SVA and PSL Local Variables - A Practical Approach

Roy Armoni, Dana Fisman, and Naiyong Jin

Synopsys

Abstract. SystemVerilog Assertions (SVA), as well as Property Specification Language (PSL) are linear temporal logics based on LTL [14], extended with regular expressions and local variables. In [6] Bustan and Havlicek show that the local variable extensions, as well as regular expressions with intersection, render the verification problem of SVA and PSL formulae EXPSPACE-complete. In this paper we show a practical approach for the verification problem of SVA and PSL with local variables. We show that in practice, for a significant and meaningful subsets of those languages, which we denote PSL^{pract}, local variables do not increase the complexity of their verification problem, keeping it in PSPACE.

1 Introduction

SystemVerilog is an industrial standard unified hardware design and verification language. SystemVerilog Assertions (SVA) [11,7] is a subclass of SystemVerilog, used to declaratively specify functional behaviours of hardware designs. Similarly, Property Specification Language (PSL) [10,8] is used to declaratively specify functional behaviours of hardware designs independent of the design language. Typically, both SVA and PSL properties are then either validated during dynamic simulation or formally verified.

Several works during the recent decade defined industrial functional specification languages and studied the complexity of the verification problem of those languages [2,1,4,6].[1] SVA, part of the IEEE 1800 SystemVerilog standard, is a linear time temporal logic, based on Pnueli's LTL [14], extended with syntactic sugaring, regular expressions with intersection, connectives between regular expressions and formulae, and local variables. PSL, IEEE 1850 standard, has similar constructs except that its local variables' semantics differs slightly [9].

The expressiveness of SVA and PSL as well as the worst case complexity of their verification problem have already been studied before. Armoni et al [1] show that the expressive power of LTL extended with regular expressions (the ForSpec temporal logic) is exactly all the ω-regular languages. They also show that the complexity of the verification problem for this language is PSPACE-complete in the absence of time windows, and becomes EXPSPACE-complete when time windows are present. Bustan and Havlicek [6] further investigate the complexity of SVA verification with four independent extensions, namely intersection of regular expressions, local variables, PSL flavoured quantified variables, and additional syntactic sugaring in the form of property

[1] Given a design $\mathcal{M}$ and a temporal logic formula φ the *verification problem* asks whether $\mathcal{M}$ satisfies φ (i.e. whether φ holds on all computations of $\mathcal{M}$).

declarations with arguments. None of this constructs enhances the expressiveness of the language, but they do add succinctness. As for complexity, [6] show that each of these additions results in bringing the verification problem to the EXPSPACE-complete class. Yet, they conclude that the usefulness of the discussed constructs overshadows their cost, thus using them is worthwhile.

In light of this discussion we would like to distinguish between syntactic sugaring and regular expressions intersection on the one hand, and different forms of variables on the other hand. We claim that property declarations with arguments usually do not add any burden to the complexity of the verification of SVA. The EXPSPACE-hardness is obtained by deep nested declarations that expand a property of exponential size, which is not a typical usage of this feature. Similarly a sporadic usage of regular expression intersections, which is how intersections are usually used, does not significantly increase the complexity of the SVA verification problem. As in property declaration, the EXPSPACE-hardness is obtained by a long series of intersections, a rare sight of this operator.

In contrast, local variables create a complexity hurdle more easily. The upper bound of [6] is achieved by constructing an alternating Büchi automaton of size proportional to the size of the property and the size of the Cartesian product of the domains of the local variables. Seeing local variables of large domain is very common, for instance when asserting data consistency on bus protocols. Thus, a 64-bit bus results in a single variable domain of size 2^{64}. Building an alternating automaton of more than 2^{64} states as proposed in [6] may be possible, but for model checking we translate it to a non-deterministic Büchi automaton of more than $2^{2^{64}}$ states, represented by 2^{64} state variables, which is clearly infeasible.

In this paper we show that despite the theoretical lower bound shown by [6], there is a significant subset of properties with local variables for which the verification problem is in PSPACE. In particular, this subset subsumes the *simple subset* of PSL [10], which is the subset supported by most dynamic/formal verification tools for PSL/SVA.

We refer to this subset as the *practical subset*, and denote it PSL^{pract}. The precise definition of the practical subset is given in Definition 6. Intuitively, any formula with local variables limited to the monotone Boolean connectives, the temporal operators *next* , *until* , *releases* and *suffix implication* belongs to the practical subset as long as there are no assignments to local variables on either the right operand of *until* or the left operand of *releases* . The formal definition allows negation to be applied only to non-temporal expression. In order to deal with negation applied to temporal operators, we can add a pre-processing step that given a PSL formula transform it to an equivalent formula in positive normal form, by using duality rules (formally given in Definition 2).

We claim that the verification problem for the practical subset is in PSPACE, namely:

Theorem 1. *The space complexity of the verification problem of any formula φ in PSL^{pract} is polynomial in $|\varphi|$.*

The proof of Theorem 1 is done by constructing an alternating Büchi automaton $\mathcal{A}$ whose size $|\mathcal{A}|$ is polynomial in $|\varphi|$ and its language $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$. By that we reduce the verification problem of formulae to the emptiness problem of alternating Büchi automata. Our key idea is to separate the part responsible for local variables updates from the alternating automaton, and use a satellite that monitors the automaton to determine the local variables' next state value. Then, in order to transform the alternating automaton into a non-deterministic one, we apply the Miyano-Hayashi construction [13] on an alternating automaton that is constructed for the property φ' obtained from φ by disregarding the local variables, thus avoiding the exponential penalty of the Miyano-Hayashi construction on the local-variables-part of the formula. Such a manipulation is valid conditioned in a run tree of the alternating automaton there are no two states at the same depth that disagree on the value of a local variable. We call an automaton adhering to this property *conflict-free*. Conflicts may arise because of updates after a universal branch. If the universal branch is not involved in a loop we can solve the conflict by introducing new local variables to the different branches; however, if it is involved in a loop we simply cannot. The construction for formulae of the practical subset guarantees that no local variable assignment occurs after a loop that contains a universal branch.

The idea of dealing with a subset of properties where there are no conflicts between local variables assignments appears in [12]. There as well it is observed that the complexity hurdle comes from the fact that overlapping instances, in general, may carry different values for the same local variables. And that if local variables are confined to certain positions in the property, then one can guarantee overlapping instances will agree on the value of the local variables. However, it is hard to infer from [12] what is the supported subset. Moreover, the implementation via alternating automata extended to local variables, and the separation to satellite, as well as the PSPACE claim and proof are novel to our paper.

We note that PSL$^{\mathit{pract}}$ subsumes PSL$^{\mathit{simple}}$, the *simple subset* of PSL [10], which is the subset supported by most dynamic/formal verification tools for PSL/SVA. This subset conforms to the notion of monotonic advancement of time, left to right through the property. The syntactic restrictions of PSL$^{\mathit{pract}}$ actually relax those of PSL$^{\mathit{simple}}$: whenever the latter requires an operand to be non-temporal, the former demands just that it does not bear assignments to local variables. We are thus confident that most commonly used properties fall into this subset.

We remark that the source of the subtle differences between the definition of local variables in PSL and SVA lies in the definition of the semantics of intersection [9] which is orthogonal to the discussion in this paper. In the absence of intersection, the semantics of SVA with local variables is exactly the same as that of PSL, except for the scope of the local variables. While PSL uses *new* and *free* to define the scope boundaries of a local variable, SVA assumes a global scope for all local variables. Thus an SVA formula of the practical subset is translatable to PSL by adding *new* for all variables at the beginning of the formula.

2 Background

We provide the syntax and semantics for the core of PSL with local variables, which we denote PSL$^{+\mathrm{V}}$. Throughout the paper we assume $\mathcal{P}$ is a non-empty set of atomic propositions and $\mathcal{V}$ is a set of local variables with domain $\mathcal{D}$. We further assume a given set $\mathcal{E}$ of (not necessarily Boolean) expressions over $\mathcal{P} \cup \mathcal{V}$, and a given set $\mathcal{B} \subseteq \mathcal{E}$ of Boolean expressions. That is, a Boolean expression may *refer* to both atomic propositions and local variables. For instance $p \wedge x{=}7$ is a Boolean expression stating that proposition $p \in \mathcal{P}$ holds and local variable $x \in \mathcal{V}$ has the value 7. In contrast, *assignments* to local variables are not part of a Boolean expression. They are given separately as the second component of an *assignment pair* whose first component is a Boolean expression. For instance, $(b, x := 7)$ is an assignment pair that reads "the boolean expression b holds and local variable x should be assigned 7". Any expression $e \in \mathcal{E}$ can be assigned to a local variable. It is also allowed to have a sequence of local variable assignments in an assignment pair. Given a sequence of local variable $\mathrm{X} = x_1, \ldots, x_n$ and a sequence of expressions $\mathrm{E} = e_1, \ldots, e_n$ of the same length we write $\mathrm{X} := \mathrm{E}$ to abbreviate $x_1 := e_1, \ldots, x_n := e_n$. Thus, $(b, \mathrm{X} := \mathrm{E})$ is a legal assignment pair.

Formulae of PSL are defined with respect to regular expressions extended with local variables (RE$^{+\mathrm{V}}$s). The atoms of an RE$^{+\mathrm{V}}$ are Boolean expressions or assignment pairs. On top of these the regular operators of Concatenation, Kleene's closure and Or are applied.

We clarify that PSL/SVA with local variables are already a part of the respective standards [10,11].

2.1 Syntax of PSL$^{+\mathrm{V}}$

Definition 1 (Regular expressions extended with local variables (RE$^{+\mathrm{V}}$s)). *Let $b \in \mathcal{B}$ be a Boolean expression. Let* X *be a sequence of local variables and* E *a sequence of expressions of the same length as* X*. The grammar below defines regular expressions r with local variables.*

$$r ::= b \mid (b, \mathrm{X} := \mathrm{E}) \mid \lambda \mid r \cdot r \mid r \cup r \mid r^{+} \mid (\mathit{new}(\mathrm{X})\ r) \mid (\mathit{free}(\mathrm{X})\ r)$$

Where $(b, \mathrm{X} := \mathrm{E})$ stipulates that b holds and the variables in X *are assigned with expressions in* E*, $(\mathit{new}(\mathrm{X})\ r)$ declares the local variables $x \in \mathrm{X}$ in parenthesis scope, and $(\mathit{free}(\mathrm{X})\ r)$ removes the local variables $x \in \mathrm{X}$ from parenthesis scope.*

PSL formulae are built out of RE$^{+\mathrm{V}}$s. The usual negation, disjunction and conjunction can be applied to PSL formulae. The temporal operators consist of the *next*, *until*, *releases* and the *suffix implication* operator $\mapsto$ aka *triggers*. Loosely speaking $r \mapsto \varphi$ holds on a word w if every prefix of w that matches r is followed by a suffix on which φ holds.

Definition 2 (PSL$^{+\mathrm{V}}$ formulae). *Let r be an RE$^{+\mathrm{V}}$,* X *a sequence of local variables and* E *a sequence of expressions of the same length as* X*. The grammar below defines PSL$^{+\mathrm{V}}$ formulae.*

$$\varphi ::= r! \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathit{next}\ \varphi \mid \varphi\ \mathit{until}\ \varphi \mid r \mapsto \varphi \mid (\mathit{new}(\mathrm{X})\ \varphi) \mid (\mathit{free}(\mathrm{X})\ \varphi)$$

We use the following common syntactic sugaring operators: $\varphi_1 \vee \varphi_2 \overset{\mathrm{def}}{=} \neg\varphi_1 \wedge \neg\varphi_2$, $\mathit{eventually}\ \varphi \overset{\mathrm{def}}{=} \mathit{true}\ \mathit{until}\ \varphi$, $\mathit{always}\ \varphi \overset{\mathrm{def}}{=} \neg\, \mathit{eventually}\ \neg\varphi$, $\varphi_1\ \mathit{releases}\ \varphi_2 \overset{\mathrm{def}}{=} \neg(\neg\varphi_1\ \mathit{until}\ \neg\varphi_2)$, $r \diamond\!\!\rightarrow \varphi \overset{\mathrm{def}}{=} \neg(r \mapsto \neg\varphi)$.

Example 1. *Let $\{start, end, data\_in, data\_out\} \subseteq \mathcal{P}$, and $x \in \mathcal{V}$. Then, the formula $(\mathit{new}(x)\ \varphi_1)$ where*

$$\varphi_1 = \textit{always}\ (((\neg start)^+ \cdot (start, x := data\_in)) \mapsto (\neg end)^+ \cdot end \wedge (data\_out = x))$$

states that if the value of $data\_in$ *is* x *when transaction starts (signal start rises) then the value of* $data\_out$ *should be* x *when the transaction ends (signal end rises). That is, values are transferred correctly from* $data\_in$ *to* $data\_out$.

Example 2. *Let* $\{start, end, get, put\} \subseteq \mathcal{P}$, *and* $x \in \mathcal{V}$. *Assume put, get and end are mutually exclusive (that is, if one of them holds the others do not). Let* not_pge *denote the expression* $(\neg put \wedge \neg get \wedge \neg end)$. *Then the formula* $(new(x)\ \varphi_2)$ *where*

$$\varphi_2 = \textit{always}\ (((start, x := 0) \cdot (not\_pge \cup (put, x{+}{+}) \cup (get, x{-}{-}))^+ \cdot end) \mapsto x{=}0)$$

states that the number of puts and gets between start and end should be the same. More accurately, since the domain of variables is bounded, the number should be the same modulo the size of $\mathcal{D}$ *assuming* ++ *and* −− *increment and decrement modulo* $|\mathcal{D}|$, *respectively.*

Note that this formula is a safety *formula and it does not demand seeing the end of a transaction (signal end holds) once a transaction has started (signal start holds). One thus might use instead the following* liveness *formula, which does demand that.*

$$\varphi_3 = \textit{always}\ ((start, x := 0) \mapsto ((not\_pge \cup (put, x{+}{+}) \cup (get, x{-}{-}))^+ \cdot end \cdot x{=}0))$$

2.2 Semantics of PSL^{+V}

The semantics of PSL^{+V} formulae is defined with respect to a word over the alphabet $\Sigma = 2^{\mathcal{P}}$ (the set of all possible valuations of the atomic propositions) and a letter from the alphabet $\Gamma = \mathcal{D}^{\mathcal{V}}$ (the set of all possible valuations of the local variables). The semantics of RE^{+V} is defined with respect to words from the alphabet $\Lambda = \Sigma \times \Gamma \times \Gamma$. We call words over Λ *extended words*. A letter $\langle \sigma, \gamma, \gamma' \rangle$ of an extended word provides a valuation σ of the atomic propositions and two valuations γ and γ' of the local variables. The valuation γ corresponds to the value of the local variables before assignments have taken place, the *pre-value*, and the valuation γ' corresponds to the value of the local variables after they have taken place, the *post-value*.

In the sequel we use σ to denote letters from the alphabet Σ, γ to denote letters from Γ, and $\mathbb{a}$ to denote letters from Λ. We use u, v, w to denote words over Σ and $\mathbb{u}, \mathbb{v}, \mathbb{w}$ to denote words over Λ.

We use i, j and k to denote non-negative integers. We denote the i^{th} letter of v by v^{i-1} (since counting of letters starts at zero). We denote by $v^{i..}$ the suffix of v starting at v^i, and by $v^{i..j}$ the finite sequence of letters starting from v^i and ending at v^j. We use $|v|$ to denote the *length* of v. The empty word ϵ has length 0, a finite word $\sigma_0\sigma_1 \ldots \sigma_k$ has length $k+1$, and an infinite word v has length ∞. The notations for extended words $\mathbb{v}^i$, $\mathbb{v}^{i..}$, $\mathbb{v}^{i..j}$, $|\mathbb{v}|$ are defined in the same manner.

Let $\mathbb{v} = \langle \sigma_0, \gamma_0, \gamma_0' \rangle \langle \sigma_1, \gamma_1, \gamma_1' \rangle \cdots$ be a word over Λ. We use $\mathbb{v}|_\sigma$,$\mathbb{v}|_\gamma$,$\mathbb{v}|_{\gamma'}$ to denote the projection of $\mathbb{v}$ onto the first, second or third component, respectively, of each letter. That is, $\mathbb{v}|_\sigma = \sigma_0\sigma_1\cdots$, $\mathbb{v}|_\gamma = \gamma_0\gamma_1\cdots$, and $\mathbb{v}|_{\gamma'} = \gamma_0'\gamma_1'\cdots$. We use $\mathbb{v}|_{\sigma\gamma}$ to denote the projection of $\mathbb{v}$ onto both the first and second components. That is, $\mathbb{v}|_{\sigma\gamma} = \langle \sigma_0, \gamma_0 \rangle \langle \sigma_1, \gamma_1 \rangle \cdots$. We say that an extended word $\mathbb{v}$ is *good* if for every

$i \in \{0, 1, \ldots, |\mathbb{v}| - 2\}$ we have $(\mathbb{v}|_{\gamma})^{i+1} = (\mathbb{v}|_{\gamma'})^{i}$, i.e., the pre-value of the local variables at letter $i+1$ is the post-value at letter i.

2.2.1 Semantics of Expressions

An expression $e \in \mathcal{E}$ over $\mathcal{P} \cup \mathcal{V}$ is identified with a mapping $e : \Sigma \times \Gamma \mapsto \mathcal{D}$ where $\mathcal{D}$ is the domain of variables in $\mathcal{V}$. A Boolean expression $b \in \mathcal{B}$ is an expression whose domain is $\{\mathrm{T}, \mathrm{F}\}$, and we define *true* and *false* to be the Boolean expressions whose domains are $\{\mathrm{T}\}$ and $\{\mathrm{F}\}$, respectively.

We assume that for an atomic proposition p we have that $p(\sigma, \gamma) = \mathrm{T}$ if $p \in \sigma$ and F otherwise, and that for a local variable v we have that $v(\sigma, \gamma)$ returns the value of v in γ. We sometimes abuse notation by writing simply $p(\sigma)$ and $v(\gamma)$. We assume that operators are closed under $\mathcal{D}$ and behave in the usual manner, i.e. that for $\sigma \in \Sigma, \gamma \in \Gamma$, $e, e_1, e_2 \in \mathcal{E}$, a binary operator $\otimes$ and a unary operator $\odot$ we have $e_1(\sigma, \gamma) \otimes e_2(\sigma, \gamma) = (e_1 \otimes e_2)(\sigma, \gamma)$ and $\odot(e(\sigma, \gamma)) = (\odot e)(\sigma, \gamma)$. In particular, we assume that Boolean disjunction, conjunction and negation behave in the usual manner.

We use $:=$ for local variable *assignments*. Given a local variable x and an expression e we write $[\![x := e]\!](\sigma, \gamma)$ to denote the valuation $\hat{\gamma}$ such that $x(\hat{\gamma}) = e(\sigma, \gamma)$ and for every local variable $v \in \mathcal{V} \setminus \{x\}$ we have that $v(\hat{\gamma}) = v(\gamma)$. Sequence of assignments are evaluated left to right. Formally, given a sequence of local variable $\mathrm{X} = x_1, \ldots, x_n$ and a sequence of expressions $\mathrm{E} = e_1, \ldots, e_n$ of the same length, we write $[\![x_1 := e_1, \ldots, x_n := e_n]\!](\sigma, \gamma)$ to denote the following recursive application: $[\![x_2 := e_2, \ldots, x_n := e_n]\!](\sigma, [\![x_1 := e_1]\!](\sigma, \gamma))$. Recall that we write $\mathrm{X} := \mathrm{E}$ to abbreviate $x_1 := e_1, \ldots, x_n := e_n$. More generally, we use $\mathcal{U}$ to denote the set of all possible sequences of assignments to variables over $\mathcal{V}$. We use $\mathrm{U}, \mathrm{U}_1, \mathrm{U}_2$ to denote elements of $\mathcal{U}$ and use ε to denote the empty assignment sequence. For $\mathrm{U}_1, \mathrm{U}_2 \in \mathcal{U}$ we use $\mathrm{U}_1 \cdot \mathrm{U}_2$ to denote the application of assignments U_2 after assignments in U_1 took place.

2.2.2 Semantics of RE$^{+\mathrm{V}}$s

The semantics of RE$^{+\mathrm{V}}$ is defined with respect to a finite good word over Λ and a set of local variables $\mathrm{Z} \subseteq \mathcal{V}$, and is given in Definition 3. The role of the set Z, which is referred to as the set of *controlled variables*, is to support scoping. Any variable in Z (i.e. a variables in scope) must keep its value if not assigned and take on the assigned value otherwise, whereas any variable not in Z (i.e. a variables not in scope) is free to take on any value.

Let $\mathrm{Z} \subseteq \mathcal{V}$. We use $\gamma_1 \overset{\mathrm{Z}}{\sim} \gamma_2$ (read "γ_1 agrees with γ_2 relative to Z") to denote that for every $z \in \mathrm{Z}$ we have that $z(\gamma_1) = z(\gamma_2)$. We say that good word $\mathbb{v}$ *preserves* Z if for every $z \in \mathrm{Z}$ and for every $i < |\mathbb{v}|$ we have $z(\mathbb{v}^i|_{\gamma'}) = z(\mathbb{v}^i|_{\gamma})$.

Definition 3 (Tight satisfaction). *Let b be a Boolean expression, r, r_1, r_2 RE$^{+\mathrm{V}}$s. Let* Z *be a set of local variables,* X *be a sequence of local variables and* E *a sequence of expression of same size. Let $\mathbb{v}, \mathbb{v}_1, \ldots, \mathbb{v}_k$ be good extended words. The notation $\mathbb{v} \models_{\mathrm{Z}} r$ means that $\mathbb{v}$ tightly satisfies r with respect to the controlled variables* Z.

- $\mathbb{v} \models_{\mathrm{Z}} b \iff |\mathbb{v}| = 1$ *and* $b(\mathbb{v}^0|_{\sigma\gamma}) = \mathrm{T}$ *and* $\mathbb{v}^0|_{\gamma'} \overset{\mathrm{Z}}{\sim} \mathbb{v}^0|_{\gamma}$
- $\mathbb{v} \models_{\mathrm{Z}} (b, \mathrm{X} := \mathrm{E}) \iff |\mathbb{v}| = 1$ *and* $b(\mathbb{v}^0|_{\sigma\gamma}) = \mathrm{T}$ *and* $\mathbb{v}^0|_{\gamma'} \overset{\mathrm{Z}}{\sim} [\![\mathrm{X} := \mathrm{E}]\!](\mathbb{v}^0|_{\sigma\gamma})$
- $\mathbb{v} \models_{\mathrm{Z}} \lambda \iff \mathbb{v} = \epsilon$

- $\mathbb{v} \models\!\!\!\equiv_{\mathrm{Z}} r_1 \cdot r_2 \iff \exists \mathbb{v}_1, \mathbb{v}_2$ *such that* $\mathbb{v} = \mathbb{v}_1\mathbb{v}_2$ *and* $\mathbb{v}_1 \models\!\!\!\equiv_{\mathrm{Z}} r_1$ *and* $\mathbb{v}_2 \models\!\!\!\equiv_{\mathrm{Z}} r_2$
- $\mathbb{v} \models\!\!\!\equiv_{\mathrm{Z}} r_1 \cup r_2 \iff \mathbb{v} \models\!\!\!\equiv_{\mathrm{Z}} r_1$ *or* $\mathbb{v} \models\!\!\!\equiv_{\mathrm{Z}} r_2$
- $\mathbb{v} \models\!\!\!\equiv_{\mathrm{Z}} r^+ \iff \exists k \geq 1$ *and* $\mathbb{v}_1, \mathbb{v}_2, \ldots, \mathbb{v}_k$ *such that* $\mathbb{v} = \mathbb{v}_1\mathbb{v}_2 \ldots \mathbb{v}_k$ *and* $\mathbb{v}_i \models\!\!\!\equiv_{\mathrm{Z}} r$ *for every* $1 \leq j \leq k$
- $\mathbb{v} \models\!\!\!\equiv_{\mathrm{Z}} (\mathit{new}(\mathrm{X})\ r) \iff \mathbb{v} \models\!\!\!\equiv_{\mathrm{Z} \cup \mathrm{X}} r$
- $\mathbb{v} \models\!\!\!\equiv_{\mathrm{Z}} (\mathit{free}(\mathrm{X})\ r) \iff \mathbb{v} \models\!\!\!\equiv_{\mathrm{Z} \setminus \mathrm{X}} r$

2.2.3 Semantics of Formulae

The semantics of formulae is defined with respect to a finite/infinite word over Σ, a valuation γ of the local variables and a set $\mathrm{Z} \subseteq \mathcal{V}$ of local variables. The role of Z is to support scoping, exactly as in tight satisfaction. The role of γ is to supply a current valuation of the local variables.

To connect to the semantics of RE$^{+\mathrm{V}}$s which uses extended words to those of formulas which use only initial valuation of local variables, we make use of the notion of *enhancement*. An extended word $\mathbb{w}$ enhances w with respect to γ, denoted $\mathbb{w} \sqsupset \langle w, \gamma\rangle$, if $\mathbb{w}|_\sigma = w$, $\mathbb{w}$ is good, and γ is the starting pre-value, i.e. $\mathbb{w}^0|_\gamma = \gamma$. The semantic of formulas using RE$^{+\mathrm{V}}$s involves quantification over the enhanced words. The quantification follows that of PSL without local variables.

Definition 4 (Satisfaction). *The notation* $\langle w, \gamma\rangle \models_{\mathrm{Z}} \varphi$ *means that the word* w *satisfies* φ *with respect to controlled variables* $\mathrm{Z} \subseteq \mathcal{V}$ *and current valuation of variables* γ.

- $\langle w, \gamma\rangle \models_{\mathrm{Z}} r! \iff \exists \mathbb{w} \sqsupset \langle w, \gamma\rangle$ *and* $j < |w|$ *such that* $\mathbb{w}^{0..j} \models\!\!\!\equiv_{\mathrm{Z}} r$
- $\langle w, \gamma\rangle \models_{\mathrm{Z}} \neg\varphi \iff \langle w, \gamma\rangle \not\models_{\mathrm{Z}} \varphi$
- $\langle w, \gamma\rangle \models_{\mathrm{Z}} \varphi \wedge \psi \iff \langle w, \gamma\rangle \models_{\mathrm{Z}} \varphi$ *and* $\langle w, \gamma\rangle \models_{\mathrm{Z}} \psi$
- $\langle w, \gamma\rangle \models_{\mathrm{Z}}$ *next* $\varphi \iff |w| > 1$ *and* $\langle w^{1..}, \gamma\rangle \models_{\mathrm{Z}} \varphi$
- $\langle w, \gamma\rangle \models_{\mathrm{Z}} \varphi$ *until* $\psi \iff \exists i < |w|$, $\langle w^{i..}, \gamma\rangle \models_{\mathrm{Z}} \psi$ *and* $\forall j < i$, $\langle w^{j..}, \gamma\rangle \models_{\mathrm{Z}} \varphi$
- $\langle w, \gamma\rangle \models_{\mathrm{Z}} r \mapsto \varphi \iff \forall \mathbb{w} \sqsupset \langle w, \gamma\rangle$ *and* $j < |w|$ *such that* $\mathbb{w}^{0..j} \models\!\!\!\equiv_{\mathrm{Z}} r$ *it holds that* $\langle w^{j+1..}, \mathbb{w}^{j+1}|_\gamma\rangle \models_{\mathrm{Z}} \varphi$
- $\langle w, \gamma\rangle \models_{\mathrm{Z}} (\mathit{new}(\mathrm{X})\ \varphi) \iff \langle w, \gamma\rangle \models_{\mathrm{Z} \cup \mathrm{X}} \varphi$
- $\langle w, \gamma\rangle \models_{\mathrm{Z}} (\mathit{free}(\mathrm{X})\ \varphi) \iff \langle w, \gamma\rangle \models_{\mathrm{Z} \setminus \mathrm{X}} \varphi$

Definition 5 (The Verification Problem). *Let* $\mathcal{M}$ *be a set of words over* Σ *and* γ_0 *an initial context of local variables. Let* φ *be a PSL*$^{+\mathrm{V}}$ *property and* Z *a set of local variables. We say that* $\mathcal{M}$ *satisfies* φ *with respect to* γ_0 *and* Z *if for every word* $w \in \mathcal{M}$, *we have that* $\langle w, \gamma_0\rangle \models_{\mathrm{Z}} \varphi$. *The* verification problem *is to check whether* $\mathcal{M}$ *satisfies* φ *with respect to* γ_0 *and* Z.

3 A Practical Subset of PSL$^{+\mathrm{V}}$

We define the following subset of PSL formulas with local variables, for which we will show that the complexity of the verification problem does not increase, i.e. remains in PSPACE, despite the presence of local variables.

Definition 6 (The Practical Subset, PSL^{pract}). *Let $b \in \mathcal{B}$ be a Boolean expression. Let* X *be a sequence of local variables and* E *a sequence of expressions of the same length as* X. *The grammar below defines the formulae φ that compose the practical subset, denoted PSL^{pract}:*

$$r ::= \; b \mid r \cdot r \mid r \cup r \mid r^+$$

$$R ::= b \mid (b, \mathrm{X} := \mathrm{E}) \mid R \cdot R \mid R \cup R \mid R^+ \mid (\mathit{new}(\mathrm{X})\; R) \mid (\mathit{free}(\mathrm{X})\; R)$$

$$\psi ::= r! \mid \neg r! \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathit{next}\,\psi \mid \psi \;\mathit{until}\; \psi \mid \psi \;\mathit{releases}\; \psi \mid r \mapsto \psi$$

$$\varphi ::= \neg R! \mid \psi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathit{next}\,\varphi \mid \varphi \;\mathit{until}\; \psi \mid \psi \;\mathit{releases}\; \varphi \mid R \mapsto \psi \mid (\mathit{new}(\mathrm{X})\; \varphi) \mid (\mathit{free}(\mathrm{X})\; \varphi)$$

To prove our main claim, we first need to enhance the notion of alternating automaton, with local variables. We start with a few words about (standard) alternating automata. A deterministic automaton has a single run on a give word. A *non-deterministic* automaton may have several runs on a given word. The automaton accepts a word if one of the runs is accepting (i.e. meets the acceptance condition). The dual of a non-deterministic automaton is a *universal automaton*. A universal automaton may also have several runs on a given word, but for it to accept the word all runs should be accepting. An alternating automaton combines existential and universal transitions. If Q is the set of states, a deterministic automaton maps a state and a letter to a state $q \in Q$. A non-deterministic automaton maps those to a disjunctive formula on Q e.g. $q_1 \vee q_5$. A universal automaton maps those to a conjunctive formula e.g. $q_2 \wedge q_4 \wedge q_5$. Finally, an alternating automaton maps those to a monotone formula on Q, e.g. $(q_1 \wedge (q_2 \vee q_7))$.[2] This would mean that either the runs from both q_1 and q_2 are accepting or the runs from both q_1 and q_7 are accepting.

A *local variable enhanced alternating automaton* maps a state and an extended letter to a pair whose first component is a monotone formula over Q as in a standard alternating automaton, and whose second component is sequence of local variables assignments. The intuition is that when this transition takes place, the local variables should be updated as indicated by the given assignments. For instance, if $\rho(q_1, \mathsf{a}) = \langle (q_1 \wedge (q_2 \vee q_7), (x_1 := 2, x_2 := x_1 + 2, x_1 := 3) \rangle$ then in addition to the above 3 is assigned to x_1, and 4 to x_2.

Definition 7. A Local Variable Enhanced Alternating Automaton *over finite/infinite words defined with respect to atomic propositions $\mathcal{P}$, local variables $\mathcal{V}$ is a tuple $\mathcal{A} = \langle \mathrm{Z}, \Lambda, Q, I, \rho, F, A \rangle$, where*

- Z is the set of local variables under control,
- Λ is the extended alphabet as defined in Section 2.2,
- Q is a finite and non-empty set of states,
- $I \in \mathcal{B}^+(Q)$ describes the initial configuration of active states,
- $\rho : Q \times \Lambda \to \mathcal{B}^+(Q) \times \mathcal{U}$ gives the transition relation,

[2] Given a set Q we use $\mathcal{B}^+(Q)$ to denote the set of monotone Boolean expressions over Q.

- $F \subseteq Q$ is a subset of states for accepting *finite* words.
- $A \subseteq Q$ is a subset of states for *accepting infinite* words (the Büchi condition).

A run of a non-deterministic automaton $\mathcal{N}$ over a word is a sequence of states. A run on a finite word is accepting if it ends in a state in F. A run on an infinite word is accepting if a state in A is visited infinitely often. A word $\mathtt{w}$ is accepted provided there is an accepting run on it. A run of an alternating automaton $\mathcal{A}$ on $\mathtt{w} \in \Lambda$ is a labelled tree $\langle T, \tau \rangle$ where T is a prefix closed subset of $\mathbb{N}^*$ and τ a mapping from a node $t \in T$ to a state in Q. We use $|t|$ to denote the depth of node t in the tree T. The root of a tree is ϵ. The depth of the root $|\epsilon|$ is 0. We use $succ(t)$ to denote the successors of node t, namely the nodes $t' \in T$ such that $t' = t \cdot n$ for some $n \in \mathbb{N}$. By abuse of notations, if $succ(t) = \{t_1, \ldots, t_m\}$ we say that $\{\tau(t_1), \ldots, \tau(t_m)\}$ are the successors of state $\tau(t)$.

A labelled tree $\langle T, \tau \rangle$ is a *consistent run* of $\mathcal{A}$ on a good extended word $\mathtt{w} \in \Lambda$ and initial context $\gamma_0 \in \Gamma$ if the following three conditions hold: (a) the initial states satisfy the automaton initial condition and the initial word pre-value is γ_0, formally, $\tau(succ(\epsilon))$ satisfies I and $\mathtt{w}^0|_\gamma = \gamma_0$ and (b) the successors of a state t satisfy the transition relation with respect to $\tau(t)$ and $\mathtt{w}^{|t|}$ and (c) the post-valuation of local variables agrees with the transition's update. Formally, let $t \neq \epsilon$ be a node in T such that $\tau(t) = q$ and $\rho(q, \mathtt{w}^{|t|-1}) = \langle b, \mathrm{U} \rangle$, then for the tree to be a consistent run we should have $\tau(succ(t))$ satisfies b and $\mathtt{w}^{|t|-1}|_{\gamma'} \overset{Z}{\sim} [\![\mathrm{U}]\!](\mathtt{w}^{|t|-1}|_{\sigma\gamma})$.

The universal branches (conjunction) in the transition relation are reflected in the successor relation of $\langle T, \tau \rangle$. The existential branches (disjunction) are reflected in the fact that one may have multiple runs of $\mathcal{A}$ on $\mathtt{w}$. After a universal branch, all active successors, namely $succ(t)$, further propagate the activeness to their successors. A run $\langle T, \tau \rangle$ is *accepting* provided all paths satisfy the acceptance condition (i.e. terminate in a state in F if the path is finite, and visit infinitely often a state in A if the path is infinite). A word $\mathtt{w}$ is accepted by $\mathcal{A}$ if there exists an accepting run on it. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$ is the set of words accepted by $\mathcal{A}$.

The method to prove that a property φ is valid in a model $\mathcal{M}$ by the automata-theoretic approach is to build an alternating automaton $\mathcal{A}_{\neg\varphi}$ that accepts the same language as $\neg\varphi$, namely $\mathcal{L}(\neg\varphi)$, then build the product (a.k.a. parallel composition) $\mathcal{M} \parallel \mathcal{A}_{\neg\varphi}$ and prove the language of the product automaton is empty.

Intuitively, the increase in complexity of the verification problem from PSPACE to EXPSPACE in the presence of local variables stems from the fact that the automaton needs to track all possible values a local variable may obtain. If a formula has k local variables over domain $\mathcal{D}$, the automaton should, in general, track all possible $\mathcal{D}^k$ values they may obtain. The practical subset tries to detect formulas for which the worst case will not be met. For instance, consider formula φ_1 from Example 1, intuitively the automaton should track just one value of the local variable x per each run — the value of $data\_in$ when $start$ rises.

The situation with φ_2 and φ_3 from Example 2 is more intricate. Here x may change unboundedly many times throughout the evaluation of the formula. But that in itself is not a problem – different automaton states may "record" different value of x. The problem is that transactions may overlap, i.e. $start$ may hold at cycles $k_1 < k_2$ where end does not hold in any cycle $k_1 \leq j \leq k_2$. Thus the automaton should track several possibilities for x on the same position of the same word! Think, for example, on the word $\langle start\rangle\langle put\rangle\langle\rangle\langle put\rangle\langle put\rangle\langle start\rangle\langle put\rangle\langle\rangle\ldots$[3]. The value of x on the 8-th letter should be 4 for the first transaction but 1 for the second. However, there is an important difference between φ_2 and φ_3. To refute φ_2 it suffices to track the change on just one transaction whereas to refute φ_3 one needs to track all transaction.

Tracking different values for same position of the word means that a run tree of the automaton may have a node with two descendants that disagree on the value of the local variables. We call an automaton in which such a situation cannot occur *conflict free*. The automaton for φ_2 will be conflict-free but the one for φ_3 will not be.

Definition 8 (Conflict Free Automata). *We say that a run* $\langle T,\tau\rangle$ *on* $\mathbb{w}$ *is* conflict-free *if there exists no pair of distinct nodes* $t_1, t_2 \in T$ *having a common ancestor such that* $\rho(\tau(t_1),\mathbb{w}^i) = \langle b_1, \mathrm{X}_1 := \mathrm{E}_1\rangle$ *and* $\rho(\tau(t_2),\mathbb{w}^i) = \langle b_2, \mathrm{X}_2 := \mathrm{E}_2\rangle$ *where exists a local variable* $z \in \mathrm{X}_1 \cap \mathrm{X}_2$*. We say that* $\mathcal{A}$ *is* conflict-free *if every run of it is conflict-free.*[4]

Lemma 2 (Automata Construction for $\mathbf{RE^{+V}}$). *Let r be an $RE^{+\mathrm{V}}$, and $\mathrm{Z} \subseteq \mathcal{V}$. There exists a conflict-free non-deterministic finite automaton $\mathcal{N}(r)$ with $O(|r|)$ states that accept exactly the set of words $\mathbb{v}$ such that $\mathbb{v} \models_{\mathrm{Z}} r$.*

Lemma 3 (Automata Construction for the Practical Subset). *Let φ be a formula in PSL^{pract}, $\mathrm{Z} \subseteq \mathcal{V}$ a set of controlled variables, and $\gamma \in \Gamma$ an initial value for local variables. There exists a conflict-free local variable enhanced alternating word automaton $\mathcal{A}_{\gamma,\mathrm{Z}}(\neg\varphi)$ with number of states $O(|\varphi|)$ and of size $|\mathcal{A}_{\gamma,\mathrm{Z}}(\neg\varphi)| = O(|\varphi|)$ that accept exactly the set of words w such that $\langle w,\gamma\rangle \models_{\mathrm{Z}} \neg\varphi$*

Proof Sketch. In Section 4, we provide a construction for properties whose negation is in PSL^{pract}. Since the subset does not support the negation operator we need to propagate it down to $\mathrm{RE}^{+\mathrm{V}}$s using the duality between operators, as provided at the end of Definition 2. Note that for *next* , *free*() and *new*() negation just propagates as is (that is, they are dual to themselves). Hence we end up with a property φ in the following set

$$r ::= b \mid r \cdot r \mid r \cup r \mid r^+$$

$$R ::= b \mid (b, \mathrm{X} := \mathrm{E}) \mid R \cdot R \mid R \cup R \mid R^+ \mid (\textit{new}(\mathrm{X})\ R) \mid (\textit{free}(\mathrm{X})\ R)$$

$$\psi ::= \neg r! \mid r! \mid \psi \vee \psi \mid \psi \wedge \psi \mid \textit{next}\ \psi \mid \psi\ \textit{until}\ \psi \mid \psi\ \textit{releases}\ \psi \mid r \mapsto \psi$$

$$\varphi ::= R! \mid \psi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \textit{next}\ \varphi \mid \varphi\ \textit{releases}\ \psi \mid \psi\ \textit{until}\ \varphi \mid R \mapsto \psi \mid (\textit{new}(\mathrm{X})\ \varphi) \mid (\textit{free}(\mathrm{X})\ \varphi)$$

Following the construction in Section 4 we see that the only universal branches introduced are the ones dealing with $\wedge$, *until* and *releases* . For $\wedge$ we create distinct copies of the local variables, and for *until* and *releases* , since ψ does not contain assignments, the automaton is conflict-free. □

[3] We assume here each $\langle\cdot\rangle$ corresponds to a letter, and the content of $\langle\cdot\rangle$ specifies the propositions holding in that letter.

[4] We say that a node t' is an ancestor of node t if there exists a sequence $t'' \in \mathbb{N}^*$ such that $t = t' \cdot t''$. Note that in particular t is an ancestor of itself.

We now show that given a conflict-free alternating automaton we can extract the part dealing with local variables to a *satellite* — a machine determining the value of local variables by observing the states of the automaton and the values of atomic propositions and local variables.

Definition 9 (A Satellite). *Let $\mathcal{B}$ be an alternating automaton with state set Q, an initial value of local variables $\gamma_0 \in \Gamma$ and a set of local variables $\mathrm{Z} \subseteq \mathcal{V}$. A* satellite *$\mathcal{S}$ with respect to $\mathcal{B}$, Z and γ_0 is a set of pairs of the form (g, U) whose first element g is a Boolean expression over $\mathcal{P}$, $\mathcal{V}$ and Q; and its second element $\mathrm{U} \in \mathcal{U}$ is a local variable update.*

Intuitively, g is a condition (*guard*) upon which the assignments in U take place. Formally, let $\mathcal{B}$ be an alternating automaton over extended words, $\mathcal{S}$ a satellite as above, and $\mathbb{v}$ be an extended good word. We say that a run tree $\langle T, \tau \rangle$ of $\mathcal{B}$ on $\mathbb{v}$ is consistent with $\mathcal{S}$ if the following two conditions hold. First, the initial context agrees with γ_0, that is $\mathbb{v}^0|_\gamma = \gamma_0$. Second, for every node $t \in T$ with $|t| = i$ and $\tau(t) = q$, and every local variable $z \in \mathrm{Z}$ if there is no pair $(g, \mathrm{X} := \mathrm{E}) \in \mathcal{S}$ such that $z \in \mathrm{X}$ and $g(\mathbb{v}^i|_{\sigma\gamma}, q) = \mathrm{T}$ then $z(\mathbb{v}^i|_{\gamma'}) = z(\mathbb{v}^i|_\gamma)$, otherwise (a) there exists no other pair $(g', \mathrm{X}' := \mathrm{E}') \in \mathcal{S}$ with $z \in \mathrm{E}'$ and $g'(\mathbb{v}^i|_{\sigma\gamma}, q) = \mathrm{T}$ and (b) the word is consistent with respect to the update of z, that is, $z(\mathbb{v}^i|_{\gamma'}) = z([\![\mathrm{X} := \mathrm{E}]\!](\mathbb{v}^i|_{\sigma\gamma}))$.[5]

Lemma 4 (Satellite Extraction). *Given a conflict-free local variable-enhanced alternating word automaton $\mathcal{A}_{\gamma,\mathrm{Z}}$, there exist an alternating Büchi automaton $\mathcal{B}_\mathrm{Z}$ and a satellite $\mathcal{S}_{\gamma,\mathrm{Z}}$ with respect to $\mathcal{B}_\mathrm{Z}$ such that $|\mathcal{B}_\mathrm{Z}| + |\mathcal{S}_{\gamma,\mathrm{Z}}| = O(|\mathcal{A}_{\gamma,\mathrm{Z}}|)$ and $\mathcal{L}(\mathcal{A}_{\gamma,\mathrm{Z}}) = \mathcal{L}(\mathcal{B}_\mathrm{Z} \parallel \mathcal{S}_{\gamma,\mathrm{Z}})$.*

Proofs of Lemmas 2 and 4 are given in section 4. The proof of our main theorem follows from the above three lemmas.

Proof Sketch of Theorem 1. Given a model $\mathcal{M}$ of words (typically described as a hardware design), and a formula φ of PSL^{pract}, we check whether $\mathcal{M}$ satisfies φ as follows. Assume $\gamma \in \Gamma$ is an initial value for the local variables and Z are the variables assigned in φ. By Lemma 3 there exists a conflict-free local variable-enhanced alternating word automaton $\mathcal{A}_{\gamma,\mathrm{Z}}$ of size $O(|\varphi|)$ such that $\mathcal{L}(\mathcal{A}_{\gamma,\mathrm{Z}}) = \mathcal{L}(\neg\varphi)$.

By Lemma 4 we can construct a traditional alternating Büchi automaton $\mathcal{B}_\mathrm{Z}$ with $O(|\varphi|)$ states, and a satellite $\mathcal{S}_{\gamma,\mathrm{Z}}$ over $\mathcal{B}_\mathrm{Z}$ of size $O(|\varphi| \times |\mathrm{Z}|)$ such that $\mathcal{L}(\mathcal{S}_{\gamma,\mathrm{Z}} \parallel \mathcal{B}_\mathrm{Z}) = \mathcal{L}(\neg\varphi)$.

Since $\mathcal{B}_\mathrm{Z}$ is a traditional automaton we can apply the Miyano-Hayashi construction [13] to it to get a non-deterministic automaton $\mathcal{N}$ whose number of states is exponential in $|\varphi|$, and is representable in $O(|\varphi|)$ space, and accepts the same language as $\mathcal{B}_\mathrm{Z}$. Thus, $\mathcal{L}(\mathcal{S}_{\gamma,\emptyset} \parallel \mathcal{N}) = \mathcal{L}(\neg\varphi)$.

It follows then that $\mathcal{L}(\mathcal{M} \parallel \mathcal{S}_{\gamma,\emptyset} \parallel \mathcal{N}) = \mathcal{M} \cap \mathcal{L}(\neg\varphi)$. Thus the complexity of checking whether $\mathcal{M} \models \varphi$ reduces to the non-emptiness of non-deterministic automata

[5] For those familiar with Verilog — note that this makes the implementation of a satellite possible using a Verilog code that uses an always block with if statement for every g such that $(g, \mathrm{U}) \in \mathcal{S}$ and the body of the if statement is the sequential updates U that take place one after one at the same clock.

with number of states polynomial in $|\mathcal{M}|$ and exponential $|\varphi|$. Non-emptiness of non-deterministic automata is NLOGSPACE-Complete with respect to their size. Thus our problem can be solved in space polynomial in $|\varphi|$ and logarithmic in $|\mathcal{M}|$.

$\square$

We can check the satisfiability of PSL^{pract} formulas, similarly — by checking the emptiness of $\mathcal{S}_{\gamma,\emptyset} \parallel \mathcal{N}$. Thus, similar arguments show that the satisfiability of PSL^{pract} can as well be checked in space polynomial in $|\varphi|$.

4 Automata Construction and Proofs

4.1 Proof of Construction for RE$^{+\mathrm{V}}$s

Proof of Lemma 2. We provide a construction of $\mathcal{N}(r)$ and then claim its correctness. For the inductive steps we assume $\mathcal{N}(r_i) = \langle Z_i, \Lambda, Q_i, I_i, \rho_i, F_i, \emptyset\rangle$ satisfy the lemma.

- $\mathcal{N}(b) = \langle \emptyset, \Lambda, \{q_0, q_{\mathrm{ACC}}\}, q_0, \rho, q_{\mathrm{ACC}}, \emptyset\rangle$, where
$$\rho(q_0, \mathtt{a}) = \begin{cases} \langle q_{\mathrm{ACC}}, \varepsilon\rangle & \text{if } b(\mathtt{a}) = \mathit{true} \\ \langle \mathit{false}, \varepsilon\rangle & \text{otherwise} \end{cases}$$

- $\mathcal{N}(b, \mathrm{X} := \mathrm{E}) = \langle \mathrm{X}, \Lambda, \{q_0, q_{\mathrm{ACC}}\}, q_0, \rho, q_{\mathrm{ACC}}, \emptyset\rangle$, where
$$\rho(q_0, \mathtt{a}) = \begin{cases} \langle q_{\mathrm{ACC}}, \mathrm{X} := \mathrm{E}\rangle & \text{if } b(\mathtt{a}) = \mathit{true} \\ \langle \mathit{false}, \varepsilon\rangle & \text{otherwise} \end{cases}$$

- $\mathcal{N}(\lambda) = \langle \emptyset, \Lambda, \{q_{\mathrm{ACC}}\}, q_{\mathrm{ACC}}, \emptyset, q_{\mathrm{ACC}}, \emptyset\rangle$

- $\mathcal{N}(r_1 \cup r_2) = \langle \mathrm{Z}_1 \cup \mathrm{Z}_2, \Lambda, Q_1 \cup Q_2, I_1 \vee I_2, \rho'_1 \cup \rho'_2, F_1 \cup F_2, \emptyset\rangle$ where
$$\rho'_1(q, \mathtt{a}) = \begin{cases} \rho_1(q, \mathtt{a}) & \text{if } \mathtt{a} \text{ } \mathit{preserves} \text{ } \mathrm{Z}_2 \setminus \mathrm{Z}_1 \\ \langle \mathit{false}, \varepsilon\rangle & \text{otherwise} \end{cases}$$
$$\rho'_2(q, \mathtt{a}) = \begin{cases} \rho_2(q, \mathtt{a}) & \text{if } \mathtt{a} \text{ } \mathit{preserves} \text{ } \mathrm{Z}_1 \setminus \mathrm{Z}_2 \\ \langle \mathit{false}, \varepsilon\rangle & \text{otherwise} \end{cases}$$

- $\mathcal{N}(r_1 \cdot r_2) = \langle \mathrm{Z}_1 \cup \mathrm{Z}_2, \Lambda, Q_1 \cup Q_2, I_1, \rho'_1 \cup \rho'_2, F_2, \emptyset\rangle$, where
$$\rho'_2(q, \mathtt{a}) = \begin{cases} \rho_2(q, \mathtt{a}) & \text{if } \mathtt{a} \text{ } \mathit{preserves} \text{ } \mathrm{Z}_1 \setminus \mathrm{Z}_2 \\ \langle \mathit{false}, \varepsilon\rangle & \text{otherwise} \end{cases}$$
$$\rho'_1(q, \mathtt{a}) = \begin{cases} \langle S_1 \vee I_2, \mathrm{U}\rangle & \text{if } \rho_1(q, \mathtt{a}) = \langle S_1, \mathrm{U}\rangle \text{ and } F_1 \cap S_1 \neq \emptyset \text{ and} \\ & \mathtt{a} \text{ } \mathit{preserves} \text{ } \mathrm{Z}_2 \setminus \mathrm{Z}_1 \\ \langle S_1, \mathrm{U}\rangle & \text{if } \rho_1(q, \mathtt{a}) = \langle S_1, \mathrm{U}\rangle \text{ and } F_1 \cap S_1 = \emptyset \text{ and} \\ & \mathtt{a} \text{ } \mathit{preserves} \text{ } \mathrm{Z}_2 \setminus \mathrm{Z}_1 \\ \langle \mathit{false}, \varepsilon\rangle & \text{otherwise} \end{cases}$$

- $\mathcal{N}(r_1^+) = \langle \mathrm{Z}_1, \Lambda, Q_1, I_1, \rho'_1, F_1, \emptyset\rangle$, where
$$\rho'_1(q, \mathtt{a}) = \begin{cases} \langle S_1 \vee I_1, \mathrm{U}\rangle & \text{if } \rho_1(q, \mathtt{a}) = \langle S_1, \mathrm{U}\rangle \text{ and } F_1 \cap S_1 \neq \emptyset \\ \langle S_1, \mathrm{U}\rangle & \text{if } \rho_1(q, \mathtt{a}) = \langle S_1, \mathrm{U}\rangle \text{ and } F_1 \cap S_1 = \emptyset \end{cases}$$

- $\mathcal{N}(\mathit{new}(\mathrm{X})\ r_1) = \langle \mathrm{Z}_1 \cup \mathrm{X}, \Lambda, Q_1, I_1, \rho_1, F_1, \emptyset\rangle$

- $\mathcal{N}(\mathit{free}(\mathrm{X})\ r_1) = \langle \mathrm{Z}_1 \setminus \mathrm{X}, \Lambda, Q_1, I_1, \rho_1, F_1, \emptyset\rangle$

The conflict-freeness of $\mathcal{N}$ follows trivially from the absence of universal branches in the transitions. For language acceptance, the cases λ, b, $r_1 \cdot r_2$, r_1^+ and $r_1 \cup r_2$ follow the traditional construction [3] with the desired adjustment for the set of local variables. In the cases $(\textit{new}(\mathrm{X})\ r_1)$ and $(\textit{free}(\mathrm{X})\ r_1)$, there are changes to Z, but no changes in the transition and the acceptance condition.

4.2 Proof of Construction for Properties of the Practical Subset

Proof of Lemma 3. For the inductive steps we assume for the operands $\varphi_1, \varphi_2, \psi_1, \psi_2, r_1, R_1$, the automata $\mathcal{A}_\gamma(\phi_i) = \langle Z_i, \Lambda, Q_i, I_i, \rho_i, F_i, A_i\rangle$ satisfy the inductive hypothesis (with $\phi \in \{r, R, \varphi, \psi\}$ and $i \in \{1,2\}$). Let U be a sequence of assignments to local variables in Z. Let $\mathrm{X} \subseteq \mathrm{Z}$ and let X' be a set of fresh variables of same size as $|\mathrm{X}|$. We use $U[\mathrm{X}{\leftarrow}\mathrm{X}']$ to denote the sequence of assignments $X' := X \cdot U'$ where U' is obtained from U by replacing all occurrences of variables in X with the respective variable in X'. For a tuple $\langle b, U\rangle$, we use $\langle b, U\rangle[\mathrm{X}{\leftarrow}\mathrm{X}']$ to denote $\langle b, U[\mathrm{X}{\leftarrow}\mathrm{X}']\rangle$.

- $\mathcal{A}(R_1!) = \langle \mathrm{Z}_1, \Lambda, Q_1 \cup \{q_{\text{ACC}}\}, I_1, \rho_1', F_1, \{q_{\text{ACC}}\}\rangle$, where
$$\rho_1'(q, \mathtt{a}) = \begin{cases} \langle q_{\text{ACC}}, \varepsilon\rangle & \text{if } q = q_{\text{ACC}} \\ \langle S_1, \mathrm{U}\rangle & \text{else if } \rho_1(q,\mathtt{a}) = \langle S_1, \mathrm{U}\rangle \text{ and } F_1 \cap S_1 = \emptyset \\ \langle S_1 \vee q_{\text{ACC}}, \mathrm{U}\rangle & \text{else if } \rho_1(q,\mathtt{a}) = \langle S_1, \mathrm{U}\rangle \text{ and } F_1 \cap S_1 \neq \emptyset \end{cases}$$

- $\mathcal{A}(\neg r_1!) = \langle \mathrm{Z}_1, \Lambda, Q_1 \cup \{q_{\text{REJ}}\}, \overline{I_1}, \rho_1', \{q_{\text{REJ}}\}, \{q_{\text{REJ}}\}\rangle$
Let b be a monotone Boolean expression. We use $\overline{b}$ to denote the Boolean expression obtained from b by replacing $\vee$ with $\wedge$ and vice versa. Let Q be a finite set $\{q_1, \ldots, q_n\}$. We use $\overline{Q}$ to denote $q_1 \wedge \cdots \wedge q_n$.
$$\rho_1'(q, \mathtt{a}) = \begin{cases} \langle q_{\text{REJ}}, \varepsilon\rangle & \text{if } q = q_{\text{REJ}} \\ \langle q_{\text{REJ}}, \varepsilon\rangle & \text{else if } \rho_1(q,\mathtt{a}) = \langle \textit{false}, \varepsilon\rangle \\ \langle \overline{S_1}, \mathrm{U}\rangle & \text{otherwise if } \rho_1(q,\mathtt{a}) = \langle S_1, \mathrm{U}\rangle \end{cases}$$

- $\mathcal{A}(\varphi_1 \vee \varphi_2) = \langle \mathrm{Z}_1 \cup \mathrm{Z}_2, \Lambda, Q_1 \cup Q_2, I_1 \vee I_2, \rho', F_1 \cup F_2, A_1 \cup A_2\rangle$ where
$$\rho'(q, \mathtt{a}) = \begin{cases} \rho_1(q,\mathtt{a}) & \text{if } q \in Q_1 \text{ and } \mathtt{a} \textit{ preserves } \mathrm{Z}_2 \setminus \mathrm{Z}_1 \\ \rho_2(q,\mathtt{a}) & \text{if } q \in Q_2 \text{ and } \mathtt{a} \textit{ preserves } \mathrm{Z}_1 \setminus \mathrm{Z}_2 \end{cases}$$

- $\mathcal{A}(\varphi_1 \wedge \varphi_2) = \langle \mathrm{Z}', \Lambda, Q_1 \cup Q_2, I_1 \wedge I_2, \rho', F_1 \cup F_2, A_1 \cup A_2\rangle$
Let $\mathrm{X} = \mathrm{Z}_1 \cap \mathrm{Z}_2$, $\mathrm{Y}_1 = \mathrm{Z}_1 \setminus \mathrm{Z}_2$, $\mathrm{Y}_2 = \mathrm{Z}_2 \setminus \mathrm{Z}_1$. Let $\mathrm{X}_1, \mathrm{X}_2$ be fresh vectors of variables of same size as X.
– $\mathrm{Z}' = \mathrm{X}_1 \cup \mathrm{Y}_1 \cup \mathrm{X}_2 \cup \mathrm{Y}_2$.
– $\rho'(q, \mathtt{a}) = \begin{cases} \rho_1(q,\mathtt{a})[\mathrm{X}{\leftarrow}\mathrm{X}_1] & \text{if } q \in Q_1 \text{ and } \mathtt{a} \textit{ preserves } \mathrm{Z}_2 \setminus \mathrm{Z}_1 \\ \rho_2(q,\mathtt{a})[\mathrm{X}{\leftarrow}\mathrm{X}_2] & \text{if } q \in Q_2 \text{ and } \mathtt{a} \textit{ preserves } \mathrm{Z}_1 \setminus \mathrm{Z}_2 \end{cases}$

- $\mathcal{A}(\textit{next}\ \varphi_1) = \langle \mathrm{Z}_1, \Lambda, Q_1 \cup \{q_0\}, q_0, \rho_1 \cup \rho', F_1, A_1\rangle$, where
$\rho'(q_0, \mathtt{a}) = \langle I_1, \varepsilon\rangle$ for every $\mathtt{a} \in \Lambda$

- $\mathcal{A}(\psi_1\ \textit{until}\ \varphi_2) = \langle \mathrm{Z}', \Lambda, Q_1 \cup Q_2 \cup \{q_0\}, I_2 \vee (I_1 \wedge q_0), \rho, F_1 \cup F_2, A_1 \cup A_2\rangle$
Let $\mathrm{X} = \mathrm{Z}_1 \cap \mathrm{Z}_2$, $\mathrm{Y}_1 = \mathrm{Z}_1 \setminus \mathrm{Z}_2$, $\mathrm{Y}_2 = \mathrm{Z}_2 \setminus \mathrm{Z}_1$. Let $\mathrm{X}_1, \mathrm{X}_2$ be fresh vectors of variables of same size as X.

– $Z' = X_1 \cup Y_1 \cup X_2 \cup Y_2$.

$$- \rho(q,\mathsf{a}) = \begin{cases} \langle I_2 \vee (I_1 \wedge q_0), \varepsilon\rangle & \text{if } q \text{ is } q_0 \\ \rho_1(q,\mathsf{a})[X{\leftarrow}X_1] & \text{if } q \in Q_1 \text{ and } \mathsf{a} \textit{ preserves } Z_2 \setminus Z_1 \\ \rho_2(q,\mathsf{a})[X{\leftarrow}X_2] & \text{if } q \in Q_2 \text{ and } \mathsf{a} \textit{ preserves } Z_1 \setminus Z_2 \end{cases}$$

– $\mathcal{A}(\varphi_1 \textit{ releases } \psi_2) = \langle Z', \Lambda, Q_1 \cup Q_2 \cup \{q_0\}, I_2 \wedge (I_1 \vee q_0), \rho, F_1 \cup F_2, A_1 \cup A_2\rangle$
Let $X = Z_1 \cap Z_2$, $Y_1 = Z_1 \setminus Z_2$, $Y_2 = Z_2 \setminus Z_1$. Let X_1, X_2 be fresh vectors of variables of same size as X.
– $Z' = X_1 \cup Y_1 \cup X_2 \cup Y_2$.

$$- \rho(q,\mathsf{a}) = \begin{cases} \langle I_2 \wedge (I_1 \vee q_0), \varepsilon\rangle & \text{if } q \text{ is } q_0 \\ \rho_1(q,\mathsf{a})[X{\leftarrow}X_1] & \text{if } q \in Q_1 \text{ and } \mathsf{a} \textit{ preserves } Z_2 \setminus Z_1 \\ \rho_2(q,\mathsf{a})[X{\leftarrow}X_2] & \text{if } q \in Q_2 \text{ and } \mathsf{a} \textit{ preserves } Z_1 \setminus Z_2 \end{cases}$$

– $\mathcal{A}(r_1 \mathrel{\circ\!\!\Rightarrow\!\!\diamond} \varphi_2) = \langle Z_1 \cup Z_2, \Lambda, Q_1 \cup Q_2, I_1, \rho', F_2, A_2\rangle$, where

$$\rho'(q,\mathsf{a}) = \begin{cases} \langle b \vee I_2, U\rangle & \text{if } q \in Q_1 \text{ and } \mathsf{a} \textit{ preserves } Z_2 \setminus Z_1 \text{ and} \\ & \rho_1(q,\mathsf{a}) = \langle b, U\rangle \text{ and } F_1 \cap \textit{supp}(b) \neq \emptyset \\ \rho_1(q,\mathsf{a}) & \text{otherwise if } q \in Q_1 \text{ and } \mathsf{a} \textit{ preserves } Z_2 \setminus Z_1 \\ \rho_2(q,\mathsf{a}) & \text{if } q \in Q_2 \text{ and } \mathsf{a} \textit{ preserves } Z_1 \setminus Z_2 \end{cases}$$

– $\mathcal{A}(\textit{new}(X)\ \varphi_1) = \langle Z_1 \cup X, \Lambda, Q_1, I_1, \rho_1, F_1, A_1\rangle$

– $\mathcal{A}(\textit{free}(X)\ \varphi_1) = \langle Z_1 \setminus X, \Lambda, Q_1, I_1, \rho_1, F_1, A_1\rangle$

We now have to show that $\mathcal{A}(\varphi)$ as constructed satisfies the premises of the lemma. It is easy to see that $\mathcal{A}(\varphi)$ is polynomial in $|\varphi|$. The proof that it accepts the same language as φ follows the one for traditional constructions [5,3]. It is left to show that it is conflict-free.

To see that it is conflict free we observe that a universal quantification is introduced in the following cases $\neg r!$, $\wedge$, *until* , *releases* . For the case of $\neg r!$ the syntax restrictions on r guarantees that r has no assignments, thus there are no updates in the generated automaton. For the case of $\wedge$ the construction introduces fresh variables for the operands, thus no conflict is met. For the constructions of the *until* and *releases* operators, the introduction of fresh variables guarantees no conflict between the updates done by the two operands. It is left to see that no conflict can arise by the several instances corresponding to the operand φ where updates of local variables are allowed. To see this note that in any run tree there will be only one layer where the successors of state q_0 (introduced in the construction for the cases of *until* and *releases*) are the initial states of $\mathcal{A}(\varphi)$. Hence, updates to the variables in φ will be done at most once in every tree. For instance, for *releases* we have that q_0 transits to $I_2 \wedge (I_1 \vee q_0)$. That is, q_0 transits to either both q_0 and I_2 or to both I_2 and I_1. So once it has chosen to transit to both I_2 and I_1 there will be no further such transitions to I_1. Since local variables updates of φ_1 will occur at the sub-tree emerging from I_1 we get that there will be only one such updates per tree.

Thus in any run of $\mathcal{A}$ if there is a local variable update to a node t at depth i then any node t' of same depth which is not a descendent of t may not have a local variable assignment to the same variable. This is since t and t' emerge from different branches of a universal quantification, and as mentioned above, in all such cases variables of the different branches where renamed. Hence the automaton is conflict-free. $\square$

Example 3. *Back to Example 2, formula φ_2 is in PSL^{pract} while φ_3 is not. The figure below shows the construction for both. See that in $\mathcal{A}(\neg\varphi_3)$ we have assignments taking place in a loop involving universal quantification.*

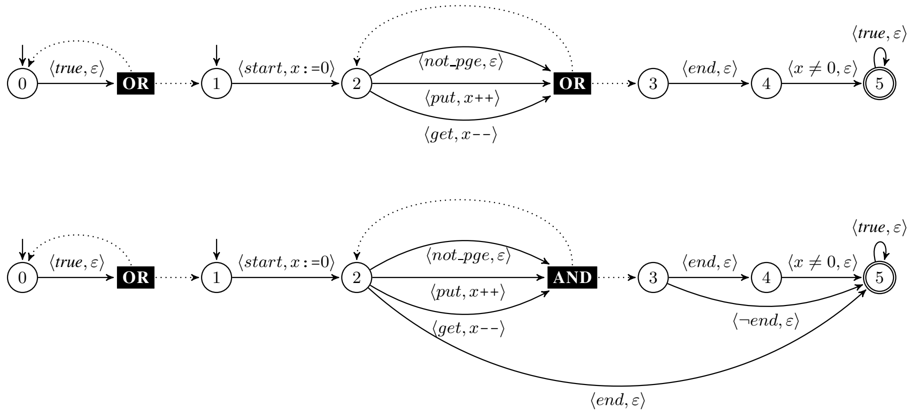

Fig. 1. Local variable enhanced alternating automata for $\neg\varphi_2$ (upper) and $\neg\varphi_3$ (lower) from Example 2

Proof of Lemma 4. Let $\mathcal{A}_{\gamma_0} = \langle \mathrm{Z}, \Lambda, Q, I, \rho, F, A\rangle$ be a conflict-free alternating Büchi automaton. We define a traditional alternating Büchi automaton $\mathcal{B}_{\mathrm{Z}}$ and a satellite $\mathcal{S}_{\gamma_0,\mathrm{Z}}$ that satisfy the theorem as follows. The automaton $\mathcal{B}_{\mathrm{Z}}$ is a traditional alternating automaton in the sense that his transitions maps to $\mathcal{B}^+(Q)$ but there are no updates of local variables associated with transitions. The automaton, does however, read the current value of local variables in its letters. It simply ignores the updates annotations of $\mathcal{A}_{\gamma_0}$. Formally, $\mathcal{B}_{\mathrm{Z}} = \langle \Sigma \times \Gamma, Q, I, \rho', F, A\rangle$ where

$$\rho'(q, \mathbb{a}) = b \quad \text{iff} \quad \rho(q, \mathbb{a}) = \langle b, \mathrm{U}\rangle \text{ for some } \mathrm{U} \in \mathcal{U}$$

The satellite $\mathcal{S}_{\gamma_0,\mathrm{Z}}$ is in-charge of making the correct updates of local variables. The definition of $\mathcal{S}_{\gamma_0,\mathrm{Z}}$ follows the transition of $\mathcal{A}_{\gamma_0}$ as follows. For every transition $\rho(q, \mathbb{a}) = \langle b, \mathrm{U}\rangle$ a pair (g, U) is added to $\mathcal{S}_{\gamma_0,\mathrm{Z}}$ where g is defined as $q \wedge b$.

To prove that $\mathcal{B}_{\mathrm{Z}}$ and $\mathcal{S}_{\gamma_0,\mathrm{Z}}$ satisfy the theorem, we show that every run of $\mathcal{A}_{\gamma_0}$ is a consistent run of $\mathcal{B}_{\mathrm{Z}}$ with respect to $\mathcal{S}_{\gamma_0,\mathrm{Z}}$. Let $\langle T, \tau\rangle$ be a run tree of $\mathcal{A}_{\gamma_0}$ on an extended word $\mathbb{w}$ with initial context γ_0. We know that the initial pre-value of $\mathbb{w}$ satisfy γ_0 and that the successors of the root satisfy the initial condition I since these requirements are the same for runs of $\mathcal{A}_{\gamma_0}$ and $\mathcal{B}_{\mathrm{Z}}$.

To see that the second condition for a consistent run is met we note that since $\mathcal{A}_{\gamma_0,\mathrm{Z}}$ is context free we are guaranteed that on every run, if there is an update to a local variable

z at node t and there is another node t' of the same depth as t then there are no updates to z from t' and any of its descendants. Thus, it cannot be that there are two pairs (g_1, U_1) and (g_2, U_2) such that both hold on a state q and they update a common variable z. Therefore the update of a local variable z will be done by the unique guard that holds at that node, if such a guard exits, and will remain the same otherwise. Therefore the second condition of a consistent run holds and $\langle T, \tau \rangle$ is a consistent run of $\mathcal{B}_\mathrm{z}$ with respect to $\mathcal{S}_{\gamma_0,\mathrm{z}}$.

The reasoning for the reversed direction is similar. $\square$

References

1. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: The forSpec temporal logic: A new temporal property-specification language. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 296–311. Springer, Heidelberg (2002)
2. Beer, I., Ben-David, S., Eisner, C., Fisman, D., Gringauze, A., Rodeh, Y.: The temporal logic sugar. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 363–367. Springer, Heidelberg (2001)
3. Ben-David, S., Bloem, R., Fisman, D., Griesmayer, A., Pill, I., Ruah, S.: Automata construction algorithm optimized for PSL. Technical Report Delivery 3.2/4, PROSYD (July 2005)
4. Ben-David, S., Fisman, D., Ruah, S.: Embedding finite automata within regular expressions. Theor. Comput. Sci. 404(3), 202–218 (2008)
5. Bustan, D., Fisman, D., Havlicek, J.: Automata construction for PSL. Technical report, Weizmann Institute of Science (May 2005)
6. Bustan, D., Havlicek, J.: Some complexity results for systemVerilog assertions. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 205–218. Springer, Heidelberg (2006)
7. Havlicek, J., Korchemny, D., Cerny, E., Dudani, S.: Havlicek Korchemny Cerny and Dudani. Springer (2009)
8. Eisner, C., Fisman, D.: Eisner and Fisman. Springer (2006)
9. Eisner, C., Fisman, D.: Augmenting a regular expression-based temporal logic with local variables. In: Cimatti, A., Jones, R.B. (eds.) FMCAD, pp. 1–8. IEEE (2008)
10. IEEE Standard for Property Specification Language (PSL). IEEE Std 1850$^{\mathrm{TM}}$-2010 (2010)
11. IEEE Standard for SystemVerilog ? Unified Hardware Design, Specification, and Verification Language. IEEE Std 1800$^{\mathrm{TM}}$-2009 (2009)
12. Martensson, J.: US patent US8225249: Static formal verification of a circuit design using properties defined with local variables. Jasper Design Automation, Inc. (June 2008)
13. Miyano, S., Hayashi, T.: Alternating finite automata on omega-words. Theor. Comput. Sci. 32, 321–330 (1984)
14. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57. IEEE Computer Society (1977)