Unified Bounded Model Checking for MSVL

Bin Yu, Zhenhua Duan(✉), and Cong Tian

ICTT and ISN Lab, Xidian University, Xi'an 710071, P.R. China
yubin@stu.xidian.edu.cn, zhenhua_duan@126.com,
{zhhduan,ctian}@mail.xidian.edu.cn

Abstract. This paper presents Unified Bounded Model Checking (UBMC) for the verification of an infinite state system described with Modeling, Simulation and Verification Language (MSVL) which is an executable subset of Projection Temporal Logic (PTL). The desired property is specified by a Propositional PTL (PPTL) formula. We present the bounded semantics of PPTL and the approach to implementing UBMC. A Bounded Labeled Normal Form Graph (BLNFG) is constructed on the fly and a counterexample of minimal length is produced to ease the interpretation and understanding for debugging purposes. Finally, a resource allocation algorithm is presented as an example to illustrate how the proposed approach works.

Keywords: Bounded model checking · Unified model checking · Propositional Projection Temporal Logic · Modeling · Verification

1 Introduction

Techniques for automatic formal verification of finite state transition systems have been studied in recent years. Compared to other formal verification techniques (e.g. theorem proving), model checking [1,2] is an automatical approach. Model checking has been widely used in many fields such as verification of hardware, software and communication protocols. In model checking, the system to be verified is modeled as a finite state machine, and the specification is formalized by a temporal logic formula.

For a system in practice, the number of states in it can be very large and the explicit traversal of the state space becomes infeasible. To fight with this problem, several approaches, such as Symbolic Model Checking (SMC) [3], Abstract Model Checking (AMC) [4], and Compositional Model Checking [5], etc. have been proposed with success. The combination of symbolic model checking with BDDs [6,7] pushed the barrier to systems with 10^{20} states and more later [3]. But the bottleneck of SMC is the amount of memory that is required for storing and manipulating BDDs. The boolean functions required to represent the set of states can grow exponentially. Bounded model checking (BMC) is an important

This research is supported by the NSFC Grant No. 61133001, 61272117, 61322202, 61420106004, and 91418201.

progress in formal verification after SMC [8]. The basic idea in BMC is to search for a counterexample in executions whose length is bounded by some integer k. If the property is not satisfied, an error is found. Otherwise, we cannot tell whether the system satisfies the property or not. In this case, we can consider to increase k, and then perform the process of BMC again. BMC problem can be efficiently reduced to a propositional satisfiability problem, and can therefore be solved by SAT methods rather than BDDs. SAT procedures do not suffer from the state space explosion problem of BDD-based methods. Modern SAT solvers can handle propositional satisfiability problems with hundreds of thousands of variables or more. Tools supporting BMC are NuSMV2 [9], bounded model checker developed by CMU [10], Thunder of Intel [11], and so on.

With model checking and bounded model checking, the mostly used temporal logics are LTL [12], CTL [1], and their variations. However, expressiveness of both LTL and CTL is not powerful enough. There are at least two types of properties in practice which cannot (or with difficulty to) be specified by LTL and CTL: (1) time related properties such as "a property P holds after the 100^{th} time unit and before the 200^{th} time unit"; (2) periodically repeated of property P. The expressiveness of Propositional Projection Temporal Logic (PPTL) [13] is full regular [14] which allows us to verify full regular properties and time related properties of systems in a convenient way.

In recent years, the verification of infinite state systems has attained increasing interest. The main limitation of model checking is that it is restricted to (essentially) finite-state systems. In general, the model checking problem is undecidable for infinite state systems, and hence, it may happen that the verification process does not terminate. In the verification of an infinite state system, theorem proving [15] is a powerful technique. Predicate abstraction has been introduced as a technique for reduction of infinite state systems to finite one in the work of Graf and Saidi [16]. Verification by abstraction can be applied to infinite state systems as shown in [17–19]. Another way to deal with the difficulty of verification is the method of compositional verification that uses the combination of temporal case splitting and data type reductions to reduce types of infinite or unbounded range to small finite types, and arrays of infinite or unbounded size to small fixed-size arrays [20,21]. In bounded model checking of infinite state systems [22], three-valued logic is employed in order to explicitly forward uncertain information in the case a proof cannot be established due to insufficient bounds.

Modeling, Simulation and Verification Language (MSVL) is a subset of Projection Temporal Logic (PTL) [13,23] with framing techniques [24]. It can be used for the purpose of modeling, simulation and verification of software and hardware systems. For the verification of a finite system by MSVL, a method named Unified Model Checking has been presented in [25]. With this method, a system is first modeled as p in MSVL. Thus, p is a non-deterministic program of MSVL and also a formula of PTL. Second, the property we want to check is specified by a formula ϕ in PPTL. To check whether or not p satisfies ϕ amounts to proving $\models p \rightarrow \phi$. It turns out to prove $\not\models p \wedge \neg\phi$. Thus, for finite state programs in MSVL, we can translate the model checking problem into a satisfiability problem in PPTL since finite state programs in MSVL are equivalent to PPTL formulas. To check the satisfiability of $p \wedge \neg\phi$, Labeled Normal Form Graph (LNFG) of $p \wedge \neg\phi$ can be constructed. But for an infinite state transition system, the path in the LNFG may be a straight infinite line and we cannot get the result that whether the system satisfies the given property forever.

Given an infinite system p in MSVL, a property of the system in terms of a PPTL formula ϕ, and a user supplied upper bound k, we present an approach named Unified Bounded Model Checking (UBMC) which combines bounded model checking and unified model checking approaches. In order to do this, bounded semantics of PPTL is presented. Further, the procedure of UBMC can be described as a process to construct the Bounded Labeled Normal Form Graph (BLNFG) of $p \wedge \neg\phi$ on the fly. BLNFG is constructed progressively as the current bound increases. If a finite or an infinite counterexample is found at a given bound that is less than the upper bound, the construction of the BLNFG stops and the counterexample is output. When there is no new node to be dealt with and no counterexample is found, the construction of BLNFG terminates and the result is given that the property is valid. If the current bound is increasing until the upper bound with no counterexamples found, it cannot be determined whether the system satisfies the property or not. At this time, we can increase the upper bound and construct the BLNFG of $p \wedge \neg\phi$ again.

The main advantages of our technique are the follows. First, our method can partially verify an infinite system described by MSVL. We can give the result that whether the property is valid in bound k. Second, our method can find counterexamples relatively quicker. This is due to the depth first nature in the construction of our BLNFG. Finding counterexamples is arguably the most important feature of model checking. Third, it finds a counterexample of minimal length. This feature helps users to understand a counterexample more easily.

This paper is organized as follows. In the next section, as a property specification language, PPTL formulas are presented. Then the language MSVL used for the description of an infinite system is formalized. In Sect. 3, the bounded semantics of PPTL formulas is given. Next, the method for constructing a BLNFG is formalized in detail. A resource allocation algorithm is presented to illustrate how our approach works in Sect. 5. Finally, conclusion is drawn in Sect. 6.

2 Preliminaries

2.1 Propositional Projection Temporal Logic

Let $Prop$ be a countable set of atomic propositions. A formula P of PPTL is given by the following grammar:

$$P ::= p \mid \bigcirc P \mid \neg P \mid P_1 \vee P_2 \mid (P_1, \cdots P_m)\ prj\ P$$

where $p \in Prop$, $P_1, \cdots, P_m$ and P are all well-formed PPTL formulas. $\bigcirc$ (*next*) and prj (*projection*) are basic temporal operators.

A mapping from $Prop$ to $B = \{true, false\}$ is used to define a state s, $s : Prop \rightarrow B$. $s[p]$ denotes the valuation of p at the state s. An interval σ is a non-empty sequence of states. The length of σ, $|\sigma|$, is the number of states minus 1 if σ is finite, and ω otherwise. The set of non-negative integers N_0 with $\{\omega\}$, $N_\omega = N_0 \cup \{\omega\}$ is used for both finite and infinite intervals. The relational operators, $=, <, \leq$, is extended to N_ω by considering $\omega = \omega$, and for all $i \in N_0, i < \omega$. Moreover, the relation symbol $\preceq$ is defined as $\leq -(\omega, \omega)$.

In an interpretation $\mathcal{I} = (\sigma, i, j)$, σ is an interval, i an integer, and j an integer or ω such that $i \preceq j \leq |\sigma|$. If formula P is interpreted and satisfied over a subinterval $< s_i, \cdots, s_j >$ of σ with the current state being s_i, it is denoted by the notation $(\sigma, i, j) \models P$. The satisfaction relation ($\models$) is inductively defined as follows:

$I - prop$ $\mathcal{I} \models p$ iff $s_i[p] = true$, and $p \in Prop$ is an proposition
$I - not$ $\mathcal{I} \models \neg P$ iff $\mathcal{I} \not\models P$
$I - or$ $\mathcal{I} \models P \vee Q$ iff $\mathcal{I} \models P$ or $\mathcal{I} \models Q$
$I - next$ $\mathcal{I} \models \bigcirc P$ iff $i < j$ and $(\sigma, i+1, j) \models P$
$I - prj$ $\mathcal{I} \models (P_1, \cdots, P_m)\ prj\ P$, if there exist integers $r_0 \leq r_1 \leq \cdots \leq r_m \leq j$ such that $(\sigma, r_0, r_1) \models P_1$, $(\sigma, r_{l-1}, r_l) \models P_l, 1 < l \leq m$, and $(\sigma', 0, |\sigma'|) \models Q$ for one of the following σ':
(a) $r_m < j and \sigma' = \sigma \downarrow (r_0, \cdots, r_m) \cdot \sigma_{(r_m+1, \cdots, j)}$, or
(b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, \cdots, r_h)$ for some $0 \leq h \leq m$.

where P, $P_1, \cdots, P_m$ and Q are PPTL formulas.

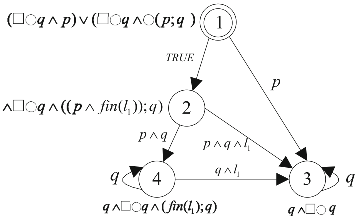
A formula P is satisfied by an interval σ, denoted by $\sigma \models P$, if $(\sigma, 0, |\sigma|) \models P$. A formula P is called satisfiable if $\sigma \models P$ for some σ. A formula P is valid, denoted by $\models P$, if $\sigma \models P$ for all σ.

For any PPTL formula Q, it can be rewritten into its normal form [26]:

$$NF(Q) \equiv \bigvee_{j=0}^{n_0} (Q_{ej} \wedge empty) \vee \bigvee_{i=0}^{n} (Q_{ci} \wedge \bigcirc Q_{fi})$$

where Q_{ej} and Q_{ci} are conjunctions of atomic propositions (or their negations) in Q_p which is the set of atomic propositions appearing in the PPTL formula Q, and Q_{fi} is an arbitrary PPTL formula.

For a PPTL formula Q, its corresponding LNFG can be constructed, which explicitly illustrates models of the formula. Here, an example is given to show the LNFG of a PPTL formula intuitively and the formal definition can be found in [27].



$(\Box\bigcirc q \wedge p) \vee (\Box\bigcirc q \wedge \bigcirc(p;q))$
1
TRUE
p
$q \wedge \Box\bigcirc q \wedge ((p \wedge fin(l_1));q)$
2
$p \wedge q$
$p \wedge q \wedge l_1$
q
4
$q \wedge l_1$
3
q
$q \wedge \Box\bigcirc q \wedge (fin(l_1);q)$
$q \wedge \Box\bigcirc q$

Fig. 1. LNFG of formula $\neg(true; \neg \bigcirc q) \wedge (p \vee \bigcirc(p;q))$

Example 1. LNFG of formula $\neg(true; \neg \bigcirc q) \wedge (p \vee \bigcirc(p; q))$ is shown in Fig. 1.

In the LNFG of a formula as shown in Fig.1, each node is specified by a PPTL formula, while each edge is a directed arc labeled with a state formula. The extra propositions l_k are employed to mark the infinite paths in the LNFG which are not the models of the PPTL formula.

2.2 Modeling, Simulation and Verification Language

MSVL is a subset of Projection Temporal Logic [13,23] with framing technique [24]. Based on the language, we have developed a model checking tool named MSV which works in three modes: modeling, simulation and verification.

The arithmetic expression e and boolean expression b of MSVL are inductively defined as follows:

$$\begin{aligned} e &::= \ n \mid x \mid \ \bigcirc x \mid \ \ominus x \mid e_0 \ op \ e_1 \ (op ::= + \mid - \mid * \mid \backslash \mid mod) \\ b &::= \ true \mid false \mid e_0 = e_1 \mid e_0 < e_1 \mid \neg b \mid b_0 \ \wedge \ b_1 \end{aligned}$$

where n is an integer and x a variable. The elementary statements in MSVL are defined as follows. The meanings of all statements in MSVL are given in [13].

Termination: $empty$ *State Assignment*: $x <== e$ *Assignment*: $x := e$
State Frame: $lbf(x)$ *Interval Frame*: $frame(x)$ *Conjunction*: $p \wedge q$
Selection: $p \vee q$ *Next*: $\bigcirc p$ *Always*: $\Box p$ *Sequence*: $p; q$
Conditional: $if\ b\ then\ p\ else\ q \overset{\text{def}}{=} (b \rightarrow p) \wedge (\neg b \rightarrow q)$
While: $while\ b\ do\ p \overset{\text{def}}{=} (p \wedge b)^* \wedge \Box(empty \rightarrow \neg b)$

where x denotes a variable, e stands for an arbitrary arithmetic expression, b denotes a boolean expression, and p, q stand for programs of MSVL.

Any MSVL program p can be rewritten into its normal form [13,24]. According to normal form, we can construct an LNFG $G = (CL(p), EL(p), v_0, \mathbb{L} = \{\mathbb{L}_1, \ldots, \mathbb{L}_m\})$ to model an infinite state MSVL program p. Each node is specified by a program in MSVL, while each edge is a directed arc labeled with a state formula p_e from node q to node r and identified by a triple, (q, p_e, r).

Note that the number of nodes is finite only when the range of values of the variables in the program is limited to a finite set. When the range of variables is infinite, we cannot construct a finite LNFG of the program. For example, in the program $frame(i)\ and\ (int\ i <== 0\ and\ skip;\ while(true)\{(i := i + 1)\})$, the value of i is increasing and a finite LNFG of the program cannot be constructed always. In the LNFG of an infinite state MSVL program, there exist three kinds of paths: finite paths, loop paths and infinite paths with infinite states.

3 Bounded Semantics for PPTL

The basic idea of bounded model checking, as explained before, is to consider only a finite prefix of a path that may be a witness to the desired property. We

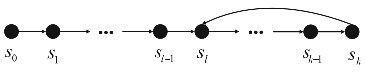


Fig. 2. σ is a (k, l)-loop

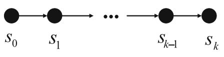


Fig. 3. σ is a finite interval

restrict the length of the prefix by some bound k. In practice, we progressively increase the bound, looking for witnesses in longer and longer traces.

A crucial observation is that, though the prefix of a path is finite, it still might represent an infinite path if there is a back loop from the last state of the prefix to any of the previous states as in Fig. 2. If there is no such loop, as in Fig. 3, the prefix does not say anything about the behavior of the path beyond state s_k.

Definition 1 ***((k,l)-loop).*** *For $l, k \in N_0$ and $l \leq k$, if there is a transition from s_k to s_l in σ i.e. $\sigma = (s_0, \cdots, s_{l-1}) \cdot (s_l, \cdots, s_k)^\omega$, we call interval σ a (k, l)-loop, k-loop for short.*

Obviously, if σ is an infinite interval with a loop, it must be a k-loop for some $k \in N_0$. We will use the notion of k-loop in order to define the bounded semantics of PPTL. The bounded semantics is an approximation to the unbounded semantics, which will allow us to define the bounded model checking problem. Since each PPTL formula can be transformed into an equivalent formula in NF, we do not need to deal with all types of PPTL formulas in the bounded semantics.

In the bounded semantics, we only consider a finite prefix of a path. In particular, we only use the first $k + 1$ states $(s_0, \ldots, s_k)$ of a path to determine the validity of a formula along the path. If a path is a k-loop then we simply maintain the original semantics of atomic propositions, $\neg$, $\vee$, and $\bigcirc$ operators, because all the information about this infinite path is contained in the prefix of length k. Since *empty* $\equiv \neg \bigcirc$ *true* and *more* $\equiv \bigcirc$*true*, the bounded semantics of *empty* and *more* can be deduced by the bounded semantics of $\neg$ and $\bigcirc$. In fact, the formula *empty* cannot be satisfied over an infinite interval, while *more* is satisfied all the time in an infinite interval.

Definition 2 ***(Bounded Semantics for a Loop).*** *Let $k \in N_0$ and σ be a k-loop interval, a PPTL formula f is valid along σ with bound k (denoted by $\sigma \models_k f$) iff $\sigma \models f$.*

We now consider the case where σ is not a k-loop. We use the notation $(\sigma, i) \models_k f$ $(0 \leq i \leq k \leq |\sigma|)$ to represent that formula f is interpreted and satisfied over the subinterval $< s_i, \cdots, s_k >$ of σ with the current state being s_i. $(\sigma, 0) \models_k f$ is denoted by $\sigma \models_k f$.

In the bounded semantics without a loop, we only consider formulas constructed from atomic propositions and negations of atomic propositions with $\vee$, $\wedge$, and $\bigcirc$ operators as well as *empty* and *more*.

Definition 3 ***(Bounded Semantics without a Loop).*** *Let $k \in N_0$ and σ be an interval that is not a k-loop. The bounded satisfaction relation $\models_k$ is defined as follows:*

$(\sigma, i) \models_k p$	*iff* $s_i[p] = true$ *if* $p \in Prop$ *is an atomic proposition*
$(\sigma, i) \models_k \neg p$	*iff* $s_i[p] = false$ *if* $p \in Prop$ *is an atomic proposition*
$(\sigma, i) \models_k P_1 \vee P_2$	*iff* $(\sigma, i) \models_k P_1$ *or* $(\sigma, i) \models_k P_2$
$(\sigma, i) \models_k P_1 \wedge P_2$	*iff* $(\sigma, i) \models_k P_1$ *and* $(\sigma, i) \models_k P_2$
$(\sigma, i) \models_k \bigcirc P$	*iff* $i + 1 \leq k$ *and* $(\sigma, i+1) \models_k P$
$(\sigma, i) \models_k empty$	*iff* $i = \lvert\sigma\rvert$
$(\sigma, i) \models_k more$	*iff* $i < \lvert\sigma\rvert$

Lemma 1. *Let $k \in N_0$, f be a PPTL formula, and σ a finite interval. We have $\sigma \models_k f \Rightarrow \sigma \models f$.*

Proof: To prove *Lemma* 1, we first prove a stronger conclusion given below:

$$(\sigma, i) \models_k f \Rightarrow (\sigma, i, |\sigma|) \models f \quad (0 \leq i \leq k)$$

Lemma 1 can be concluded by setting $i = 0$. We prove the above conclusion by induction on the structure of formula f:

Base case:

$f \equiv p \in Prop : (\sigma, i) \models_k p \Rightarrow s_i[p] = true \Rightarrow (\sigma, i, |\sigma|) \models p$

$f \equiv \neg p \in Prop : (\sigma, i) \models_k \neg p \Rightarrow s_i[p] = false \Rightarrow (\sigma, i, |\sigma|) \not\models p \Rightarrow (\sigma, i, |\sigma|) \models \neg p$

$f \equiv empty : (\sigma, i) \models_k empty \Rightarrow i = |\sigma| \Rightarrow (\sigma, i, |\sigma|) \models empty$

Inductive cases: Suppose for any PPTL formula f, $(\sigma, i) \models_k f \Rightarrow (\sigma, i, |\sigma|) \models f$.

1. By hypothesis, when $i < k \leq |\sigma|$, we have $(\sigma, i+1) \models_k f \Rightarrow (\sigma, i+1, |\sigma|) \models f$. By the definitions of semantics, $(\sigma, i+1) \models_k f$ iff $(\sigma, i) \models_k \bigcirc f$ and $(\sigma, i+1, |\sigma|) \models f$ iff $(\sigma, i, |\sigma|) \models \bigcirc f$, so we can get $(\sigma, i) \models_k \bigcirc f \Rightarrow (\sigma, i, |\sigma|) \models \bigcirc f$. When $i = k$, $(\sigma, i+1) \models_k f$ is $false$. Because $false \Rightarrow (\sigma, i, |\sigma|) \models \bigcirc f$, we can get $(\sigma, i) \models_k \bigcirc f \Rightarrow (\sigma, i, |\sigma|) \models \bigcirc f$.
2. By hypothesis, we have $(\sigma, i) \models_k P_1 \Rightarrow (\sigma, i, |\sigma|) \models P_1$ and $(\sigma, i) \models_k P_2 \Rightarrow (\sigma, i, |\sigma|) \models P_2$. By the definitions of bounded semantics, we can easily get $(\sigma, i) \models_k P_1 \vee P_2 \Rightarrow (\sigma, i, |\sigma|) \models P_1 \vee P_2$. Similarly, $(\sigma, i) \models_k P_1 \wedge P_2 \Rightarrow (\sigma, i, |\sigma|) \models P_1 \wedge P_2$.

Note that we do not need to deal with *more* in the above inductive cases since $more \equiv \bigcirc true$.

Lemma 2. *Let f be a PPTL formula and σ a finite interval. Then $\sigma \models f \Rightarrow \exists k, k \in N_0, \sigma \models_k f$.*

Proof: Since σ is a finite interval, so $|\sigma| \in N_0$.

$$\begin{aligned}\sigma \models f &\Rightarrow (\sigma, 0, |\sigma|) \models f\\ &\Rightarrow (\sigma, 0, k) \models f \wedge k = |\sigma|\\ &\Rightarrow \exists k, k \geq 0, (\sigma, 0) \models_k f\\ &\Rightarrow \exists k, k \geq 0, \sigma \models_k f\end{aligned}$$

4 Unified Bounded Model Checking of MSVL

In the Unified Model Checking implemented in [25], LNFG of $p \wedge \neg\phi$ is constructed to check whether a finite state MSVL program p satisfies a PPTL formula ϕ. According to [27], finite and infinite models of $p \wedge \neg\phi$ are precisely characterized by the paths which satisfy certain conditions in the LNFG. For a PPTL formula Q, an interval σ_π can be defined for a given path π in the LNFG of formula Q and given a model σ of formula Q , $\sigma \models Q$, a path π_σ can be constructed according to the transition rules [27].

In the previous section, we defined the bounded semantics of PPTL. According to it, a BLNFG can be constructed to describe the model of $p \wedge \neg\phi$ in bound k.

Definition 4. ***(Bounded Labeled Normal Form Graph, BLNFG).*** *For a MSVL program p, a PPTL formula ϕ, and $k \in N_0$, its BLNFG is a tuple $G = (CL(p \wedge \neg\phi), EL(p \wedge \neg\phi), v_0, \mathbb{L} = \{\mathbb{L}_1, \ldots, \mathbb{L}_m\}, \mathbb{C} = \{\mathbb{C}_1, \ldots, \mathbb{C}_k\})$, where $CL(p \wedge \neg\phi), EL(p \wedge \neg\phi), V_0$ and $\mathbb{L}$ are identical to the ones defined in* [27] and each $\mathbb{C}_i \subseteq CL(p \wedge \neg\phi), 0 \leq i \leq k$, is the set of nodes with $c_q = i$, where c_q represents the depth of a node q.

Since the set $CL(p\wedge\neg\phi)$ of nodes and the set $EL(p\wedge\neg\phi)$ of edges are inductively produced by repeatedly rewriting the new created nodes into their normal forms, the BLNFG can be constructed progressively with the current bound increasing until a user supplied upper bound k. When constructing BLNFGs by normal form reductions, for any chop formula $P;Q$, we equivalently rewrite it by $P \wedge fin(l_k);Q$ as implemented in [27]. For convenience, we use $\mathsf{inf}(\pi)$ to denote the set of nodes which infinitely often occur in the infinite path π.

By conclusions in [27], we can get the Corollaries 1 and 2 respectively as follows.

Corollary 1. *If π is a finite or an infinite path with $\mathsf{inf}(\pi) \nsubseteq \mathbb{L}_i$ for all $1 \leq i \leq m$ and $c_q \leq k$ $(k \in N_0)$ for all nodes q on π in the BLNFG of formula $p \wedge \neg\phi$, then the interval obtained from the path $\sigma_\pi \models_k p \wedge \neg\phi$.*

Corollary 2. *For a finite or an infinite interval σ, if $\sigma \models_k p \wedge \neg\phi$, then π_σ translated from σ with $\mathsf{inf}(\pi_\sigma) \nsubseteq \mathbb{L}_i$ for all $1 \leq i \leq m$ and $c_q \leq k$ for all nodes q on π_σ can be found in the BLNFG of formula $p \wedge \neg\phi$.*

Corollaries 1 and 2 tell us that a finite or an infinite interval $\sigma \models_k p \wedge \neg\phi$ iff there exists a corresponding finite or infinite path satisfying certain conditions in the BLNFG of formula $p \wedge \neg\phi$ in bound k.

Because a MSVL program p does not satisfy a PPTL formula ϕ iff $p \wedge \neg\phi$ is satisfiable, then we can get the following corollary which is important in our unified bounded model checking.

Corollary 3. *In the LNFG of an infinite system p in MSVL, finite paths precisely characterize finite models of p; loop paths with $\mathsf{inf}(\pi) \nsubseteq \mathbb{L}_i$ for all $1 \leq i \leq m$, precisely characterize loop models of p.*

According to Corollary 3, our bounded model checking approach can be deduced to the construction of the BLNFG of $p \wedge \neg\phi$. In the on-the-fly construction of $p \wedge \neg\phi$, i is the current bound which stops increasing once a counterexample is found or reaches the upper bound. Initially, we create the root node $p \wedge \neg\phi$ and set $c_{p\wedge\neg\phi}$ to 0.

For a given bound i, nodes whose depth equals i are dealt with. In order to retain consistency, we use $p \wedge \neg\phi$ to represent the node that will be dealt with. For a node $p \wedge \neg\phi$, p and $\neg\phi$ are rewritten into their normal forms respectively. The function $NF()$ defined in [26] is called to produce normal form of a PPTL formula or MSVL program. Then we can get the conjunction of these two normal forms. The first part of the conjunction is a disjunction. A finite path ended by the empty node is found once one disjunct can be satisfied. At this time, the construction of the BLNFG terminates and a finite counterexample is output. Because the depth of nodes that are dealt with is increasing progressively, our process produces counterexamples of minimal length, which eases the understanding for debugging purposes. If no counterexample is found by checking the

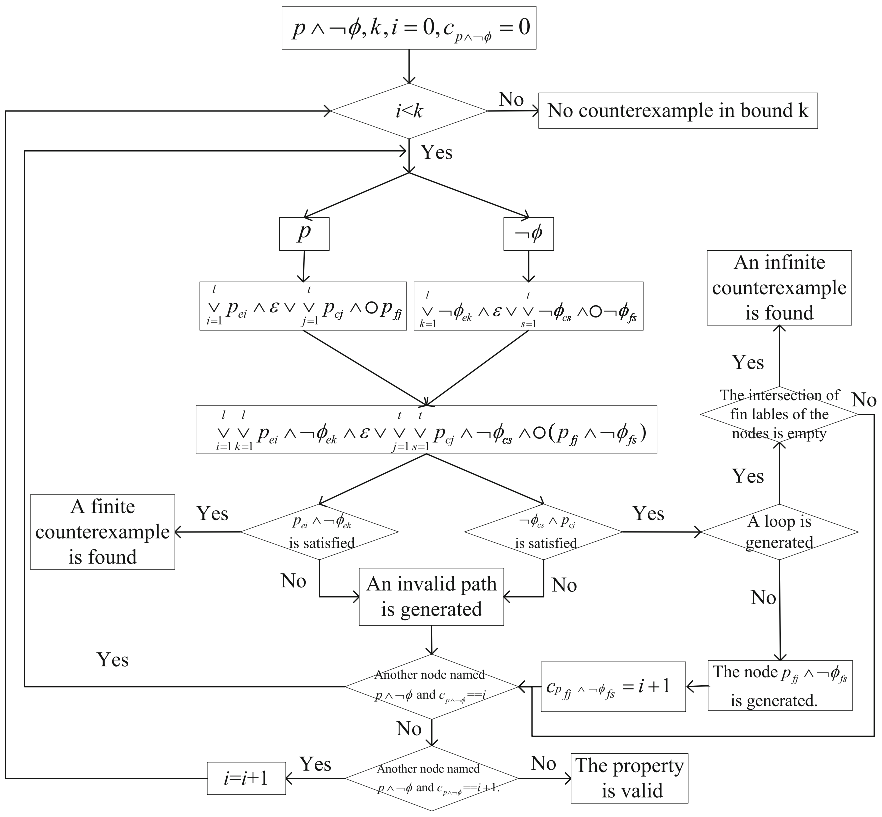

$p \wedge \neg\phi, k, i=0, c_{p\wedge\neg\phi}=0$
$i<k$
No
No counterexample in bound k
Yes
p
$\neg\phi$
$\bigvee_{i=1}^{l} p_{ei} \wedge \varepsilon \vee \bigvee_{j=1}^{t} p_{cj} \wedge \bigcirc p_{fj}$
$\bigvee_{k=1}^{l} \neg\phi_{ek} \wedge \varepsilon \vee \bigvee_{s=1}^{t} \neg\phi_{cs} \wedge \bigcirc \neg\phi_{fs}$
$\bigvee_{i=1}^{l}\bigvee_{k=1}^{l} p_{ei} \wedge \neg\phi_{ek} \wedge \varepsilon \vee \bigvee_{j=1}^{t}\bigvee_{s=1}^{t} p_{cj} \wedge \neg\phi_{cs} \wedge \bigcirc(p_{fj} \wedge \neg\phi_{fs})$
An infinite counterexample is found
Yes
The intersection of fin lables of the nodes is empty
No
Yes
A finite counterexample is found
Yes
$p_{ei} \wedge \neg\phi_{ek}$ is satisfied
$\neg\phi_{cs} \wedge p_{cj}$ is satisfied
Yes
A loop is generated
No
An invalid path is generated
No
No
Yes
Another node named $p \wedge \neg\phi$ and $c_{p\wedge\neg\phi}$==i
$c_{p_{fj} \wedge \neg\phi_{fs}} = i+1$
The node $p_{fj} \wedge \neg\phi_{fs}$ is generated.
No
i=i+1
Yes
Another node named $p \wedge \neg\phi$ and $c_{p\wedge\neg\phi}$==i+1.
No
The property is valid

Fig. 4. The construction of the BLNFG for $p \wedge \neg\phi$

first part, the second part of the conjunction will be dealt with. For every disjunct, a new node will be generated if the present state can be satisfied. If the node $p_{fj} \wedge \neg\phi_{fs}$ does not exist, a new node $p_{fj} \wedge \neg\phi_{fs}$ whose depth is $i+1$ is generated. Otherwise, a loop will be generated. If $\mathsf{inf}(\pi) \nsubseteq \mathbb{L}_i$ for all $1 \leq i \leq m$, the infinite path is output as a counterexample and the construction of BLNFG terminates. If not, this infinite interval is not a model of $p \wedge \neg\phi$ and is not taken into account. At this time, we will check whether there exist other nodes whose depth is equal to i. If these nodes exist, they will be dealt with by the same process above. Those new generated nodes whose depth is equal to $i+1$ will not be dealt with immediately. When the current bound increases to $i+1$, $p_{fj} \wedge \neg\phi_{fs}$ will be considered.

In case that all nodes whose depth is i have been dealt with and no counterexample has been found, the current bound i will increase. If there is no node to be dealt with at this time, it means that the whole LNFG of $p \wedge \neg\phi$ has been constructed. Because no valid path exists in the LNFG, we can get the result that the property ϕ is valid. If the value of i is still less than the upper bound k, the nodes whose depth equals i will be dealt with by the same process above. If the value of i is larger than the upper bound k, the process has to stop and it cannot be determined whether the system satisfies the property or not. The construction of the BLNFG for $p \wedge \neg\phi$ is depicted in Fig. 4.

Based on our UBMC algorithm, the model checker acting as a module in the MSV toolkit [25] has been developed. Under the bounded verification mode, a finite or an infinite system model is described by a MSVL program and the property is specified by a PPTL formula. A upper bound can be set by the user, otherwise it will be the default value.

5 A Case Study: Verification of Resource Allocation Algorithm

Banker's algorithm [28] is a resource allocation and deadlock avoidance algorithm developed by Dijkstra. It tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources. The algorithm tests for possible deadlock conditions for all other pending activities before deciding whether allocation is permitted.

The algorithm is originally developed in the design process of operating systems. When a new process enters a system, it must declare the maximum number of each resource type that it may ever claim. When a process gets all its requested resources, it must return them in a finite time interval. Many variations of Banker's algorithm have been applied in cloud computing where different computing tasks may be assigned to different platforms. Because resources available are usually limited on a given platform, it becomes necessary to check whether the tasks to be executed are schedulable.

We describe a variation of the resource allocation algorithm by a MSVL program where it is assumed that five tasks need to be scheduled every time and there exist four types of resources to be allocated. An array *maxres* is

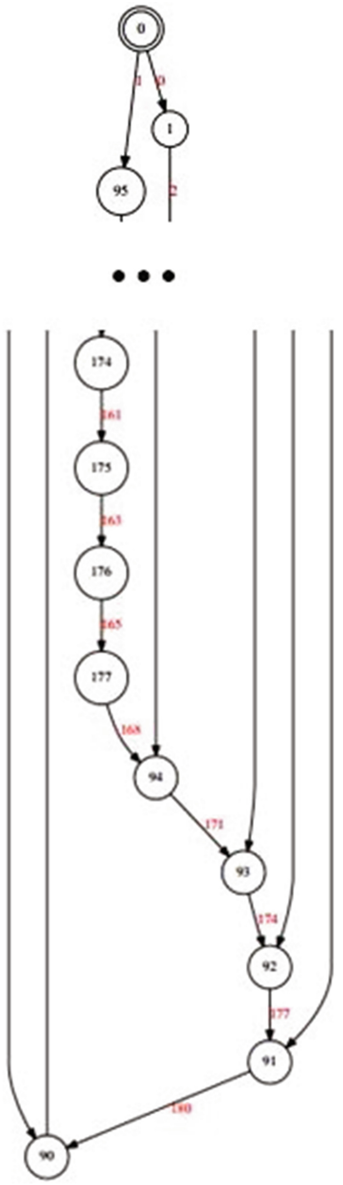


Fig. 5. The verification result of the resource allocation algorithm

used to indicate the number of resources available of each type. A 5×4 matrix *maxclaim* defines the maximum demand of each task and another 5×4 matrix *curr* defines the number of resources which are already allocated to each task. In the program, the platform keeps running through a nonterminal loop and the maximum demand of each task changes among three cases randomly. A boolean variable $fail$ is used to represent whether the tasks are schedulable in each case and $order$ is an array used to record every task's execution order.

Assume that we want to check whether or not the tasks are schedulable in each case and the third task is always executed after the second one. The property can be specified by $\Box(p \wedge q)$ in PPTL where p is defined as $fail = 0$ and q is defined as $order[1] < order[2]$. The upper bounded length is set to 100. Then we can verify the model with the bounded model checker. The verification result is shown in Fig. 5.

The counterexample is found when the bound increases to 89. By analysing the counterexample, we find that the resources available are too limited to schedule these five tasks. When the variable $maxclaim$=[[4, 2, 1, 4], [2, 2, 5, 2], [5, 1, 3, 5], [3 ,5 ,3 ,0], [3 ,2 ,3 ,3]], the first and the fourth types of resources are not enough. The program is verified again after we add one resource to the first and the fourth types respectively. Then we can get the result that the given property is valid for the modified program.

6 Conclusion

In this paper, we have proposed an approach named UBMC, which combines bounded model checking with unified model checking for verifying infinite state programs in MSVL. In our approach, a BLNFG is constructed on the fly to find whether there exist counterexamples in the given bound. This new technique produces counterexamples of minimal length and speeds up the verification. We also use a resource allocation example to show our approach. To examine our method, several case studies with larger examples are required in the near future. Moreover, lots of efforts are needed to improve our bounded model checker.

References

1. Clarke, E., Emerson, E.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, pp. 52–71. Springer, Heidelberg (1982)
2. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
3. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2), 142–170 (1992)
4. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. (TOPLAS) **16**(5), 1512–1542 (1994)
5. Clarke, E.M., Long, D.E., McMillan, K.L.: Compositional model checking. In: Proceedings of Fourth Annual Symposium on Logic in Computer Science, LICS 1989, IEEE, pp. 353–362 (1989)
6. McMillan, K.L.: Symbolic Model Checking. Springer, New York (1993)
7. Coudert, O., Madre, J.C.: A unified framework for the formal verification of sequential circuits. In: 1990 IEEE International Conference on Computer-Aided Design, ICCAD-90, Digest of Technical Papers, pp. 126–129. IEEE (1990)

8. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Adv. Comput. **58**, 117–148 (2003)
9. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: an opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
10. http://www.cs.cmu.edu
11. Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.Y.: Benefits of bounded model checking at an industrial setting. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 436–453. Springer, Heidelberg (2001)
12. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE (1977)
13. Duan, Z.: An extended interval temporal logic and a framing technique for temporal logic programming. Ph.D. thesis, University of Newcastle upon Tyne (1996)
14. Tian, C., Duan, Z.: Propositional projection temporal logic, b*ü*chi automata and ω-regular expressions. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 47–58. Springer, Heidelberg (2008)
15. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems—Safety. Springer, New York (1995)
16. Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Grumberg, Orna (ed.) CAV 1997. LNCS, vol. 1254. Springer, Heidelberg (1997)
17. Dingel, J., Filkorn, T.: Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In: Proceedings of 7th International Conference Computer Aided Verification, pp. 54–69 (1995)
18. Graf, S.: Characterization of a sequentially consistent memory and verification of a cache memory by abstraction. Distrib. Comput. **12**(2–3), 75–90 (1999)
19. Mouawad, A.E., Nishimura, N., Raman, V., Simjour, N., Suzuki, A.: On the Parameterized Complexity of Reconfiguration Problems. In: Gutin, G., Szeider, S. (eds.) IPEC 2013. LNCS, vol. 8246, pp. 281–294. Springer, Heidelberg (2013)
20. McMillan, K.L.: Verification of infinite state systems by compositional model checking. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 219–237. Springer, Heidelberg (1999)
21. Jhala, R., McMillan, K.L.: Microarchitecture verification by compositional model checking. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 396–410. Springer, Heidelberg (2001)
22. Schüle, T., Schneider, K.: Bounded model checking of infinite state systems. Formal Methods Syst. Des. **30**(1), 51–81 (2007)
23. Duan, Z.: Temporal Logic and Temporal Logic Programming. Science Press, Beijing (2005)
24. Duan, Z., Yang, X., Koutny, M.: Framed temporal logic programming. Sci. Comput. Program. **70**(1), 31–61 (2008)
25. Duan, Z., Tian, C.: A unified model checking approach with projection temporal logic. In: Liu, S., Araki, K. (eds.) ICFEM 2008. LNCS, vol. 5256, pp. 167–186. Springer, Heidelberg (2008)
26. Duan, Z., Tian, C., Zhang, L.: A decision procedure for propositional projection temporal logic with infinite models. Acta Informatica **45**(1), 43–78 (2008)
27. Duan, Z., Tian, C.: A practical decision procedure for propositional projection temporal logic with infinite models. Theor. Comput. Sci. **554**, 169–190 (2014)
28. Dijkstra, E.W.: Cooperating Sequential Processes. Springer, New York (2002)