PSL in HolCheck

Thomas Tuerk

February 1st, 2006

Overview

HolCheck

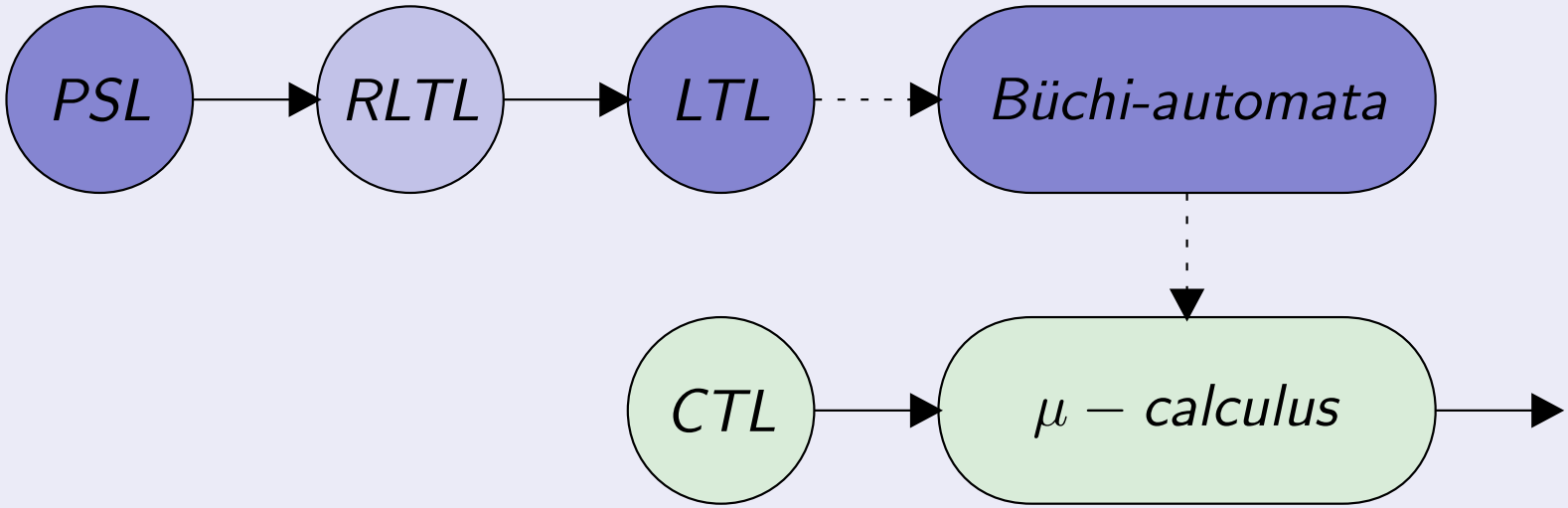
- symbolic model checker embedded in the HOL theorem prover
- all steps of the algorithm are proved in HOL
- based on *HolBdd* developed by Mike Gordon
- developed at the ARG, mainly by Hasan Amjad
- supports μ-calculus and CTL

PSL

- 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
- *PSL* is quite similar to *ForSpec* Temporal Logic (*FTL*) presented by Moshe Vardi last week

Integration of *PSL* into *HolCheck*

- in my diploma thesis I formally validated a translation of a significant subset of *PSL* to generalised Büchi automata
- the emptiness problem of generalised Büchi automata remains
- this problem can be easily translated to a μ-calculus formula
- *HolCheck* is able to handle μ-calculus formulas



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
- these operators are used to model hardware resets
- introduced by Armoni, Bustan, Kupferman and Vardi (TACAS 2003):
  - abort in *PSL* 1.0 leads to non-elementary blowup

  $\Rightarrow$ abort changed according to *RLTL* in *PSL* 1.1
- *RLTL* is as expressive as *LTL*
- translation of *RLTL* to *LTL* known
- unsurprisingly a significant subset of *PSL* 1.1 can be translated to *RLTL*

Reset Linear Temporal Logic (*RLTL*) II

Example

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

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

Accellera's Property Specification Language (*PSL*)

- *PSL* is an industrial strength property specification language
- *PSL* is based on IBM's sugar and Intel's *FTL* 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*

Accellera's Property Specification Language (*PSL*) II

- the foundation language (*FL*) consists of
  - linear temporal logic operators
  - reset operators
  - clocking operators
  - regular expressions (*SEREs*)
  - a lot of syntactic sugar
- the semantics of *FL* consider finite and infinite paths

- we consider a significant subset of *FL* called *SUFL*, that consists of
  - linear temporal logic operators
  - reset operators
- we consider only infinite paths
- i. e. we consider *FL* without regular expressions for model checking

Accellera's Property Specification Language (*PSL*) II

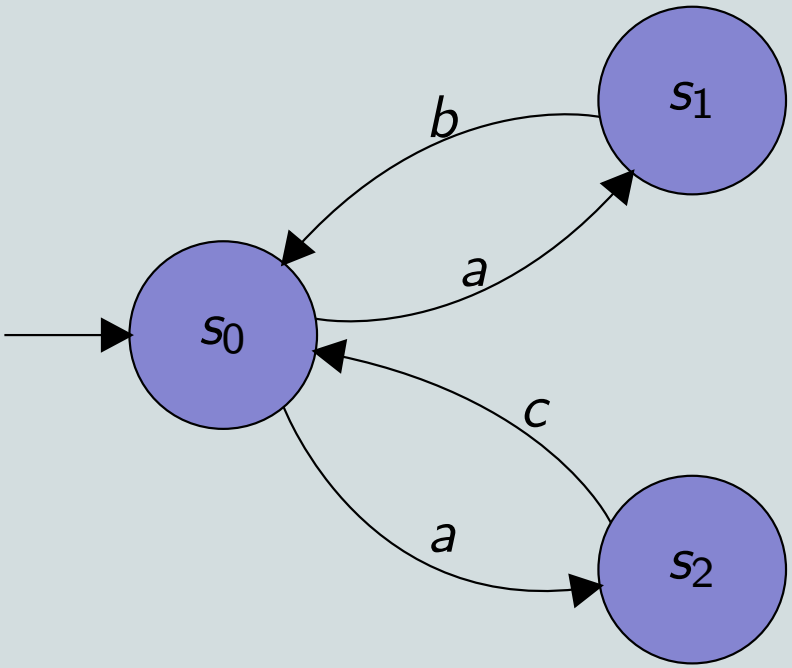
- the foundation language (*FL*) consists of
  - linear temporal logic operators
  - reset operators
  - clocking operators
  - regular expressions (*SEREs*)
  - a lot of syntactic sugar
- the semantics of *FL* consider finite and infinite paths
- we consider a significant subset of *FL* called *SUFL*, that consists of
  - linear temporal logic operators
  - reset operators
- we consider only infinite paths
- i. e. we consider *FL* without regular expressions for model checking

Generalised Büchi Automata

- generalised Büchi automata are finite automata on infinite words
- as the input is infinite, there is no last state of a run
- thus, the acceptance condition has to be defined somehow different
- a input is accepted by a generalised Büchi automaton, iff there is a run that visits some sets of states infinitely often
- these sets of states are called *fairness constraints*

Generalised Büchi Automata II

Example



Initial state: s_0
Fairness Constraints: $\{s_1\}, \{s_2\}$

Then the given generalised Büchi automaton accepts all runs that visit s_1 and s_2 infinitely often

Translation of *SUFL* to *RLTL*

- translation quite easy: replace every *PSL* operator with the corresponding *RLTL* operator
- however, correctness proof tricky
- the semantics of *SUFL* is given using special states $\top$, $\bot$
- in contrast *RLTL* uses acceptance / rejection conditions
- these concepts have to be mapped to each other
- a lot of technical problems occur

Translation of *RLTL* to *Generalised Büchi Automata*

Translation of *RLTL* to *LTL*

- translation given by Armoni, Bustan, Kupferman and Vardi
- consists of simple rewrite rules
- correctness proof straightforward

Translation of *LTL* to Generalised Büchi Automata

- translation is well known
- we use a symbolic representation of ω-automata
- translation can be done in linear time with respect to the size of the *LTL* formula

Overall Model Checking Procedure

- given a kripke structure M and a *SUFL* formula f the problem is to check whether all paths through M satisfy f
- translate $\neg f$ to a generalised Büchi automaton A
- build the product $M \times A$ of M and A
- check whether there exists a fair path through $M \times A$
- this emptiness check can be expressed in μ-calculus or in *FairCTL*
- use *HolCheck* to evaluate this μ-calculus formula

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*
  - a translations of *LTL* to automaton formulas

Current Work

- refactoring
- optimising the translation of *LTL* to automaton formulas
- translation of generalised Büchi automata to μ-calculus
- implementing interfaces to *HolCheck*

Possible Future Work

- adding *FairCTL*, *CTL** and ω-automata as input languages to *HolCheck*
- extending the subset of *PSL* to full *FL*
- we are already able to translate *LTL* safety properties to alternation free μ-calculus
- add a specialised translation of *LTL* liveness properties to alternation free μ-calculus
- ...

Conclusions

- *PSL* is an important specification language
- *SUFL* is a significant subset of *PSL*
- we translated *SUFL* to *RLTL* and further to *LTL* and generalised Büchi automata
- we deeply embedded *LTL*, *RLTL* and automaton formulas
- we will soon be able to use *SUFL* and *LTL* as input languages for *HolCheck*