Model Checking PSL Using HOL and SMV

Thomas Tuerk[1,*], Klaus Schneider[1], and Mike Gordon[2]

[1] Reactive Systems Group
Department of Computer Science, University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
http://rsg.informatik.uni-kl.de
[2] University of Cambridge Computer Laboratory
William Gates Building, JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
http://www.cl.cam.ac.uk

Abstract. In our previous work, we formally validated the correctness of a translation from a subset of Accellera's Property Specification Language (PSL) to linear temporal logic (LTL) using the HOL theorem prover. We also built an interface from HOL to the SMV model checker based on a formal translation of LTL to ω-automata. In the present paper, we describe how this work has been extended and combined to produce a model checking infrastructure for a significant subset of PSL that works by translating model checking problems to equivalent checks for the existence of fair paths through a Kripke structure specified in higher order logic. This translation is done by theorem proving in HOL, so it is proven to be correct. The existence check is carried out using the interface from HOL to SMV. Moreover, we have applied our infrastructure to implement a tool for validating the soundness of a separate PSL model checker.

1 Introduction

The Property Specification Language (PSL) [1] is an industrial-strength temporal logic. It was developed by the Functional Verification Technical Committee of Accellera based on IBM's Sugar language [3] and has now become an IEEE standard. It is designed both for formal verification and for simulation and has been described as the most popular property specification language in industry [10].

The linear time subset of PSL is a complex language that includes many special cases with subtle semantics. It is well known how LTL can be translated to equivalent ω-automata [30,9,13,12,23], but PSL additionally provides a reset (`abort`) operator whose semantics has been the subject of debate. In order to study the impact of different kinds of abort operators on the complexity of the translation and verification, a logic RLTL [2] was introduced that extends LTL by a reset operator. It turned out that, in the worst case, Version 1.01 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 were changed in Version 1.1 (the current

[*] This work has been done while this author visited the University of Cambridge Computer Laboratory.

version). Thus, a significant subset of PSL can now be translated to RLTL. A further translation from RLTL to LTL has already been presented in [2].

Because of the subtle semantics of PSL, it is non-trivial to ensure that implementations accurately reflect the official language standard. Thus, we feel that there is value in using automated formal methods to reason about the semantics of PSL in general, and to verify model checking algorithms for this logic. PSL has already been deeply embedded in HOL [15] and a translation from a significant subset of PSL to ω-automata via RLTL and LTL has been verified [26,25]. However, in this previous work only the correctness of these translations has been proved.

In this paper, we use revised versions of the correctness translation theorems to create PSL implementation infrastructure directly on top of the formalisation of the standard PSL semantics. Model checking problems for PSL can be handled fully automatically. We have used this infrastructure to build a specific tool to check the accuracy of an implementation of PSL used by IBM's RuleBase CTL model checker. We were able to detect an incorrectness (unknown to us, but known to IBM) in the implementation of clocked `aborts` (they are treated as synchronous but should have been asynchronous).

Our infrastructure includes formal translators, implemented by theorem-proving in HOL, from the linear time fragment of PSL to LTL and from LTL to ω-automata. Although these are based on previous work, they were largely rewritten so that they could be turned into a new automatic tool for translating PSL to automata. To check the existence of fair paths, we use a link from HOL to SMV. This is based on Schneider's earlier work, though we changed from a shallow to a deep embedding of LTL in HOL and modified many details. Model checking problems for PSL can be translated, using theorem proving, to equivalent checks for the existence of fair paths through a Kripke structure. A proof of the correctness of the emptiness check is created by these translation procedures. The resulting check is finally performed by SMV [19].

The rest of this paper is organised as follows. The formalisms we use are explained in the next section. We then briefly sketch translations between them. In Section 4, we describe the infrastructure and in Section 5, we outline its application to build a tool to validate the handling of PSL by RuleBase. Finally, we draw some conclusions and show directions for future work.

2 Basic Notions

Temporal logics like LTL, RLTL and PSL use propositional logic to describe (static) properties of the current point of time. The semantics of temporal properties is based on sequences of points of time called *paths*, which are usually defined by transition systems. Thus, we first define propositional logic, paths and transition systems in this section. Then, the logics LTL, RLTL, and PSL are presented. Finally, ω-automata are introduced.

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 formula φ.*

We use the operators $\vee$, $\rightarrow$ and $\leftrightarrow$ and the constants true and false as syntactic sugar with their usual meaning.

A finite word v over a set Σ of length $|v| = n+1$ is a function $v : \{0, \dots n\} \rightarrow \Sigma$. 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 $v^0 v^1 v^2 \dots v^n$ or $v^0 v^1 \dots$. The set of all finite and infinite words over Σ is denoted by Σ^* and Σ^ω, respectively.

Counting starts from zero, i. e. v^{i-1} refers to the i-th letter of v. Furthermore, $v^{i..}$ denotes the suffix of v starting at position i, i. e. $v^{i..} = v^i v^{i+1} \dots$ for all $i < |v|$. The finite word $v^i v^{i+1} \dots v^j$ is denoted by $v^{i..j}$. Notice that in case $j < i$, the expression $v^{i..j}$ 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. The union $v_1 \cup v_2$ of two words v_1, v_2 with $|v_1| = |v_2|$ over sets is defined as the word v with $|v| = |v_1| = |v_2|$ and $v^j = v_1^j \cup v_2^j$ for all $j < |v|$. Analogously, the intersection $v_1 \cap v_2$ of v_1 and v_2 is defined. We write l^ω for the infinite word v with $v^j = l$ for all j.

2.1 Kripke Structures

Systems used with model checking techniques are usually given as labelled transition systems that are often called Kripke structures. In this paper, we use symbolically represented Kripke structures as usual in symbolic model checking.

Definition 2 (Symbolically Represented Kripke Structures). *A* symbolically represented Kripke structure $\mathcal{K}$ *over a set of variables $\mathcal{V}$ is a tuple $\mathcal{K} = (\mathcal{I}, \mathcal{R})$ such that*

- *$\mathcal{I}$ is a propositional formula over $\mathcal{V}$*
- *$\mathcal{R}$ is a propositional formula over* $\mathcal{V} \cup \{\mathtt{X}v \mid v \in \mathcal{V}\}$

A path *p through $\mathcal{K} = (\mathcal{I}, \mathcal{R})$ is an infinite word over $\mathcal{V}$ such that for all i, the relation $p^i \cup \{\mathtt{X}v \mid v \in p^{i+1}\} \models_{\mathsf{prop}} \mathcal{R}$ holds. A path p is called* initial, *iff $p^0 \models_{\mathsf{prop}} \mathcal{I}$ holds. A path is called* fair *according to some propositional formula f, called the* fairness condition, *iff infinitely many letters of p satisfy the fairness condition, i. e. iff the set $\{i \mid p^i \models_{\mathsf{prop}} f\}$ is infinite. The set of all initial paths through $\mathcal{K}$ is denoted by $\mathsf{IPath}(\mathcal{K})$. The set of all initial paths that satisfy all fairness constraints in the set fc is denoted by $\mathsf{IPath}_{\mathsf{fair}}(\mathcal{K}, fc)$.*

According to this definition, the new variable $\mathtt{X}v$ is used to denote the value of the variable v at the next state. It is often convenient to evaluate a whole propositional formula instead of just one variable at the next state, so the $\mathbf{X}$ operator is introduced as a shorthand for replacing every occurrence of a variable v by $\mathtt{X}v$ in a propositional formula. Similarly, $\mathbf{X}$ is also used to replace every variable v in a set by $\mathtt{X}v$.

2.2 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) has been proposed for the specification of reactive systems by Pnueli in [20]. 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.

Definition 3 (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* $\varphi, \psi \in \mathsf{ltl}_{\mathcal{V}}$

Further temporal operators can be defined as syntactic sugar, for example, $\mathsf{F}\varphi := (\mathsf{true}\ \underline{\mathsf{U}}\ \psi)$, $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$, $\varphi\ \mathsf{U}\ \psi := \varphi\ \underline{\mathsf{U}}\ \psi \vee \mathsf{G}\varphi$, 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 [23].

Definition 4 (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:*

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

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 $v \models_{\mathsf{ltl}}^0 \varphi$; a Kripke structure $\mathcal{K}$* satisfies *φ (denoted $\mathcal{K} \models_{\mathsf{ltl}} \varphi$) iff all paths $v \in \mathsf{IPath}(\mathcal{K})$ satisfy φ.*

2.3 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. The logic RLTL [2] extends LTL with an abort operator called ACCEPT. This operator aborts the evaluation and accepts a path, if a boolean condition is detected.

Definition 5 (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* $\varphi, \psi \in \mathsf{rltl}_{\mathcal{V}}$
- $\mathsf{X}\varphi$, $\varphi \;\underline{\mathsf{U}}\; \psi \in \mathsf{rltl}$, *if* $\varphi, \psi \in \mathsf{rltl}_{\mathcal{V}}$
- $\mathsf{ACCEPT}(\varphi, b) \in \mathsf{rltl}_{\mathcal{V}}$, *if* $\varphi \in \mathsf{rltl}_{\mathcal{V}}$, $b \in \mathsf{prop}_{\mathcal{V}}$

Definition 6 (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 needed in addition. 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* $\big(\langle v, a, r\rangle \models^{t+1}_{\mathsf{rltl}} \varphi$ *and* $v^t \not\models_{\mathsf{prop}} r\big)$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \;\underline{\mathsf{U}}\; \psi$
  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; a Kripke structure* $\mathcal{K}$ satisfies φ *(denoted* $\mathcal{K} \models_{\mathsf{rltl}} \varphi$*) iff all paths* $v \in \mathsf{IPath}(\mathcal{K})$ *satisfy* φ*.*

2.4 Accellera's Property Specification Language

PSL is a standardised industrial-strength property specification language [1] chartered by the Functional Verification Technical Committee of Accellera. The Sugar language [3] 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 [2]) 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 as readable as possible, 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 [11], which is widely used and well understood. The *verification layer* has the task of instructing 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 properties that cannot be represented by formulas of the temporal layer or auxiliary hardware that is not part of the design. 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 slightly differently to the formal syntax to avoid confusion with similar LTL operators.

In this paper, only the linear temporal logic FL is considered. It 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 FL that is interesting for the translation will be presented (e.g. clocks are eliminated using standard rewriting rules). Note that we do not handle SEREs yet.

As described in Version 1.1 of the PSL standard, two special states $\top$ and $\bot$ are needed to define the formal semantics of FL. The 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 $w \in \mathcal{XP}(\mathcal{V})^* \cup \mathcal{XP}(\mathcal{V})^\omega$, 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, the formal syntax and semantics of SERE-free, unclocked FL (which we call SUFL) is defined as follows.

Definition 7 (Syntax of 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* $\varphi, \psi \in \mathsf{sufl}_\mathcal{V}$
- $\underline{\mathsf{X}}\varphi$, $\varphi\ \underline{\mathsf{U}}\ \psi$[1] $\in \mathsf{sufl}_\mathcal{V}$, *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 8 (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* 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* $\forall j.\ j < k$ *implies* $v^{j..} \models_{\mathsf{sufl}} \varphi$
- $v \models_{\mathsf{sufl}} \varphi\ \mathsf{ABORT}\ b$ *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 a* SUFL *formula* φ *iff* $v \models_{\mathsf{sufl}} \varphi$ *holds; a Kripke structure* $\mathcal{K}$ satisfies φ *(denoted* $\mathcal{K} \models_{\mathsf{sufl}} \varphi$*) iff all paths* $v \in \mathsf{IPath}(\mathcal{K})$ *satisfy* φ*.*

Some standard syntactic sugar is defined for SUFL:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\mathsf{X}\varphi := \neg\underline{\mathsf{X}}\neg\varphi$
- $\mathsf{F}\varphi := \mathsf{true}\ \underline{\mathsf{U}}\ \varphi$
- $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$
- $\varphi\ \mathsf{U}\ \psi$[2] $:= \varphi\ \underline{\mathsf{U}}\ \psi\ \vee\ \mathsf{G}\varphi$
- $\varphi\ \mathsf{B}\ \psi$[3] $:= \neg(\neg\varphi\ \underline{\mathsf{U}}\ \psi)$

[1] Written as $\varphi\ \mathsf{U}\ \psi$ in [1].
[2] Written as $[\varphi\ \mathsf{W}\ \psi]$ in [1].
[3] Written as $[\varphi\ \mathsf{BEFORE!\_}\ \psi]$ in [1].

All SUFL operators correspond to RLTL operators. A difference from RLTL is that SUFL is able, in addition, to 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.

2.5 ω-Automata

ω-automata were introduced by J. R. Büchi in 1960 [7]. They are similar to finite state automata as introduced by Kleene in 1956 [18]. While finite state automata decide whether a finite word belongs to some language, ω-automata decide this property for infinite words. There are different kinds of ω-automata and some of slightly different definitions. In this paper, we will use a symbolic representation of nondeterministic ω-automata, which is closely related to the formalism of automaton formulas described in previous work [23]. For conciseness, some details in this paper have been simplified.

Definition 9 (Symbolic Representation of ω-Automata). *A* symbolically represented nondeterministic or universal ω-automaton *over a set of variables $\mathcal{V}$ is a tuple $(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$ such that*

- *$\mathcal{Q} \subseteq \mathcal{V}$ is a finite set of* state variables,
- *$\mathcal{I}$ and $\mathcal{R}$ represent a Kripke structure over $\mathcal{V}$ and*
- *l is a* LTL *formula over $\mathcal{V}$ called* acceptance condition.

Symbolically represented nondeterministic ω-automata are often written in the form $\mathcal{A}_{\exists}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$, universal ones are denoted by $\mathcal{A}_{\forall}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$. A run *of some input $i \in \mathcal{P}(\mathcal{V} \setminus \mathcal{Q})^{\omega}$ through $\mathfrak{A} := \mathcal{A}_{\exists/\forall}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$ is an infinite word $r \in \mathcal{P}(\mathcal{Q})^{\omega}$ such that $i \cup r$ is a path through the Kripke structure represented by $(\mathcal{I}, \mathcal{R})$. The input i satisfies the ω-automaton (denoted by $i \models_{\mathsf{omega}} \mathcal{A}_{\exists/\forall}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$) iff for at least one run / all runs r of i through $\mathfrak{A}$ the path $r \cup i$ satisfies l. For inputs that contain state variables, this definition is extended by restricting the inputs: $i \models_{\mathsf{omega}} \mathfrak{A} \quad := \quad i \cap (\mathcal{V} \setminus \mathcal{Q})^{\omega} \models_{\mathsf{omega}} \mathfrak{A}$. As usual, a Kripke structure $\mathcal{K}$* satisfies *$\mathfrak{A}$ (denoted $\mathcal{K} \models_{\mathsf{omega}} \mathfrak{A}$) iff all paths $i \in \mathsf{IPath}(\mathcal{K})$ satisfy $\mathfrak{A}$.*

An ω-automaton $\mathfrak{A} := \mathcal{A}_{\exists/\forall}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$ is called total, *iff for all $i, i' \subseteq \mathcal{V} \setminus \mathcal{Q}$, $s \subseteq \mathcal{Q}$ a state $s' \subseteq \mathcal{Q}$ exists such that $i \cup s' \models_{\mathsf{prop}} \mathcal{I}$ and $i \cup s \cup \mathbf{X}i' \cup \mathbf{X}s' \models_{\mathsf{prop}} \mathcal{R}$ holds. $\mathfrak{A}$ is called* deterministic *iff for all i, i', s an unique s' with these properties exists.*

For all input paths there is exactly one run through a deterministic automaton. Thus, the semantics of $\mathcal{A}_{\exists}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$ and $\mathcal{A}_{\forall}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$ coincide for deterministic automata. Therefore, the notation $\mathcal{A}_{\mathrm{det}}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, l)$ is used as well in the deterministic case.

3 Translations

As already mentioned, all operators of the sublanguage SUFL of PSL correspond to RLTL operators. As shown in previous work [26,25] these correspondences lead to a very simple translation procedure from SUFL on infinite words to RLTL.

However, during this translation not only the formula itself but also the input words have to be translated, because in contrast to RLTL, inputs for PSL may contain the special states $\top$ and $\bot$. The translation used considers only *infinite proper words* [17]. An *infinite proper word* over $\mathcal{V}$ is an infinite word $v \in \mathcal{XP}(\mathcal{V})^\omega$ such that $\forall j.\ v^j{=}\top \longrightarrow v^{j+1}{=}\top$ and $\forall j.\ v^j{=}\bot \longrightarrow v^{j+1}{=}\bot$ hold.

The set of all infinite proper words over $\mathcal{V}$ is denoted by $\mathcal{XP}(\mathcal{V})^{\omega^{\top\bot}}$. At first glance, it may seem to be a restriction to consider only proper words, however, this is not the case. Special states are just an auxiliary means used to explain the semantics; they do not occur in practise and proper words are sufficient to explain the semantics.

Theorem 1. *With the definitions of Figure 1, the following are equivalent for all $f \in \mathsf{sufl}_\mathcal{V}$, all infinite proper words $v \in \mathcal{XP}(\mathcal{V})^{\omega^{\top\bot}}$ and all $t, b \notin \mathcal{V}$:*

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

If v does not contain $\top$ and $\bot$, i. e. in case $v \in \mathcal{P}(\mathcal{V})^\omega$, this can be simplified to:

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

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

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

Fig. 1. Translation of SUFL to RLTL

After the translation to RLTL, the formula can easily be translated further to LTL. This translation step is due to [2].

Theorem 2 (Translation of RLTL to LTL). *With the definition of Figure 2, the following holds 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 $t \in \mathbb{N}$:*

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

Obviously, this can be instantiated to:

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

```
function $\mathit{RLTL\_TO\_LTL}(a, r, \Phi)$
    case $\Phi$ of
        $b$                    : return $a \vee (b \wedge \neg r)$;
        $\neg\varphi$                 : return $\neg\mathsf{RLTL\_TO\_LTL}(r, a, \varphi)$;
        $\varphi \wedge \psi$              : return $\mathsf{RLTL\_TO\_LTL}(a, r, \varphi) \wedge \mathsf{RLTL\_TO\_LTL}(a, r, \psi)$;
        $\mathsf{X}\varphi$                 : return $a \vee \Big(\mathsf{X}\big(\mathsf{RLTL\_TO\_LTL}(a, r, \varphi)\big) \wedge \neg r\Big)$;
        $\varphi\ \underline{\mathsf{U}}\ \psi$              : return $\mathsf{RLTL\_TO\_LTL}(a, r, \varphi)\ \underline{\mathsf{U}}\ \mathsf{RLTL\_TO\_LTL}(a, r, \psi)$;
        $\mathsf{ACCEPT}(\varphi, b)$: return $\mathsf{RLTL\_TO\_LTL}(a \vee (b \wedge \neg r), r, \varphi)$;
    end
end
```

Fig. 2. Translation of RLTL to LTL

$\mathsf{IPath}(\mathcal{K}) \subseteq \mathcal{P}(\mathcal{V})^\omega$ holds for all Kripke structures $\mathcal{K}$ over $\mathcal{V}$. This leads to the following corollary.

Corollary 1 (Direct translation of PSL to LTL). *For all $f \in \mathsf{sufl}_\mathcal{V}$ and all Kripke structures $\mathcal{K}$ over $\mathcal{V}$ the following holds:*

$$\mathcal{K} \models_{\mathsf{sufl}} f \Longleftrightarrow \mathcal{K} \models_{\mathsf{ltl}} \mathit{RLTL\_TO\_LTL}(\mathsf{false}, \mathsf{false}, \mathsf{PSL\_TO\_RLTL}(f))$$

It remains to translate LTL to ω-automata. It is well known how this can be achieved [30,5,13,9,12]. In this work, we use an algorithm by Klaus Schneider [22,23] to translate LTL to the symbolic representation of ω-automata introduced above. This algorithm is very similar to the one used in [27] to translate LTL to alternating ω-automata.

Theorem 3 (Translation of LTL to ω-automata). *For all* LTL *formulas $\Phi \in \mathsf{LTL}_\mathcal{V}$ and $(Q, \mathcal{I}, \mathcal{R}, \mathcal{F}, p) := \mathsf{TopProp}_\sigma(\Phi)$, where* TopProp *is defined as in Figure 3, the following holds:*

– *for $\sigma = \mathsf{true}$ and all $v \in \mathcal{P}(\mathcal{V})^\omega$, the following holds:*

$$v \models_{\mathsf{ltl}} \Phi \Longleftrightarrow v \models_{\mathsf{omega}} \mathcal{A}_\exists(Q, \mathcal{I} \wedge p, \mathcal{R}, \bigwedge_{\xi \in \mathcal{F}} \mathsf{GF}\,\xi)$$

– *for $\sigma = \mathsf{false}$ and all $v \in \mathcal{P}(\mathcal{V})^\omega$, the following holds:*

$$v \models_{\mathsf{ltl}} \neg\Phi \Longleftrightarrow v \models_{\mathsf{omega}} \mathcal{A}_\exists(Q, \mathcal{I} \wedge \neg p, \mathcal{R}, \bigwedge_{\xi \in \mathcal{F}} \mathsf{GF}\,\xi)$$

```
function $\mathsf{TopProp}_\sigma(\Phi)$
    case $\Phi$ of
        $p$              : return $(\{\}, \mathsf{true}, \mathsf{true}, \{\}, p)$;
        $\neg\varphi$    : $(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, p_\varphi) := \mathsf{TopProp}_{\neg\sigma}(\varphi)$;
                           return $(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, \neg p_\varphi)$;
        $\varphi \wedge \psi$ : return $\mathsf{TopProp}_\sigma(\varphi) \times \mathsf{TopProp}_\sigma(\psi)$;
        $\mathsf{X}\varphi$   : $(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, p_\varphi) := \mathsf{TopProp}_\sigma(\varphi)$;
                           $q := \mathsf{new\_var}$;
                           return $(Q_\varphi \cup \{q\}, \mathcal{I}_\varphi, \mathcal{R}_\varphi \wedge (q \leftrightarrow \mathbf{X} p_\varphi), \mathcal{F}_\varphi, q)$;
        $\varphi \, \underline{\mathsf{U}} \, \psi$ : $(Q_\Phi, \mathcal{I}_\Phi, \mathcal{R}_\Phi, \mathcal{F}_\Phi, p_\varphi \wedge p_\psi) := \mathsf{TopProp}_\sigma(\varphi) \times \mathsf{TopProp}_\sigma(\psi)$;
                           $q := \mathsf{new\_var}$;
                           $\mathcal{R}_Q := q \leftrightarrow (p_\psi \vee (p_\varphi \wedge \mathtt{X} q))$;
                           $\mathcal{F}_Q :=$ if $\sigma$ then $\{q \vee p_\psi\}$ else $\{\}$;
                           return $(Q_\Phi \cup \{q\}, \mathcal{I}_\Phi, \mathcal{R}_\Phi \wedge \mathcal{R}_Q, \mathcal{F}_\varphi \cup \mathcal{F}_Q, q)$;
    end
end
```

Fig. 3. Translation of LTL to ω-automata [22,23]

4 Infrastructure

The HOL System [14,16] is an interactive theorem prover for higher order logic. In this work the HOL4 implementation is used. The version of higher order logic used in HOL is predicate calculus with terms from the typed lambda calculus [8]. The interactive front-end of HOL is the functional programming language ML, in which terms and theorems of the logic, proof strategies and logical theories are implemented. This language is used to implement the translations described here and also to interface to SMV.

In earlier work, Gordon deeply embedded PSL in HOL [15], Schneider created a shallow embedding of symbolically represented ω-automata and a shallow embedding of LTL and verified a translation between these two embeddings [24]. Also, we have previously deeply embedded RLTL, LTL and symbolically represented ω-automata, and verified the translations described in Sec. 3 [25,26]. However, no automatic translations or other parts that could be used for tools existed.

In this work, we describe such tools. In particular, we have implemented validating compilers for all the translations described here, i. e. we have implemented ML-functions that translate an LTL term to an ω-automaton and also produce a correctness proof of the generated automaton. Thus, possible bugs in these implementations may only lead to exceptions and failing translations, but no wrong results can be produced. In addition, we have implemented validating compilers to convert model checking problems for SUFL and LTL to check the existence of fair paths through a Kripke structure. For example, we can translate the check $\mathcal{K} \models_{\mathsf{sufl}} f$ to a Kripke structure M and a set of propositional formulas fc such that $\mathcal{K} \models_{\mathsf{sufl}} f \iff \mathsf{IPath}_{\mathsf{fair}}(M, \mathrm{fc}) = \emptyset$ holds. This emptiness check can be handled by CTL model checkers that can handle fairness. In this work, we use the model checker SMV [19] and reuse an interface already developed in previous work [24]. However, interfaces to other model checkers can easily be added.

As a result of this work, we can perform model checking for SUFL using HOL and SMV. Assuming that SMV and HOL are correct, we have high confidence that the whole tool is correct, since only the interface between HOL and SMV is not verified and this interface is very small and simple.

Provably correct SUFL model checking is interesting in its own right as SUFL is a significant subset of PSL and PSL is difficult to model check. PSL is a complex language, so errors in designing and implementing model checking procedures for it are potentially very easy to make. However, the main purpose of the work reported here is to create a library of theorems and ML functions as a basis for building special purpose tools. One example of such an tool is described in the next section. This enables implementers of PSL tools to validate their code on concrete examples with respect to the Version 1.1 PSL semantics.

5 Application: Validating a Translator from PSL to CTL

Our tool aims to validate how IBM's model checker RuleBase [4] handles PSL. RuleBase checks if a Kripke structure $\mathcal{K}$ satisfies a PSL specification f, by translating the specification f to a total transition system $\mathcal{T} = (\mathcal{Q}, \mathcal{I}, \mathcal{R})$ and a CTL formula of the form $\mathsf{AG}\,p$ with propositional p. This translation is a blackbox to us. Then $\mathcal{K} \,||\, \mathcal{T} \models_{\mathsf{CTL}} \mathsf{AG}\,p$ is checked. Neither CTL semantics nor this combination of $\mathcal{K}$ and $\mathcal{T}$ are explained here, but note that $\mathcal{K} \,||\, \mathcal{T} \models_{\mathsf{CTL}} \mathsf{AG}\,p$ is equivalent to $\mathcal{K} \models \mathcal{A}_{\forall}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, \mathsf{G}\,p)$. Thus, given $f, \mathcal{Q}, \mathcal{I}, \mathcal{R}$ and p one would like to automatically prove

$$\forall \mathcal{K}.\ \mathcal{K} \models_{\mathsf{omega}} \mathcal{A}_{\forall}(\mathcal{Q}, \mathcal{I}, \mathcal{R}, \mathsf{G}\,p) \Longleftrightarrow \mathcal{K} \models_{\mathsf{sufl}} f.$$

We are able to solve this problem for all SUFL formulas f with the library we have developed. Moreover, clock operators can also be handled, since they can be considered as syntactic sugar and eliminated by rewrite rules [1]. Thus only regular expressions can not be handled. However, we have a preprocessing step that tries to eliminate regular expressions by rewriting them to SUFL formulas. Regular expression strictly increase the expressiveness of FL [28,29,25]. Thus, we can not eliminate all of them. But luckily we can eliminate most of the regular expressions occurring in practise.

It is of course vital to anybody using RuleBase with PSL specifications, that the translation to CTL model checking is sound. However, we think our work should be of interest to implementers of other model checking tools also. Note that neither IBM's translation nor our tool can handle full FL. Nevertheless, the intersection between the subsets that our tool can handle and RuleBase can handle is big enough to be interesting.

To implement a tool to solve the translation validation problem using our library, formal translation (implemented by theorem proving) is used to convert a PSL formula f to an equivalent LTL formula l. Then the quantification over the models $\mathcal{K}$ is replaced by quantification over all paths i, which is equivalent for this problem. $\mathcal{A}_\forall(\mathcal{Q},\mathcal{I},\mathcal{R},\mathsf{G}\,p)$ is then translated to a deterministic automaton $\mathcal{A}_{\mathsf{det}}(\mathcal{Q}_{\mathsf{det}},\mathcal{I}_{\mathsf{det}},\mathcal{R}_{\mathsf{det}},\mathsf{G}\,p_{\mathsf{det}})$, which is possible because the input automaton is total [23]. Thus, the original problem is equivalent to

$$\forall i.\ i \models_{\mathsf{omega}} \mathcal{A}_\forall(\mathcal{Q}_{\mathsf{det}},\mathcal{I}_{\mathsf{det}},\mathcal{R}_{\mathsf{det}},\mathsf{G}\,p_{\mathsf{det}}) \Longleftrightarrow i \models_{\mathsf{ltl}} l.$$

This is in turn equivalent to $(\mathcal{I}_{\mathsf{det}},\mathcal{R}_{\mathsf{det}}) \models_{\mathsf{ltl}} \mathsf{G}\,p_{\mathsf{det}} \leftrightarrow l$. Thus, the library can be used to translate the original problem to a LTL model checking problem, which can be solved using the techniques described in Section 4. Moreover, all steps needed for this translation to a LTL model checking problem are formally verified in HOL. Therefore, we have an automatic tool to prove the correctness of the translation for concrete examples.

We have validated several examples provided by IBM. Most of these examples can be verified in a few minutes, some take several hours. However, the determinisation of the input automaton leads to an exponential blowup. Thus, small PSL formulas may lead to huge model checking problems. However, we have been able to show that the tool we developed is able to handle non toy examples. Moreover, we have been able to detect an error in the translation of the ABORT operator under clocks using it (though, unknown to us, this problem was already known to the RuleBase developers).

6 Conclusions and Future Work

We have developed a library that allows us to handle a significant subset of PSL using HOL and SMV. There are theorems about PSL and especially about translations between PSL, LTL and ω-automata, and also ML-functions that solve common problems automatically. Model checking problems of SUFL and LTL can be tackled using these automatic procedures. However, the main purpose of the library is to provide a basis to build tools that can handle special PSL problems.

We used the library to validate the handling of PSL by RuleBase. We were able to show that our tool could handle interesting examples. Moreover, we were even able to detect an error in IBM's procedure.

A lot of implementation details could be improved. However, the main challenge will be to extend the subset of PSL. Adding regular expression strictly increases the expressiveness such that the resulting subset of PSL can no longer be translated to LTL [28,29,25]. However, it is possible to translate PSL directly to ω-automata. There is an approach by Bustan, Fisman and Havlicek [6] which translates PSL to alternating ω-automata. Another approach by Zaks and Pnueli [21] translates PSL to symbolically represented automata. It would be interesting to use this work to verify a direct translation to symbolically described nondeterministic ω-automata using HOL.

Acknowledgements

Thomas Tuerk's visit to Cambridge is partly supported by an IBM Faculty Award to Mike Gordon and partly by EPSRC grant GR/T06315/01. The problem of checking conformance of the PSL-to-CTL translation used by RuleBase with the semantics of PSL in the official standard was originally formulated by Cindy Eisner and Dana Fisman of IBM, and we thank them for much helpful advice and for supplying examples.

References

1. Accellera. Property specification language reference manual, version 1.1. http://www.eda.org, June 2004.
2. 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.
3. 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.
4. Beer, I., Ben-David, S., Eisner, C., Geist, D., Gluhovsky, L., Heyman, T., Landver, A., Paanah, P., Rodeh, Y., Ronin, G., and Wolfsthal, Y. RuleBase: Model checking at IBM. In *Conference on Computer Aided Verification (CAV)* (Haifa, Israel, 1997), O. Grumberg, Ed., vol. 1254 of *LNCS*, Springer, pp. 480–483.
5. Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L. Symbolic model checking: 10^{20} states and beyond. In *Symposium on Logic in Computer Science (LICS)* (Washington, D.C., June 1990), IEEE Computer Society, pp. 1–33.
6. Bustan, D., Fisman, D., and Havlicek, J. Automata construction for PSL. Technical Report MCS05- 04, The Weizmann Institute of Science, Israel, 2005.
7. Büchi, J. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology and Philosophy of Science* (Stanford, CA, 1960), E. Nagel, Ed., Stanford University Press, pp. 1–12.
8. Church, A. A formulation of the simple theory of types. *Journal of Symbolic Logic 5* (1940), 56–68.
9. 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.
10. DeepChip survey on assertions. http://www.deepchip.com/items/dvcon04-06.html, June 2004.
11. Emerson, E., and Clarke, E. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming 2*, 3 (1982), 241–266.
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 *Symposium on Protocol Specification, Testing, and Verification (PSTV)* (Warsaw, June 1995), North Holland.
14. Gordon, M. HOL: A machine oriented formulation of higher order logic. Tech. Rep. 68, Computer Laboratory, University of Cambridge, May 1985.
15. Gordon, M. PSL semantics in higher order logic. In *Workshop on Designing Correct Circuits (DCC)* (Barcelona, Spain, 2004).

16. GORDON, M., AND MELHAM, T. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.
17. 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.
18. 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.
19. MCMILLAN, K. *Symbolic Model Checking.* Kluwer, Norwell Massachusetts, 1993.
20. 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.
21. PNUELI, A., AND ZAKS, A. PSL model checking and run-time verification via testers. In *FM* (2006), J. Misra, T. Nipkow, and E. Sekerinski, Eds., vol. 4085 of *Lecture Notes in Computer Science*, Springer, pp. 573–586.
22. SCHNEIDER, K. Improving automata generation for linear temporal logic by considering the automata hierarchy. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)* (Havanna, Cuba, 2001), vol. 2250 of *LNAI*, Springer, pp. 39–54.
23. SCHNEIDER, K. *Verification of Reactive Systems – Formal Methods and Algorithms.* Texts in Theoretical Computer Science (EATCS Series). Springer, 2003.
24. 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.
25. TUERK, T. A hierarchy for Accellera's property specification language. Master's thesis, University of Kaiserslautern, Department of Computer Science, 2005.
26. TUERK, T., AND SCHNEIDER, K. From PSL to LTL: A formal validation in HOL. In *International Conference on Theorem Proving in Higher Order Logics (TPHOL)* (Oxford, UK, 2005), J. Hurd and T. Melham, Eds., vol. 3603 of *LNCS*, Springer, pp. 342–357.
27. VARDI, M. Branching vs. linear time: Final showdown. In *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (Genova, Italy, 2001), T. Margaria and W. Yi, Eds., vol. 2031 of *LNCS*, Springer, pp. 1–22.
28. 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.
29. WOLPER, P. Temporal logic can be more expressive. *Information and Control 56*, 1-2 (1983), 72–99.
30. 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.