VerICS 2008 - a Model Checker for Time Petri Nets and High-Level Languages*

M. Kacprzak[1], W. Nabiałek[2], A. Niewiadomski[2], W. Penczek[2,3], A. Półrola[4], M. Szreter[3], B. Woźna[5], and A. Zbrzezny[5]

[1] Bialystok University of Technology, FCS, Wiejska 45a, 15-351 Bialystok, Poland
[2] University of Podlasie, ICS, Sienkiewicza 51, 08-110 Siedlce, Poland
[3] Polish Academy of Sciences, ICS, Ordona 21, 01-237 Warsaw, Poland
[4] University of Lodz, FMCS, Banacha 22, 90-238 Lodz, Poland
[5] Jan Dlugosz University, IMCS, Armii Krajowej 13/15, 42-200 Czestochowa, Poland
verics@ipipan.waw.pl

Abstract. The paper presents the current stage of the development of VerICS - a model checker for high-level languages, as well as real-time and multi-agent systems. Depending on the type of a system considered, it enables to test various classes of properties - from reachability to temporal, epistemic and deontic formulas. The model checking methods used to this aim include both SAT-based and enumerative ones. In the paper we focus on new features of the verifier: model checking of time Petri nets (TPNs) as well as of high-level languages: UML, Java, and Promela.

1 Introduction

The paper presents the current stage of the development of VerICS, a model checker for high-level languages, as well as real-time and multi-agent systems. Depending on the type of a system considered, the verifier enables to test various classes of properties - from reachability of a state satisfying certain conditions to more complicated features expressed by formulas of (timed) temporal, epistemic, or deontic logics. The model checking methods implemented include both SAT-based and enumerative ones (where by the latter we mean these consisting in generating abstract models for systems). Our previous work [18] presenting VerICS dealt mainly with verification of real-time systems (RTS) and multi-agent systems (MAS). In this paper we focus on VerICS' new features, i.e., SAT-based model checking for time Petri nets and systems specified in UML [28], Java [10], and Promela [12]. Next, we discuss some related verification approaches and tools.

The well-known tools for time Petri nets include the systems discussed below. Tina [2] is a toolbox for analysis of (time) Petri nets, which constructs state class graphs (abstract models) and exploits them for LTL, CTL, or reachability verification. Romeo [34] is a tool for time Petri nets analysis, which provides

* Partly supported by the Ministry of Education and Science under the grant No. N N516 370436 and N N206 258035.

several methods for translating TPNs to timed automata and computation of state class graphs. CPN Tools [4] is a software package for modelling and analysis of both timed and untimed coloured Petri nets, enabling their simulation, generating occurrence (reachability) graph, and analysis by place invariants.

There have been a lot of attempts to verify UML state machines - all of them based on the same idea: translate an UML specification to the input language of some model checker, and then perform verification using the model checker. Some of the approaches [15, 20] translate UML to the language Promela and then make use of the Spin [12] model checker. Other [7, 19] exploit timed automata as an intermediate formalism and exploit UPPAAL [1] for verification. The third group of tools (e.g., [8]) apply the symbolic model checker NuSMV [5] via translating UML to its input language. One of the modules of VeriCS follows this idea. An UML subset is translated to the Intermediate Language (IL) of VeriCS. However, we have developed also a symbolic model checker that deals directly with UML specifications by avoiding any intermediate translations. The method is implemented as the module BMC4UML.

Another situation prevails in the field of Promela verification. There exists only a few model checkers for Promela and its time extensions. SPIN [12] is a model checker for Promela specifications. Correctness properties can be specified as system or process invariants (using assertions), linear temporal logic formulas (LTL), formal Büchi automata, or more broadly as general omega-regular properties in the syntax of never claims. As the first attempt to verification of timed systems, Real Time Promela was developed [37]. This extension of Promela introduces explicit definitions of clocks that can be used in expressions and reset. The other approach, Discrete Time Promela [3], instead of clocks, introduces a new special type - count down timer. Timers can be set to some values and tested if they have expired. We offer a translator of Timed Promela [23] (a large subset of Promela extended by time annotations) to timed automata with discrete data (TADD) [42] as a VeriCS module.

Model checking of Java programs has become increasingly popular during the last decade. However, to the best of our knowledge, there are only two existing model checkers that can verify Java codes: JavaPathFinder (JPF) [11, 29] and Bandera [6]. JPF is a system to verify executable Java bytecode programs and it is one of the backend model-checkers supported by Bandera. Thus, both tools operate on the Java bytecode. On the contrary, we analyse Java programs themselves and translate them to a network of TADDs.

The rest of the paper is organised as follows. In Section 2 we briefly present a theoretical background of the SAT-based verification methods implemented in our tool (i.e., bounded and unbounded model checking). The next section contains a description of the verification system. In Section 4 we provide some experimental results obtained for several typical benchmarks used to test efficiency of model checkers. Finally, Section 5 contains a summary and some concluding remarks.

2 Theoretical Background

A network of communicating (timed) automata is the basic formalism of VerICS for modelling a system to be verified. Timed automata are used to specify RTS (possibly with clock differences expressing constraints on their behaviour), whereas timed or untimed automata are applied to model MAS (possibly extended in a way to handle certain features of interest, like deontic automata in [17]). The current version of VerICS makes extensive use also of timed automata extended by integer variables, called timed automata with discrete data (TADD) [42]. A set (*network*) of timed automata with discrete data consists of n TADDs which run in parallel. The automata communicate with each other via shared (i.e., common for some automata) variables, and perform transitions with shared labels synchronously. We assume the scheme of *multi-synchronisation*, which requires the transitions with a shared label to be executed synchronously by each automaton that contains this label in its set of labels. To obtain a clear semantics of variable updating it is necessary to fix the order of instructions in the case of synchronous execution of transitions. Thus, the transition whose instruction is to be taken first (called the *output transition*) is marked with the symbol !, whereas these which are to be taken later (the *input* ones) are marked with the symbol ?.

The tuples of *local states* of the automata in a network $\mathfrak{A}$ define the *global states* of the system considered. The set of all the possible *runs* (i.e., infinite evolutions from a given initial state) of a system modelled by $\mathfrak{A}$ gives us a *computation tree* which, after labelling the states with propositions from a given set PV which are true at these states (i.e., changing the tree into a *model*), is used to interpret the formulas of timed or untimed temporal logics. These are variants of Computation Tree Logic (CTL) or its timed version (TCTL) expressing properties to be checked. In the case of modelling a MAS we augment the model with *epistemic* or *deontic accessibility relations*. The resulting structure enables us to interpret formulas involving temporal operators, epistemic operators - to reason about *knowledge* of agents [9], and deontic operators - to reason about *correctness* of their behaviour.

The current version of VerICS accepts also an input in the form of *distributed time Petri nets* [32], which are another formalism used to specify RTS. A distributed time Petri net consists of a set of 1-safe sequential[6] TPNs (called *processes*), of pairwise disjoint sets of places, and communicating via joint transitions. Moreover, the processes are required to be *state machines*, which means that each transition has exactly one input place and exactly one output place in each process it belongs to. A state of the system considered is given by a marking of the net and by the values of the clocks associated with the processes[7].

SAT-based verification methods represent the models and properties of systems in the form of boolean formulas in order to reduce the state explosion.

[6] A net is sequential if none of its reachable markings concurrently enables two transitions.

[7] A detailed description of the nets, as well as their semantics, can be found in [33].

These for MAS involve bounded (BMC) and unbounded model checking (UMC). Currently, VerICS implements UMC for $\text{CTL}_\text{p}\text{K}$ (Computation Tree Logic with knowledge and past operators) [16], and BMC for ECTLKD (the existential fragment of CTL extended with knowledge and deontic operators) [17, 30, 38, 39] as well as TECTLK (the existential fragment of timed CTL extended with knowledge operators) [21].

Considering verification of RTS, the current version of VerICS offers BMC for proving (un)reachability [40] (also for timed automata with clock differences [41]), and UMC for proving CTL properties for slightly restricted timed automata [36].

Below we present some intuition behind BMC and UMC.

2.1 Bounded Model Checking

Bounded Model Checking (BMC) is a symbolic method aimed at verification of temporal properties of distributed (timed) systems. It is based on the observation that some properties of a system can be checked over a part of its model only. In the simplest case of reachability analysis, this approach consists in an iterative encoding of a finite symbolic path (computation) as a propositional formula.

In order to apply Bounded Model Checking to testing reachability of a state satisfying a certain (usually undesired) property, we unfold the transition relation of a given automaton/TPN up to some depth k, and encode this unfolding as a propositional formula. Then, the property to be tested is encoded as a propositional formula as well, and satisfiability of the conjunction of these two formulas is checked using a SAT-solver. If the conjunction is satisfiable, one can conclude that a counterexample (a path to an undesirable state) was found. Otherwise, the value of k is incremented. The above process can be terminated when the value of k is equal to the diameter of the system, i.e., to the maximal length of a shortest path between its two arbitrary states.

2.2 Unbounded Model Checking

Unlike BMC, UMC is capable of handling the whole language of the logic. Like any SAT-based method, UMC consists in translating the model checking problem of a $\text{CTL}_\text{p}\text{K}$ formula into the problem of satisfiability of a propositional formula. UMC exploits the characterisation of the basic modalities in terms of Quantified Boolean Formulas (QBF), and the algorithms that translate QBF and fixed point equations over QBF into propositional formulas. In order to adapt UMC for checking $\text{CTL}_\text{p}\text{K}$, we use three algorithms. The first one, implemented by the procedure "forall" (based on the Davis-Putnam-Logemann-Loveland approach) eliminates the universal quantifier from a QBF formula representing a $\text{CTL}_\text{p}\text{K}$ formula, and returns the result in conjunctive normal form. The remaining algorithms, implemented by the procedures "gfp" and "lfp" calculate the greatest and the least fixed points for the modal formulas in use here. Ultimately, the technique allows for a $\text{CTL}_\text{p}\text{K}$ formula to be translated into a propositional

formula in CNF, which characterises all the states of the model, where the formula holds. Next we apply a SAT-solver to check satisfiability of the obtained propositional formula.

3 Implementation

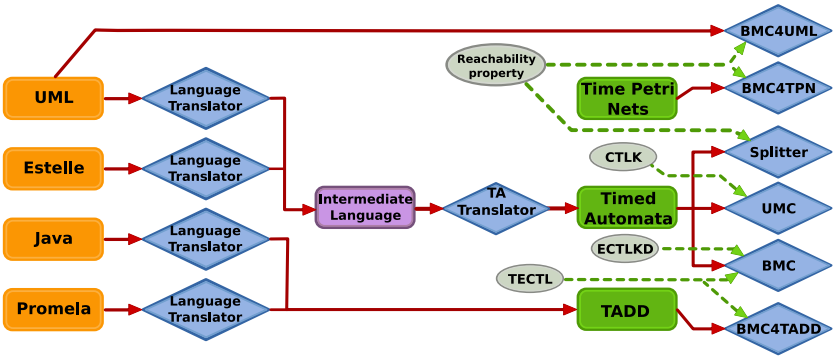


Fig. 1. Architecture of VerICS

The architecture of VerICS is shown in Fig. 1. The new components are:

- **UML to Intermediate Language (IL) translator**, which translates UML specification consisting of class, object and statemachine diagrams to corresponding IL program.
- **Java to TADD translator**, which translates a concurrent multi-threaded Java program to a network of TADDs.
- **Promela to TADD translator**, which generates a network of TADDs corresponding to the given Promela specification, possibly extended by time annotations.
- **BMC4UML module** - a Bounded Model Checker for UML, which applies SAT-based BMC algorithm directly on UML specification, avoiding intermediate translations.
- **BMC4TADD module** - a Bounded Model Checker for a network of TADDs.
- **BMC4TPN module** - a Bounded Model Checker for time Petri nets.

The remaining components, presented in [18], are listed below:

- **Estelle to IL translator**, which enables to handle specifications written in a subset of Estelle [13] (the standardised language for specifying communicating protocols and distributed systems);
- **IL to timed automata translator**, which, given an IL specification, generates the corresponding network of timed automata or the global timed automaton;

– **BMC module**, which implements BMC-based verification for the classes of properties shown in the figure. The SAT-solver used is MiniSat [22] or RSat [35]; the system can be configured to work with other solvers;
– **UMC module**, which provides preliminary implementations of UMC verification methods for properties described above. The module is integrated with a modified version of the SAT-solver ZChaff [44];
– **Splitter module**, which performs reachability verification on abstract models generated for timed automata.

VeriCS is implemented in C++ and Java; its internal functionalities are available via a interface written in Java. The demo of current distribution can be accessible from `http://verics.ipipan.waw.pl`. A more detailed description of the tool, and in particular the new high-level languages translators, are presented in the following subsections.

3.1 Model checking of UML

At the moment we deal with model checking of UML in two ways: either translating an input specification to IL, and then use the standard verification path (translation to a network of TA and application of BMC, UMC or Splitting), or use the BMC module, which performs model checking directly, without intermediate translations.

Both the methods require as an input a specification in the XMI format, making use of a similar subset of UML. An input specification should consist of: one class diagram, one object diagram, and one state machine diagram per each class of the class diagram. The class and object diagrams define the static structure of the system, while the state machines determine its behaviour.

Translation of UML to Intermediate Language. In this section we give the main ideas behind our translation from UML to IL. The details can be found in [24]. Objects are mapped onto processes of IL and the number of UML objects corresponds to the number of IL processes. The attributes of objects are translated into process variables. We allow boolean, integer, and object types. The methods are translated into arrays of IL buffers, whereas each method call is realized by placing a special element - *call marker* - in the corresponding buffer, possibly followed by the method's parameters. Each of UML simple- and pseudo-states is mapped onto a state of an IL process. Entry and exit activities are merged with actions of incoming and outgoing transitions. The transitions in State Diagrams are translated directly into transitions of IL processes. A triggered event, a guard, and a sequence of actions can be associated with the transition. The time events in UML are translated into time constraints of IL transitions, using *delay* construction. The latter allows to specify the amount of time that may elapse before certain actions take place. The guards in UML are formed using attributes of objects and parameters of the actions called. These expressions are directly transformed into IL guards, using the variables that correspond to UML attributes and parameters.

BMC4UML - a Bounded Model Checking for UML. In order to perform symbolic model checking directly on an UML specification, an operational semantics of the considered UML subset is defined [25] in terms of a labelled transition system. Then, the transition system is symbolically encoded and the prototype implementation is developed [26].

In general the main ideas of BMC are applied to the transition system representing the executions of UML system. However, very complex elements of the UML state machines semantics (concerning e.g. hierarchy of states and regions, concurrent regions, priorities of transitions and properly handling of completion events and RTC-steps) require numerous non-trivial solutions at the level of symbolic encoding and implementation [27].

3.2 Translation of Java to TADDs

Below we sketch the main ideas behind a translation of a concurrent multi-threaded Java program to a network of TADDs. Each state of TADD is an abstraction of a state of the Java program, and each transition represents the execution of the code transforming this abstract state. The subset of Java that can be translated to TADDs contains: definitions of integer variables, standard programming language constructs like assignments, expressions with most operators, conditional statements and loops (*for*, *while*, *do while*), instructions *break* and *continue* without labels, definitions of classes, objects, constructors and methods, static and non-static methods and synchronisation of methods and blocks. There are recognised standard thread creation constructs as well as special methods: *Object.wait()*, *Object.notify()*, *Thread.sleep(int)*, and *Random.nextInt(int)*.

A theoretical method of constructing a network of TADDs that models a Java program is shown in [43]. To implement the translation we first translate a Java code to an internal assembler. Then, the resulting assembler is translated to timed automata with discrete data.

3.3 Translation of Promela to TADDs

The translation is performed in three stages. The first one consits in a translation of control flow of each Promela process into an automaton structure. The next one concerns representation of Promela data structures and operations on them. Finally, a set of TADDs corresponding to all the instances of the Promela processes is defined. The translation is inductive. The procedure starts with a block (a sequence of statements) representing the behaviour of a whole process and operates in a top-down fashion up to basic statements.

Each Promela process is translated to a TADD, and if it is necessary additional TADDs for *init* and *never-claim* processes are created. The local variables are mapped into global ones, while arrays and channels are translated onto set of global variables. Each Promela statement is represented as a transition, or - in the case of more complex constructions (e.g. loops or selections) - as a set of transitions. The operations on arrays and channels need also more than one

transition. Their number depends on the size of an array or the capacity of a channel.

Our translation covers most of Promela constructs. Moreover it is extended by time expressions, in order to specify real-time systems. The details can be found in [23].

3.4 Bounded Model Checking for TPNs

In order to benefit from the concurrent structure of a system, we consider distributed nets only [31], and assume that all their processes are *state machines*. It is important to mention that a large class of distributed nets can be decomposed to satisfy the above requirement [14]. The interpretation of such a system is a collection of sequential, non-deterministic processes with communication capabilities (via joint transitions). An example of a distributed TPN (Fischer's mutual exclusion protocol) is shown in Fig. 2. The net consists of three communicating processes with the sets of places $P_i = \{idle_i, trying_i, enter_i, critical_i\}$ for $i = 1, 2$, and $P_3 = \{place0, place1, place2\}$. All the transitions of the process N_1 and all the transitions of the process N_2 are joint with the process N_3.

The current implementation supports *reachability* checking, i.e., verification whether the system (net) can ever be in a state satisfying certain properties. The details of the method can be found in [33]. This solution can be also adapted to verification of other classes of properties for which BMC methods exist and is still to be implemented.

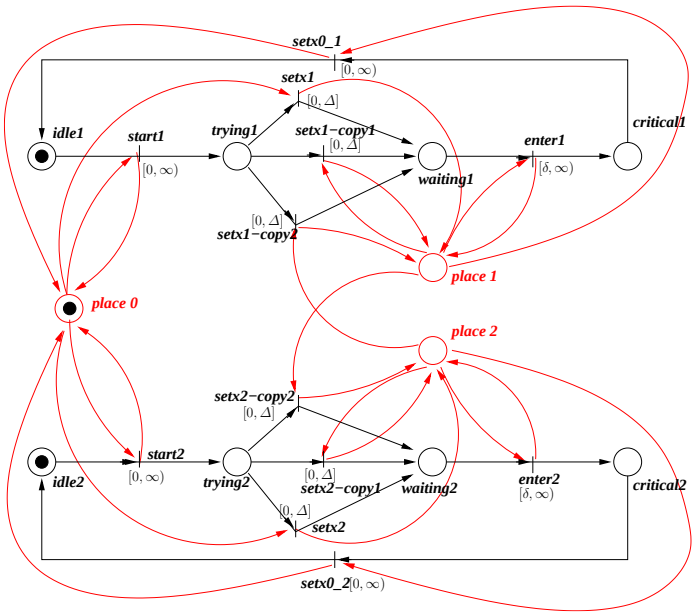
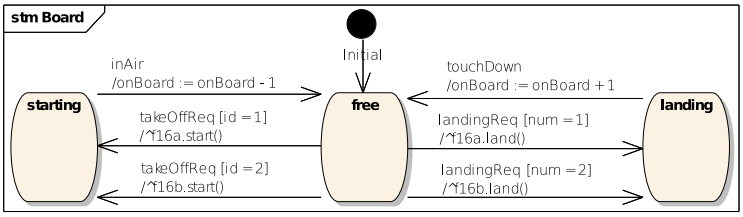


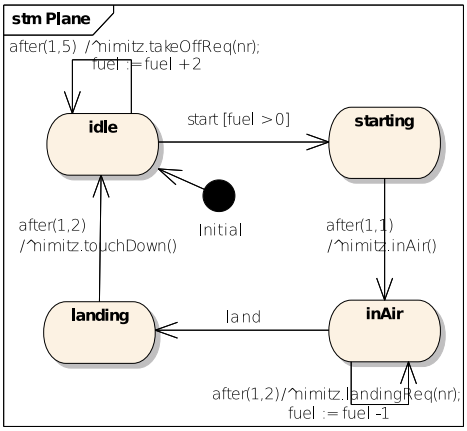
Fig. 2. A net for Fischer's mutual exclusion protocol for $n = 2$

4 Experimental Results

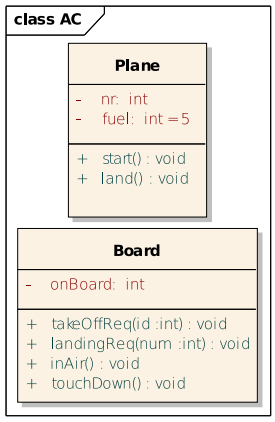
One of the important elements taken into account while rating a model checker is its efficiency. In this section we present some well known benchmarks: Alternating Bit Protocol (ABP) specified in UML and Java and Fischer's mutual exclusion protocol specified in TPNs and Promela, as well as the Aircraft Carrier (AC) UML specification.



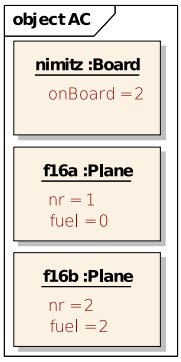
(a) State machine of class Board



(b) State machine of class Plane



(c) Class diagram



(d) Object diagram

Fig. 3. Specification of Aircraft Carrier system

Table 1(a) presents some experimental results of testing reachability of a deadlock state in ABP version written in Java (slightly modified in order to introduce deadlock) and Table 1(b) presents the results of the verification of the negation of the property: *"after the message and an acknowledgement have been received, both (Sender's and Receiver's) internal bits are equal"* for ABP specification in UML.

Table 2 presents the results of verification of the Aircraft Carrier (AC) specification (Fig. 3). AC consists of a ship and a number of aircrafts taking off and landing continuously, after issuing a request being accepted by the controller.

Table 1. Experimental results of verification of ABP

(a) Java via translation to TADDs

k	Clauses	BMC [s]	RSAT [s]	SAT
0	559	0.0	0.0	NO
12	57346	0.5	0.1	NO
24	121540	1.2	0.7	NO
50	**282621**	**2.8**	**82.4**	**YES**
	In total:	**33.2**	**122.9**	

(b) UML via translation to IL and TA

k	Clauses	BMC [s]	zChaff [s]	SAT
1	44098	0.27	0.01	NO
5	214598	1.44	0.40	NO
10	427723	2.84	7.22	NO
13	**555598**	**3.73**	**5.74**	**YES**
	In total:	**25.89**	**35.22**	

The events of answering these requests may be marked as deferred. Each aircraft refills fuel while on board and burns fuel while airborne. We check the property whether an aircraft can run out of fuel during its flight.

Moreover, we have introduced some elements of parametric reachability checking. Using our approach, we are able to verify not only that a property is reachable, but also to find a minimal (integer) time c, *when* this is the case (Table 2, the last column). More examples and a broader comparison with other model checkers for UML can be found in [26, 27].

Table 2. Results of verification of AC system (with/without deferred events)

N	k	Hugo+Uppaal [s]	BMC4UML [s]	Parametric [s], c = 4
3	19	1.32 / 1.25	67.59 / 51.26	31.34 / 22.64
4	20	13.15 / 11.41	101.58 / 81.28	45.44 / 42.38
5	21	147.43 / 95.67	155.63 / 132.34	60.49 / 37.01
6	22	Out of mem	257.08 / 216.42	52.23 / 75.08
7	23	- / -	686.06 / 421.85	101.86 / 199.09

We have also tested the systems modelling the standard Fischer's mutual exclusion protocol (Mutex). In general, the system consists of n processes which run in parallel. *Mutual exclusion* means that no two processes are in their critical sections at the same time. The preservation of this property depends on the relative values of the time-delay constants δ and Δ. In particular, the following holds: "*Fischer's protocol ensures mutual exclusion iff* $\Delta < \delta$".

A TPN model for Mutex consists of n time Petri nets, each one modelling a process, plus one additional net used to coordinate their access to the critical sections. The resulting distributed TPN, for the case of $n = 2$, is shown in Figure 2.

We have checked that if $\Delta \geq \delta$, then the mutual exclusion is violated. We considered the case with $\Delta = 2$ and $\delta = 1$. It turned out that the conjunction of the propositional formulae encoding the k-path and the negation of the mutual exclusion property is unsatisfiable for every $k < 12$. The witness was found for $k = 12$. We were able to test 40 processes. The results are shown in Table 3.

Table 3. Verification of time Petri Nets - Fischer's protocol (40 processes)

		tpnBMC				RSat		
k	**n**	**variables**	**clauses**	**sec**	**MB**	**sec**	**MB**	**sat**
0	-	1937	5302	0.2	3.5	0.0	1.7	NO
2	-	36448	107684	1.4	7.9	0.4	9.5	NO
4	-	74338	220335	2.9	12.8	3.3	21.5	NO
6	-	112227	332884	4.2	17.6	14.3	37.3	NO
8	-	156051	463062	6.1	23.3	257.9	218.6	NO
10	-	197566	586144	7.8	28.5	2603.8	1153.2	NO
12	-	240317	712744	9.7	34.0	87.4	140.8	YES
				32.4	34.0	2967.1	1153.2	

Table 4 presents experimental results for a timed Promela version of Fischer's mutual exclusion protocol. The time parameters of the protocol have been set in this way that the protocol is not correct. We have looked for the situation when any pair of processes is in their critical sections at the same time. The tests are done with latest distributions of RTSpin, DTSpin, and Verics.

Table 4. Experimental results of verification of timed version of Fisher's Mutual Exclusion protocol specified in Promela.

# proc.	Spin				Verics				
	RTSpin		DTSpin			BMC		SAT	
	mem	cpu	mem	cpu	depth	mem	cpu	mem	cpu
8	34.21	5.4	57.86	0.08	12	3.4	0.34	5.83	0.26
80	—	—	146.03	2.49	12	12.0	12.57	93.65	85.67
100	—	—	228.06	4.43	12	16.4	20.21	256.38	339.32
130	—	—	529.19	30.76	12	24.4	32.56	103.91	48.97
135	—	—	—	—	12	25.8	34.91	459.14	1139.66

5 Final Remarks

As it can be seen from the above results, Verics in many cases is able to handle relatively large examples taken from the standard scalable benchmarks. This allows to expect the same also in the case of "real world" systems. However, it should be said that the size of the system, which can be verified using the BMC method, depends on the formula tested: the more shallow the counterexample and the less paths needed to test the formula, the bigger system can be verified. On the other hand, a strong limitation for both the SAT-based methods we use are the capabilities of the SAT-solvers available, which, in many cases, are not

able to handle the set of clauses generated by the method, or to solve it in a reasonable time. This, however, proves also that the development of solvers can result in an improvement of efficiency of our tool.

References

1. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi, and C. Weise. New generation of UPPAAL. In *Proc. of the Int. Workshop on Software Tools for Technology Transfer*, 1998.
2. B. Berthomieu, P-O. Ribet, and F. Vernadat. The tool TINA - construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14), 2004.
3. D. Bosnacki and D. Dams. Discrete-time Promela and SPIN. In *Proc. of the 5th Int. Conf. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'98)*, volume 1486 of *LNCS*, pages 307–310. Springer-Verlag, 1998.
4. S. Christensen, J. Jørgensen, and L. Kristensen. Design/CPN - a computer tool for coloured Petri nets. In *Proc. of the 3rd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, volume 1217 of *LNCS*, pages 209–223. Springer-Verlag, 1997.
5. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV2: An open-source tool for symbolic model checking. In *Proc. of the 14th Int. Conf. on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 359–364. Springer-Verlag, 2002.
6. J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasareanu, Robby, and H. Zheng. Bandera: Extracting finite-state models from Java source code. In *Proc. of the 22nd Int. Conf. on Software Engineering (ICSE 2000)*, pages 439–448. ACM, 2000.
7. K. Diethers, U. Goltz, and M. Huhn. Model checking UML statecharts with time. In *Proc. of the Workshop on Critical Systems Development with UML (CSDUML'02)*, pages 35–52. Technische Universität München, 2002.
8. J. Dubrovin and T. Junttila. Symbolic model checking of hierarchical UML state machines. Technical Report HUT-TCS-B23, Helsinki Institute of Technology, Espoo, Finland, 2007.
9. R. Fagin, J. Y. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
10. J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification. Third Edition*. Addison-Wesley, 2005.
11. K. Havelund and T. Pressburger. Model checking Java programs using Java PathFinder. *Int. Journal on Software Tools for Technology Transfer*, 2(4):366–381, 1998.
12. G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Eng.*, 23(5):279–295, 1997.
13. ISO/IEC 9074(E), Estelle - a formal description technique based on an extended state-transition model. International Standards Organization, 1997.
14. R. Janicki. Nets, sequential components and concurrency relations. *Theoretical Computer Science*, 29:87–121, 1984.
15. T. Jussila, J. Dubrovin, T. Junttila, T. Latvala, and I. Porres. Model checking dynamic and hierarchical UML state machines. In *Proc. of the 3rd Int. Workshop on Model Design and Validation (MoDeVa 2006)*, pages 94–110. CEA, 2006.

16. M. Kacprzak, A. Lomuscio, T. Łasica, W. Penczek, and M. Szreter. Verifying multiagent systems via unbounded model checking. In *Proc. of the 3rd NASA Workshop on Formal Approaches to Agent-Based Systems (FAABS III)*, volume 3228 of *LNCS*, pages 189–212. Springer-Verlag, 2005.
17. M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundamenta Informaticae*, 72(1-2):215–234, 2006.
18. M. Kacprzak, W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1-4):313–328, 2008.
19. A. Knapp, S. Merz, and C. Rauh. Model checking - timed UML state machines and collaborations. In *Proc. of the 7th Int. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'02)*, volume 2469 of *LNCS*, pages 395–416. Springer-Verlag, 2002.
20. J. Lilius and I. Paltor. vUML: A tool for verifying UML models. In *Proc. of the 14th IEEE Int. Conf. on Automated Software Engineering (ASE'99)*, pages 255–258. IEEE Computer Society, 1999.
21. A. Lomuscio, B. Woźna, and A. Zbrzezny. Bounded model checking real-time multi-agent systems with clock differences: Theory and implementation. In *Proc. of the 4th Int. Workshop on Model Checking and Artificial Intelligence (MoChArt'06)*, pages 62–78. ECCAI, 2006.
22. MiniSat. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat, 2006.
23. W. Nabiałek, A. Janowska, and P. Janowski. Translation of timed Promela to timed automata with discrete data. *Fundamenta Informaticae*, 85(1-4):409–424, 2008.
24. A. Niewiadomski, W. Penczek, S. Lasota, and J. Kowalski. Weryfikacja UML z wykorzystaniem systemu VerICS. In *Mat. XII Konf. Systemy Czasu Rzeczywistego (SCR'06)*, pages 79–91. Wyd. Komunikacji i Łączności, 2006. In Polish.
25. A. Niewiadomski, W. Penczek, and M. Szreter. Semantyka operacyjna wybranych diagramów UML. Technical Report 1009, ICS PAS, Ordona 21, 01-237 Warsaw, March 2008. In Polish.
26. A. Niewiadomski, W. Penczek, and M. Szreter. Towards bounded model checking of UML. In *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'08)*, volume 225(3) of *Informatik-Berichte*, pages 386–397. Humboldt University, 2008.
27. A. Niewiadomski, W. Penczek, and M. Szreter. Towards checking parametric reachability for UML state machines. In *Proc. of the 7th Int. Ershov Memorial Conf. 'Perspective of System Informatics' (PSI'09)*, 2009. To appear.
28. OMG. Unified Modeling Language. http://www.omg.org/spec/UML/2.1.2, 2007.
29. C. Pasareanu and W. Visser. Verification of Java progtams using symbolic execution and invariant generation. In *Proc. of the 11th Int. SPIN Workshop (SPIN'04)*, volume 2989 of *LNCS*, pages 164–181. Springer-Verlag, 2004.
30. W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proc. of the 2nd Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, pages 209–216. ACM, 2003.
31. W. Penczek and A. Półrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*, volume 20 of *Studies in Computational Intelligence*. Springer-Verlag, 2006.

32. W. Penczek, A. Półrola, B. Woźna, and A. Zbrzezny. Bounded model checking for reachability testing in time Petri nets. In *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'04)*, volume 170(1) of *Informatik-Berichte*, pages 124–135. Humboldt University, 2004.
33. W. Penczek, A. Półrola, and A. Zbrzezny. SAT-based (parametric) reachability for distributed time Petri nets. In *Proc. of the Int. Workshop on Petri Nets and Software Engineering (PNSE'09)*, June 2009. To appear.
34. Romeo: A tool for time Petri net analysis. http://www.irccyn.ec-nantes.fr/irccyn/d/en/equipes/TempsReel/logs, 2000.
35. RSat. http://reasoning.cs.ucla.edu/rsat, 2006.
36. M. Szreter. *SAT-Based Model Checking of Distributed Systems*. PhD thesis, ICS PAS, January 2007.
37. S. Tripakis and C. Courcoubetis. Extending Promela and SPIN to real-time. In *Proc. of the 2nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 329–348. Springer-Verlag, 1996.
38. B. Woźna, A. Lomuscio, and W. Penczek. Bounded model checking for deontic interpreted systems. In *Proc. of the 2nd Int. Workshop on Logic and Communication in Multi-Agent Systems (LCMAS'04)*, volume 126 of *ENTCS*, pages 93–114. Elsevier, 2005.
39. B. Woźna, A. Lomuscio, and W. Penczek. Bounded model checking for knowledge and real time. In *Proc. of the 4th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'05)*, pages 165–172. ACM, 2005.
40. A. Zbrzezny. Improvements in SAT-based reachability analysis for timed automata. *Fundamenta Informaticae*, 60(1-4):417–434, 2004.
41. A. Zbrzezny. SAT-based reachability checking for timed automata with diagonal constraints. *Fundamenta Informaticae*, 67(1-3):303–322, 2005.
42. A. Zbrzezny and A. Półrola. Sat-based reachability checking for timed automata with discrete data. *Fundam. Inform.*, 79(3-4):579–593, 2007.
43. A. Zbrzezny and B. Woźna. Towards verification of Java programs in VerICS. *Fundamenta Informaticae*, 85(1-4):533–548, 2008.
44. L. Zhang. Zchaff. http://www.ee.princeton.edu/∼chaff/zchaff.php, 2001.