Optimizing Bounded Model Checking for Linear Hybrid Systems

Further experimental results

Erika Ábrahám[1], Bernd Becker[1], Felix Klaedtke[1,2], and Martin Steffen[3]

[1] Albert-Ludwigs-Universität Freiburg, Germany
[2] ETH Zurich, Switzerland
[3] Christian-Albrechts-Universität zu Kiel, Germany

In this document we describe our experimental results for the application of bounded model checking with optimization and explanation learning to linear hybrid systems. Furthermore, we describe termination conditions and their optimizations.

Section 1 contains short descriptions of the examples in our test suite. We report on our experimental results in Section 2. The termination conditions are described in Section 3.

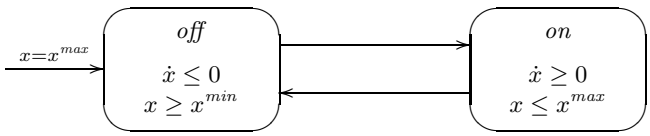
1 The hybrid systems

1.1 Thermostat

This is a modified version of the thermostat example in [1].

The temperature of a room is controlled by a *thermostat*, which continuously senses the temperature and turns a heater on and off. When the heater is off, the temperature, denoted by x, decreases according to the function $x(t) \leq x(0)$, where $x(0)$ is the initial temperature and t the time. With the heater off, the temperature increases, i.e., $x(t) \geq x(0)$. The initial temperature is x^{max} degrees ($x^{max} > 0$) and the heater is off initially. The heater turns on at latest when the temperature reaches a threshold value of x^{min} degrees ($0 < x^{min} < x^{max}$). Conversely the heater turns off at latest when the temperature reaches x^{max} degrees.

The temperature is to be kept between x^{min} and x^{max} degrees.



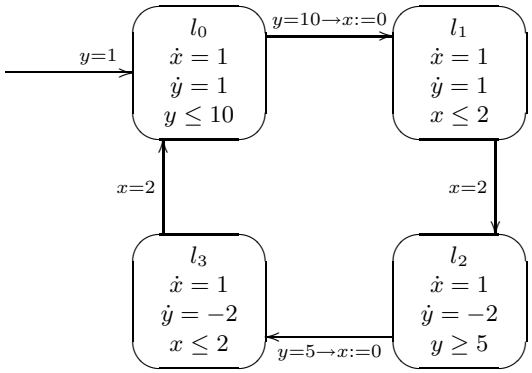
1.2 Water-level monitor

This is the *water-level monitor* example from [1]. The water level in a tank is controlled by a monitor, which continuously senses the water level and turns a pump on and off.

There is a *delay* of 2 seconds from the time the monitor signals a status change to the time that the change becomes effective. The water level changes as a piecewise-linear function over time. When the pump is off, the water level, denoted by the variable y, falls by 2 inches per second; when the pump is on, the water level rises by 1 inch per second. Suppose that initially the water level is 1 inch and the pump is turned on.

The monitor signals whenever the water level passes 5 and 10 inches, resp. The system has 4 locations: in locations l_0 and l_1, the pump is turned on; in locations l_2 and l_3, the pump is off. The clock x is used to specify the delays.

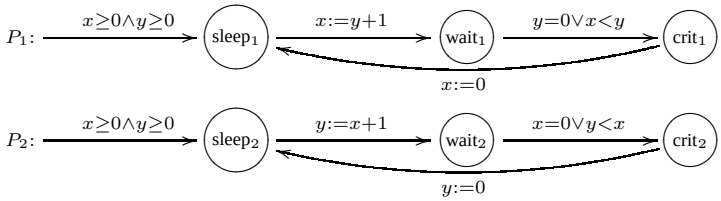
We show that the water level is always between 1 and 12 inches.



1.3 Bakery protocol

The Bakery protocol is a protocol assuring mutual exclusion between two or more processes. We analyze the protocol for two processes.

As you can see on the specification below, this example can be represented as a discrete system, i.e., time does not play a role.

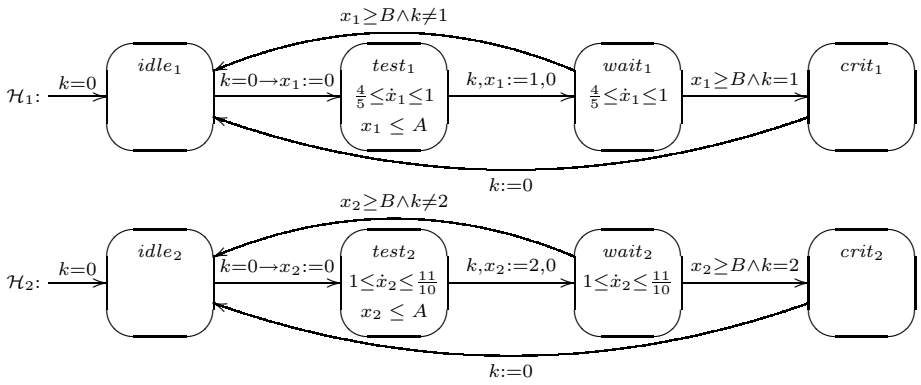


The protocol works as follows: If a process will get into the critical section, it takes a number that is greater than the number of the other process and moves from *sleep*$_i$ into *wait*$_i$. If the other process does not try to get into the critical section, i.e., if the number of the other process is 0, the process may move on into *crit*$_i$. Otherwise the process that took a number first may move into the critical section. The other process will wait until the critical section gets free and moves on afterwards. Exiting the critical section signs that the critical section got free by setting the number of the exiting process to 0.

We prove that $x \geq 0 \wedge y \geq 0 \wedge \neg(at_{P_1} = \mathit{crit}_1 \wedge at_{P_2} = \mathit{crit}_2)$ is an invariant property.

1.4 Fischer's mutual exclusion protocol

Fischer's mutual exclusion protocol assures mutual exclusion between processes. We apply the protocol for two processes. The corresponding hybrid systems are specified as follows:



If one of the processes i will get into its critical section, it waits until k gets 0 indicating that the other process is not in the critical section or in a location wait_i. Then it moves from location idle_i into test_i and may wait there some upper bounded time intervall. Afterwards the process moves on into location wait_i, sets k to its identity i, and waits there some lower bounded time intervall. The parameters A and B with $8B > 11A$ are chosen such a way that each process which is not in its idle_i location must enter wait_i before one of the competing processes enters its critical section. Finally, the process i which was the last one setting k to its own identity may enter its critical section. The other process j returns into location idle_j and tries the loop again.

We prove mutual exclusion by showing invariance of the property $x_1 \geq 0 \wedge x_2 \geq 0 \wedge \neg(at_{\mathcal{H}_1} = \mathit{crit}_1 \wedge at_{\mathcal{H}_2} = \mathit{crit}_2)$.

We deal also with Fischer's protocol for three processes. The third process is similar to the second one, where the index 2 is replaced everywhere by 3, and $k\#2$ with $\# \in \{=, \neq, :=\}$ is replaced by $k\#3$. In the case of three processes all flows increment the values of all clocks x_i, $i = 1, 2, 3$ according to $\dot{x}_i = 1$ and we assume $0 < A < B$. The invariant property is $x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_3 \geq 0 \wedge \bigwedge_{i,j \in \{1,2,3\},\, i \neq j} \neg(at_{\mathcal{H}_i} = \mathit{crit}_i \wedge at_{\mathcal{H}_j} = \mathit{crit}_j$

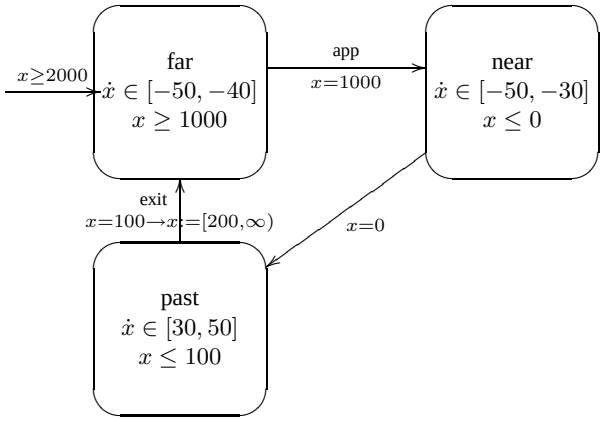
1.5 Railroad crossing

This system has three components: a train, a gate, and a controller. If the train gets near to the gate, the controller signals the gate to start to lower with some maximal delay

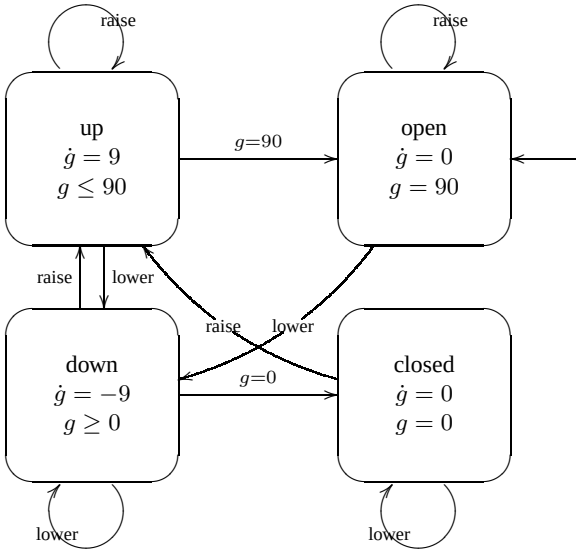
$\alpha < 49/5$. After the train has passed, the gate begins to raise, again with a maximal delay α.

We show that the gate is always fully closed when the train is within 10 meters to the gate.

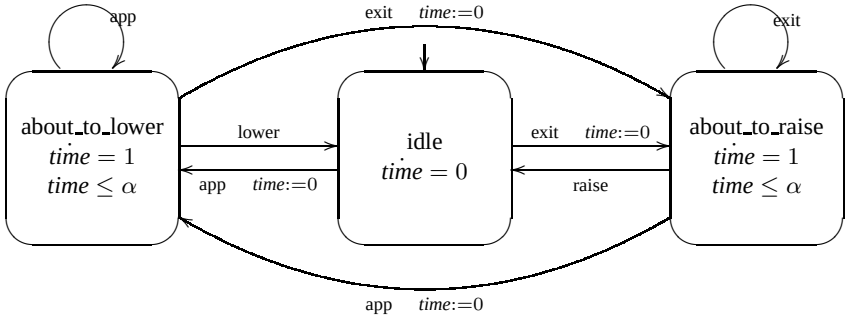
Train:



Gate:



Controller:

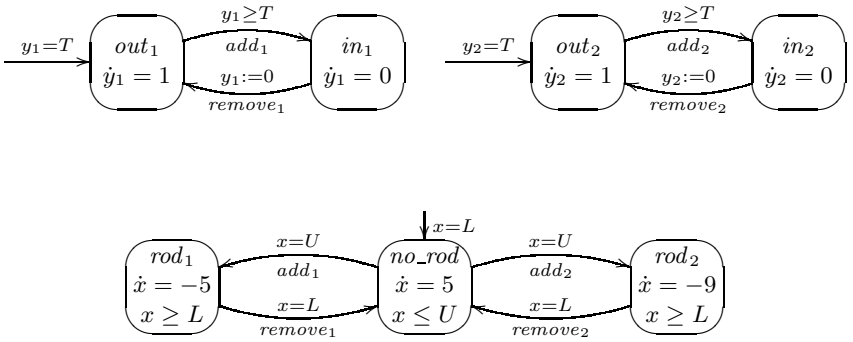


We also apply our method to an extended version of this example with two trains, see the HyTech homepage for the specification.

1.6 Nuclear reactor

The temperatur x within a nuclear reactor is controlled by a controller and two rods. If no rods are in the reactor, the temperatur increases by 5 degrees per second. Whenever the temperature reaches U degrees, one of the rods gets put into the reactor to decrease the temperature. The rod gets taken out when the temperature reaches L degrees. After a rod has been taken out, it is blocked for T seconds.

We show that whenever the temperature reaches U degrees, oneof the rods can be put in.



1.7 Audio-control protocol

This example first appeared in [2]. The hybrid automaton model is described in [4]. See also the HyTech homepage.

Example	Last iteration below 200 secs. of CPU time		
	naive	**optimized**	**optimized+learning**
Thermostat	70	> 1500	> 1500
Water-level monitor	39	> 1500	> 1500
Railroad crossing	14	52	872
Extended railroad crossing	10	12	80
Fischer's protocol (2 processes)	10	15	1254
Fischer's protocol (3 processes)	9	14	31
Bakery protocol (2 processes)	10	45	742
Nuclear reactor	20	82	> 1500
Audio-control protocol	20	62	357

Table 1. Maximal number of BMC iterations k.

2 Experimental results

We carried out tests for evaluating the BMC approach for linear hybrid systems with the different encodings and techniques described in the paper. All experiments were performed on a SUN Blade 1000 with 8 Gbytes of main memory and two 900 Mhz UltraSparc III+ processors; each one with an 8 Mbyte cache. We used ICS (version 2.0b) [3] for checking satisfiability of the formulas in the BMC approach.

As in the paper, we report on experimental results for the following three different encodings of finite runs: (A) the *naive* encoding; (B) the *optimized* encoding; (C) the optimized encoding as in (B) with additional *learning* of explanations.

2.1 Running times

Figures 1 and 2 show the running times for the encodings (A), (B), and (C) for our examples with k ranging from 0 to 200. Table 1 lists for each example the maximal iteration depth with each satisfiability check requiring less than 200 secs. CPU time.

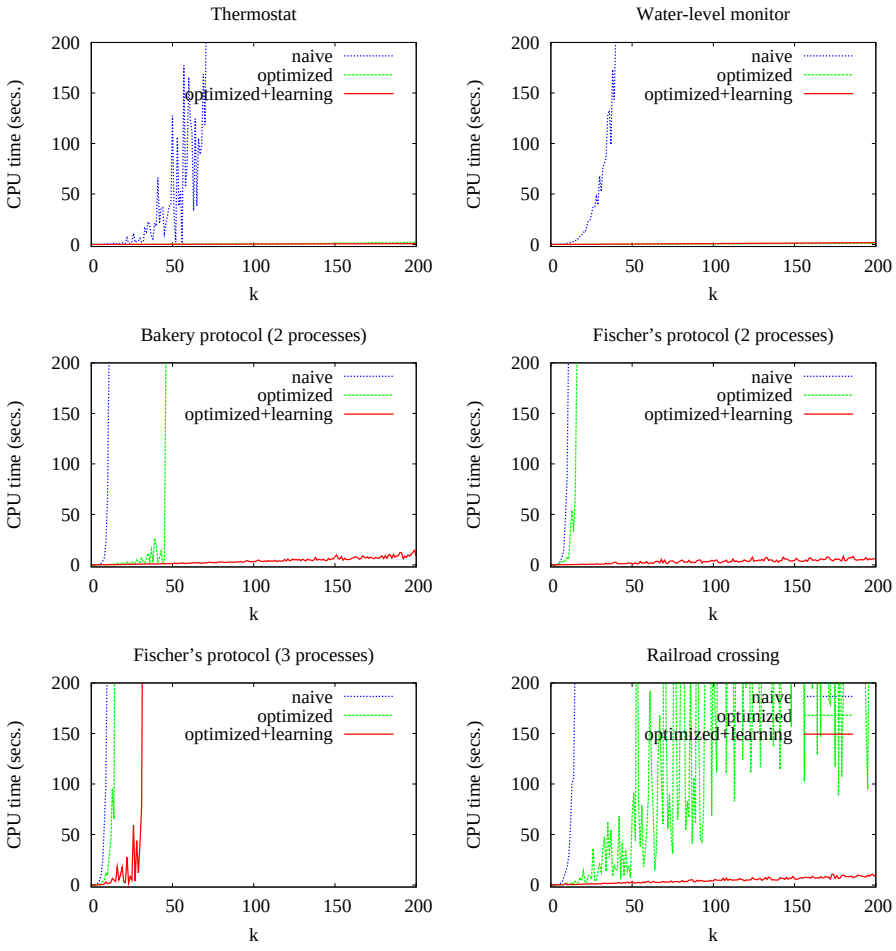


Fig. 1. Running times for the satisfiability checks for the naive encoding, the optimized encoding, and the optimized encoding with learning explanations.

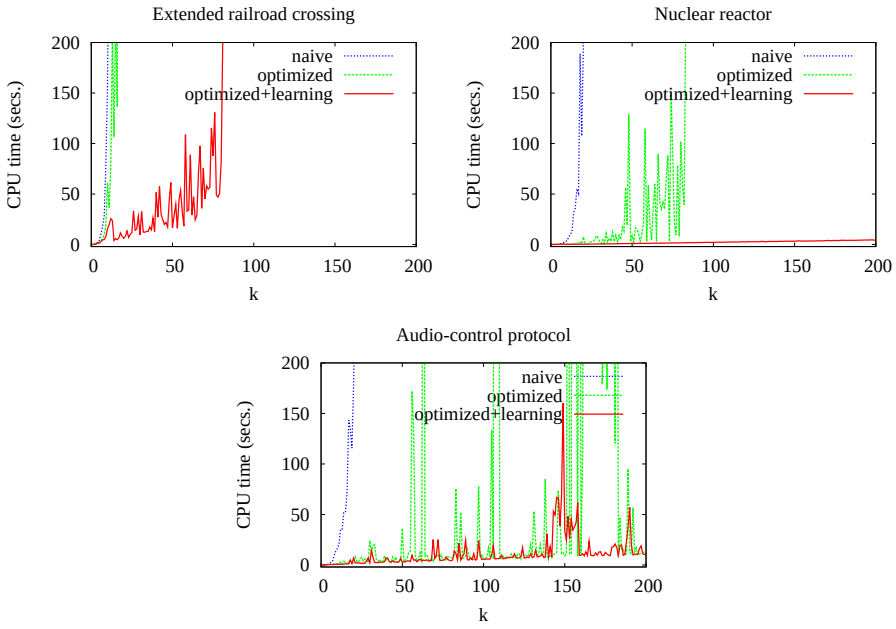


Fig. 2. Running times for the satisfiability checks for the naive encoding, the optimized encoding, and the optimized encoding with learning explanations.

2.2 Number of explanations

The tables 2-10 list for the different examples the numbers of explanations generated in each of the iterations 0 to 15 with the encodings (A), (B), and (C). Additionally, for (C) we list the mean size of the explanations (i.e., the number of (in)equations an explanation consists of), the number and mean size of the explanations after minimization and after removing subsumed explanations, and the CPU time (secs.) that was needed by ICS for the satisfiability checks for the minimization. The sign '$-$' means that the value could not be computed because one of the previous iterations exceeded our time limit of 200 secs. of CPU time.

	naive	**optimized**	**optimized+learning**				
k	# expl.	# expl.	# expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	2	1	1	1	3	3	0.1
1	3	1	1	1	7	4	0.3
2	6	1	2	2	6	4	0.5
3	10	1	0	0	0	0	0.0
4	15	1	0	0	0	0	0.0
5	16	1	0	0	0	0	0.0
6	27	1	0	0	0	0	0.0
7	63	1	0	0	0	0	0.0
8	78	1	0	0	0	0	0.0
9	67	1	0	0	0	0	0.0
10	220	1	0	0	0	0	0.0
11	111	1	0	0	0	0	0.0
12	78	1	0	0	0	0	0.0
13	106	1	0	0	0	0	0.0
14	296	1	0	0	0	0	0.0
15	100	1	0	0	0	0	0.0

Table 2. Number of explanations that are generated during the satisfiability checks for the thermostat example.

k	**naive** # expl.	**optimized** # expl.	**optimized+learning** # expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	2	2	2	2	3	3	0.2
1	6	2	3	2	4	3	0.3
2	12	2	2	2	4	3	0.4
3	22	2	5	5	3	3	1.0
4	31	2	2	1	6	5	0.3
5	49	2	3	3	5	4	0.9
6	75	2	0	0	0	0	0.0
7	91	2	0	0	0	0	0.0
8	120	2	0	0	0	0	0.0
9	150	2	0	0	0	0	0.0
10	185	2	0	0	0	0	0.0
11	225	2	0	0	0	0	0.0
12	172	2	0	0	0	0	0.0
13	293	2	0	0	0	0	0.0
14	326	2	0	0	0	0	0.0
15	378	2	0	0	0	0	0.0

Table 3. Number of explanations that are generated during the satisfiability checks for the water-level monitor example.

k	**naive** # expl.	**optimized** # expl.	**optimized+learning** # expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	1	2	2	2	3	3	0.3
1	16	4	4	4	5	4	1.1
2	56	8	2	2	3	2	0.3
3	156	8	0	0	0	0	0.0
4	332	18	19	8	7	4	2.4
5	575	15	0	0	0	0	0.0
6	1081	28	0	0	0	0	0.0
7	1815	43	25	14	9	5	5.2
8	3233	10	0	0	0	0	0.0
9	5871	27	0	0	0	0	0.0
10	11651	67	0	0	0	0	0.0
11	15379	37	0	0	0	0	0.0
12	–	30	0	0	0	0	0.0
13	–	99	0	0	0	0	0.0
14	–	88	0	0	0	0	0.0
15	–	49	0	0	0	0	0.0

Table 4. Number of explanations that are generated during the satisfiability checks for the bakery protocol for 2 processes.

k	**naive**	**optimized**	**optimized+learning**				
	# expl.	# expl.	# expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	1	2	2	2	4	4	0.4
1	14	4	4	3	6	4	1.0
2	43	5	2	1	9	2	0.4
3	106	9	0	0	0	0	0.0
4	215	10	0	0	0	0	0.0
5	321	19	0	0	0	0	0.0
6	572	29	6	4	16	6	4.5
7	850	33	0	0	0	0	0.0
8	1796	16	0	0	0	0	0.0
9	2434	22	0	0	0	0	0.0
10	3931	30	0	0	0	0	0.0
11	8138	26	0	0	0	0	0.0
12	–	76	0	0	0	0	0.0
13	–	88	0	0	0	0	0.0
14	–	55	0	0	0	0	0.0
15	–	103	0	0	0	0	0.0

Table 5. Number of explanations that are generated during the satisfiability checks for Fischer's protocol for 2 processes.

k	**naive**	**optimized**	**optimized+learning**				
	# expl.	# expl.	# expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	2	3	3	3	4	4	0.7
1	27	7	7	4	7	4	1.6
2	66	16	6	4	6	2	1.1
3	179	20	0	0	0	0	0.0
4	423	34	0	0	0	0	0.0
5	859	24	0	0	0	0	0.0
6	1563	78	13	5	23	8	6.0
7	2920	142	0	0	0	0	0.0
8	6217	128	12	1	28	10	2.0
9	10457	191	26	1	36	14	2.1
10	17222	163	6	1	24	10	1.3
11	–	610	0	0	0	0	0.0
12	–	469	0	0	0	0	0.0
13	–	1002	36	2	41	13	9.9
14	–	615	0	0	0	0	0.0
15	–	2387	0	0	0	0	0.0

Table 6. Number of explanations that are generated during the satisfiability checks for Fischer's protocol for 3 processes.

k	**naive** # expl.	**optimized** # expl.	**optimized+learning** # expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	1	1	1	1	3	2	0.1
1	6	1	1	1	15	8	0.7
2	16	2	2	1	14	5	0.4
3	31	3	1	1	25	18	1.4
4	63	3	4	3	9	6	1.7
5	95	4	0	0	0	0	0.0
6	179	12	0	0	0	0	0.0
7	294	16	0	0	0	0	0.0
8	433	9	11	7	11	3	2.5
9	651	40	27	6	19	8	5.3
10	1047	53	0	0	0	0	0.0
11	1703	12	0	0	0	0	0.0
12	2500	20	9	2	21	13	2.5
13	3518	80	0	0	0	0	0.0
14	4402	31	0	0	0	0	0.0
15	6462	109	0	0	0	0	0.0

Table 7. Number of explanations that are generated during the satisfiability checks for the railroad crossing example.

k	**naive** # expl.	**optimized** # expl.	**optimized+learning** # expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	2	0	0	0	0	0	0.0
1	19	6	6	1	18	5	0.7
2	54	21	23	8	27	5	8.4
3	128	66	21	12	7	3	2.7
4	259	91	50	19	21	3	10.2
5	646	74	32	9	20	6	4.7
6	1207	149	57	11	21	7	8.4
7	1529	301	62	14	28	7	14.0
8	1808	291	32	7	31	8	6.9
9	3768	584	26	7	36	12	14.0
10	7136	1526	148	36	28	3	20.8
11	9945	1096	72	21	53	14	73.1
12	–	1342	27	7	81	15	23.5
13	–	4741	20	10	54	18	32.3
14	–	–	0	0	0	0	0.0
15	–	–	0	0	0	0	0.0

Table 8. Number of explanations that are generated during the satisfiability checks for the extended railroad example.

k	**naive**	**optimized**	**optimized+learning**				
	# expl.	# expl.	# expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	1	1	1	1	9	6	0.4
1	3	0	0	0	0	0	0.0
2	13	2	2	2	19	17	2.1
3	18	0	0	0	0	0	0.0
4	31	4	4	2	23	16	2.4
5	51	0	0	0	0	0	0.0
6	79	4	31	2	36	11	2.7
7	129	4	0	0	0	0	0.0
8	185	8	21	7	36	8	11.3
9	260	13	0	0	0	0	0.0
10	393	12	0	0	0	0	0.0
11	640	7	0	0	0	0	0.0
12	807	14	0	0	0	0	0.0
13	1131	2	0	0	0	0	0.0
14	1423	17	0	0	0	0	0.0
15	1767	14	0	0	0	0	0.0

Table 9. Number of explanations that are generated during the satisfiability checks for the nuclear reactor example.

k	**naive**	**optimized**	**optimized+learning**				
	# expl.	# expl.	# expl.	# expl. after subsumption check	mean expl. size	mean expl. size after minimization	CPU time (secs.) for minimization
0	1	0	0	0	0	0	0.0
1	11	0	0	0	0	0	0.0
2	33	0	0	0	0	0	0.0
3	74	0	0	0	0	0	0.0
4	172	12	12	3	7	6	1.1
5	561	14	17	10	8	5	4.0
6	771	6	7	7	8	5	3.2
7	840	14	0	0	0	0	0.0
8	1778	16	0	0	0	0	0.0
9	2333	29	0	0	0	0	0.0
10	2513	14	0	0	0	0	0.0
11	3536	12	0	0	0	0	0.0
12	4377	37	0	0	0	0	0.0
13	3742	13	0	0	0	0	0.0
14	6062	37	0	0	0	0	0.0
15	4888	34	0	0	0	0	0.0

Table 10. Number of explanations that are generated during the satisfiability checks for the audio-control protocol.

3 Termination conditions

3.1 Formalization

The BMC approach can be extended not only to search for counter-example but also to verify state properties [5, 3]. The methods proposed in [5] can only be applied to verify finite state systems. An extension of these methods to infinite state systems was presented in [3]. The verification is based on the k-induction scheme, for some $k \geq 0$ by trying to show that a state property is invariant in the first k states of any execution. If no counterexamples of length k are detected but the k-induction fails, we increase k and repeat trying to show invariance of the state property. An outline of the verification algorithm is as follows:

1. Set k to -1.
2. Repeat
3. Increase k by 2.
4. Return counterexample if ψ_k is satisfiable.
5. Until the formula χ_k is unsatisfiable, where the formula χ_k is defined by $\chi_k = \pi'_k(s_0, \ldots, s_k, t_1, \ldots, t_k) \wedge \bigwedge_{0 \leq i < k-1} safe(s_i) \wedge \neg safe(s_k)$.

This algorithm is essentially the BMC algorithm extended by a termination condition, which corresponds to the induction's step case. Note that we increase k by 2 in each iteration, since we assume that flows and jumps alternate in runs. Furthermore, we are interested only in runs with a single bad state; since we allow flows of duration 0, such a bad state can be reached in one of the last two steps. Also note that if the formula χ_k is unsatisfiable we know that the property $safe(s)$ is invariant. We can use the lazy theorem proving algorithm or its variants to check unsatisfiability of χ_k.

Several other refinements have been proposed in [5] and [3] for strengthening the termination condition. For instance, we can use

$$\chi_k \wedge \bigwedge_{0 \leq i < j \leq k \text{ and } j \text{ even}} s_i \neq s_j \wedge \bigwedge_{0 \leq i < j \leq k \text{ and } j \text{ odd}} (t_j = 0 \vee s_i \neq s_j)$$

instead of χ_k in line 5 of the algorithm, since we do not need to consider runs containing loops; note that we do not compare states prior and after flows with duration 0. Similarly, we can require that none of the states after the first jump satisfy the initial condition, since in such cases there is a shorter computation leading to the same bad state. Moreover, we can use τ-transitions as introduced in the paper, such that k can be increased by an arbitrary even value larger than 2 in each iteration of the algorithm.

3.2 Experimental results for the termination conditions

Our optimizations also improve the running times for checking the termination condition and decrement of the number of explanations. The experimental results are summarized in Table 11. The number k is the number of iterations until the algorithm terminates. A '$-$' means that the BMC method does not termination within our limit of 200 secs. per satisfiability check. The running times that are shown in the table are the sums of the running times of all satisfiability checks until termination. Similarly, the number

	naive			optimized			optimized+learning		
	term.	time (secs.)	# expl.	term.	time (secs.)	# expl.	term.	time (secs.)	# expl.
Thermostat	$k = 1$	0.2	9	$k = 1$	0.2	4	$k = 1$	0.2	4
Water-level monitor	–	–	–	$k = 1$	0.2	12	$k = 1$	0.1	13
Bakery protocol (2 proc.)	$k = 7$	9.5	6022	$k = 7$	3.3	455	$k = 7$	2.4	108
Fischer's protocol (2 proc.)	–	–	–	$k = 5$	12.89	248	$k = 5$	4.01	91
Fischer's protocol (3 proc.)	–	–	–	$k = 9$	74.4	642	$k = 9$	60.6	403
Railroad crossing	–	–	–	$k = 4$	2.2	107	$k = 4$	2.5	51
Extended railroad crossing	–	–	–	–	–	–	–	–	–
Nuclear reactor	–	–	–	$k = 7$	2.4	128	$k = 7$	2.7	66
Audio-control protocol	–	–	–	–	–	–	–	–	–

Table 11. Experimental results for the termination condition.

of explanations is the sum of all generated explanations during all satisfiability checks until termination.

For most of the examples the naive method does not terminate. The reason is that the naive method allows successive flows such that there are arbitrary long computations satisfying the termination condition. Successive flows are excluded in the optimized versions. For many of our examples, the termination condition leads to termination of the BMC algorithm and thus to the verification of the state properties after a small number of iterations.

References

1. R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995. A preliminary version appeared in the proceedings of 11th. International Conference on Analysis and Optimization of Systems: Discrete Event Systems (LNCI 199).
2. D. Bosscher, I. Polak, and F. Vaandrager. Verification of an audio control protocol. In *Proceedings of Formal Techniques in Real Time and Fault Tolerant Systems Symposium*, 1994.
3. L. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In A. Voronkov, editor, *CADE'02*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 438–455. Springer-Verlag, 2002.
4. P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In *Proceedings of the Seventh Conference on Computer-Aided Verification*, pages 381–394, Liege, Belgium, 1995. Springer-Verlag. Lecture Notes in Computer Science 939.
5. M. Sheeran, S. Singh, and G. Stalmårck. Checking safety properties using induction and a SAT-solver. In W. A. J. Hunt and S. D. Johnson, editors, *FMCAD 2000*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer-Verlag, 2000.