verbs: Verification of Finite State Systems by Bounded Correctness Checking

Wenhui Zhang
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China

1 Introduction

verbs[1] is a tool for verification of finite state systems by bounded correctness checking (a kind of bounded model checking [2, 1]). The model is a kind of finite state systems, and the properties are specified with CTL. The implementation relies on solving QBF-formulas for verification of CTL properties (the reader is referred to [10] for the details). Such an implementation has advantages over that based on traditional symbolic model checking [6, 4] in cases where a small bound is sufficient for verification or falsification of a property. Therefore bounded correctness checking and the traditional symbolic model checking may be considered complementary with their own advantages. The reader is referred to Appendix A for an experimental comparison of the efficiency of the implementations of these two approaches. For verification of ACTL formulas, *verbs* also includes an implementation based on solving SAT-formulas [2, 7–9]. This is specialized for ACTL properties and may be more efficient than the general one relying on solving QBF-formulas, when it is applicable.

2 Modeling and Specification

For concise representation of finite state systems, we use variables for representing system states, and logical formulas for transition relations. For property specification, CTL is used as the specification language.

2.1 Models and the Modeling Language

Firstly, we present a theoretical definition of finite state systems and then we discuss the modeling language with respect to the definition.

Finite State Systems A finite state system is a triple $M = \langle V, I, T \rangle$ where

- V: A finite set of typed *system variables* each with a finite domain. The set of states (interpretation) over V is denoted by Σ.
- I: The *initial condition* is an assertion (state formula) characterizing the initial states.
- T: The *transition relation* is an assertion $T(V, V')$ relating the variables in V of a state $s \in \Sigma$ to the variables in V' of a successor state $s' \in \Sigma$.

[1] http://lcs.ios.ac.cn/∼zwh/verbs/

Computation A state s' is an M-successor of s iff $T(s, s')$ holds. A computation of M is an infinite sequence of states $s_0, s_1, s_2, \ldots$ such that $s_0 \models I$, and for each $j = 1, 2, \ldots$, the state s_j is a M-successor of s_{j-1}.

Modeling Language The langauge used by the verification tool *verbs* for modeling finite state systems and specifying their properties is called VVM (*verbs* verification model). An example is given as follows.

An Illustrative Example The following is a specification of a mutual exclusion algorithm with two processes, and with properties including: mutual exclusion, progress, and non-starvation.

```
VVM    mutual exclusion
DEFINE critical=(p0.a=s2|p1.a=s2)
VAR    x[0..1]:0..1; t:0..1;
INIT   x[0]=0; x[1]=0;
PROC   p0:p0m(x[],t,0); p1:p0m(x[],t,1);
SPEC
       AF(((critical)));
       AG(!(p0.a=s2&p1.a=s2));
       AG((!p0.a=s1|AF(critical))&(!p1.a=s1|AF(critical)));
       AG((!p0.a=s1|AF(p0.a=s2))&(!p1.a=s1|AF(p1.a=s2)));
       AG((!p0.a=s1|EF(p0.a=s2))&(!p1.a=s2|EF(p1.a=s2)));

MODULE p0m(x[],t,i)
VAR    a: {s0,s1,s2,s3};
INIT   a=s0;
TRANS
       a=s0:                 (x[1-i],t,a):=(1,1-i,s1);
       a=s1&(x[i]=0|t=i):    (a):=(s2);
       a=s2:                 (x[1-i],a):=(0,s3);
       a=s2:                 (a):=(s2);
       a=s3:                 (x[1-i],t,a):=(1,1-i,s1);
```

In the program, the initial value of t is not given, this means that it can be either 0 or 1.

Explanation The relation between the specification in the sections in the modeling language and the components in a finite state system $\langle V, I, T\rangle$ is explained as follows.

Components	Specifications in
V :	VAR sections
I :	INIT sections
T :	TRANS sections

In order to increase the convenience of writing such a model, array variables may be used, and the transition relation represented by a set of guarded commands may be divided into different processes.

2.2 Property Specification Language

The properties are to be specified with the Computation Tree Logic (CTL) which is a propositional branching-time temporal logic [5] introduced by Emerson and Clarke as a specification language for finite state systems. Let AP be a set of propositional symbols and p range over AP. The set of CTL formulas Φ over AP is defined as follows:

$$\begin{aligned}\Phi ::= {} & p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \\ & AX\,\Phi \mid AF\,\Phi \mid AG\,\Phi \mid A(\Phi\,U\,\Phi) \mid A(\Phi\,R\,\Phi) \mid \\ & EX\,\Phi \mid EF\,\Phi \mid EG\,\Phi \mid E(\Phi\,U\,\Phi) \mid E(\Phi\,R\,\Phi)\end{aligned}$$

For the specification of a property in VVM, a proposition is written as $e_1 \sim e_2$ where e_1, e_2 are expressions and $\sim$ is an operator comparing the values of the two expressions.

3 Verification Methods

The verification tool implements two types of *bounded correctness checking* approaches: a QBF-based one for the verification of CTL properties, and a SAT-based one specialized to the verification of ACTL properties. In the following presentation, we assume that a finite state system is represented by $M = \langle V, I, T\rangle$ where V is a set of Boolean variables, since every such system where V is a set of variables of finite domain, can be represented by $M' = \langle V', I', T'\rangle$ where V' is a set of Boolean variables.

3.1 QBF-based Bounded Correctness Checking

k-Paths A k-path is a sequence of states $s_0, s_1, s_2, \ldots, s_k$ such that for each $j = 1, \ldots, k$, the state s_j is a M-successor of s_{j-1}.

Symbolic Representation of k-Paths Let $k \geq 0$. Let $u_0, ..., u_k$ be a finite sequence of state variables (each of the state variables is represented by a set of m propositional variables, i.e., a copy of V). The sequence $u_0, ..., u_k$ (denoted by $\overrightarrow{u}$) is intended to be used as a representation of a k-path of M. This is captured by the following definition of $P_k(\overrightarrow{u})$.

Definition 1.

$$P_k(\overrightarrow{u}) := \bigwedge_{j=0}^{k-1} T(u_j, u_{j+1})$$

Every assignment to the set of state variables $\{u_0, ..., u_k\}$ satisfying $P_k(\overrightarrow{u})$ represents a valid k-path of M. Let $rs_k(\overrightarrow{u})$ denote that the k-path represented by $\overrightarrow{u}$ is a repeating state path. Formally, we have the following definition of $rs_k(\overrightarrow{u})$.

Definition 2.

$$rs_k(\vec{u}) := \bigvee_{x=0}^{k-1} \bigvee_{y=x+1}^{k} u_x = u_y.$$

Let $p \in AP$ be a proposition symbol and $p(v)$ be the propositional formula such that $p(v)$ is true whenever v is assigned the truth value representing a state s in which p holds.

Definition 3 (Transformation of CTL Formulas). *Let $k \geq 0$. Let v be a state variable and φ be an CTL formula. The encoding $[[\varphi, v]]_k$ is defined as follows.*

$[[p, v]]_k$	$= p(v)$
$[[\neg p, v]]_k$	$= \neg p(v)$
$[[\varphi \vee \psi, v]]_k$	$= [[\varphi, v]]_k \vee [[\psi, v]]_k$
$[[\varphi \wedge \psi, v]]_k$	$= [[\varphi, v]]_k \wedge [[\psi, v]]_k$
$[[A\varphi, v]]_k$	$= \forall \vec{u}.(P(\vec{u}) \wedge v = u_0 \rightarrow [[\varphi, \vec{u}]]_k)$
$[[E\varphi, v]]_k$	$= \exists \vec{u}.(P(\vec{u}) \wedge v = u_0 \wedge [[\varphi, \vec{u}]]_k)$
$[[X\varphi, \vec{u}]]_k$	$= k \geq 1 \wedge [[\varphi, u_1]]_k$
$[[F\psi, \vec{u}]]_k$	$= \bigvee_{j=0}^{k} [[\psi, u_j]]_k$
$[[G\psi, \vec{u}]]_k$	$= \bigwedge_{j=0}^{k} [[\psi, u_j]]_k \wedge rs_k(\vec{u}))$
$[[\varphi U \psi, \vec{u}]]_k$	$= \bigvee_{j=0}^{k} ([[\psi, u_j]]_k \wedge \bigwedge_{t=0}^{j-1} [[\varphi, u_t]]_k)$
$[[\varphi R \psi, \vec{u}]]_k$	$= \bigwedge_{j=0}^{k} ([[\psi, u_j]]_k \vee \bigvee_{t=0}^{j-1} [[\varphi, u_t]]_k) \wedge (\bigvee_{t=0}^{k} [[\varphi, u_t]]_k \vee rs_k(\vec{u}))$

Let $I(v)$ denote the propositional formula that restricts potential values of v to the initial states of M.

Proposition 1. *Let φ be a CTL formula. $M \models \varphi$ iff there is a $k \geq 0$ such that $\forall v.(I(v) \rightarrow [[\varphi, v]]_k)$, and $M \not\models \varphi$ iff there is a $k \geq 0$ such that $\exists v.(I(v) \wedge [[\neg\varphi, v]]_k)$.*

QBF-based Bounded Correctness Checking Algorithm Let φ be a CTL formula. The corresponding QBF-based bounded correctness checking algorithm for $M \models \varphi$ is then as follows.

```
for ($k = 0$;1;$k$++) if ($\forall v.(I(v) \rightarrow [[\varphi, v]]_k)$ or $\exists v.(I(v) \wedge [[\neg\varphi, v]]_k)$) break;
report that $M \models \varphi$ holds iff $\forall v.(I(v) \rightarrow [[\varphi, v]]_k)$ holds;
```

The correctness and the termination of this approach are guaranteed by Proposition 1.

3.2 SAT-based Bounded Correctness Checking

For SAT-based bounded correctness checking , we do not need quantification over propositional variables. However, we may need a set of k-paths. We may use some of the symbols already defined in the previous subsection, however, for self-containedness, we may redefine the necessary symbols in this subsection.

Symbolic Representation of k-Paths Let $k \geq 0$. Let $u_{i,0}, ..., u_{i,k}$ be a finite sequence of state variables for each $i \in \{1, ..., b\}$ for a given b. The sequence $u_{i,0}, ..., u_{i,k}$ is intended to be used as a representation of a k-path of M. This is captured by the following definition of $P_k(i)$.

Definition 4.

$$P_k(i) := \bigwedge_{j=0}^{k-1} T(u_{i,j}, u_{i,j+1})$$

Definition 5 (Transition Relation). *Let $k \geq 0$.*

$$[[M]]_k^b := \bigwedge_{i=1}^{b} P_k(i)$$

This is a collection of $P_k(l)$ for $l = 1, ..., b$. Therefore we have $[[M]]_k^b \rightarrow P_k(l)$ for $l = 1, ..., b$. Let $rs_k(i)$ denote that there are same states appearing in different positions in path $P_k(i)$. Formally, we have the following definition of $rs_k(i)$.

Definition 6.

$$rs_k(i) := \bigvee_{x=0}^{k-1} \bigvee_{y=x+1}^{k} u_{i,x} = u_{i,y}.$$

Let $p \in AP$ be a proposition symbol and $p(v)$ be the propositional formula such that $p(v)$ is true whenever v is assigned the truth value representing a state s in which p holds.

Definition 7 (Translation of ACTL and ECTL formulas). *Let $k \geq 0$. Let v be a state variable and φ be an ACTL or ECTL formula. The encoding $[[\varphi, v]]_k^b$ is defined as follows.*

$$\begin{array}{ll}
\hline
[[p, v]]_k^b & = p(v) \\
[[\neg p, v]]_k^b & = \neg p(v) \\
[[\varphi \vee \psi, v]]_k^b & = [[\varphi, v]]_k^b \vee [[\psi, v]]_k^b \\
[[\varphi \wedge \psi, v]]_k^b & = [[\varphi, v]]_k^b \wedge [[\psi, v]]_k^b \\
\hline
[[A\varphi, v]]_k^b & = \bigwedge_{i=1}^{b}(v = u_{i,0} \rightarrow [[\varphi, i]]_k^b) \\
[[E\varphi, v]]_k^b & = \bigvee_{i=1}^{b}(v = u_{i,0} \wedge [[\varphi, i]]_k^b) \\
\hline
[[X\varphi, i]]_k^b & = k \geq 1 \wedge [[\varphi, u_{i,1}]]_k^b \\
[[F\psi, i]]_k^b & = \bigvee_{j=0}^{k}[[\psi, u_{i,j}]]_k^b \\
[[G\psi, i]]_k^b & = \bigwedge_{j=0}^{k}[[\psi, u_{i,j}]]_k^b \wedge rs_k(i) \\
[[\varphi U \psi, i]]_k^b & = \bigvee_{j=0}^{k}([[\psi, u_{i,j}]]_k^b \wedge \bigwedge_{t=0}^{j-1}[[\varphi, u_{i,t}]]_k^b) \\
[[\varphi R \psi, i]]_k^b & = \bigwedge_{j=0}^{k}([[\psi, u_{i,j}]]_k^b \vee \bigvee_{t=0}^{j-1}[[\varphi, u_{i,t}]]_k^b) \wedge (\bigvee_{t=0}^{k}[[\varphi, u_{i,t}]]_k^b \vee rs_k(i)) \\
\hline
\end{array}$$

Definition 8. *Let φ be an ACTL formula. $n_k(\varphi)$ is defined as follows.*

$n_k(p)$	$= 0 \ if \ p \in AP$
$n_k(\neg p)$	$= 0 \ if \ p \in AP$
$n_k(\varphi \wedge \psi)$	$= max(n_k(\varphi), n_k(\psi))$
$n_k(\varphi \vee \psi)$	$= n_k(\varphi) + n_k(\psi)$
$n_k(AX\varphi)$	$= n_k(\varphi) + 1$
$n_k(AF\varphi)$	$= (k+1) \cdot n_k(\varphi) + 1$
$n_k(AG\varphi)$	$= n_k(\varphi) + 1$
$n_k(A(\varphi U \psi))$	$= k \cdot max(n_k(\varphi), n_k(\psi)) + n_k(\psi) + n_k(\varphi) + 1$
$n_k(A(\varphi R \psi))$	$= k \cdot n_k(\varphi) + max(n_k(\varphi), n_k(\psi)) + 1$

Let $I(v)$ denote the propositional formula that restricts potential values of v to the initial states of M.

Proposition 2. *Let φ be an ACTL formula. Let $a = n_k(\varphi)$. $M \models \varphi$ iff there is a $k \geq 0$ such that $I(v) \wedge [[M]]_k^a \rightarrow [[\varphi, v]]_k^a$ is valid, and $M \not\models \varphi$ iff there is a $k \geq 0$ such that $I(v) \wedge [[M]]_k^a \wedge [[\neg\varphi, v]]_k^a$ is satisfiable.*

SAT-based Bounded Correctness Checking Algorithm Let φ be an ACTL formula. The corresponding SAT-based bounded correctness checking algorithm for $M \models \varphi$ is then as follows.

```
for ($k = 0$;1;$k$++)
{
 $a = n_k(\varphi)$;
 if ($I(v) \wedge [[M]]_k^a \wedge \neg[[\varphi, v]]_k^a$ is unsat or $I(v) \wedge [[M]]_k^a \wedge [[\neg\varphi, v]]_k^a$ is sat) break;
}
report that $M \models \varphi$ holds iff $I(v) \wedge [[M]]_k^a \wedge \neg[[\varphi, v]]_k^a$ is unsat;
```

The correctness and the termination of this approach are guaranteed by Proposition 2.

4 Running the Verification Tool

Some basic verification options are discussed in the following. Suppose that the model is contained in the file *mutex01.vvm*.

Verification without Options The following command checks the first property specified in the file. The verification method used in the verification depends on the property to be checked. If it is an ACTL formula, SAT-based bounded correctness checking is used, otherwise, QBF-based one is used.

```
verds mutex01.vvm
```

Verification Option -ck The following command checks the i-th property specified in the file, in which i must be instantiated to some number ≥ 1.

```
verds -ck i mutex01.vvm
```

Verification Option -QBF The following command uses QBF-based bounded correctness checking for check the given property.

```
verds -QBF -ck i mutex01.vvm
```

Verification Option -SAT The following command uses QBF-based bounded correctness checking for check the given property. In case the property is not an ACTL formula, the program signals an error.

```
verds -SAT -ck i mutex01.vvm
```

External QBF and SAT-Solvers The efficiency depends very much on the efficiency of QBF-solvers and SAT-solvers. External QBF-solver may be provided by using the option "-qbfsolver solver" and external SAT-solver may be provided by using the option "-satsolver solver" where *solver* is the name (possibly necessary with path information) of the solver.

Summary of the Basic Options The following table is a summary of the basic options discussed above.

no option	checks the first property in the file.
-ck i	checks the *i*-th property in the file.
-QBF	uses QBF-based bounded correctness checking
-SAT	uses SAT-based bounded correctness checking
-qbfsolver solver	uses *solver* for QBF-solving
-satsolver solver	uses *solver* for SAT-solving

Output of the Verification The output of the verification provides among others, information on whether the property is true, and the value of k indicating the bound that is sufficient for verification or falsification. Considering the 5 properties in the example in subsection 2.1, the verification results show that the 4th property is false, while the ones are true. It is also reported that the bounds are respectively 3, 10, 10, 2, and 10 for verification and falsification of the properties.

5 Concluding Remarks

The verification tool *verbs* with its modeling and specification language and its verification methods has been presented. A simple example has also been presented to show the use of the languages and the verification tool. Two distinguished features of *verbs* are that it implements QBF-based bounded correctness checking for the verification of CTL properties, and a SAT-based one specialized to the verification of ACTL properties.

Experimental Evaluation An experimental evaluation[2] of the relative efficiency of the QBF-based bounded correctness checking implemented in *verbs* and the BDD-based symbolic model checking implemented in NuSMV [3] version 2.5.0 has been done based on the used of two types of random boolean programs and a set of CTL properties. Based on the test cases, the experimental evaluation shows that each of the approaches has advantages in a significantly large subset of the test cases, and moreover, the advantages and disadvantages are well distributed among verification and falsification of universal properties. Although the test cases may not be typical in practical applications, the test cases have shown that bounded correctness checking and the traditional symbolic model checking may be considered complementary with their own advantages in different types of verification problems.

References

1. A. Biere, A. Cimmatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded Model Checking. Advances in Computers 58, Academic Press, 2003.
2. A. Biere, A. Cimmatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. LNCS 1579:193-207. TACAS 99.
3. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A New Symbolic Model Verifier. CAV 1999: 495-499.
4. E. M. Clarke, O. Grumberg and D. Peled. Model Checking. The MIT Press. 1999.
5. E. Allen Emerson and E. M. Clarke. Using Branching-time Temporal Logics to Synthesize Synchronization Skeletons. *Science of Computer Programming* 2(3): 241-266. 1982.
6. K. L. McMillan. Symbolic Model Checking. Kluwer Academic Publisher,1993.
7. W. Penczek, B. Wozna, and A. Zbrzezny. Bounded Model Checking for the Universal Fragment of CTL. Fundamenta Informaticae 51:135-156. 2002.
8. W. Zhang. Model Checking with SAT-Based Characterization of ACTL Formulas. Lecture Notes in Computer Science 4789 (ICFEM 2007):191-211.
9. W. Zhang. Bounded Semantics of CTL and SAT-based Verification. ICFEM 2009: 286-305.
10. W. Zhang. Bounded Semantics of CTL. Technical Report ISCAS-SKLCS-16, Institute of Software, Chinese Academy of Sciences. 2010.

[2] The report is available at http://lcs.ios.ac.cn/∼zwh/verbs/docfiles/verbs1ee1.pdf