A Compositional Symbolic Verification Framework for Concurrent Software *

Conghua Zhou
School of Computer Science and Telecommunication Engineering, Jiangsu University
No. 301 Xufu Road
Zhenjiang, China, 212013
chzhou@mail.edu.cn

ABSTRACT

For concurrent software systems state/event linear temporal logic SE-LTL is a specification language with high expressive power and the ability to reason about both states and events. Until now, SE-LTL model checking algorithm is still explicit. For SE-LTL we provide a SAT-based Bounded Model Checking procedure. We also present a framework for model checking concurrent software systems which integrates three powerful verification techniques, SAT-based Bounded Model Checking, counterexample-guided abstraction refinement and compositional reasoning. In the framework the abstraction and refinement steps are performed over each component separately, and the model checking step is symbolic.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification —*Model checking , Formal methods*

General Terms

Verification

Keywords

model checking, SAT, abstract, composition

1. INTRODUCTION

The specification logic SE-LTL[3] is a state/event derivative of LTL[5]. This allows us to represent both software implementations and specifications directly without any program annotations or privileged insights into program execution. Chaki further showed how standard automata-theoretic LTL model checking algorithms can be ported to their framework at no extra cost, enabling them to directly benefit from the large body of research on efficient LTL verification. However, until now SE-LTL model checking algorithm is still explicit.

Bounded Model Checking(BMC)[2] based on SAT methods has recently been introduced as a complementary technique to BDD-based symbolic model checking[6]. In this paper for SE-LTL we provid its SAT-based BMC procedure. Given an LKS M, an SE-LTL formula ϕ and a natural number k, our BMC procedure decides whether there exists a computation in M of length k or less that violates ϕ, i.e. $M \models_k E\neg\phi$. Our BMC is performed by generating a propositional formula, which is satisfiable if and only if such a computation exists.

We also present an efficient verification strategy which combines SAT-based BMC, counterexample-guided abstraction refinement[1, 4] and compositional reasoning[7]: starting with a coarse initial abstraction, our scheme computes increasingly precise abstractions of the target system by analyzing spurious counterexamples until either a real counterexample is obtained or the system is found to be correct.

More precisely, given a system M composed of n concurrent components $M_1, \ldots, M_n$, and a SE-LTL specification ϕ, the verification of $M \models \phi$ proceeds as follows:

(1) Abstract. Create an abstraction A such that all behaviors of A are preserved by M. This is done component-wise without constructing the full state space of M.

(2) Verify. Verify whether $A \models_k E\neg\phi$. If so, extract an abstract counterexample C that indicates in which ϕ fails in A. Otherwise check whether $k \leq CT$. If so, let $k = k + 1$, goto 2. Otherwise return $M \models \phi$.

(3) Check whether C gives rise to a real counterexample over A. If C corresponds to a genuine behavior of M then report a failure along with a fragment of each M_i that illustrates why $M \not\models \phi$. If C is spurious, on the other hand, refine A using C to obtain a more precise abstraction and repeat from step 1. This refinement step, like the initial abstraction, is performed component-wise.

Of the three steps in this abstract-verify-refine process only the verification stage of our technique requires the explicit composition of a system. The other stages can be performed one component at a time. To the best of our knowledge, our strategy is the first SAT-based, counterexample-guided, compositional abstraction refinement scheme to perform verification of linear time temporal specifications.

*Supported by the National Natural Science Foundation of China No.60603041, the Province Natural Science Foundation of Jiangsu No.BK2006073

Infoscale 2007 June 6-8, 2007, Suzhou, China

2. PRELIMINARIES

2.1 Labeled Kripke Structure

Definition 1. A labeled Kripke structure M is a 6-tuple $(S, s_0, AP, \Sigma, L, R)$ where, S is a finite non-empty set of states; $s_0 \in S$ is an initial state; AP is a finite set of atomic state propositions; Σ is a finite set of events; $L : S \to 2^{AP}$ is a state-labeling function; $R \subseteq S \times \Sigma \times S$ is a transition relation that must be total, that is, for every state $s \in S$ there is an event $a \in \Sigma$ and a state $s' \in S$ such that $R(s, a, s')$ holds.

Given an LKS $M = (S, s_0, AP, \Sigma, L, R)$ we write $S(M)$, $s_0(M)$, $AP(M)$, $L(M)$, $\Sigma(M)$ and $R(M)$ to mean S, s_0, AP, L, Σ and R respectively. A path $\pi = s_0, a_0, s_1, a_1, \ldots$ of M is an alternating infinite sequence of states and events subject to the following: for each $i \geq 0, s_i \in S, a_i \in \Sigma$ and $R(s_i, a_i, s_{i+1})$ holds. For the path $\pi = s_0, a_0, s_1, a_1, \ldots$ we use $\pi(i)$ to denote the ith state s_i, use $\pi(i, E)$ to denote the ith event a_i. We right $Path(M)$ to denote the set of infinite and finite paths whose first state is $s_0(M)$.

Definition 2. A path π is a (k, l)-*loop*, with $l < k$, if $(\pi(k), \pi(k, E), \pi(l)) \in R$ and $\pi = u \cdot v^{\omega}$, where $u = \pi(0), \pi(0, E), \ldots, \pi(l\text{-}1), \pi(l\text{-}1, E)$ and $v = \pi(l), \pi(l, E), \ldots, \pi(k), \pi(k, E)$. We call π simply a k-*loop* if there is an integer l with $0 \leq l \leq k$ such that which π is a (k, l)-*loop*.

2.2 Abstraction

The notion of abstraction is central to our approach.We list below the properties that we require of any abstraction scheme to be usable in our framework. Let $M = (S, s_0, AP, \Sigma, L, R)$ and $A = (S^A, s_0^A, AP^A, \Sigma^A, L^A, R^A)$ be two LKSs. We say that A is an abstraction of M, written $M \subseteq A$ iff $AP^A \subseteq AP$, $\Sigma^A = \Sigma$, for every path $\pi = s_0, a_0, s_1, a_1, \ldots$ of M there exists a path $\pi' = s'_0, a'_0, s'_1, a'_1, \ldots$ of A such that for each $i \geq 0$, $a'_i = a_i$ and $L^A(s'_i) = L(s_i) \cap AP^A$.

2.3 Parallel Composition

Let $M_1 = (S_1, s_0^1, AP_1, \Sigma_1, L_1, R_1)$ and $M_2 = (S_2, s_0^2, AP_2, \Sigma_2, L_2, R_2)$ be two LKSs. M_1 and M_2 are said to be compatible, i.e., that they do not share variables: $S_1 \cap S_2 = AP_1 \cap AP_2 = \emptyset$. The parallel composition of M_1 and M_2 is given by $M_1 \| M_2 = (S_1 \times S_2, s_0^1 \times s_0^2, AP_1 \cup AP_2, \Sigma_1 \cup \Sigma_2, L_1 \cup L_2, R)$ where $(L_1 \cup L_2)(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$ and $R((s_1, s_2), a, (s'_1, s'_2))$ holds iff one of the following holds:

1. $a \in \Sigma_1 \setminus \Sigma_2$ and $R_1(s_1, a, s'_1)$ holds and $s'_2 = s_2$.
2. $a \in \Sigma_2 \setminus \Sigma_1$ and $R_2(s_2, a, s'_2)$ holds and $s'_1 = s_1$.
3. $a \in \Sigma_1 \cap \Sigma_2$ and $R_1(s_1, a, s'_1)$ holds and $R_2(s_2, a, s'_2)$ holds.

Let M_1 and M_2 be as above, and let $\pi = (s_0^1, s_0^2), a_0, \ldots$ be an alternating infinite sequence of states and events of $M_1 \| M_2$. The projection $\pi \uparrow M_i$ of π on M_i consists of(possibly finite) the subsequence of $s_0^i, a_0, \ldots$ obtained by simply removing all pairs (a_j, s_{j+1}^i) for which $a_j \notin \Sigma_i$. In other words, we keep from π only those states that belong to M_i, and excise any transition labeled with an event not in Σ_i.

2.4 State/Event Linear Temporal Logic SE-LTL

Given an LKS $M = (S, s_0, AP, \Sigma, L, R)$, we consider linear temporal logic state/event formulas SE-LTL over the sets AP, Σ as follows:

- if $p \in AP$ then $p, \neg p$ is an SE-LTL formula; if $\sigma \in \Sigma$ then σ is an SE-LTL formula.
- if f is an SE-LTL formula, then Xf, Ff, Gf are SE-LTL formulas.
- if f, g are SE-LTL formulas, then $f \wedge g, f \vee g, fUg, fRg$ are SE-LTL formulas.

Definition 3. (Semantics of SE-LTL) Let $\pi = s_0, a_0, \ldots$ be an infinite path, π^i stands for the suffix of π starting in state s_i. $\pi \models f$ is inductively defined as follows:

- $\pi \models p$ if and only if $p \in L(\pi(0))$.
- $\pi \models \neg p$ if and only if $p \notin L(\pi(0))$.
- $\pi \models a$ if and only if $\pi(0, E) = a$.
- $\pi \models f \wedge g$ if and only if $\pi \models f$ and $\pi \models g$.
- $\pi \models f \vee g$ if and only if $\pi \models f$ or $\pi \models g$.
- $\pi \models Xf$ if and only if $\pi^1 \models f$.
- $\pi \models Ff$ if and only if for some $i \geq 0, \pi^i \models f$.
- $\pi \models Gf$ if and only if for all $i \geq 0$, $\pi^i \models f$.
- $\pi \models fUg$ if and only if there is some $i \geq 0$ such that $\pi^i \models g$ and for all $j \leq i - 1, \pi^j \models f$.
- $\pi \models fRg$ if and only if for all $i \geq 0$, $\pi^i \models g$ or there is some $i \geq 0$ such that $\pi^i \models f$ and for all $j \leq i, \pi^j \models g$.

Definition 4. $M \models f$ if and only if for all path π of M, $\pi \models f$. $M \models Ef$ if and only if there is a path π of M such that $\pi \models f$.

3. SAT-BASED BOUNDED MODEL CHECKING FOR SE-LTL

Bounded model checking based on SAT methods has been introduced as a complementary technique to BDD-based symbolic model checking. The main idea of bounded model checking is to search for an execution of the system of some length k, which constitutes a counterexample for a verified property. To perform bounded model checking on SE-LTL, we first define a bounded semantics for SE-LTL, which is an approximation to the unbounded semantics. Second we reduce bounded model checking to propositional satisfiability. Third we discuss the completeness threshold for bounded model checking on SE-LTL.

3.1 Bounded Semantics for SE-LTL

In bounded model checking a crucial observation is that 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. If there is no such back loop, then the prefix does not say anything about the infinite behavior of the path. Thus when we define bounded semantics for SE-LTL we must consider whether a finite path represents an infinite behavior.

Definition 5. (Bounded Semantics for a Loop) Let π be a *k-loop*. Then an SE-LTL formula f is valid along the path π with bound k (in symbols $\pi \models_k f$) iff $\pi \models f$.

Definition 6. (Bounded Semantics without a Loop) Let π be a path that is not k-loop. Then an SE-LTL formula f is valid along π with bound k (in symbols $\pi \models_k f$ iff $\pi \models_k^0 f$where

- $\pi \models_k^i p$ iff $p \in L(\pi(i))$; $\pi \models_k^i \neg p$ iff $p \notin L(\pi(i))$.
- $\pi \models_k^i a$ iff $a \equiv \pi(i, E)$.
- For all other cases, refer to [2].

We use the notation $M \models_k Ef$ to represent that there exists a path π of M such that $\pi \models_k f$.

Theorem 1. Let AP be a set of propositions, M be an LKS over AP, π be a path of M, f be an SE-LTL formula, and k be a bound. Then $\pi \models_k f$ implies $\pi \models f$.

If π is a k-loop then the conclusion in Theorem 1 follows by the definition. In the other case we can prove the conclusion by induction over the structure of f straightly.

Definition 7. For every LKS M and an SE-LTL property f, the natural number k called a CT of f if and only if the following condition holds: if there is no counterexample to f in M of length k or less, then $M \models f$.

Theorem 2. Let AP be a set of propositions, M be an LKS over AP, π be a path of M, f be an SE-LTL formula, and k be a natural number. Then $M \models Ef$ implies there exists a bound $k \leq |M| \times 2^{|f|}$ such that $M \models_k Ef$. In other words, $|M| \times 2^{|f|}$ is a CT of f.

3.2 Translation

In the previous section we defined the semantics for bounded model checking. We now show how to reduce bounded model checking to propositional satisfiability. This reduction enables us to use efficient propositional SAT solvers to perform model checking.

Given an LKS M, an SE-LTL formula f and a bound k, we will construct a propositional formula $[M, f]_k$. Let $s_0, a_0, \ldots, s_k, a_k$ be a finite sequences of states and events on a path π. Each s_i represents a state at time step i and consists of an assignment of truth values to the set of state variables. Each a_i represents an event at time step i and consists of an assignment of truth values to the set of event variables. The formula $[M, f]_k$ encodes constraints on $s_0, a_0, \ldots, s_k, a_k$ such that $[M, f]_k$ is satisfiable iff π is a witness for f. The definition of formula $[M, f]_k$ will be presented as three separate components. We first define a propositional formula $[M]_k$ that constraints $s_0, a_0, \ldots, s_k, a_k$ to be a valid path starting from the initial state. We then define the *loop condition*, which is a propositional formula that is evaluated to true only if the path π contains a loop. Finally, we define a propositional formula that constrains π to satisfy f.

Definition 8. (Unfolding the Transition Relation) For the bound k, we define $[M]_k := I(s_0) \wedge \bigwedge_{j=0}^{k-1} R(s_i, a_i, s_{i+1})$, where $I(s_0)$ is true if and only if s_0 is the initial state.

The translation of an SE-LTL formula depends on the shape of the path π. We define the propositional formula ${}_lL_k$ to be true if and only if there is a transition from sate s_k to state s_l.

Definition 9. (Loop condition) For two integers k, l with $k \geq l \geq 0$, let ${}_lL_k := R(s_k, a_k, s_l)$.

Depending on whether a path is a *k-loop*, we have two different translations of a SE-LTL formula f . First we consider the case where the path is a *k-loop*. We give a recursive translation of an SE-LTL formula f for a *k-loop* path π. The translation of f recurses over its subterms and the states in π. The intermediate formula ${}_l[\cdot]_k^i$ depends on three parameters: l, k and i. We use l for the start position of the loop, k for the bound, and i for the current position in π.

Definition 10. (Translation of an SE-LTL formula on a (k, l)-*loop*)

- ${}_l[a]_k^i := (a \equiv a_i)$;
- For all other cases, refer to [2].

For the case where π is not a *k-loop*, the translation can be treated as a special case of the *k-loop* translation. For LKS with total transition relations, every finite path π can be extended to an infinite one. Since the property of the path beyond event a_k is unknown, we make a conservative approximation and assume all properties beyond a_k are false.

Definition 11. (Translation of an SE-LTL formula without a Loop)

- $[a]_k^i := (a \equiv a_i)$
- For all other cases, refer to [2].

Combining all components, the encoding of a bounded model checking problem in propositional logic is defined as follows.

Definition 12. (General translation) Let M be an LKS, f be an SE-LTL formula, and k be a bound. We define $[f]_k := ([f]_k^0 \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge_l [f]_k^0)$ and $[M, f]_k := [M]_k \wedge [f]_k$

The translation scheme guarantees the following theorem, which we state without proof.

Theorem 3. Let M be an LKS, f be an SE-LTL formula, and k be a bound. Then $[M, f]_k$ is satisfiable if and only if $M \models_k Ef$.

Thus, the reduction of bounded model checking to SAT is sound and complete with respect to the bounded semantics.

Corollary 1. Let M be an LKS, f be an SE-LTL formula, and k be a bound. $M \models Ef$ if and only if there exists an integer $k \leq |M| \times 2^{|f|}$ such that $[M, f]_k$ is satisfiable.

4. COMPOSITIONAL SAT-BASED SE-LTL VERIFICATION

We now discuss how our framework enables us to verify SE-LTL specification on parallel compositions of LKSs incrementally and compositionally. When trying to determine whether an SE-LTL specification holds on a given LKS, the following result is the key ingredient needed to exploit abstractions in the verification process.

Theorem 4. Let M and A be LKSs with $M \subseteq A$. Then for any SE-LTL formula ϕ over M which mentions only propositions (and events) of A, if $A \models \phi$ then $M \models \phi$.

Suppose now that we are given a collection $M_1, \ldots, M_n$ of LKSs, as well as an SE-LTL specification ϕ, with the task of determining whether $M_1 || \ldots || M_n \models \phi$. We first create initial abstractions $M_1 \subseteq A_1, \ldots, M_n \subseteq A_n$. Then we check whether $A_1 || \ldots || A_n \models_k \neg\phi$. In the affirmative, we are provided with an abstract counterexample π such that $\pi \models_k \neg\phi$. We must then determine whether this counterexample is real or spurious, i.e., whether it corresponds to a counterexample in $M_1 || \ldots || M_n$ or not. In the negative, we check whether $k \leq CT$. If so, then check whether $A_1 || \ldots || A_n \models_{k+1} \neg\phi$. Otherwise return that $M_1 || \ldots || M_n \models \phi$.

This counterexample validation check can be performed compositionally, as follows. The counterexample is real iff for each i, the projection $\pi \uparrow A_i$ corresponds to (the prefix of) a valid behavior of M_i . To this end, we 'simulate' $\pi \uparrow A_i$ on M_i. If M_i accepts the path, we go on to the next component. Otherwise, we refine our abstraction A_i , yielding a new abstraction A_i' with $M_i \subseteq A_i' \subseteq A_i$ and such that A_i' also rejects the projection $\pi \uparrow A_i'$ of the spurious counterexample π.

Let us return to our SE-LTL specification ϕ, and let us fix throughout P_ϕ to be the set of all atomic state propositions appearing in ϕ. Consider any of the M_i's. An abstraction of M_i is entirely determined by a P_ϕ-respecting partition $\approx_i$ of the set of states of M_i: such an abstraction is denoted $M_i/\approx_i$.

The initial abstraction $M_i/\approx_i^1$ is the coarsest possible:$s \approx_i^1$ iff $L(M_i)(s) \cap P_\phi = L(M_i)(s') \cap P_\phi$. Suppose now that we are handed $\pi_i \in Path(M_i/\approx_i^k)$ (i.e., π_i is either a lasso or a finite path of $M_i/\approx_i^k$). We must determine whether π_i is a real or spurious counterexample component, i.e., whether π_i gives rise to a valid path of M_i or not. Moreover, in the latter case, we want to refine our partition $\approx_i^k$ into $\approx_i^{k+1}$ so that π_i is rejected by $M_i/\approx_i^{k+1}$.

The validation/refinement step is similar to that originally proposed by S. Chaki *et al.*[3]. Our validation/refinement step proceeds as follows. For any set Q of states of M_i and event a, let $Succ(Q, a) = \{s' | \exists s \in Q, (s, a, s') \in R\}$ denote the set of a-successors of Q in M_i. Let us first suppose that $\pi_i = s_0, a_0, s_1, a_1, ..., a_{m-1}, s_m, a_m$ is a finite path of $M_i/\approx_i^k$. Start with the set $Q_0 = \{s_0(M_i)\} \cap s_0$ and compute successively $Q_{j+1} = succ(Q_j, a_j) \cap s_j$. If, upon reaching the end of π, Q_{m+1} is non-empty, then clearly π is a valid finite path of M_i. Otherwise, let Q_{j+1} be the first empty set thus generated. Refine the partition $\approx_i^k$ by splitting s_j into Q_j and $s_j - Q_j$, yielding a new partition $\approx_i^{k+1}$. It is then easy to see that $M_i/\approx_i^{k+1}$ will reject π_i.

In case π is a lasso, things are slightly more complicated. If M_i rejects π, then the algorithm above will establish his in the very same manner, by eventually producing an empty set of states Q_{j+1}. On the other hand, if π is accepted by M_i then there will be sets of states $Q_j = Q_{j+p}$ such that Q_{j+p} is obtained from Q_j by following the loop part of π a finite number of times. Since all state spaces involved are finite the search will always terminate with one or the other answer after a finite number of iterations.

The full algorithm for checking whether $M_1 || \ldots || M_n \models \phi$ is given in Fig. 1. Note that the abstraction, counterexample-validation, and refinement steps are all performed one component at a time.

Algorithm SE-LTL Model Checking $(M_1, \ldots, M_n; \phi)$
for $i := 1$ to n: let A_i be the initial abstract of M_i;
for $k := 1$ to CT of $M_1 \parallel \ldots \parallel M_n$
{(1)decide whether $[A_1 \parallel \ldots \parallel A_n, \neg\phi]_k$ is satisfiable
if $[A_1 \parallel \ldots \parallel A_n, \neg\phi]_k$ is satisfiable, then suppose
π be path in $A_1 \parallel \ldots \parallel A_n$ violating ϕ;
looking for i such that $\pi \uparrow A_i$ is spurious;
if no such i then
return $M_1 \parallel \ldots \parallel M_n \not\models \phi$ along with
the counterexample derived from π;
else refine A_i, yielding a new abstraction A_i'
with $M_i \subseteq A_i' \subseteq A_i$ and A_i' also rejects $\pi \uparrow A_i'$;
Let $A_i = A_i'$, goto (1)
else if $[A_1 \parallel \ldots \parallel A_n, \neg\phi]_k$ is unsatisfiable
if $k = CT$ then return $M_1 \parallel \ldots \parallel M_n \models \phi$
else $k := k + 1$.}

Figure 1: The overall SE-LTL model checking algorithm for a concurrent system $M_1 \parallel \ldots \parallel M_n$ and specification ϕ

5. CONCLUSION AND FUTURE WORK

In this paper, we presented a SAT-based Bounded Model Checking procedure for the state/event linear time temporal logic SE-LTL. We further provided a new verification strategy for concurrent software system which integrates SAT-based BMC, counterexample-guided abstraction refinement and compositional reasoning. We are developing the tool for implementing the new strategy. For future work we would like to discuss other equivalent relations.

6. REFERENCES

[1] S. Bensalem, Y. Lakhnech, and S. Owre. Computing abstractions of infinite state systems compositionally and automatically. *LNCS*, 1254:319–331, 1998.

[2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without bdds. *LNCS*, 1579:193–207, 1999.

[3] S. Chaki, E. M. Clarke, J. Quaknine, N. Sharygina, and N. Sinha. Concurrent software verification with states, events, and deadlock. *Formal Aspects of Computing*, 17(4):461–483, April 2004.

[4] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. *LNCS*, 1885:154–169, 2000.

[5] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, MA, 1999.

[6] K. L. McMillan. *Symbolic model checking*. Kluwer Academic Publishers, Netherlands, 1993.

[7] K. L. McMillan. A compositional rule for hardware design refinement. *LNCS*, 1254:24–35, 1997.