ObsSlice: A Timed Automata Slicer based on Observers

Víctor Braberman[1] *, Diego Garbervetsky[1], and Alfredo Olivero[2]**

[1] Computer Science Department, FCEyN, Universidad de Buenos Aires, Argentina
{vbraber|diegog}@dc.uba.ar
[2] Department of Information Technology, FIyCE, Universidad Argentina de la Empresa, Argentina aolivero@uade.edu.ar

Abstract. ObsSlice is an optimization tool suited for the verification of networks of timed automata using virtual observers. It discovers the set of modelling elements that can be safely ignored at each location of the observer by synthesizing behavioral dependence information among components. ObsSlice is fed with a network of timed automata and generates a transformed network which is equivalent to the original one up to branching-time observation. Experiments suggest that the approach may lead to significant time and space savings during verification phase as well as reductions in the length of counterexamples.

1 Introduction

In formal models of concurrent systems, safety and liveness requirements are commonly expressed in terms of virtual components (observers) which are composed in parallel with the set of components that constitutes the system under analysis (*SUA*). Our tool ObsSlice, based on [1], is fed with a *SUA* and an observer specified as a network of Timed Automata (TAs) and statically discovers, for each observer location, a set of modelling elements (automata and clocks) that can be safely ignored without compromising the validity of TCTL formulas stated over the observer (i.e., an exact reduction method wrt. branching time analysis). Eliminating irrelevant activity seems to mitigate state space explosion and have a positive impact on the performance of verification tools in terms of time, size and length of counterexamples. ObsSlice seems to be well suited for treatment of models comprising several concurrent timed activities over observers that check for the presence of event scenarios (e.g, [2]).

2 ObsSlice Architecture

Figure 1 shows a modular view illustrating the way ObsSlice solves the stated slicing problem by combining concepts presented in [1] . Currently, the tool takes

* Research supported by UBACyT grant X156, ANCyT grant PICT 11738, Microsoft Research Embedded Innovation Award
** Research supported by UADE grant ING6-01

a network of TAs compatible with KRONOS [3] and OPENKRONOS [4] formats and an I/O classification of labels that appear in each TA.The main goal of the

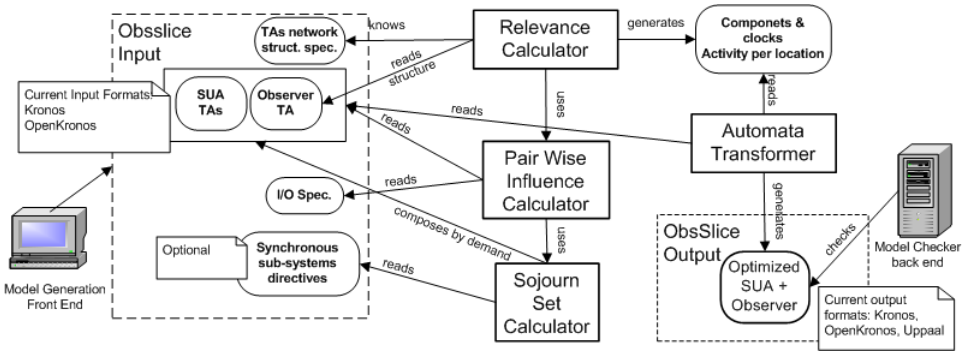

Fig. 1. OBSSLICE architecture

`Relevance Calculator` module is to estimate, for each observer location, a set of components and clocks whose activity can be safely ignored during the observed evolution of the *SUA*. To achieve that goal, it relies on `Pair Wise Influence Calculator` which statically calculates if a given component A may influence the behavior of another component B when sojourning a given observer location. Currently, I/O specification (uncontrollable/controllable classification of events for each automata as stated in [5]) helps to check potential influence due to communication, assignments or predicates. I/O declarations are, in general, intuitively known by modelers or can be automatically provided by high-level front-end modelling languages [3].

On the other hand, `Sojourn Set Calculator` provides an over approximation of the set of locations that may be traversed by a given component when sojourning a given observer location. This information serves as a way to obtain a more precise pair-wise influence prediction. `Sojourn Set Calculator`, by default, performs an untimed composition of each of the *SUA* TAs with the observer. Optionally, the sojourn set calculus can be improved by specifying which sets of TAs should be composed together due to a suspected synchronous behavior among them (`synchronous subsystem directives`).

Finally, the `Automata Translator` receives the *activity tables* and generates the network of transformed TAs. The enabling and disabling of modelling elements is achieved differently depending on the target dialect. Currently, the transformed models can be checked by KRONOS [3], OPENKRONOS [4] and UPPAAL [6]. For KRONOS and OPENKRONOS models, the reduction is done by adding sleep locations and the corresponding transitions (when possible, deactivation of clocks is also informed together with the model). For UPPAAL, a similar

[3] OBSSLICE is robust in the sense that a wrongly specified I/O label classification would only compromise the exactness of the method but reachability results would still be conservative.

approach is taken except that broadcasting is replaced by committed chains of disabling and enabling transitions.

3 Experiments

Being a preprocessing tool based on an exact reduction technique, ObsSlice is suited for integration with virtually any verification strategy built in current modelcheking tools [4]. Experiments reported are not meant to be benchmarks of modelchecking engines but rather they aim at showing the improvements achieved by "ObsSlicing" models for different tools and features. OpenKronos tool was run with the option `profounder` enabling a DFS strategy (DBMs as data structure). This option was run for the cases when the error state is indeed reachable. On the other hand, Uppaal was run using the `-Was` option and BFS traversal (in order to generate the whole state space when error is not reachable), `-Was -t1` to generate the shortest trace to the error (Minimal Constraint system as data structure), and `-Was -A` to apply a conservative abstraction (convex hull) for some unreachable cases. Table 1 shows the examples sizes: number of TAs of the *SUA*, clocks (*SUA*+observer) and observer details (locations and transitions). Table 2 shows times and memory consumed by the modelchecking tools over the original and obsSliced models. We also provide the size of the shortest trace [5] and the time consumed to generate it. Time consumed by ObsSlice itself is not reported since it is negligible compared with verification times (less than a couple of seconds).

Model	#TAs	#Clocks	Observer	
			#Loc	#Tran
Fddi10	21	32	21	221
Pipe6	13	14	15	197
Pipe7	15	16	17	227
RemoteBR	12	13	29	395
MinePump	8	8	9	58

Table 1. Examples sizes

`Fddi10` is an extension of the FDDI token ring protocol similar to the one presented in [1] where the observer monitors the time the token takes to return to a given station. `Pipe6` and `Pipe7` are pipe-lines of sporadic processes that forward a signal emitted by a quasi periodic source [1] (6 and 7 stages resp.). The observer captures a scenario violating an end-to-end constraint for signal propagation. The rest of examples are designs of distributed real-time system generated using the technique presented in [7]. Observers were obtained using VTS [2], a tool that automatically produces timed observer from scenario specifications. `RemoteBR` is a design [2] composed of a central component and two remote sensors. When the central component needs sampled data, it broadcasts a request to the sensors. Each sensor handles requests by reading the last stored value by the sampler and sending it back to the central component. There, the readings are paired for further processing. The observer captures scenarios where a request for collecting a pair of data items is not fully answered in less than a given amount of time. `Minepump` is a design of a fault-detection mechanism for a distributed mine-drainage controller [8]. A watchdog task periodically checks the availability of a water level sensor device by sending a request and extracting acks that were received and queued during the previous cycle by another sporadic task. When the watchdog finds the queue empty, it registers a fault condition in

[4] for a discussion on related work, please refer to [1]

[5] including committed synchronization with the observer

a shared memory which is periodically read, and forwarded to a remote console, by a proxy task. The observer checks if the failure of high-low water sensor is always informed to the remote operator before a given deadline.

Model	OpenKronos		Uppaal property satisfied				Uppaal prop. not satisfied			
	Original	ObsSliced	Original		ObsSliced		Original		ObsSliced	
	Time	Time	Time	Depth	Time	Depth	Time	Mem	Time	Mem
Fddi10	630.08s	**1.21s**	835.95s	32	**0.65s**	**32**	O/M	O/M	**0.44s**	**5.09**
c.h. (-A)							1141.24s	230.28	**0.15s**	**5.02**
Pipe6	994.20s	**0.05s**	306.76s	103	**31.59s**	**56**	21.11s	16.06	**6.77s**	**9.59**
Pipe7	O/M	**0.03s**	O/M	O/M	**324.44s**	**65**	407.36s	162.79	**84.02s**	**49.87**
RemoteBR	O/M	**1.10s**	O/M	O/M	**1.69s**	**101**	O/M	O/M	**1.75s**	**6.23**
c.h. (-A)							29.72s	19.68	**0.93s**	**8.60**
MinePump	O/M	**0.86s**	368.75s	81	**10.82s**	**54**	2856.47s	139.43	**65.66s**	**20.02**

Table 2. Verification benchmarks (Mem expressed in MB, Time in seconds)

Experiments were run on a SunBlade 2000 with 2GB RAM. Please notice the important savings in verification times (even using convex hull when the whole state space cannot fit in memory [6]), memory consumed and size of counterexamples.

A Java version of our tool together with a set of examples can be found at `http://www.dc.uba.ar/people/proyinv/rtar/obsslice`.

4 Future Work

Future extensions comprise end-to-end support of more timed automata dialects. We also plan to extend the concept of influence at a finer grain of analysis (not only at the location basis) and to use time information to make a more precise analysis of sojourn sets. On the other hand, we believe that our abstraction based on activity, can be cheaply performed on-the-fly by adapting verification engines therefore avoiding the use of chains of committed locations that produce an unnecessary state-space traversal.

References

1. V. Braberman, D. Garbervetsky, and A. Olivero. Improving the verification of timed systems using influence information. In *TACAS-02, LNCS 2280*, pages 21–36, 2002.
2. A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero. Visual timed event scenarios. In *Accepted for publication ICSE 2004*, 2004.
3. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. *The Tool KRONOS*. In *Proceedings of Hybrid Systems III*, volume 1066 of *LNCS*, pages 208–219. Springer-Verlag, 1996.
4. S. Tripakis. *L'Analyse Formelle des Systemès Temporisés en Practique*. Ph d. thesis, Univesité Joseph Fourier, 1998.
5. V. Braberman and A. Olivero. Extending timed automata for compositional modeling healthy systems. In *Proc. of MTCS-01, ENTCS 52*, 2001.
6. G. Behrmann, A. David, K.G. Larsen, O. Mller, P. Pettersson, and W. Yi. Uppaal - present and future. In *IEEE CDC-01*. IEEE Computer Society Press, 2001.
7. V. Braberman and M. Felder. Verification of real-time designs: Combining scheduling theory with automatic formal verification. In *ESEC/FSE-99, LNCS 1687*, 1999.
8. V. Braberman. *Modeling and Checking Real-Time Systems Designs*. PhD thesis, FCEyN, Universidad de Buenos Aires, 2000.

[6] Moreover, for Minepump, -A option is useless since it yields a MAYBE result.