

An Operational Semantics for Weak PSL

Koen Claessen* and Johan Mårtensson†
koen@cs.chalmers.se, johan@safelogic.se

*Chalmers University of Technology †Safelogic AB and Göteborg University

Abstract. Extending linear temporal logic by adding regular expressions increases its expressiveness. However, as for example, problems in recent versions of Accellera’s Property Specification Language (PSL) as well as in OpenVera’s ForSpec and other property languages show, it is a non-trivial task to give a formal denotational semantics with desirable properties to the resulting logic. In this paper, we argue that specifying an *operational semantics* may be helpful in guiding this work, and as a bonus leads to an implementation of the logic for free. We give a concrete operational semantics for *Weak PSL*, which is the safety property subset of PSL. We also propose a denotational semantics which we show to be equivalent to the operational one. This semantics is inspired by a new denotational semantics proposed in recent related work.

1 Introduction

Accellera and PSL Accellera [1] is an organization supported by major actors in the electronic design industry with the objective to promote the use of standards in this industry. In spring 2003, a standard property specification language for hardware designs was agreed upon, PSL 1.0 [2]. The standard defines the syntax and semantics of PSL formally. A new version of the language, PSL 1.1 [3] was finalized in spring 2004.

The logical core of PSL consists of standard Linear Temporal Logic (LTL) constructs augmented with regular expressions and aborts. Thus, PSL contains the notion of *formula*, which is an entity of extended LTL that can be satisfied by an infinite sequence of letters, and the notion of *expression*, which is a regular expression that can only be satisfied by a finite sequence of letters. A letter simply defines the values of all variables at one point in time.

Expressions can be converted into formulas, by using, for example, the *weak embedding* of an expression r , written $\{r\}$ in PSL 1.1 syntax. In both PSL 1.0 and 1.1, a sequence s of letters makes $\{r\}$ true, if there is a finite prefix of s that satisfies r or all finite prefixes of s can be extended to satisfy r . However, the nature of this extension is different in the two PSL versions, which accounts for differences mentioned below.

Further, the semantics of PSL has to cope with the fact that properties are supposed to be used both in *static verification* — checking that a property holds solely by analyzing the design — and in *dynamic verification* — checking that

a property holds for a concrete and finite trace of the design. To deal with dynamic verification, satisfiability is extended to finite (*truncated*) sequences even for formulas [7, 8].

Anomalies The semantics of both PSL 1.0 and 1.1 are given by means of denotational semantics. One problem with this approach is that for some constructs it may be far from obvious what definition should be chosen. Making a seemingly intuitively correct decision can lead to undesirable properties of the resulting logic.

For example in PSL 1.0 the formula¹ $\{[*]; a\}$, which is the weak embedding of an expression that is satisfied by any sequence ending with the atom a , is satisfied by any sequence that makes a always false. However, the formula $\{[*]; F\}$ (where we have simply replaced a by the false constant F) is not satisfied by any sequence. Thus, $\{[*]; F\}$ is not satisfied even if it is aborted at the first time instance. This issue is discussed in for example [7, 4].

In response to this, Accellera has developed a different semantical paradigm that is used in the current iteration, PSL 1.1. In this semantics, the notion of model is changed by introducing a new semantical concept; a special letter \top that can satisfy any one-letter expression, regardless if it is contradictory or not. Unfortunately, PSL 1.1 suffers from a similar anomaly: F and $\{a \& \& \{a; a\}\}$ (a so-called *structural contradiction*) are equivalent in an intuitive sense (neither can be satisfied on actual runs of a system), but are not interchangeable in formulas. Thus $\{[*]; F\}$ is satisfied if it is aborted at the first time instance whereas $\{[*]; \{a \& \& \{a; a\}\}\}$ is not. This peculiarity is not specific to PSL; it is for example also present in ForSpec’s reset semantics.

There is work underway within Accellera to deal with this anomaly either by discouraging the use of particular ”degenerate formulas” (e.g. those containing structural contradictions), thus excluding the ones that are not well-behaved in this respect, or by extending the model concept further to include models on which structural contradictions are satisfied.

Operational semantics The company Safelogic develops tools for static and dynamic verification of PSL properties of hardware designs. When implementing our tools, we faced two problems. Firstly, particular simplification rules we expected to hold in the logic actually did not hold and could thus not be used. Secondly, in the denotational semantics definitions there is no indication as how to implement checkers and verifiers for PSL properties.

Our approach was to define a *structural operational semantics* for the subset of PSL we considered. This subset is precisely the subset of PSL in which safety properties can be expressed. The operational semantics is a small-step letter-by-letter semantics with judgments of the form $\phi \xrightarrow{\ell} \psi$. The intention is that in order to check if a sequence s starting with the letter ℓ satisfies ϕ , we simply check that the tail of s (without ℓ) satisfies ψ , and so on.

¹ In PSL 1.0 weak embedding does occur but not in the form of independent formulas. This difference to PSL 1.1 is irrelevant to the present point, so we ignore it for the sake of simplicity of presentation.

There are two advantages of this approach. (1) The operational semantics can directly be used for implementing dynamic verification of properties, and also forms the basis of the implementation of our static verification engine. (2) When specifying a structural operational semantics, there are far less choices to be made than in a denotational semantics, so there is less room for mistakes.

In a recent related work a new denotational semantics for LTL with regular expressions on truncated words has been investigated [8]. This semantics can be extended to a full PSL semantics that fixes the anomalies in the semantics of PSL 1.0 and 1.1 [10]. We have shown that our operational semantics is sound and complete on the weak fragment of PSL with respect to such an extension of this denotational semantics to full PSL. Hopefully, the next iteration of PSL will adopt a denotational semantics with this property!

This paper The rest of this paper is organized as follows. In Section 2, we specify a safety property subset of PSL. In Section 3, we define a structural operational semantics for this language. In Section 4, we present a denotational semantics for our subset of PSL, corresponding to [8, 10]. In Section 5, we show lemmas relating the two semantics, and state soundness and completeness of our operational semantics. Section 6 concludes.

2 Weak Property Language

In this section, we identify a subset of PSL, called Weak Property Language (WPL). This subset can only be used to write safety properties. As is done in the PSL Language Reference Manuals [2, 3], we start by assuming a non-empty set P of *atomic propositions*, and a set of boolean expressions B over P . We assume two designated boolean expressions **true**, **false** belonging to B .

We start by defining regular expressions, which then provide the base case in the definition of full WPL.

Definition 1 (RE). *If $b \in B$, the language of regular expressions (REs) r has the following grammar:*

$$r ::= \perp \mid \varepsilon \mid b \mid r_1; r_2 \mid r_1|r_2 \mid r_1 \&\& r_2 \mid r^*.$$

The expression \perp denotes the expression with the empty language, ε is the expression that only contains the empty word (see Section 4.4 for an explanation of why those expressions not present in [2, 3] were introduced), $r_1; r_2$ stands for sequential composition between r_1 and r_2 , $r_1|r_2$ stands for choice, $r_1 \&\& r_2$ stands for intersection, r^* is the Kleene star.

Now we are in a position to define full WPL.

Definition 2 (WPL). *If r, r_1 and r_2 are REs, and b a boolean expression, the language of WPL formulas ϕ and ψ has the following grammar:*

$$\phi, \psi ::= \{r\} \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid X\phi \mid \phi_1 W \phi_2 \mid r \Rightarrow \phi \mid \phi \mathbf{abort} b.$$

The formula $\{r\}$ is the weak embedding of the expression r , $\phi_1 \wedge \phi_2$ is formula conjunction, $\phi_1 \vee \phi_2$ is formula disjunction, $X\phi$ is the next operator, $\phi_1 W \phi_2$ is the weak until operator, $r \Rightarrow \phi$ is suffix implication, and $\phi \mathbf{abort} b$ is the abort operator. The logical negation operator \neg only appears at the boolean level, and not at the formula level, because that would enable the creation of non-safety formulas.

Suffix implication $r \Rightarrow \phi$ is satisfied by a word if whenever r accepts a prefix of the word, the formula ϕ holds on the rest of that word. The formula $\phi \mathbf{abort} b$ is satisfied by a word if ϕ is not made false by that word before b holds. A formal definition of these constructs is given in Section 3 by means of an operational semantics, and in Section 4 by means of a denotational semantics.

For reasons of simplicity we have omitted the treatment of the overlapping operators $:$ and \vdash . The semantic definitions for those operators (See [6, 10]) are very similar to those for $;$ and \Rightarrow .

3 A Structural Operational Semantics for WPL

It is customary to give semantics to temporal logics using sequences of states, where each state contains information about the truth-values of all atoms. In the PSL formal semantics [2, 3], a state is called a *letter*, written ℓ , and sequences of states are called *words*. The set of all letters is written Σ . The details of letters are not important here. However, we assume that there is a satisfaction relation \Vdash between states Σ and boolean expressions B , such that for all letters $\ell \in \Sigma$, $\ell \Vdash \mathbf{true}$ and $\ell \not\Vdash \mathbf{false}$.

In this section, we present a structural operational semantics for WPL. Our operational semantics is inspired by Brzozowski's derivatives of regular expressions [5]. We use judgments of the form

$$\phi \xrightarrow{\ell} \psi.$$

The intuition behind such a judgment is that in order to check if a word starting with the letter ℓ satisfies ϕ , one can just as well check that ψ is satisfied by the word without the first letter. So, for a finite word $w = (\ell_0, \ell_2, \dots, \ell_n)$, we can check if w satisfies ϕ by finding $\phi_0 \dots \phi_n$ such that

$$\phi \xrightarrow{\ell_0} \phi_0 \xrightarrow{\ell_1} \phi_1 \dots \xrightarrow{\ell_n} \phi_n,$$

and check that none of ϕ_i is false. We will be more formal about this later.

3.1 Letters and Words

We define the following preliminaries. A *word* is a finite or infinite sequence of letters from Σ . We use ϵ to denote the empty word. We use juxtaposition to denote concatenation, i.e. if $w = (\ell_0, \dots, \ell_n)$ and $v = (\ell'_0, \dots, \ell'_n)$ then $wv = (\ell_0, \dots, \ell_n, \ell'_0, \dots, \ell'_n)$. If w is infinite then wv is w . We observe that

concatenation is associative, i.e. $w(vu) = (wv)u$ for all w, v and u , and ϵ is the identity, i.e. $\epsilon w = w\epsilon = w$ for all w . We will use ℓ both for denoting the letter ℓ and the word consisting of the single letter ℓ .

Word indexing is defined as follows. If $i < |w|$ then w^i is the $i + 1^{\text{st}}$ letter of w and $w^{i\dots}$ is the suffix of w starting at i . If $i \geq |w|$ then $w^{i\dots} = \epsilon$. If $k \leq j < |w|$, then $w^{k\dots j}$ means (w^k, \dots, w^j) . If $j < k < |w|$, then $w^{k\dots j}$ is ϵ .

We use $v \leq w$ and ' v is a *prefix* of w ' to say that there is a u such that $vu = w$ and $v < w$ to say that $v \leq w$ and $v \neq w$.

3.2 Operational Rules for RE

We start by giving rules for the basic REs. A boolean expression b accepts a letter ℓ only if ℓ satisfies b . In that case, the remaining expression is the empty word. Falsity and the empty word accept no letters.

$$\text{(BOOL)} \quad b \xrightarrow{\ell} \begin{cases} \epsilon & \text{if } \ell \Vdash b \\ \perp & \text{otherwise} \end{cases}$$

$$\text{(BOT)} \quad \perp \xrightarrow{\ell} \perp$$

$$\text{(EMPTY)} \quad \epsilon \xrightarrow{\ell} \perp$$

For sequential composition $r_1; r_2$, we use two rules. If r_1 cannot accept the empty word, r_2 will not be touched. However, if r_1 can accept the empty word, we need to consider the case that r_2 accepts ℓ as well. Thus, we need a function **em** that calculates if a given RE can accept the empty word or not.

Definition 3. We define (inductively) for REs:

$$\begin{aligned} \text{em}(\perp) &= \text{false} \\ \text{em}(\epsilon) &= \text{true} \\ \text{em}(b) &= \text{false} \\ \text{em}(r_1; r_2) &= \text{em}(r_1) \text{ and } \text{em}(r_2) \\ \text{em}(r_1|r_2) &= \text{em}(r_1) \text{ or } \text{em}(r_2) \\ \text{em}(r_1 \&\&r_2) &= \text{em}(r_1) \text{ and } \text{em}(r_2) \\ \text{em}(r^*) &= \text{true} \end{aligned}$$

The rules for sequential composition then look as follows.

$$\text{(SEQ1)} \quad \frac{r_1 \xrightarrow{\ell} r'_1}{r_1; r_2 \xrightarrow{\ell} r'_1; r_2} \quad \text{not } \text{em}(r_1)$$

$$\text{(SEQ2)} \quad \frac{r_1 \xrightarrow{\ell} r'_1 \quad r_2 \xrightarrow{\ell} r'_2}{r_1; r_2 \xrightarrow{\ell} (r'_1; r_2)|r'_2} \quad \text{em}(r_1)$$

The rules for choice and intersection simply apply the rules to both of the operands.

$$\begin{aligned}
 (\text{REOR}) \quad & \frac{r_1 \xrightarrow{\ell} r'_1 \quad r_2 \xrightarrow{\ell} r'_2}{r_1 | r_2 \xrightarrow{\ell} r'_1 | r'_2} \\
 (\text{REAND}) \quad & \frac{r_1 \xrightarrow{\ell} r'_1 \quad r_2 \xrightarrow{\ell} r'_2}{r_1 \&\&r_2 \xrightarrow{\ell} r'_1 \&\&r'_2}
 \end{aligned}$$

And finally, for a Kleene star r^* to accept a letter, the expression r must be able to accept the letter.

$$(\text{STAR}) \quad \frac{r \xrightarrow{\ell} r'}{r^* \xrightarrow{\ell} r'; r^*}$$

3.3 Operational Rules for WPL

Weak embedding of expressions simply parses the ℓ through the expression until what is left of the expression can accept the empty word.

$$\begin{aligned}
 (\text{RE1}) \quad & \frac{r \xrightarrow{\ell} r'}{\{r\} \xrightarrow{\ell} \{r'\}} \quad \text{not em}(r) \\
 (\text{RE2}) \quad & \{r\} \xrightarrow{\ell} \{\mathbf{true}^*\} \quad \text{em}(r)
 \end{aligned}$$

Here, we use the formula $\{\mathbf{true}^*\}$ since it accepts every word.

Formula disjunction and conjunction are identical to their regular expression counterparts.

$$\begin{aligned}
 (\text{WPLOR}) \quad & \frac{\phi_1 \xrightarrow{\ell} \phi'_1 \quad \phi_2 \xrightarrow{\ell} \phi'_2}{\phi_1 \vee \phi_2 \xrightarrow{\ell} \phi'_1 \vee \phi'_2} \\
 (\text{WPLAND}) \quad & \frac{\phi_1 \xrightarrow{\ell} \phi'_1 \quad \phi_2 \xrightarrow{\ell} \phi'_2}{\phi_1 \wedge \phi_2 \xrightarrow{\ell} \phi'_1 \wedge \phi'_2}
 \end{aligned}$$

The rule for next simply drops the next operator.

$$(\text{NEXT}) \quad X\phi \xrightarrow{\ell} \phi$$

The rule for weak until is directly derived from the fact that weak until is a solution of the following equation: $\phi_1 W \phi_2 = \phi_2 \vee (\phi_1 \wedge X(\phi_1 W \phi_2))$.

$$(\text{UNTIL}) \quad \frac{\phi_1 \xrightarrow{\ell} \phi'_1 \quad \phi_2 \xrightarrow{\ell} \phi'_2}{\phi_1 W \phi_2 \xrightarrow{\ell} \phi'_2 \vee (\phi'_1 \wedge (\phi_1 W \phi_2))}$$

For suffix implication, there are two rules: one that triggers the formula ϕ to be true when r accepts the empty word, and one that does not trigger ϕ . One can see these rules as dual to the rules for sequential composition.

$$(S11) \quad \frac{r \xrightarrow{\ell} r'}{r \models \phi \xrightarrow{\ell} r' \models \phi} \quad \text{not } \text{em}(r)$$

$$(S12) \quad \frac{r \xrightarrow{\ell} r' \quad \phi \xrightarrow{\ell} \phi'}{r \models \phi \xrightarrow{\ell} (r' \models \phi) \wedge \phi'} \quad \text{em}(r)$$

Finally, an abort checks its formula until the boolean becomes true. So, when checking $\phi \mathbf{abort} b$ with respect to ℓ we first check whether ϕ is already contradicted. If not and b is satisfied by ℓ then we abort the checking of $\phi \mathbf{abort} b$ by accepting. If b is not satisfied by ℓ or ϕ is contradicted already then we do not abort.

$$(ABORT1) \quad \frac{\phi \xrightarrow{\ell} \phi'}{\phi \mathbf{abort} b \xrightarrow{\ell} \phi' \mathbf{abort} b} \quad \text{not } \text{ok}(\phi) \text{ or } \ell \not\models b$$

$$(ABORT2) \quad \phi \mathbf{abort} b \xrightarrow{\ell} \{\mathbf{true}^*\} \quad \text{ok}(\phi) \text{ and } \ell \models b$$

In order to calculate if a regular expression or formula has been contradicted, we use the function ok , which is to be defined in the next section.

This concludes the operational rules for WPL. As standard, we define \rightarrow to be the least relation satisfying the above operational rules. However, it is easy to see that \rightarrow actually is a total function from formulas and letters to formulas.

3.4 Not Yet Contradicted

Here, we define the function ok that calculates whether a given regular expression or formula has been contradicted yet, w.r.t. the sequence of letters that has already been visited. An expression or formula is said to be ok when it has not yet been contradicted in this sense. For basic regular expressions, only \perp is not ok . For composite expressions and formulas, this information is simply propagated.

Definition 4. *We define (inductively) for REs and WPLs:*

$$\begin{array}{ll} \text{ok}(\perp) = \mathbf{false} & \text{ok}(\{r\}) = \text{ok}(r) \\ \text{ok}(\varepsilon) = \mathbf{true} & \text{ok}(\phi_1 \wedge \phi_2) = \text{ok}(\phi_1) \mathbf{and} \text{ok}(\phi_2) \\ \text{ok}(b) = \mathbf{true} & \text{ok}(\phi_1 \vee \phi_2) = \text{ok}(\phi_1) \mathbf{or} \text{ok}(\phi_2) \\ \text{ok}(r_1; r_2) = \text{ok}(r_1) & \text{ok}(X\phi) = \mathbf{true} \\ \text{ok}(r_1|r_2) = \text{ok}(r_1) \mathbf{or} \text{ok}(r_2) & \text{ok}(\phi_1 W \phi_2) = \text{ok}(\phi_1) \mathbf{or} \text{ok}(\phi_2) \\ \text{ok}(r_1 \&\&r_2) = \text{ok}(r_1) \mathbf{and} \text{ok}(r_2) & \text{ok}(r \models \phi) = \mathbf{true} \\ \text{ok}(r^*) = \mathbf{true} & \text{ok}(\phi \mathbf{abort} b) = \text{ok}(\phi) \end{array}$$

Finally, we make the following observation, which is that any regular expression accepting the empty string is an ok expression.

Lemma 1 (Empty is OK). *For all REs r , $\text{em}(r) \Rightarrow \text{ok}(r)$.*

The function `ok` is used in the operational rules for `abort`, but also in the definition of the operational semantics.

3.5 The Operational Semantics

As we have seen in the informal explanation of the operational rules, we are interested in the result of applying the rules above iteratively to formulas with respect to words from the alphabet Σ . This is possible to do since the rules presented above are deterministic; given a formula ϕ and a letter ℓ , there is a unique formula ϕ' such that $\phi \xrightarrow{\ell} \phi'$. Thus, the relation $\xrightarrow{\ell}$ is a total function. Iteratively applying the operational rules on a formula ϕ over the letters of a word w is written $\phi\langle w \rangle$:

Definition 5 (After a Word). *For a RE or WPL p and a finite word w , we (inductively) define $p\langle w \rangle$ as follows:*

$$\begin{aligned} p\langle \epsilon \rangle &= p, \\ p\langle \ell w \rangle &= p'\langle w \rangle, \text{ where } p \xrightarrow{\ell} p'. \end{aligned}$$

Now we are ready to define what it means for a formula to be true according to the operational semantics, denoted by \vdash .

Definition 6 (The Operational Semantics). *For all WPLs and REs p , and all words w we define*

$$w \vdash p \Leftrightarrow \text{for all finite } v \text{ such that } v \leq w, \text{ok}(p\langle v \rangle).$$

Intuitively, this means that a word w makes a formula ϕ true if and only if iteratively applying the operational semantics on ϕ using w only produces ok formulas.

We observe the following useful lemma, which says that if an expression of formula is not ok, it will stay not ok even after applying it to a word.

Lemma 2 (Conservation of Misery). *For all WPLs and REs p , we have*

$$\neg \text{ok}(p) \Rightarrow \text{for all finite } u, \neg \text{ok}(p\langle u \rangle).$$

It immediately follows, that for finite words w , in order to decide if $w \vdash p$, it suffices to check the final result $p\langle w \rangle$.

Lemma 3. *For all WPLs and REs p , if w is finite*

$$w \vdash p \Leftrightarrow \text{ok}(p\langle w \rangle).$$

Since all functions involved in the above are computable, this gives us a simple procedure for checking if a finite word satisfies a formula according to the operational semantics.

Note that the operational semantics presented above does not have the anomaly described in the introduction. For example, we have, for all words w , that $w \vdash \{\mathbf{true}^*; (a\&\&(a; a))\}$. To see this, observe that, for any ℓ , $\{\mathbf{true}^*; (a\&\&(a; a))\} \xrightarrow{\ell} \{(\mathbf{true}^*; (a\&\&(a; a)))|r\}$, for some r . Thus, the **ok**-ness of the formula is not affected by accepting any finite word.

3.6 Properties of the Operational Semantics

We observe the following interesting properties of the operational semantics and iterated application of the operational rules. These lemmas are key steps in the correctness proofs for the completeness and soundness theorems in Sections 5.2 and 5.3. Apart from this, the details of this section are not important for the remainder of the paper.

We start with some observations related to applying an expression or formula to a word.

Lemma 4. *For all REs and WPLs p , all letters ℓ , and all finite words w and v , we have*

$$\begin{aligned} p\langle w \rangle \langle v \rangle &= p\langle wv \rangle, \\ p &\xrightarrow{\ell} p\langle \ell \rangle, \\ p\langle w \rangle &\xrightarrow{\ell} p\langle w\ell \rangle. \end{aligned}$$

The second observation we make is that applying a word preserves disjunctions and conjunctions.

Lemma 5 (Preservation of Disjuncts and Conjuncts). *For all finite words w , for WPLs ϕ and ψ , and for REs r_1 and r_2 ,*

$$\begin{aligned} (\phi \vee \psi)\langle w \rangle &= \phi\langle w \rangle \vee \psi\langle w \rangle \\ (\phi \wedge \psi)\langle w \rangle &= \phi\langle w \rangle \wedge \psi\langle w \rangle \\ (r_1|r_2)\langle w \rangle &= r_1\langle w \rangle | r_2\langle w \rangle \\ (r_1\&\&r_2)\langle w \rangle &= r_1\langle w \rangle \&\& r_2\langle w \rangle \end{aligned}$$

Finally, a direct consequence of the above lemma and Lemma 2 is that disjunction and conjunction are compositional w.r.t. the operational semantics.

Lemma 6 (Operational Compositionality). *For all finite words w , for WPLs ϕ and ψ , and for REs r_1 and r_2 ,*

$$\begin{aligned} w \vdash \phi \vee \psi &\Leftrightarrow w \vdash \phi \text{ or } w \vdash \psi \\ w \vdash \phi \wedge \psi &\Leftrightarrow w \vdash \phi \text{ and } w \vdash \psi \\ w \vdash r_1|r_2 &\Leftrightarrow w \vdash r_1 \text{ or } w \vdash r_2 \\ w \vdash r_1\&\&r_2 &\Leftrightarrow w \vdash r_1 \text{ and } w \vdash r_2 \end{aligned}$$

4 Denotational Semantics

Alternatively, we can define a denotational semantics for WPL. The following definitions are inspired by a related not yet published work [8]. In Section 4.4 we briefly describe the relation between this semantics and the official PSL 1.1 one.

4.1 Weak and Neutral Words

We have noted that for all finite w and all REs r , $w \vdash \mathbf{true}^*; r$. It doesn't matter whether r is satisfiable or not. We want the denotational semantics to mirror this. So our definition must provide a kind of partial matching where the word w is only required to match "the beginning" of the RE r , mirroring the way in which an RE is true according to the operational semantics if it is not yet contradicted when the word finishes.

To accomplish this we introduce in addition to the usual *neutral* words also *weak* words.

Let N denote the set of finite and infinite words over Σ , and $N^f \subset N$ the set of finite words over Σ . The elements of N are called *neutral words*. Let $W = \{u^- \mid u \in N^f\}$. We assume that W and N are disjoint and that the mapping $(-)^-$ is injective. Whenever the notation u^- is used, it is understood that $u \in N^f$. The elements of W are called *weak words*. Note in particular that $\epsilon^- \in W$.

Let $A = N \cup W$, and define concatenation in A as follows. For all $u, v \in N$, uv is equal to the concatenation in N , and if u is finite then $u(v^-) = (uv)^-$. For all $u, v \in A$, if u is infinite or $u \in W$ then $uv = u$. With this definition concatenation in A is associative and ϵ is the unique identity element. Define the length of an element w in N as the number of letters in w if w is finite and ω otherwise, and in A according to $|u^-| = |u|$ for all $u \in N^f$.

Word indexing in A is defined as follows. For $i < |w|$, $(w^-)^i = w^i$. We let $(w^-)^{i\dots} = (w^{i\dots})^-$. We also let $(w^-)^{k\dots j} = w^{k\dots j}$ for $j, k < |w|$.

4.2 Tight Satisfaction

We start by giving a definition of *tight satisfaction* \models . Tight satisfaction relates finite words from A to REs. A finite neutral word intuitively tightly satisfies a regular expression if the word completely matches the expression. A finite weak word intuitively tightly satisfies a regular expression if the process of matching the word (from left to right) does not contradict the expression.

Definition 7. Let r, r_1 and r_2 denote REs, b a boolean, and w, w_1, \dots, w_j words in A . We define inductively:

$$\begin{aligned}
w &\not\models \perp \\
w &\models b && \Leftrightarrow \text{either } w = \epsilon^-, \text{ or } |w| = 1 \text{ and } w^0 \Vdash b \\
w &\models r_1; r_2 && \Leftrightarrow \text{there are } w_1, w_2 \text{ such that } w = w_1 w_2 \text{ and } w_1 \models r_1 \text{ and } w_2 \models r_2 \\
w &\models r_1 | r_2 && \Leftrightarrow w \models r_1 \text{ or } w \models r_2 \\
w &\models r_1 \&\& r_2 && \Leftrightarrow w \models r_1 \text{ and } w \models r_2 \\
w &\models r^* && \Leftrightarrow \text{either } w = \epsilon, \text{ or} \\
&&& \text{there exists } w_1, w_2, \dots, w_j \text{ such that } w = w_1 w_2 \dots w_j \\
&&& \text{and for all } i \text{ such that } 1 \leq i \leq j, w_i \models r \\
w &\models \epsilon && \Leftrightarrow w \models \mathbf{false}^*
\end{aligned}$$

We note the following lemmas.

Lemma 7. For all REs r that do not syntactically contain \perp as a subexpression, we have that $\epsilon^- \models r$.

Lemma 8. For all REs r and $v, w \in A$ such that $v \leq w$, we have that $w \models r \Rightarrow v \epsilon^- \models r$.

4.3 Formula Satisfaction

We now define *formula satisfaction* \models . Formula satisfaction relates words from N to WPLs, and defines when a finite or infinite word satisfies a formula. A word intuitively satisfies a formula if the process of accepting the word does not contradict the formula.

Definition 8. Let ϕ and ψ denote WPLs, b a boolean, r an RE, and w, u, v etc. words in N . We define inductively:

$$\begin{aligned}
w &\models \{r\} && \Leftrightarrow \text{for all finite } v \leq w \text{ there is } u \leq v^-, \text{ such that } u \models r \\
w &\models \phi \wedge \psi && \Leftrightarrow w \models \phi \text{ and } w \models \psi \\
w &\models \phi \vee \psi && \Leftrightarrow w \models \phi \text{ or } w \models \psi \\
w &\models X\phi && \Leftrightarrow \text{if } |w| \geq 1 \text{ then } w^{1\dots} \models \phi \\
w &\models \phi W\psi && \Leftrightarrow \text{for all } k \text{ such that } w^{k\dots} \not\models \phi \text{ there is } j \leq k \text{ such that } w^{j\dots} \models \psi \\
w &\models r \rightrightarrows \phi && \Leftrightarrow \text{for all } u, v \text{ such that } uv = w \text{ if } u \models r \text{ then } v \models \phi \\
w &\models \phi \mathbf{abort} b && \Leftrightarrow \text{either } w \models \phi, \text{ or} \\
&&& \text{there is } k < |w| \text{ such that } w^k \Vdash b \text{ and } (w^{0\dots k-1}) \models \phi
\end{aligned}$$

The following lemma follows by structural induction from Lemma 7:

Lemma 9. For all WPLs ϕ that do not syntactically contain \perp as a subexpression, we have that $\epsilon \models \phi$.

Note that the denotational semantics presented above does not have the anomaly described in the introduction. For example, we have, for all words w , that $w \models \{\mathbf{true}^*; (a \&\& (a; a))\}$. To see this, take any finite prefix v of w . Then we have $v \models \mathbf{true}^*$ and $\epsilon^- \models a \&\& (a; a)$. It follows that $v \epsilon^- \models \mathbf{true}^*; (a \&\& (a; a))$.

4.4 Relations to PSL Semantics

An investigation into the relation between the semantics of Section 4 and the official PSL semantics is outside the scope of this article. We have however investigated this relation thoroughly. We have provided a refined criterion of degeneracy and a denotational relation of satisfaction for the entire unlocked PSL language on weak, neutral and strong words, and showed that ours is equivalent to the official PSL 1.1 one on formulas that are non-degenerate in this sense. For details of this see [10].

The semantics in Section 4 is a simplified version of the semantics of [10]. It is simplified in three ways:

1. It only covers the weak fragment of PSL.
2. It omits certain operators like $:$ and \mapsto , as explained in Section 2.
3. It omits a requirement of non-emptiness present in the case for $\{r\}$.

This last omission was made for the sake of simplicity of presentation. A consequence of the PSL 1.1. semantics is that if $|w| > 0$ then $w \not\models \{\varepsilon\}$. It is perhaps not impossible to define operational rules that mirror this requirement, but it seems to require more complicated rules than the ones we present.

We also introduced the RE symbols \perp and ε that are not present in [2, 3]. It was necessary to differentiate falsity that is already visited (\perp which should be false on ε^-) from falsity that is not already visited (**false** which should be true on ε^-) in the operational rules to get Lemma 10. It was also convenient for defining the operational rules in a succinct way to introduce a symbol ε that is only tightly satisfied by empty words.

5 Relations Between the Semantics

In this section we show that the operational semantics and denotational semantics are tightly coupled. The proofs are merely outlines; for more details see [6].

5.1 The Stepping Lemmas

We can show the following two basic lemmas, which confirms our intuition about the operational judgments $r \xrightarrow{\ell} r'$ and $\phi \xrightarrow{\ell} \phi'$.

Lemma 10 (RE Stepping). *For REs r and r' , if $r \xrightarrow{\ell} r'$, then for all $w \in A$*

$$\ell w \models r \Leftrightarrow w \models r'.$$

Using Lemma 10 we can prove the following.

Lemma 11 (WPL Stepping). *For WPLs ϕ and ψ , if $\phi \xrightarrow{\ell} \phi'$, then for all $w \in N$*

$$\ell w \models \phi \Leftrightarrow w \models \phi'.$$

The above lemmas are key steps in the completeness and soundness proofs.

5.2 Completeness

In order to show completeness of the operational semantics with respect to the denotational semantics, we first observe the following. If a words satisfies an RE, then that RE is ok.

Lemma 12 (Tight True is Ok). *For any RE r , and for $w \in A$*

$$w \models r \Rightarrow \text{ok}(r).$$

We can lift that observation to the level of WPLs.

Lemma 13 (True is Ok). *For any WPL ϕ , and for $w \in A$*

$$w \models \phi \Rightarrow \text{ok}(\phi).$$

Lemmas 11 and 4 can be used to show the following generalization of Lemma 11 which shows the tight relationship between the denotational semantics \models and applying a formula to a word.

Lemma 14. *For any WPL ϕ , and for $w \in N$*

$$w \models \phi \Leftrightarrow \text{for every finite } v \text{ such that } vu = w, u \models \phi(v).$$

Finally, we use Lemmas 14 and 13 to show completeness.

Theorem 1 (Completeness). *For any WPL ϕ , and for $w \in N$*

$$w \models \phi \Rightarrow w \vdash \phi.$$

5.3 Soundness

In order to show soundness of the operational semantics with respect to the denotational semantics, we use Lemmas 1, 5 to show a tight relationship between the denotational semantics \models and applying an expression to a word.

Lemma 15 (Empty is Tight). *For any RE r , and for all finite $v \in N$*

$$\text{em}(r(v)) \Leftrightarrow v \models r.$$

Lemmas 5 and 15 can be used to show that the operational semantics is compositional w.r.t. sequential composition.

Lemma 16 (Seq is Sound). *For any two REs r_1 and r_2 , and for all words w*

$$w \vdash r_1; r_2 \Rightarrow \\ \text{either } w \vdash r_1 \text{ or there are } v, u \text{ such that } vu = w \text{ and } v \models r_1 \text{ and } u \vdash r_2.$$

We use Lemma 16, 10, 7, 6 to show the following very strong lemma.

Lemma 17 (Tight Soundness). *For any RE r , and for all finite $w \in N$*

$$w \vdash r \Rightarrow w^- \models r.$$

Finally, we use Lemma 17, 11, 6 to show soundness.

Theorem 2 (Soundness). *For any WPL ϕ , and for $w \in N$*

$$w \vdash \phi \Rightarrow w \models \phi.$$

6 Conclusions and Future Work

We have defined an operational semantics for the weak fragment of PSL, and proved it sound and complete with respect to a new denotational semantics. This denotational semantics is a straightforward extension of earlier work [8], but it is not equivalent to either the official PSL 1.0 or PSL 1.1 semantics. Since our goal was to fix the anomalies in these semantics, it is not surprising that we end up with a different semantics.

However, there is work underway within Accellera to introduce the concept of a "degenerated formula", which is a formula that contains a structural contradiction (such as $a \&\&(a; a)$). The idea is that users of PSL are discouraged to use these degenerated formulas since they are the cause of the anomalies in the official semantics. We have created a formal definition of 'degenerate', and shown that (a variant of) our semantics agrees with the PSL 1.1 semantics for all non-degenerate formulas. The details of this however are beyond the scope of this paper (but see [10]).

Defining an operational semantics can help in guiding the work of defining a denotational one, but it also gives a direct way of implementing dynamic property checking. We also use the operational semantics as a basis of an algorithm for static property checking. It is far from clear how the denotational semantics could guide an implementation.

For space reasons, we have not included all weak PSL operators in our language WPL. Some of these operators, such as the clock operators, are actually expressible in terms of the operators presented here. Others, such as the overlapping versions of sequential composition (written $:$) and suffix implication (written \mapsto), are not expressible in terms of our operators. In any case, for reasons of clarity and efficiency, it is often a good idea to introduce dedicated operational rules for new operators. The actual operational rules for the overlapping operators $:$ and \mapsto are very similar to their non-overlapping counterparts. Dedicated clock rules require some more work; we believe that annotating the \rightarrow operator with extra clock information may be the right way to do this.

We are collaborating with Mike Gordon of Cambridge University in encoding our denotational semantics for full PSL [10] in the HOL higher order logic formalism with the purpose of proving relevant properties.² Earlier Gordon encoded the

² This ongoing work is documented at <http://cvs.sourceforge.net/viewcvs.py/hol/hol98/examples/PSL/experimental-semantics/>.

official PSL 1.0 and 1.1 semantics in HOL but experienced problems relating to the anomalies when trying to derive observers from those formal specifications [9]. We hope that these problems will be overcome using our semantics. Currently, we are also working on extending our operational semantics to also deal with non-safety properties. The next step would then be to relate that semantics to the strong satisfiability described in [10].

Acknowledgments

We thank the referees, Cindy Eisner and John Havlicek for useful comments on this paper. The second author would also like to thank the two last mentioned and Dana Fisman for good collaboration in developing the weak/strong words semantics for LTL with regular expressions [8], which did inspire the denotational semantics of this paper.

References

1. Accellera. www.accellera.org.
2. Accellera, 1370 Trancas Street, #163, Napa, CA 94558. *Property Specification Language, Reference Manual*, April 2003. www.accellera.org/pslv101.pdf.
3. Accellera, 1370 Trancas Street, #163, Napa, CA 94558. *Property Specification Language, Reference Manual*, 2004. www.eda.org/vfv/docs/PSL-v1.1.1.pdf.
4. R. Armoni, D. Bustan, O. Kupferman, and M. Vardi. Resets vs. aborts in linear temporal logic. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS03*, pages 65–80. LNCS 2619, April 2003.
5. J. A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, October 1964.
6. K. Claessen and J. Mårtensson. Relating operational and denotational semantics for WPL. Tech. Rep. Available on request from the authors.
7. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. van Campenhout. Reasoning with temporal logic on truncated paths. In *Computer Aided Verification, CAV03*, pages 27–39. LNCS 2725, July 2003.
8. C. Eisner, D. Fisman, J. Havlicek, and J. Mårtensson. Truncating regular expressions. Unpublished.
9. M. Gordon, J. Hurd, and K. Slind. Executing the formal semantics of the Accellera property specification language by mechanised theorem proving. In *Correct Hardware Design and Verification Methods, CHARME03*, pages 200–215. LNCS 2860, October 2003.
10. J. Mårtensson. A weak/strong words semantics for PSL. Tech. Rep. Available on request from the author.