基于动态 Kripke 语义直接模型检测及其应用分析*

陶志红[1+], Hans Kleine Büning[2], 丁德成[1]

[1](南京大学 数学系,江苏 南京 210093)

[2](Department of Computer Science, Paderborn University, Germany)

Direct Model Checking Based on Dynamic Kripke Structure and Its Application Analysis

TAO Zhi-Hong[1+], Hans Kleine Büning[2], DING De-Cheng[1]

[1](Department of Mathematics, Nanjing University, Nanjing 210093, China)

[2](Department of Computer Science, Paderborn University, Germany)

+ Corresponding author: Phn: +86-10-62754591, Fax: +86-10-62755413, E-mail: tzh@cs.pku.edu.cn, http://www.cs.pku.edu.cn

Received 2004-01-16; Accepted 2004-03-29

Tao ZH, Büning HK, Ding DC. Direct model checking based on dynamic Kripke structure and its application analysis. *Journal of Software*, 2004,15(Suppl.):83~89.

Abstract: During the past two decades Model Checking based on Kripke Semantics Structure has proven its efficacy and powerful in circuit design, network protocol analysis, program verification and bug hunting. Recently there has been considerable research on Model Checking without OBDDs such as using SAT Solver or Bounded Model Checking (BMC). A Dynamic Kripke Semantics Structure is introduced through allow the AP set changed. Based on this method, a new direct model checking algorithm is proposed.

Key words: model checking; dynamic Kripke semantics structure; CTL logic

摘 要: 在过去的20年里,基于Kripke语义结构的模型检测技术在集成电路设计,网络协议分析,程序正确性验证及程序错误发现等方面证明了其有效性和能力.最近,在诸如使用SAT分析工具、有界模型检测等避免OBDDs的模型检测研究方面取得了相当大的进展.提出的动态Kripke语义结构是让原子命题集合AP可以改变.基于这个方法,提出了一个直接模型检测算法.

关键词: 模型检测;动态Kripke语义结构;分支树逻辑

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA113171 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312006 (国家重点基础研究发展规划(973))

TAO Zhi-Hong was born in 1965. He is a post-PhD at Department of Computer Science, Peking University. His current research interests include CBSD, run-time model checking, computational modeling analysis and component currency. **Hans Kleine Büning** was born in 1948. He is a professor at Department of Computer Science, Paderborn University. His current research interests include software engineering and artificial intelligence. **DING De-Cheng** was born in 1943. He is a professor at Department of Mathematics, Nanjing University. His current research interests include computer science logic.

1 Introduction

Given a dynamic semantic Kripke structure $M=(S,R,L,AP)$ that represents a finite state concurrent system and a temporal logic formula f expressing some desired specification, find the sets in S that satisfy M, $s|=f$ and this is called the model checking problem. We notate $Mf=\{s\in S|M,s|=f\}$. Compared to other formal verification techniques (e.g. theorem proving) model checking is largely automatic. In run-time environment, the transition is obtained dynamically. The expansion of the model satisfying the specification may not keep the semantic consistency. Thus the semantic interpretations of different models under the same state are not consistency. The aim by introducing dynamic Kripke semantic is to introduce a dynamic view of true and false into the verifying of the design of hardware or software. This paper is organized as follows. In the following section we introduce the definition of the dynamic semantic Kripke structure. In Section 3 we give an introduction to CTL*[7~10,14] logic and CTL logic. In Section 4 we give our new direct CTL model checking algorithm. In Section 5 we conclude the paper with some application examples.

2 The Dynamic Semantic Kripke Structure

Definition 2.1. Let AP be a set of atomic proposition. A Kripke structure M over AP is a four triple $M=(S,R,L,AP)$ where

1. S is a finite set of states.
2. $S_0\subseteq S$ is the set of initial states.
3. $R\subseteq S\times S$ is a transition relation that must be total, that is, for every state $s\in S$ there is a state $s'\in S$ such that $R(s,s')$.
4. $L: S\to 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.

$L(s)=\{p|p\in AP$ and p is true in the state $s\}$. $Mp=\{s|s\in S$ and p is true in the state $s\}$.

Definition 2.2. Let $M=(S,R,L,AP)$ be a Kripke structure, a path in the structure M from a state s is an infinite sequences $\pi=s_0s_1s_2\ldots$ such that $s=s_0$ and $R(s_i,s_{i+1})$ holds for all $i>0$. We use π to denote the suffix of π starting at s_i, that is, $\pi_i=s_is_{i+1}s_{i+2}\ldots$

In the traditional Kripke Semantics Structure the set of atomic proposition AP can't be changed. Here the set of atomic proposition AP can be changed dynamically, that is, the new atomic proposition p can be put into AP, so we get the new set of atomic proposition AP_1 and a new Kripke structure $M_1=(S,S_0,R,L_1)$ where

1. $AP_1=AP\cup\{p\}$
2. $\forall\ q\in AP\subseteq AP_1$, $Mq=M_1q$
3. $\forall\ s\in S$, $L(s)=L_1(s)/\{p\}$

Lemma 2.3. The above Kripke structure M_1 is consistency dynamic expansion of the Kripke structure M. This kind of Kripke structure is called the Kripke structure with dynamic semantics, notated: $M\subseteq M_1$. Later the dynamic Kripke structure is notated: $M=(S,S_0,R,L,AP)$ in order to emphasize the importance of the set of atomic proposition.

3 The Computation Tree Logic CTL*

Conceptually, CTL* formulas describe properties of computation trees. Designating a state in a Kripke structure as the initial state forms the tree and then unwinding the structure into an infinite tree with the designated state at the root, as illustrated in Fig.1. The computation tree shows all of the possible executions starting from the initial state.

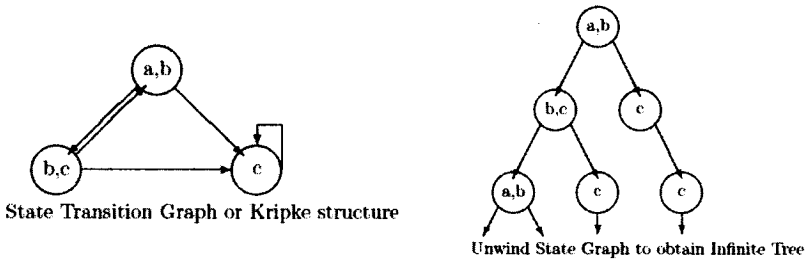

Fig.1

In CTL* formulas are composed of path quantifiers and temporal operators. The path quantifiers are used to describe the branching structure in the computation tree. There are two such quantifiers A (for all computation paths) and E (for some computation tree path). These quantifiers are used in a particular state to specify that all of the paths or some of the paths starting at that state have some property. The temporal operators describe properties of a path through the tree. There are five basic operators:

X (next time) requires that a property hold in the second state of the path.

F (eventually or in the future) operator is used to assert that a property will hold at some state on the path.

G (always or globally) specifies that a property hold at every state on the path

U (until) operator is a little bit more complicated since it is used to combine two properties. It holds if there is a state on the path where the second property holds, and at every preceding state on the path, the first property holds.

R (release) is the logical dual of the U operator. It requires that the second property hold along the path up to and including the first state where the first property holds. However, the first property is not required to hold eventually.

There are two types of formulas in CTL*: state formulas (which are true in a specific state) and path formulas (which are true along a specific path). Let AP be the set of atomic proposition names. The syntax of state formulas is given by the following rules:

If $p \in AP$, then p is a state formula. If f and g are state formulas, then $\neg f, f \wedge g, f \vee g$ are state formulas.

If f is a path formula, then Ef and Af are state formulas. Two additional rules are needed to specify the syntax of path formulas:

If f is a state formula, then f is also a path formula.

If f and g are path formulas, then $\neg f, f \wedge g, f \vee g$, X$f$, F$f$, G$f$, fUg and fRg are path formulas.

CTL* is the set of state formulas by the above rules. We define the semantics of CTL* with respect to a Kripke structure M. Recall that a path in M is an infinite sequence of states, $\pi = s_0 s_1 s_2 \ldots$ such that for every $i > 0$, $(s, s') \in R$ We use π^i to denote the suffix of π starting at s_i. If f is a state formula, the notation $M, s \models f$ means that f holds at state s in the Kripke structure M. Similarly, if f is a path formula, $M, \pi \models f$ means f holds along path π in the Kripke structure M. The relation $\models$ is defined inductively as follows (assuming that f_1 and f_2 are state formulas and g_1 and g_2 are path formulas):

1. $M, s \models p \Leftrightarrow p \in L(s)$
2. $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$
3. $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$
4. $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$
5. $M, s \models E g_1 \Leftrightarrow$ there is a path π from s such that $M, \pi \models g_1$
6. $M, s \models A g_1 \Leftrightarrow$ for all path π starting from s, $M, \pi \models g_1$
7. $M, \pi \models f_1 \Leftrightarrow s$ is the first state of π and $M, s \models f_1$

8. $M, \pi \models \neg g_1 \Leftrightarrow M, \pi \not\models g_1$

9. $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$ or $M, \pi \models g_2$

10. $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, \pi \models g_1$ and $M, \pi \models g_2$

11. $M, \pi \models Xg_1 \Leftrightarrow M, \pi_1 \models g_1$ if $\pi = s_0 s_1 s_2 \ldots$ then $\pi^1 = s_1 s_2 s_3 \ldots$

12. $M, \pi \models Fg_1 \Leftrightarrow$ there exists a $k \geq 0$, such that $M, \pi^k \models g_1$

13. $M, \pi \models Gg_1 \Leftrightarrow$ for all $k \geq 0$, such that $M, \pi^k \models g_1$

14. $M, \pi \models g_1 U g_2 \Leftrightarrow$ there is a $k \geq 0$, such that $M, \pi^k \models g_2$, and for all $0 \leq j < k$, $M, \pi \models g_1$

15. $M, \pi \models g_1 R g_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i < j$, $M, \pi \not\models g_1$ then $M, \pi \models g_2$

Computation Tree Logic (CTL) is a restricted subset of CTL* in which each of the temporal operator *X*, *F*, *G*, *U*, and *R* must be immediately preceded by a path quantifier. More precisely, CTL is the subset of CTL* that is obtained by restricting the syntax of path formulas using the following rule:

If *f* and *g* are state formulas, then *Xf*, *Ff*, *Gf*, *f Ug*, and *f Rg* are path formulas. In the specification described by CTL logic there are ten basic CTL operators: AX and EX, AF and EF, AG and EG, AU and EU, AR and ER. Each of ten operators can be expressed in terms of three operators EX, EG, and EU:

$AXf \equiv \neg EX(\neg f)$,

$EFf \equiv E[TrueUf]$

$AGf \equiv \neg EF(\neg f)$

$AFf \equiv \neg EG(\neg f)$

$A[fUg] \equiv \neg E[\neg gU(\neg f \wedge \neg g)] \wedge \neg EG\neg g$

$A[fRg] \equiv \neg E[\neg fU\neg g]$

$E[fRg] \equiv \neg A[\neg fU\neg g]$

4 Direct Model Checking Algorithm Based on Dynamic Kripke Semantic Structure

Given a dynamic semantic Kripke structure $M = (S, R, L, AP)$ and assume that $S = \{s_1, s_2, s_3, \ldots, s_n\}$ and $AP = \{p_1, p_2, \ldots, p_m\}$. We use a matrix to represent *R* and a vector to represent M*f*

$R = (r[i,j])_{n \times n}$ if $R(si,sj)$ is true then $r[i,j] = 1$ otherwise $r[i,j] = 0$

$Mf = (m[i])_n$, if *f* is true in state *si* then $m[i] = 1$ otherwise $m[i] = 0$

Especially to the atomic proposition *pj*, $Mpj = (m[i])_n$ if $pj \in L(si)$ then $m[i] = 1$ otherwise $m[i] = 0$

$I_0 = \langle Mp_1, Mp_2, Mp_3, \ldots, Mp_m \rangle$

When a dynamic semantic model $M = (S, R, L, AP)$ is given *R*, *Mpj*, I_0 is determined. If a new atomic proposition, for example p_{m+1}, is needed to add to *AP*, a new model M_1 is got. When we want to calculate $Mf = \{s \in S \mid M, s \models f\}$ we start from the shortest, most deeply nested sub formula of *f* and work outward to include all sub formulas of *f*. For example, when the sub formula f_1 has been processed, we have got Mf_1. Then we use a new atomic proposition, e.g. p_{m+1}, to replace f_1 in *f* and get a new formula $f[p_{m+1} \to f_1]$. Therefore the satisfaction of the new formula $f[p_{m+1} \to f_1]$ in model M_1 is consistency with the satisfaction of *f* in model M. When we get a new model M_1 from *M* by setting $AP_1 = AP \cup \{p_{m+1}\}$ the satisfied states set are not changed, i.e. $M, s \models f$ iff. $M, s \models f[p_{m+1} \to f_1]$ (Notice: the set of states S and the transition R are never be changed). Thus we get a model checking algorithm.

Theorem 4.1. Given a dynamic semantic Kripke structure $M = (S, R, L, AP)$ and a CTL formula *f*. The Kripke Structure M_1 is the dynamic semantic expansion of *M* with respect to *f* by adding a new atomic proposition $pm+1$ to AP. f_1 is the sub formula of *f*. $Mpm+1 = Mf'$, $f' = f[pm+1 \to f_1]$. We have that M, s| = *f* if and only if $M_1, s \models f'$.

Proof. We omit its proof.

Theorem 4.2. Given a Kripke structure $M = (S, R, L, AP)$ and a CTL formula *f*, there is a direct algorithm to compute the set of states $Mf = \{s \in S \mid M, s \models f\}$ which is based on the dynamic semantic expansion of *M*.

Proof. Recall that any CTL formula can be expressed in terms of $\neg, \vee$, EX, EU, and EG. Thus, for the intermediate stages of the algorithm it is sufficient to be able to handle six cases, depending on whether g is an atomic p or has one of the following forms: $\neg f1$, $f1 \vee f2$, EX$f1$, E[$f1$U$f2$], EG$f1$.

Step1. If f is an atomic proposition $p \in AP$, we have that $Mf = Mp = \{s \mid s \in S \wedge M, s \models p\} = \{s \mid s \in S \wedge p \in L(s)\}$. Now we prove the theorem inductively and assume that any sub formula of f has been processed, i.e., for the sub formula g of f we have $Mg = \{s \in S \mid M, s \models g\}$

Step2. $f = \neg f1$ by the inductive assume, we have $Mf1 = \{s \in S \mid M, s \models f1\}$. According to the semantic interpretation $Mf = \{s \in S \mid M, s \not\models f\} = S \setminus Mf1$

Step3. $f = f1 \vee f2$ by the inductive assume $Mf1$, $Mf2$, have been got. According to semantic interpretation we have

$$\begin{aligned} Mf = Mf1 \cup Mf2 &= \{s \in S \mid M, s \models f1\} \cup \{s \in S \mid M, s \models f2\} \\ &= \{s \in S \mid M, s \models f1 \text{ or } M, s \models f2\} \\ &= \{s \in S \mid M, s \models f1 \vee f2\} \\ &= \{s \in S \mid M, s \models f\} \end{aligned}$$

Step4. $f = EXf1$ by the inductive assume $Mf1 = \{ s \in S \mid M, s \models f1 \}$ has been got. Now we construct the expansion of the model M. We introduce a new atomic proposition $pf1$ (replacing $f1$ by $pf1$) and get AP$final$ =AP U $\{pf1\}$. Thus $Mf1 = Mpf1$. By the Theorem 4.1 $M, s \models f$ if and only if $M1, s \models EXpf1$. The transition R and $Mpj, 1<j<m$ in the model $M1$ are the same as those in M. In $M1$

$I1 = \langle Mp1, Mp2, Mp3, \ldots, Mpm, Mpf1 \rangle$

$Mf = MEXpf1 = \{s \in S \mid M1, s \models EXpf1 \} = \{t \in S \mid \exists s \in S(R(t,s) \wedge s \in Mpf1) \}$

Step5. $f = E[f1 \mathrm{U} f2]$ according to the inductive assume $Mf1 = \{ s \in S \mid M, s \models f1 \}$, $Mf2 = \{ s \in S \mid M, s \models f2 \}$, have been obtained. We now introduce two new atomic propositions $pf1$, $pf2$ to construct the new model M1. We get that $APfinal = AP\ U\ \{pf1, pf2\}$. Thus $Mf1 = Mpf1$ and $Mf2 = Mpf2$. By the Theorem 4.1 $M, s \models f$ if and only if

$M1, s \models E[pf1 \mathrm{U} pf2]$. The transition R and $Mpj, 1<j<m$ in the model $M1$ are the same as those in M. In $M1$:

$I1 = \langle Mp1, Mp2, Mp3, \ldots, Mpm, Mpf1, Mpf2 \rangle$

For calculating $E[f1 \mathrm{U} f2]$ we first find all states that satisfying $f2$. We then backwards using the converse of the transition relation R and find all states can be reached by path in which each state satisfying $f1$. All such states should satisfy E[$f1$U$f2$]. Following is the procedure CheckEU finding all the states satisfying E[$f1$U$f2$]

```
Procedure CheckEU(pf1, pf2)
T:= M pf2; We call T working set and let ME[pf1Upf2] = Ø
For all s∈T do computing ME[pf1Upf2] in Kripke model M1
    While T ≠ Ø do
        Choose s∈T
        T:=T\{s}
        For all t such that R(t,s) do
            If t∈Mpf1 and t∉ME[pf1Upf2] then
              ME[pf1Upf2]:= ME[pf1Upf2] ∪ {t}
              T: = T ∪ {t}
            End if
        End for all
    End while
Return: ME[pf1Upf2]
End Procedure.
```

Step6. $f = EGf1$ according to the inductive assume $Mf1 = \{ s \in S \mid M, s \models f1 \}$ has been obtained. We now introduce a new atomic proposition $pf1$ to construct the new model $M1$. We get that $APfinal = AP\ U\ \{pf1\}$ Thus $Mf1 = M\ pf1$. By Theorem 4.1 $M, s \models f$ if and only $M1, s \models EGpf1$. The transition R and $Mpj, 1<j<m$ in the model $M1$ are the same as those in M. In $M1$

$I1 = \langle Mp1, Mp2, Mp3, \ldots, Mpm, M\ pf1 \rangle$

Let M' be obtained from M_1 by deleting from S all of those states at which f_1 does not hold and restricting R and L accordingly. Thus, $M1'=(S', R', L')$ where $S'= Mf1$, $L'=$ L|S' and R'=R|S'X S'. Note that R may not be total in this case. The algorithm depends on the following observation.

```
Procedure CheckEG(pf1)
    S': = M pf1 = (Mf1)
    SCC: = {C | C is a nontrivial SCC of S'}
     T:= ∪C∈SCC{s| s∈C} we call T working set and let MEG(pf1) = Ø
     For all s∈T do computing MEG(pf1) in Kripke model M1'
          While T≠Ø do
            Choose s∈T
                  T:=T\{s}
               For all t such that t∈S' and R(t,s) do
                     If t∉MEG(pf1) then
                     MEG(pf1):= MEG(pf1) ∪{t}
                     T:=T∪{t}
                    End if
               End for all
          End while
    End for all
  Return: MEG(pf1)
End procedure.
```

Thus the proof of the Theorem 4.2 has been finished.

5 Application Analyses and the Conclusion

Over the past 15 years different workflow techniques and CBSD (component based software development) methods have been proposed. How to verify large software systems especially in the run-time environment developed by the above methods is become more and more complicated and critical. In this paper we introduced a direct algorithm for CTL model checking based on the Dynamic Kripke Semantics Structure. We plan to use this algorithm to modeling the software systems developed by CBSD and research component currency problem. We believe that this work will be useful in the analysis of Workflow related software systems.

References:

[1] Aho AV, Hopcroft JE, Ullman JD. The Design and Analysis of Computer Algorithms Addison Wesley, 1974.

[2] Audemard G, Bertoli P, Cimatti A, Kornilowicz A, Sebastiani R. A sat based approach for solving formulas over boolean and linear mathematical propositions. In: Proc. of the 18th Int'l Conf. of Automated Deduction (CADE2002). LNAI, Copenhagen, July 2002. Springer-Verlag, 2003.

[3] Ben-Ari M, Manna Z, Pnueli A. The temporal logic of branching time. Acta Information, 1983,20:207~226.

[4] Biere A, Cimatti A, Clarke EM, Fujita M, Zhu Y. Symbolic Model Checking using SAT procedures instead of BDDs ACM, 1999.

[5] Biere A, Cimatti A, Clarke E, Zhu Y. Symbolic model checking without BDDs. In: Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems(TACAS'99). Springer-Verlag, 1999.

[6] Bjesse P, Leonard T, Mokkedem A. Finding bugs in an alpha microprocessor using satisfiability solvers. In: Berry G, Comon H, Finkel A, eds. Proc. of the 12th Int'l Conf. on Computer Aided Verification (CAV'01) Springer-Verlag, 2001.

[7] Burch JR, Clarke EM, McMillan KL. Symbolic model checking: 10^{20} states and beyond. Information and Computation, 1992,98:142~170.

[8] Clarke EM, Emerson EA. Design and synthesis of synchronization skeletons using branching time temporal logic.In: Logic of Programs: Workshop. Yorktoen Heights: Springer, 1981.

[9] Clarke EM, Emerson EA, Sistla AP. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. on Programming Languages and Systems, 8(2):244~263.

[10] Clarke EM, Grumberg O, Peled DA. Model Checking. Cambridge: The MIT Press, 2000.

[11] Clarke E, Biere A, Raimi R, Zhu Y. Bounded model checking using satisfiability solving. Formal Methods in System Design, 2001,19(1):7~34.

[12] Emerson EA, Clarke EM. Characterizing correctness properties of parallel programs using fixpoints. In: Automata, Languages and Programming. Springer, 1980. 169~181.

[13] Emerson EA, Halpern JY. "Sometimes" and "Not Never" revisited: On branching time versus linear time. Journal of the ACM, 33:151~178.

[14] McMillan KL. Symbolic Model Checking: An Approach to the State Explosion Problem. Kluwer Academic Publishers, 1993.

[15] Holzmann G. The Spin model checker. IEEE Trans. Software Engineering, 1997,23(5):279~295.