From PSL to LTL: A Formal Validation in HOL

Thomas Tuerk Klaus Schneider

University of Kaiserslautern

August 23th, 2005

Overview

Why *PSL*?

- model and equivalence checking are state of the art in modern hardware circuit design
- standardised hardware description languages like *VHDL* and *Verilog* are widespread
- but specification languages were not standardised
- many different formalisms for specifications exists like
  - *LTL*
  - *CTL*
  - *CTL**
  - ω-automata
  - monadic second order logics
  - μ-calculus
- Accellera's Property Specification Language (*PSL*) is a standardised industrial-strength property specification language
  - Version 1.0: April 2003
  - Version 1.1: June 2004

Why *PSL*?

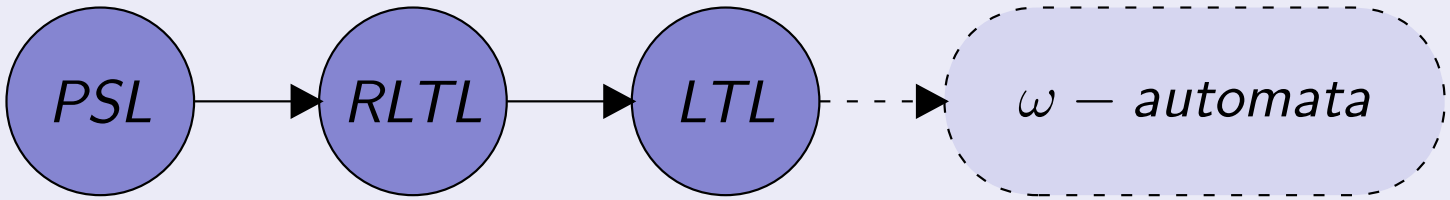
- model and equivalence checking are state of the art in modern hardware circuit design
- standardised hardware description languages like *VHDL* and *Verilog* are widespread
- but specification languages were not standardised
- many different formalisms for specifications exists like
  - *LTL*
  - *CTL*
  - *CTL**
  - ω-automata
  - monadic second order logics
  - μ-calculus
- Accellera's Property Specification Language (*PSL*) is a standardised industrial-strength property specification language
  - Version 1.0: April 2003
  - Version 1.1: June 2004

Problem Description

- model checking of *PSL* turned out to be quite difficult
- *PSL* is a complex language
  - semantics often tricky
  - many special cases
- especially the abort operator is interesting
- Armoni, Bustan, Kupferman and Vardi introduced *RLTL*:
  - abort in *PSL* 1.0 leads to non-elementary blowup

  $\Rightarrow$ abort changed according to *RLTL* in *PSL* 1.1

Translation of *PSL* to *LTL*



- unsurprisingly a significant subset of *PSL* can be translated to *RLTL*
- translations of *RLTL* to *LTL* and of *LTL* to ω-automata are well known
- however, the semantics of *PSL* is quite tricky
- correctness proof of the translation technical, but tricky

$\Rightarrow$ *HOL* very useful

Linear Temporal Logic (*LTL*)

- introduced by Pnueli in 1977
- essentially consists of propositional logic enriched with temporal operators **X** and $\underline{\mathbf{U}}$
- for $w : \mathbb{N} \to 2^{\mathcal{P}}$ the semantics is given by:
  - the usual semantics of propositional operators
  - $w \models p$ iff $p \in w^0$
  - $w \models \mathbf{X}\,\varphi$ iff $w^{1..} \models \varphi$
  - $w \models \varphi\,\underline{\mathbf{U}}\,\psi$ iff φ holds on w until ψ holds and ψ eventually holds
- additional operators are added as syntactic sugar, e. g. **G**, **F**

Linear Temporal Logic (*LTL*) II

Example

The *LTL* formula

$$\mathbf{G}(\text{req} \rightarrow \mathbf{F}\,\text{ack})$$

specifies, that every request (req) has to be followed by an acknowledge (ack).

Reset Linear Temporal Logic (*RLTL*)

- *RLTL* is an extension of *LTL* with reset operators called **ACCEPT** and **REJECT**
- *RLTL* is as expressive as *LTL*
- translation of *RLTL* to *LTL* known

Example

$$\mathbf{G}\Big((\text{req} \rightarrow \mathbf{F}\,\text{ack})\ \mathbf{ACCEPT}\ \text{cancel}\Big)$$

specifies, that every request (req) has to be followed by an acknowledge (ack), unless a cancellation (cancel) occurs.

Reset Linear Temporal Logic (*RLTL*) II

- φ **REJECT** $b := \neg(\neg\varphi$ **ACCEPT** $b)$
- the formal semantics of *RLTL* use special acceptance / rejection conditions to model the **ACCEPT**/**REJECT** statements occuring in the context:
  - start with $\langle w, \text{false}, \text{false}\rangle$
  - $\langle w, a, r\rangle \models \varphi$ **ACCEPT** b iff $\langle w, a \vee (b \wedge \neg r), r\rangle \models \varphi$
  - $\langle w, a, r\rangle \models \neg\varphi$ iff $\langle w, r, a\rangle \not\models \varphi$
  - $\langle w, a, r\rangle \models p$ iff $w^0 \models a \vee (\neg r \wedge p)$

Accellera's Property Specification Language (*PSL*)

- *PSL* is an industrial strength property specification language
- *PSL* is based on IBM's sugar language
- a significant subset of *PSL* called *SUFL* consists of
  - linear temporal logic operators
  - a reset operator called **ABORT**
- semantics defined using special states $\top, \bot$
- instead of using acceptance / rejection conditions the input path is modified:

  $w \models \varphi$ **ABORT** b iff
  $w \models \varphi$ or $(\exists j.\ w^j \models b$ and $w^{0..j-1}\top^\omega \models \varphi)$

Translation of *SUFL* to *RLTL*

- translation quite easy: replace every *PSL* operator with the corresponding *RLTL* operator, i. e. essentially replace **ABORT** with **ACCEPT**
- however, correctness proof tricky
- $\top, \bot$ have to be mapped to acceptance / rejection conditions
- a lot of technical problems occur

Problems with the translation

- $\langle w, a, r\rangle \models_{RLTL} \mathbf{X}\varphi$ iff
  $w^0 \models a$ or $(w^0 \not\models r$ and $\langle w^{1..}, a, r\rangle \models_{RLTL} \varphi)$
- $w \models_{PSL} \mathbf{X}\varphi$ iff $w^{1..} \models_{PSL} \varphi$

$\Rightarrow$ only proper words are considered

$\Rightarrow$ $\top^\omega \models \varphi$ and $\bot^\omega \not\models \varphi$ had to be shown

- $\langle w, a_1 \vee a_2, r\rangle \models \varphi \iff \big(\langle w, a_1, r\rangle \models \varphi \ \vee \ \langle w, a_2, r\rangle \models \varphi\big)$
- invariant: a and r never hold at the same point of time
- $\neg_{\text{prop}}$ and $\neg_{\text{LTL}}$ have to be distinguished

Work done in *HOL*

- we used Mike Gordon's deep embedding of *PSL*
- we deeply embedded:
  - *RLTL*
  - *LTL*
  - automaton formulas, a symbolic representation of nondeterministic ω-automata
- we formally verified:
  - the translation of *PSL* to *RLTL*
  - a translation of *RLTL* to *LTL*
  - two translations of *LTL* to automaton formulas
- we discovered a small bug in Mike Gordon's deep embedding of *PSL*

Conclusions

Summary

- *SUFL* is a significant subset of *PSL*
- we translated *SUFL* to *RLTL*
- we discovered a small bug in the deep embedding of *PSL*
- we deeply embedded *LTL*, *RLTL* and automaton formulas

Future Work

- translate *PSL* directly to ω-automata to incorporate *SEREs*
- translate *PSL* on finite paths to finite automata on finite words
- deeply embed alternating automata

Scale of *HOL* theories

Theory	**LOC**
GeneralLemmataScript.sml	341
TemporalModelScript.sml	1098
TemporalModel_LemmataScript.sml	234
LTLScript.sml	422
LTL_LemmataScript.sml	565
ResetLTLScript.sml	536
ResetLTL_LemmataScript.sml	605
PSL_DefinitionsScript.sml	1140
PSL_LemmataScript.sml	1174
Omega_AutomataScript.sml	1539
Omega_Automata_LemmataScript.sml	2118
PSLToRLTLScript.sml	870
LTLToOmegaScript.sml	1971
LTLToOmegaOptScript.sml	2125
other	533
	16650

Accellera's Property Specification Language (*PSL*)

- *PSL* is an industrial strength property specification language
- *PSL* is based on IBM's sugar language
- *PSL* consists of different layers and different flavours
- here only the temporal layer is considered
- the temporal layer consists of
  - the *Foundation Language*
  - the *Optional Branching Extension*, which is essentially *CTL*

PSL Foundation Language (*FL*)

the Foundation Language of *PSL* consists of:

- support for finite and infinite paths
- linear temporal logic operators
- a clocking operator
- Sequential Extended Regular Expressions (*SEREs*)
- a reset operator called **ABORT**
  - semantics defined using special states $\top, \bot$ instead of acceptance / rejection conditions

we translate only *SERE*-free unclocked *FL* (*SUFL*)

Small Bug in the Deep Embedding of *PSL*

- formal semantics of **ABORT**:

  $w \models_{PSL} \varphi$ **ABORT** b iff

  $w \models_{PSL} \varphi$ or

  $(\exists j.j < |w| \text{ s.t. } w^j \models b \text{ and } w^{0..j-1}\top^\omega \models_{PSL} \varphi)$

- this had been literally implemented in *HOL*
- consider special case $j = 0$
  - $0 - 1$ is evaluated to -1 by the formal semantics of *PSL*
  - *HOL* embedding uses the type *num*
  - thus $0 - 1$ was evaluated to 0 by the *HOL* embedding