Real-Time system verification techniques based on abstraction/deduction and model checking

Eunyoung Kang

Real-time system verification techniques based on abstraction/deduction and model checking

EunYoung Kang

LORIA-INRIA, France
Technical University of Delft, The Netherlands
Eun-Young.Kang@loria.fr

Abstract. Our research focuses on verification techniques for real-time systems based on predicate abstractions. These techniques aim to combine abstract interpretation, model checking, and theorem proving in order to obtain a powerful and highly automatic verification environment for real-time systems. One drawback of current real-time model checking approaches is the limited size of the systems that can be analyzed. For the computation of finite abstractions in the way of infinite-state systems analysis, we propose an Iterative-Abstract-Refinement algorithm. Using our algorithm, we can reduce the aforementioned drawbacks associated with the application of real-time model checking such as the limited applicability due to state space explosion characteristics

1 Introduction

The automatic verification problem for finite-state real-time systems has been considered and solved [1, 2, 16]. In many cases, theoretically optimal algorithms are known [4, 18]. Unfortunately, even if these algorithms are fully automatic, they are confronted with the state-explosion problem. They are typically exponential in the number and maximum values of clocks.

On the other hand, deductive techniques can in principle be used to verify infinite-state systems, based on suitable sets of axioms and inference rules. Although they are supported by theorem provers and interactive proof assistants, their use requires considerable expertise and tedious user interaction.

Abstract interpretation [8] provides a different approach to computing finite-state abstractions. For instance, predicate abstraction [10, 17] is a well known approach; given a transition system and a finite set of predicates, this method determines a finite abstraction, where each state of the abstract state space is a truth assignment to the abstraction predicates.

Model checking and abstraction/deductive techniques are therefore complementary. We propose an efficient scheme by combinations of those approaches that should give rise to powerful verification environments. For example, a theorem prover can be used to verify that a finite-state model is a correct abstraction of a given system, and properties of that finite-state abstraction can then be established using model checking. It relies on the fact that most properties can be shown correct without the need to maintain precise timing information for

the system. A complex set of timed states can be safely abstracted by a simpler (abstract) one.

In general, the relationship between concrete and abstract models that underlies abstract interpretation is described by a Galois connection. The abstract domain is that Boolean lattice whose atoms are the set of predicates true or false of a set of states in a concrete model. The model obtained by abstraction w.r.t. this lattice exhibits at least behaviours of the concrete system; it may also include some bahaviours that have no counterpart in the concrete system.

The idea is to reduce the number of states of a model by abstracting away behaviours not essential to the verification. The genetic techniques are known as incremental abstraction refinement [6] and counterexample-guided abstraction refinement [7]. We have studied and partially formalised a variant that combines several techniques [7, 6], with some modifications. Abstraction can be constructed manually to exploit the structure of the model under consideration.

The idea is to perform an initial over-approximating abstraction of the model and then check for two requirements: (1)-the *conformance* between an abstract model and its concrete model, and (2)-the required property. If the abstract model conforms to its concrete model and the properties of interest can be successfully verified (or a positive result) over the abstract model, they also hold of the concrete system as well.

Otherwise, either the abstract model is not the correct representation of its concrete model or a negative result may be spurious (caused by extra states added during the over-approximation). In this case, we iteratively concretize (or refine) the abstract model in order to construct stronger invariants and rule out some of their extra behaviours until a positive or true negative result is obtain. We call such a concretization process an *abstraction-refinement*, and we call the model resulting from the *abstraction-refinement* a *complete model*

Our idea for advances in the size of systems that can be analyzed by using several abstraction methods is not the first. A comparison with similar work on abstract interpretations, approximations for real-time systems [9] and predicate abstractions appears in the section 3.

2 Approach and proposed solution

We have proposed a tool supported methodology based on the combination of abstract/deductive and real-time model checking techniques. In order to make our approach more concrete, we present one algorithm, called *Iterative-abstract-refinement algorithm* (IRA) to verify a rich class of safety and liveness properties of a timed system based on computing a finite abstraction of the system by succ essive *abstraction-refinement*.

Figure 1 shows models over an abstract, a complete, and a concrete domain. The abstract model shown in Fig.1.(a) cannot guarantee whether the abstract model-(a) is able to verify given properties and preserve every possible behaviour of a concrete model-(c). The model of Fig.1.(b) has been obtained by *abstraction-refinement*; it is *complete* for verifying the properties of the concrete model-(c).

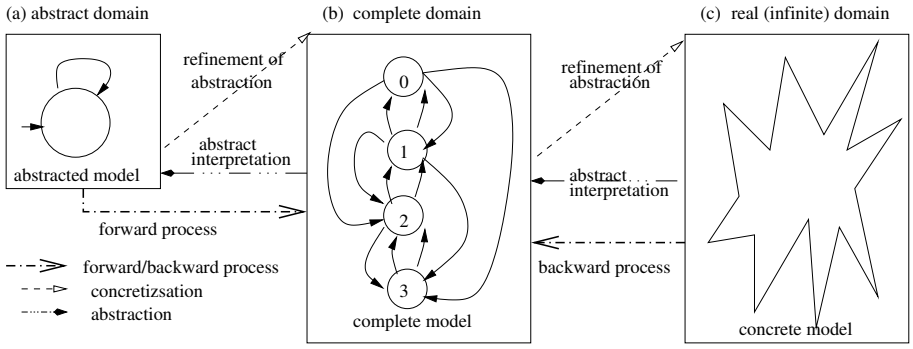

Fig. 1. Abstract, complete, and concrete models

We expect to obtain a *complete* model-(b) from an abstract model-(a) using IRA.

In this paper, we will use eXtended Timed automata Graphs [3] as the formalism to represent (concrete) timed system-(c), and Predicate Diagrams for Timed systems [14] as the way of presenting predicate abstraction for XTG.

The input to IRA consists of three parameters; the XTG representing the concrete system, the property to be verified, and a finite set of predicates to be used for refinements. For the sake of efficiency we require predicates (Boolean expressions) over a set of *configurations* of XTG that are constraints, invariants, guards, locations and transitions of XTG.

Our methodology is based on *abstraction-refinement* framework. IRA has two (forward/backward) processes. An initial abstraction PDT can be obtained from the XTG by a backward step. This backward step (process) is performed by selecting a set of XTG configurations and considering them as predicates of PDT. Starting from the initial PDT, IRA iterates forward process until PDT is *complete*:

Backward process: direction (c) to (a) A trivial and potentially incomplete PDT is extracted from XTG by Boolean abstraction. Its atoms are the subset of given predicates over a set of XTG configurations and properties of interest as well.

Forward process: direction (a) to (b) IRA picks a PDT and – if the PDT appears to be complete – infers successful verification and preservation, or – if the PDT is not complete – enforces completeness by two operations: a *splitting* operation and an *excluding* operation

1. A *splitting operation* is done while IRA checks correctness of preservation in the way of *conformance* checking between a PDT and XTG.
   - If IRA fails to prove *conformance* then IRA does splitting the PDT w.r.t predicates in order to enrich the PDT and also to add details in the PDT.

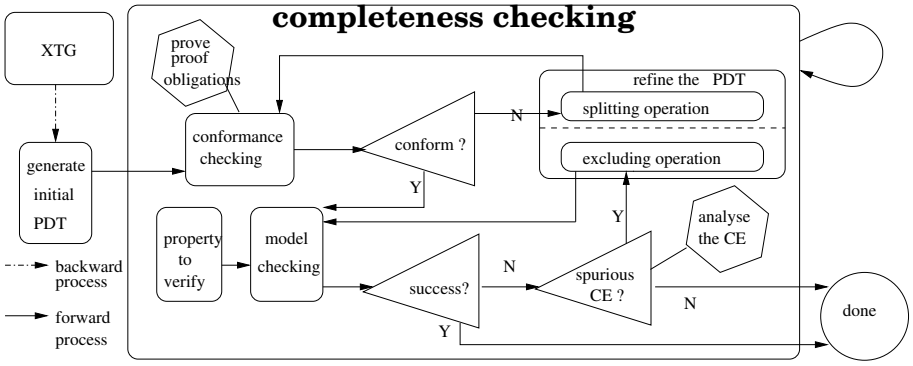

Fig. 2. Overview of IRA

- During the operation, a set of proof obligations (a number of verification conditions expressed in first-order logic) are proved in order to eliminate extra duplicated relations (paths) among PDTs caused by splitting. This is continued until the PDT conforms to XTG.

2. An *excluding operation* is done after model checking (or after *conformance* checking).
   - If we limit ourselves to univeral properties, then if a property fails in the PDT we can generate a counterexample trace in the PDT and attempt to find a corresponding concrete trace in XTG.
   - If one exists then the property is false in XTG and the verification fails (true negative result is obtain). Otherwise the abstract counterexample is spurious and abstraction is too coarse so we refine it by *excluding* some abstracted bogus transition-relations that are not present in XTG (found by analysing the counterexample).
   - We then recheck the property. This is continued until either the property is verified or a concrete conterexample found.

Figure 2 shows the overall framework. The above approach is tested in [14] and partially validated. The main contribution up to now is that we have identified a suitable format that serves as an interface between deductive and model checking techniques, intended for the verification of real-time systems. We also have established a set of verification conditions that are sufficient to prove *conformance* between PDT and XTG.

However, it still lacks automation both in the computation of abstraction and in identifying the predicates for splitting. Moreover, We have not fully validated the above approach, for instance, our experiment in [14] does not consider a spurious counterexample trace upon failure during model checking procedure, thus any counterexample-guided abstraction refinement method does not really used during the execution of the *abstraction-refinement* procedure. Considering such current weakpoints and comparing with other related work in the next section, we will discuss future work at the end of this paper.

3 Related work

The techniques described in this paper can be viewed in an abstract interpretation sense as a combination of abstraction, operation on an abstract domain and concretization. Our refinement bears resemblance to refinement as in the B method [6]. It allows one to enrich a model in a step by step approach. Refinement provides a way to construct stronger invariants and also to add details in a model. It is also used to transform an abstract model into a more concrete version by modifying the state description.

Halbwachs [11] successfully applies abstract interpretation to synchronous reactive systems as a way of state space exploration. But he does not consider abstractions over control information (only data information is abstracted). Dill and Wong-Toi [9] use both over- and under-approximations as abstractions, and for finite-state systems, automatically determine whether there are reachable violating sates. Their refinements are different than ours. They refine (over-approximations only) the set of reachable states on paths to violating states. However their techniques are limited to proving invariants.

Predicate abstraction has emerged as a fruitful basis for software verification. Based on predicate abstraction, Namjoshi and Kurshan [15] compute finite bisimulations of timed automata. However, currently it is unclear whether their approach is applicable in practice.

Our basic assumption undeliying predicate abstraction is that for the verification of a given property, the state space of an XTG can be partitioned into finitely many equivalence classes. For example, the precise amount of time elapsed in a transition does not really matter as long as the clock values are within certain bounds and similarly, the precise values of the data can be abstracted with the help of predicates that indicate characteristic properties.

Predicate abstraction also underlies tools such as SLAM [5] and BLAST [12] that compute abstraction refinements on the basis of spurious counter-examples provided by model checking. They refine abstractions in such a way that the spurious counter-example is avoided.

In symbolic model checking for real-time systems, difference bound matrices (DBM) are used to represent a set of state spaces (regions) over real variables. The representation we use is rather tailored to IRA approaches, since it is able to efficiently deal with the two (*splitting/excluding*) operations required for such approaches. For those operations it is not necessary to have a canonical model available. In IRA, the represented region is the consequence of repeated *abstraction-refinement* by splitting PDT and excluding its abstracted bogus transition-relations.

Our early work on combining tools for abstract interpretation and state space exploration has been reported in [13]. However, extra steps used in the algorithm proposed there often fail to significantly reduce the state space.

4 Further work

In IRA, the predicates are assumed to be given by the user, or they are extracted syntactically from the system description. It is obviously difficult for us to find the right set of predicates. We are investigating further heuristics to be able to discover automatically all the needed predicates for the practical algorithm.

We abstract from the precise amount of time that may elapse in a time-passing transition. Thus, we cannot easily verify properties that describe the timing behavior of a system. We intend to study two possible solutions to this problem, either by using a timed temporal logic (TLTL) or by introducing auxiliary clocks during verification. In any case, we would want to take advantage of model checking tools for real-time systems.

Besides, we aim at reducing the number of verification conditions that users have to discharge with the help of a theorem prover in order to establish *conformance*. It will be interesting to restrict attention to specific classes of systems that give rise to decidable proof obligations.

We are also interested in adding counterexample-guided abstraction refinement method in SLAM and BLAST to IRA and validating a full scheme proposed in Figure 2.

Although IRA for now shows that we have not reached the ideal combining algorithm yet, it clearly helps in identifying opportunities for proper incorporation of abstraction/deduction and model checking for real-time systems in practical situations.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.
2. R. Alur and D. Dill. The theory of timed automata. *Theoretical Computer Science*, (126):183–235, 1994.
3. M. Ammerlaan, R. Lutje Spelberg, and W.J. Toetenel. XTG – an engineering approach to modelling and analysis of real-time systems. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, pages 88–97. IEEE press, 1998.
4. Tobias Amnell and many others. UPPAAL: Now, next, and future. In F. Cassez et al., editor, *Modeling and Verification of Parallel Processes*, LNCS(2067):99-124. Springer-Verlag, Berlin, 2001.
5. T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *Principles of Programming Languages (POPL 2002):1–3*, 2002.
6. Dominique Cansell and Dominique Mery. Tutorial on the event-based B method. Technical report, LORIA-INRIA, 2004.
7. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *12th CAV00, LNCS(1855):154–169*. Springer-Verlag, 2000.
8. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2-3):103–179, 1992.
9. D. Dill and H. Wong-Toi. Verification of real-time systems by successive over and under approximation. In *7th CAV95, LNCS(939):409–422*. Springer-Verlag, 1995.

10. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *9th CAV97, LNCS(1254):72–83*. Springer-Verlag, 1997.
11. N. Halbwachs. Delay analysis in synchronous programs. In *CAV93, LNCS(697)*. Springer-Verlag, 1993.
12. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth McMillan. Abstractions from proofs. In *31st POPL*. ACM Press, 2004.
13. EunYoung Kang. Parametric analysis of real-time embedded systems with abstract approximation interpretation. In *26th ICSE*, 2004.
14. EunYoung Kang and Stephan Merz. Predicate diagrams for the verification of real-time system. In *5th AVoCS05*, ENTCS, 2005.
15. K. Namjoshi and R.Kurshan. Syntactic program transformations for automatic abstraction. LNCS(1855):435–449, 2000.
16. S.Tripakis. *The Formal Analysis of Timed Systems in practice*. PhD thesis, University of Joseph Fourrier de Grenoble, 1998.
17. Y.Kesten and A.Pnueli. Modularization and abstraction: The keys to practical formal verification. In *23th MFCS98, LNCS(1450):54-71*. Springer-Verlag, 1998.
18. S. Yovine. Kronos: A verification tool for real-time systems. *Springer International Journal of Software Tools for Technology Transfer*, 1997.