On Invariance of Dynamic CTL Model Checking in Iterative Design of Moore Machine-based System

LI Shao-rong¹, YANG Shi-han², and WU Jn-zhao¹

(1. School of Optoelectronic Information, University of Electronic Science and Technology of China Chengdu 610054; 2. Chengdu Institute of Computer Applications, Chinese Academy of Sciences Chengdu 610041)

Abstract  Model checking is a promising approach to verifying safety properties of trusted computing systems in the design phase of system-level. Dynamic model checking is the model checking in which the model changes frequently along the design process. A serious problem for dynamic model checking is that the cost of re-checking is too expensive due to model being changed trivially, so a key issue of the problem is to seek invariance in order to avoid the checking repeatedly. An invariance is a true predicate that will remain true throughout a sequence of model checking. In this paper, a formal framework of dynamic model checking is constructed, and an invariance theory is proposed based on an iterative design process of flow control oriented systems described by Moore machines. It is proved that some non-trivial computation tree logic (CTL) properties can be preserved in the iteration.

Key words  computation tree logic; dynamic model checking; invariance; iterative design; Moore machine

1 Introduction

Model checking is an automated formal verification technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for a given state in that model. Model checking has been successfully applied to integrated circuit design and verification in last two decades. In some critical system (such as trusted computing system) designs, model checking could be the best formally method to apply for verification. But model checking of complex system is very expensive in time and space. Nowadays iteration is a common design strategy in modern system design and development, such as flow control oriented systems (FCOS)¹-², which is a kind of embedded control system stressing to handle data transformation in trusted computing system, and whose model is described by Moore machine. Iterative design is a system design methodology based on a cyclic process...
of prototyping, testing and verification, analyzing, and refining an application in progress. In detail, the iterative design process consists of 4 stages[3]: (1) definition of the specifications and design development; (2) production (model, mockup, prototype and product); (3) Verification / testing; (4) analyzing and evaluation. The four stages work through successively. At the end of each cycle the result is analyzed and evaluated. This serves as an input for the next cycle. In the iterative design process, it is very expensive and impractical to use the static model checking (the general model checking) approach along iterative process due to the high cost of re-checking.

In the static model checking framework, the system is assumed to be fixed. But this assumption is usually untenable in real iterative designing situation, in which model checking is embedded and models are subject to change frequently[4].

The dynamic problem is usually involved in model checking. Dynamic model checking is the problem of the model checking in which the model changes frequently along the designing process. How can the property be re-verified effectively when the model changes? To solve this, the following two basic questions have to be considered: (1) how can it be ensured that the verified property still holds after the model changes, and (2) how can a new property be verified artfully and effectively when model changes. For the first question, if there is a sequence of models of a system, says $M_0, M_1, \cdots, M_n = M$, where $M$ is the final model, it is of great benefit to establish implication relation $M_i \models \phi \rightarrow M_{i+1} \models \phi$, since the cost of re-verifying is usually much higher than the cost of proving the implication relation. For the second question, if some properties of the system are added or changed according to the requirement with the proceeding of development process, information verified in the anterior iterative phase may be possibly reused in verifying of current iterative phase.

In recent years, the investigation on dynamic problem of model checking focuses on restricting changes of the model or re-checking adaptively. If the development process is thought of a linear incremental design, where a final model $M$ is built through a successive sequence of models $M_0 \subseteq M_1 \subseteq \cdots \subseteq M_n = M$, such that $M_{i+1}$ somehow incorporates $M_i$ and enriches it with additional state or behavior, then incremental model checking methodology can be employed[5-6]. For a certain property, $\phi$, $\exists M_i \models \phi$ implying $M_{i+1} \models \phi$ means that the property $\phi$ remains unchanged during incremental design, $M_i \not\models \phi$ implying $M_{i+1} \not\models \phi$ means that the refutation of $\phi$ still remains. These results are valuable when the verification of $M_{i+1} \models \phi$ costs much more than the effort to prove $M_i \models \phi \rightarrow M_{i+1} \models \phi$. This incremental situation, however, is not always correct[7]. In iterative design processes, changes to model are almost arbitrary, which could be incremental or decremental. Reference[8] exploit the verification results to assist automatically learning the required updates of the model, then accelerate model rechecking processes when model changes. In addition, reference[9] gives a framework of dynamic model checking by applying game theory, and analyzes its complexity.

In these works, however, a key concept invariance is neglected. Moreover there are not works on model checking embedded in iterative design process. There is a very important issue on dynamic model checking, i.e. how can we reuse the information verified in the anterior iterative phase? The invariance can dramatically avoid the verifying repeatedly. If the invariance of dynamic model checking can be found, the re-checking is not an obstacle any more and model checking embedded in the iterative design process is practical in the industry. In this paper, we focus on preservation of computation tree logic (CTL) properties in model checking along iterative process based on FCOS design. We construct a formal framework of dynamic model checking, and propose an invariance theory of dynamic model checking, and prove that properties expressed by CTL formulae are preserved along the iterative design process in the theory. We show that if some properties hold in some model $M_i$, then they still hold in the successive model $M_j$ ($j > i$). Our approach is better because that the monotonic increment assumption of model
sequence is not required anymore, the limit (in reference [5], the new behaviors added must not override the previous ones) of system behavior evolving is relaxed, and the new valid properties are also expediently discovered in the theory. This theory is an extension and a supplement of the classical CTL case. This also provides a supplement that simplifies the design process by carrying on verification as soon as they involve instead of having to proceed to the later complex model, where model checking procedure usually has to face the notorious state space explosion problem.

In the rest of the paper, Section 2 reviews the CTL model checking. We formalize the dynamic problem of model checking in Section 3. Section 4 formalizes the iterative design process of Moore machine-based system. A fundamental theory on invariance of dynamic model checking is constructed in Section 5. Conclusion and future works are presented in Section 6.

2 CTL Model Checking

Model checking problem could be simply described as:

Given a simplified mode of a system, test automatically whether this model meets a given specification.

CTL model-checking is an automatic technique for verifying properties expressed in a propositional branching time temporal logic called computation tree logic (CTL). The system is defined by a Kripke structure, and properties are evaluated on a tree of infinite computations produced by the model of the system. The standard notation $K = \langle S, S_0, \text{AP}, L, R \rangle$ is an infinite sequence of states $s_0, s_1, s_2, \cdots$, such that $(s_i, s_{i+1}) \in R$ for all $i \geq 0$, and $s_0 \in S_0$.

For a Kripke structure $K = \langle S, S_0, \text{AP}, L, R \rangle$ and a state $s \in S$, there is an infinite computation tree with the root labelled $s$, such that $(s', s'')$ is an edge in the tree if and only if $(s', s'') \in R$. This tree is obtained by unfolding the Kripke structure at state $s$. The features of the computation tree could be captured by CTL logic language.

Let $f, g$ be CTL formulae, $\neg, \wedge$ boolean logic operators, $A$ an universal quantifier, $E$ an existent quantifier, and $X, F, G, U$ temporal operators. The syntax of CTL formulae is given as follows:

- Every atomic proposition is a CTL formula;
- If $f$ and $g$ are CTL formulae, then so are $\neg f, (f \wedge g), AXf, EFg, AFg, AGf, E(ufg)$. The other operators ($\vee, \rightarrow, AF, EF, AG, EF$) are viewed as being derived as usual from the following equations:

$$
f \vee g = \neg (\neg f \wedge \neg g)$$
$$f \rightarrow g = \neg f \vee g$$
$$AFg = A(true \ U g)$$
$$EFg = E(true \ U g)$$
$$AGf = E(\neg E(true \ U \neg f))$$
$$EFg = S(true \ U \neg f)$$

Let $K$ be a Kripke structure, $f, g$ CTL formulae, $s \in S$ and $i, j \in N$, $N$ a set of natural numbers. The interpretation of a CTL formula with respect to a Kripke model $K$ is given as following:

$$K, s \models p \iff p \in L(s)$$
$$K, s \models \neg f \iff s \not\models f$$
$$K, s \models f \wedge g \iff s \models f \text{ and } s \models g$$
$$K, s_0 \models AFf \iff \text{ for all paths } (s_0, s_1, \cdots), s_i \models f$$
$$K, s_0 \models EXf \iff \text{ for some path } (s_0, s_1, \cdots)$$
3 Dynamic Model Checking Framework

Static model checking is to compute $M \vDash \phi$ only once, where $M$ is usually the final model in design process. The model $M$ is often very complex and the model checking process is incidental to meet the state space explosion problem. However, the designer often wants to verify the model when it does not come to the end, because he thinks intuitively that some properties should be verified and that the model is not so complex yet. The frequency of re-verification is wanted to be reduced when the model improves. The designer even wants not to re-verify some properties when the model evolves. But now the fact is that the re-verifying / regressive-testing has to be made at each step in the iterative design process. Those are involved in the problems of dynamic model checking. How can the designer check / recheck the model when it does not evolve into the end? How can the cost of model checking / rechecking be reduced effectively when the model becomes complex. Let’s investigate the process of dynamic model checking at first.

Dynamic model checking is actually an iterative process of model checking, which is embedded in the iterative design process:

1. Checking $M \vDash \phi$;
2. Correcting or improving model $M$;
3. Going back to 1.

The reason why correcting or refining model $M$ is that system designs have to be corrected or refined according to the requirements along the iterative design process. Intuitively, dynamic model checking could be seen as a sequence of static model checking based on the natural arithmetic structure, which is described as follows.

Given a relational structure $\mathcal{A}$, the universe, denoted $|\mathcal{A}|$, is an initial segment of the natural numbers, that is, $|\mathcal{A}| = 0, 1, \ldots, n - 1$, where $n \in \mathcal{N}$, $\mathcal{N}$ is the set of natural numbers. In addition, we assume that the structure is provided with the built-in predicate $\leq$ (with the natural interpretation) and the built-in predicate $\text{BIT}$, which is used to query the binary representation of the numbers building the universe. Moreover, we assume that the structure has at least two elements, and we identify 0 with false and 1 with true. This kind of structures will be referred to as arithmetic structures. Given a relational vocabulary $\tau$, we write $\text{Struc}[\tau]$ for the set of all arithmetic structures with vocabulary $\tau$ (that is, $\{\leq, \text{BIT}\}$) and $\text{Struc}_n[\tau]$ for all such structures with $n$ elements.

A dynamic problem is usually specified by (1) a set of operations that can be used to build instances of the problem, (2) a set of all possible solutions to the instance represented by a sequence of those operations, and (3) an interpreting with the solution set on the sequence.

As far as dynamic model checking is concerned, the model $M$ of a system, a Kripke structure, is to be changed, but the property, specified by formula $\phi$, is assumed to be fixed. So, (1) the instance of dynamic model checking is one static model checking, and the operations of the instance(changes of the model) are insertion and deletion of states and relabelling of the states, that is, formally described as:

$$\sum^{mc} = \{\text{Insert}, \text{Delete}\} \cup \{\text{SetVar}_i| U \subseteq \text{AP}\}$$

where $\sum^{mc}$ is the vocabulary of operations, and we also use $\sum^{mc}_n$ to denote the set of operations for constructing the instance of the problem with size $n$, and we use $(\sum^{mc}_n)^*$ to denote transitive closure of $\sum^{mc}_n$. The meanings of Insert is an operator of inserting states into the Kripke structure. The meaning of Delete is an operator of deleting states from the Kripke structure, and the meaning of SetVar$_i$ is that the $i$th state of the Kripke structure gets label $U$, $\text{AP}$ is the finite set of atomic propositions. (2) The solution to the model checking instance is represented by a Boolean constant, $v$ (for verified), so the solution set is

$$\tau^{mc} = \{v\}, v \in \{\text{true}, \text{false}\}$$

and (3) an interpretation to the sequence of operations
is a partial function:

\[ s^{m\phi}_u : u \mapsto A^{m\phi}_u \]

where \( u \in (\sum^{m\phi})' \), \( A^{m\phi}_u \in \text{Struc}_u[\tau^{m\phi}] \), and \( \phi \vdash v \)
if and only if \( \phi \) holds in the Kripke structure with state set \( \{0,1,\ldots,n-1\} \), with the initial state 0 and with edge relation (transition relation) labeling according to \( u \).

So dynamic model checking is formally given by

Definition 3 (Dynamic model checking)
Dynamic model checking (DMC) is formally described as a 3-tuple:

\[ \text{DMC} = (\sum^{m\phi}, \tau^{m\phi}, s^{m\phi}_u) \]

where \( \sum^{m\phi} \) is a set of operators that can be used to change the model, \( \tau^{m\phi} \) is a set of solution to model checking, i.e. \{true, false\}. It is true when \( M \vdash \phi \)
holds, and false when others, and \( s^{m\phi}_u \) is a partial function defined above, says how a sequence of operators maps to the solution set of model checking.

Intuitively, dynamic model checking is a successive sequence of static model checking during system design process. It comprises three essential parts, a stepwise improving sequence of the system model, a sequence of solution to model checking problem for the property and every model among the model sequence, and a relation function between the model sequence and the solution sequence. Of course, an invariance of dynamic model checking is a key issue. The invariance is defined as follows:

Definition 4 (Invariance) In dynamic model checking, an invariance is a predicate \( \vdash \) that, if holds, will remain holds throughout a sequence of static model checking, where the predicate \( \vdash (M \vdash \phi) \) is satisfiable relation between a model \( M \) and a property \( \phi \) as usual.

According to the different set of operators \( (\sum^{m\phi}) \), different kinds of problem of dynamic model checking could be formalized. In this paper, the iterative FCOS design is considered. Let \( \sum^{m\phi} = \{\text{Insert, Delete} \} \cup \{\text{SetVar}^{(b)}_i\} \), and dynamic model checking embedded in iterative system design process is formally defined as above.

4 Formalization of Iterative Design Process

In this section, we describe formally the system-level design process of FCOS, whose model is described by Moore machines. The FCOS is composed of components. A component is viewed as a control part driving a data path, and a component presents an interface made of directed typed signals. A component could be added into the system or be deleted from the system or be revised by the design engineers during iterative design process. A component could be usually modeled by a complete and deterministic synchronous Moore machines. So changes of the system along iterative design process could be seen as changes of Moore machines, into which states are added, or from which states are deleted. For the formal description of a component, signal and configuration are key concepts.

4.1 Configuration and Component

Signal is the basic concept in the field of digital design. A signal is a time-varying or spatial-varying quantity. In FCOS design, the signal is defined as follows.

Definition 5 (Signal) Each signal is defined by a variable name \( s \) and an associated finite definition domain \( \text{Dom}(s) \).

The value of all signals at a special moment is formed as a configuration of the component. The configuration expresses the state of the component at that point.

Definition 6 (Configuration) Let \( E \) be a set \( E = s_1, s_2, \cdots, s_n \) of signals. A configuration \( c(E) \) is a conjunction of the associations: for each signal in \( E \), one signal associates one value of its definition domain \( \text{Dom}(s) \).

The set of all configurations \( c(E) \), named \( C(E) \), is \( \text{Dom}(s_1) \times \text{Dom}(s_2) \times \cdots \times \text{Dom}(s_n) \).

Sometimes we do not consider all signals at a moment, and we are only interested in part of them. The value of that part of signals at a moment could be seen as the projection of a configuration.

Definition 7 (Projection) A projection of a configuration \( c(E) \) on the \( i \)th signal is a function \( p_i(c(E)) = v_i, v_j \in \text{Dom}(s_i), s_i \in E \).

Moreover, we denote \( p_i(c(E)) = (v_1, v_2, \cdots, v_m) \), for \( 1 \leq k \leq m \), \( v_i \in \text{Dom}(s_i) \), \( s_i \in I, I \subseteq E \) as projection of a set \( I \) of signals, and denote \( p_i(c(E)) = (v_1, v_2, \cdots, v_{i-1}, v_{i+1}, \cdots, v_m) \), \( v_j \in \text{Dom}(s_j), j \neq i \) as a
sub-configuration without ith signal participating in. We also denote \( p_\tau(c(E)) \) as a sub-configuration without set \( I \) of signals participating in, and denote \( p_\tau(C(E)) \) as a set of all sub-configurations without set \( I \) of signals participating in.

We consider applying changes to a component \( W_i \) to evolve the next different component \( W_{i+1} \) in the sequence of components, where \( W_i \) refers to the component resulting from the ith successive changes. At first, a component could be defined as a complete and deterministic synchronous Moore machines by concepts described above.

**Definition 8 (Component)** A component \( W_i = \langle S_i, I_i, O_i, T_i, L_i, s_{in} \rangle \) is described as a deterministic and complete Moore machine, where \( S_i \) is a finite set of states, \( I_i \) is a finite set of input signals with their finite definition domain, \( O_i \) is a finite set of output signals with their finite definition domain, \( T_i \subseteq S_i \times C(I_i) \times S_i \). Finite set of transitions, \( \forall s \in S_i, \forall c \in C(I_i), \exists s' \in S_i \) s.t. \((s, c, s') \in T_i \) (\( \exists ! \) means there exists exactly one), \( s_{in} \in S_i \) is the initial state, and \( L_i = l_0, \ldots, l_{|L_i|} \) is a vector of generation functions, and each function defines the value of exact one output signal in each state; for all output signal \( o_j \), \( 0 \leq j < |O_i| \), we have \( l_j : S_i \rightarrow \text{Dom}(o_j) \). Applying the vector of generation functions to a given state of \( S_i \) produces a configuration \( c(O_i) \).

### 4.2 Changes of the Component

A change is a set of modifications applied to a component’s architecture for getting a new component with more correctness during the iterative design processes. It reflects the occurrence of a new event at the component’s interface. The new event affects the component by means of two basic ways, either just addition or just deletion of behaviors and a set of states and output signals. The new event is modeled by the appearance of a changed set of input signals with their definition domain. The set of all configurations corresponding to the changed input signals is split into two disjoint sets representing that the changed event is active or not.

**Definition 9 (Event)** An event \( e \) to component \( W_i \) is a triple \[ e = \langle \Delta I, C_{act}(\Delta I), C_{q}(\Delta I) \rangle \]

where \( \Delta I = I_+ \cup I_- \) is the set of added input signals with their definition domain, \( I_+ \) is the set of deleted input signals, \( I_- \cap I_+ = \emptyset \), i.e. signal name should not be reused. If \( I \neq \emptyset \) then some input signals have been deleted. Assuming domain of all signals is \( \{0,1\} \), if a signal \( s \in I_- \) and \( p_\tau(c(E)) = 1 \), then configuration \( c(E) \) should be already deleted, and configurations \( c(E) \) should be changed into \( p_\tau(c(E)) \). When some configurations are deleted, (i.e. some edges of the Moore machine have been deleted) some states maybe have zero in-degree, so these states should be deleted from the Moore machine of \( W_i \).

\( C_{act}(\Delta I) \) is the set of configurations representing the occurrence of the changed event. If one such configuration occurred, the event would be said to be active.

\( C_{q}(\Delta I) \) is the set of configurations representing the absence of the changed event. If one such configuration occurred, the event would be said to be quiet.

There are two kinds of basic event, addition event and deletion event.

**Definition 10 (Addition Event)** An event \( e \) is an addition event, denoted \( e_{\text{ADD}} \), if \( \Delta I = I_+ \) in above definition of event \( e \).

**Definition 11 (Definition Event)** An event \( e \) is a deletion event, denoted \( e_{\text{DEL}} \), if \( \Delta I = I_- \) in above definition of event \( e \).

Obviously, we have \( C_{act}(\Delta I) \cup C_{q}(\Delta I) = C(\Delta I) \) and \( C_{act}(\Delta I) \cap C_{q}(\Delta I) = \emptyset \).

By means of the concept of events, the change of a component is formally defined as follows.

**Definition 12 (Change)** A change to a component \( W_i \) is a four tuple:

\[ \Delta W_i = \langle E, \Delta \Sigma, \Delta T, \Delta O \rangle \]

where \( E \) is the set of events described above, \( \Delta \Sigma \) is the set of changed states, \( \Delta T \) is the set of changed transition, and \( \Delta O \) is the set of changed output signals and their definition domain.

There are two basic changes to component \( W_i \), i.e. increment and decrement. If only addition events impose on \( W_i \), we say an incremental change to it. If
only deletion events impose on $W_i$, we say a decremental change to it. If both addition and deletion events occur to $W_i$, we say that there is a composite change (or general change) to $W_i$.

**Definition 13 (Increment)** An incremental change to a component $W_i$ is a four tuple:

$$INC = \langle e_{ADD}, \Sigma_c, T_s, O_c \rangle$$

Where $e_{ADD}$ is the additional event described above, $\Sigma_c$ is the set of new reachable states, $T_s \subseteq (S \times C(I_1 \cup I_3) \times S) \cup (\Sigma \times C(I_1 \cup I_3) \times \Sigma) \cup (S \times C(I_1 \cup I_3) \times \Sigma) \cup (\Sigma \times C(I_1 \cup I_3) \times S)$, and $O_c$ is the set of new output signals and their definition domain, with $C_{inc}(O_c)$, the set of configurations representing the non-activation of the output. The output functions associated to $O_c$ return a configuration in $C_q(O_c)$ for all states in $\Sigma$.

**Definition 14 (Decrement)** A decremental change to a component $W_i$ is a four tuple:

$$DEC = \langle e, \Sigma_c, T_s, O_c \rangle$$

Where $e_{DEL}$ is the deletional event described above, $\Sigma_c$ is the set of states deleted by event $e_{DEL}$, $T_s \subseteq S \setminus \Sigma \times C(I_1 \cup I_3) \setminus S \setminus \Sigma$, where $S \setminus \Sigma$ is a set difference of $S$ and $\Sigma$, and $I_1 \setminus I_3$ a set difference of $I_1$ and $I_3$. $C(I_1 \setminus I_3) = p_T(C(I_1)))$, and $O_c$ is the set of output signals deleted, $O_c \subseteq O_c$.

A component $W_{i+1}$ is obtained by applying changes to a component $W_i$. There are three cases by definition above, i.e. incremental change, decremental change and compositive change. There is a simulation relation between two Moore Machines when we consider the relation of model checking problems between $W_i$ and $W_{i+1}$.

**Definition 15 (Simulation relation)** Let $M$ and $M'$ be two Moore Machines with $I \subseteq I'$ and $O \subseteq O'$, $s_0 \in S$ (resp. $s_0' \in S'$). A relation $H \subseteq S \times S'$ is a simulation relation from $(M,s_0)$ to $(M',s_0')$ if and only if the following conditions hold:

1. $H(s_0,s_0)$;
2. For all $s$ and $s'$, $H(s,s')$ implies: (1) the projection of $L'(s')$ onto $O'$ is equal to $L(s)$, (2) for every $p$ such that $(s,C(I_1),p) \in T$, there exists $p'$ $(s',C(I'),p') \in T$ and $H(p,p')$.

With this simulation relation we have

**Theorem 1** $(W_{i+1},s_{i+1})$ simulates $(W_i,s_i)$, if $W_{i+1}$ is obtained from $W_i$ by only applying incremental changes.

Proof: We build a binary relation $\rho$ between the states of two consecutive components $W_i$ and $W_{i+1}$ such that $\rho \subseteq S_i \times S_{i+1}$: $\forall (s,c,p) \in T_i$ and $(s',c',p') \in T_{i+1}$, we set $(s,s') \in \rho$ iff $s' = s$ and $c' = c \land e_q t_c e_q t_c A C I p T$ . By the construction, $\rho$ is a simulation relation.

**Theorem 2** $(W_i,s_i)$ simulates $(W_{i+1},s_{i+1})$, if $W_{i+1}$ is obtained from $W_i$ by only applying decremental changes.

Proof: We build a binary relation $\rho$ between the states of two consecutive components $W_i$ and $W_{i+1}$, such that $\rho \subseteq S_i \times S_{i+1}$: $\forall (s,c,p) \in T_i$ and $(s',c',p') \in T_{i+1}$, we set $(s,s') \in \rho$ iff $s = s'$ and $c = p_T(c')$. By the construction, $\rho$ is a simulation relation.

### 4.3 Translation into Kripke Structures

Using the framework of dynamic model checking of flow control oriented system design given above, we can interpret insert and delete operators in operational set ($\Sigma^\omega$) based on the changes defined above, and the input configurations that label the transitions by the Moore machine are incorporated into the states in a Kripke structure (resp. SetVar operator).

It is easy to deduce a Kripke structure from a given component $W_i$ by the following definition.

**Definition 16 (Deduce Kripke structure)** Given a component $W_i$, the corresponding Kripke structure can be obtained as follows:

$$K(W_i) = \langle S_{K(W_i)}, \Sigma_{K(W_i)}, \Delta P_{K(W_i)}, L_{K(W_i)}, R_{K(W_i)} \rangle$$

Where $S_{K(W_i)} = S_i \times C(I_1)$, $\Delta P_{K(W_i)} = \Delta P \times C(I_1)$, $L_{K(W_i)} = \{l_{i_0}, \ldots, l_{i_{k-1}}\}, \{l_{i_0}, \ldots, l_{i_{k-1}}\}$ is a vector of $|\Delta P_{K(W_i)}|$ function, and is vector concatenation, and $R_{K(W_i)} \subseteq S_{K(W_i)} \times S_{K(W_i)}$ and $\forall (s,c) \in S_{K(W_i)}$, $\forall (s',c') \in S_{K(W_i)}$, $(s,c) \in R_{K(W_i)}$ iff $(s,c) \in R_{K(W_i)}$. $W_{i+1}$ is produced from $W_i$ by applying change operators defined above. $K(W_{i+1})$ is considered to be produced from $K(W_i)$ by applying some operators in $\Sigma^\infty$, although it is not in fact. We are interested in
whether a CTL specification formula $\phi$ still holds or not in $K(W_{i+1})$ if it was verified previously in $K(W_i)$.

5 Invariance of Dynamic Model Checking

Some implications between the Kripke structures deduced from the components, which are in different phases of the iterative FCOS design process, show the invariance in this iterative dynamic model checking. Suppose component $W_{i+1}$ is changed from component $W_i$ by applying change events, $K(W_i)$ and $K(W_{i+1})$ are the Kripke structures deduced from $W_i$ and $W_{i+1}$ respectively. During the design processes mentioned previously, we investigate whether every CTL formula holds in component $W_i$ and in the successive component $W_{i+1}$. At first, relationships between $K(W_i)$ and $K(W_{i+1})$ are observed by the following two definitions.

Definition 17 (Enrichment relation) For all states $t=(s,c)\in K(W_i)$, if there exist $t'=(s',c')$ and $t^s=(s^*,c^*)\in K(W_{i+1})$ such that (1) $s=s',c'=c\land e\_qt$ and (2) $s=s^*,c^*=c\land e\_act$, then $t'$ and $t^s$ are said to enrich $t$, and $K(W_{i+1})$ is said to enrich $K(W_i)$.

Definition 18 (Impoverishment relation) For all states $t=(s,c)\in K(W_i)$, if there exists $t'=(s',c')\in K(W_{i+1})$ such that $s=s',c'=p_f(c)$, then $t'$ is said to impoverish $t$, and $K(W_{i+1})$ is said to impoverish $K(W_i)$.

Then simulation relations between $W_i$ and $W_{i+1}$ can deduce relationships between $K(W_i)$ and $K(W_{i+1})$. This is demonstrated by the following theorems (easy to prove them by definition of simulation relations).

Theorem 3 If $(W_{i+1},s_{i+1})$ simulates $(W_i,s_i)$ by applying an incremental change to $W_i$, then $K(W_{i+1})$ enriches $K(W_i)$.

Theorem 4 If $(W_i,s_i)$ simulates $(W_{i+1},s_{i+1})$ by applying a decremental change to $W_i$, then $K(W_{i+1})$ impoverishes $K(W_i)$.

So the invariance of dynamic model checking in the iterative design process of FCOS can be described by the following two theorems, it is easy to prove these theorems by induction on the structure of CTL formulae and semantics of CTL. The property, expressed by a kind of CTL formulae, can be preserved in the iterative design process, and the verified information on counter example can be reused in the verifying of current iterative phase.

Theorem 5 $K(W_i), s\models \phi \rightarrow K(W_{i+1}), s'\models \phi$ with enrichment relation, if a CTL formula $\phi$ obtained by applying recursively the rules of CTL formulae: (1) every atomic proposition is a CTL formula, (2) if $f$ and $g$ are CTL formulae, then so are $\neg f$, $(f \land g), EXf, E(fUg)$.

Proof: By induction on the structure of CTL formulae, let $q$ be an atomic proposition, $f,g$ be two CTL formulae, $s_0$ (resp. $s_y$) be the initial state in $K(W_i)$ (resp. $K(W_{i+1})$).

- $\phi=q$. $K(W_i), s\models q$, so $q\in L_{K(W_i)}(s)$. By addition event to $W_i$, we have $q\land e\_qt\in L_{K(W_{i+1})}(s')$, so $q\in L_{K(W_{i+1})}(s')$, i.e. $K(W_{i+1}), s'\models q$.

- $\phi=f\land g$. By the semantics of CTL formulae, $K(W_i), s\models f$ and $K(W_i), s\models g$, then $f\in L_{K(W_i)}(s)$ and $g\in L_{K(W_i)}(s)$. Because $s'$ enriches $s$, so $f\land e\_qt\in L_{K(W_{i+1})}(s')$ and $g\land e\_qt\in L_{K(W_{i+1})}(s')$, so $f\land g\in L_{K(W_{i+1})}(s')$, i.e. $K(W_{i+1}), s'\models f\land g$.

- $\phi=EXf$. Assume that $\exists s_y, s_t, \cdots, s_i\models f$, then $f\in L_{K(W_i)}(s_i)$, so $f\land e\_qt\in L_{K(W_i)}(s_i)$. Suppose $s_y$ enriches $s_0$, $s_i$ enriches $s_1$, $\cdots$, etc. So there is a path $s_0, s_t, \cdots$, such that $f\in L_{K(W_i)}(s_i)$, i.e. $K(W_{i+1}), s'\models EXf$.

The other cases could be proved same.

As for the CTL formulae $AXf, A(fUg)$, one could not assure whether they still hold or not in the changed component $W_{i+1}$, because this kind CTL formulae is universal quantifier path formulae, where we can not figure out any universal path features only from a part of the path.

Theorem 6 $K(W_{i+1}), s'\models \phi \rightarrow K(W_{i+1}), s''\models \phi$ with impoverishment relation, if a CTL formula $\phi$ obtained by recursively applying the constructive rules of CTL formulae: (1) every atomic proposition is a CTL formula, (2) if $f$ and $g$ are CTL formulae, then so are $\neg f$, $(f \land g), EXf, E(fUg)$.

The significance of this theorem lies in its converse-negative proposition:

$K(W_i), s\not\models \phi \rightarrow K(W_{i+1}), s''\not\models \phi$
which tells us that a counterexample in the previous component $W_i$ must be a counterexample in the next component $W_{i+1}$ changed from $W_i$ by applying deletion event. Moreover, given a formula to be verified on a model, if we can find a decremental change of the model (impoeverishing it), then the verification could be applied to the simpler model.

6 Conclusions

We have investigated how to handle dynamic model checking problems by very special perspective during iterative design process of Moore machine-base system, and proposed a theory of invariance on dynamic model checking embedded in the iterative design process. The invariance can dramatically reduce the cost of rechecking.

In the near future, we are interested in finding a decrement such that a complex model could be impoverished to a simpler model by applying decremental changes, then one can verify formulae in the simpler model according to Theorem 6. And we are specially paying attention to dynamic problems of model checking in iterative developing process of software engineering, in which the dynamic problems are much frequently encountered. How to reuse the reserved information that could be obtained from both previous verifications and changed-events to accelerate model rechecking processes is another researching interest of us.

References


编 辑 漆 蓉

李绍荣，电子科技大学教授，现任电子科技大学光电技术工程中心主任；中国兵工学会光学专业委员会委员；《光学技术》杂志编委。近五年，在国际国内专业刊物以及国际学术会议发表论文20余篇，被SCI、EI检索10余篇。

先后承担和参与了国家“973”、“863”、四川省重大科技攻关、成都市重大科技专项等国家重点科研项目6项；主持完成40多项涉及到航空航天、公共安全、石油天然气、精密设备制造等行业中的项目与产品研发。其技术领域涉及光电控制、ASIC芯片设计及验证、嵌入式系统、通信及网络应用软件系统等。

获2007年国家科技进步二等奖一项；获2006年教育部科技进步二等奖一项。获专利6项；软件著作权2项。

研究方向为复杂电子系统、集成电路验证技术及专用集成电路设计及其应用。