Asian Journal of Information Technolgy 4 (11): 1055-1060, 2005

SYMTC: Towards a Symbolic Model Checking for the Codesign

R. Boudour and M.T. Kimour
Department of Computer Science, University of Annaba Bp. 12, Annaba, Algeria

Abstract: The verification of finite-state systems by model-checking often requires to generate (a large part of) the state space of the system under analysis. In this study, we aim at improving the performances of state space construction by using an efficient method to avoid state explosion problem in model checking through the use of-DBM (Difference Bounded Matrices) and on the fly strategy. This approach requires at any time, only the needed states to be in memory and allows for checking several properties, especially, safety, bounded liveness and temporal correctness, which are the most important ones in reactive systems. The specifications are expressed in timed automata and TCTL for the system and properties respectively. The effectiveness of our approach has been demonstrated on many academic examples. The results obtained demonstrate that it is able to verify several properties that could not be checked by other state-of-the-art tools.

Key words: Model checking, timed automata, DBM, on the fly strategy, TCTL

INTRODUCTION

Model checking is emerging as a practical tool for automated debugging of complex reactive systems. It is the most successful approach that's emerged for verifying requirements[1-4]. A model-checking tool accepts system requirements or design (called models) and a property (called specification) that the final system is expected to satisfy. In model checking, a high-level description of a system is compared against a logical correctness requirement to discover inconsistencies. The model checker will either terminate with the answer true, indicating that the model satisfies the specification, or false, indicating that the model does not satisfy the property and provides a counterexample execution that shows an execution trace that violates the claim. Counterexamples are one of the most useful features of model checking, as they allow users to quickly understand why a claim is not satisfied.

In this field, many researchers[5-10]are mainly faced with application complexity especially in the reactive system domain. Among these is, the state space explosion problem. The latter is the subject of most model checking study. Usually, this problem results from the fact that the size of the state space is exponential in the number of variables and concurrent units in the system.

In this study, we provide an approach to reduce state space complexity and runtime. It consists of an appropriate exploration on the fly, which is based on a data structure called DBM[11,12]. We model the system by timed automata and express requirements by TCTL (Timed Computation tree Logic)[13-15].

PRELIMINARIES AND STATE OF THE ART

Timed automata: Since its introduction by Alur and Dill[16], timed automata were studied under multiple facets, both on the theory languages level and temporized models for specification and checking. Indeed, problems such as non-determinism[12,17], minimizing[5], the expressive power of clocks[18,19] and logical characterization of timed languages[14,20], were studied. This model was used successfully in the specification and the checking of timed systems[21]. Timed automata are Kripke structures with variables called clocks. Clocks are variables that evolve in time, all at the same speed, except the one that represents the universal time which is never set to zero. This general clock is often implicit.

Formally, a Kripke structure is a transition system $M = <S, R, I, L, \lambda>$. S consists of the set of possible states, It is the set of initial states $I \subseteq S$, R a set of transitions, $R \subseteq SxS$, L is a set of labels and λ an application of S in 2^{π}, which associates with each state an element of π (π: a set of atomic propositions).

A path in the Kripke model is an infinite sequence $\sigma = s_0, s_1, s_2, \ldots \in S^*$, $s_0 \in I$ and $(s_i\ s_{i+1}) \in R$. A state s is reachable in M if there is a path from s.

TCTL: A temporal logic with timing constraints: TCTL formulae are built according to the following syntax:

$$\varphi_s ::= \neg\varphi_s \mid \varphi_s \varphi, \Psi_s \mid E\,\varphi_p \mid p_1 \mid p_2 \mid \bullet\bullet\bullet \text{ (state formulae)}$$
$$\varphi_p ::= \neg\varphi_p \mid X\varphi_s \mid \varphi_s U_\alpha \Psi_s \text{ (path formulae)}$$

Corresponding Author: R. Boudour, Department of Computer Science, University of Annaba Bp. 12, Annaba, Algeria

where α, called the timing constraint, is a predicate on durations. Typically, all constraints of the form $\leq k$, $= k$ and $= k$ for k a value in the relevant domain (here N) are allowed. Sometimes, slight extensions are allowed. For example, nuSMV allows interval constraints of the form $\in[k_1..k_2]$ (but does not admit $= k$ constraints). We write $n \models \alpha$ when duration $n \in N$ satisfies the constraint α.

We write $s \models \varphi_s$ and $\pi \models \varphi_p$ when a state $s \in S$ satisfies a state formula φ_s (resp. a path π satisfies a path formula φ_p). The definition is as expected and we only spell out the few cases that are specific to TCTL and TKS's:

$$s \models E\varphi_p \underset{\Leftrightarrow}{\text{def}} \text{ there exists a } \pi \in \prod(s) \text{ st } \pi \models \varphi_p,$$

$$\pi \models X\varphi_p \underset{\Leftrightarrow}{\text{def}} \text{ there exists a } \pi(1) \models \varphi_s, \in \Pi(s) \text{ st } \pi \models \varphi_p,$$

$$\pi \models \varphi_s \underset{\Leftrightarrow}{\text{def}} \pi(1) \models \varphi_s,$$

$$\pi \models \varphi_s U_\alpha \varphi_s$$

$$\Leftrightarrow \begin{cases} \text{there exists } i \in N \text{ st } D\pi[0..i] \models \alpha, \\ \pi(i) \models \psi_s \text{ and } \pi(j) \models \varphi_s \text{ for all } 0 \leq j < i < \lambda. \end{cases}$$

We use all the classical abbreviations: T, $\perp$, $\varphi_s \vee \Psi s$, $\varphi_s \rightarrow \Psi s$, $A\varphi_p$ (for $\neg E\neg\varphi_p$), $F_\alpha\varphi_s$ (for $T U_\alpha \varphi_s$) and $G_\alpha\varphi_s$ (for $\neg F_\alpha \neg\varphi_s$). Then $EF_\alpha\varphi$ means that it is possible to reach a state satisfying φ via a finite path whose cumulative duration satisfies α and $EG_\alpha\varphi$ means that there is a path along which φ holds at all states visited after a cumulative duration satisfying a[22-26].

Decision algorithm: One of the criteria of development for the model-checking is its decidability, i.e. it is possible to develop algorithms which calculate if the model of the system checks or not the property specification. There are varieties of algorithms according to the formalism used to model system[8,27-30] as well as the type of property and its specification language. The main criteria of development of these algorithms are mainly the effectiveness and the facility of implementation. Of course, the effectiveness is strongly related to the theoretical complexity of the problem of model-checking, but it is not equivalent. Moreover, the development of the algorithms goes with the importance of data structures for their implementation. These structures must be relatively compact.

Zone: In practice, verification tools for timed automata use regions and zones and BDD[13,31,32], for exploring the state space. A zone is a set of clock valuations definable by a conjunction of constraints on the form $x \sim c$ or $x-y \sim c$, where x and y are clocks, c is a constant and $\sim$ is one of the relational operators in $\{<;\leq, \leq;>\}$. A zone describes a union of several regions and is thus a coarser representation of the state space. If we restrict the set of guards and invariants by disallowing negations, then we can easily restate the semantics of a timed automaton in terms of zones rather than clock valuations: A symbolic state is a pair (l;Z), where l is a location and Z is a zone. When computing the successor of a symbolic state we first compute the effect of an edge by applying the guard of the edge and then projecting the zone according to the updates on the edge. Second we compute the future of the zone, i.e., the set of states which can be reached by delaying and apply the invariant of the target location. Using zones, we obtain a countable representation of the state space. Here it becomes important to distinguish between diagonal-free and non-diagonal-free timed automata. The first class is the subset of timed automata where guards and invariants are limited to conjunctions of the form $x \sim c$, where $\sim \in \{<;\leq;\geq;>\}$. Bouyer[8] was proven that for diagonal-free timed automata, we can construct abstractions and do all the operations effectively appearing in the algorithm.

Usaually the model-checker users simplify the model which they analyse, until being able to control it. That involves more distance between the model and the real system. It is difficult to manage this compromise. Currently, many theoretical studies seek to automate certain aspects of this simplification step.

Remark 1: The main problem of the algorithms of model-checking was the explosion of the number of states. This explosion occurs each time one decides to enumerate and to explicitly represent in memory all the states of the automaton examined. To surmount this obstacle, many techniques are used such as, the symbolic model-checking calling upon the zones to represent the sets of states.

SYMBOLIC MODEL CHECKING

In short, model checking is a collection of techniques for automated formal verification of finite-state concurrent systems. We start off with a bird's eye view of the process (Fig. 1). and proceed to refine it in the following paragraphs.

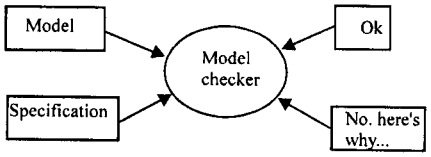

Fig. 1: Model checking: A high level view

Given a model (an abstraction of the system) and a specification of the properties that are required to hold (relating to absence of deadlocks, liveness, invariants etc.), the model checker verifies whether the former satisfies the latter. A counterexample is produced upon discovery of a violation. This approach uses an algorithm of symbolic model checking which calculates the unit characteristic of a formula tctl i.e. the entire configurations i checked by this formula. The principal problem of the symbolic approach relates to the procedure of decision. Indeed, to decide if a set of states is included in another, we decide if a predicate implies another. In order to solve this problem, we propose a representation of the sets characteristic of the formulas which is at the base of a decision procedure implemented efficiently. The algorithm of symbolic model checking comprises 4 steps: 1) to represent the predicates of states, 2) to represent the temporal constraints in form matrix, 3) to calculate the operator > and 4) to evaluate symbolically the formulas of tctl.

Being given a formula φ of tctl and a timed automaton, the symbolic algorithm consists in calculating for φ the unit S (φ) of symbolic states which represents the characteristic unit of φ. Either A = (S, H, E, S_i,δ, P) an timed automaton and φ is a formula of TCTL, S (φ) is built by induction on the structure of φ in the following way:

$S(p) = D(p)$; $S(x\ \pi\ c) = D(x\ \pi\ c)$

$S(x\text{-}y\ \pi\ c) = D(x\text{-}y\ \pi\ c)$; $S(x.\ \varphi_1) = S(\varphi_1)\ [\ x:=0\ ]$

$S(\neg\varphi_1) = \neg S(\varphi_1)$; $S(\varphi_1 \wedge \varphi_2) = S(\varphi_1) \wedge S(\varphi_2)$

$$S(E\varphi_1 \cup \varphi_2) = \bigvee_{k\in N} X_k$$

Where: $X_0 = S(\varphi_2)$ and $X_{k+1} = X_k \vee S(\varphi_1) > X_k$

$$S(A\varphi_1 \cup \varphi_2) = \bigvee_{k\in N} X_k$$

Where: $X_0 = S(\varphi_2)$ and $X_{k+1} = X_k \vee \neg(Y^k\ [z:=0])$ and

$$Y = \bigvee_{j\in N} Y_j^k \text{ with } Y_0^k = (\neg S(\varphi_1) \wedge \neg X_k) \vee S(z > 1)$$

$$\text{and } Y_{j+1}^k = Y_j^k \vee \neg X_k > Y_j^k$$

In order to implement the algorithm, an adapted data structure is needed to represent zones and this data structure allows testing inclusion of zones and computing easily the different operations used in the algorithm, that is the intersection of two zones, the future of a zone, the image of a zone by a reset and the k-approximation of a zone. By using the automata theoretic approach to model checking, it is possible in many cases to avoid construction the entire state space of the modelled system. This is because the states of the automaton are generated only when needed, while checking the emptiness of its intersection with the property automaton. This tactic is called on-the-fly mode checking.

Remark 2: we have presented an algorithm which is implemented in some tools such as Knonos[33]. and Uppaal[34].

RESULTS

To validate our tool, we chose two examples of reactive systems, the Mouse with only one button[35], temperature controller[36] and a third abstract example[15]. To facilitate the comprehension and comparison, we have chosen often used examples in the literature.

TCTL properties: We have not only focused on safety properties that constitute the majority of properties required to verify systems, but also on other important ones such as the bounded liveness and temporal correctness properties. Bounded liveness property guarantees that something will take place with giving the deadline information. Temporal correctness property avoids deadlock situations.

Temporal correctness property: It is stated as follows: φ = (EFw = constant > 0, tested on the example 1.

The temporal correctness property is used to check that the system is well modeled i.e. one cannot reach a state of deadlock. This deadlock might occur when the temporal constraints are not expressed correctly. For example, an invariant of state 0 is $x \leq 5$, the condition of transition from the state 0 to the state 1 is $x < 3$. In the state 0, the system can remain no later than 5 time units. If the system remains in the state 0 more than 3 time units, it cannot fire the transition between states 0 and 1. This situation is a deadlock. It must be corrected. An example of such situation is illustrated below.

Bounded liveness property: For the mouse, we wish to check the following property: if the button in a hurry and were slackened once then inevitably before an time equal with $t_{cs}+ t_{cd}$, the system detects a simple click or a double click. This property is expressed in TCTL by the formula:

$$\varphi = (p1 \wedge r1 \wedge x = 0) \rightarrow A\,F < 5\ (cs \vee cd).$$

Safety property: The specification of the controller is established for reasons of safety. If the controller does not receive a new order of refrigeration before a time t_{max} since the last order received, then it must also refrigerate.

Table1: Recapitulative table of the results obtained by the SYMTC Checker

Example	Property	Checking result	No.of visits	A No. of bytes
Mouse	EFy=5	satisfied	103	857
	(p1 ∧ r1 ∧ x = 0) →A F< 5 (cs ∨ cd)	satisfied	131	429
	P1 → ¬ E ¬ cs U (r1 ∧ y > 5)	not satisfied	47	349
Temperature controller	EFz=10	satisfied	83	1997
	r → ¬ E ¬ b U (r ∧ z > 35)	satisfied	69	563
	(r ∧ b ∧ x = 0) → A F< 35 (r ∨ b)	not satisfied	30	62
Abstract example	EFw = 1	not satisfied	34	2632
Train	AF < 90(Su)	satisfied	27	120
	AG(Ar- AG(T ≥ 120)not Su)	satisfied	36	210
Gate	AF<38(Sb)	not satisfied	75	390
	AG(Ar- AG(B < 45)Bs)	satisfied	55	230

This property is expressed by the following formula:

$$\varphi = r \rightarrow \neg E \neg b \, U \, (r \wedge z > 35),$$

This means that it is not possible to reach a state which satisfies r in a time more than 35 time units, starting from a state which satisfies r without passing by a state which satisfies b.

Results: The results given by SYMTC are synthesized on Table 1. We can deduce that the complexity of the model-checking is a function of the parameters such as the number of model states, the number of clocks and the type of the property to be checked. We show here three examples, which exhibit different characteristics (in terms of number of states, number of clocks)[10]. For the mouse model, the obtained results (satisfied property or not) for the bounded liveness property is the same as those shown by Shaw[35] and Yovine[37]. For the temperature controller's model, the positive result of safety property is identical to that obtained Jaffe and Leveson[36] while the other obtained results are specific to our tool and can be used as a basis of comparison in research works.

Table 2 shows the comparison between the regions and symbolic approaches. It is worth noting that the gain on the memory space is very important; it goes from 20 to 8547 times less.

Compared to the region's method, our approach reduces at a large extent, both the required memory size and the execution time. Firstly, we explore all the model states by saving the state invariants in a matrix, next, we execute the algorithm at every matrix entry consultation to recover the data. This will allow us to avoid exploration of the whole graph. Secondly, to reduce the memory size, we use the characteristic set concept or zone.

On the other hand, in contrast to other similar extensions, we have used TCTL that has two main advantages: it is based on traditional structures of kripke (Kripke) and the model checking is done in polynomial time. In addition to its similarity with Kronos[37-39], SYMTC has a graphic interface of timed models and a simulation module, but it is not limited in reachability property, like UPPAAL[34,40] or HYTEC[33]. Furthermore, our approach allows more properties to be checked such as the safety, bounded liveness and correctness ones, etc.

Table 2: Comparison between the two approaches

Complexity (bytes) examples	Regions approach]	Symbolic approach
Mouse	8956	429
Controller temperature	4 812 000	563
Example standard	824 000	2632

CONCLUSIONS

In this article, we proposed an approach to handle the memory explosion problem in symbolic model checking through the use of timed automata for reactive systems model. The approach allows for checking various important properties like safety, bounded liveness and temporal correctness. To this end, we used DBM representation that brings a significant advantage due to its ability to control the memory required. This approach is based on the fly invariant checking and checking TCTL formulas expressible as least fix point. Another distinctive feature of our approach relies in the use of dynamic cache that brings more state space exploration speedups.

Compared to state of the art techniques, experimental results of our approach, show that it scales very well and reduces the memory required without affecting results quality and performance. It can be applied to different domains of hardware and software verification.

This has been validated through a tool support, implemented using Visual C++6.0. It has been applied on examples and was able to handle checking complex properties.

Future study would focus on varied verification application domains, where state of the art model checkers are not very efficient. We also plan to extend the technique to verify more varied and complex properties. There are a number of other ways that model checkers can be improved to make them easier to use by engineers. one problem with current systems is how to make the

specification language more expressive and easier to use. Some type of timing diagram notation may be more natural for engineers than TCTL. One trend for research is to develop even more concise techniques for representing Boolean functions. When better representations are developed, they can easily be incorporated into model-checking algorithms. Other direction is in investigation Model Checking techniques combined with very different styles of reasoning into a single framework.. The problem is how to combine and can smoothly integrate the results obtained by each.

REFERENCES

1. Abadi, M. and L. Lamport 1993. Composing specifications. ACM Transactions on programming Languages and Systems, 15: 73-132.
2. Abadir, M., K. Albin, J. Havlicek, N. Krishnamurthy and A. Martin, 2003. Formal verification successes at motorola. Formal Methods in System Design, 22: 117-123.
3. Alur, R., L. Fix and T.A. Henzinger, 1994. A determinizable class of timed automata. Proceedings CAV, 94, Lecture Notes in Computer Science, Springer-Verlag, pp: 1-13.
4. Alur, R. and T.A. Henzinger, 1992. Logic and models of real time: A survey. In: Real-time: Theory in practie. Computer Sci., 600: 74-106.
5. Alur, R., C. Courcoubetis, N. Halbwachs, D.L. Dill and et.H. Wong-Toi, 1992. Minimisation of timed transition systems. Computer Science, 630: 340-354.
6. Berard, B., B. Bidoit, A. Finkel, L. Laroussinie, A. Petit, L. Petrucci and P. Schnoebelen, 2001. Systems and software verification: Model-checking techniques and tools. Berlin-Heidelberg: Springer Verlag.
7. Boyer, M. and J.C. Pince, 2004. Model checking aléatoire: Une Approche Entre Test et Vérification, FAC.
8. Bouyer, P., 2002. Modèles et algorithmes pour la vérification des systèmes temporisés, ENS de Cachan.
9. Clarke, E.M., E.A. Emerson and A.P. Et Sistla, 1986. Automatic verication of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems, 8: 244-263.
10. Laroussinie, F., P. Schnoebelen and M. Turuani, 2000. On the expressive and complexity of quantitative branching-time temporal Logics. Comp. Sci., 1776: 437- 446.
11. Alur, R., C. Courcoubetis and D. Dill. 1993. Model-checking in dense real-time, Information and Computation, 104: 2-34.
12. Alur, R. and D.L. Dill, 1994. Automata For Modeling Real-Time Systems. ICALP 1990:322-335 also as A Theory of Timed Automata. Theor. Comput. Sci., 126: 183-235.
13. Bryant, R.E., 1986. Graph based algorithms for boolean function manipulation. IEEE,Transactions on Computers, 35: 677-691.
14. Wilke, T., 1994. Specifying timed state sequences in powerful decidable logics and timed automata. Comp. Sci., Springer-Verlag, 863: 694-715.
15. Yovine, S., 1992. Méthodes et outils pour la vérification symbolique de systèmes temporisés, Mars.
16. Alur, R. and D.L. Dill, 1990. Automata for modelling real-time systems. Computer Science, 443: 322-335.
17. Alur, R. and D.L. Dill, 1994. A theory of timed automata. Theor. Comp. Sci., 126: 183-235.
18. Alur, R., C. Courcoubetis and T.A. Henzinger, 1994. The observational power of clocks. Comp. Sci., pp: 162-177.
19. Henzinger, T.A., P.W. Kopke and H. Wong-Toi, 1995. The expressive power of clocks. ICALP, 95, Comp. Sci., Springer -Verlag, pp: 335-346.
20. Laroussinie, F., K. Larsen and G, Weise, 1995. From timed automata to logic and bac. f MFCS, 95, Comp. Sci., 969: 27-41.
21. Henzinger, T.A., X. Nicollin, J. Sifakis and S. Yovine, 1994. Symbolic model checking for real-time systems. Inform. Comp., 111: 193-244.
22. Clarke, E. and M. Edmund, 1999. Orna Grumberg and Doron A. Peled. Model Checking, Cambridge, MIT Press, Cambridge.
23. D'Orso, J., 2003. New directions in symbolic model checking, Université d'Uppsala.
24. Henzinger, T.A., P.H. Ho and H. Wong-Toi, 1997. HyTech: A model -checker for hybrid systems. J. Software Tools for Technol. Tran., 1: 110-122.
25. Koymans, R., 1990. Specifying real-time properties with metric. Real -Time Systems, 2: 255-299.
26. Pnueli, A., 1997. The temporal logic of programs. 18th IEEE Symposium on Foundations of Comp. Sci., FOCS77, pp: 46-57.
27. Pistore, M. and M. Roveri, 2004. Symbolic model checking, IIT, Delhi, India, Fevrier.
28. Ramakrishna, Y.S., P.M. Melliar-Smith, L.E. Moser, L.K. Dillon and G. Kutty, 1996. Interval Logics and their decision procedures Part I: An interval Logic. Theor. Comp. Sci., 166: 1-47.
29. Ramakrishna, Y.S., P.M. Melliar-Smith, L.E. Moser, L.K. Dillon and G. Kutty, 1996. Interval Logics and their decision procedures, Part II: A real- time interval Logic. Theor. Comp. Sci., 170: 1-46.

30. Wolper, P., 1989. On the relation of programs and computations to models of temporal logic. Comp. Sci., Springer, 398: 75-123.
31. Bryant, R.E., 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys, 32: 293-318.
32. Clarke, E.M. and J.M. Wing, 1996. Formal methods: State of the art and future directions. ACM Computing Surveys, 28: 626-643.
33. http://www-verimag.Imag.fr/TEMPORISE/kronos/
34. http://www.uppaal.com/
35. Shaw, A., 1992. Communicating real-time state machines. IEEE Transactions on Software Engineering, 18: 805-816.
36. Jaffe, M., N. Leveson, M. Heimdahl and B. Melhart, 1997. Software requirements analysis for real-time process-control systems. IEEE Trans. Software Eng., 17: 241-258.
37. Yovine, S., 1997. Kronos: A verification tool for real-time systems. J. Software Tools Technol. Transfer, 1: 123-133.
38. Yovine, S., 1998. Model-Checking Timed Automata. In School on Embedded Systems, vol.1494, Computer Science. Springer-Verlag.
39. http://www-cad.eecs.berkeley.edu/~tah/Hytech/
40. Larsen, K.G., P. Pettersson and W. Yi, 1997. UPPAAL in nutshell, J. Software Tools for Technology Transfer, 1: 134-152.