Symbolic model checking APSL

Wanwei LIU, Ji WANG, Huowang CHEN, Xiaodong MA, Zhaofei WANG
National Laboratory for Parallel and Distributed Processing, School of Computer,
National University of Defense Technology, Changsha 410073, China

© Higher Education Press and Springer-Verlag 2009

Abstract Property specification language (PSL) is a specification language which has been accepted as an industrial standard. In PSL, SEREs are used as additional formula constructs. In this paper, we present a variant of PSL, namely APSL, which replaces SEREs with finite automata. APSL and PSL are of the exactly same expressiveness. Then, we extend the LTL symbolic model checking algorithm to that of APSL, and then present a tableau based APSL verification technique, which can be easily implemented via the BDD based symbolic approach. Moreover, we implement an extension of NuSMV, and this adapted version supports symbolic model checking of APSL. Experimental results show that this variant of PSL can be efficiently verified. Henceforth, symbolic model checking PSL can be carried out by a transformation from PSL to APSL and symbolic model checking APSL.

Keywords property specification language, symbolic model checking, tableau approach, extended NuSMV

1 Introduction

The daily increasing complexity of modern software/hardware raises spinous difficulties in guaranteeing correctness. Model checking has been proved to be an applicable technique in verifying high-level designs with respect to specifications. Early model checking technique (for linear time), usually boils problem to down that of empty-judging of \( \omega \)-automata. However, explicit model checking suffers from the fetter of state explosion. In the 1990s, the BDD [1] based technique was incorporated into model checking [2]. This dramatically enlarged the scale that can be verified with model checking tools.

To run the verification, we need two inputs: the model and the specification. Models are the formal abstraction of designs/ implementations, usually described as (fair) transition systems, while specifications are the formulated properties to be verified, and for reactive systems, specifications are usually written in various temporal logics.

Property specification language (PSL) is a temporal logic standardized by Accellera [3], and it has become an industrial standard (IEEE 1850-2005). This logic has received more and more frequent uses since 2004 [4, 5].

PSL has two perspectives: the linear perspective consists of FLs (Foundation Formulas), and the branching perspective consists of OBE formulas (Optional Branching Extension). For several reasons, we focus on FLs for the following reasons.

- OBE formulas are essentially CTL formulas, and the symbolic CTL model checking technique has been well studied.

- Temporal logics on linear structures have several advantages in comparison to that on branching structures [6]. In addition, FL is the core part of PSL, and FLs are much more frequently used in practice.

In the last two years, various converting approaches from FL fragments to finite automata [5, 7–9] are provided. These are useful in the automata based explicit PSL model checking.

In Ref. [10], it shows how to acquire a BDD-based LTL model checking procedure. That approach converts LTL model checking into CTL model checking with fairness constraints by building tableaux, and tableaux can be easily transformed into their symbolic representations. Henceforth, both LTL and CTL can be verified in the symbolic approach. As an implementation, the NuSMV tool [11] is developed, which is an extension of SMV without much modification.

Received September 1, 2008; accepted December 18, 2008
E-mail: {wwliu,wj}@nudt.edu.cn
FL formulas employ SEREs as additional constructs. SERE is an extension of RE. It is known that SEREs have precisely the expressiveness of REs [7], but can usually express properties much more succinctly.

In this paper, we present a variant of PSL, namely APSL$^1$. In this logic, in addition to several built-in temporal operators, finite automata are also used as formula constructs. The inter-conversion between standard PSL and APSL is succinct and straightforward.

Replacing SEREs with finite automata benefits from several aspects. First, automata have the clearer transition structures, which provide great succinctness in defining tableaux. Therefore, the verification technique of APSL is relatively easier to develop. Second, though SEREs are easier to write when defining specifications, we believe automata are much more intuitive, and it is possible to provide “reusable connective patterns” for the users.

We generalize the tableau-based LTL symbolic model checking technique to that of APSL, and the method is proposed to convert the APSL model checking problem to a CTL model checking problem by building a tableau, and then performing the BDD-based symbolic model checking.

Furthermore, we have adapted the NuSMV tool to support symbolic model checking of APSL. This extension enables users to customize their own temporal connectives, hence all omega temporal properties can be verified. The experimental results show that APSL can be efficiently verified via the symbolic approach.

Temporary logics employing non-star-free temporal operators (such as REs, automata) extend the expressiveness to full $\omega$-regular language. Thus, integrating the symbolic approach towards these operators within the existing symbolic model checking algorithm results in a more general symbolic model checking framework.

The remainder of this paper is organized as follows. Section 2 briefly revisits basic notions like finite automata (on finite words), recalls the definition of standard PSL, and presents APSL. Section 3 gives a tableau based APSL model checking algorithm which converts APSL model checking into CTL model checking, and then illustrates how to implement this algorithm via the BDD-based symbolic approach. Section 4 reveals an extension of NuSMV, which supports symbolic model checking of APSL. We introduce the extended syntactical features and then give some experimental results on APSL model checking. Finally, Section 5 gives concluding remarks of this paper.

$^1$The acronym “A” here means “automata”.

$^2$In this paper, we always let the index of letters start from 0.

$^3$Clocking operators are omitted here.

## 2 Preliminaries

### 2.1 SEREs and finite automata

Fix a finite set $AP$ of atomic propositions. A letter is a subset of $AP$, and a word is a sequence of letters. A **Boolean expression** is a formula built up from propositions in $AP$ and Boolean connectives (like $\land$, $\lor$, $\neg$, etc). In the sequel, we write $l$, $w$, $b$, $r$ (possibly with subscripts) to designate an individual letter, word, Boolean expression, SERE (to be defined later), respectively.

A letter $l$ satisfies the Boolean expression $b$ if and only if $b$ is evaluated to be True when assigning propositions in $l$ to True and assigning propositions not in $l$ to False.

We denote the **length** of $w$ by $|w|$, i.e., if $w$ is a finite word $l_0 l_1 \cdots l_m$, then $|w| = m + 1$; and if $w$ is an infinite word, then $|w| = \infty$. The $i$-th letter $l_i$ of $w$ is denoted by $w_i$ and the segment of $w$ starting from its $i$-th letter and ending with its $j$-th letter is denoted by $w_{i:j}$. Moreover, we denote by $w_{i:j}$ the suffix of $w$ starting from its $i$-th letter.

**Sequential Extended Regular Expressions (SEREs, for short)** are inductively defined as follows$^3$.

- Each Boolean expression is a SERE.
- If $r$, $r_1$ and $r_2$ are SEREs, then the following are SEREs:
  \[ r_1 ; r_2; \quad r_1 ; r_2; \quad r_1 | r_2; \]
  \[ r_1 \& r_2; \quad [\ast 0]; \quad r[\ast]. \]

Each SERE $r$ derives a set of finite words, denoted as $L(r)$, and are inductively defined as follows.

- $L(b) = \{ l \mid l \text{ satisfies } b \}$.
- $L(r_1 ; r_2) = \{ w_1 w_2 \mid w_1 \in L(r_1), w_2 \in L(r_2) \}$.
- $L(r_1 | r_2) = \{ w_1 l w_2 \mid w_1 l \in L(r_1), l w_2 \in L(r_2) \}$.
- $L(r_1 \& r_2) = L(r_1) \cup L(r_2)$.
- $L(r[\ast]) = \{ \ast \}$.
- $L([\ast 0]) = \{ \ast \}$.
- $L(r[\ast]) = \{ w_1 \cdots w_m \mid m \geq 0, \text{ and for each } 0 < i < m, w_i \in L(r) \}$.

A **non-deterministic automaton** is a tuple $A = \langle \Sigma, Q, \delta, Q_0, F \rangle$ where: $\Sigma$ is an alphabet; $Q$ is a finite set of states; $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function; $Q_0 \subseteq Q$, is a set of initial states; $F \subseteq Q$, is a set of accepting states.

A run of $A$ over a word $w$ is a sequence $\sigma = q_0 q_1 \cdots q_m \in Q^*$, where $q_0 \in Q_0$, $q_m \in F$ and each $q_{i+1} \in \delta(q_i, w_i)$. A word $w$ is accepted by $A$ if there is a run of $A$ over $w$. We