Computer Science*, 18(4):727–743, August 2007.

[7] M. Boulé and Z. Zilic. Efficient Automata-Based Assertion-Checker Synthesis of PSL Properties. In *HLDVT*, 2006.

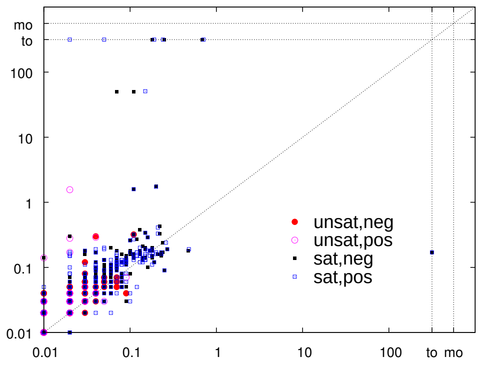
[8] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M. Y. Vardi. Regular Vacuity. In *CHARME*, pages 191–206, 2005.



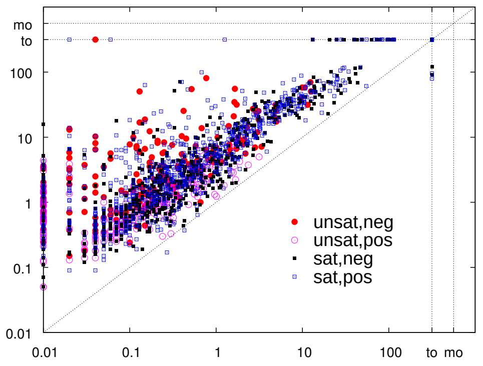
(a) SBMC, MOD (X axis) vs MONO (Y axis)



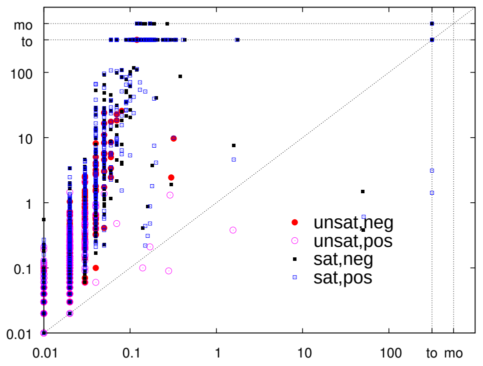
(b) BDD, MOD (X axis) vs MONO (Y axis)



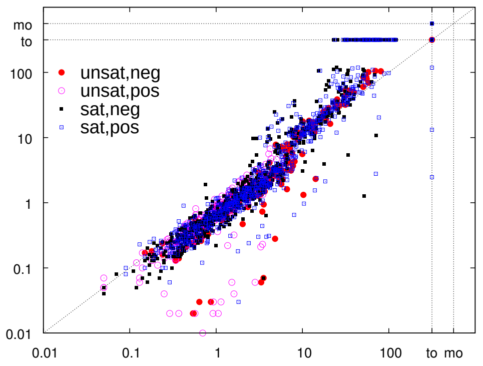
(c) SBMC, MOD (X axis) vs MODNOSO (Y axis)



(d) BDD, MOD (X axis) vs MODNOSO (Y axis)



(e) SBMC MODNOSO (X axis) vs MONO (Y axis)



(f) BDD, MODNOSO (X axis) vs MONO (Y axis)

Fig. 4. Construction time

[9] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: a new Symbolic Model Verifier. In *CAV*, pages 495 – 499, 1999.

[10] A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean abstraction for temporal logic satisfiability. In *CAV*, pages 532–546, 2007.

[11] A. Cimatti, M. Roveri, S. Semprini, and S. Tonetta. From PSL to NBA: a Modular Symbolic Encoding. In *FMCAD*, pages 125–133, 2006.

[12] A. Cimatti, M. Roveri, S. Semprini, and S. Tonetta. From PSL to NBA: a Modular Symbolic Encoding. Technical Report 18-08-06, 2006.

[13] A. Cimatti, M. Roveri, and S. Tonetta. Syntactic Optimizations for PSL Verification. In *TACAS*, pages 505–518, 2007.

[14] E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, 10(1):47–71, 1997.

[15] E.M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 1999.

[16] N. Eén and N. Sörensson. MiniSAT, 2005. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/Main.html.

[17] C. Eisner and D. Fisman. *A Practical Introduction to PSL (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc., 2006.

[18] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. Van Campenhout. Reasoning with temporal logic on truncated paths. In *CAV*, pages 27–39, 2003.

[19] E.A. Emerson and C.L. Lei. Efficient Model Checking in Fragments of the Propositional μ-Calculus. In *LICS*, pages 267–278, 1986.

[20] K. Etessami and G. Holtzmann. Optimizing Büchi Automata. In *CONCUR*, 2000.

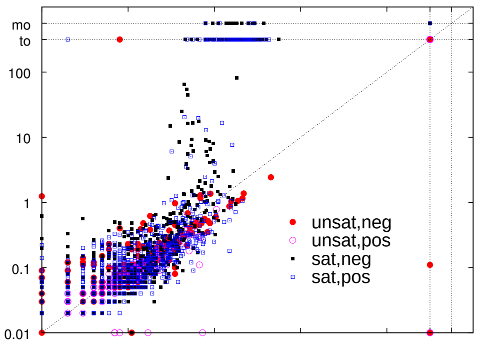
[21] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.

[22] M. Fisher. A resolution method for temporal logic. In *IJCAI*, pages 99–104, 1991.

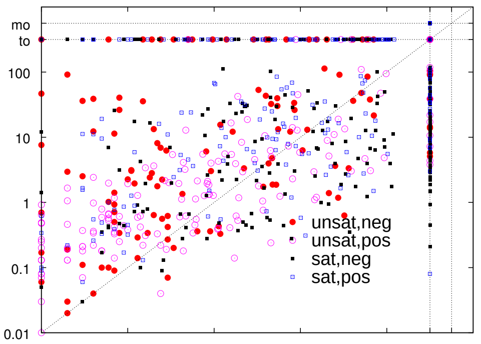
[23] C. Fritz. Constructing Büchi Automata from Linear Temporal Logic Using Simulation Relations for Alternating Büchi Automata. In *CIAA*, pages 35 – 48, 2003.

[24] P. Gastin and D. Oddoux. Fast LTL to Büchi Automata Translation. In *CAV*, pages 53–65, 2001.

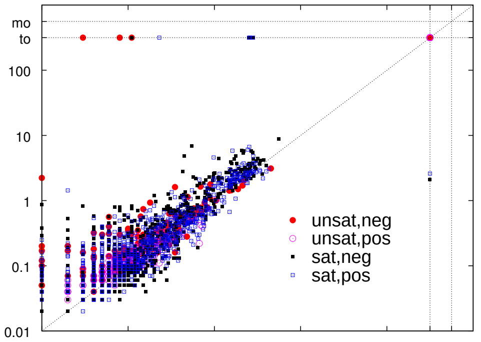
[25] K. Heljanko, T. Junttila, M. Keinänen, M. Lange, and T. Latvala. Bounded Model Checking for Weak Alternating Büchi Automata. In



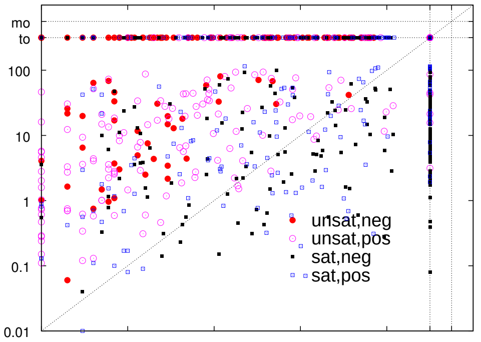
(a) SBMC, MOD (X axis) vs MONO (Y axis)



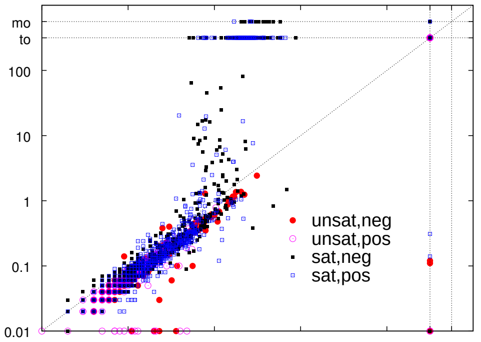
(b) BDD, MOD (X axis) vs MONO (Y axis)



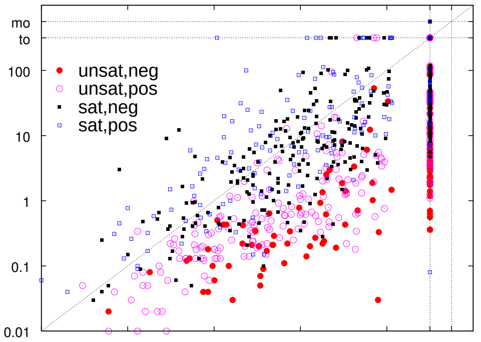
(c) SBMC, MOD (X axis) vs MODNOSO (Y axis)



(d) BDD, MOD (X axis) vs MODNOSO (Y axis)



(e) SBMC MODNOSO (X axis) vs MONO (Y axis)



(f) BDD, MODNOSO (X axis) vs MONO (Y axis)

Fig. 5. Search time

CAV, pages 95–108, 2006.

[26] K. Heljanko, T. A. Junttila, and T. Latvala. Incremental and complete bounded model checking for full PLTL. In *CAV*, volume 3576 of *LNCS*, pages 98–111. Springer, 2005.

[27] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[28] IEEE-Commission. IEEE standard for Property Specification Language (PSL), 2005. IEEE Std 1850-2005.

[29] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. In *ISTCS*, pages 147–158, 1997.

[30] O. Kupferman and S. Zuhovitzky. An Improved Algorithm for the Membership Problem for Extended Regular Expressions. In *MFCS*, pages 446–458, 2002.

[31] M. Lange. Linear Time Logics Around PSL: Complexity, Expressiveness, and a Little Bit of Succinctness. In *CONCUR*, pages 90–104, 2007.

[32] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*. Springer Verlag, New York, 1992.

[33] S. Miyano and T. Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984.

[34] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theor. Comput. Sci.*, 97(2):233–244, 1992.

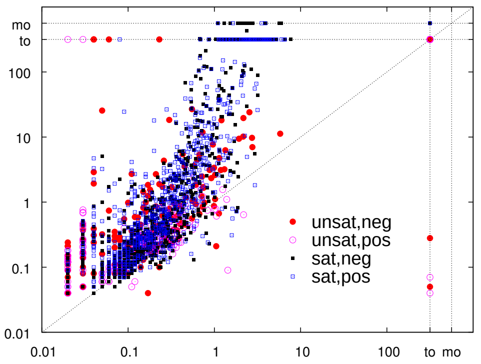
[35] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *DAC*, pages 821–826, 2006.

[36] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

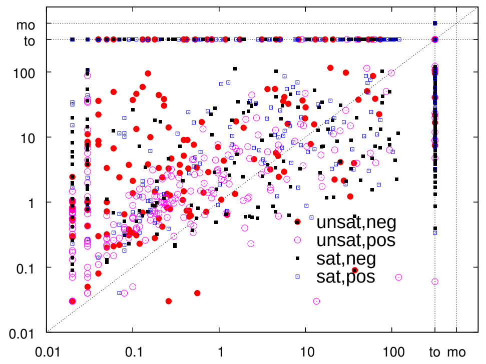
[37] A. Pnueli and A. Zaks. PSL Model Checking and Run-time Verification via Testers. In *FM*, pages 573–586, 2006.

[38] M. Schickel, V. Nimbler, M. Braun, and H. Eveking. *An Efficient Synthesis Method for Property-Based Design in Formal Verification*, chapter 10, pages 163–182. Kluwer Acad. Publishers, 2007.

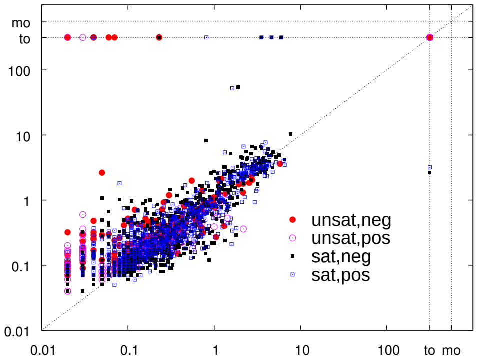
[39] R. Sebastiani, S. Tonetta, and M. Y. Vardi. Symbolic Systems, Explicit Properties: On Hybrid Approaches for LTL Symbolic Model Checking. In *CAV*, pages 350–363, 2005.



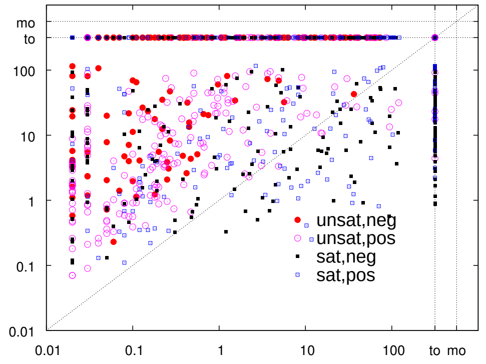
(a) SBMC, Mod (X axis) vs Mono (Y axis)



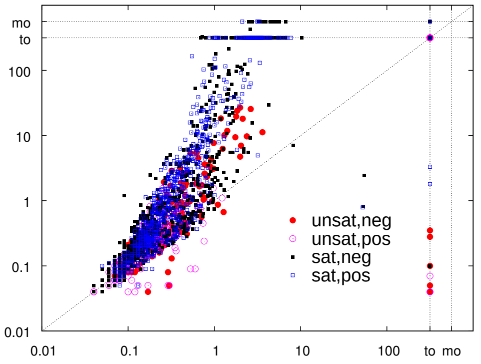
(b) BDD, Mod (X axis) vs Mono (Y axis)



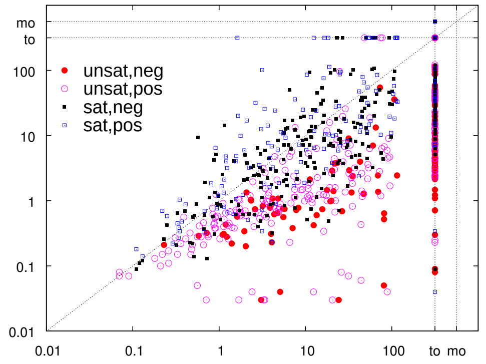
(c) SBMC, Mod (X axis) vs ModNoSO (Y axis)



(d) BDD, Mod (X axis) vs ModNoSO (Y axis)



(e) SBMC ModNoSO (X axis) vs Mono (Y axis)



(f) BDD, ModNoSO (X axis) vs Mono (Y axis)

Fig. 6. Total time

[40] A.P. Sistla, M.Y. Vardi, and P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. In *ICALP*, pages 465–474, 1985.

[41] F. Somenzi and R. Bloem. Efficient Büchi Automata from LTL Formulae. In *CAV*, pages 247–263, 2000.

[42] D. Tabakov and M.Y. Vardi. Experimental Evaluation of Classical Automata Constructions. In *LPAR*, pages 396–411, 2005.

[43] M. Y. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.

[44] M. Y. Vardi. From Church and Prior to PSL. In *Proceedings of Workshop on 25 Years of Model Checking*, August 2006.

[45] M. Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proc. of the 1st Symposium on Logic in Computer Science*, pages 332–344. IEEE Computer Society, 1986.

[46] M.Y. Vardi. A Temporal Fixpoint Calculus. In *POPL*, pages 250–259, 1988.

[47] S. Vijayaraghavan and M. Ramanathan. *A Practical Guide for SystemVerilog Assertions*. Springer, 2005.

[48] P. Wolper. Temporal Logic Can Be More Expressive. In *FOCS*, pages 340–348, 1981.

[49] H. Yamamoto. An Automata-Based Recognition Algorithm for Semi-extended Regular Expressions. In *MFCS*, pages 699–708, 2000.

Appendix

A. Correctness for Suffix Operator Normal Form

Theorem 9:

Let ϕ be a SERELTL formula over $\mathcal{A}$ in NNF, let P be a variable not occurring in ϕ, and let ψ be a SERELTL subformula of ϕ that occurs only positively in ϕ. If $\phi' := \phi[P/\psi] \wedge \mathbf{G}\,(P \rightarrow \psi)$ then $\mathcal{L}_{\mathcal{A}}(\phi) = \mathcal{L}_{\mathcal{A}}(\phi')$.

Proof: Let $\mathcal{A}' := \mathcal{A} \cup \{P\}$ and $\Sigma_{\mathcal{A}'} := 2^{\mathcal{A}'}$. If $w' \in \Sigma^{\omega}_{\mathcal{A}'}$, we will use $w'_{|\mathcal{A}}$ to denote the word in $\Sigma^{\omega}_{\mathcal{A}}$ such that $w'^{i}_{|\mathcal{A}} := w'^{i} \cap \mathcal{A}$ for all $i \geq 0$. It is sufficient to prove that

1) if $w \in \Sigma^{\omega}_{\mathcal{A}}$ and $w \models \phi$, then there exists $w' \in \Sigma^{\omega}_{\mathcal{A}'}$ such that $w' \models \phi'$ and $w = w'_{|\mathcal{A}}$;
2) vice versa, if $w' \in \Sigma^{\omega}_{\mathcal{A}'}$ and $w' \models \phi'$, then $w'_{|\mathcal{A}} \models \phi$.

1) Suppose that $w \in \Sigma^{\omega}_{\mathcal{A}}$ and $w \models \phi$. Define $w' \in \Sigma^{\omega}_{\mathcal{A}'}$ as follows:

$$w'^{i} := \begin{cases} w^{i} \cup \{P\} & \text{if } w^{i..} \models \psi \\ w^{i} & \text{if } w^{i..} \not\models \psi \end{cases}$$

Notice that, for all $i \geq 0$, $w'^{i..} \models P$ iff $w'^{i..} \models \psi$. Thus, $w' \models \mathbf{G}\,(P \rightarrow \psi)$. We prove by induction on the positive occurrence of ψ in ϕ that $w' \models \phi[P/\psi]$:

- if $\phi = \psi$, then $\phi[P/\psi] = P$; since $w \models \psi$, $w' \models P$; thus $w' \models \phi[P/\psi]$;
- if $\phi = \phi_1 \wedge \phi_2$, then, $w \models \phi_1$ and $w \models \phi_2$; for inductive hypothesis, $w' \models \phi_1[P/\psi]$ and $w' \models \phi_2[P/\psi]$; thus, $w' \models \phi[P/\psi]$;
- if $\phi = \phi_1 \vee \phi_2$, then $w \models \phi_1$ or $w \models \phi_2$; consider for example the case $w \models \phi_1$; for inductive hypothesis, $w' \models \phi_1[P/\psi]$; thus, $w' \models \phi[P/\psi]$;
- if $\phi = \mathbf{X}\,\phi_1$, then $w^{1..} \models \phi_1$; for inductive hypothesis, $w'^{1..} \models \phi_1[P/\psi]$; thus, $w' \models \phi[P/\psi]$;
- if $\phi = \phi_1 \,\mathbf{U}\, \phi_2$, then, for some $j \geq 0$, $w^{j..} \models \phi_2$ and, for all $0 \leq k < j$, $w^{k..} \models \phi_1$; for inductive hypothesis, $w'^{j..} \models \phi_2[P/\psi]$ and, for all $0 \leq k < j$, $w'^{k..} \models \phi_1[P/\psi]$; thus, $w' \models \phi[P/\psi]$;
- if $\phi = \phi_1 \,\mathbf{R}\, \phi_2$, then, for all $j \geq 0$, either $w^{j..} \models \phi_2$ and, for some $0 \leq k < j$, $w^{k..} \models \phi_1$; for inductive hypothesis, for all $j \geq 0$, either $w'^{j..} \models \phi_2[P/\psi]$ or, for some $0 \leq k < j$, $w'^{k..} \models \phi_1[P/\psi]$; thus, $w' \models \phi[P/\psi]$;
- if $\phi = r \diamond\!\!\rightarrow \phi_1$, then, for some $j \geq 0$, $w^{0..j} \models\!\!\equiv r$ and $w^{j..} \models \phi_1$; for inductive hypothesis, $w'^{j..} \models \phi_1[P/\psi]$; thus, $w' \models \phi[P/\psi]$;
- if $\phi = r \mapsto \phi$, then, for all $j \geq 0$, if $w^{0..j} \models\!\!\equiv r$, then $w^{j..} \models \phi_1$; for inductive hypothesis, for all $j \geq 0$, if $w^{0..j} \models\!\!\equiv r$, then $w^{j..} \models \phi_1[P/\psi]$; thus, $w' \models \phi[P/\psi]$.

2) Suppose $w' \in \Sigma^{\omega}_{\mathcal{A}'}$, $w' \models \phi'$ and $w = w'_{|\mathcal{A}}$. $w' \models \mathbf{G}\,(P \rightarrow \psi)$ means that, for all $i \geq 0$, if $w'^{i..} \models P$, then $w^{i..} \models \psi$. We prove by induction on the positive occurrence of ψ in ϕ that $w \models \phi$:

- if $\phi = \psi$, then $\phi[P/\psi] = P$; since $w' \models P$, $w \models \psi$; thus $w \models \phi$;
- if $\phi = \phi_1 \wedge \phi_2$, then, $w' \models \phi_1[P/\psi]$ and $w' \models \phi_2[P/\psi]$; for inductive hypothesis, $w \models \phi_1$ and $w \models \phi_2$; thus, $w \models \phi$;
- if $\phi = \phi_1 \vee \phi_2$, then $w' \models \phi_1[P/\psi]$ or $w' \models \phi_2[P/\psi]$; consider for example the case $w' \models \phi_1[P/\psi]$; for inductive hypothesis, $w \models \phi_1$; thus, $w \models \phi$;
- if $\phi = \mathbf{X}\,\phi_1$, then $w'^{1..} \models \phi_1[P/\psi]$; for inductive hypothesis, $w^{1..} \models \phi_1$; thus, $w \models \phi$;
- if $\phi = \phi_1 \,\mathbf{U}\, \phi_2$, then, for some $j \geq 0$, $w'^{j..} \models \phi_2[P/\psi]$ and, for all $0 \leq k < j$, $w'^{k..} \models \phi_1[P/\psi]$; for inductive hypothesis, $w^{j..} \models \phi_2$ and, for all $0 \leq k < j$, $w^{k..} \models \phi_1$; thus, $w \models \phi$;
- if $\phi = \phi_1 \,\mathbf{R}\, \phi_2$, then, for all $j \geq 0$, either $w'^{j..} \models \phi_2[P/\psi]$ or, for some $0 \leq k < j$, $w'^{k..} \models \phi_1[P/\psi]$; for inductive hypothesis, for all $j \geq 0$, either $w^{j..} \models \phi_2$ and, for some $0 \leq k < j$, $w^{k..} \models \phi_1$; thus, $w \models \phi$;
- if $\phi = r \diamond\!\!\rightarrow \phi_1$, then, for some $j \geq 0$, $w^{0..j} \models\!\!\equiv r$ and $w'^{j..} \models \phi_1[P/\psi]$; for inductive hypothesis, $w^{j..} \models \phi_1$; thus, $w \models \phi$;
- if $\phi = r \mapsto \phi$, then, for all $j \geq 0$, if $w^{0..j} \models\!\!\equiv r$, then $w^{j..} \models \phi_1[P/\psi]$; for inductive hypothesis, for all $j \geq 0$, if $w^{0..j} \models\!\!\equiv r$, then $w^{j..} \models \phi_1$; thus, $w \models \phi$.

■

B. Correctness for the encoding of $\mathbf{G}\,(p \rightarrow (r \mapsto p'))$

Theorem 10: Let ϕ be $\mathbf{G}\,(p \rightarrow (r \mapsto p'))$ where p and p' are atoms and r is a SERE. Suppose to have an NFA $A_r = \langle \Sigma, Q, q_0, \rho, F \rangle$ such that $\mathcal{L}(A_r) = \mathcal{L}(r)$. We build an NGBA B_ϕ such that $B_\phi = \langle \Sigma, Q^B, q^B_0, \rho^B, \{F^B\} \rangle$ where:

- $Q^B := 2^Q$
- $q^B_0 := Q^B$
- $\rho^B(L, \ell) := \{L' \mid$
  if $\rho(q_0, \ell_{|\mathcal{A}}) \cap F = \emptyset$, then $L' \models p \rightarrow \rho(q_0, \ell_{|\mathcal{A}})$,
  else $L' \models p \rightarrow (\rho(q_0, \ell_{|\mathcal{A}}) \wedge p')$;
  $L' \models \bigwedge_{q_L \in L, \rho(q_L, \ell_{|\mathcal{A}}) \cap F = \emptyset} \rho(q_L, \ell_{|\mathcal{A}})$,
  $L' \models \bigwedge_{q_L \in L, \rho(q_L, \ell_{|\mathcal{A}}) \cap F \neq \emptyset} \rho(q_L, \ell_{|\mathcal{A}}) \wedge p'\}$
- $F^B := Q^B$.

Then $\mathcal{L}(B_\phi) = \mathcal{L}(\phi)$.

Lemma 2: Let ϕ be $\mathbf{G}\,(p \rightarrow (r \mapsto p'))$ where p and p' are atoms and r is a SERE. Suppose to have an NFA $A_r = \langle \Sigma, Q, q_0, \rho, F \rangle$ such that $\mathcal{L}(A_r) = \mathcal{L}(r)$. Let us consider the ABA $A_\phi = \langle \Sigma, Q^A, q^A_0, \rho^A, F^A \rangle$ where:

- $Q^A := \{q_G, q_{\neg p}, q_{p'}, q_\top, q_\bot\} \cup Q$
- $Q^A_0 := \{q_G\}$
- $\rho^A(q_G, \ell) := q_G \wedge (\rho^A(q_{\neg p}, \ell) \vee \rho^A(q_0, \ell))$

$$\rho^A(q_{\neg p}, \ell) := \begin{cases} q_\top & \text{if } p \notin \ell \\ q_\bot & \text{if } p \in \ell \end{cases}$$

$$\rho^A(q_{p'}, \ell) := \begin{cases} q_\top & \text{if } p' \in \ell \\ q_\bot & \text{if } p' \notin \ell \end{cases}$$

$\rho^A(q_\top, \ell) := q_\top$

$\rho^A(q_\bot, \ell) := q_\bot$

$$\rho^A(q, \ell) := \begin{cases} \rho(q, \ell_{|\mathcal{A}}) \wedge \rho^A(p', \ell) & \text{if } \rho(q, \ell_{|\mathcal{A}}) \cap F \neq \emptyset \\ \rho(q, \ell_{|\mathcal{A}}) & \text{if } \rho(q, \ell_{|\mathcal{A}}) \cap F = \emptyset \end{cases}$$

- $F^A := \{q_G, q_\top\} \cup Q$.

Then $\mathcal{L}(A_\phi) = \mathcal{L}(\phi)$.

Proof: It is the classic construction from PSL to ABA. ■

Lemma 3: Let us consider the NGBA $B'_\phi = \langle \Sigma, Q', q'_0, \rho', \{F'\} \rangle$ where:

- $Q' := \{(L,R) \in 2^{Q^A} \times 2^{Q^A \setminus F^A} \mid R \subseteq L\}$
- $Q'_0 := \{(L,R) \in Q' \mid q_G \in L\}$
- $\rho'((L,R),\ell) := \{(L', R' \setminus F^A) \mid$
  $L' \models \bigwedge_{q_L \in L} \rho^A(q_L,\ell)$,
  if $R = \emptyset$, then $R' = L'$,
  if $R \neq \emptyset$, $R' \subseteq L'$, $R' \models \bigcap_{q_R \in R} \rho^A(q_R,\ell)\}$.
  $\}$
- $F' := 2^{Q^A} \times \{\emptyset\}$.

Then $\mathcal{L}(B'_\phi) = \mathcal{L}(A_\phi)$.

Proof: It is the classic construction from ABA to NGBA (with a slight variant for the initial condition). ■

Proof of Theorem 10: From B'_ϕ, we can derive B_ϕ by simplifying the state space and the transition relation. First, notice that every reachable state of B'_ϕ contains q_G in the left part. Thus, we take the projection of the state-space on the sub-space $\{(L,R) \in Q' \mid q_G \in L\}$. The resulting NGBA is $B''_\phi = \langle \Sigma, Q'', q''_0, \rho'', \{F''\}\rangle$ where:

- $Q'' := \{(L,R) \in 2^{Q^A \setminus \{q_G\}} \times 2^{Q^A \setminus F^A} \mid R \subseteq L\}$
- $Q''_0 := Q''$
- $\rho''((L,R),\ell) := \{(L', R' \setminus F^A) \mid$
  $L' \models (\rho^A(q_{\neg p},\ell) \vee \rho^A(q_0,\ell)) \wedge \bigwedge_{q_L \in L} \rho^A(q_L,\ell)$,
  if $R = \emptyset$, then $R' = L'$,
  if $R \neq \emptyset$, $R' \subseteq L'$, $R' \models \bigcap_{q_R \in R} \rho^A(q_R,\ell)\}$.
  $\}$
- $F' := 2^{Q^A \setminus \{q_G\}} \times \{\emptyset\}$.

It is easy to see that $\mathcal{L}(B''_\phi) = \mathcal{L}(B'_\phi)$.

Then, we remove $q_{\neg p}, q_{p'}, q_\top, q_\bot$ by considering p and p' as variables. In particular, if $\ell_{|\mathcal{A}} = \ell \cap \mathcal{A}$,
$\rho^A(q_{\neg p},\ell) \vee \rho^A(q_0,\ell) \equiv p \rightarrow \rho^A(q_0,\ell)$
$$\rho(q,\ell_{|\mathcal{A}}) \wedge \rho^A(p',\ell) \equiv \begin{cases} \rho(q,\ell_{|\mathcal{A}}) & \text{if } \rho(q,\ell_{|\mathcal{A}}) \cap F = \emptyset \\ \rho(q,\ell_{|\mathcal{A}}) \wedge p' & \text{if } \rho(q,\ell_{|\mathcal{A}}) \cap F \neq \emptyset \end{cases}$$
Finally, we can simplify the automaton and remove the right part which is constantly empty. The fairness set now coincides with the whole set of states. The resulting automaton is B_ϕ. ■

C. Correctness for the encoding of $\mathbf{G}\,(p \rightarrow (r \Diamond\!\!\!\rightarrow p'))$

Theorem 11: Let ϕ be $\mathbf{G}\,(p \rightarrow (r \Diamond\!\!\!\rightarrow p'))$ where p and p' are atoms and r is a SERE. Suppose to have an NFA $A_r = \langle \Sigma, Q, q_0, \rho, F\rangle$ such that $\mathcal{L}(A_r) = \mathcal{L}(r)$. We build an NGBA B_ϕ such that $B_\phi = \langle \Sigma, Q^B, q^B_0, \rho^B, \{F^B\}\rangle$ where:

- $Q^B := \{(L,R) \in 2^Q \times 2^Q \mid R \subseteq L\}$
- $Q^B_0 := Q^B$
- $\rho^B((L,R),\ell) := \{(L',R') \mid$
  if $\rho(q_0,\ell_{|\mathcal{A}}) \cap F = \emptyset$, then $L' \models p \rightarrow \rho(q_0,\ell_{|\mathcal{A}})$,
  else $L' \models p \rightarrow (\rho(q_0,\ell_{|\mathcal{A}}) \vee p')$;
  $L' \models \bigwedge_{q_L \in L, \rho(q_L,\ell_{|\mathcal{A}}) \cap F = \emptyset} \rho(q_L,\ell_{|\mathcal{A}})$,
  $L' \models \bigwedge_{q_L \in L, \rho(q_L,\ell_{|\mathcal{A}}) \cap F \neq \emptyset} \rho(q_L,\ell_{|\mathcal{A}}) \vee p'$,
  if $R = \emptyset$, then $R' = L'$,
  else $R' \subseteq L'$,
  $(\ell, R') \models \bigwedge_{q_R \in R, \rho(q_R,\ell_{|\mathcal{A}}) \cap F = \emptyset} \rho(q_R,\ell_{|\mathcal{A}})$,
  $(\ell, R') \models \bigwedge_{q_R \in R, \rho(q_R,\ell_{|\mathcal{A}}) \cap F \neq \emptyset} \rho(q_R,\ell_{|\mathcal{A}}) \vee p'$.
  $\}$
- $F^B := 2^Q \times \{\emptyset\}$.

Then $\mathcal{L}(B_\phi) = \mathcal{L}(\phi)$.

Lemma 4: Let ϕ be $\mathbf{G}\,(p \rightarrow (r \Diamond\!\!\!\rightarrow p'))$ where p and p' are atoms and r is a SERE. Suppose to have an NFA $A_r = \langle \Sigma, Q, q_0, \rho, F\rangle$ such that $\mathcal{L}(A_r) = \mathcal{L}(r)$. Let us consider the ABA $A_\phi = \langle \Sigma, Q^A, q^A_0, \rho^A, F^A\rangle$ where:

- $Q^A := \{q_G, q_{\neg p}, q_{p'}, q_\top, q_\bot\} \cup Q$
- $Q^A_0 := \{q_G\}$
- $\rho^A(q_G,\ell) := q_G \wedge (\rho^A(q_{\neg p},\ell) \vee \rho^A(q_0,\ell))$
  $\rho^A(q_{\neg p},\ell) := \begin{cases} q_\top & \text{if } p \notin \ell \\ q_\bot & \text{if } p \in \ell \end{cases}$
  $\rho^A(q_{p'},\ell) := \begin{cases} q_\top & \text{if } p' \in \ell \\ q_\bot & \text{if } p' \notin \ell \end{cases}$
  $\rho^A(q_\top,\ell) := q_\top$
  $\rho^A(q_\bot,\ell) := q_\bot$
  $\rho^A(q,\ell) := \begin{cases} \rho(q,\ell_{|\mathcal{A}}) \vee \rho^A(p',\ell) & \text{if } \rho(q,\ell_{|\mathcal{A}}) \cap F \neq \emptyset \\ \rho(q,\ell_{|\mathcal{A}}) & \text{if } \rho(q,\ell_{|\mathcal{A}}) \cap F = \emptyset \end{cases}$
- $F^A := \{q_G, q_\top\}$.

Then $\mathcal{L}(A_\phi) = \mathcal{L}(\phi)$.

Proof: It is the classic construction from PSL to ABA. ■

Lemma 5: Let us consider the NGBA $B'_\phi = \langle \Sigma, Q', q'_0, \rho', \{F'\}\rangle$ where:

- $Q' := \{(L,R) \in 2^{Q^A} \times 2^{Q^A \setminus F^A} \mid R \subseteq L\}$
- $Q'_0 := \{(L,R) \in Q' \mid q_G \in L\}$
- $\rho'((L,R),\ell) := \{(L', R' \setminus F^A) \mid$
  $L' \models \bigwedge_{q_L \in L} \rho^A(q_L,\ell)$,
  if $R = \emptyset$, then $R' = L'$,
  if $R \neq \emptyset$, $R' \subseteq L'$, $R' \models \bigcap_{q_R \in R} \rho^A(q_R,\ell)\}$.
  $\}$
- $F' := 2^{Q^A} \times \{\emptyset\}$.

Then $\mathcal{L}(B'_\phi) = \mathcal{L}(A_\phi)$.

Proof: It is the classic construction from ABA to NGBA (with a slight variant for the initial condition). ■

Proof of Theorem 11: From B'_ϕ, we can derive B_ϕ by simplifying the state space and the transition relation. First, notice that every reachable state of B'_ϕ contains q_G in the left part. Thus, we take the projection of the state-space on the sub-space $\{(L,R) \in Q' \mid q_G \in L\}$. The resulting NGBA is $B''_\phi = \langle \Sigma, Q'', q''_0, \rho'', \{F''\}\rangle$ where:

- $Q'' := \{(L,R) \in 2^{Q^A \setminus \{q_G\}} \times 2^{Q^A \setminus F^A} \mid R \subseteq L\}$
- $Q''_0 := Q''$
- $\rho''((L,R),\ell) := \{(L', R' \setminus F^A) \mid$
  $L' \models (\rho^A(q_{\neg p},\ell) \vee \rho^A(q_0,\ell)) \wedge \bigwedge_{q_L \in L} \rho^A(q_L,\ell)$,
  if $R = \emptyset$, then $R' = L' \setminus F^A$,
  if $R \neq \emptyset$, then there exists $S' \subseteq L'$ such that
  $S' \models \bigwedge_{q_R \in R} \rho^A(q_R,\ell)$ and $R' = S' \setminus F^A$.
  $\}$
- $F' := 2^{Q^A \setminus \{q_G\}} \times \{\emptyset\}$.

It is easy to see that $\mathcal{L}(B''_\phi) = \mathcal{L}(B'_\phi)$.

Then, we remove $q_{\neg p}, q_{p'}, q_\top, q_\bot$ by considering p and p' as variables. In particular, if $\ell_{|\mathcal{A}} = \ell \cap \mathcal{A}$,
$\rho^A(q_{\neg p},\ell) \vee \rho^A(q_0,\ell) \equiv p \rightarrow \rho^A(q_0,\ell)$

$$\rho(q,\ell_{|\mathcal{A}}) \vee \rho^{A}(p',\ell) \equiv \begin{cases} \rho(q,\ell_{|\mathcal{A}}) & \text{if } \rho(q,\ell_{|\mathcal{A}}) \cap F = \emptyset \\ \rho(q,\ell_{|\mathcal{A}}) \vee p' & \text{if } \rho(q,\ell_{|\mathcal{A}}) \cap F \neq \emptyset \end{cases}$$

The resulting automaton is B_ϕ. ■

D. Proof of Rules for Simplification of Suffix Operators

Theorem 13: For each of the following rewriting rules, the formula on the left is equivalent to the formula on the right:

$$\boxed{\begin{array}{rcl}
\{\mathbf{[0^*]}\} \Diamond\!\!\rightarrow \phi & \Rightarrow & False \\
\{b\} \Diamond\!\!\rightarrow \phi & \Rightarrow & b \wedge \phi \\
\{r_1 \mathbin{\mathbf{:}} r_2\} \Diamond\!\!\rightarrow \phi & \Rightarrow & \{r_1\} \Diamond\!\!\rightarrow (\{r_2\} \Diamond\!\!\rightarrow \phi) \\
\{r_1 \mathbin{\mathbf{;}} r_2\} \Diamond\!\!\rightarrow \phi & \Rightarrow^{\dagger} & \{r_1\} \Diamond\!\!\rightarrow \mathbf{X}\, (\{r_2\} \Diamond\!\!\rightarrow \phi) \\
\{r_1 \mid r_2\} \Diamond\!\!\rightarrow \phi & \Rightarrow & (\{r_1\} \Diamond\!\!\rightarrow \phi) \vee (\{r_2\} \Diamond\!\!\rightarrow \phi) \\
\{r \mathbin{\mathbf{;}} b\mathbf{[^*]}\} \Diamond\!\!\rightarrow \phi & \Rightarrow^{\ddagger} & \{r\} \Diamond\!\!\rightarrow ((\mathbf{X}\, b)\, \mathbf{U}\, \phi) \\
\{b\mathbf{[^*]} \mathbin{\mathbf{;}} r\} \Diamond\!\!\rightarrow \phi & \Rightarrow^{\ddagger} & b\, \mathbf{U}\, (\{r\} \Diamond\!\!\rightarrow \phi) \\
 & & \\
\{\mathbf{[0^*]}\} \mapsto \phi & \Rightarrow & True \\
\{b\} \mapsto \phi & \Rightarrow & b \rightarrow \phi \\
\{r_1 \mathbin{\mathbf{:}} r_2\} \mapsto \phi & \Rightarrow & \{r_1\} \mapsto (\{r_2\} \mapsto \phi) \\
\{r_1 \mathbin{\mathbf{;}} r_2\} \mapsto \phi & \Rightarrow^{\dagger} & \{r_1\} \mapsto \mathbf{X}\, (\{r_2\} \mapsto \phi) \\
\{r_1 \mid r_2\} \mapsto \phi & \Rightarrow & (\{r_1\} \mapsto \phi) \wedge (\{r_2\} \mapsto \phi) \\
\{r \mathbin{\mathbf{;}} b\mathbf{[^*]}\} \mapsto \phi & \Rightarrow^{\ddagger} & \{r\} \mapsto ((\mathbf{X}\, \neg b)\, \mathbf{R}\, \phi) \\
\{b\mathbf{[^*]} \mathbin{\mathbf{;}} r\} \mapsto \phi & \Rightarrow^{\ddagger} & \neg b\, \mathbf{R}\, (\{r\} \mapsto \phi)
\end{array}}$$

Proof: We prove only the rules concerning the suffix conjunction. The corresponding rules for the suffix implication can be obtained with the equivalences $r \mapsto \phi \Leftrightarrow \neg(r \Diamond\!\!\rightarrow \neg\phi)$ and $\phi_1\, \mathbf{R}\, \phi_2 \Leftrightarrow \neg(\neg\phi_1\, \mathbf{U}\, \neg\phi_2)$.

- For any PSL formula ϕ and infinite word w, $w \models \{\mathbf{[0^*]}\} \Diamond\!\!\rightarrow \phi$ iff there exists $i \geq 0$ such that $w^{0..i} \models \mathbf{[0^*]}$ iff $w \models False$ (because $w^{0..i} \neq \epsilon$).
- For any PSL formula ϕ, Boolean expression b, and infinite word w, $w \models \{b\} \Diamond\!\!\rightarrow \phi$ iff there exists $i \geq 0$ such that $w^{0..i} \models b$ and $w^{i..} \models \phi$ iff $w \models b \wedge \phi$ (because i must be 0).
- For any PSL formula ϕ, SERE r_1 and r_2, and infinite word w, $w \models \{r_1 \mathbin{\mathbf{:}} r_2\} \Diamond\!\!\rightarrow \phi$ iff there exist $0 \leq i \leq j$ such that $w^{0..i} \models r_1$, $w^{i..j} \models r_2$, and $w^{j..} \models \phi$ iff $w \models \{r_1\} \Diamond\!\!\rightarrow (\{r_2\} \Diamond\!\!\rightarrow \phi)$
- For any PSL formula ϕ, SERE r_1 and r_2, and infinite word w, if $\epsilon \notin \mathcal{L}(r_1)$ and $\epsilon \notin \mathcal{L}(r_2)$, $w \models \{r_1 \mathbin{\mathbf{;}} r_2\} \Diamond\!\!\rightarrow \phi$ iff there exist $0 \leq i < j$ such that $w^{0..i} \models r_1$, $w^{i+1..j} \models r_2$, and $w^{j..} \models \phi$ iff there exists $i \geq 0$ s.t. $w^{0..i} \models r_1$, $w^{i+1..} \models r_2 \Diamond\!\!\rightarrow \phi$ iff $w \models \{r_1\} \Diamond\!\!\rightarrow \mathbf{X}\, (\{r_2\} \Diamond\!\!\rightarrow \phi)$
- For any PSL formula ϕ, SERE r_1 and r_2, and infinite word w, $w \models \{r_1 \mid r_2\} \Diamond\!\!\rightarrow \phi$ iff there exists $i \geq 0$ s.t. $w^{0..i} \models r_1 \mid r_2$ and $w^{i..} \models \phi$ iff $w \models \{r_1\} \Diamond\!\!\rightarrow \phi \vee \{r_2\} \Diamond\!\!\rightarrow \phi$
- For any PSL formula ϕ, SERE r, Boolean expression b, and infinite word w, if $\epsilon \notin \mathcal{L}(r)$ $w \models \{r \mathbin{\mathbf{;}} b\mathbf{[^*]}\} \Diamond\!\!\rightarrow \phi$ iff there exist $0 \leq i \leq j$ such that $w^{0..i} \models r$, $w^{i+1..j} \models b\mathbf{[^*]}$, and $w^{j..} \models \phi$ iff there exist $0 \leq i \leq j$ such that $w^{0..i} \models r$, $w^{j..} \models \phi$, and for all $i+1 \leq h \leq j$ $w^h \models b$ iff there exist $0 \leq i \leq j$ such that $w^{0..i} \models r$, $w^{j..} \models \phi$, and for all $i \leq h < j$ $w^{h..} \models \mathbf{X}\, b$ iff $w \models \{r\} \Diamond\!\!\rightarrow ((\mathbf{X}\, b)\, \mathbf{U}\, \phi)$.
- For any PSL formula ϕ, SERE r, Boolean expression b, and infinite word w, if $\epsilon \notin \mathcal{L}(r)$, then $w \models \{b\mathbf{[^*]} \mathbin{\mathbf{;}} r\} \Diamond\!\!\rightarrow \phi$ iff there exist $0 \leq i \leq j$ such that $w^{0..i-1} \models b\mathbf{[^*]}$, $w^{i..j} \models r$, and $w^{j..} \models \phi$ iff there exist $0 \leq i \leq j$ such that $w^{i..j} \models r$, $w^{j..} \models \phi$, and for all $0 \leq h \leq i-1$ $w^h \models b$ iff $b\, \mathbf{U}\, \{r\} \Diamond\!\!\rightarrow \phi$.

■

E. Proof of rule for Star Extraction

Let us consider the rule:

$$\boxed{\mathbf{G}\,(P \rightarrow (\{r\mathbf{[^*]}\} \mapsto P')) \quad \Rightarrow \quad \mathbf{G}\,(P \rightarrow (\{r\} \mapsto (P' \wedge \mathbf{X}\, P)))}$$

We first note that the rule is not correct in the general case. In fact, we rely on two assumptions:

1) the formula being transformed is in Suffix Operator Normal Form (SONF);
2) the subformula we are replacing is a Suffix Operator Subformula (SOS). This means that the input formula is in the form $\phi \wedge \phi'$ where ϕ' is the left-hand side of fig 4, P and P' are fresh variables introduced in the SONF-ization process; in particular P occurs only positively in ϕ.

However, although the two formulas are not equivalent, the transformation preserves the language with regards to the original alphabet.

Formally stated, let ϕ and ϕ' be resp. the left-hand and the right-hand side of the rule. Then, if a word w satisfies ϕ, there exists w' satisfying ϕ' such that the restrictions of w and w' to the original alphabet (without P and P') are equal.

Theorem 14: Let $\mathcal{A}$ a set of propositions. Let ψ be an PSL formula over $\mathcal{A}$. Let ψ' be the SONF of ψ over $\mathcal{A}' = \mathcal{A} \cup \mathcal{A}_N$ where $\mathcal{A}_N$ is a set of new variables. Suppose $\psi' = \mathbf{G}\,(P \rightarrow (r\mathbf{[^*]} \mapsto P')) \wedge \varphi$ for some PSL formula φ, SERE r and variables $P, P' \in \mathcal{A}_N$. Let $\psi'' = \mathbf{G}\,(P \rightarrow (\{r\} \mapsto P' \wedge \mathbf{X}\, P)) \wedge \varphi$. Then $\mathcal{L}(\psi') = \mathcal{L}(\psi'')$.

Proof:

$\Longrightarrow$ Let $\Sigma' = 2^{\mathcal{A}}$. We prove that if $w \in \Sigma'^{\omega}$ and $w \models \psi'$, there exists $w' \in \Sigma'^{\omega}$ such that $w' \models \psi''$ and $w^i \cap \mathcal{A} = w'^i \cap \mathcal{A}$ for every $i \geq 0$.
Let w a word such that: $w \models \mathbf{G}\,(P \rightarrow (r\mathbf{[^*]} \mapsto P')) \wedge \varphi$,
then we can construct w' as follows

$$w'^i := \begin{cases} w^i \cup \{P\} \text{ if } \exists j, 0 \leq j < i, P \models_b w^j, \\ \qquad\qquad \text{and } w^{j..(i-1)} \models_r r\mathbf{[^*]} \\ w^i \text{ otherwise} \end{cases}$$

By construction, $w' \models \mathbf{G}\,(P \rightarrow (\{r\} \mapsto \mathbf{X}\, P))$
Since $w \models \mathbf{G}\,(P \rightarrow (r\mathbf{[^*]} \mapsto P'))$ we deduce that $w' \models \mathbf{G}\,(P \rightarrow (\{r\} \mapsto P'))$
Then $w' \models \mathbf{G}\,(P \rightarrow (\{r\} \mapsto P' \wedge \mathbf{X}\, P))$
also $w' \models \varphi$ because P occurs only positively in φ
Then $w' \models \psi''$

$\Longleftarrow$ If $w \models \psi''$, then $w \models \psi'$. In fact, suppose $i \geq 0$ such that $P \models_b w^i$ and there exists $j \geq i$ such that $w^{i..j} \models_r r*$.
Then $\exists h_1, ..., h_n$ such that $w^{i..h_1-1} \models_r r$, $w^{h_1..h_2-1} \models_r r, \ldots, w^{h_n..j} \models_r r$.
Then $P \models_b w^{h_1}$, $P \models_b w^{h_2}, \ldots\ P \models_b w^{h_n}$.
Then $P' \models_b w^j$. Then $w \models \psi'$.

■