Model Checking PSL using HOL and SMV

Thomas Tuerk Klaus Schneider Mike Gordon

University of Kaiserslautern

University of Cambridge

HVC 2006, October 23th, 2006

Overview

Accellera's Property Specification Language (*PSL*)

- *PSL* is an industrial standard property specification language (IEEE P1850)
- it contains
  - *CTL*
  - *LTL*
  - semantics on finite and infinite paths
  - regular expressions
  - special operators for resets and clocks
  - a lot of syntactic sugar
- semantics are quite tricky
- there are many special cases

The HOL Theorem Prover

- developed by Mike Gordon's group in Cambridge
- the first version *HOL*88 has been released in 1988
- latest release in January 2006 called *HOL4 Kananaskis 3*
- implements classical **H**igher-**O**rder **L**ogic
- evolved from the Edinburgh LCF project, so has a small logical kernel to ensure soundness
- good choice for tool development because of *ML*
- links to external proof tools, either as oracles (e.g. *SMV*) or by translating their proofs

Why use *HOL* for *PSL* problems?

Drawbacks

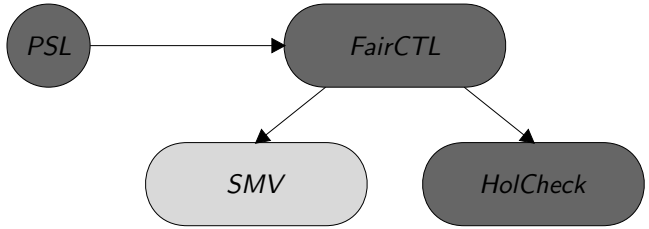
- even modelling simple formalisms or non optimised algorithms in *HOL* takes a long time
- many things obvious to human beings are difficult to formalise
- resulting tools are very slow

... but

- *PSL* semantics are tricky
- it's very easy to develop algorithms that do not consider all special cases
- correct implementations may be even more difficult to produce
- results of our *HOL* tools are **proved** to be correct

Our Goals

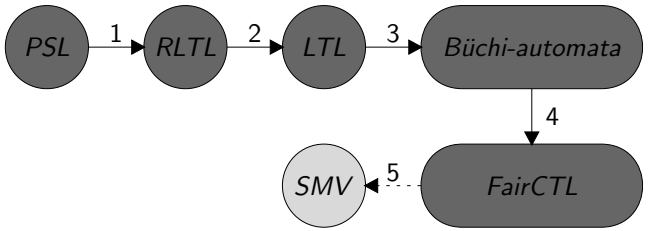
- develop a framework for *PSL* model checking
- framework should be usable for
  - playing with *PSL* semantics
  - exploration of new ideas, algorithms
  - verification of *PSL* model checking algorithms
  - validation of existing tools
- as part of this framework (partly) verified *PSL* model checking should be provided



Embedded Formalisms

- deep embeddings of important formalisms for model checking:
  - *PSL* with help from C. Eisner and D. Fisman
  - *LTL*
  - *RLTL*
  - *CTL*, *FairCTL*, *CTL**
  - Kripke structures, symbolically and explicitly represented
  - automaton formulae, i. e. symbolically represented nondeterministic and universal ω-automata
  - explicitly represented nondeterministic and universal ω-automata
  - alternating ω-automata

Verified Translations



2) by Armoni, Bustan, Kupferman and Vardi (TACAS 2003)
3) by Schneider

Restriction

regular expressions can not be handled

HOL implementation

- main focus of this work is on implementation
- just minor additions to existing algorithms
- but implementation in *HOL* means a very formal, machine-readable proof
- such proofs are interesting
- most proofs exist in two versions
  - a clean version used for proving additional theorems
  - a specialised, difficult to read version for automation
- *ML* functions allow automated *PSL* model checking using the external model checker *SMV*

Validation of *RuleBase*'s method to handle *PSL* II

- given *PSL* specification f
- use *RuleBase* to translate f to ω-automaton $\mathfrak{A}_1$
- use our framework to translate (provably correct) f to $\mathfrak{A}_2$
- check that $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are equivalent
- therefore, translate this equivalence check inside *HOL* to a *FairCTL* model checking problem
- use a model checker like *SMV* to check
- we detected an incorrectness (unknown to us, but known to IBM) in the implementation of clocked aborts

Validation of *RuleBase*'s method to handle *PSL*

- IBM's *RuleBase* is essentially a *FairCTL* model checker
- to handle a *PSL* specification, it is translated to
  - a satellite automaton A
  - a CTL formula of the form AG p,
- thus *PSL* specifications are translated to ω-automata
- this preprocessing step is a blackbox to us
- is this preprocession step correct?

Conclusions

- we implemented a framework for model checking *PSL* in *HOL*
- (partly) verified *PSL* model checking can be done using *HOL* and *SMV*
- this framework can be used for industrial applications
- automated model checking is limited to *PSL* without regular expressions at the moment

Future Work

- extend the subset of *PSL* that can be handled to full *PSL* using e. g. ideas from *PSL Model Checking and Run-time Verification via Testers* by A. Zaks and A. Pnueli
- build an interface to *HolCheck* to allow fully verified *PSL* model checking

A glimpse of *HOL*

Definitions

```
val fl_def =
 Hol_datatype
  `fl = F_STRONG_BOOL  of 'a bexp
      ...
      | F_CLOCK        of fl # 'a bexp
      | F_SUFFIX_IMP   of 'a sere # fl`;

val UF_SEM_def =
 Define
   `(UF_SEM v (F_NOT f) = ~(UF_SEM (COMPLEMENT v) f)) /\
    (UF_SEM v (F_AND(f1,f2)) = UF_SEM v f1 / UF_SEM v f2) /\
    (UF_SEM v (F_STRONG_BOOL b) = (LENGTH v > 0) / B_SEM (ELEM v 0) b) /\
    (UF_SEM v (F_WEAK_BOOL b) = (LENGTH v = XNUM 0)  B_SEM (ELEM v 0) b) ...`
```

A glimpse of *HOL* II

Theorems

```
val F_CLOCK_COMP_CORRECT =
 store_thm
  ("F_CLOCK_COMP_CORRECT",
   ``!f v c. F_SEM v c f = UF_SEM v (F_CLOCK_COMP c f)``,
   INDUCT_THEN fl_induct ASSUME_TAC
     ...)
```

ML Functions

```
fun ltl2omega fast neg l =
   let
      val (typeString, ltl_type) = (dest_type (type_of l));
      val _ = if (typeString = ltl") then T else raise NoLTLTerm;
      ...

fun modelCheckFairEmptyness ks_term thm =
  let
    val file_st = TextIO.openOut(!model_check_temp_file);
    val vars = fair_empty_ks2smv_string ks_term file_st;
    val _ = TextIO.closeOut file_st;
    val res = SMV_RUN_FILE (!model_check_temp_file);
  in
    ...
```