TSMV: A Symbolic Model Checker for Quantitative Analysis of Systems

Nicolas Markey
Département d'Informatique
Université Libre de Bruxelles
1050 Bruxelles – Belgium
nmarkey@ulb.ac.be

Philippe Schnoebelen
Lab. Spécification & Vérification
ENS de Cachan & CNRS UMR 8643
94235 Cachan – France
phs@lsv.ens-cachan.fr

***Abstract*— TSMV is an extension of NuSMV, the open-source symbolic model checker, aimed at dealing with timed versions of (models of) circuits, PLC controllers, protocols, etc. The underlying model is an extension of Kripke structures, where every transition carries an integer duration (possibly zero). This simple model supports efficient symbolic algorithms for RTCTL formulae.**

I. Checking models with durations on steps

TSMV extends NuSMV (the open-source version of SMV [3]) and provides RTCTL model checking for "(Simply) Timed Kripke Structures", or "TKS's", *i.e.* models where transitions carry an arbitrary *duration*.

RTCTL is a well-known timed extension of CTL that allows quantitative constraints on modalities [4]. One can write "timed" specifications like $\mathsf{AG}(\mathtt{req} \Rightarrow \mathsf{AF}_{\leq 100}\mathtt{grant})$ requiring that the `grant` always eventually arrives in at most 100 time units (t.u.) after the request.

With (Nu)SMV, one step in the Kripke structure equals one time unit. With TSMV one can specify arbitrary natural numbers (possibly 0) as durations of the steps.

The advantage of modeling with TKS's is twofold. First, it allows for succinct encoding of long steps. A transition that takes 100 t.u. would be encoded with 99 intermediary states in standard NuSMV. For such long steps, TSMV has a special semantics where no intermediate state occurs [6]. This small change in semantics makes RTCTL model checking provably easier [5]. Second, TKS's may have zero-duration transitions. These are convenient for modeling micro-steps that are part of one single macro-step. They are also a very convenient way of counting specific events or conditions along a path (by assigning null durations to the other steps). In this sense, TSMV supports a temporal logic with condition-counting facilities.

TSMV has specific algorithms for verifying RTCTL specifications on TKS's, as well as for computing minimum and maximum delays between sets of states. These algorithms are more or less insensitive to scaling-up of durations [6]. They were easily implemented on top of NuSMV.

II. Verifying the PCI local bus

Campos *et al.* verified a SMV model of the PCI local bus to illustrate their condition counting algorithms [2]. Here we describe how TSMV handles the same model (available in the NuSMV distribution): One simply adds the lines below.

One introduces an extra variable, `duration`, used to define the duration of the current step. For counting the *number of transactions started* between a request and a grant, we use:

```
VAR duration: 0..1;
ASSIGN next(duration)=case
          start_transaction: 1;
          1                : 0;
       esac;
```

We compute the maximum number of transactions started between a bus request by the processor and its granting with:

```
COMPUTE MAX[cpu.req, cpu.grant]
```

The answer is 2.

If we restrict our attention to the transactions *started by the processor*, we modify the `ASSIGN` statement above and the `COMPUTE MAX` instruction now returns 0. We can check that transactions are only initiated once between two grants with the following formula:

```
SPEC (AF=0 cpu.grant) &
     (AG (cpu.grant -> AF=1 (cpu.grant)) )
```

Obviously this property is not satisfied in general, but it holds if some fairness assumption on processor requests is made. This is achieved with the following line:

```
FAIRNESS  cpu.req
```

III. Insensibility to scaling up

Thanks to efficient BDD manipulation (*e.g.* detection of interesting durations, splitting of the transition relation [6]), our algorithms are almost insensitive to scaling up of durations. We illustrate this claim with scaled-up variants of the "bridge crossing" problem. The execution time increases very slowly, compared to other symbolic discrete-time model-checkers.

The bridge-crossing problem is a famous mathematical puzzle with time critical aspects [7]. A group of four persons, called P1, P2, P3 and P4, cross a bridge at night. It is dark and one can only cross the bridge with a lamp. Only one lamp is

	TSMV		NuSMV		Verus		RAVEN	
	time	memory	time	memory	time	memory	time	memory
bridge	0.02 s.	1.3 MB	0.13 s.	9.5 MB	7.14 s.	16.5 MB	4.01 s.	5.9 MB
bridge × 10	0.04 s.	1.3 MB	4.14 s.	18.5 MB	259.54 s.	39.2 MB	3 098 s.	371 MB
bridge × 20	0.07 s.	1.3 MB	22.44 s.	19.5 MB	573.05 s.	44.1 MB		
bridge × 50	0.22 s.	9.0 MB	176.55 s.	28.0 MB	3 626 s.	55.0 MB		
bridge × 100	0.45 s.	11.0 MB	844.25 s.	50.0 MB	17 870 s.	59.2 MB		

TABLE I
SCALE UP (IN)SENSITIVITY

available and at most two persons can cross at the same time. Therefore any solution requires that, after the first two persons cross the bridge, one of them returns, carrying back the lamp for the remaining persons. The four persons have different maximal speeds: Here P1 crosses in 5 time units (t.u.), P2 in 10 t.u., P3 in 20 t.u. and P4 in 25 t.u. When a pair crosses the bridge, they move at the speed of the slowest person in the pair. Now, how much time is required before the whole group is on the other side?

A person is described as an SMV module with his crossing time as a parameter. His possible steps are to stay where he is, or move to the other side. He can only cross when the lamp is on his side (and then the lamp crosses with him). When he crosses, the transition takes at least his crossing time. This way, when four persons are synchronized, the crossing time is any integer greater or equal to the maximum crossing time of the crossing persons. The complete system is obtained by combining four persons (four instances of the same `person` module, with different crossing times) with a Boolean `lamp` value keeping track of the position of the lamp, and adding a further constraint (an `INVAR` in SMV) telling that at most two persons cross in one move.

We can ask how much time is required for crossing:

```
COMPUTE MIN[initial, final]
```

`initial` and `final` being defined to suit our needs. The answer (60 t.u.) is obtained in a few milliseconds.

The same example can be treated with *e.g.* NuSMV, Verus or RAVEN. Since NuSMV and Verus only handle unit steps, we use the method advocated in [1, p.106] and introduce a counter forcing several t.u. between actual system moves. With RAVEN, we can directly specify duration intervals for each transitions, even though it internaly considers the semi-continuous semantics. For this basic case, those tools are slightly slower than TSMV.

When we define a model "`bridge x 10`" by replacing 5, 10, 20 and 25 with (resp.) 50, 100, 200, and 250, TSMV computes the minimum delay of 600 t.u. in more or less the same time it needed for the initial problem.

By contrast, the computation time for NuSMV, Verus and RAVEN increases dramatically when we scale up the durations. In fact, there is no way to avoid this: These tools do not know about TKS's and are bound to compute all sets associated with different values of the counter for intermediate states. Computing these sets is a tedious and mostly repetitive task that cannot be avoided unless a notion of TKS is introduced.

REFERENCES

[1] S. V. A. Campos and E. M. Clarke. The Verus Language: Representing Time Efficiently with BDDs. *Theoretical Computer Science*, 253(1), pages 95–118, Elsevier Science, Feb. 2001.

[2] S. V. A. Campos, E. M. Clarke, W. R. Marrero, and M. Minea. Verifying the performance of the PCI local bus using symbolic techniques. In *Proceedings of the 5th International Conference on Computer Design (ICCD'95)*, Oct. 1995, pages 72–78. IEEE Comp. Soc. Press, Oct. 1995.

[3] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A New Symbolic Model Verifier. In N. Halbwachs and D. A. Peled, eds, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, July 1999, vol. 1633 of *Lecture Notes in Computer Science*, pages 495–499. Springer-Verlag, July 1999.

[4] E. A. Emerson, A. K.-L. Mok, A. P. Sistla, and J. Srinivasan. Quantitative Temporal Reasoning. *Real-Time Systems*, 4, pages 331–352, Kluwer Academic, 1992.

[5] F. Laroussinie, N. Markey, and Ph. Schnoebelen. On Model Checking Durational Kripke Structures (Extended Abstract). In M. Nielsen and U. Engberg, eds, *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS 2002)*, Apr. 2002, vol. 2303 of *Lecture Notes in Computer Science*, pages 264–279. Springer-Verlag, Apr. 2002.

[6] N. Markey and Ph. Schnoebelen. Symbolic Model Checking for Simply-Timed Systems. In *Proceedings of the Joint Conferences Formal Modelling and Analysis of Timed Systems* (FORMATS 2004) and *Formal Techniques in Real-Time and Fault-Tolerant Systems* (FTRTFT 2004), Lecture Notes in Computer Science. Springer-Verlag. To appear.

[7] G. Rote. Crossing the Bridge at Night. *EATCS Bulletin*, 78, pages 241–246, EATCS, Oct. 2002.