Efficient Model Checking of PSL Safety Properties

Tuomas Launiainen, Keijo Heljanko, Tommi Junntila

April 3, 2011
Introduction

Model checking safety properties

Results

Conclusions
Motivation

- **Safety properties** are an important subclass of temporal properties. Intuitively they are properties whose violation is of the type “something bad happened” as opposed to “something good did not happen”.

- **Model checking safety properties**, especially with BDDs, can be done more efficiently than general properties (Hardin et al. Formal Methods in System Design vol. 18, no. 2, 2001).

- **Our approach** handles PSL safety properties more efficiently than a general purpose checker.
Overview

- Known way of dealing with safety properties
- **Observer** is a finite state automaton that accepts counter-examples
PSL (Property Specification Language) is an industry standard language for specifying temporal properties, i.e. statements about values (or signals) that change over time.

**PSL operators**

<table>
<thead>
<tr>
<th>Boolean connectives:</th>
<th>G</th>
<th>globally</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>finally</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td>until</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>releases</td>
</tr>
<tr>
<td>Sequential regular expressions:</td>
<td>catenation, Kleene star, etc.</td>
<td></td>
</tr>
</tbody>
</table>

| →  | tail implication |
|    |                   |
| ◊→ | tail conjunction |
Transducers

- **Transducers** are a variant of **finite state automata**; their state is represented by a set of **boolean variables**.
- Their transition relation is symbolic.
- Transducers also have initial and final states, but the final states cannot be thought of as accepting.
- In addition to state variables, transducers have **input variables**. One of the state variables is the **output variable**, through which accepting is done.
- Transducers can be **combined** by plugging the output variable of one to an input variable of another.
Transducer example

Transducer for the formula $X \ i$ ($q$ is the output variable and the only state variable):

All states are initial, $\emptyset$ and $\{i\}$ are final.
The tail implication $r \rightsquigarrow \phi$ states that whenever a match is found for $r$, $\phi$ must hold.

Example: $\text{open; read}^*; \text{close} \rightsquigarrow X \text{ G stopped}$

- open read close stopped
- open read fail deadlock
Transducer for tail implication

The tail implication $r \mapsto \phi$ states that whenever a match for $r$ is found, $\phi$ must hold. A transducer for the formula can be constructed in the following way:

- An automaton is constructed for $r$.
- Multiple copies are simulated in the transducer.
- When a simulated copy reaches an accepting state, $\phi$ must hold.
- The states of the automaton for $r$ are the state variables. The variable corresponding to the initial state is the output variable.
Transducer for tail implication example

Example formula: open; read*; close $\mapsto X \ G \ stopped$
Results

Results with real life PSL safety formulas with random-generated parts:

![Graph showing results comparison between BDD and SBMC implementations.](image)
Results with real life models and random-generated safety formulas:

- BDD implementation from NuSMV 2.4.3 with observer
- True
- False

- SBMC implementation from NuSMV 2.4.3 with observer
- True
- False
Conclusion

- **Safety properties** are an important subclass of specifications that can be dealt with more efficiently than general properties.
- **PSL** is an industry standard specification language whose safety properties have been specially dealt with only if they are written in the syntactically restricted safety simple subset.
- Our approach is an efficient way to deal with PSL safety properties.
- Our implementation is freely available and works with the open source model checking tool NuSMV.
References