ObsSlice: A Timed Automata Slicer based on Observers

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

[1] Departamento de Computación – FCEyN, Universidad de Buenos Aires, Argentina
{vbraber|diegog}@dc.uba.ar
[2] Centro de Estudios Avanzados, Universidad Argentina de la Empresa, Argentina
aolivero@uade.edu.ar

Abstract. ObsSlice is an optimization tool suited for the verification 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 one provided up to branching-time observation. Preliminary results have proven that eliminating irrelevant activity mitigates state space explosion and has a positive -and sometimes dramatic- impact on the performance of verification tools in terms of time, size and counterexample length.

1 Introduction

A common practice in the verification of concurrent systems is to express safety and liveness requirements as virtual components (observers) and composed in parallel with 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 illustrates in a modular view the way ObsSlice solves this slicing problem by combining concepts presented in [1]. Currently, the tool takes a network

* Research partially supported by UBACyT 2004 X020, ANCyT grant BID 1201/OC-AR PICT 11738, Microsoft Research Embedded Innovation Excellence Award
** Research partially supported by UADE projects ISI03B and TSI04B

of TAs compatible with KRONOS [3] and OPENKRONOS [4] formats and an I/O classification of the labels in the TAs. The main goal of `Relevance Calculator`

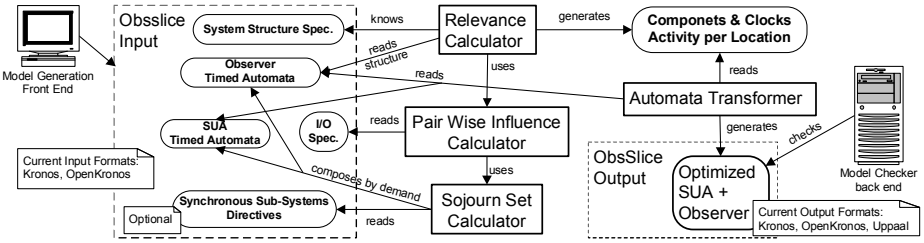

Fig. 1. OBSSLICE architecture

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 C may influence the behavior of another component C' when sojourning a given observer location. Currently, I/O classification of events for each automata helps to check potential influence due to communication, assignments or predicates. I/O declarations are, in general, intuitively known by verification engineers or can be automatically provided by high-level front-end modelling languages. 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. 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 an 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`). In addition to SUA TAs, constraint automata can be used to compactly express ordering of events in the composed system (e.g., FIFO ordering in a communication chain of components). Such constraints can be derived through abstraction techniques or directly proposed by verification engineers, in order to keep `Sojourn Set Calculator` from performing large compositions.

Finally, the `Automata Transformer` 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 [5]. The reduction is performed through the addition of sleep locations and their corresponding transitions (when possible, deactivation of clocks is also informed together with the model).

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. For a discussion on related work, please refer to [1]. OpenKronos tool was run using DFS search order and DBMs as state space representation when error is known to be reachable. Uppaal was run using `-Was` option and BFS order generating the whole state space when error is not reachable, `-Was -t1` to generate the shortest trace to the error, and `-Was -A` to apply a conservative abstraction (convex hull) for some of the unreachable error cases. Table 1 shows the examples sizes: number of TAs of the SUA, clocks of SUA+observer and the number of locations and transitions of the observers. Table 2 shows times and memory consumed by the modelchecking tools over the original and "obssliced" models. We also provide the length of the shortest trace (including committed synchronization with the observer) 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).

SUA		SUA+O	Observer	
Name	#TAs	#Clocks	#Loc	#Tran
`Fddi10`	21	32	21	221
`Pipe6`	13	14	15	197
`Pipe7`	15	16	17	227
`RemoteS.`	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] (with 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 [6]. Observers were obtained using VTS [2], a tool that automatically produces timed observer from scenario specifications. `Remote Sensing` [2], is a system consisting of a central component and two remote sensors. Sensors periodically sample a set of environmental variables and store their values in shared memory. When the central component needs them, it broadcasts a signal to the sensors. Each sensor runs a thread devoted to handle this message by reading the last stored value from shared memory and sending it back to the central component. The latter pairs the readings so that another process can use that piece of information to perform certain actions on some actuators. 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 [7]. A watchdog task periodically checks the availability of a water level sensing device by sending a request and extracting acknowledgements 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 a shared memory which is periodically read, and forwarded to a remote console, by a proxy task. The proposed scenario verifies that the failure of the high-low water sensor is always

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

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

informed to the remote operator before a given deadline. Experiments were run on a SunBlade 2000 with 2GB RAM and show important savings in verification times (even using convex hull abstraction[3]), memory consumed and length 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 Past and Future

ObsSlice evolves from a manually-integrated proof of concept tools [1]. Currently, it features improved transference algorithms, a weakened set of influence rules, clock deactivation rules, and a brand new set of transformation rules that work for broadcast model of communication (produces transformed networks ready to be checked with Uppaal). Besides, it provides flexible mechanisms to guide sojourn set calculus. 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. We believe that slicing over observers is an idea that may also be applied to other fields such as the verification of concurrent and distributed applications where asynchronicity exacerbates state space explosion.

References

1. Braberman, V., Garbervetsky, D., Olivero, A.: Improving the verification of timed systems using influence information. In: Proc. TACAS '02, LNCS 2280. (2002)
2. Alfonso, A., Braberman, V., Kicillof, N., Olivero, A.: Visual timed event scenarios. In: Proc. of the 26th ACM/IEEE ICSE '04 (to appear). (2004)
3. Daws, C., Olivero, A., Tripakis, S., Yovine, S.: *The Tool KRONOS.* In: Proc. of Hybrid Systems III, LNCS 1066, Springer-Verlag (1996) 208–219
4. Tripakis, S.: L'Analyse Formelle des Systemès Temporisés en Practique. PhD thesis, Univesité Joseph Fourier (1998)
5. Behrmann, G., David, A., Larsen, K., Möller, O., Pettersson, P., Yi, W.: Uppaal - present and future. In: Proc. IEEE CDC '01, IEEE Computer Society Press (2001)
6. Braberman, V., Felder, M.: Verification of real-time designs: Combining scheduling theory with automatic formal verification. In: ESEC/FSE '99, LNCS 1687. (1999)
7. Braberman, V.: Modeling and Checking Real-Time Systems Designs. PhD thesis, FCEyN, Universidad de Buenos Aires (2000)

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