MOBY/DC – A Tool for Model-Checking Parametric Real-Time Specifications

Henning Dierks[1] and Josef Tapken[2]

[1] University of Oldenburg,
Department of Computer Science,
P.O.Box 2503
26111 Oldenburg, Germany
[2] cewe digital GmbH,
Meerweg 30-32,
26133 Oldenburg, Germany

Abstract. We define an operational subset of Duration Calculus, called phase automata, which serves as an intermediate language for the analysis and verification of real-time system descriptions that contain timing parameters. We introduce the tool MOBY/DC which implements a model-checking algorithm for phase automata. The algorithm applies compositional model-checking techniques and handles parameters by built-in procedures or by a link to $CLP(R)$. Due to the parameters the model-checking problem is undecidable in general. Hence, we have to accept that the results are overapproximations only in order to guarantee termination. The overapproximation together with the compositional technique makes the model-checker especially well suited for proving the absence of error traces instead of finding them.

1 Introduction

Most timed automata model-checkers demand concrete values for the timing bounds. However, in high-level system descriptions often parameters are used for the timing bounds. To cope with parameters we designed and implemented a new model-checking algorithm for parametric phase automata. As an option the tool solves the parametric constraints by translation of the parametric timing bounds into fragments of constraint logic programs [JM94] which are fed into the interpreter $CLP(R)$ [HJM+92]. Due to an undecidability result for our parametric language [Tap01], we have to pay a certain price. Therefore, we chose an approximation technique in order to guarantee the termination of the model-checking procedure (in contrast to the semi-decision procedure of HYTECH [HHW97]).

Our model-checking algorithm was designed to complement the approaches provided by the timed automata model-checkers UPPAAL and KRONOS. The experiences with these model-checkers have shown that the strength of their approaches is to find errors in the specification rather than to prove their absence. We chose a compositional model-checking technique [And95] which is also implemented in the timed automata tool CMC [LL98]. Our algorithm often performs better for correctness proofs because it can happen that only few of the parallel components of the model are needed to prove

H. Garavel and J. Hatcliff (Eds.): TACAS 2003, LNCS 2619, pp. 271–277, 2003.

the absence of an error. To show the presence of an error always the whole network has to be examined.

2 Phase Automata

In this section we explain phase automata by an example[1]. The phase automata build an operational subset of the Duration Calculus [HZ97] since the semantics is given in Duration Calculus.

Example: Generalized Railroad Crossing: In [HL94] the case study of a "Generalized Railroad Crossing" (GRC for short) is proposed as a benchmark for the comparison of formal methods. The task is to develop a gate controller which shall secure a railroad crossing. The system is modeled by two observables. The first one describes the environment of the gate controller, which includes tracks and trains. The observable is called Track and has the range $\{\mathsf{E}, \mathsf{A}, \mathsf{Cr}\}$ with the following meaning: $\mathsf{Cr} \stackrel{\text{df}}{=}$ at least one train is in the crossing, $\mathsf{A} \stackrel{\text{df}}{=}$ at least one train is approaching the crossing but no train is in it, and $\mathsf{E} \stackrel{\text{df}}{=}$ no train is approaching or crossing, i.e. the track is empty. The second observable Gate describes the gate on an abstract level by the following three values: $\mathsf{Cl} \stackrel{\text{df}}{=}$ the gate is fully closed, $\mathsf{O} \stackrel{\text{df}}{=}$ the gate is fully open, and $\mathsf{I} \stackrel{\text{df}}{=}$ the gate is not open and not closed.

Two formal approaches [ORS96,DD97] to this case study using Duration Calculus based methods assume the following properties for trains:

1. It cannot happen that a train passes through the approaching area in less than ε_1 time units if the track was empty before.
2. The slowest train needs at most ξ_1 time units to approach the crossing.

In Fig. 1 a phase automaton is given (lhs) that expresses the first property. The automaton consists of four phases $(p_1, \ldots, p_4)$ and several transitions between them. Phases are inscribed by a DC state assertion (e.g. p_4 with $\lceil \mathsf{Cr} \rceil$). The assertions can be built over all observables of the system and do not have to be disjoint (cf. phase p_2 and p_3). Initial phases are marked by an incoming edge (see p_1, p_2, and p_4). To express timing aspects a phase automaton is equipped with a set of clocks (here c_1) each of them being inscribed by a time interval over the reals (in this example $[\varepsilon_1, \infty]$ for clock c_1). The bounds of the interval can be specified as ∞ or as a linear contraint, ie. $c + \sum c_i \cdot x_i$ where the c's are constants and the x's are parameters. Finally, clock scopes are defined by associating to each clock an arbitrary set of phases. Here, the scope of clock c_1 comprises only the phase p_2.

The informal behaviour of the left automaton in Fig. 1 is as follows. Initially, the automaton allows an arbitrary interpretation of the observables due to the state assertions of the initial phases. If the automaton is in phase p_1 with the track being empty ($\lceil \mathsf{E} \rceil$) and an approaching phase shall succeed, then the automaton has to change to p_2 (and not to p_3 due to the missing transition). The clock c_1 requires that the phase p_2 is stable for at least ε_1 time units. Thus, it is not allowed that an $\lceil \mathsf{A} \rceil$-phase is shorter than ε_1 time units if $\lceil \mathsf{E} \rceil$ precedes.

[1] For a full and formal treatment we refer the reader to [Tap01].

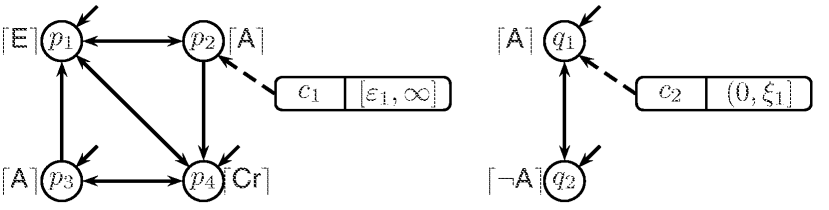

Fig. 1. Phase automaton for (1) and (2).

The property (2) is described by the right automaton in Fig. 1. The semantics of the parallel composition of these automata is simply the intersection of all interpretations of Track and Gate that belong to the semantics of the components. Thus, the parallel composition corresponds directly to the conjunction in Duration Calculus. The semantics of the cross product[2] of phase automata is equivalent to the semantics of the parallel composition. Phase automata without parameters can be transformed into timed automata [Mru00] and vice versa [Sch01].

3 Moby/DC

In Fig. 2 the specification of the GRC in MOBY/DC is presented. The description contains several text nodes with declarations of observables, parameters, and given constraints on those. Moreover, several phase automata are defined which either belong to the description of the system or describe *negated properties* of the system. Above we specified properties (1) and (2) by the phase automata in Fig. 1. Both are shown in Fig. 2 and named `Stab` resp. `Prog` with ε_1, ξ_1 as `e1` and `x1`.

In MOBY/DC each phase has as default a single clock with interval $(0, \infty]$ (written as `(0,w]`) allowing the system to stay there arbitrarily long — even forever. In case of $(0, \infty)$ the phase has to be left eventually. The requirements of the GRC are given in [HL94] as follows:

Safety: The gate is down during all occupancy intervals.
Utility: The gate is up if no train was crossing during the last ξ_2 seconds and no train will cross during the next ξ_1 seconds.

Both requirements are represented in Fig. 2 (`Safety` and `Utility`) as negations, i.e. as phase automata which express counter examples of these properties. The parameters ξ_1, ξ_2 (`x1`,`x2`) appear in `Utility` and in some of the specification automata.

The compositional model-checking algorithm is sketched in Fig. 3. It utilizes a function `isEmpty` that tests some sufficient conditions for emptiness. Hence, only the return value `true` is reliable. The procedure `minimise` applies reductions on a phase automaton which simplify the automaton in terms of phases and clocks. It returns a smaller automaton which possibly allows more behaviour.

[2] The syntactic cross product operation on phase automata is defined as expected except for the transition relation. Transitions (p, p') and (q, q') in the corresponding automata leads to transitions $((p, q), (p', q))$, $((p, q), (p, q'))$, *and* $((p, q), (p', q'))$.

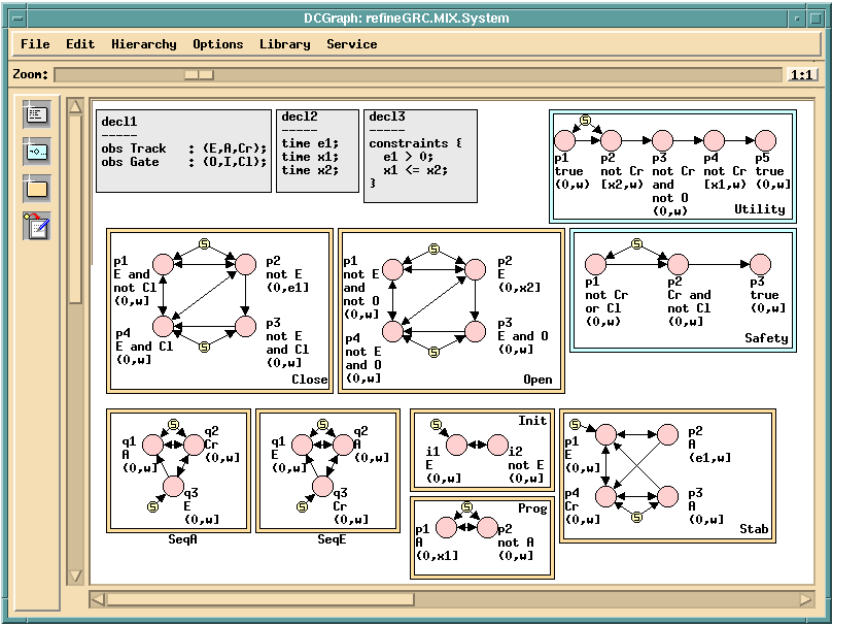

Fig. 2. A screenshot of MOBY/DC.

```
Procedure: semantics of A_1||...||A_n = ∅?
Input: A_1, ..., A_n;
    A := A_1; i := 2;
    minimise(A);
    while ( i ≤ n and not isEmpty(A) ) do
        A := A × A_i; i := i + 1;
        minimise(A);
    od
Output: isEmpty(A);
```

Fig. 3. Basic model-checking algorithm

To prove that a system $\mathcal{A}_1||\ldots||\mathcal{A}_k$ satisfies a requirement Req, the user has to provide a phase automaton $\mathcal{N}$ that describes (or overapproximates) $\neg Req$. If the model-checking computes that the semantics of $\mathcal{N}||\mathcal{A}_1||\ldots||\mathcal{A}_k$ is empty, then the system satisfies Req. Due to the construction of `isEmpty` and `minimise` only the positive result is reliable, i.e. the tool might produce false-negatives but no false-positives.

Due to space limitations we cannot explain `isEmpty` and `minimise` in detail. The analysis of the parameters is done in `minimise` where the model-checker tries to find superfluous transitions and phases in the products. This is the place where (optionally) $CLP(R)$ is employed. Its input is a set of constraints over the parameters and a partial

execution path of the product phase automaton. If $CLP(R)$ cannot find a solution of these constraints then the model-checker can identify a superfluous transition or phase.

4 Comparison

Now we use the GRC example for a performance comparison of MOBY/DC with the timed automata model-checker KRONOS (version 2.4.4[3]) and CMC (version 1.5). In order to apply the timed automata model-checkers we have to replace the parameters by concrete values. Therefore, we choose $\varepsilon_1 \stackrel{\text{df}}{=} 2$, $\xi_1 \stackrel{\text{df}}{=} 4$, and $\xi_2 \stackrel{\text{df}}{=} 4$ whereas MOBY/DC was working with the constraints $\varepsilon > 0$ and $\xi_1 \leq \xi_2$.

Table 1. Performance comparison for the GRC-example

phase automata	CMC	KRONOS	MOBY/DC
Safety, Close, SeqE, Init, Stab	1.5s	0.1s	0.7s
Safety, Close, SeqE, Init, Stab, Open, SeqA, Prog	2.1s	37.1s	0.7s
Safety, Open, SeqA, Prog, Close, SeqE, Init, Stab	29.8s	15.3s	0.9s
Utility, Open, SeqA, Prog, Stab	50.4s	0.2s	0.9s
Utility, Open, SeqA, Prog, Stab, Close, SeqE, Init	445.9s	5.3s	0.9s
Utility, Close, SeqE, Init, Open, SeqA, Prog, Stab	426.8s	8.3s	1.3s

Furthermore, we employ the translations of phase automata into timed automata [Mru00]. In both tools we check for the non-zenoness of the parallel composition of the timed automata. If it is non-zeno then the semantics is not empty. Both translations are similar and produce networks of timed automata instead of building the cross product. For this purpose the translations exploit the multisynchronisation concepts of the timed automata tools[4].

For the performance comparison we measured the execution times on an UltraSPARC-II with 296 MHz. The results are shown in Table 1. For both requirements we made checks with three different models. First we checked the smallest model which is necessary to prove the requirement. Then, we added the remaining automata which are not needed for the proof. Adding them at the end is better for the checkers which are based on compositional techniques. In contrast to CMC the performance of MOBY/DC is not at all influenced by automata added at the end. The reason for this difference is that CMC always examines the whole model and MOBY/DC can benefit from the monotonicity of the parallel composition of phase automata. KRONOS first builds the cross product of the given automata and then starts the reachability check. The KRONOS-entries in Table 1 comprises the time spent on model-checking and for building the cross product.

[3] using breadth first with the parameters -ai (inclusion abstraction) and -ax (extrapolation abstraction)

[4] Due to this fact the translations cannot be ported directly to Uppaal.

5 Related Work

Based on linear hybrid automata or parametric timed automata several tools have been developed: HyTech [HHW97] applies a semi-decision procedure on linear hybrid automata whereas LMPC [LTA98], TReX [ABS01] and a parametric extension of Uppaal work on parametric timed automata [HRSV02]. None of these tools apply compositional techniques for verification.

Acknowledgements. The authors thank E.-R. Olderog and the members of the "semantics group" in Oldenburg for fruitful discussions on the subject of this paper. Furthermore we thank the anonymous referees for their helpful suggestions and hints.

References

[ABS01] A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: A Tool for Reachability Analysis of Complex Systems. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of CAV 2001*, number 2102 in LNCS. Springer, 2001.

[And95] H.R. Andersen. Partial Model Checking. In *Proc. of the 10th Annual IEEE Symp. on Logic in Computer Science*, pages 398–407. IEEE Press, 1995.

[DD97] H. Dierks and C. Dietz. Graphical Specification and Reasoning: Case Study "Generalized Railroad Crossing". In J. Fitzgerald, C.B. Jones, and P. Lucas, editors, *FME'97*, volume 1313 of *LNCS*, pages 20–39, Graz, Austria, September 1997. Springer.

[HHW97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *STTT*, 1(1+2):110–122, December 1997.

[HJM+92] N.C. Heintze, J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap. *The CLP(R) Programmer's Manual, Version 1.2*, September 1992.

[HL94] C. Heitmeyer and N. Lynch. The Generalized Railroad Crossing. In *IEEE Real-Time Systems Symposium*, 1994.

[HRSV02] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear Parametric Model Checking of Timed Automata. *Journal of Logic and Algebraic Programming*, 2002.

[HZ97] M.R. Hansen and Zhou Chaochen. Duration Calculus: Logical Foundations. *FAC*, 9:283–330, 1997.

[JM94] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *The Journal of Logic Programming*, 19/20:503–582, May–July 1994.

[LL98] F. Laroussinie and K.G. Larsen. CMC: A Tool for Compositional Model-Checking of Real-Time Systems. In *Proceedings of IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic, November 1998.

[LTA98] R. Lutje Spelberg, H. Toetenel, and M. Ammerlaan. Partition Refinement in Real-Time Model Checking. In A.P. Ravn and H. Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1486 of *LNCS*, pages 143–157, Lyngby, Denmark, September 1998. Springer.

[Mru00] C. Mrugalla. Transformation of phase automata into timed automata. Master's thesis, University of Oldenburg, Department of Computer Science, Oldenburg, Germany, 2000. (in German).

[ORS96] E.-R. Olderog, A.P. Ravn, and J.U. Skakkebæk. Refining System Requirements to Program Specifications. In C. Heitmeyer and D. Mandrioli, editors, *Formal Methods for Real-Time Computing*, volume 5 of *Trends in Software*, pages 107–134. Wiley, 1996.

[Sch01] A. Schäfer. Fault tree analysis and real-time model-checking. Master's thesis, University of Oldenburg, Department of Computer Science, Oldenburg, Germany, 2001. (in German).

[Tap01] J. Tapken. *Model-Checking of Duration Calculus Specifications*. PhD thesis, University of Oldenburg, June 2001.