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

LI Shao-rong[1], YANG Shi-han[2], and WU Jin-zhao[1]

(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

Moore机表示的系统迭代设计动态CTL模型检验的不变性研究

李绍荣[1]，杨世翰[2]，吴尽昭[1]

(1. 电子科技大学光电信息学院 成都 610054; 2. 中国科学院成都计算机应用研究所 成都 610041)

【摘要】模型检验是系统级设计中验证可信计算系统安全性性质的有效方法。动态模型检验是模型随设计过程而变化的模型检验，动态模型检验过程中遇到的最严重问题之一是模型变化所带来的重复检验代价太高。因此，寻找不变性以避免重复检验显得尤为重要。不变性是一种贯穿系列模型检验而保值为真的性质。该文构建动态模型检验的形式化框架，进而提出基于Moore机描述的流控制系统迭代设计过程的不变性理论，该系统是一种嵌入式控制系统，在可信通信中用以处理数据转换，最后展示了若干非平凡CTL性质在迭代过程中的可保持性。

关 键 词 计算树逻辑; 动态模型检验; 不变性; 迭代设计; Moore机
中图分类号 TP301.6 **文献标识码** A doi:10.3969/j.issn.1001-0548.2009.05.026

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

Received date: 2009 – 08 – 05
收稿日期：2009 – 08 – 05
Foundation item: 863 Program(2007AA01Z143)
基金项目：国家863计划(2007AA01Z143)
Biography: LI Shao-rong, born in 1964, male, Professor, research interests include verification and evaluation of complex systems.
作者简介：李绍荣(1964 –)，男，教授，主要从事复杂系统验证与评估技术方面的研究.

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 $M_i \vDash \phi \to M_{i+1} \vDash \phi$, 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, $\phi\$, \$M_i \vDash \phi$ implying $M_{i+1} \vDash \phi$ means that the property ϕ remains unchanged during incremental design, $M_i \nvDash \phi$ implying $M_{i+1} \nvDash \phi$ means that the refutation of ϕ still remains. These results are valuable when the verification of $M_{i+1} \vDash \phi$ costs much more than the effort to prove $M_i \vDash \phi \to M_{i+1} \vDash \phi$. 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 $M, s \vDash p$ indicates that a formula *p* holds in a state *s* of a model *M*. If a formula holds in the initial state, it is considered to hold in the model.

Definition 1 (Kripke structure) A Kripke structure is a five-tuple:

$$K = \langle S, S_0, \mathrm{AP}, L, R\rangle$$

where S is a finite set of states, $S_0 \subseteq S$ is the set of initial states, AP is a finite set of atomic propositions.

$L = l_0, l_1, \cdots, l_{|\mathrm{AP}|-1}$ is a vector of $|\mathrm{AP}|$ functions. Each function defines the value of exactly one atomic proposition; for all $0 \leqslant i \leqslant |\mathrm{AP}|$ we have $l_i : S \to \mathbb{B}$, $\mathbb{B} = \{\text{true}, \text{false}\}$; for all $s \in S$, we have that $l_i(s)$ is true if and only if the atomic proposition associated to l_i is true in *s*, and $R \subseteq S \times S$ is the transition relation.

A computation of system is an infinite sequence of states, where each state is obtained from the previous state by some transition. Paths in a Kripke structure model computations of the system.

Definition 2 (Path) A path in a Kripke structure $K = \langle S, S_0, \mathrm{AP}, L, R\rangle$ is an infinite sequence of states $s_0 s_1 s_2 \cdots$, such that $(s_i, s_{i+1}) \in R$ for all $i \geqslant 0$, and $s_0 \in S_0$.

For a Kripke structure $K = \langle S, S_0, \mathrm{AP}, L, R\rangle$ 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[10] is given as follows:

- Every atomic proposition is a CTL formula;
- If f and g are CTL formulae, then so are $\neg f, (f \wedge g), AXf, EXf, A(fUg), E(fUg)$.

The other operators $(\vee, \to, AF, EF, AG, EF)$ are viewed as being derived as usual from the following equations:

$$f \vee g = \neg(\neg f \wedge \neg g)$$
$$f \to g = \neg f \vee g$$
$$AFg = A(\text{true } U\ g)$$
$$EFg = E(\text{true } U\ g)$$
$$AGf = \neg E(\text{true } U\ \neg f)$$
$$EFg = \neg S(\text{true } U\ \neg f)$$

Let *K* be a Kripke structure, f, g CTL formulae, $s \in S$ and $i, j \in \mathcal{N}$, $\mathcal{N}$ a set of natural numbers. The interpretation of a CTL formula[10] with respect to a Kripke model *K* is given as following:

$K, s \vDash p$	iff	$p \in L(s)$
$K, s \vDash \neg f$	iff	$s \nvDash f$
$K, s \vDash f \wedge g$	iff	$s \vDash f$ and $s \vDash g$
$K, s_0 \vDash AXf$	iff	for all paths $(s_0, s_1, \cdots)$, $s_1 \vDash f$
$K, s_0 \vDash EXf$	iff	for some path $(s_0, s_1, \cdots)$,

$s_1 \vDash f$
for all paths $(s_0, s_1, \cdots)$,

$K, s_0 \vDash A(fUg)$ iff for some $i, s_i \vDash g$ and
for all $j < i, s_j \vDash f$

for some path $(s_0, s_1, \cdots)$,

$K, s_0 \vDash E(fUg)$ iff for some $i, s_i \vDash g$ and
for all $j < i, s_j \vDash f$

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, $|\mathcal{A}| = 0,1,\cdots,n-1$, 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 $\leqslant$ (with the natural interpretation) and the built-in predicate $\mathrm{BIT}^{(2)}$, 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 τ, we write $\mathrm{Struc}[\tau]$ for the set of all arithmetic structures with vocabulary τ (that is, $\tau \bigcup \{\leqslant, \mathrm{BIT}\}$) and $\mathrm{Struc}_n[\tau]$ for all such structures with n elements.

A dynamic problem is usually specified[11] 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 ϕ, 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:

$$\Sigma^{mc_\phi} = \{\mathrm{Insert}, \mathrm{Delete}\} \cup \{\mathrm{SetVar}_U \mid U \subseteq \mathrm{AP}\}$$

where Σ^{mc_ϕ} is the vocabulary of operations, and we also use $\Sigma_n^{mc_\phi}$ to denote the set of operations for constructing the instance of the problem with size *n*, and we use $(\Sigma_n^{mc_\phi})^*$ to denote transitive closure of $\Sigma_n^{mc_\phi}$. 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 $\mathrm{SetVar}_U(i)$ is that the *i*th state of the Kripke structure gets label *U*, 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

$$\tau^{mc_\phi} = \{v\}, v \in \{\mathrm{true}, \mathrm{false}\}$$

and (3) an interpretation to the sequence of operations

is a partial function:

$$s_n^{mc_\phi} : u \mapsto \mathcal{A}_u^{mc_\phi}$$

where $u \in (\sum_n^{mc_\phi})^*, \mathcal{A}_u^{mc_\phi} \in \mathrm{Struc}_n[\tau^{mc_\phi}]$, and $s_n^{mc_\phi} \vDash v$ if and only if ϕ holds in the Kripke structure with state set $\{0,1,\cdots,n-1\}$, 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:

$$\mathrm{DMC} = \langle \sum^{mc_\phi}, \tau^{mc_\phi}, s_n^{mc_\phi} \rangle$$

where $\sum^{mc_\phi}$ is a set of operators that can be used to change the model, τ^{mc_ϕ} is a set of solution to model checking, i.e. $\{\mathrm{true}, \mathrm{false}\}$. It is *true* when $M \vDash \phi$ holds, and false when others, and $s_n^{mc_\phi}$ 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 ϕ as usual.

According to the different set of operators ($\sum^{mc_\phi}$), different kinds of problem of dynamic model checking could be formalized. In this paper, the iterative FCOS design is considered. Let $\sum^{mc_\phi} = \{\mathrm{Insert}, \mathrm{Delete}\} \cup \{\mathrm{SetVar}_U^{(1)}\}$, 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 $\mathrm{Dom}(s)$.

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

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 ith signal is a function $p_i(c(E)) = v_i, v_i \in \mathrm{Dom}(s_i), s_i \in E$.

Moreover, we denote $p_I(c(E)) = (v_1, v_2, \cdots, v_m)$, for $1 \leqslant k \leqslant m$, $v_k \in \mathrm{Dom}(s_k)$, $s_k \in I, I \subseteq E$ as projection of a set I of signals, and denote $p_{\bar{i}}(c(E)) = (v_1, v_2, \cdots, v_{i-1}, v_{i+1}, \cdots, v_n)$, $v_j \in \mathrm{Dom}(s_j)$, $j \neq i$ as a

sub-configuration without *i*th signal participating in. We also denote $p_{\bar{I}}(c(E))$ as a sub-configuration without set I of signals participating in, and denote $p_{\bar{I}}(C(E))$ 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 *i*th 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

$$W_i = \langle S_i, I_i, O_i, T_i, L_i, s_{i_0} \rangle$$

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, $\forall s \in S_i, \forall c \in C(I_i), \quad \exists! s' \in S_i s.t.$ $(s, c, s') \in T_i$ ($\exists!$ means there exists exactly one), $s_{i_0} \in S_i$ is the initial state, and $L_i = l_0, \cdots, l_{|O_i|-1}$ 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, $0 \leqslant j < |O_i|$, we have $l_j : S_i \to \mathrm{Dom}(o_j)$. 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

$$e = \langle \Delta I, C_{\mathrm{act}}(\Delta I), C_{\mathrm{qt}}(\Delta I) \rangle$$

where $\Delta I = I_+ \cup I_-$, I_+ is the set of added input signals with their definition domain, I_- is the set of deleted input signals, $I_+ \cap I_- = \varnothing$, i.e. signal name should not be reused. If $I_- \neq \varnothing$ then some input signals have been deleted. Assuming domain of all signals is $\{0,1\}$, if a signal $s_i \in I_-$ and $p_i(c(E)) = 1$, then configuration $c(E)$ should be already deleted, and configurations $c(E)$ should be changed into $p_{\bar{I}}(c(E))$. 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.

$C_{\mathrm{act}}(\Delta I)$ 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.

$C_{\mathrm{qt}}(\Delta I)$ 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_{ADD}, if $\Delta I = I_+$ in above definition of event *e*.

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

Obviously, we have $C_{\mathrm{act}}(\Delta I) \cup C_{\mathrm{qt}}(\Delta I) = C(\Delta I)$ and $C_{\mathrm{act}}(\Delta I) \cap C_{\mathrm{qt}}(\Delta I) = \varnothing$.

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, ΔT is the set of changed transition, and ΔO 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 compositive change (or general change) to W_i.

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

$$INC = \langle e_{\text{ADD}}, \Sigma_+, T_+, O_+ \rangle$$

Where e_{ADD} is the additional event described above, Σ_+ is the set of new reachable states, $\Sigma_+ \cap S_i = \varnothing$, $T_+ \subseteq (S_i \times C(I_i \cup I_+) \times S_i) \cup (\Sigma_+ \times C(I_i \cup I_+) \times \Sigma_+) \cup (S_i \times C(I_i \cup I_+) \times \Sigma_+) \cup (\Sigma_+ \times C(I_i \cup I_+) \times S_i)$, and O_+ is the set of new output signals and their definition domain, with $C_{\text{act}}(O_+)$, the set of configurations representing the activation of the output, and $C_{\text{qt}}(O_+)$, and the set of configurations representing the non-activation of the output. The output functions associated to O_+ return a configuration in $C_{\text{qt}}(O_+)$ for all states in S_i

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

$$\text{DEC} = \langle e, \Sigma_-, T_-, O_- \rangle$$

Where e_{DEL} is the deletional event described above, Σ_- is the set of states deleted by event e_{DEL}, $T_- \subseteq S_i \setminus \Sigma_- \times C(I_i \setminus I_-) \times S_i \setminus \Sigma_-$, where $S_i \setminus \Sigma_-$ is a set difference of S_i and Σ_-, $I_i \setminus I_-$ a set difference of I_i and I_-. $C(I_i \setminus I_-) = p_{\overline{I_-}}(C(I_i))$, and O_- is the set of output signals deleted, $O_- \subseteq O_i$.

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 $(s, C(I), p) \in T$, 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 ρ 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 $c' = c \wedge e\_qt, e\_qt \in C_{qt}(I_+)$. By the construction, ρ 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 ρ between the states of two consecutive components W_{i+1} and W_i, such that $\rho \subseteq S_{i+1} \times S_i : \forall (s, c, p) \in T_{i+1}$ and $(s', c', p') \in T_i$, we set $(s, s') \in \rho$ iff $s = s'$ and $c = p_{\overline{I_-}}(c')$. By the construction, ρ 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 (Σ^{mc_ϕ}) 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:

$$K(W_i) = \langle S_{K(W_i)}, s_{K(W_i),0}, \text{AP}_{K(W_i)}, L_{K(W_i)}, R_{K(W_i)} \rangle$$

Where $S_{K(W_i)} = S_i \times C(I_i)$, $s_{K(W_i),0} = s_i \times C(I_i)$, $\text{AP}_{K(W_i)} = I_i \cup O_i$, $L_{K(W_i)} = \{l_{I_0}, \cdots, l_{I_{|I_i|}-1}\} \cdot \{l_{O_0}, \cdots, l_{I_{|O_i|}-1}\}$ is a vector of $|\text{AP}_{K(W_i)}|$ function, and is vector concatenation, and $R_{K(W_i)} \subseteq S_{K(W_i)} \times S_{K(W_i)}$ and $\forall (s, c_i) \in S_{K(W_i)}, \forall (s', c_{i'}) \in S_{K(W_i)}$, $((s, c_i), (s', c_{i'})) \in R_{K(W_i)}$ iff $(s, c_i, s') \in T_i$.

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 Σ^{mc_ϕ}, although it is not in fact. We are interested in

whether a CTL specification formula ϕ 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 amd 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 $t''=(s'',c'')\in K(W_{i+1})$ such that (1) $s=s', c'=c\wedge e\_\mathrm{qt}$ and(2) $s=s'', c''=c\wedge e\_\mathrm{act}$, where $e\_\mathrm{qt}\in C_{\mathrm{qt}}(I_+), e\_\mathrm{act}\in C_{\mathrm{act}}(I_+)$, then t' and t'' 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 $s=s', c'=p_{I_-}(c)$, 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 $K(W_i), s\vDash\phi\rightarrow K(W_{i+1}), s'\vDash\phi$ with enrichment relation, if a CTL formula ϕ 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 $\neg f$, $(f\wedge g), EXf, E(fUg)$.

Proof: By induction on the structure of CTL formulae, let q be an atomic proposition, f, g be two CTL formulae, s_0 (resp. $s_{0'}$) be the initial state in $K(W_i)$ (resp. $K(W_{i+1})$).

- $\phi=q$. $K(W_i), s\vDash q$, so $q\in L_{K(W_i)}(s)$. By addition event to W_i, we have $q\wedge e\_qt\in L_{K(W_{i+1})}(s')$, so $q\in L_{K(W_{i+1})}(s')$, i.e. $K(W_{i+1}), s'\vDash q$.
- $\phi=f\wedge g$. By the semantics of CTL formulae, $K(W_i), s\vDash f$ and $K(W_i), s\vDash g$, then $f\in L_{K(W_i)}(s)$ and $g\in L_{K(W_i)}(s)$. Because s' enriches *s*, so $f\wedge e\_qt\in L_{K(W_{i+1})}(s')$ and $g\wedge e\_qt\in L_{K(W_{i+1})}(s')$, so $f\wedge g\in L_{K(W_{i+1})}(s')$, i.e. $K(W_{i+1}), s'\vDash f\wedge g$.
- $\phi=EXf$. Assume that $\exists s_0, s_1, \cdots$, $s_1\vDash f$, then $f\in L_{K(W_i)}(s_1)$, so $f\wedge e\_qt\in L_{K(W_{i+1})}(s_{1'})$. Suppose $s_{0'}$ enriches s_0, $s_{1'}$ enriches s_1, $\cdots$, etc. So there is a path $s_{0'}, s_{1'}, \cdots$, such that $f\in L_{K(W_{i+1})}(s_{1'})$, i.e. $K(W_{i+1}), s'\vDash EXf$.

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 $K(W_{i+1}), s'\vDash\phi\rightarrow K(W_i), s\vDash\phi$ with impoverishment relation, if a CTL formula ϕ 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 $\neg f, (f\wedge g), EXf, E(fUg)$.

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

$$K(W_i), s\nvDash\phi\rightarrow K(W_{i+1}), s'\nvDash\phi$$

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 (impoverishing 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

[1] HOROWITZ M, STARK D, ALON E. Digital circuit design trends. Solid-State Circuits[J]. IEEE Journal, 2008, 43(4): 757-761.

[2] RABAEY J M, CHANDRAKASAN A, NIKOLIC B. Digital integrated circuits: a design perspective[M]. 2nd edition. [S. l.]: Prentice Hall, 2003.

[3] WYNN D C, ECKERT C M, CLARKSON P J. Modelling iteration in engineering design. In ICED'07, 2007.

[4] HUTH M. Some current topics in model checking[J]. Int J Softw Tools Technol Transfer, 2007, 9(1): 25-36.

[5] BRAUNSTEIN C, ENCRENAZ E. Ctl-property transformations along an incremental design process[J]. Int J Softw Tools Technol Transf, 2007, 9(1): 77-88.

[6] HELJANKO K, JUNTTILA T, LATVALA T . Computer aided verification[J]. LNCS, 2005, 3576: 98-111.

[7] GUELEV D P, RYAN M D, SCHOBBENS P Y. Model-checking the preservation of temporal properties upon feature integration[J]. Int J Software Tools Technology Transf, 2007, 9(1): 53-62.

[8] ELKIND E, GENEST B, PELED D, et al. Formal techniques for networked and distributed systems[J]. LNCS, 2006, 4229: 420-435.

[9] RADMACHER F G, THOMAS W. A game theoretic approach to the analysis of dynamic networks[J]. Electron Notes Theor Comput Sci, 2008, 200(2): 21-37.

[10] MCMILLAN K L. Symbolic model checking[M]. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[11] KAHLER D, WILKE T. Program complexity of dynamic ltl model checking[J]. In CSL, 2003, 2803: 271-284.

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，在国际和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；已主持完成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、集成电路验证技术和专用集成系统设计及其应用。