Parametric Model Checking with VerICS*

(Tool Paper)

Michał Knapik[1], Artur Niewiadomski[2], Wojciech Penczek[1,2], Agata Półrola[3], Maciej Szreter[1], and Andrzej Zbrzezny[4]

[1] Institute of Computer Science, PAS, Ordona 21, 01-237 Warszawa, Poland
{Michal.Knapik,penczek,mszreter}@ipipan.waw.pl
[2] Siedlce University, ICS, 3 Maja 54, 08-110 Siedlce, Poland
artur@ii.uph.edu.pl
[3] University of Łódź, FMCS, Banacha 22, 90-238 Łódź, Poland
polrola@math.uni.lodz.pl
[4] Jan Długosz University, IMCS, Armii Krajowej 13/15, 42-200 Częstochowa, Poland
a.zbrzezny@ajd.czest.pl

Abstract. The paper presents the verification system VerICS, extended with the three new modules aimed at parametric verification of Elementary Net Systems, Distributed Time Petri Nets, and a subset of UML. All the modules exploit Bounded Model Checking for verifying parametric reachability and the properties specified in the logic PRTECTL – the parametric extension of the existential fragment of CTL.

1 Introduction

VerICS is a model checker for high-level languages as well as real-time and multi-agent systems. Depending on the type of a considered system, the verifier enables to test various classes of properties - from reachability of a state satisfying certain conditions to more complicated features expressed with formulas of (timed) temporal, epistemic, or deontic logics. The implemented model checking methods include SAT-based ones as well as these based on generating abstract models for systems.

The architecture of VerICS is depicted in Fig. 1. At its right-hand side there are visualized the model checking methods offered for real-time systems: bounded model checking (BMC) for proving reachability in Time Petri Nets (TPN) and in Timed Automata (TA) (including TADD - TA with Discrete Data) as well as for testing TECTL (Timed Existential CTL) formulas for TA, unbounded model checking (UMC) for proving CTL properties for slightly restricted TA, and splitting for testing reachability for TA. The modules implementing the above methods are described in [8, 17] and [18]. In the boxes with rounded corners the input formalisms are depicted: both the low-level (TPN, TA) and high-level languages (Java, Promela, UML, Estelle). The languages for expressing properties

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

to be verified are depicted in ovals. Considering multi-agent systems, VerICS implements UMC for CTLpK (Computation Tree Logic with knowledge and past operators) and BMC for ECTLKD (the existential fragment of CTL extended with knowledge and deontic operators) as well as TECTLK (the existential fragment of timed CTL extended with knowledge operators). All these modules are covered in [17]. Moreover, our verifier offers also SAT-based model checking for high-level languages like UML [27], Java [13], and Promela [14]. The details can be found in [18].

In this paper we present the three new modules of VerICS, aimed at parametric verification. These are BMC4EPN, BMC4TPN, and BMC4UML displayed in the VerICS' architecture diagram in Fig. 1 (the right upper corner). The modules allow for checking properties, expressed in the PRTECTL logic [12, 19], of systems modelled as Elementary Net Systems, Distributed Time Petri Nets, and in a subset of UML, respectively.

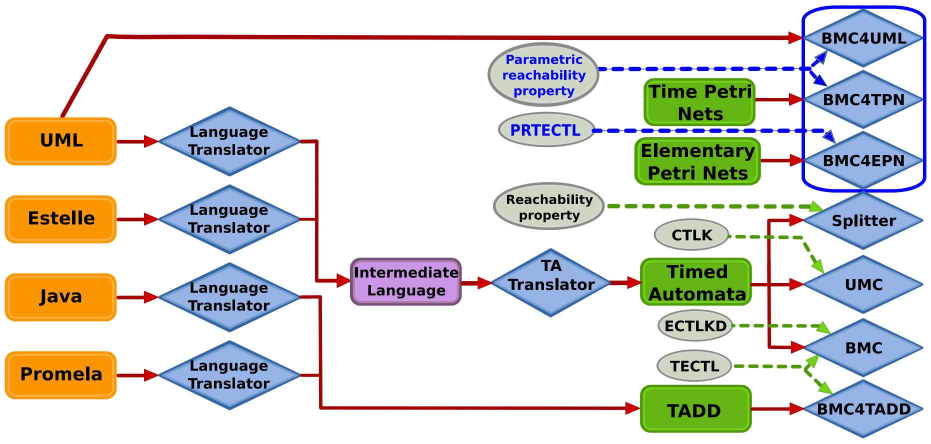


Fig. 1. Architecture of VerICS

Below, we give a short overview of some well-known tools for Petri Nets. Tina [4] 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 [22] is a tool for analysis of Time Petri Nets. It provides several methods for translating TPN to TA and for computing state class graphs. CPN Tools [6] is a software package for modelling and analysis of both timed and untimed Coloured Petri Nets, enabling their simulation, generating occurrence (reachability) graphs, 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 consisting in translation of a UML specification to the input language of some model checker, and then performing verification using the model checker. Some of the approaches [16, 21] translate UML to the

Promela language and then make use of the Spin [14] model checker. Other [10, 20] exploit timed automata as an intermediate formalism and employ the UPPAAL [3] for verification. The third group of tools (e.g., [11]) applies the symbolic model checker NuSMV [7] via translating UML to its input language. One of the modules of Verics follows this idea. A 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.

There are only a few tools designed for parametric verification. One of UPPAAL's modules offers a possibility to verify some properties of Parametric Timed Automata [15] – namely, reachability with additional time constraints; LPMC [33] (a module of the TVS toolset) offers parametric verification of Timed Automata based on partition refinement; Moby/DC [9] implements model checking algorithms for Phase Automata; the Romeo tool contains a module aimed at parametric verification of Time Petri Nets with stopwatches [34]. To our best knowledge, there are no tools dealing with parametric verification of UML.

The rest of the paper is organised as follows. Section 2 introduces Elementary Net Systems (ENS), Time Petri Nets (TPN) (both restricted to 1-safe nets, as required by our tool), and the subset of UML recognized by the tool. Main ideas behind Bounded Model Checking and parametric verification are presented in Section 3, while Section 4 describes major steps of implementation of parametric BMC. In Section 5 we present some experimental results, whereas Section 6 contains some concluding remarks and discusses directions of future work.

2 Input Formalisms of Verics

In this section we define the input formalisms of Verics, for which parametric verification is available. These are Elementary Net Systems, Distributed Time Petri Nets, and a subset of UML.

2.1 Elementary Net Systems (ENS)

Elementary Net Systems (called also Elementary Petri Nets) are one of the formalisms used to specify concurrent systems. An *elementary net system* is a 4-tuple $EN = (P, T, F, m^0)$, where P (the *places*) and T (the *transitions*) are finite sets s.t. $P \cap T = \emptyset$, the *flow relation* $F \subseteq (P \times T) \cup (T \times P)$ has the property that for every $t \in T$ there exist $p, p' \in P$ s.t. $(p, t), (t, p') \in F$, and $m^0 \subseteq P$ is the *initial marking* (*initial configuration*). For each transition $t \in T$ we define the set of ingoing places $pre(t) = \{p \in P \mid (p, t) \in F\}$, and the set of outgoing places $post(t) = \{p \in P \mid (t, p) \in F\}$. A state (configuration) of the system EN is given by a *marking* of N, i.e., by a set of places $m \subseteq P$, represented graphically as containing "tokens". We consider 1-*safe* nets only, i.e., these in which each place can be marked by at most one token. Let $m \subseteq P$ be a configuration.

If t is a transition, $pre(t) \subseteq m$ and $(post(t) \setminus pre(t)) \cap m = \emptyset$, then we say that the transition t is *enabled* in m (denoted by $m[t\rangle$). Let $m, m' \subseteq P$ be two configurations. A transition t *fires* from m to m' (denoted by $m[t\rangle m'$) if $m[t\rangle$ and $m' = (m \setminus pre(t)) \cup post(t)$. A configuration $m \subseteq P$ is *reachable* if there are configurations $m_0, m_1, \ldots, m_n \subseteq P$ with $m_0 = m^0, m_n = m$, and transitions $t_0, t_1, \ldots, t_n \in T$ such that $m_{i-1}[t_i\rangle m_i$ for all $1 \leq i \leq n$. We denote the set of all the reachable configurations of EN by M_{EN}.
An example of an elementary net system with $P = \{w_i, c_i, r_i, p\}$ and $T = \{in_i, out_i, d_i\}$, for $i = 1, \ldots, n$ (modelling a mutual exclusion protocol) is presented in Fig. 2.

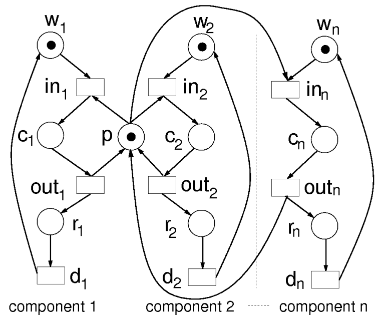


Fig. 2. Elementary net system for a mutual exclusion protocol

2.2 Time Petri Nets

The tool Verics accepts also an input in the form of Time Petri Nets, which are one of the formalisms for specifying real-time systems. A Time Petri Net is a tuple $N = (P, T, F, m^0, Eft, Lft)$, in which the elements P, T, F, and m^0 are the same as in an ENS, while $Eft : T \to \mathbb{N}$ and $Lft : T \to \mathbb{N} \cup \{\infty\}$, satisfying $Eft(t) \leq Lft(t)$ for each $t \in T$, are functions which assign to the transitions their *earliest* and *latest firing times*. The values of these functions are represented graphically as intervals annotating the transitions. In the current version of Verics, we consider **Distributed Time Petri Nets** [29] only[5]. A Distributed Time Petri Net consists of a set of 1-safe sequential[6] TPNs (called

[5] The restriction follows from efficiency reasons: the distributed form allows to reduce the size of the timed part of the concrete states of nets by assigning clocks to the processes (instead of assigning them to the transitions like in the standard case), which is important for optimising the implementation. Similarly, 1-safety allows to reduce the size of the information stored in the marking part of concrete states.

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

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 (a detailed description of the nets as well as their semantics can be found in [30]). An example of a distributed TPN (Fischer's mutual exclusion proto-

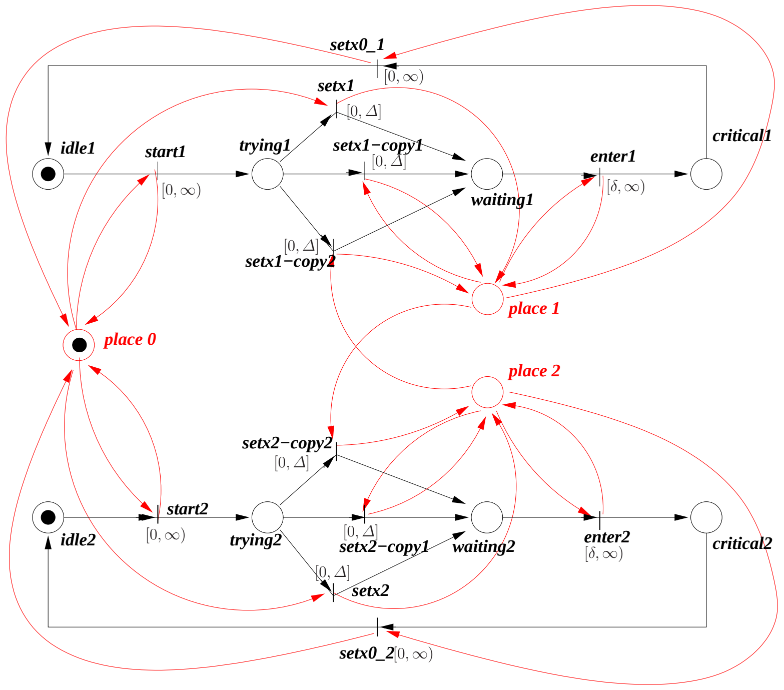


Fig. 3. A net for Fischer's mutual exclusion protocol for $n = 2$

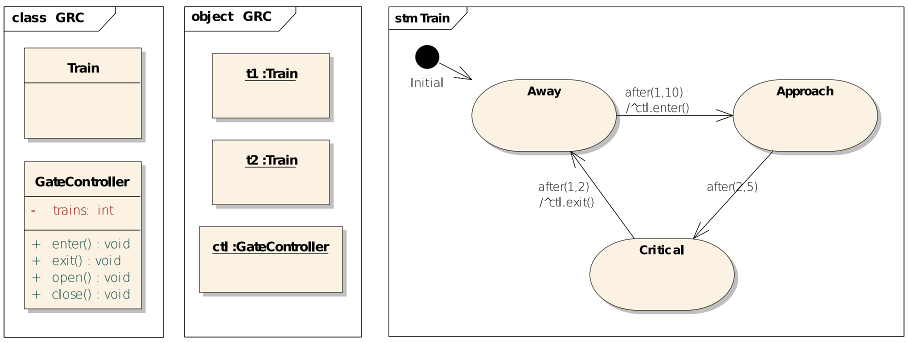
col[7]) is shown in Fig. 3. The net consists of three communicating processes with the sets of places $P_i = \{idle_i, trying_i, enter_i, critical_i\}$ for $i = 1, 2$, and $P_3 = \{place0, place1, place2\}$. All the transitions of the process N_1 and all the transitions of the process N_2 (i.e., of the nets modelling the competing processes) are joint with the process N_3 (modelling the process aimed at coordinating the access).

[7] The system consists of n processes (here $n = 2$) trying to enter their critical sections, and one process aimed at coordinating their access. The system is parameterised by the time-delay constants δ and Δ, whose relationship influences preservation of the mutual exclusion property.

2.3 UML

Another language handled as an input by our tool - a subset of UML - is sketched below. The syntax is illustrated with the diagrams of the Generalised Railroad Crossing (GRC) system, which is also used as a benchmark in Section 5. Due to the space limitations we give only intuitive explanations, but all the remaining details and formal definitions can be found in the papers [24, 25].

The systems considered are specified by a single class diagram, which defines k classes (e.g. see Fig. 4(a)), a single object diagram which defines n objects (e.g. in Fig. 4(a)), and k state machine diagrams (e.g. in Fig. 4(b), 5), each one assigned to a different class of the class diagram.



(a) Class and object diagrams (b) State machine diagram of class Train

Fig. 4. Specification of GRC system

The class diagram defines a list of attributes and a list of operations (possibly with parameters) for each class. The object diagram specifies the instances of classes (objects) and (optionally) assigns the initial values to variables. All the objects are visible globally. Moreover the set of objects is constant during the life time of the system and therefore dynamic object creation and termination is not allowed. Each object is assigned an instance of a state machine that determines the behaviour of the object. A state machine diagram typically consists of states, regions and transitions connecting source and target states. We consider several types of states, namely: simple states (e.g., *Away* in Fig. 4(b)), composite states, (e.g., *Main* in Fig. 5), final states, and initial pseudo-states, (e.g., *Initial* in Fig. 5). The areas filling the composite states are called *regions*. The regions contain states and transitions, and thus introduce a *hierarchy* of state machines. The labels of transitions are expressions of the form $trigger[guard]/action$, where each of these components can be empty. The transitions with non-empty trigger are called *triggered transitions*. A transition can be fired if the source state is *active*, the guard (a Boolean expression) is satisfied, and the trigger matching event occurs. An event can be of the following three types: an *operation call*, a *completion event*, or a *time event*. The operation calls coming to the given object are put into the *event queue* of the object, and then, one at a time, they are handled. The completion events and the time events are processed without placing them in queues.

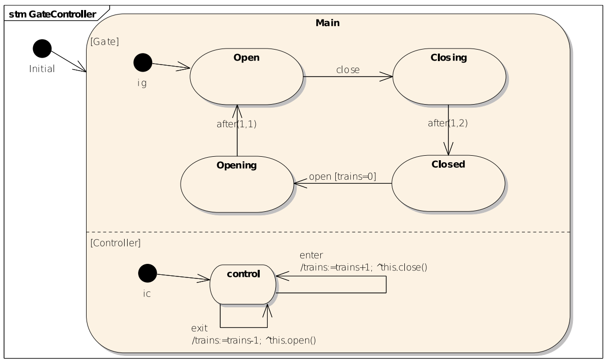


Fig. 5. Specification of GRC system - state machine diagram of class GateController

The highest priority is bound to completion events. A completion event occurs for a state, when all internal activities of the state terminate. Then the completion event is discarded (when it cannot fire any transition – level 1 of the hierarchy), or it fires a transition and is consumed (level 2). If none of the above cases holds, triggered transitions are considered. The event from the head of the queue, or a time event possibly fires a transition (level 3), and is consumed. If the event from the head of the queue cannot fire any transition and the matching trigger is *deferred* in the current state, then the event is deferred (level 4), i.e., it will be processed later. Otherwise, the event is discarded (level 5). We refer to the processing of a single event from the queue or a time event as the *Run-To-Completion (RTC) step*. Next, an event can be handled only if the previous one has been fully processed, together with all the completion events which eventually have occurred.

The execution of the whole system follows the interleaving semantics, similar to [10]. During a single step only one object performs its RTC step. If more than one object can execute such a step, then an object is chosen in a non-deterministic way. However, if none of the objects can perform an *untimed action*, then the time flows (level 6).

3 Bounded Model Checking methodology

In this section we present some basic ideas behind Bounded Model Checking and parametric verification.

3.1 SAT-based 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 using only a part of its model.

In order to apply SAT-based BMC to testing whether a system satisfies a certain (usually undesired) property, we unfold the transition relation of a given system up to some depth k, and encode this unfolding as a propositional formula. Then, the formula expressing a 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 witness 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.

All the above processes are performed fully automatically and do not require any assistance from the user.

3.2 Parametric Verification

The parametric verification performed in our system is of two kinds. In the case of systems modelled by Elementary Petri Nets, BMC is applied to verification of properties expressed in the existential fragment of the logic PRTCTL [12], i.e., PRTECTL. The logic PRTCTL is an extension of Computation Tree Logic (CTL), which allows to formulate properties involving lengths of paths in a model. Informally, the fragment of PRTCTL we consider consists of all the existential CTL formulas, these formulas augmented with superscripts of the form $\leq c$ (where $c \in \mathbb{N}$) restricting the length of the path of interest, and the formulas as above, but with c replaced by a linear expression over a set of natural-valued parameters, preceded by the existential or universal quantifier over (possibly bounded) valuations of these parameters. As to give an example, consider $\mathrm{EG}^{\leq 3}p$, which expresses that there is a path such that in the first four states of this path p holds. Another example is $\forall_{\Theta_1 \leq 1} \exists_{\Theta_2 \leq 2} \mathrm{EF}^{\leq \Theta_1 + \Theta_2} p$, which expresses that for each value of the parameter Θ_1 not greater than 1 there is a value of the parameter Θ_2 not exceeding 2 s.t. for some path we can reach p while visiting not more than $\Theta_1 + \Theta_2 + 1$ first states of that path. The propositions appearing in these formulas correspond to the places of the net considered. To the aim of verification using BMC, the qualitative properties expressed in PRTECTL are directly encoded as propositional formulas. Using some basic features of the considered logic, we introduce bounds on the parameters where not present, and replace the universal and existential quantifiers with, respectively, conjunctions and disjunctions.

Parametric BMC for PRTECTL. The verification of PRTECTL is performed by a translation of a part of a model, together with a property to be checked, to a propositional formula. We have proved in [19] that the method is correct and theoretically complete, albeit in practice translated formulas might be too complex for current SAT-solvers – that is often the case, when the depth of the unwinding is close to the size of a model. We use the Kripke structures induced by Petri nets (i.e., the marking graphs) as the models for PRTECTL logic, where the transitions are assumed to take one unit of time. Such an approach has several benefits listed below.

1. Our choice of parameterized logic is practical, as it is known that the parameterized extensions of temporal logics tend to be difficult problems in model-checking. For example, the joint complexity of verification of the existential fragment of TCTL, is 3EXPTIME [5], which is due to the presence of satisfiability in Presburger Arithmetics as a subproblem.
2. It is shown in [12] that the complexity of model-checking of a PRTCTL formula ϕ containing k parameters, over a Kripke model M, is $O(|M|^{k+1} \cdot |\phi|)$, where $|M|$ stands for the size of M and $|\phi|$ stands for the length of ϕ. Notice, however, that the Kripke structures induced by Petri nets tend to be very large (exponential in the size of the nets), hence the general verification problem may be untractable for many real-life systems. The BMC approach allows for checking some of difficult properties in a part of the model, giving a chance to succeed where the explicit approaches fail.
3. The PRTECTL logic can be model-checked by means of the standard BMC, using an extension of the efficient translation proposed in [37]. This is a serious advantage as the encoding employs an adequate set of symbolic paths for each subformula of a given formula.

A detailed description of the method can be found in [19].

Parametric Reachability. Parametric verification of Time Petri Nets and systems specified in UML is restricted to testing parametric reachability only. Again, the propositions used to build the properties tested correspond to the places of the nets. Besides "ordinary" reachability verification, VerICS enables searching for the minimal time $c \in \mathbb{N}$ in which a state satisfying p can be reached, or finding the minimal time $c \in \mathbb{N}$ after which p does not hold. This is formally expressed respectively as finding the minimal c such that $\mathrm{EF}^{<c}p$ (or $\mathrm{EF}^{\leq c}p$) holds, and finding a maximal c such that $\mathrm{EF}^{>c}p$ (or $\mathrm{EF}^{\geq c}p$) holds. To this aim, *parametric reachability checking* is used. The algorithm for finding the minimal $c \in \mathbb{N}$ such that $EF^{\leq c}p$ holds is as follows:

1. Using the standard BMC approach, find a reachability witness of minimal length[8],
2. Read from the witness the time required to reach p (denoted x). Now, we know that $c \leq \lceil x \rceil$ (where $\lceil \cdot \rceil$ is the *ceiling* function),
3. Extend the verified TPN with a new process N, which is composed of one transition t s.t. $Eft(t) = Lft(t) = n$, and two places p_{in}, p_{out}, where p_{in} is the input place and p_{out} is the output place of t,

[8] If we cannot find such a witness, then we try to prove unreachability of p.

4. Set n to $\lceil x \rceil - 1$,
5. Run BMC to test reachability of a state satisfying $p \wedge p_{in}$ in the extended TPN,
6. If such a state is reachable, set $n := n - 1$ and go to 5,
7. If such a state is unreachable, then $c := n + 1$, STOP.

The algorithms for other cases are similar. The details can be found in [30].

Concerning the systems specified in UML, VerICS offers similar options as in the case of Time Petri Nets, i.e., reachability and parametric reachability verification. However, in this case we are restricted to searching for the minimal integer c s.t. $EF^{\leq c}p$ holds in the model. The algorithm is based on the same idea as before. The main differences can be formulated as follows:

– instead of the additional process we introduce an additional clock that is never reset,
– the values of the clocks can take only natural numbers.

The details of the verification method can be found in [25].

4 The implementation

Our goal is to symbolically represent all the possible computations of a fixed length k of a given system. In all the cases, i.e., for ENS, TPN, and UML, the main idea is similar: we represent a set of global states (configurations) as a single *symbolic state*, i.e., as a vector of propositional variables $\mathbf{w}$ (called a *state variable vector*). Then, $k + 1$ state variable vectors stand for a symbolic k-path, where the first symbolic state encodes the initial state of the system, and the last one corresponds to the last states of the k-paths. Next, we encode the transition relation over the symbolic k-path as a propositional formula, and compute a formula encoding the property tested. Hence, in the general case, the formula encoding the symbolic k-path is defined as follows:

$$path_k(\mathbf{w}^0, \ldots, \mathbf{w}^k) = \mathfrak{I}(\mathbf{w}^0) \wedge \bigwedge_{i=0}^{k-1} \mathfrak{T}(\mathbf{w}^i, \mathbf{w}^{i+1}) \qquad (1)$$

where $\mathfrak{I}(\mathbf{w}^0)$ encodes the initial state of the system, and $\mathfrak{T}(\mathbf{w}^i, \mathbf{w}^{i+1})$ encodes a transition between symbolic states represented by the global state vectors $\mathbf{w}^i$ and $\mathbf{w}^{i+1}$.

In what follows we shortly describe how to define $\mathfrak{T}(\mathbf{w}^i, \mathbf{w}^{i+1})$ for ENS, TPN, and systems in UML.

4.1 Implementation for ENS

Consider an Elementary Net System $EN = (P, T, F, m^0)$, where the places are denoted with the integers smaller or equal than $n = |P|$. We use the set $\{p_1, \ldots, p_n\}$ of propositions, where p_i is interpreted as the presence of a token in the place i. We define the model $M = (S, \rightarrow, \mathcal{L})$ for ENS, where $S = M_{EN}$ is the set of the reachable configurations, $s \rightarrow s'$ iff there exists $t \in T$ such that $s[t\rangle s'$ for $s, s' \in S$, and $p_i \in \mathcal{L}(s)$ iff $i \in s$. The states of S are encoded by valuations of a vector of state variables $\mathbf{w} = (\mathbf{w}[1], \ldots, \mathbf{w}[n])$, where $\mathbf{w}[i] = p_i$ for $0 \leq i \leq n$. The initial state and the transition relation $\rightarrow$ are encoded as follows:

- $I_{m^0}(w) := \bigwedge_{i \in m^0} w[i] \wedge \bigwedge_{i \in P \setminus m^0} \neg w[i]$,
- $\mathfrak{T}(\mathbf{w}, \mathbf{v}) := \bigvee_{t \in T} \big(\bigwedge_{i \in pre(t)} \mathbf{w}[i] \wedge \bigwedge_{i \in (post(t) \setminus pre(t))} \neg \mathbf{w}[i] \wedge \bigwedge_{i \in (pre(t) \setminus post(t))} \neg \mathbf{v}[i]$ $\wedge \bigwedge_{i \in post(t)} \mathbf{v}[i] \wedge \bigwedge_{i \in (P \setminus (pre(t) \cup post(t))) \cup (pre(t) \cap post(t))} \mathbf{w}[i] \iff \mathbf{v}[i] \big)$.

As to give an outline of the encoding of the PRTECTL formulas, consider the formula $\beta = \exists_{\Theta \leq c} \alpha(\Theta)$, where $\alpha(\Theta)$ is a PRTECTL formula containing free variable Θ, and $c \in \mathbb{N}$. For a given depth k of the unfolding let $d = min(c, k)$. We define the translation of β in m–th symbolic state of n–th symbolic path, using the set of indices of symbolic k–paths A, as follows:

$$\big[\beta\big]_k^{[m,n,A]} := \big[\alpha(d)\big]_k^{[m,n,\hat{g}_L(A, f_k(\alpha(d)))]} \vee \big[\exists_{\Theta \leq d-1} \alpha(\Theta)\big]_k^{[m,n,\hat{g}_L(A, f_k(\exists_{\Theta \leq d-1} \alpha(\Theta)))]},$$

where f_k a function returning the number of k–paths sufficient to perform the translation of a given formula, and $\hat{g}_L(A, b)$ returns the subset of b least elements of A. The f_k function is an extension of its counterpart for ECTL from [31].

Similarly, consider the formula $\gamma = \forall_{\Theta \leq c} \alpha(\Theta)$. The translation of γ is defined as follows:

$$\big[\gamma\big]_k^{[m,n,A]} := \big[\alpha(c)\big]_k^{[m,n,\hat{g}_L(A, f_k(\alpha(c)))]} \wedge \big[\forall_{\Theta \leq c-1} \alpha(\Theta)\big]_k^{[m,n,\hat{g}_R(A, f_k(\forall_{\Theta \leq c-1} \alpha(\Theta)))]},$$

where $\hat{g}_R(A, b)$ returns the subset of b greatest elements of A.

The consecutive application of any of the above rules leads to the conjunction of $d+1$ propositional formulas containing no existential quantifiers. These formulas are in turn translated using the efficient translation inspired by its counterpart for ECTL (see [37]), where $\hat{g}_L$ and $\hat{g}_R$ are introduced along with several other methods for an optimal path selection.

A more detailed description and the encoding of the formulas of PRTECTL can be found in [19].

4.2 Implementation for TPN

The main difference between symbolic encoding of the transition relation of ENS and TPN consists in the time flow. A current state of a TPN $\mathcal{N}$ is given by its marking and the time passed since each of the enabled transitions became enabled (which influences the future behaviour of the net). Thus, a *concrete state* σ of $\mathcal{N}$ can be defined as an ordered pair $(m, clock)$, where m is a marking, and $clock : \mathfrak{I} \rightarrow \mathbb{R}_+$ is a function, which for each index $\mathfrak{i}$ of a process of $\mathcal{N}$ gives the time elapsed since the marked place of this process became marked most recently [32].

Given a set of propositional variables PV, we introduce a *valuation function* $V_c : \Sigma \to 2^{PV}$, which assigns the same propositions to the states with the same markings. We assume the set PV to be such that each $q \in PV$ corresponds to exactly one place $p \in P$, and use the same names for the propositions and the places. The function V_c is defined by $p \in V_c(\sigma) \Leftrightarrow p \in m$ for each $\sigma = (m, \cdot)$. The structure $M_c(\mathcal{N}) = ((T \cup \mathbb{R}_+, \Sigma, \sigma^0, \to_c), V_c)$ is called a *concrete* (*dense*) *model of* $\mathcal{N}$. It is easy to see that concrete models are usually infinite.

In order to deal with countable structures instead of uncountable ones, we introduce *extended detailed region graphs* for distributed TPNs. They correspond to the well-known graphs defined for timed automata in [1] and adapted for Time Petri Nets [26, 35], but involve disjunctions of constraints, the reflexive transitive closure of the time successor of [1], and make no use of the maximal constant appearing in the invariants and enabling conditions. To do this, we assign a clock to each of the processes of a net.

To apply the BMC approach to verification of a particular distributed Time Petri Net $\mathcal{N}$, we deal with a model obtained by a *discretisation* of its extended detailed region graph. The model is of an infinite but countable structure, which, however, is sufficient for BMC (which deals with finite sequences of states only). Instead of dealing with the whole extended detailed region graph, we *discretise* this structure, choosing for each region one or more appropriate representatives. The discretisation scheme is based on the one for timed automata [36], and preserves the qualitative behaviour of the underlying system. The details and the formal definitions can be found in [30].

4.3 Implementation for UML

Symbolic semantics. Below we sketch a symbolic encoding of the operational semantics introduced in [24, 25].As usual, the global states are represented by state variable vectors. However each global state g is represented by n subvectors, where each one stands for a state of one object (n is the number of objects in the system). The representation of a state of a single object consists of five components that encode respectively a set of active states, a set of completed states, a contents of the event queue, a valuation of the variables, and a valuation of the clocks.

Moreover, according to the OMG semantics, the transition relation is *hierarchical*, i.e., we distinguish between 6 levels (types) of transitions, where the ordering follows their priorities. Therefore, we ensure that a transition of each level becomes enabled only if the transitions of the preceding levels cannot be executed, by nesting the conditions for the consecutive levels [24, 25].

Symbolic encoding. The implementations described above for Timed Petri Nets were closely following the general idea of implementing formalisms based on transitions systems, e.g., with symbolic states capable of representing every part of system states, and Boolean constraints added for transitions with respective guards, invariants, etc. While the symbolic operational semantics of UML could be expressed in terms of timed automata or Petri nets, the resulting translation would be highly complex because of the need for representing by means of automata every language construct, most of which are semantically not a close match with this formalism. We have tried another way by encoding these constructions directly into propositional logic, without introducing any intermediate transition systems. The general idea can be described as follows: states of state machines are encoded directly, by assigning boolean variables in symbolic vectors representing system states. Then, propositional formulas are defined for encoding every transition type. Finally, these formulas are structured according to the hierarchy of corresponding transitions (see Section 2.3, page 7), giving as a result the formula encoding the transition relation.

We define propositional formulas $EOi(o, \mathbf{w})$ for transitions of types $1 \leq i \leq 5$ that encode their preconditions over the vector $\mathbf{w}$ for the object o. Also we define the propositional formulas $XOi(o, \mathbf{w}, \mathbf{v})$ for $1 \leq i \leq 5$ encoding an execution of these transitions over the vectors $\mathbf{w}, \mathbf{v}$ for the object o and the formula $X6(\mathbf{w}, \mathbf{v})$ encoding the time flow.

The transitions of types 1–5 are called *local* as their execution does not depend on the type of a transition that can be fired by other objects. The execution of local transitions for object o over the vectors of the state variables $\mathbf{w}$ and $\mathbf{v}$ is recursively encoded as:

$$\begin{aligned} f_5(o, \mathbf{w}, \mathbf{v}) &= EO5(o, \mathbf{w}) \wedge XO5(o, \mathbf{w}, \mathbf{v}) \\ f_i(o, \mathbf{w}, \mathbf{v}) &= EOi(o, \mathbf{w}) \wedge XOi(o, \mathbf{w}, \mathbf{v}) \\ &\quad \vee \neg EOi(o, \mathbf{w}) \wedge f_{i+1}(o, \mathbf{w}, \mathbf{v}) \texttt{ for } i \in \{1,2,3,4\}, \end{aligned} \tag{2}$$

and we denote $f_1(o, \mathbf{w}, \mathbf{v})$ by $XO(o, \mathbf{w}, \mathbf{v})$.

We ensure that a transition of each level becomes enabled only if the transitions of the preceeding levels cannot be executed, by nesting the conditions for the consecutive levels. All the components of the encoded model are represented by propositional formulas. For example, an event queue is encoded along with the corresponding operations of inserting and removing an event, testing if a queue is empty or full and so on. A queue is modeled by a circular buffer, with indices pointing to first deferred event, an event to be processed next, and the first free position of the queue (a place for inserting new events).

Then, iterating over the objects of class c, we encode the execution of the local transitions for the class c:

$$XC(c, \mathbf{w}, \mathbf{v}) = \bigvee_{o \in Objects(c)} XO(o, \mathbf{w}, \mathbf{v}). \tag{3}$$

Now, we are ready to give the encoding of the transition relation:

$$\mathfrak{T}(\mathbf{w}, \mathbf{v}) = \bigvee_{c \in Classes} XC(c, \mathbf{w}, \mathbf{v}) \vee E6(\mathbf{w}) \wedge X6(\mathbf{w}, \mathbf{v}), \tag{4}$$

where $E6(\mathbf{w})$ encodes the enabling conditions of the time flow transition.

In this way we have managed to deal with the hierarchical structure of the transition relation simply by nesting the appropriate formula encoding every level of the hierarchy.

5 Experimental Results

We present some experimental results for parametric verification of systems specified as ENS, Distributed TPN, or in our subset of UML. We deal with standard benchmarks, i.e., timed and untimed versions of the mutual exclusion protocol and the Generalised Railroad Crossing system as well as with timed and untimed versions of Generic Pipeline Paradigm model [28].

Our motivation for presenting the results is to give an idea about the overall performance and about typical problems the tool deals with. To the best of our knowledge there are no other tools performing parametric model checking for a similar input, so we cannot follow a common pattern of comparing several tools for the same input. However, we provide some comparisons with other tools for non-parametric problems in order to give a hint about the efficiency of VerICS.

5.1 Elementary Net Systems

In the case of Elementary Petri Nets we have tested two models. The first one is a system which models the well-known mutual exclusion problem, shown in Fig. 2. The system consists of $n+1$ processes (where $n \geq 2$) of which n processes compete for the access to the shared resource, while one process guards that no two processes use the resource simultaneously. The presence of a token in the place labelled by w_i means that the i-th process is waiting for the access to the critical section while the token in c_i means that the i-th process has acquired the permission and entered the critical section. The place r_i models the unguarded part of the process and the presence of the token in the place p indicates that the resource is available.

We have explored the validity of the formula $\varphi_1^b = \forall_{\Theta \leq b} EF(\neg p \wedge EG^{\leq \Theta} c_1)$ with respect to the value of b. Intuitively, φ_1^b expresses that for each parameter valuation Θ bounded by b, there exists a future state such that the first process stays in the critical section for at least Θ time steps. It is quite obvious that the non-parameterized counterpart of the formula φ_1, i.e., $EF(\neg p \wedge EGc_1)$ does not hold in our model as no process can stay in the critical section forever. The experimental results for the protocol are presented in Table 1.

The Generic Pipeline Paradigm modelled by an Elementary Petri Net is the second system we have tested. The model consists of three parts, namely: the Producer which is able to produce data ($ProdReady$), the Consumer being able to receive data ($ConsReady$) and a chain of n intermediate Nodes – having capabilities of data receiving ($Node_i Ready$), processing ($Node_i Proc_j$), and sending ($Node_i Send$). Notice that the example is scalable in two ways: firstly – the number n of the processing Nodes can be increased in order to modify the length of the pipeline chain, secondly – the length m of the processing queue in each intermediate Node can be incremented, which may be interpreted as modifying the time spent on internal calculations. In the experiments we explore how these changes affect the validity of the formula tested and the tool performance.

formula	n	k	PBMC				MiniSAT	SAT?
			vars	clauses	sec	MB	sec	
φ_1^1	3	2	1063	2920	0.01	1.3	0.003	NO
φ_1^1	3	3	1505	4164	0.01	1.5	0.008	YES
φ_1^2	3	4	2930	8144	0.01	1.5	0.01	NO
φ_1^2	3	5	3593	10010	0.01	1.6	0.03	YES
φ_1^2	30	4	37825	108371	0.3	7.4	0.2	NO
φ_1^2	30	5	46688	133955	0.4	8.9	0.52	YES
φ_1^3	4	6	8001	22378	0.06	2.5	0.04	NO
φ_1^3	4	7	9244	25886	0.05	2.8	0.05	YES

Table 1. Elementary Net Systems: Mutual exclusion, testing the formula φ_1^b

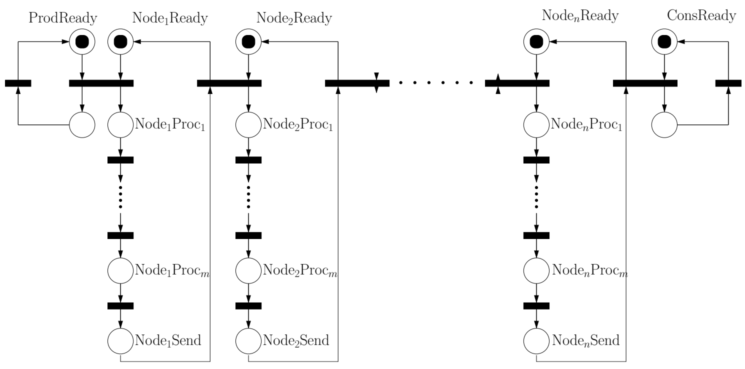


Fig. 6. Elementary Net Systems: Generic Pipeline Paradigm

We have tested the following property:

$$\varphi_4^{n,m} = \forall_{\Theta \leq nm-1} EFEG^{\leq \Theta}(\neg ProdReady \wedge \neg ConsReady \wedge \bigvee_{i=1}^{n} \neg Node_i Ready).$$

The intuitive meaning of $\varphi_4^{n,m}$ is that it is possible that for some state of the system, during some time (bounded by $nm - 1$) neither Producer is able to produce, nor Consumer is able to receive, while the intermediate Node chain is processing or transferring data. The experimental results for the Generic Pipeline Paradigm modelled by an Elementary Petri Net are presented in Table 2.

The experiments were performed on a Linux machine with dual core 1.6 GHz processor; satisfiability was tested using the MiniSAT solver [23].

5.2 Distributed Time Petri Nets

We have tested Distributed Time Petri Nets specifying the standard (timed) *Fischer's mutual exclusion protocol* (mutex) [2] and a timed version of Generic Pipeline Paradigm.

formula	n	m	k	PBMC				MiniSAT	SAT?
				vars	clauses	sec	MB	sec	
$\varphi_4^{2,1}$	2	1	6	4086	11315	0.03	2.07	0.008	NO
$\varphi_4^{2,1}$	2	1	7	4696	13079	0.036	2.7	0.02	YES
$\varphi_4^{2,2}$	2	2	8	5980	16811	0.06	2.6	0.01	NO
$\varphi_4^{2,2}$	2	2	9	13484	37927	0.08	2.7	0.06	YES
$\varphi_4^{2,3}$	2	3	10	8844	24873	0.07	2.9	0.02	NO
$\varphi_4^{2,3}$	2	3	11	9776	27509	0.08	3.1	0.2	YES
$\varphi_4^{3,1}$	3	1	8	7416	20739	0.04	2.6	0.01	NO
$\varphi_4^{3,1}$	3	1	9	8292	23207	0.03	2.7	0.07	YES
$\varphi_4^{3,2}$	3	2	11	20025	56568	0.12	4.6	0.04	NO
$\varphi_4^{3,2}$	3	2	12	21768	61517	0.14	4.7	0.43	YES
$\varphi_4^{10,1}$	10	1	24	74488	212315	0.4	13.2	0.43	NO
$\varphi_4^{10,1}$	10	1	25	77548	221055	0.52	13.6	19.2	YES
$\varphi_4^{10,2}$	10	2	32	111844	320863	1.6	37.3	0.98	NO
$\varphi_4^{10,2}$	10	2	33	230812	662175	1.64	38.4	23.1	YES

Table 2. Elementary Net Systems: Generic Pipeline Paradigm, testing the formula $\varphi_4^{n,m}$

The mutex model consists of N Time Petri Nets, each one modelling a process, together with one additional net used to coordinate their access to the critical sections. A distributed TPN modelling the system for $n = 2$ is shown in Fig. 3. *Mutual exclusion* means that no two processes can reach 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 parametrizing the system: they correspond to the earliest firing time of the the transition entering the critical section and the latest firing time of the transition entering the waiting section, respectively. In particular, the following property holds: "*Fischer's protocol ensures mutual exclusion iff $\Delta < \delta$*".

We have searched for the minimal c such that $\mathrm{EF}^{\leq c}p$ holds. The results are presented in Table 3 (see the table denoted with **Tb**). We considered the case, where $\Delta = 2$ and $\delta = 1$, and the net of 25 processes. The witness was found for $k = 12$, while the time of the path found was between 8 and 9. The column marked with n shows the values of the parameter in the additional component. For $n = 1$ and $k = 12$ unsatisfiability was returned. Testing the property on a longer path could not be completed in a reasonable time.

Comparing the last two rows of Table **Tb** one can see the typical behaviour of SAT testers, namely that diagnosing unsatisfiability is usually much harder than satisfiability given formulas of similar size.

We have compared the efficiency of Verics with the tool Romeo [22], using exactly the same mutex benchmark and testing the same property. The left part (denoted with **Ta**) of Table 3 contains the results, where N is the number of processes.

Ta	Romeo		VerICS	
N	**sec**	**MB**	**sec**	**MB**
2	0.1	1	0.65	3
3	0.14	6	2.65	4
4	0.61	9	1.92	5
5	4.99	38	2.93	6
6	44.76	364	6.02	7
7	>360	>2000	8.63	9
8	-	-	12.67	12
9	-	-	20.74	15
10	-	-	30.28	23
12	-	-	43.72	32
15	-	-	136.78	92
20	-	-	551.64	299
25	-	-	1304.2	538

Tb		tpnBMC				RSat		
k	N	**variables**	**clauses**	**sec**	**MB**	**sec**	**MB**	**sat**
0	-	840	2192	0.02	1.688	0	1.4	NO
2	-	16263	47705	0.31	3.621	0	4.9	NO
4	-	33835	99737	0.66	5.805	0.3	9.1	NO
6	-	51406	151697	1.04	8.113	0.8	13.8	NO
8	-	72752	214851	1.49	10.81	8.5	27.7	NO
10	-	92629	273489	1.95	13.27	152.3	200.8	NO
12	-	113292	334355	2.4	15.84	6.7	39.0	YES
12	8	120182	354944	2.57	16.74	13.8	47.9	YES
12	7	120254	355178	2.67	16.74	5.7	38.3	YES
12	3	115675	341405	2.52	16.22	12.7	49.0	YES
12	2	115669	341387	2.56	16.23	3.3	32.3	YES
12	1	115729	341579	2.54	16.23	1100.1	538.4	NO
sum for all rows:				20.73	16.23	1304.2	538.4	

Table 3. Distributed Time Petri Nets: Results for mutex, $\Delta = 2$, $\delta = 1$, mutual exclusion violated. Left: comparison with Romeo. Right: details of verification with VerICS for 25 processes. The tpnBMC column shows the results for the part of the tool used to represent the problem as a propositional formula (a set of clauses); the column RSat displays the results of running the RSat solver for the set of clauses obtained from tpnBMC.

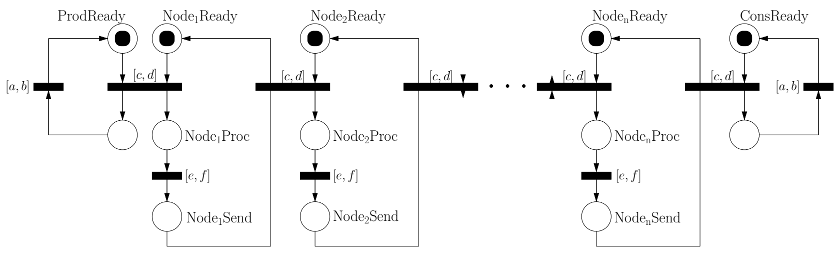


Fig. 7. Distributed Time Petri Nets: Generic Timed Pipelining Paradigm

The Generic Timed Pipelining Paradigm modelled by a Distributed Time Petri Net has a structure similar to its untimed version. Again, the system consists of Producer, Consumer, and a chain of n intermediate Nodes. As previously, the length of the chain is scalable, this time however we are able to manipulate the time properties of all the parts by means of parameters a, b, c, d, e, and f. In this way we can specify, for example, the maximal idle time allowed for Producer or Consumer. We have tested the following formula:

$$\varphi_5 := EF^{\leq \Theta}(\neg ProdReady \wedge \neg ConsReady).$$

Intuitively, it expresses that the system can reach, in the time smaller or equal than some value Θ, such a state that Producer is not able to produce, and Consumer receiving abilities are disabled. The aim of the experiment was to synthesize the smallest Θ value for which φ_5 holds, using parametric reachability methods. We have checked the property for various instances of Generic Timed Pipelining Paradigm, by setting some arbitrary values of the chain length n, and the time constraint parameters $a, b, c, d, e,$ and f. The results can be found in Table 4.

n	k	**time params**						**PBMC**				**RSat**		**synth.** Θ
		a	b	c	d	e	f	**clauses**	**vars**	**time**	**mem**	**time**	**mem**	
1	10	1	1	1	3	1	3	19686	6915	0.25	2.203	0.1	2.8	6
1	10	1	3	3	4	3	5	22928	7974	0.3	2.332	0.1	3.1	12
2	14	1	2	1	3	2	4	41467	14378	0.77	3.105	0.5	4.6	7
2	14	1	1	2	4	2	5	44191	15306	0.57	3.234	0.4	4.8	10
3	18	1	4	2	2	1	5	73913	25554	0.94	4.52	1.9	7.2	11
3	26	1	2	1	3	2	5	133423	46028	1.7	7.098	19.3	13.9	10
4	30	1	4	2	2	1	5	109339	37923	1.82	6.059	3.4	10	11
4	22	2	3	1	2	1	3	115001	29661	1.47	6.289	5.2	10.4	9
5	34	1	4	2	2	1	5	92628	268942	3.74	12.75	93.7	28.8	17
5	36	1	3	1	2	2	3	291710	100530	4.09	13.66	61.8	27	16
6	30	2	3	1	4	0	1	235115	81118	3.82	11.34	27.8	21.6	7
6	38	1	4	2	2	1	5	354108	121937	4.64	16.36	92.2	34.1	20
7	34	2	3	1	4	0	1	311252	107432	4.78	14.56	31.8	26.3	8
7	54	1	4	2	2	1	5	687242	236971	11.17	30.41	2565.0	177.2	23

Table 4. Distributed Time Petri Nets: Testing the formula φ_5. Memory in megabytes, time in seconds.

The experiments were performed on the computer equipped with Intel Pentium Core 2 Duo CPU (2.40 GHz), 2 GB main memory and the operating system Linux 2.6.28.

5.3 UML

The UML specification tested is a variant of the well known Generalised Railroad Crossing (GRC) benchmark (Fig. 4, 5). The system, operating a gate at a railroad crossing, consists of a gate, a controller, and N tracks which are occupied by trains. Each track is equipped with sensors that indicate a position of a train and send an appropriate message to the controller. Depending on the track occupancy the controller can either open or close the gate. We check whether a train can cross the road while the gate is not closed. The tests have been performed on the computer equipped with Pentium M 1.73 GHz CPU and 1.2 GB RAM running Linux.

Table 5 presents the results of verification, where N denotes the number of trains, and k - the depth of a symbolic path at which the tested property is satisfiable. The next columns show the time consumed by Hugo/Uppaal toolset, and our BMC4UML tool (reachability and parametric reachability testing). The minimal time in which property can be reached (as introduced in Section 3.2, p. 9) was found to be 6. The results in the column marked with the asterisk

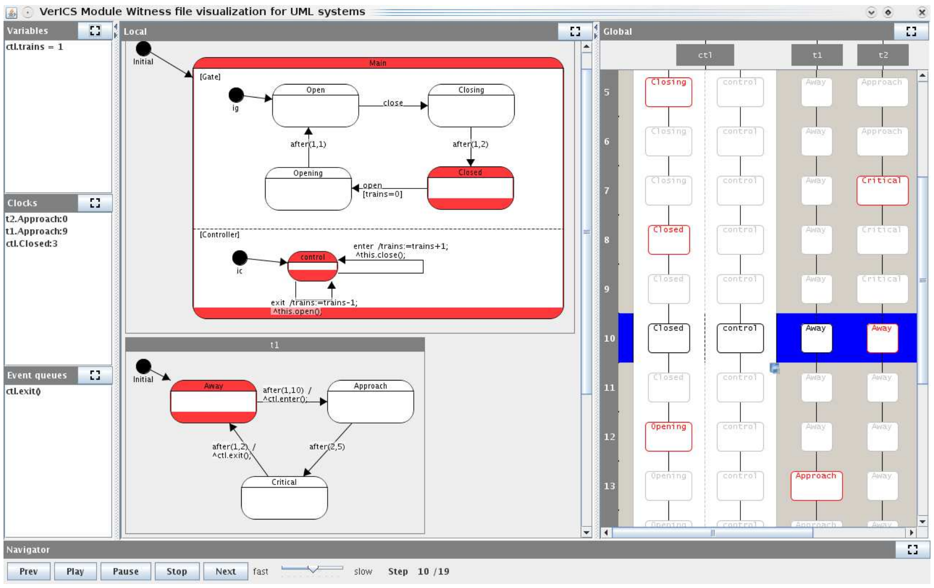


Fig. 8. GUI screenshot: Simulation of a UML witness

concern the symbolic paths of length 18 that do not start from the initial state of the GRC system, but from the state where all the trains are in the states *Away* and the object *ctl* is in the states *Main*, *Open*, and *control* (see Fig. 4, 5). In other words, the paths have been made shorter by the "initialisation part", which could be removed, because it always leads to the state defined above. This optimisation can be applied to all the systems for which the time must flow just after the initial transitions.

Another UML benchmark is Aircraft Carrier (AC - Fig. 9). AC consists of a ship and a number of aircrafts taking off and landing continuously, after issuing a request being accepted by the controller. The events of answering these requests may be marked as deferred. Each aircraft refills fuel while on board and burns

N	Hugo & Uppaal[s]	BMC4UML[s]			BMC4UML*[s]		
		k	reachability	par.reach.	k	reachability	par.reach.
3	2.89	24	86.07	140.9	18	40.44	27.51
4	175.41	25	139.4	83.45	18	50.41	85.01
5	>2500	26	221.4	240.9	18	59.90	131.5
6	—	27	1354.9	365.4	18	75.21	175.6
7	—	—	—	—	18	92.60	191.5
20	—	—	—	—	18	448.6	620.7

Table 5. UML: The results of verification of GRC

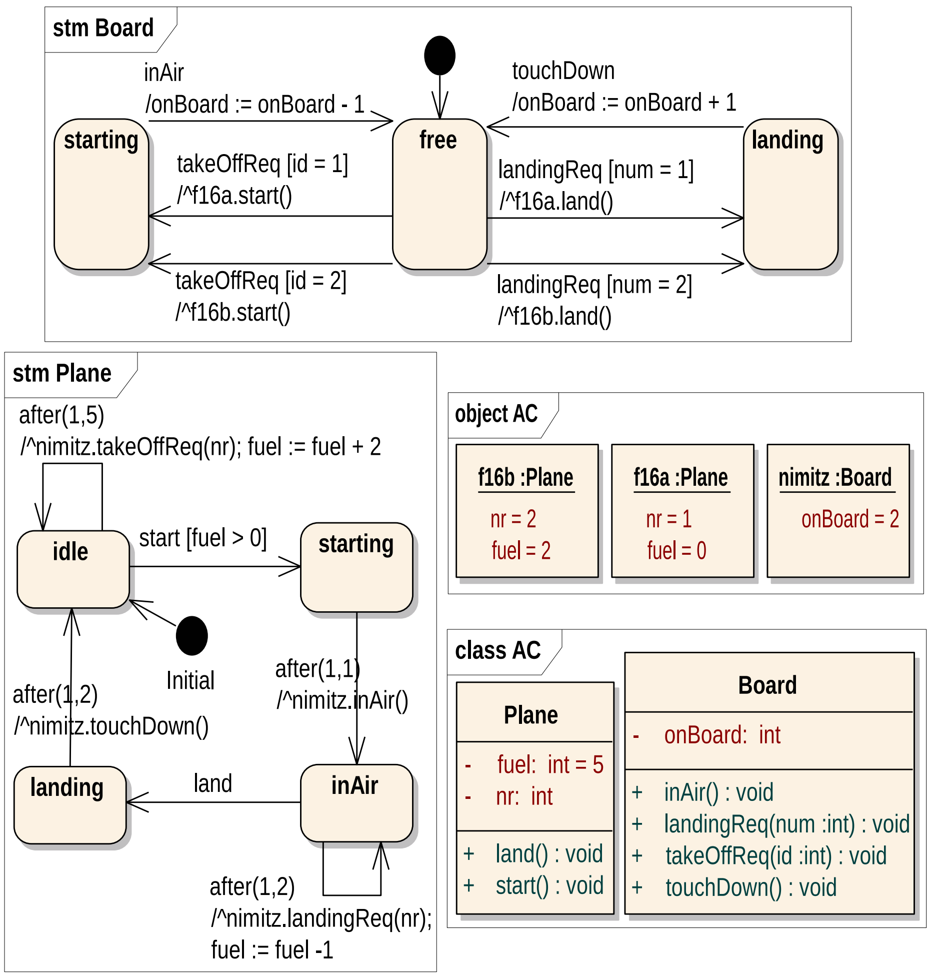


Fig. 9. UML diagrams of Aircraft Carrier

fuel while airborne. We check the property whether an aircraft can run out of fuel during its flight. The results are presented in Table 6, where N denotes the number of planes, and k the depth at which the property is satisfiable. The next columns stand for verification results of AC system in two versions: with and without deferred events, obtained with Hugo/Uppaal toolset and BMC4UML (reachability + parametric reachability testing, the value of the parameter c found is 4).

N	**k**	**Hugo+Uppaal** [s]		**BMC4UML** [s]	
		deferred	no defer	deferred	no defer
3	19	1.32	1.25	67.59 + 31.34	51.26 + 22.64
4	20	13.15	11.41	101.58 + 45.44	81.28 + 42.38
5	21	147.43	95.67	155.63 + 60.49	132.34 + 37.01
6	22	Out of mem		257.08 + 52.23	216.42 + 75.08
7	23	—		686.06 + 101.86	421.85 + 199.09

Table 6. UML: Results of verification of Aircraft Carrier system

Finally, if the tested property is satisfiable, then it can be visualised and simulated with the VerICS graphical user interface (see Fig. 8).

5.4 Analysis of the results

The experiments have confirmed the well-known fact that BMC is a method of falsification rather than verification. The method performs best when a counterexample is found before the resources are exhausted when increasing the depth of the unfolding. The parametric features seem to work best within the range of successful BMC applications.

Our general observation on the non-parametric features is that we have obtained the efficient state representations and the compact encoding of the transition relation for all the types of the presented systems. The overall performance is determined by the characteristics of the method to which these representations and encodings are applied (BMC in this case).

Concerning our parametric extensions, our motivation was to show that they can be added to the framework relatively easily. We learned that the best performance is achieved for the scenarios in which BMC works well, i.e., on short paths. The overhead introduced by testing parametric properties is then relatively small. The parameter synthesis in the parametric reachability problem is a difficult task, yet the results are quite promising – we have been able to test simple models in matter of seconds or minutes, however the verification of some properties in the large networks (e.g., Generic Timed Pipelining Paradigm with 7 nodes, $a = 1$, $b = 4$, $c = d = 2$, $e = 1$, $f = 5$) took a large amount of time (see Figure 4). The latter is mostly due to the size of the tested propositional formulas and may heavily depend on heuristics implemented in the applied SAT-solver.

The verification of PRTECTL properties seems to be quite efficient, despite the fact that the translation of the universal quantifier bounded with b results with the conjunction of length $b+1$. Notice that the disjunction being the result of the translation of the existential quantifier depends on the depth of the unwinding – hence in case of purely existential properties we can expect satisfactory results.

6 Final Remarks

We have presented the three new modules of VerICS aimed at SAT-based parametric verification, together with experimental results obtained for some typical benchmarks. The results prove efficiency of the method. However, it would be interesting to check practical applicability of parametric BMC for more complicated examples of Petri Nets and UML systems. On the other hand, in the future we are going to extend the tool to support parametric verification of other kinds of systems and properties.

As far as a comparison with other tools is concerned, in this paper we compared VerICS with Hugo (see Table 5) and Romeo (see Table 3). It seems that both these tools suffer from the exponential explosion for large systems, while VerICS can still deal with k-models for some small values of k.

The VerICS parametric toolset described in the paper can be downloaded from the site `http://verics.ipipan.waw.pl/parametric` together with all the benchmarks presented. All the input files for running experiments with Romeo, Hugo, and UppAal tools are available at `http://ii.uph.edu.pl/~artur/topnoc`.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model checking for real-time systems. In *Proc. of the 5th Symp. on Logic in Computer Science (LICS'90)*, pages 414–425. IEEE Computer Society, 1990.
2. R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. of the 13th IEEE Real-Time Systems Symposium (RTSS'92)*, pages 157–166. IEEE Computer Society, 1992.
3. 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.
4. 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.
5. V. Bruyére, E. Dall'Olio, and J-F. Raskin. Durations and parametric model-checking in timed automata. *ACM Transactions on Computational Logic*, 9(2), 2008.
6. 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.

7. 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.
8. P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS: A tool for verifying timed automata and Estelle specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 278–283. Springer-Verlag, 2003.
9. H. Dierks and J. Tapken. Moby/DC - a tool for model-checking parametric real-time specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 271–277. Springer-Verlag, 2003.
10. K. Diethers, U. Goltz, and M. Huhn. Model checking UML statecharts with time. In *Proc. of the Workshop on Critical Systems Development with UML (CS-DUML'02)*, pages 35–52. Technische Universität München, 2002.
11. 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.
12. E. A. Emerson and R. Trefler. Parametric quantitative temporal reasoning. In *Proc. of the 14th Symp. on Logic in Computer Science (LICS'99)*, pages 336–343. IEEE Computer Society, July 1999.
13. J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification. Third Edition*. Addison-Wesley, 2005.
14. G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Eng.*, 23(5):279–295, 1997.
15. T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. In *Proc. of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031 of *LNCS*, pages 189–203. Springer-Verlag, 2001.
16. 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.
17. 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.
18. M. Kacprzak, W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS 2008 - a model checker for time Petri nets and high-level languages. In *Proc. of Int. Workshop on Petri Nets and Software Engineering (PNSE'09)*, pages 119–132. University of Hamburg, 2009.
19. M. Knapik, W. Penczek, and M. Szreter. Bounded parametric model checking for elementary net systems. In *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) – submitted*. Springer, 2010.
20. 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.
21. 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.

22. Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for petri nets with stopwatches. In *TACAS*, pages 54–57, 2009.
23. MiniSat. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat, 2006.
24. A. Niewiadomski, W. Penczek, and M. Szreter. A new approach to model checking of UML state machines. *Fundamenta Informaticae*, 93 (1-3):289–303, 2009.
25. A. Niewiadomski, W. Penczek, and M. Szreter. Towards checking parametric reachability for UML state machines. In *Proc. of the 7th Int. Ershov Memorial Conf. 'Perspectives of Systems Informatics' (PSI'09)*, volume 5947 of *LNCS*, pages 319–330. Springer-Verlag, 2010.
26. Y. Okawa and T. Yoneda. Symbolic CTL model checking of time Petri nets. *Electronics and Communications in Japan, Scripta Technica*, 80(4):11–20, 1997.
27. OMG. Unified Modeling Language. http://www.omg.org/spec/UML/2.1.2, 2007.
28. D. Peled. All from one, one for all: On model checking using representatives. In *Proc. of the 5th Int. Conf. on Computer Aided Verification (CAV'93)*, volume 697 of *LNCS*, pages 409–423. Springer-Verlag, 1993.
29. 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.
30. W. Penczek, A. Półrola, and A. Zbrzezny. SAT-based (parametric) reachability for distributed time Petri nets. In *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) – accepted*. Springer, 2010.
31. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
32. A. Półrola and W. Penczek. Minimization algorithms for time Petri nets. *Fundamenta Informaticae*, 60(1-4):307–331, 2004.
33. R. L. Spelberg, H. Toetenel, and M. Ammerlaan. Partition refinement in real-time model checking. In *Proc. of the 5th Int. Conf. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'98)*, volume 1486 of *LNCS*, pages 143–157. Springer-Verlag, 1998.
34. L-M. Tranouez, D. Lime, and O. H. Roux. Parametric model checking of time Petri nets with stopwatches using the state-class graph. In *Proc. of the 6th Int. Workshop on Formal Analysis and Modeling of Timed Systems (FORMATS'08)*, volume 5215 of *LNCS*, pages 280–294. Springer-Verlag, 2008.
35. I. B. Virbitskaite and E. A. Pokozy. A partial order method for the verification of time Petri nets. In *Fundamental of Computation Theory*, volume 1684 of *LNCS*, pages 547–558. Springer-Verlag, 1999.
36. A. Zbrzezny. SAT-based reachability checking for timed automata with diagonal constraints. *Fundamenta Informaticae*, 67(1-3):303–322, 2005.
37. A. Zbrzezny. Improving the translation from ECTL to SAT. *Fundamenta Informaticae*, 85(1-4):513–531, 2008.