Direct Model Checking Matrix Algorithm

Zhi-Hong Tao1 (陶志红), Hans Kleine Büning2, and Li-Fu Wang3 (王立福)

1Department of Computer Science and Technology, Peking University, Beijing 100871, P.R. China
2Department of Computer Science, Paderborn University, D-33095, Germany
E-mail: tzh@cs.pku.edu.cn

Received December 17, 2004; revised January 18, 2006.

Abstract During the last decade, Model Checking has proven its efficacy and power in circuit design, network protocol analysis and bug hunting. Recent research on automatic verification has shown that no single model-checking technique has the edge over all others in all application areas. So, it is very difficult to determine which technique is the most suitable for a given model. It is thus sensible to apply different techniques to the same model. However, this is a very tedious and time-consuming task, for each algorithm uses its own description language. Applying Model Checking in software design and verification has been proved very difficult. Software architectures (SA) are engineering artifacts that provide high-level and abstract descriptions of complex software systems. In this paper a Direct Model Checking (DMC) method based on Kripke Structure and Matrix Algorithm is provided. Combined and integrated with domain specific software architecture description languages (ADLs), DMC can be used for computing consistency and other critical properties.

Keywords direct model checking (DMC), Kripke semantics structure, CTL logic, matrix algorithm

1 Introduction

In the last decade, Model Checking[1−7] has proven to be a valuable technique in the design of digital circuits and communication protocols. Famous tools such as SMV[8], SPIN/Promela[9] and CSP/FDR are very useful in hardware verification. Model Checking requires a number of steps. From the beginning of the formal representation of a system under designing, e.g., an ADL description, a correct abstraction model from this design must be defined in the input language of the model checker. Often this requires a translation from the problem domain to the concepts used in the model checker. This model must be validated in order to ensure that no mistakes are introduced by the abstraction. Thereafter, the correctness requirements must be formulated in the corresponding requirement language such as temporal logic formulae. Finally, the specifications can be checked for their satisfaction of the requirements. If the requirements are not satisfied, both the requirement specifications and the system model must be checked: either the system does not satisfy the requirement, the requirement is not defined correctly, or the model is an incorrect abstraction of the system. To do so, we must translate the counter example back to the application domain. The above means of model checking is a complex and cumbersome: designing a system is not easy, developing specifications is a complex task and defining the right correctness requirements must be done carefully. Model Checking is a complex and time-consuming activity for skilled computer scientists and engineers.

Recent research on automatic verification has shown that no single model-checking technique has the edge over all others in all application areas. So, it is very difficult to determine which technique is the most suit-
algorithm. A case study of DMC will be given in Section 5 and Section 6 concludes the paper with some directions for future research.

2 Kripke Structure and Computer Tree Logic (ctl)

Definition 1 (Kripke Structure with Two Fixed Atomic Propositions). Let AP = \{p_1, p_2, \ldots, p_m, \overline{p}_m\} be a set of atomic propositions. A Kripke structure M over AP is a four triple M = (S, R, L, AP) where:

1) S = \{s_1, s_2, s_3, \ldots, s_n\} 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 which 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.

\[ L(s) = \{ p \in AP \text{ and } p \text{ is true in state } s \} \]

\[ M_0 = \{ s | s \in S \text{ and } p \text{ is true in state } s \} \]

Now we introduce a matrix and vector representation:

\[ R = (r[i, j])_{n \times n}, \text{ if } R(s_i, s_j) \text{ is true then } r[i, j] = 1, \text{ otherwise } r[i, j] = 0 \]

\[ M_p = (m[s])_n \text{ if } p \in L(s_i) \text{ then } m[s] = 1, \text{ otherwise } m[s] = 0 \]

\[ M_{\text{true}} = \{1, \ldots, 1\}_n, \text{ otherwise } M_{\text{false}} = \{0, \ldots, 0\}_n \]

\[ I = (M_0, M_{p_1}, \ldots, M_{p_m}, M_{\overline{p}_m}) \]

Definition 2. Let M = (S, R, L, AP) be a Kripke structure, a path in the structure M from a state s is an infinite sequence \(s = s_0 s_1 s_2 \cdots\), such that \(s = s_0\) and \(R(s_i, s_{i+1})\) holds for all \(i \geq 0\). We use \(s_i\) to denote the suffix of \(s\) starting from \(s_i\). If \(s\) is a state formula, the notation \(M, s = f\) means that \(f\) holds at state \(s\) in the Kripke structure \(M\). Similarly, if \(f\) is a path formula, \(M, p = f\) means \(f\) holds along path \(p\) in the Kripke structure \(M\). The relation \(=\) 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) \(M, s = p \iff p \in L(s)\)
2) \(M, s \not= \neg f_1 \iff M, s \not= f_1\)
3) \(M, s \not= f_1 \land f_2 \iff M, s \not= f_1 \text{ or } M, s \not= f_2\)
4) \(M, s \not= f_1 \lor f_2 \iff M, s \not= f_1 \text{ and } M, s \not= f_2\)
5) \(M, s \not= E_{g_1} \iff \text{ there is a path } \pi \text{ from } s \text{ such that } M, \pi = g_1\)
6) \(M, s \not= A_{g_1} \iff \text{ for all path } \pi \text{ starting from } s, M, \pi = g_1\)
7) \(M, \pi \not= f_1 \iff s \text{ is the first state of } p \text{ and } M, s \not= f_1\)
8) \(M, \pi \not= g_1 \iff M, \pi \not= g_1\)
9) \(M, \pi \not= g_1 \land g_2 \iff M, \pi \not= g_1 \text{ or } M, \pi \not= g_2\)
10) \(M, \pi \not= g_1 \lor g_2 \iff M, \pi \not= g_1 \text{ and } M, \pi \not= g_2\)
11) \(M, \pi \not= X_{g_1} \iff M, \pi = g_1\)
12) \(M, \pi \not= F_{g_1} \iff \text{ there is a } k \geq 0 \text{ such that } M, \pi_k = g_1\)
13) \(M, \pi \not= G_{g_1} \iff \text{ for all } i \geq 0, M, \pi_i \not= g_1\)
14) \(M, \pi \not= g_1 U g_2 \iff \text{ there is a } k \geq 0 \text{ such that } M, \pi_k \not= g_2 \text{ and for all } 0 \leq j < k, M, \pi_j \not= g_1\)
15) \(M, \pi \not= g_1 RG_2 \iff \text{ for all } j \geq 0, \text{ if for every } i < j, M, \pi_i \not= g_1 \text{ then } M, \pi_j \not= g_2\)

It is easy to see that the operators \(\lor, \neg, X, U, \text{ and } E\) are sufficient to express any other \(\text{ctl}^*\) formula. \(f \land g \equiv \neg(\neg f \lor \neg g)\), \(fRg \equiv \neg f \land \neg g\), \(Ff \equiv (\text{true } U f)\), \(Gf \equiv \neg F\neg f\) and \(Af \equiv E\neg f\).

Computation Tree Logic (ctl) is a restricted subset of \(\text{ctl}^*\) in which each of the temporal operators \(X, F, G, U, \text{ and } R\) must be immediately preceded by a path quantifier. More precisely, \(\text{ctl}\) is the subset of \(\text{ctl}^*\) which is obtained by restricting the syntax of path formulas using the following rule.