SCIENCE CHINA
Information Sciences

• INSIGHT •

November 2016, Vol. 59 118101:1–118101:3
doi: 10.1007/s11432-015-0882-6

Model checking concurrent systems with MSVL

Nan ZHANG, Zhenhua DUAN* & Cong TIAN*

Institute of Computing Theory and Technology, Xidian University, Xi'an 710071*, China*

Received March 3, 2016; accepted April 22, 2016; published online October 10, 2016

Citation Zhang N, Duan Z H, Tian C. Model checking concurrent systems with MSVL. Sci China Inf Sci, 2016, 59(11): 118101, doi: 10.1007/s11432-015-0882-6

Model checking, proposed by Clarke and Emerson [1] as well as Queille and Sifakis [2], is an automatic verification approach for hardware and software systems. However, as Clarke pointed out [3], model checking suffers from (1) the state explosion problem, which is typically caused by models growing exponentially in the number of parallel components or data elements of an argument system; (2) different notations used to model a system and required properties; (3) the expressive power of most often used temporal logics such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) being weak. To conquer these problems, computer scientists have made significant progress to the original model checking. The most significant improvements are compositional, partial order [4], symbolic [5], bounded [6] and abstract model checking [7, 8].

The Unified Model Checking (UMC) approach proposed in [9] is based on Projection Temporal Logic (PTL) [10]. An outline of the algorithm is shown in Figure 1(a). With this approach, both system models and desired properties are described by the same formal language, namely, Modeling, Simulation and Verification Language (MSVL) [10] which is a subset of PTL. To verify whether or not a system satisfies a property, the system is modeled as an MSVL program P and the property is specified by a formula ϕ of Propositional PTL (PPTL) which is the propositional subset of PTL [10] or an MSVL program. To check whether P satisfies ϕ amounts to proving whether $P \rightarrow \phi$ is valid ($\models P \rightarrow \phi$), which is equivalent to proving unsatisfiability of $P \wedge \neg\phi$. Thus, the model checking problem is equivalent to the satisfiability problem in PPTL since any finite state program in MSVL such as a hardware system description is equivalent to a PPTL formula [11]. However, the current version of MSVL has been extended with plenty of complex types such as float. As a result, an MSVL program may not be a finite state program. That is the state space of an MSVL program may be infinite. To abstract an MSVL program as a finite state program, Counter Example Guarded Abstract Refinement (CEGAR) [12] and Dynamic Symbolic Execution (DSE) methods are employed. Furthermore, in [13, 14], we have proved that PPTL is decidable and given decision procedures. With these procedures, a PPTL formula is satisfiable if and only if there is a valid path in its Labeled Normal Form Graph (LNFG). Therefore, the problem of checking whether or not P satisfies ϕ is eventually turned to the problem of checking whether or not the LNFG of $P \wedge \neg\phi$ contains a valid finite or infinite path. If not, the property is valid otherwise an acceptable path of the LNFG determines a counter example.

C programming language is popular in the world. However, C programs are complicated and error prone. How to verify C programs is a big

*Corresponding author (email: zhhduan@mail.xidian.edu.cn, ctian@mail.xidian.edu.cn)
The authors declare that they have no conflict of interest.

challenge to computer scientists and engineers. Although some verification tools such as CBMC, BLAST and SLAM, etc. [15–17] for C programs are available, their effectiveness and efficiency are limited. This encourages us to develop a technique and a translator for transforming a C program into an MSVL program. Thus, verification of a C program can be done by means of verifying an MSVL program.

Furthermore, Verilog and VHDL languages dominate hardware designs and descriptions. How to verify Verilog/VHDL programs is also a challenge since a lot of embedded real time systems are realized using Verilog/VHDL. To guarantee the correctness and reliability of these systems, lots of simulations and tests have to be conducted since very few verification tools are available. Therefore, we are motivated to formalize a mechanism and develop a translator to transform a Verilog/VHDL program to an MSVL program so that the verification of Verilog/VHDL programs can be done by meas of verifying MSVL programs.

In addition, we have proved that the expressiveness of PPTL is full regular [18], hence it subsumes LTL and CTL. As a result, two kinds of properties can now be specified and proved using PPTL which cannot be defined by LTL or CTL: (1) interval sensitive properties; for instance, a property R holding after 100 time units and before 200 time units over an interval can be defined by $\mathrm{len}(100), \Diamond R, \mathrm{len}(200)$; (2) periodically repeated properties; for example, even order property, that is, an atomic proposition Q holding at each even state ignoring odd ones over an interval can be defined by $(Q \wedge \bigcirc^2(Q \wedge \varepsilon))^*$. Since a PPTL formula can be transformed to an LNFG which can further be transformed to a Büchi Automaton (BA) [19], thus, this facilitates us to formalize and develop a partial order model checker PMC4PPTL based on SPIN [4], symbolic and bounded model checkers SMC4PPTL and BMC4PPTL based on SMV [5, 6], and 1-safe Petri Nets model checker PNMC4PPTL based on Reachability Graph (RG). Our method has some advantages. For instance, (1) the model and property of a system can be described in the same logic; (2) the model checking algorithm relies on constructing the LNFG of a PPTL formula and stops as soon as a valid finite or infinite path is constructed if we do not expect to have all counterexamples; (3) the existing SAT/SMT algorithms [20] can be reused to check the satisfaction of the state formulas with the present components of a normal form; (4) the expressiveness of PPTL is full regular [18] and more powerful than LTL and CTL.

The arithmetic and boolean expressions of MSVL can be inductively defined as follows [10, 21]:

$$e ::= \ c \mid x \mid \bigcirc x \mid \circleddash x \mid e_0 \ \mathrm{op} \ e_1,$$

$$b ::= \ \mathsf{true} \mid \mathsf{false} \mid \neg b \mid b_0 \wedge b_1 \mid e_0 = e_1 \mid e_0 < e_1,$$

where op ::= $+ \mid - \mid \times \mid$mod, c is a constant in the domain and x is a static or dynamic variable. One may refer to the value of a variable at the previous state or the next state. MSVL programs can be inductively defined by sixteen basic statements given in [10, 21]. A synchronization construct, $\mathsf{await}(c)$ is defined as $\mathsf{await}(c) \stackrel{\mathrm{def}}{=} \mathsf{halt}(c) \wedge \mathsf{frame}(x_1, x_2, \ldots, x_n)$. The $\mathsf{await}(c)$ does not change any variables, but waits until the condition c becomes true, at which point it terminates. Here, $x_1, x_2, \ldots, x_n$ are dynamic variables appeared in c.

Some common data types are included in MSVL. They are (unsigned) int, float, (unsigned) char, string, list, array, pointer, struct and union [22]. Moreover, typed functions and predicates can be defined [22]. There are two kinds of function callings in MSVL programs [23]: one is internal calling, in which the execution interval of the callee is inserted into the execution interval of the caller; the other is external calling, in which the execution interval of the callee is not inserted into the execution interval of the caller.

To support modeling, simulation and verification of a system using MSVL, we have developed a tool kit called MSV. The architecture of MSV is shown in Figure 1(b). MSV integrates four kinds of tools: (1) a modeling tool, used to construct an LNFG of a program; (2) a simulation tool, used to execute a program; (3) a group of model checkers consisting of unified, partial order, symbolic, bounded and abstraction model checkers (the common feature of these model checkers is that PPTL is employed to describe the desired properties); (4) several translators including C2MSVL, Veri2MSVL, VHDL2MSVL and PN2MSVL as shown in Figure 1(b). The MSV tool kit has successfully been used to verify several practical applications.

Basically, PTL is also a kind of interval based TLs. It extends both original interval temporal logic (ITL) proposed by Moszkowski and Choppy logic proposed by Barringer, Kuiper and Pnueli [24] in the following aspects as summarised by Bowman and Thompson [25]: Duan's work extends Moszkowszki's interval temporal logic in a number of respects: (1) past operators are added; (2) a new projection operator is defined; (3) framing of variables is investigated; (4) infinite models

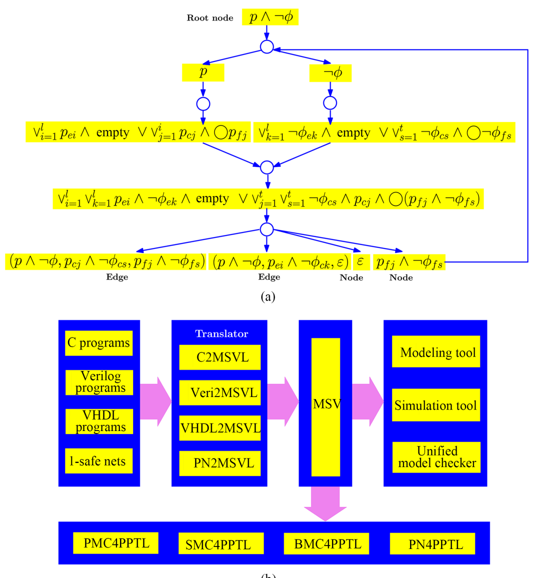
Root node
$p \wedge \neg\phi$
p
$\neg\phi$
$\bigvee_{i=1}^{l} p_{ei} \wedge$ empty $\vee \bigvee_{j=1}^{i} p_{cj} \wedge \bigcirc p_{fj}$
$\bigvee_{k=1}^{l} \neg\phi_{ek} \wedge$ empty $\vee \bigvee_{s=1}^{t} \neg\phi_{cs} \wedge \bigcirc\neg\phi_{fs}$
$\bigvee_{i=1}^{l}\bigvee_{k=1}^{l} p_{ei} \wedge \neg\phi_{ek} \wedge$ empty $\vee \bigvee_{j=1}^{t}\bigvee_{s=1}^{t} \neg\phi_{cs} \wedge p_{cj} \wedge \bigcirc(p_{fj} \wedge \neg\phi_{fs})$
$(p \wedge \neg\phi, p_{cj} \wedge \neg\phi_{cs}, p_{fj} \wedge \neg\phi_{fs})$
Edge
$(p \wedge \neg\phi, p_{ei} \wedge \neg\phi_{ck}, \varepsilon)$
Edge
ε
Node
$p_{fj} \wedge \neg\phi_{fs}$
Node
(a)
C programs
Verilog programs
VHDL programs
1-safe nets
Translator
C2MSVL
Veri2MSVL
VHDL2MSVL
PN2MSVL
MSV
Modeling tool
Simulation tool
Unified model checker
PMC4PPTL
SMC4PPTL
BMC4PPTL
PN4PPTL
(b)

Figure 1 (Color online) (a) MSVL unified model checking; (b) MSV toolkit.

are incorporated and (5) concurrency and communication primitives are considered. However, as far as we know, their work is focused on theorem proving rather than model checking. MSVL is a temporal logic programming language which is closely related to XYZ/E [26] and Tempura [27]. Tang first proposed the idea of temporal logic programming and designed XYZ/E system based on LTL [26]. Tempura is an executable subset of ITL and designed by Moszkowski [27]. Both XYZ/E and Tempura are mainly used for the purpose of programming and theorem proving rather than model checking. Compared with Tempura, MSVL creates several useful techniques including framing, data types such as pointer, struct and union, as well as external function calling mechanism in order to make MSVL effective in practice.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61133001, 61420106004, 91418201).

References

1 Clarke E M, Emerson E A. Design and synthesis of synchronization skeletons using branching time temporal logic. In: Proceedings of the Workshop on Logic of Programs, New York, 1981. 52–71
2 Queille J, Sifakis J. Specification and verification of concurrent systems in CESAR. In: Proceedings of the Colloquium on International Symposium on Programming, Turin, 1982. 337–351
3 Clarke E M, Emerson E A, Sifakis J. Model checking: algorithmic verification and debugging. Commun ACM, 2009, 52: 74–84
4 Valmari A. A stubborn attack on state explosion. Form Method Syst Des, 1992, 1: 297–322
5 Burch J R, Clarke E M, McMillan K L, et al. Symbolic model checking: 10^{20} states and beyond. Inform Comput, 1992, 98: 142–170
6 Biere A, Cimati A, Clarke E M, et al. Bounded model checking. Adv Comput, 2003, 58: 117–148
7 Clarke E M, Grumberg O, Long D E. Model checking and abstraction. ACM Trans Progr Lang Syst, 1992, 16: 1512–1542
8 Tian C, Duan Z H, Duan Z. Making CEGAR more efficient in software model checking. IEEE Trans Softw Eng, 2014, 40: 1206–1223
9 Duan Z H, Tian C. A unified model checking approach with projection temporal logic. In: Proceedings of the International Conference on Formal Engineering Methods, Kitakyushu-City, 2008. 167–186
10 Duan Z H. Temporal Logic and Temporal Logic Programming. Beijing: Science Press, 2005
11 Godefroid P, Wolper P. A partial approach to model checking. Inform Comput, 1994, 110: 305–326
12 Clarke E M, Grumberg O, Jha S, et al. Counterexample-guided abstraction refinement for symbolic model checking. J ACM, 2003, 50: 752–794
13 Duan Z H, Tian C, Zhang N. A canonical form based decision procedure and model checking approach for propositional projection temporal logic. Theor Comput Sci, 2016, 609: 544–560
14 Duan Z H, Tian C. A practical decision procedure for propositional projection temporal logic with infinite models. Theor Comput Sci, 2014, 554: 169–190
15 Kroening D, Tautschnig M. CBMC — C bounded model checker. In: Proceedings of the International Conference Tools and Algorithms for the Construction and Analysis of Systems, Grenoble, 2014. 389–391
16 Henzinger T A, Jhala R, Majumdar R, et al. Software verification with Blast. In: Proceedings of the International SPIN Workshop on Model Checking of Software, Portland, 2003. 235–239
17 Ball T, Bounimova E, Kumar R, et al. SLAM2: static driver verification with under 4% false alarms. In: Proceedings of the International Conference on Formal Methods in Computer-Aided Design, Lugano, 2010. 35–42
18 Tian C, Duan Z H. Expressiveness of propositional projection temporal logic with star. Theor Comput Sci, 2011, 412: 1729–1744
19 Büchi J R. Symposium on decision problems: on a decision method in restricted second order arithmetic. Stud Logic Found Math, 1966, 44: 1–11
20 Gomes C P, Kautz H, Sabharwal A, et al. Satisfiability solvers. Found Artif Intell, 2008, 3: 89–134
21 Duan Z H, Yang X X, Koutny M. Framed temporal logic programming. Sci Comput Program, 2008, 70: 31–61
22 Wang X B, Duan Z H, Zhao L. Formalizing and implementing types in MSVL. In: Proceedings of the International Workshop of Structured Object-Oriented Formal Language and Method, Queenstown, 2013. 62–75
23 Zhang N, Duan Z H, Tian C. A mechanism of function calls in MSVL. Theor Comput Sci, in press. doi: 10.1016/j.tcs.2016.02.037
24 Rosner R, Pnueli A. A choppy logic. In: Proceedings of the Symposium on Logic in Computer Science, Cambridge, 1986. 306–313
25 Bowman H, Thompson S J. A decision procedure and complete axiomatization of finite interval temporal logic with projection. J Logic Comput, 2003, 13: 195–239
26 Tang C S. Toward a Unified Logical Basis for Programming Languages. Technology Report, No. STAN-CS-81-865. 1981
27 Moszkowski B C. Executing Temporal Logic Programs. Cambridge: Cambridge University Press, 1986