ELSE: A new symbolic state generator for timed automata

Sarah Zennou, Manuel Yguel and Peter Niebert

Laboratoire d'Informatique Fondamentale de Marseille
Université de Provence – CMI
39, rue Joliot-Curie / F-13453 Marseille Cedex 13
[zennou,niebert]@cmi.univ-mrs.fr

Abstract. We present ELSE, a new state generator for timed automata. ELSE is based on VERIMAG's IF-2.0 specification language and is designed to be used with state exploration tools like CADP. In particular, it compiles IF-2.0 specifications to C programs that link with CADP. It thus concentrates on the generation of comparatively small state spaces and integrates into existing tool chains. The emphasis of the ELSE development is on fundamentally different data structures and algorithms, notably on the level of zones. Rather than representing possible values of clocks at a given symbolic state, event zones represent in an abstract way the timing constraints of past and future events. This approach is useful for certain partial order reduction approaches.

1 Introduction

Timed automata [AD94] are a powerful tool for the modeling and analysis of timed systems. They extend classical automata by *clocks*, continuous variables "measuring" the flow of time. A state of a timed automaton is thus a combination of its discrete control locations and the *clock values* taken from the real domain. While the resulting state spaces are infinite, *clock constraints* have been introduced to abstract the state spaces to a finite set of equivalence classes, thus yielding a finite (although often huge) symbolic state graph on which reachability and some other verification problems can be resolved.

While the theory, algorithms and tools like Kronos [NSY92,BFG+99] and Uppaal [LPY95] for timed automata represent a considerable achievement, they suffer for various reasons from combinatory explosion which still limits their applicability in practice. A great effort has been invested into optimization of representations of clock constraints, e.g. [DY96,BLP+99]. Another line of research is devoted to the overall reduction of reachability to logic constraint solving, e.g. [NMA+02].

ELSE, developed at the Laboratoire d'Informatique in Marseille, is a new state generator – engine of algorithmic analysis – for timed automata incorporating alternative semantics that allow certain partial order reduction approaches. ELSE is designed to be compatible with IF [BFG+99], notably the IF-2.0 specification language, for which it implements a new semantics allowing

state space reduction with respect to parallelism while preserving reachability properties: The components of a parallel system (like networks of communicating transition systems) can sometime progress independently, sometimes interact. Basic verification techniques rely on an interleaving approach, where *global states* are *tuples of local states*. The resulting global transition systems can have a very redundant structure (which is responsible for the state explosion) including so called *diamonds*, pairs of commuting transitions that can be executed in either order leading to the same state. *Partial order reduction techniques* [Val89,Pel93,God96,NHZL01] together with their tools (e.g. SPIN) give an answer to this phenomenon in reducing the search space based on this redundancy for discrete systems.

Partial order reduction methods for timed automata. A natural question is the applicability of partial order reductions to networks of timed automata. However, as has been observed by several authors, the standard interleaving semantics combined with symbolic states via clock constraints results in transitions that do *not commute*. Several kinds of answers have been given to this problem: Adapt *persistent set method* [YS97]; Define a local time semantics [BJLY98,Min99]. In [DT98], the authors starts adapting the notion of equivalence classes of transition sequences in timed automata.

This approach followed by *ELSE* is formally defined in [LNZ03]. Basically, the chosen semantics relaxes constraints between independent components (automata and clocks) so that diamonds are almost preserved. The termination and bound on the number of symbolic states, a problem known from [BJLY98,Min99], is ensured by adding certain constraints (which we call *lower catch up* and *upper catch up*) as for the interleaving approach. The price for this guarantee, that our symbolic transition systems are not bigger than those resulting from the classical approach, is that "independent transitions" commute in a weaker sense than for discrete systems.

2 Scope and Architecture of ELSE

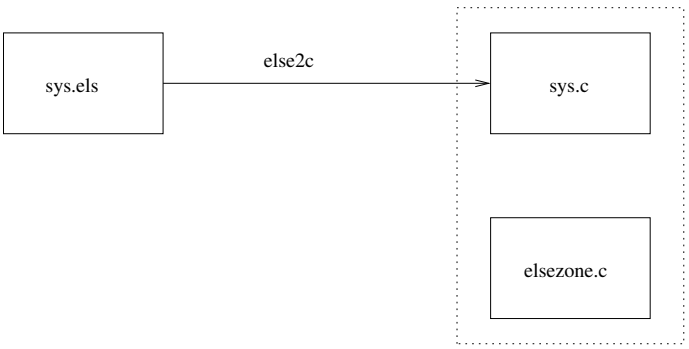
The generator *ELSE* is a tool to automatically translate a description of a network of timed automata in the IF syntax to C code providing following: A data structure for the symbolic states as defined in [LNZ03]; A mechanism to compute symbolic transitions from a given symbolic state. These elements can then be used by tools for exploring a symbolic reachability graph, CADP in particular. ELSE thus remains just one, yet a crucial component in a tool chain: Specifications may either directly be written in IF or obtained by (existing) translations to IF from various commonly used specification languages; ELSE provides a state generator; CADP may be used to explore it for reachability or more complex verification algorithms.

To achieve this, *ELSE* is composed of:

- A compiler, *else2c*, which generates the symbolic transition systems. It translates the description of a set of timed automata into functions C which compute them. The description must be done in a language called *Else*;

– A library computing operations on clock constraints, called *elsezone*. This one contains all functions which are needed by functions generated by the compiler.

To summurize, the following figure shows how *ELSE* works: Given a network of timed automata which are described in *Else* (file sys.els), the compiler *else2c* generates C functions (file sys.c) computing the corresponding symbolic transition system. The functions of computing transitions of this one may call functions on clock constraints in the library of clock constraints (file elsezone.c).



Internals of else2c. After the syntaxic verification of the system description, the generation of the C code used as entry by the graph explorer is done in two main steps: First, create the C representation of (symbolic) states of the reachability graph. The generated data structure, a static record with some dynamic attachments, repesents in a hierarchical manner the hierarchy of the system structure, as much as is possible in C, e.g. the automata are represented as subrecords, etc. Zones in our setting do not have a fixed size and are kept apart.

Then the translator computes a function for the interface, which, given a symbolic state of the reachability graph, computes and returns implicitly the list of its successors. This function is compatible with different kinds of search like DFS or BFS, etc. Partial order reductions are transparent for CADP: The transition systems it sees are already reduced, in particular concerning operations on "event zones" which are internal functions of the library elsezone. The initial state can be generated by calling a specific function.

3 Event zones and the library elsezone

For this section, we assume some familiarity with zones as used in classical timed automata tools.

The library *elsezone* contains all the functions computing the operations on constraints, called *event zones*, as defined in [LNZ03]:

Compared to classical zones, which represent *sets of clock values*, we have done a philosophical shift for else: Event zones represent constraints for the

occurrence times of certain event. A constraint "$C \leq 3$" guarding a transition can indeed be read in two ways: (a) at the moment of occurrence of the transition, clock C must have a value inferior to 3;. (b) the difference of occurrence times of the transition in question and the last preceeding reset of clock C is ≤ 3.

The first – classical – view (a) means on the level of the symbolic states this means that the transition is only possible if the set of clock value tuples before the transition contains such a concrete value, checked by cutting the clock value polyhedron with a hyperplain corresponding to the condition.

In view (b), we do not explicitely model clock values, nor the passage of time, but constraints between event occurrences. An event is an occurrence of a transition in an execution, in a concrete execution each one having a time of occurrence. View (b) tells you under which conditions such a vector of occurrence times is possible. As a new transition is added, it is linked to some preceeding occurrences of transitions, adding a new dimension and new constraints.

However, the constraints linking future events with the events already added to a symbolic path result either from clock constraints or from causality (an event f causally depending on a preceeding event e must occur *later*). After a reset of clock C, no preceeding reset of C can be linked to by a clock constraint on C in a future transition. Since no more reference to this event is possible, we can remove it like "garbage collection". In particular, for each clock, just one past event can be addressed, limiting the number of events in an event zone

This gives rise to the data structure of event zones: An event zone consists of a set of variables corresponding to event occurrences $E = \{e_1, \ldots, e_n\}$, two vectors from references (mostly for the clocks, some for causality) to E, one for use with upper bound constraints to future events, another for use with lower bound constraints to future events. Moreover, there is a rectangular matrix of constraints between elements of E, more precisely between events e that can be linked to future events by an upper bound and events f that can be linked to future events by a lower bound.

As a transition occurs,(1) add a new event, (2) recompute constraints with respect to its links to previous events (this is an incremental Floyd-Warshall algorithm that is at worst of quadratic complexity), (3) change references, (4) collect garbage lines and columns in the matrix that lack a reference on either side (upper or lower bound), then (5) remove completely unreferenced events.

Static analysis techniques used to check whether a certain clock will be referenced before its next reset, may be integrated here and refined to consider separately its use in upper and in lower bounds. Note also that the dimension of the constraints is not constant and implicitly compact (an event resetting two clocks will be represented by just one dimension). The initial zone has dimension 1. As the system evolves, the number of dimensions may grow but is limited to $n + m$, where n is the number of clocks and m is a measure for the degree of parallelism in the system. In case of a fully sequential system, this bound is equal to $n + 1$ which corresponds to the dimension of classical zones (one dimension per clock and one dimension for "0").

Depending on the constraints inserted into event zones, we may or may not obtain the same semantics on the level of state reachability. However, the transitions of the symbolic transition system are not identical, compared to IF for instance, event zones do not provide a transition to symbolize an unspecific passage of time. It is therefore obvious that event zones are more than just a different representation of constraints but are tightly coupled to the code of the transition generator of the symbolic transition system.

4 Status of development and future work

The development of ELSE is recent, up to now the invested effort is about 10 person months. The complete translation chain is running, i.e. syntactic analysis, semantic analysis and code generation, and there exists a prototype implementation of the elsezone library. However, coverage of the IF-2.0 language is very partial, we add code when we need it for modelling.

We have begun experimenting with the prototype, in particular exploring artificial academic examples where the event zone approach seems superior to classical zones. On some example series, exponential savings in running Else against itself with the two semantics have been achieved.

The main direction in the near future is to improve the efficiency of the zone library and to fully integrate partial order reductions that are currently only partially realized.

References

[AD94] R. Alur and D. Dill, *A theory of timed automata*, Theoretical Computer Science **126(2)** (1994), 183–235.

[BFG+99] Marius Bozga, Jean-Claude Fernandez, Lucian Ghirvu, Susanne Graf, Jean-Pierre Krimm, and Laurent Mounier, *IF: An intermediate representation and validation environment for timed asynchronous systems*, World Congress on Formal Methods (1), 1999, pp. 307–327.

[BJLY98] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, *Partial order reductions for timed systems*, Proceedings, Ninth International Conference on Concurrency Theory, Lecture Notes in Computer Science, vol. 1466, Springer-Verlag, 1998, pp. 485–500.

[BLP+99] G. Behrmann, K. Larsen, J. Pearson, C. Weise, W. Yi, and J. Lind-Nielsen, *Efficient timed reachability analysis using clock difference diagrams*, International Conference on Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 1633, 1999, pp. 341–353.

[DT98] D. D'Souza and P.S. Thiagarajan, *Distributed interval automata: A subclass of timed automata*, 1998, Internal Report TCS-98-3.

[DY96] C. Daws and S. Yovine, *Reducing the number of clock variables of timed automata*, IEE Real-Time Systems Symposium, December 1996, pp. 73–81.

[God96] P. Godefroid, *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, Lecture Notes in Computer Science, vol. 1032, Springer-Verlag Inc., New York, NY, USA, 1996.

[LNZ03] Denis Lugiez, Peter Niebert, and Sarah Zennou, *Clocked mazurkiewicz traces for partial order reductions of timed automata*, 2003, draft.

[LPY95] K. Larsen, P. Pettersson, and W. Yi, *Model-checking for real-time systems*, Fundamentals of Computation Theory, Lecture Notes in Computer Science, August 1995, Invited talk, pp. 62–88.

[Min99] Marius Minea, *Partial order reduction for verification of timed systems*, Ph.D. thesis, Carnegie Mellon University, 1999.

[NHZL01] P. Niebert, M. Huhn, S. Zennou, and D. Lugiez, *Local first search – a new paradigm in partial order reductions*, International Conference on Concurrency Theory (CONCUR), LNCS, no. 2154, 2001, pp. 396–410.

[NMA+02] P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, N. Jain, and O. Maler, *Verification of timed automata via satisfiability checking*, Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS, vol. 2469, 2002, pp. 225–244.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine, *Compiling real-time specifications into extended automata*, IEE Transactions on Software Engineering, vol. 18, September 1992, pp. 794–804.

[Pel93] D. Peled, *All from one, one for all: On model checking using representatives*, International Conference on Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 697, 1993, pp. 409–423.

[Val89] A. Valmari, *Stubborn sets for reduced state space generation*, 10th International Conference on Application and Theory of Petri Nets, vol. 2, 1989, pp. 1–22.

[YS97] Tomohiro Yoneda and Bernd-Holger Schlingloff, *Efficient verification of parallel real-time systems*, Formal Methods in System Design **11** (1997), no. 2, 197–215.