From PSL to LTL: A Formal Validation in HOL

Thomas Tuerk and Klaus Schneider

Reactive Systems Group, Department of Computer Science,
University of Kaiserslautern, P.O. Box 3049,
67653 Kaiserslautern, Germany
{tuerk, klaus.schneider}@informatik.uni-kl.de
http://rsg.informatik.uni-kl.de

Abstract. Using the HOL theorem prover, we proved the correctness of a translation from a subset of Accellera's property specification language PSL to linear temporal logic LTL. Moreover, we extended the temporal logic hierarchy of LTL that distinguishes between safety, liveness, and more difficult properties to PSL. The combination of the translation from PSL to LTL with already available translations from LTL to corresponding classes of ω-automata yields an efficient translation from PSL to ω-automata. In particular, this translation generates liveness or safety automata for corresponding PSL fragments, which is important for several applications like bounded model checking.

1 Introduction

Model checking and equivalence checking are state of the art in hardware circuit design flows. Standardised languages like the hardware description languages VHDL [2,30] and Verilog [21] are widespread and allow the convenient exchange of modules, which can also be sold as IP blocks. However, specifications of temporal properties that are required for model checking cannot be easily described with these languages [23,6]. Hence, the research on model checking during the last two decades considered mainly temporal logics like LTL [22], CTL [9] and CTL* [10], and other formalisms like ω-automata [29], monadic second order logics, and the μ-calculus [25].

The discussed temporal logics differ dramatically in terms of syntax, semantics, expressiveness and the complexity of the related verification problem. For example, LTL model checking is PSPACE-complete, while CTL model checking can be done in polynomial time. Of course, this corresponds to the different expressive powers of these logics: It can be shown that temporal logics, ω-automata, monadic predicate logics, and the μ-calculus form a hierarchy in terms of expressiveness [25].

The incompatibility of temporal logics used for specification complicates the exchange of data between different tools; which is a situation similar to circuit design before the standardisation of hardware description languages. Hence, the increased industrial interest in verification naturally lead to standardisation efforts for specification logics [5,4]. Accellera's Property Specification Language (PSL) [1] is a result.

J. Hurd and T.F. Melham (Eds.): TPHOLs 2005, LNCS 3603, pp. 342–357, 2005.

However, the translation from (the linear time fragment of) PSL to equivalent ω-automata, as required for model-checking, turned out to be quite difficult, although many partial results of this translation already exist: It is well known how LTL can be translated to equivalent ω-automata [33,8,13,28,12]. There is a hierarchy of ω-automata [18,17,31,25] that distinguishes between safety, liveness and four other classes of increasing expressiveness. Recently, the related subsets of LTL of this hierarchy have been syntactically characterised [24,25] and linear-time translations have been presented [24,25] that translate the temporal logic classes to corresponding symbolic descriptions of ω-automata (these can be directly used for symbolic model checking).

In addition to the temporal operators of LTL, PSL also provides certain abort operators whose semantics turned out to be problematic: In [3], a logic RLTL was introduced that extends LTL by an abort operator in order to show the impact of different abort operators on the complexity of the translation and verification. As a result, it turned out that in the worst case, the previous version of PSL lead to a non-elementary blow-up in the translation to ω-automata. For this reason, the semantics of PSL's reset operator has been changed in version 1.1 following the ideas in [3]. Thus, it is not surprising that a significant subset of PSL can now be translated to RLTL. A further translation from RLTL to LTL has already been presented in [3].

The subtle differences of the reset operators in PSL version 1.01 and version 1.1 demonstrate that the semantics of complex temporal logics like PSL should not be underestimated. In fact, PSL is a complex language that includes many special cases. Therefore, we feel the need to formally verify all parts of the translation of PSL to ω-automata by a theorem prover like HOL. To this end, we implemented deep embeddings of RLTL, LTL and ω-automata in HOL[1]. Using the existing deep embedding[2] of PSL [14] and the existing LTL library [26], *we have formally proved the correctness of the entire translation from* PSL *to* ω*-automata via* RLTL *and* LTL. By a detailed examination of the translation from PSL to LTL, we could moreover extend the known temporal logic classes of LTL to corresponding classes of PSL. In particular, we will present in this paper a syntactic characterisation of subsets of PSL in the spirit of [24] that match with corresponding ω-automata classes for safety, liveness and other properties. Translations to safety or liveness automata are of particular interest for bounded model checking as shown in [27].

The paper is organised as follows: In the next section, we present the temporal logics PSL, RLTL and LTL in detail. Then, we briefly sketch the translations from PSL to RLTL and from RLTL to LTL. In Section 5, we then define classes of PSL that correspond with the temporal logic hierarchy [18,24,25] and hence, also with the ω-automaton hierarchy [17,31,25]. Finally, we draw some conclusions and show directions for future work.

[1] The HOL library is available at `http://rsg.informatik.uni-kl.de/tools`.

[2] Although some members of the Accellera Formal Property Language Technical Committee reviewed this embedding, we found a small, until then unknown bug in the embedding.

2 Basics

Temporal logics like LTL, RLTL and PSL use propositional logic to describe (static) properties of the current point of time. The semantics of dynamic, i. e., temporal properties is based on a sequence of points of time, a so-called *path*. Thus, we first define propositional logic and paths in this section.

Definition 1 (Propositional Logic). *Let $\mathcal{V}$ be a set of variables. Then, the* set of propositional formulas over $\mathcal{V}$ *(short* $\mathsf{prop}_{\mathcal{V}}$*) is recursively given as follows:*

- *each variable $v \in \mathcal{V}$ is a propositional formula*
- $\neg\varphi \in \mathsf{prop}_{\mathcal{V}}$*, if* $\varphi \in \mathsf{prop}_{\mathcal{V}}$
- $\varphi \wedge \psi \in \mathsf{prop}_{\mathcal{V}}$*, if* $\varphi, \psi \in \mathsf{prop}_{\mathcal{V}}$

An assignment *over $\mathcal{V}$ is a subset of $\mathcal{V}$. In our context, assignments are also called* states*. The set of all states over $\mathcal{V}$, which is the power set of $\mathcal{V}$, is denoted by $\mathcal{P}(\mathcal{V})$. The semantics of a propositional formula with respect to a state s is given by the relation $\models_{\mathsf{prop}}$ that is defined as follows:*

- $s \models_{\mathsf{prop}} v$ *iff* $v \in s$
- $s \models_{\mathsf{prop}} \neg\varphi$ *iff* $s \not\models_{\mathsf{prop}} \varphi$
- $s \models_{\mathsf{prop}} \varphi \wedge \psi$ *iff* $s \models_{\mathsf{prop}} \varphi$ *and* $s \models_{\mathsf{prop}} \psi$

If $s \models_{\mathsf{prop}} \varphi$ holds, then the assignment s is said to satisfy *the propositional formula φ.*

Moreover, we use the following abbreviations as syntactic sugar:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\mathsf{true} := v \vee \neg v$ for an arbitrary variable $v \in \mathcal{V}$
- $\mathsf{false} := \neg\mathsf{true}$

A finite word v over a set Σ of length $|v| = n+1$ is a function $v : \{0, \ldots n\} \rightarrow \Sigma$. An infinite word v over Σ is a function $v : \mathbb{N} \rightarrow \Sigma$ and its length is denoted by $|v| = \infty$. The set Σ is called the *alphabet* and the elements of Σ are called *letters*. The finite word of length 0 is called the *empty word* (denoted by ε). For reasons of simplicity, $v(i)$ is often denoted by v^i for $i \in \mathbb{N}$. Using this notation, words are often given in the form $v^0 v^1 v^2 \ldots v^n$ or $v^0 v^1 \ldots$. The set of all finite and infinite words over Σ is denoted by Σ^* and Σ^ω, respectively.

Counting of letters starts with zero, i. e. v^{i-1} refers to the i-th letter of v. Furthermore, $v^{i\cdot\cdot}$ denotes the suffix of v starting at position i, i. e. $v^{i\cdot\cdot} = v^i v^{i+1} \ldots$ for all $i < |v|$. The finite word $v^i v^{i+1} \ldots v^j$ is denoted by $v^{i\cdot\cdot j}$. Notice that in case $j < i$ the expression $v^{i\cdot\cdot j}$ evaluates to the empty word ε. For two words v_1, v_2 with $v_1 \in \Sigma^*$, we write $v_1 v_2$ for their concatenation. Finally, we write l^ω for the infinite word v with $v^j = l$ for all j.

A *path* of a labelled transition system corresponds to a word whose letters are the labels of the states of the path. However, the terminology of PSL does not distinguish between paths and words [1]. Therefore, the terms 'path' and 'word' are also used synonymously in this work.

2.1 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) has been introduced by Pnueli in [22]. LTL essentially consists of propositional logic enriched with the temporal operators X and $\underline{\mathsf{U}}$. The formula $\mathsf{X}\varphi$ means that the property φ holds at the next point of time, $\varphi\ \underline{\mathsf{U}}\ \psi$ means that φ holds until ψ holds and that ψ eventually holds. The operators $\overleftarrow{\underline{\mathsf{X}}}$ and $\overleftarrow{\underline{\mathsf{U}}}$ express the same properties for the past instead of the future. Therefore, the operators X and $\underline{\mathsf{U}}$ are called *future operators*, while $\overleftarrow{\underline{\mathsf{X}}}$ and $\overleftarrow{\underline{\mathsf{U}}}$ are called *past operators*.

LTL without past operators is as expressive as LTL with past operators [11]. For this reason, past operators are often neglected, although new results advocate the use of past operators [19] since some properties require an exponential blow-up when past operators are eliminated. Hence, our deep embedding of LTL contains past operators as well, so that we distinguish between the full logic LTL and its future fragment FutureLTL.

Definition 2 (Syntax of Linear Temporal Logic (LTL)). *The set* $\mathsf{ltl}_\mathcal{V}$ *of* LTL *formulas over a given set of variables* $\mathcal{V}$ *is defined as follows:*

- $p \in \mathsf{ltl}_\mathcal{V}$ *for all* $p \in \mathsf{prop}_\mathcal{V}$
- $\neg\varphi,\ \varphi \wedge \psi \in \mathsf{ltl}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{ltl}_\mathcal{V}$
- $\mathsf{X}\varphi,\ \varphi\ \underline{\mathsf{U}}\ \psi \in \mathsf{ltl}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{ltl}_\mathcal{V}$
- $\overleftarrow{\underline{\mathsf{X}}}\varphi,\ \varphi\ \overleftarrow{\underline{\mathsf{U}}}\ \psi \in \mathsf{ltl}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{ltl}_\mathcal{V}$

As usual a lot of further temporal operators can be defined as syntactic sugar like $\mathsf{F}\varphi := (\mathsf{true}\ \underline{\mathsf{U}}\ \psi)$, $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$, $\varphi\ \mathsf{U}\ \psi := \varphi\ \underline{\mathsf{U}}\ \psi \vee \mathsf{G}\varphi$, and $\varphi\ \mathsf{B}\ \psi := \neg(\neg\varphi)\ \underline{\mathsf{U}}\ \psi$. LTL with the operators $\underline{\mathsf{U}}$ and X is, however, already expressively complete with respect to the first order theory of linear orders [25].

Definition 3 (Semantics of Linear Temporal Logic (LTL)). *For* $b \in \mathsf{prop}_\mathcal{V}$ *and* $\varphi, \psi \in \mathsf{ltl}_\mathcal{V}$ *the semantics of* LTL *with respect to an infinite word* $v \in \mathcal{P}(\mathcal{V})^\omega$ *and a point of time* $t \in \mathbb{N}$ *is given as follows:*

- $v \models^t_{\mathsf{ltl}} b$ *iff* $v^t \models_{\mathsf{prop}} b$
- $v \models^t_{\mathsf{ltl}} \neg\varphi$ *iff* $v \not\models^t_{\mathsf{ltl}} \varphi$
- $v \models^t_{\mathsf{ltl}} \varphi \wedge \psi$ *iff* $v \models^t_{\mathsf{ltl}} \varphi$ *and* $v \models^t_{\mathsf{ltl}} \psi$
- $v \models^t_{\mathsf{ltl}} \mathsf{X}\varphi$ *iff* $v \models^{t+1}_{\mathsf{ltl}} \varphi$
- $v \models^t_{\mathsf{ltl}} \varphi\ \underline{\mathsf{U}}\ \psi$ *iff* $\exists k.\ k \geq t\ \wedge\ v \models^k_{\mathsf{ltl}} \psi\ \wedge\ \forall j.\ t \leq j < k \rightarrow v \models^j_{\mathsf{ltl}} \varphi$
- $v \models^t_{\mathsf{ltl}} \overleftarrow{\underline{\mathsf{X}}}\varphi$ *iff* $t > 0\ \wedge\ v \models^{t-1}_{\mathsf{ltl}} \varphi$
- $v \models^t_{\mathsf{ltl}} \varphi\ \overleftarrow{\underline{\mathsf{U}}}\ \psi$ *iff* $\exists k.\ k \leq t\ \wedge\ v \models^k_{\mathsf{ltl}} \psi\ \wedge\ \forall j.\ k < j \leq t \rightarrow v \models^j_{\mathsf{ltl}} \varphi$

A word $v \in \mathcal{P}(\mathcal{V})^\omega$ satisfies *a* LTL *formula* $\varphi \in \mathsf{ltl}_\mathcal{V}$ *(written as* $v \models_{\mathsf{ltl}} \varphi$*) iff* $v \models^0_{\mathsf{ltl}} \varphi$.

2.2 Reset Linear Temporal Logic (RLTL)

To evaluate a formula $\varphi\ \underline{\mathsf{U}}\ \psi$, one has to consider a (potentially infinite) prefix of a path, namely the prefix up to a state where $\neg(\varphi \wedge \neg\psi)$ holds. As simulations may stop before that prefix is completely examined, the evaluation of formulas could be incomplete, and is thus aborted. In order to return a definite truth value, abort operators are introduced. In particular, RLTL [3] is such an extension of FutureLTL:

Definition 4 (Syntax of Reset Linear Temporal Logic (RLTL)). *The following mutually recursive definitions introduce the set* $\mathsf{rltl}_\mathcal{V}$ *of* RLTL *formulas over a given set of variables* $\mathcal{V}$*:*

- *each propositional formula* $p \in \mathsf{prop}_\mathcal{V}$ *is a* RLTL *formula*
- $\neg\varphi$, $\varphi \wedge \psi \in \mathsf{rltl}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{rltl}_\mathcal{V}$
- $\mathsf{X}\varphi$, $\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{rltl}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{rltl}_\mathcal{V}$
- $\mathsf{ACCEPT}(\varphi, b) \in \mathsf{rltl}_\mathcal{V}$, *if* $\varphi \in \mathsf{rltl}_\mathcal{V}$, $b \in \mathsf{prop}_\mathcal{V}$

Some operators like $\neg, \wedge$ or $\underline{\mathsf{U}}$ are used by several logics discussed in this paper. In most cases, it is clear by the context or it does not matter to which logic one of these operators belongs. If it matters, we use subscripts like $\neg_{\mathsf{prop}}$, $\neg_{\mathsf{ltl}}$ and $\neg_{\mathsf{rltl}}$. For example, with $a, b, c \in \mathcal{V}$, note that $\langle \{a,b\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} \neg_{\mathsf{prop}} c$ holds, but $\langle \{a,b\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} \neg_{\mathsf{rltl}} c$ does not hold.

Definition 5 (Semantics of Reset Linear Temporal Logic (RLTL)). *The semantics of* LTL *is defined with respect to a word* v *and a point of time* t*. To define the semantics of* RLTL*, an* acceptance condition $a \in \mathsf{prop}_\mathcal{V}$ *and a* rejection condition $r \in \mathsf{prop}_\mathcal{V}$ *are additionally needed. These conditions are used to capture the required information about* ACCEPT *operators in the context of the formula. Thus, for* $b \in \mathsf{prop}_\mathcal{V}$ *and* $\varphi, \psi \in \mathsf{rltl}_\mathcal{V}$*, the semantics of* RLTL *with respect to an infinite word* $v \in \mathcal{P}(\mathcal{V})^\omega$*, acceptance / rejection conditions* $a, r \in \mathsf{prop}_\mathcal{V}$ *and a point of time* $t \in \mathbb{N}$ *is defined as follows:*

- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} b$ *iff* $v^t \models_{\mathsf{prop}} a$ *or* $(v^t \models_{\mathsf{prop}} b$ *and* $v^t \not\models_{\mathsf{prop}} r)$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \neg\varphi$ *iff* $\langle v, r, a\rangle \not\models^t_{\mathsf{rltl}} \varphi$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \wedge \psi$ *iff* $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi$ *and* $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \psi$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \mathsf{X}\varphi$ *iff* $v^t \models_{\mathsf{prop}} a$ *or* $\left(\langle v, a, r\rangle \models^{t+1}_{\mathsf{rltl}} \varphi \text{ and } v^t \not\models_{\mathsf{prop}} r\right)$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \,\underline{\mathsf{U}}\, \psi$
  $$\text{iff } \exists k.\ k \geq t \ \wedge\ \langle v, a, r\rangle \models^k_{\mathsf{rltl}} \psi \ \wedge\ \forall j.\ t \leq j < k \rightarrow \langle v, a, r\rangle \models^j_{\mathsf{rltl}} \varphi$$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \mathsf{ACCEPT}(\varphi, b)$ *iff* $\langle v, a \vee (b \wedge \neg r), r\rangle \models^t_{\mathsf{rltl}} \varphi$

A word $v \in \mathcal{P}(\mathcal{V})^\omega$ *satisfies a* RLTL *formula* $\varphi \in \mathsf{rltl}_\mathcal{V}$ *(written as* $v \models_{\mathsf{rltl}} \varphi$*) iff* $\langle v, \mathsf{false}, \mathsf{false}\rangle \models^0_{\mathsf{rltl}} \varphi$ *holds.*

$\mathsf{ACCEPT}(\varphi, b)$ aborts the evaluation of the formula φ as soon as the propositional condition b holds: assume we have to check $v \models_{\mathsf{rltl}} \mathsf{ACCEPT}(\varphi \,\underline{\mathsf{U}}\, \psi, b)$ with propositional formulas φ, ψ and b, but only know a finite prefix of v, say $v^{0..t}$. Assume further that on every state v^i with $i \leq t$, we have $v^i \models_{\mathsf{prop}} \varphi \wedge \neg\psi$. Then, we can not decide whether $v \models_{\mathsf{ltl}} \varphi \,\underline{\mathsf{U}}\, \psi$ holds, but nevertheless $v \models_{\mathsf{rltl}} \mathsf{ACCEPT}((\varphi \,\underline{\mathsf{U}}\, \psi), b)$ holds, provided that $v^t \models_{\mathsf{prop}} b$ holds.

For example, the word $\{a\}\{c\}\emptyset^\omega$ does not satisfy the RLTL formula $a \,\underline{\mathsf{U}}\, b$ (since b is never satisfied), but it satisfies the formula $\mathsf{ACCEPT}(a \,\underline{\mathsf{U}}\, b, c)$, since c aborts the until then incomplete evaluation of $a \,\underline{\mathsf{U}}\, b$. On the other hand, the word $\emptyset\{c\}\emptyset^\omega$ does not satisfy $\mathsf{ACCEPT}(a \,\underline{\mathsf{U}}\, b, c)$, since the evaluation of $a \,\underline{\mathsf{U}}\, b$ is completed before c occurs. To understand the impact of the acceptance and rejection conditions and thus, to understand the semantics of the ACCEPT operator, the following lemma is important:

Lemma 1 (Immediate Accept or Reject). *For all infinite words $v \in \mathcal{P}(\mathcal{V})^\omega$, all formulas $\varphi \in \mathsf{rltl}_\mathcal{V}$, all acceptance / rejection conditions $a, r \in \mathsf{prop}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$, the following holds:*

$$\begin{array}{l}\left(v^t \models_{\mathsf{prop}} a \ \wedge \ v^t \not\models_{\mathsf{prop}} r\right) \Longrightarrow \langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \text{ and} \\ \left(v^t \not\models_{\mathsf{prop}} a \ \wedge \ v^t \models_{\mathsf{prop}} r\right) \Longrightarrow \langle v, a, r\rangle \not\models^t_{\mathsf{rltl}} \varphi\end{array}$$

The lemma can be easily proved by structural induction[3] and states that if the acceptance condition immediately holds, every formula (even false) is true. On the other hand, if the rejection condition holds, every formula (even true) is false.

If the acceptance and the rejection condition would hold at the same point of time, then a lot of problems would occur with the semantics. Fortunately, all pairs of acceptance / rejection conditions (a, r) that appear during the evaluation of RLTL formulas satisfy the invariant $\forall s.\, s \models_{\mathsf{prop}} \neg(a \wedge r)$: Initially, the pair $(\mathsf{false}, \mathsf{false})$ is used, and the rules that determine the semantics are easily seen to maintain the invariant.

Therefore, $\forall s.\, s \models_{\mathsf{prop}} \neg(a \wedge r)$ can be assumed for pairs of acceptance / rejection conditions (a, r). This assumption simplifies some proofs, because unreasonable cases can be excluded. In particular, it does not matter if $\neg_{\mathsf{prop}}$ or $\neg_{\mathsf{rltl}}$ is used, if $\forall s.\, s \models_{\mathsf{prop}} \neg(a \wedge r)$ holds[4]. Moreover, the invariant $\neg(a \wedge r)$ is necessary to formulate certain important lemmata like the following one:

Lemma 2. *For all infinite words $v_1, v_2 \in \mathcal{P}(\mathcal{V})^\omega$, all formulas $\varphi \in \mathsf{rltl}_\mathcal{V}$, all acceptance / rejection conditions $a, r \in \mathsf{prop}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$, the following holds[5]:*

$$\begin{array}{l}\Big(\exists k.\, k \geq t \ \wedge \ v_1^{t..k-1} = v_2^{t..k-1} \ \wedge \\ \quad \Big(\left(v_1^k \models_{\mathsf{prop}} a \ \wedge \ v_2^k \models_{\mathsf{prop}} a \ \wedge \ v_1^k \not\models_{\mathsf{prop}} r \ \wedge \ v_2^k \not\models_{\mathsf{prop}} r\right) \vee \\ \quad\quad \left(v_1^k \not\models_{\mathsf{prop}} a \ \wedge \ v_2^k \not\models_{\mathsf{prop}} a \ \wedge \ v_1^k \models_{\mathsf{prop}} r \ \wedge \ v_2^k \models_{\mathsf{prop}} r\right)\Big)\Big) \Longrightarrow \\ \Big(\langle v_1, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \Longleftrightarrow \langle v_2, a, r\rangle \models^t_{\mathsf{rltl}} \varphi\Big)\end{array}$$

Lemma 2 states that if either the acceptance or the rejection condition holds at some point of time $k \geq t$, then it is sufficient to consider the finite prefix $v^{t..k}$ to evaluate arbitrary RLTL formulas at position t. This does no longer hold if both the acceptance and the rejection condition would hold at some point of time: For example, we have $\langle\{a, b\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} a \ \underline{\mathsf{U}} \ \neg_{\mathsf{rltl}} c$, but $\langle\{a, b\}\{c\}\emptyset^\omega, a, b\rangle \not\models^0_{\mathsf{rltl}} a \ \underline{\mathsf{U}} \ \neg_{\mathsf{rltl}} c$. The remaining RLTL operators have the same semantics as the corresponding LTL operators (since RLTL is a superset of LTL).

2.3 Accellera's Property Specification Language

As mentioned above, PSL is a standardised industrial-strength property specification language [1]. PSL was chartered by the Functional Verification Technical

[3] Theorem `RLTL_ACCEPT_REJECT_THM` in theory `ResetLTL_Lemmata`.
[4] Theorem `RLTL_SEM_PROP_RLTL_OPERATOR_EQUIV` in theory `ResetLTL`.
[5] Theorem `RLTL_EQUIV_PATH_STRONG_THM` in theory `ResetLTL_Lemmata`.

Committee of Accellera. The Sugar language [5] was chosen as the basis for PSL. The Language Reference Manual for PSL version 1.0 was released in April 2003. Finally, in June 2004 version 1.1 [1] was released, where some anomalies (like those reported in [3]) were corrected.

PSL is designed as an input language for formal verification and simulation tools as well as a language for documentation. Therefore, it has to be easy to read, and at the same time, it must be precise and highly expressive. In particular, PSL contains features for simulation like finite paths, features for hardware specification like clocked statements and a lot of syntactic sugar.

PSL consists of four layers: The Boolean layer, the temporal layer, the verification layer and the modelling layer. The *Boolean layer* is used to construct expressions that can be evaluated in a single state. The *temporal layer* is the heart of the language. It is used to express properties concerning more than one state, i. e. temporal properties. The temporal layer is divided into the *Foundation Language* (FL) and the *Optional Branching Extension* (OBE). FL is, like LTL, a linear time temporal logic. In contrast, OBE is essentially the branching time temporal logic CTL [9], which is widely used and well understood. The *verification layer* has the task to instruct tools to perform certain actions on the properties expressed by the temporal layer. Finally, the *modelling layer* is used to describe assumptions about the behaviour of inputs and to model auxiliary hardware that is not part of the design. Additionally, PSL comes in four flavours, corresponding to the hardware description languages SystemVerilog, Verilog, VHDL and GDL. These flavours provide a syntax for PSL that is similar to the syntax of the corresponding hardware description language.

In this paper, only the Boolean and the temporal layers will be considered. Furthermore, mainly the formal syntax of PSL is used, which differs from the syntax of all four flavours. However, some operators are denoted differently to the formal syntax to avoid confusion with LTL operators that have the same syntax, but a different semantics.

In this paper, only FL is considered. Therefore, only this subset of PSL is formally introduced here. FL is a linear temporal logic that consists of:

- propositional operators
- future temporal (LTL) operators
- a clocking operator for defining the granularity of time, which may vary for subformulas
- Sequential Extended Regular Expressions (SEREs), for defining finite regular patterns, together with strong and weak promotions of SEREs to formulas and an implication operator for predicating a formula on match of the pattern specified by a SERE
- an abort operator

Due to lack of space, only the subset of PSL that is interesting for the translation will be presented. Therefore, clocks and SEREs are omitted in the following.

The definition of the formal semantics of PSL makes use of two special states $\top$ and $\bot$. State $\top$ satisfies every propositional formula, even the formula false,

and state $\bot$ satisfies no propositional formula, even the formula true is not satisfied. Using these two special states, the semantics of a propositional formula $\varphi \in \mathsf{prop}_{\mathcal{V}}$ with respect to a state $s \in \mathcal{P}(\mathcal{V}) \cup \{\top, \bot\}$ is defined as follows:

- $\top \models_{\mathsf{xprop}} \varphi$
- $\bot \not\models_{\mathsf{xprop}} \varphi$
- $s' \models_{\mathsf{xprop}} \varphi$ iff $s' \models_{\mathsf{prop}} \varphi$ for $s' \in \mathcal{P}(\mathcal{V})$, i. e. for $s' \notin \{\top, \bot\}$

For a given set of variables $\mathcal{V}$, the set of *extended states over* $\mathcal{V}$ is denoted by $\mathcal{XP}(\mathcal{V}) := \mathcal{P}(\mathcal{V}) \cup \{\top, \bot\}$. The definition of the formal syntax of PSL uses a special function for words over these extended states. For finite or infinite words $w \in \mathcal{XP}(\mathcal{V})^\omega \cup \mathcal{XP}(\mathcal{V})^*$, the word $\overline{w}$ denotes the word over states that is obtained from w by replacing every $\top$ with $\bot$ and vice versa, i. e. for all $i < |w|$, the following holds:

$$\overline{w}^i := \begin{cases} \bot & : \text{if } w^i = \top \\ \top & : \text{if } w^i = \bot \\ w^i & : \text{otherwise} \end{cases}$$

Using these extended states and words over these states, it is possible to define the formal syntax and semantics of SERE-free, unclocked FL (short SUFL):

Definition 6 (Syntax of Unclocked, SERE-free Foundation Language (SUFL)). *The set of* SUFL*-formulas* $\mathsf{sufl}_{\mathcal{V}}$ *over a given set of variables* $\mathcal{V}$ *is defined as follows:*

- $p, p! \in \mathsf{sufl}_{\mathcal{V}}$, *if* $p \in \mathsf{prop}_{\mathcal{V}}$
- $\neg\varphi \in \mathsf{sufl}_{\mathcal{V}}$, *if* $\varphi \in \mathsf{sufl}_{\mathcal{V}}$
- $\varphi \wedge \psi \in \mathsf{sufl}_{\mathcal{V}}$, *if* $\varphi, \psi \in \mathsf{sufl}_{\mathcal{V}}$
- $\underline{\mathsf{X}}\varphi$, $\varphi\ \underline{\mathsf{U}}\ \psi$[6] $\in \mathsf{sufl}_{\mathcal{V}}$, *if* $\varphi, \psi \in \mathsf{sufl}_{\mathcal{V}}$
- $\varphi\ \mathsf{ABORT}\ b \in \mathsf{sufl}_{\mathcal{V}}$, *if* $\varphi \in \mathsf{sufl}_{\mathcal{V}}$, $b \in \mathsf{prop}_{\mathcal{V}}$

Definition 7 (Semantics of SUFL). *For propositional formulas* $b \in \mathsf{prop}_{\mathcal{V}}$ *and* SUFL *formulas* $\varphi, \psi \in \mathsf{sufl}_{\mathcal{V}}$, *the semantics of unclocked* SUFL *with respect to a finite or infinite word* $v \in \mathcal{XP}(\mathcal{V})^* \cup \mathcal{XP}(\mathcal{V})^\omega$ *is defined as follows:*

- $v \models_{\mathsf{sufl}} b$ *iff* $|v| = 0$ *or* $v^0 \models_{\mathsf{xprop}} b$
- $v \models_{\mathsf{sufl}} b!$ *iff* $|v| > 0$ *and* $v^0 \models_{\mathsf{xprop}} b$
- $v \models_{\mathsf{sufl}} \neg\varphi$ *iff* $\overline{v} \not\models_{\mathsf{sufl}} \varphi$
- $v \models_{\mathsf{sufl}} \varphi \wedge \psi$ *iff* $v \models_{\mathsf{sufl}} \varphi$ *and* $v \models_{\mathsf{sufl}} \psi$
- $v \models_{\mathsf{sufl}} \underline{\mathsf{X}}\varphi$ *iff* $|v| > 1$ *and* $v^{1..} \models_{\mathsf{sufl}} \varphi$
- $v \models_{\mathsf{sufl}} \varphi\ \underline{\mathsf{U}}\ \psi$ *iff* $\exists k.\, k < |v|$ *s.t.* $v^{k..} \models_{\mathsf{sufl}} \psi$ *and* $\forall j < k.\, v^{j..} \models \varphi$
- $v \models_{\mathsf{sufl}} \varphi\ \mathsf{ABORT}\ b$ *iff either* $v \models_{\mathsf{sufl}} \varphi$ *or* $\exists j.j < |v|$ *s.t.* $v^j \models_{\mathsf{sufl}} b$ *and* $v^{0..j-1}\top^\omega \models_{\mathsf{sufl}} \varphi$

A word v *satisfies an unclocked* FL *formula* φ *iff* $v \models_{\mathsf{sufl}} \varphi$ *holds.*

[6] Written as $\varphi\ \mathsf{U}\ \psi$ in [1].

As usual, some syntactic sugar is defined for SUFL:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\mathsf{X}\varphi := \neg\underline{\mathsf{X}}\neg\varphi$
- $\mathsf{F}\varphi := \mathsf{true}\ \underline{\mathsf{U}}\ \varphi$
- $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$
- $\varphi\ \mathsf{U}\ \psi$[7] $:= \varphi\ \underline{\mathsf{U}}\ \psi\ \vee\ \mathsf{G}\varphi$
- $\varphi\ \mathsf{B}\ \psi$[8] $:= \neg(\neg\varphi\ \underline{\mathsf{U}}\ \psi)$

All SUFL operators correspond to RLTL operators. A difference to RLTL is, that SUFL is able to additionally consider finite paths. Thus, for a propositional formula b, a strong variant $b!$ is introduced that does not hold for the empty word ε, while every propositional formula b holds for the empty word. Analogously, $\underline{\mathsf{X}}$ is introduced as a strong variant of X. The semantics of $\underline{\mathsf{X}}$ requires that a next state exists, while $\mathsf{X}\varphi$ trivially holds if no next state exists. For the remaining temporal operator $\underline{\mathsf{U}}$, a weak variant U is already available in RLTL. Apart from finite paths, the meaning of the FL operators is the same as the meaning of the corresponding RLTL operators. The role of the two special states $\top, \bot$ is played by the acceptance / rejection conditions of RLTL. The proof of this connection between PSL and RLTL is one important part of the translation presented in this work and will be explained in Section 3.

3 From PSL to RLTL

As mentioned above, the temporal layer of PSL consists of FL and OBE. OBE is essentially the well known temporal logic CTL [9]. Since CTL can be directly used for model checking without further translations [7,25], this work only considers FL.

FL with SEREs is strictly more expressive than LTL. For example, it is well known that there is no LTL formula expressing that a proposition φ holds at every even point of time [20,32,25]. However, there is an unclocked FL formula with SEREs expressing this property[9]. As RLTL is as expressive as LTL [3], FL with SEREs cannot be translated to RLTL. Therefore, only SERE-free FL formulas are considered. Moreover, clock-statements can be omitted for reasons of simplicity, because clocked formulas can be easily rewritten to equivalent unclocked ones [1]. Thus, we only consider the translation of unclocked, SERE-free FL to RLTL.

The semantics of SUFL is similar to the semantics of RLTL. There are only two important differences: first, SUFL is able to additionally consider finite paths, and second, SUFL additionally makes use of the special states $\top$ and $\bot$, while RLTL makes use of acceptance and rejection conditions. The first difference is not important in the scope of this paper, because the overall goal is to translate SUFL to ω-automata. Therefore, only infinite paths are of interest. To handle

[7] Written as $[\varphi\ \mathsf{W}\ \psi]$ in [1].
[8] Written as $[\varphi\ \mathsf{BEFORE!\_}\ \psi]$ in [1].
[9] Theorem `PSL_WITH_SERES__STRICTLY_MORE_EXPRESSIVE_THAN__LTL__EXAMPLE` in theory `PSLToRLTL`.

the second difference, the special states $\top$ and $\bot$ are simulated with the acceptance / rejection conditions of RLTL. However, the special states and acceptance / rejection conditions have slightly different semantics: The states $\top$ and $\bot$ determine whether an arbitrary proposition is fulfilled by the current state. However, the remaining states are still important. In contrast, if either the acceptance or the rejection condition occurs, the remaining states can be neglected according to Lemma 2. An example showing this difference is $\bot\{p\}^\omega \models_{\mathsf{sufl}} \underline{\mathsf{X}}p$, but $\langle\{r\}\{p\}^\omega, a, r\rangle \not\models^0_{\mathsf{rltl}} \mathsf{X}p$ for $a, r, p \in \mathcal{V}$. To overcome this slightly different semantics only special inputs are considered:

Definition 8 (PSL-Paths). *A* finite PSL-path *over a set of variables $\mathcal{V}$ is a finite word $v \in \mathcal{P}(\mathcal{V})^*$, i. e. a finite word not containing the states $\top$ and $\bot$. An* infinite PSL-path *over $\mathcal{V}$ is an infinite word $v \in \mathcal{XP}(\mathcal{V})^\omega$ with the following properties:*

- $\forall j.\ v^j{=}\top \longrightarrow v^{j+1}{=}\top$
- $\forall j.\ v^j{=}\bot \longrightarrow v^{j+1}{=}\bot$

The set of all infinite PSL-paths over $\mathcal{V}$ is denoted by $\mathcal{XP}(\mathcal{V})^{\omega^{\top\bot}}$. Notice that $\mathcal{P}(\mathcal{V})^\omega \subset \mathcal{XP}(\mathcal{V})^{\omega^{\top\bot}}$ holds.

In this work, we only consider infinite PSL-paths. At the first glance, this may seem to be a restriction, however, this is not the case: Note that special states are just an auxiliary means used to explain the semantics; however, they do not occur in practice. Hence, only paths that fulfil the additional property of PSL-paths are considered in the following. In [15], PSL-paths are called *proper words*.

Since paths containing the special states $\top$ and $\bot$ are allowed as input of SUFL formulas, but these special states are not allowed as input of RLTL formulas, both paths and formulas have to be translated. To translate the paths, two new atomic propositions t and b are chosen, i. e. t and b do neither occur on the path nor in the formula. Every occurrence of $\top$ on the path is replaced by the state $\{t\}$. In the same way, every occurrence of $\bot$ is replaced by $\{b\}$. For the formula itself, only minor changes are required: Essentially, only the PSL operators are exchanged with the corresponding RLTL operators. Additionally, t and b are used as acceptance and rejection conditions, respectively, while evaluating the translated formula on the translated path.

Lemma 3. *With the definitions of Figure 1, the following are equivalent*[10] *for all $f \in \mathsf{sufl}_\mathcal{V}$, all infinite PSL-paths $v \in \mathcal{XP}(\mathcal{V})^{\omega^{\top\bot}}$ and all $t, b \notin \mathcal{V}$:*

- $v \models_{\mathsf{sufl}} f$
- $\langle \mathsf{RemoveTopBottom}(t, b, v), t, b\rangle \models^0_{\mathsf{rltl}} \mathsf{PSL\_TO\_RLTL}\ f$
- $\mathsf{RemoveTopBottom}(t, b, v) \models_{\mathsf{rltl}} \mathsf{ACCEPT}(\mathsf{REJECT}((\mathsf{PSL\_TO\_RLTL}\ f), b), t)$

Note that t and b never occur at the same point of time on the translated path $\mathsf{RemoveTopBottom}(t, b, v)$.

[10] Theorems `PSL_TO_RLTL_THM` and `PSL_TO_RLTL__ELIM_ACCEPT_REJECT_THM` in theory `PSLToRLTL`.

$$\mathsf{RemoveTopBottom}(t,b,v)^j := \begin{cases} \{t\} & : \textit{if } v^j = \top \\ \{b\} & : \textit{if } v^j = \bot \\ v^j & : \textit{otherwise} \end{cases}$$

function PSL_TO_RLTL(Φ)
  case Φ **of**
    b : **return** b;
    $b!$: **return** b;
    $\neg\varphi$: **return** $\neg$PSL_TO_RLTL(φ);
    $\varphi \wedge \psi$: **return** PSL_TO_RLTL(φ) $\wedge$ PSL_TO_RLTL(ψ);
    $\underline{\mathsf{X}}\varphi$: **return** X(PSL_TO_RLTL(φ));
    $\varphi\ \underline{\mathsf{U}}\ \psi$: **return** PSL_TO_RLTL(φ) $\underline{\mathsf{U}}$ PSL_TO_RLTL(ψ);
    φ ABORT b : **return** ACCEPT(PSL_TO_RLTL(φ), b);
  end
end

Fig. 1. Translation of SUFL to RLTL

The proof of Lemma 3 is based on a structural induction and requires some lemmata about RLTL. In particular, Lemma 1 and 2 are important. To express other important properties in a convenient way, some abbreviations about the occurrence of propositions on a path are convenient:

$$\begin{aligned} \mathsf{NAND\_ON\_PATH}(v,a,r) &:= \forall t.\, \neg(v^t \models_{\mathsf{prop}} a \ \wedge\ v^t \models_{\mathsf{prop}} r) \\ \mathsf{IS\_ON\_PATH}(v,p) &:= \exists t.\, v^t \models_{\mathsf{prop}} p \\ \mathsf{BEFORE\_ON\_PATH}(v,a,b) &:= \forall t.\, (v^t \models_{\mathsf{prop}} b) \Rightarrow \exists t_0.\, (t_0 \leq t \wedge v^{t_0} \models_{\mathsf{prop}} a) \end{aligned}$$

Using these abbreviations, we can formulate the following lemma:

Lemma 4. *For all $v \in \mathcal{P}(\mathcal{V})^\omega$, $a_1, a_2, r \in \mathsf{prop}_\mathcal{V}$, all $\varphi \in \mathsf{rltl}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$, the following holds*[11]*:*

$$\begin{pmatrix} \mathsf{NAND\_ON\_PATH}(v^{t\cdot\cdot}, a_1, r) \ \wedge\ \mathsf{BEFORE\_ON\_PATH}(v^{t\cdot\cdot}, a_1, a_2) \end{pmatrix} \Longrightarrow \\ \begin{pmatrix} \langle v, a_2, r\rangle \models^t_{\mathsf{rltl}} \varphi \Rightarrow \langle v, a_1, r\rangle \models^t_{\mathsf{rltl}} \varphi \end{pmatrix}$$

Informally, this lemma states that valid RLTL formulas do not become invalid if the acceptance condition is strengthened. That is an important property of RLTL. A consequence of Lemma 4 is:

Lemma 5. *For all $v \in \mathcal{P}(\mathcal{V})^\omega$, $a_1, a_2, r \in \mathsf{prop}_\mathcal{V}$, all $\varphi \in \mathsf{rltl}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$, the following property holds*[12]*:*

$$\begin{pmatrix} \mathsf{NAND\_ON\_PATH}(v^{t\cdot\cdot}, a_1, r) \ \wedge\ \mathsf{NAND\_ON\_PATH}(v^{t\cdot\cdot}, a_1, r) \end{pmatrix} \Longrightarrow \\ \begin{pmatrix} \langle v, a_1 \vee a_2, r\rangle \models^t_{\mathsf{rltl}} \varphi \Longleftrightarrow \left(\langle v, a_1, r\rangle \models^t_{\mathsf{rltl}} \varphi \ \vee\ \langle v, a_2, r\rangle \models^t_{\mathsf{rltl}} \varphi \right) \end{pmatrix}$$

[11] Theorem `RLTL_SEM_TIME___ACCEPT_BEFORE_ON_PATH` in theory `ResetLTL_Lemmata`.
[12] Theorem `RLTL_SEM_TIME___ACCEPT_OR_THM` in theory `ResetLTL_Lemmata`.

Using these lemmata about the acceptance / rejection conditions of RLTL, the remaining proof of Lemma 3 by structural induction is mainly technical. The cases for $b, b!, \neg\varphi$ and $\varphi \wedge \psi$ are obvious. The case for $\underline{\mathsf{X}}\varphi$ uses the fact that only infinite PSL-paths are considered, the rest it is technical. The same holds for $\varphi\ \underline{\mathsf{U}}\ \psi$. Therefore, the only interesting case is that for φ ABORT b, where the presented lemmata about RLTL are required in a case analysis.

The usage of HOL to formally prove Lemma 3 has been shown valuable. The case analysis used to prove the case for ABORT is quite tricky. During this case analysis, a small, until then unknown bug in Mike Gordon's deep-embedding of PSL has been discovered: The unclocked semantics of ABORT is defined by $v \models_{\mathsf{sufl}} \varphi$ ABORT b iff either $v \models_{\mathsf{sufl}} \varphi$ or $\exists j.j < |v|$ s.t. $v^j \models_{\mathsf{sufl}} b$ and $v^{0..j-1}\top^\omega \models_{\mathsf{sufl}} \varphi$ holds. This has been literally implemented in HOL. In case of $j = 0$, the word $v^{0..0-1}\top^\omega$ is evaluated to $\top^\omega$ by the formal semantics of PSL. However, because the datatype used to model j in HOL represents natural numbers, $v^{0..0-1}\top^\omega$ evaluated to $v^{0..0}\top^\omega$ and therefore, to $v^0\top^\omega$ in the HOL representation. After reporting this bug to Mike Gordon, it has been fixed.

Lemma 3 is the central result for the translation of PSL to RLTL. It considers arbitrary infinite PSL-paths as inputs. However, one is usually interested only in paths without special states. Restricting the allowed input paths, Lemma 3 directly leads to the following theorem:

Theorem 1 (Translation of SUFL to RLTL). *For all infinite words $v \in \mathcal{P}(\mathcal{V})^\omega$ and all $\varphi \in \mathsf{sufl}_\mathcal{V}$[13], the following holds:*

$$v \models_{\mathsf{sufl}} \varphi \Longleftrightarrow v \models_{\mathsf{rltl}} \mathsf{PSL\_TO\_RLTL}(\varphi).$$

4 From RLTL to LTL

The translation of RLTL to LTL that is used here is due to [3]. The correctness of this translation can be easily proved by structural induction.

Theorem 2 (Translation of RLTL to LTL). *With the definition of Figure 2, the following holds*[14] *for all infinite words $v \in \mathcal{P}(\mathcal{V})^\omega$, all acceptance / rejection conditions $a, r \in \mathsf{prop}_\mathcal{V}$, all RLTL formulas $\varphi \in \mathsf{rltl}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$:*

$$\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \Longleftrightarrow v \models^t_{\mathsf{ltl}} \mathsf{RLTL\_TO\_LTL}(a, r, \varphi)$$

Obviously, this can be instantiated to:

$$v \models_{\mathsf{rltl}} \varphi \Longleftrightarrow v \models_{\mathsf{ltl}} \mathsf{RLTL\_TO\_LTL}(\mathsf{false}, \mathsf{false}, \varphi)$$

[13] Theorem `PSL_TO_RLTL___NO_TOP_BOT_THM` in theory `PSLToRLTL`.
[14] Theorem `RLTL_TO_LTL_THM` in theory `ResetLTL_Lemmata`.

```
function RLTL_TO_LTL(a, r, Φ)
    case Φ of
        b              : return a ∨ (b ∧ ¬r);
        ¬φ             : return ¬RLTL_TO_LTL(r, a, φ);
        φ ∧ ψ          : return RLTL_TO_LTL(a, r, φ) ∧ RLTL_TO_LTL(a, r, ψ);
        Xφ             : return a ∨ (X(RLTL_TO_LTL(a, r, φ)) ∧ ¬r);
        φ U ψ          : return RLTL_TO_LTL(a, r, φ) U RLTL_TO_LTL(a, r, ψ);
        ACCEPT(φ, b)   : return RLTL_TO_LTL(a ∨ (b ∧ ¬r), r, φ);
    end
end
```

Fig. 2. Translation of RLTL to LTL

5 Temporal Logic Hierarchy for PSL

In [25], LTL classes $\mathsf{LTL_F}$, $\mathsf{LTL_G}$, $\mathsf{LTL_{Prefix}}$, $\mathsf{LTL_{FG}}$, $\mathsf{LTL_{GF}}$ and $\mathsf{LTL_{Streett}}$ are syntactically identified, that are as expressive as deterministic, noncounting liveness ($\mathsf{TDET_F}$), safety ($\mathsf{TDET_G}$), prefix ($\mathsf{TDET_{Prefix}}$), persistence ($\mathsf{TDET_{FG}}$), Büchi ($\mathsf{TDET_{GF}}$) and Streett automata ($\mathsf{TDET_{Streett}}$), respectively.

The translation from PSL to LTL adds additional Boolean expressions to the translated formulas. Adding Boolean expressions does not affect the membership of a formula in these classes. Therefore, it is straightforward to identify classes of unclocked, SERE-free FL, which correspond to the classes of LTL (see Figure 3). Similarly, we have identified a hierarchy of RLTL.

$$
\begin{aligned}
b &\in \mathsf{SUFL_G} \\
b! &\in \mathsf{SUFL_G} \\
\neg\varphi &\in \mathsf{SUFL_G} = \varphi \in \mathsf{SUFL_F} \\
\varphi \wedge \psi &\in \mathsf{SUFL_G} = \varphi \in \mathsf{SUFL_G} \wedge \psi \in \mathsf{SUFL_G} \\
\underline{\mathsf{X}}\varphi &\in \mathsf{SUFL_G} = \varphi \in \mathsf{SUFL_G} \\
\varphi \,\underline{\mathsf{U}}\, \psi &\in \mathsf{SUFL_G} = \mathsf{false} \\
\varphi \;\mathsf{ABORT}\; b &\in \mathsf{SUFL_G} = \varphi \in \mathsf{SUFL_G}
\end{aligned}
\qquad
\begin{aligned}
b &\in \mathsf{SUFL_F} \\
b! &\in \mathsf{SUFL_F} \\
\neg\varphi &\in \mathsf{SUFL_F} = \varphi \in \mathsf{SUFL_G} \\
\varphi \wedge \psi &\in \mathsf{SUFL_F} = \varphi \in \mathsf{SUFL_F} \wedge \psi \in \mathsf{SUFL_F} \\
\underline{\mathsf{X}}\varphi &\in \mathsf{SUFL_F} = \varphi \in \mathsf{SUFL_F} \\
\varphi \,\underline{\mathsf{U}}\, \psi &\in \mathsf{SUFL_F} = \varphi \in \mathsf{SUFL_F} \wedge \psi \in \mathsf{SUFL_F} \\
\varphi \;\mathsf{ABORT}\; b &\in \mathsf{SUFL_F} = \varphi \in \mathsf{SUFL_F}
\end{aligned}
$$

$$
\begin{aligned}
b &\in \mathsf{SUFL_{GF}} \\
b! &\in \mathsf{SUFL_{GF}} \\
\neg\varphi &\in \mathsf{SUFL_{GF}} = \varphi \in \mathsf{SUFL_{FG}} \\
\varphi \wedge \psi &\in \mathsf{SUFL_{GF}} = \varphi \in \mathsf{SUFL_{GF}} \wedge \psi \in \mathsf{SUFL_{GF}} \\
\underline{\mathsf{X}}\varphi &\in \mathsf{SUFL_{GF}} = \varphi \in \mathsf{SUFL_{GF}} \\
\varphi \,\underline{\mathsf{U}}\, \psi &\in \mathsf{SUFL_{GF}} = \varphi \in \mathsf{SUFL_{GF}} \wedge \psi \in \mathsf{SUFL_F} \\
\varphi \;\mathsf{ABORT}\; b &\in \mathsf{SUFL_{GF}} = \varphi \in \mathsf{SUFL_{GF}}
\end{aligned}
\qquad
\begin{aligned}
b &\in \mathsf{SUFL_{FG}} \\
b! &\in \mathsf{SUFL_{FG}} \\
\neg\varphi &\in \mathsf{SUFL_{FG}} = \varphi \in \mathsf{SUFL_{GF}} \\
\varphi \wedge \psi &\in \mathsf{SUFL_{FG}} = \varphi \in \mathsf{SUFL_{FG}} \wedge \psi \in \mathsf{SUFL_{FG}} \\
\underline{\mathsf{X}}\varphi &\in \mathsf{SUFL_{FG}} = \varphi \in \mathsf{SUFL_{FG}} \\
\varphi \,\underline{\mathsf{U}}\, \psi &\in \mathsf{SUFL_{FG}} = \varphi \in \mathsf{SUFL_{FG}} \wedge \psi \in \mathsf{SUFL_{FG}} \\
\varphi \;\mathsf{ABORT}\; b &\in \mathsf{SUFL_{FG}} = \varphi \in \mathsf{SUFL_{FG}}
\end{aligned}
$$

$$
\begin{aligned}
b &\in \mathsf{SUFL_{Prefix}} \\
b! &\in \mathsf{SUFL_{Prefix}} \\
\neg\varphi &\in \mathsf{SUFL_{Prefix}} = \varphi \in \mathsf{SUFL_{Prefix}} \\
\varphi \wedge \psi &\in \mathsf{SUFL_{Prefix}} = \varphi \in \mathsf{SUFL_{Prefix}} \wedge \psi \in \mathsf{SUFL_{Prefix}} \\
\underline{\mathsf{X}}\varphi &\in \mathsf{SUFL_{Prefix}} = \mathsf{X}\varphi \in \mathsf{SUFL_G} \cup \mathsf{SUFL_F} \\
\varphi \,\underline{\mathsf{U}}\, \psi &\in \mathsf{SUFL_{Prefix}} = \varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL_G} \cup \mathsf{SUFL_F} \\
\varphi \;\mathsf{ABORT}\; b &\in \mathsf{SUFL_{Prefix}} = \varphi \in \mathsf{SUFL_{Prefix}}
\end{aligned}
\qquad
\begin{aligned}
b &\in \mathsf{SUFL_{Streett}} \\
b! &\in \mathsf{SUFL_{Streett}} \\
\neg\varphi &\in \mathsf{SUFL_{Streett}} = \varphi \in \mathsf{SUFL_{Streett}} \\
\varphi \wedge \psi &\in \mathsf{SUFL_{Streett}} = \varphi \in \mathsf{SUFL_{Streett}} \wedge \psi \in \mathsf{SUFL_{Streett}} \\
\underline{\mathsf{X}}\varphi &\in \mathsf{SUFL_{Streett}} = \mathsf{X}\varphi \in \mathsf{SUFL_{GF}} \cup \mathsf{SUFL_{FG}} \\
\varphi \,\underline{\mathsf{U}}\, \psi &\in \mathsf{SUFL_{Streett}} = \varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL_{GF}} \cup \mathsf{SUFL_{FG}} \\
\varphi \;\mathsf{ABORT}\; b &\in \mathsf{SUFL_{Streett}} = \varphi \in \mathsf{SUFL_{Streett}}
\end{aligned}
$$

Fig. 3. Classes of SUFL

We formally proved in HOL that the presented translation translate every PSL class to the corresponding classes of RLTL and LTL. Moreover, the classes of FutureLTL are as expressive as the classes of LTL [25], and FutureLTL is a subset of RLTL and SUFL. Therefore, we formally proved in HOL that the classes of FutureLTL can be translated to the corresponding classes of PSL and RLTL. This leads to the following theorem:

Theorem 3 (Hierarchy of PSL). *For any* $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$*, the logics* LTL_κ*,* $\mathsf{FutureLTL}_\kappa$*,* RLTL_κ *and* SUFL_κ *are as expressive as* TDET_κ*. Furthermore,* LTL*,* FutureLTL*,* RLTL *and* SUFL *are as expressive as* $\mathsf{TDET}_{\mathsf{Streett}}$*.*

6 Conclusion and Future Work

We presented a translation of a significant subset of PSL to LTL. This translation is interesting by its own, since it allows an efficient translation from this significant subset of PSL to ω-automata. Moreover, it is possible to extend the temporal logic hierarchy [18,24,25] to PSL. In particular, we were able to characterise subsets of PSL that can be translated to liveness and safety automata. This is of practical evidence, since these kinds of automata are very useful to handle finite inputs which is required for bounded model checking or for simulation.

Our main goal is to translate PSL to ω-automata. Since the translation of LTL to ω-automata is well known, we have already done a big step. Unfortunately, regular expressions can, in general, not be translated to LTL. However, they can be translated to finite state automata [16]. Therefore, the next step will be to translate PSL directly to ω-automata. We have already deeply embedded automaton formulas [24,25], a symbolic representation of ω-automata. Furthermore, we have validated a basic and an improved translation of LTL to ω-automata, which are both presented in [24,25]. The improved translation allows us to formally validate the translation of SUFL_F, SUFL_G and $\mathsf{SUFL}_{\mathsf{Prefix}}$ to TDET_F, TDET_G or $\mathsf{TDET}_{\mathsf{Prefix}}$, respectively. Next, we will validate more optimised translations. This will allow us to formally validate that also the other classes of PSL can be translated to the corresponding classes of ω-automata. Then, we can use these optimised translations to directly translate a subset of PSL including regular expressions to ω-automata.

References

1. Accellera. Property specification language reference manual, version 1.1. http://www.eda.org, June 2004.
2. ANSI/IEEE Std 1076-1987. *IEEE Standard VHDL Language Reference Manual.* New York, USA, March 1987.
3. Armoni, R., Bustan, D., Kupferman, O., and Vardi, M. Resets vs. aborts in linear temporal logic. In *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (Warsaw, Poland, 2003), H. Garavel and J. Hatcliff, Eds., vol. 2619 of *LNCS*, Springer, pp. 65–80.

4. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M., and Zbar, Y. The ForSpec temporal logic: A new temporal property-specification language. In *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (Grenoble, France, 2002), J.-P. Katoen and P. Stevens, Eds., vol. 2280 of *LNCS*, Springer, pp. 296–211.
5. Beer, I., Ben-David, S., Eisner, C., Fisman, D., Gringauze, A., and Rodeh, Y. The temporal logic Sugar. In *Conference on Computer Aided Verification (CAV)* (Paris, France, 2001), vol. 2102 of *LNCS*, Springer, pp. 363–367.
6. Chang, K.-H., Tu, W.-T., Yeh, Y.-J., and Kuo, S.-Y. A temporal assertion extension to Verilog. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)* (2004), vol. 3299 of *LNCS*, Springer, pp. 499–504.
7. Clarke, E., Emerson, E., and Sistla, A. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS) 8*, 2 (April 1986), 244–263.
8. Daniele, M., Giunchiglia, F., and Vardi, M. Improved automata generation for linear temporal logic. In *Conference on Computer Aided Verification (CAV)* (Trento, Italy, 1999), N. Halbwachs and D. Peled, Eds., vol. 1633 of *LNCS*, Springer, pp. 249–260.
9. Emerson, E., and Clarke, E. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming 2*, 3 (1982), 241–266.
10. Emerson, E., and Lei, C.-L. Modalities for model checking: Branching time strikes back. *Science of Computer Programming 8* (1987), 275–306.
11. Gabbay, D., Pnueli, A., Shelah, S., and Stavi, J. On the temporal analysis of fairness. In *Symposium on Principles of Programming Languages (POPL)* (New York, 1980), ACM, pp. 163–173.
12. Gastin, P., and Oddoux, D. Fast LTL to Büchi automata translation. In *Conference on Computer Aided Verification (CAV)* (Paris, France, 2001), vol. 2102 of *LNCS*, Springer, pp. 53–65.
13. Gerth, R., Peled, D., Vardi, M., and Wolper, P. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification (PSTV)* (Warsaw, June 1995), North Holland.
14. Gordon, M. PSL semantics in higher order logic. In *Workshop on Designing Correct Circuits (DCC)* (Barcelona, Spain, 2004).
15. Havlicek, J., Fisman, D., and Eisner, C. Basic results on the semantics of Accellera PSL 1.1 foundation language. Technical Report 2004.02, Accellera, 2004.
16. Kleene, S. Representation of events in nerve nets and finite automata. In *Automata Studies*, C. Shannon and J. McCarthy, Eds. Princeton University Press, Princeton, NJ, 1956, pp. 3–41.
17. Landweber, L. Decision problems for ω-automata. *Mathematical Systems Theory 3*, 4 (1969), 376–384.
18. Manna, Z., and Pnueli, A. A hierarchy of temporal properties. In *Symposium on Principles of Distributed Computing* (1990), pp. 377–408.
19. Markey, N. Temporal logic with past is exponentially more succinct. *Bulletin of the European Association for Theoretical Computer Science 79* (2003), 122–128.
20. McNaughton, R., and Papert, S. *Counter-free Automata*. MIT, 1971.
21. Moorby, P. History of Verilog. *IEEE Design and Test of Computers* (September 1992), 62–63.

22. Pnueli, A. The temporal logic of programs. In *Symposium on Foundations of Computer Science (FOCS)* (New York, 1977), vol. 18, IEEE Computer Society, pp. 46–57.
23. Reetz, R., Schneider, K., and Kropf, T. Formal specification in VHDL for formal hardware verification. In *Design, Automation and Test in Europe (DATE)* (February 1998), IEEE Computer Society.
24. Schneider, K. Improving automata generation for linear temporal logic by considering the automata hierarchy. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)* (Havanna, Cuba, 2001), vol. 2250 of *LNAI*, Springer, pp. 39–54.
25. Schneider, K. *Verification of Reactive Systems – Formal Methods and Algorithms.* Texts in Theoretical Computer Science (EATCS Series). Springer, 2003.
26. Schneider, K., and Hoffmann, D. A HOL conversion for translating linear time temporal logic to omega-automata. In *Higher Order Logic Theorem Proving and its Applications (TPHOL)* (Nice, France, 1999), Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, Eds., vol. 1690 of *LNCS*, Springer, pp. 255–272.
27. Schuele, T., and Schneider, K. Bounded model checking of infinite state systems: Exploiting the automata hierarchy. In *Formal Methods and Models for Codesign (MEMOCODE)* (San Diego, CA, June 2004), IEEE, pp. 17–26.
28. Somenzi, F., and Bloem, R. Efficient Büchi automata from LTL formulae. In *Conference on Computer Aided Verification (CAV)* (Chicago, IL, USA, 2000), E. Emerson and A. Sistla, Eds., vol. 1855 of *LNCS*, Springer, pp. 248–263.
29. Thomas, W. *Automata on Infinite Objects*, vol. B. Elsevier, 1990, ch. Automata on Infinite Objects, pp. 133–191.
30. *IEEE Standard VHDL Language Reference Manual.* New York, USA, June 1993. ANSI/IEEE Std 1076-1993.
31. Wagner, K. On ω-regular sets. *Information and Control 43* (1979), 123–177.
32. Wolper, P. Temporal logic can be more expressive. In *Symposium on Foundations of Computer Science (FOCS)* (New York, 1981), IEEE Computer Society, pp. 340–348.
33. Wolper, P., Vardi, M., and Sistla, A. Reasoning about infinite computations paths. In *Symposium on Foundations of Computer Science (FOCS)* (New York, 1983), IEEE Computer Society, pp. 185–194.