Parametric Bounded Model Checking for UML*

Artur Niewiadomski[1], Wojciech Penczek[1,2], and Maciej Szreter[2]

[1] ICS, University of Podlasie, Siedlce, Poland,
artur@iis.ap.siedlce.pl
[2] ICS, Polish Academy of Sciences, Warsaw, Poland,
{penczek,mszreter}@ipipan.waw.pl

Abstract. The paper presents a SAT-based approach to Bounded Model Checking of systems specified in UML. Contrary to other UML verification tools we do not exploit any of the existing model checkers as we do not translate UML specifications into their input formalisms. Instead we directly encode an unfolding of a UML system as a propositional formula. Then, the conjunction of this formula and a propositional translation of a property to be tested is checked for satisfiability using a SAT-solver. This paper builds upon our previous work on model checking of UML systems. The original contribution is twofold. The UML subset considered is extended with fork and join pseudostates along with a new type of a trigger reacting to change events. The properties tested can be now expressed in PRTECTL (parametric extension of the existential fragment of CTL). The method has been implemented as the tool BMC4UML and several experimental results are presented.

1 Introduction

We present the results of our work aiming at development of a novel symbolic verification method dedicated for systems specified in the Unified Modeling Language (UML) [1]. UML is a graphical language widely used for specification and documentation of various types of systems. In this paper we consider a subset of UML in version 2.1. The proposed verification method is a variant of SAT-based Bounded Model Checking [2] that avoids an intermediate translation and operates directly on UML. All the possible executions of a system (unfolded to a given depth) are encoded into a boolean propositional formula. Similarly, the tested property, expressed as a (parametrized) temporal formula, is encoded. Then, the conjunction of these two formulae is checked for satisfiability using a SAT-solver.

The current paper is based upon the papers [3, 4], which introduce the UML subset considered, its syntax, operational and symbolic semantics, and methods of checking (parametric) reachability for UML systems. The main contribution of this paper consists in an extension of the above to verification of properties expressed as PRTECTL (parametric extension of the existential fragment of

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

CTL) [5] formulae. Moreover, we extend the UML subset considered with several new elements: triggers specified by *when* keyword and reacting for *change events*, fork and join pseudostates, and finally associations and links in class and object diagrams used to specify object variables.

According to our best knowledge there are no other papers on parametric verification of UML systems. However, contrary to our papers, most of the approaches make use of the existing model checkers, e.g., [6, 7] translate UML to Promela and then make use of the model checker Spin. Others [8, 9] exploit timed automata as an intermediate formalism and use UPPAAL for verification. The another group of tools [10–12] apply the symbolic model checkers SMV or NuSMV via translating UML to their input languages.

The rest of the paper is organised as follows. The next section recalls the subset of UML under consideration and sketches its semantics. In Section 3 the logic PRTECTL is introduced, while in Section 4 the details of our translation to SAT are given. In the next sections, we present some experimental results, followed by the conclusions.

2 Specification Language - a Subset of UML

In this section we present the subset of UML considered in the paper and accepted by our tool. Moreover, the semantics of this subset is discussed and references to its more detailed description are given. We assume that the reader is familiar with the basic UML state machine concepts.

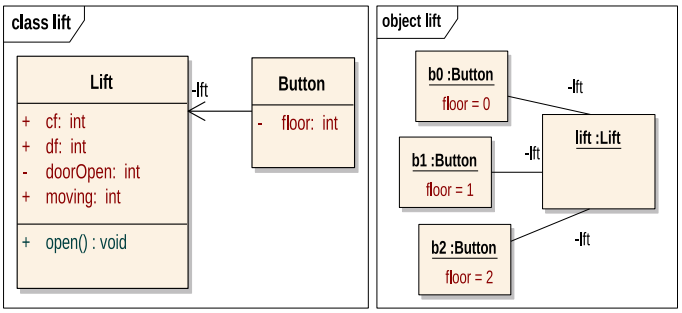

Fig. 1. Class diagram (left) and object diagram (right) of the simple lift system

The syntax is illustrated with the diagrams of a simple lift system, which is also used as a benchmark in Section 5. The systems considered are specified by a single class diagram which defines k classes (e.g. see Fig. 1), a single object diagram which defines n objects (e.g. in Fig. 1), and k state machine diagrams (e.g. in Fig. 2, 3), each one assigned to a different class of the class diagram. The class diagram defines a list of attributes (of integer and object types) 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 (instances of attributes). The object attributes are specified by

directed associations in the class diagram, while the links in the object diagram are used for initialization of the instances.

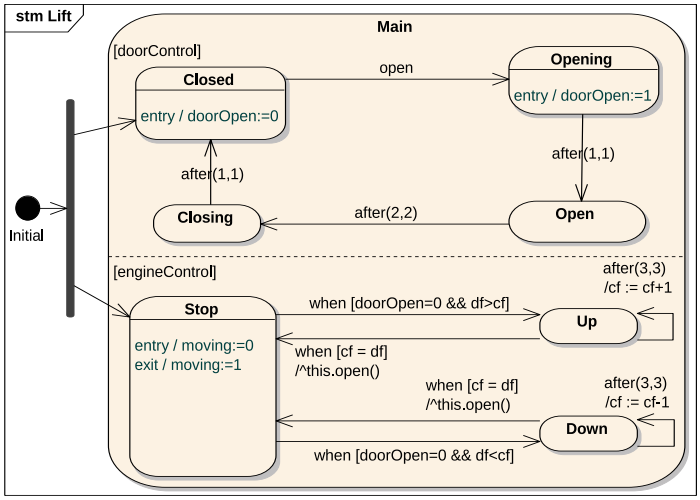

Fig. 2. State machine of class Lift

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. *Stop* in Fig. 2), composite states, (e.g. *Main* in Fig. 2), final states, initial pseudo-states, (e.g. *Initial* in Fig. 2), and fork/join pseudo-states (e.g. vertical bar in Fig. 2). The areas filling the composite states are called *regions*. The regions contained in the same composite state are *orthogonal* (e.g. *DoorControl* and *EngineControl* in Fig. 2). The regions contain states and transitions, and thus introduce a *hierarchy* of state machines. We assume that a definition of the hierarchy relation is given, and we implicitly refer to this relation by using the terms ancestor and descendant.

The labels of transitions are expressions of the form $trigger[guard]/action$, where each of these components can be empty. 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 four types: an *operation call*, a *completion event*, a *time event*, or a *change event*. In general, firing of a transition causes deactivation and activation of some states (depending on the type of the transition and the hierarchy of given state machine). We say that the *state machine configuration* changes then. More details can be found in [3, 4].

A time event, defined by an expression of the form $after(\delta_1, \delta_2)$, where $\delta_1, \delta_2 \in \mathbb{N}$ and $\delta_1 \leq \delta_2$, can occur not earlier than after passing of δ_1 time units and no later than before passing of δ_2 time units. This is the extension of the standard $after(x)$ expression, which allows one to specify an interval of time in which a transition is enabled. However, we follow the discrete-time semantics

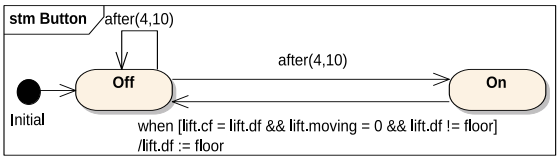

Fig. 3. State machine of class Button

where the clock valuations are natural numbers. The time flow is measured from entering the *time state*, which is the source state of a transition with the trigger of the form $after(\delta_1, \delta_2)$.

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 event from the head of the queue possibly fires a transition, and is consumed. If it cannot fire any transition and the matching trigger is *deferred* in the current state, then the event is deferred, i.e., it will be consumed later. Otherwise, the event is discarded.

The transitions with non-empty trigger are called *triggered transitions*. We refer to the processing of a single non-completion event as the *Run-To-Completion (RTC) step*. Next, a (non-completion) event can be handled only if the previous one has been fully processed, together with all the completion events which eventually have occurred. A completion event (denoted by κ) occurs for a state that has completed all of its internal activities. The completion events fire the *completion transitions*, i.e., transitions without a trigger defined explicitly. The completion transitions have priority over the triggered transitions.

The execution of the whole system follows the interleaving semantics, similar to [9]. 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. The time flow possibly causes occurrences of time events, which are processed in the next RTC steps.

The semantics has been formalized in terms of labelled transition system [3, 4]. There are six types of transitions with the following priorities:
1. Consumption of the completion events. Removes all the completion events that cannot fire a completion transition for the i-th object.
2. Execution of a completion transition. Handles one completion event causing a firing of one completion transition, and changes the valuation according to the sequence of actions: exit actions and deactivation of leaving states, the transition action, the entry actions and activation of the entered states, and producing completion events for some of the activated states. Moreover the clocks of the entered timed states are reset.
3. Execution of triggered transitions. If a transition is triggered by an event from the queue, then it is additionally consumed. The second possibility is the firing of a timed transition triggered by a time event. In this case the enabling condition depends rather on the clock valuation than the queue contents. In the case of *when*-transitions triggered by change events, the enabling condition

depends on variables valuation. We deal with changes of the valuation in a way similar to the transitions of type 2.
4. Deferring an event. An event is deferred, i.e., it cannot be dispatched while staying at the current state, but it will be considered again after leaving this state.
5. Discarding of an event. Discards an event from the head of the i-th event queue, when it does not enable any transition.
6. Time flow. If all the event queues are empty, all completion events have been processed, and none of the *when* transition is enabled, then x time units pass. Note that $0 < x \leq Exp$, where by Exp we denote the earliest expiration time of the considered time events.

The extensions of UML subset. The main extensions of the UML subset considered concern state machines. Now we are taking into account also the fork and join pseudostates as well as the *when* triggers reacting to change events.

The transitions passing through the join or fork pseudostates are restricted to completion transitions only and treated in a special way. They are combined into one transition with a number of source (or target) states, in the case of join (or fork, respectively) pseudostates. Due to the interleaving semantics and the higher priority of completion events consumption, the join transitions are not enabled by completion events, but they are enabled when all source states are active.

The transitions with *when* triggers react to change events. A change event occurs when the corresponding boolean expression is evaluated to *true*, e.g., for the expression $x > 0$ the change event occurs when the variable x is assigned a positive value. The syntax of a standard change event is as follows: *when* ⟨*boolean expression*⟩. For technical reasons we have simplified it by moving the boolean expression to the guard.

It is important to mention that the above extensions simplify modeling of the verified systems, but at the price of a bit more complicated semantics. On the other hand, the use of fork pseudostates reduces the number of steps needed to activate all the state machines nested in the orthogonal regions, and the use of the *when* triggers is often a convenient way for modeling reactions for changes in the system as well as avoiding unnecessary operation calls.

3 Parametrized Temporal Logic PRTECTL

In this section we recall two extensions of Computation Tree Logic (CTL) [13], introduced in [5], namely PRTCTL and vRTCTL. PRTCTL sentences are built of expressions containing CTL-like modalities with additional parametric superscripts expressing the time length of paths. The universal and existential quantifiers over forementioned parameters are also allowed in PRTCTL. The vRTCTL logic can be perceived as an intermediate logic in building PRTCTL sentences. The formulae of vRTCTL do not contain quantifiers and are evaluated under the accompanied parameter valuations.

3.1 Syntax

Let $\Theta_1, \ldots, \Theta_n$ be variables, called here *parameters*. An expression of the form $\eta = \sum_{i=1}^{n} c_i \cdot \Theta_i + c_0$, where $c_0, \ldots, c_n \in \mathbb{N}$, is called a *linear expression*. A function $\upsilon : \{\Theta_1, \Theta_2, \ldots, \Theta_n\} \longrightarrow \mathbb{N}$ is called a *parameter valuation*. Let Υ be the set of all the parameter valuations.

Definition 1. *Let $\mathcal{PV}$ be a set of propositional variables containing the symbol* true*. Define inductively the formulae of* vRTCTL :

1. *every member of $\mathcal{PV}$ is a formula,*
2. *if α and β are formulae, then so are $\neg\alpha$, $\alpha \wedge \beta$ and $\alpha \vee \beta$,*
3. *if α and β are formulae, then so are $EX\alpha$, $EG\alpha$, and $E\alpha U\beta$,*
4. *if η is a linear expression, α and β are formulae of* vRTCTL*, then so are $EG^{\leq\eta}\alpha$ and $E\alpha U^{\leq\eta}\beta$.*

The conditions $1, 2$, and 3 alone define CTL. Notice that η is allowed to be a constant. The logic defined by a modification of the above definition, where $\eta = a$ for $a \in \mathbb{N}$, is called RTCTL in [5].

Definition 2. *The formulae of* PRTCTL *are defined as follows:*

1. *if $\alpha \in$* vRTCTL*, then $\alpha \in$* PRTCTL*,*
2. *if $\alpha(\Theta) \in$* PRTCTL*, where Θ is a free parameter, then $\forall_\Theta\alpha(\Theta), \exists_\Theta\alpha(\Theta), \forall_{\Theta\leq a}\alpha(\Theta), \exists_{\Theta\leq a}\alpha(\Theta) \in$* PRTCTL*, for $a \in \mathbb{N}$.*

The formulae of PRTCTL are built from formulae of vRTCTL by preceding them with additional existential or universal quantifiers, which may be restricted or unrestricted. As an example consider the vRTCTL formula $\alpha(x) = EF^{\leq x}p$, where $p \in \mathcal{PV}$. Then, by preceding $\alpha(x)$ with the unrestricted quantifiers, we get $\forall_x EF^{\leq x}p$ and $\exists_x EF^{\leq x}p$ - formulas of PRTCTL.

3.2 Bounded semantics

We interpret the formulae, following Emerson's and Trefler's approach [5] in timed Kripke structures - i.e., standard Kripke structures having the transitions labeled with natural numbers. The value of a label shows how much time it takes to traverse the transition.

Definition 3. *Let $\mathcal{PV}$ be a set of propositional variables containing the symbol* true*. A timed Kripke structure (a* model*) is a tuple $(S, s^0, \rightarrow, \mathcal{L})$, where S is a finite set of* states*, $s^0 \in S$ is an* initial state*, $\rightarrow \subseteq S \times \mathbb{N} \times S$ is a transition relation such that for every $s \in S$ there exists $s' \in S$ and $n \in \mathbb{N}$ with $(s, n, s') \in \rightarrow$ (i.e., the relation is total), and $\mathcal{L} : S \longrightarrow 2^{\mathcal{PV}}$ is a labelling function satisfying true $\in \mathcal{L}(s)$ for each $s \in S$.*

As we are interested in BMC methods, we consider the existential subsets of the logics, where the negation can be applied to propositions only, i.e., PRTECTL and vRTECTL. To verify the properties expressed in the aforementioned logics we unfold the computation tree of a given model to a limited depth and check the validity of the property in question along such a finite structure.

Definition 4. *Let M be a model, and $k \in \mathbb{N}$. By $Path_k$ let us denote the set of all the sequences $(s_0, n_0, s_1, n_1, \ldots, s_k)$, where s_i is a state and $s_i \xrightarrow{n_i} s_{i+1}$ for all $0 \leq i < k$. The pair $(Path_k, \mathcal{L})$ is called the* k-model *of M and denoted by M_k. An element $\pi_k \in Path_k$ is called a* k-path.

Let $\pi_k(i)$ denote the i-th state present on a k-path π_k.

Definition 5. *Let M_k be the k-model of M and $\pi_k \in Path_k$. We define the function loop : $Path_k \longrightarrow 2^{\mathbb{N}}$ as $loop(\pi_k) = \{l \mid l \leq k \wedge \exists_{m \in \mathbb{N}}(\pi_k(k) \xrightarrow{m} \pi_k(l))\}$.*

A k-path π_k is called a *loop* if $loop(\pi_k) \neq \emptyset$. Observe that loops are essentially a way of representing infinite paths in a finite way. We define the *time distance* between positions $\pi_k(0)$ and $\pi_k(j)$ as $\delta^j_{\pi_k} := \sum_{i=0}^{j-1} n_i$, assume that $\delta^0_{\pi_k} = 0$. We abbreviate $\delta^k_{\pi_k}$ as δ_{π_k}. For a k-path π_k and $c \in \mathbb{N}$ we define $\Delta^c_{\pi_k} = max\{i \mid i \leq k \wedge \delta^i_{\pi_k} \leq c\}$ - the maximal index i of π_k s.t. $\delta^i_{\pi_k} \leq c$. For a parameter valuation υ and a linear expression η, by $\upsilon(\eta)$ we mean the evaluation of η under υ.

The number of the states of M is called the size of M and denoted by $|M|$. Let σ_{max} be the greatest time label of the transitions present in M. It follows from Lemma 1 and Theorem 1 of [14], being respectively versions of Proposition 4 and Theorem 1 of [5] that each unbounded quantifier can be replaced, without changing the validity of a formula in a fixed model M with a version bounded with the $\sigma_{max}|M|$ value. Similarly we can argue that non-superscripted modalities can be replaced by versions with $\leq \sigma_{max}|M|$ superscript. Therefore, from now on we omit the unbounded quantifiers and non-superscripted EU and EG in our considerations.

Definition 6 (Bounded semantics for vRTECTL**).** *Let $M_k = (Path_k, \mathcal{L})$ be the k-model of M, s – a state, $p \in \mathcal{PV}$, $\alpha, \beta \in$ vRTECTL and υ – a parameter valuation. By $M_k, s \models_\upsilon \alpha$ we denote that α holds in M_k under valuation υ. We omit M_k where it is implicitly understood. We define the relation $\models_\upsilon$ as follows:*

1. $s \models_\upsilon p$ *iff* $p \in \mathcal{L}(s)$,
2. $s \models_\upsilon \neg p$ *iff* $p \notin \mathcal{L}(s)$,
3. $s \models_\upsilon \alpha \vee \beta$ *iff* $s \models_\upsilon \alpha$ *or* $s \models_\upsilon \beta$,
4. $s \models_\upsilon \alpha \wedge \beta$ *iff* $s \models_\upsilon \alpha$ *and* $s \models_\upsilon \beta$,
5. $s \models_\upsilon EX\alpha$ *iff* $\exists_{\pi_k \in Path_k}\big(\pi_k(0) = s \wedge \pi_k(1) \models_\upsilon \alpha\big)$,
6. $s \models_\upsilon EG^{\leq \eta}\alpha$ *iff* $\exists_{\pi_k \in Path_k}\big[\pi_k(0) = s \wedge \big((\delta_{\pi_k} > \upsilon(\eta) \wedge \forall_{i \leq \Delta^{\upsilon(\eta)}_{\pi_k}} \pi_k(i) \models_\upsilon \alpha) \vee (\delta_{\pi_k} \leq \upsilon(\eta) \wedge \forall_{i \leq k} \pi_k(i) \models_\upsilon \alpha \wedge loop(\pi_k) \neq \emptyset)\big)\big]$,
7. $s \models_\upsilon E\alpha U^{\leq \eta}\beta$ *iff* $\exists_{\pi_k \in Path_k}\big(\pi_k(0) = s \wedge \exists_{i \leq \Delta^{\upsilon(\eta)}_{\pi_k}}(\pi_k(i) \models_\upsilon \beta \wedge \forall_{j<i} \pi_k(j) \models_\upsilon \alpha)\big)$.

Definition 7 (Bounded semantics for PRTECTL**).** *Let M_k be the k-model of M, σ_{max} – the greatest time label of the transitions present in M, s – a state, α – a sentence (a formula without free variables) of* PRTECTL *and $a \in \mathbb{N}$. We recursively define the relation $\models$ as follows:*

1. $M_k, s \models \forall_{\Theta \leq a}\alpha(\Theta)$ *iff* $\forall_{0 \leq i_\Theta \leq min\{a, k \cdot \sigma_{max}\}} M_k, s \models \alpha(\Theta \leftarrow i_\Theta)$,
2. $M_k, s \models \exists_{\Theta \leq a}\alpha(\Theta)$ *iff* $\exists_{0 \leq i_\Theta \leq min\{a, k \cdot \sigma_{max}\}} M_k, s \models \alpha(\Theta \leftarrow i_\Theta)$,

where i_Θ is a fresh integer variable.

4 Bounded Model Checking

Having already presented the bounded semantics, we aim at applying the bounded model checking method to verification of the properties formulated in vRTECTL, and PRTECTL. The BMC method is based on the idea of a translation of a part of the model together with a property in question to a propositional formula. Satisfiability of the result means that the translated formula holds in the model.

4.1 Submodels

In order to obtain an acceptable efficiency, the (parametric) temporal formula should be checked over a number of symbolic paths, which as small as possible, but sufficient to determine its validity. Hence the BMC algorithm works on submodels of the k-model.

Definition 8. *Let $M_k = (Path_k, \mathcal{L})$ be the k-model. A substructure $M'_k = (Path'_k, \mathcal{L}')$, where $Path'_k \subseteq Path_k$ and $\mathcal{L}'$ is the restriction of $\mathcal{L}$ to the states present in the paths of $Path'_k$, is called a* submodel *of M_k. The* size *of M'_k is equal to $|Path'_k|$.*

The bounded semantics of vRTECTL formulae and PRTECTL sentences over submodels is defined as for k-models.

It was proven in [15] that in order to determine the truth of an ECTL formula in M_k it is sufficient to consider only submodels of size restricted by the (special) function f_k on the checked formula. We recall the following definition from [14], which extends these results to PRTECTL.

Definition 9. *Let $\alpha, \beta \in$ PRTECTL, p be an atomic proposition, and η - a linear expression. Define recursively the special function f_k : PRTECTL $\longrightarrow \mathbb{N}$ as follows:*

1. $f_k(p) = f_k(\neg p) = 0$,
2. $f_k(\alpha \vee \beta) = max(f_k(\alpha), f_k(\beta))$,
3. $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
4. $f_k(EX\alpha) = f_k(\alpha) + 1$,
5. $f_k(EG^{\leq \eta}\alpha) = (k+1) \cdot f_k(\alpha) + 1$,
6. $f_k(E\alpha U^{\leq \eta}\beta) = k \cdot f_k(\alpha) + f_k(\beta) + 1$,
7. $f_k(\forall_{\Theta \leq c}\beta(\Theta)) = (c+1) \cdot f_k(\beta(\Theta))$,
8. $f_k(\exists_{\Theta \leq c}\beta(\Theta)) = f_k(\beta(\Theta))$.

4.2 Translation to SAT

In order to translate the problem of validity of a sentence $\alpha \in$ PRTECTL in the submodel M'_k to the problem of satisfiability of a propositional formula $\left[\alpha\right]_k$ we have to encode M'_k and α, and then combine the results together. We present an adapted version of the efficient translation introduced in [16].

Consider the model M. As the number of the states of M is finite, they can be perceived as a bit vectors of the length[3] $r = \lceil log_2|M| \rceil$. Therefore, we can

[3] The $r = \lceil log_2|M| \rceil$ value is given here only as an example. It can be different for various systems representations.

represent the states as the valuations of the vector $w = (w_1, \ldots, w_r)$. Due to the fact that we need to count the time passed along a path, we augment the representation of states with an additional bit vector $d = (d_1, \ldots, d_z)$ of the length $z = \lceil \log_2(k \cdot \sigma_{max} + 1) \rceil$ where σ_{max} denotes, as previously, the maximal value of transition label present in M. The vector $\mathbf{w} = (w_1, \ldots, w_r, d_1, \ldots, d_z)$ is called a *global state variable*, while each its member is called a *state variable*. Moreover, by $w_{\mathbf{w}}$ and $d_{\mathbf{w}}$ we denote the subvectors w and d of the vector $\mathbf{w}$. Denote by $\mathcal{SV}$ the set of the state variables, and by V a valuation $V : \mathcal{SV} \longrightarrow \{0, 1\}$. Additionally, extend the valuation V to the vectors of the state variables: $V : \mathcal{SV}^m \longrightarrow \{0, 1\}^m$ as $V(w_1, \ldots, w_m) = \big(V(w_1), \ldots, V(w_m)\big)$. This naturally extends to the valuations $\hat{V} : \mathcal{SV}^r \longrightarrow \{0, 1\}^r$ and $\hat{D} : \mathcal{SV}^z \longrightarrow \{0, 1\}^z$ in such a way that $\hat{V}(w_1, \ldots, w_r) = (V(w_1), \ldots, V(w_r))$ and $\hat{D}(d_1, \ldots, d_z) = (V(d_1), \ldots, V(d_z))$. With a slight notational abuse, we denote by $\hat{V}(\mathbf{w})$ the state encoded by $w_{\mathbf{w}}$, and by $\hat{D}(\mathbf{w})$ - the value of time encoded by $d_{\mathbf{w}}$. The *symbolic k-path* is a vector of global state variables. As we need a number of symbolic k-paths to represent the k-paths in a translated submodel, by $(\mathbf{w}_{0,i}, \mathbf{w}_{1,i}, \ldots, \mathbf{w}_{k,i})$ we denote the i-th symbolic k-path, where $\mathbf{w}_{j,i}$ is a global state variable.

Let $\mathbf{w}, \mathbf{w}'$ be global state variables, s a state and p a proposition. In the rules of the translation the following propositional formulae are used:

1. $p(\mathbf{w})$ denotes a formula such that $V \models p(\mathbf{w})$ iff $p \in \mathcal{L}(\hat{V}(\mathbf{w}))$,
2. $T(\mathbf{w}, \mathbf{w}')$ denotes a formula such that $V \models T(\mathbf{w}, \mathbf{w}')$ iff $\hat{V}(\mathbf{w}) \rightarrow \hat{V}(\mathbf{w}')$ (i.e., there exists a transition between $\hat{V}(\mathbf{w})$ and $\hat{V}(\mathbf{w}')$ in the model M, but the values of $\hat{D}(\mathbf{w})$, $\hat{D}(\mathbf{w}')$ are not taken into account),
3. $Td(\mathbf{w}, \mathbf{w}')$ denotes a formula such that $V \models Td(\mathbf{w}, \mathbf{w}')$ iff $\hat{V}(\mathbf{w}) \xrightarrow{n} \hat{V}(\mathbf{w}')$ for some $n \in \mathbb{N}$, and $\hat{D}(\mathbf{w}') = \hat{D}(\mathbf{w}) + n$ (i.e., there exists a transition between $\hat{V}(\mathbf{w})$ and $\hat{V}(\mathbf{w}')$ in the model M, and the difference between $\hat{D}(\mathbf{w}')$ and $\hat{D}(\mathbf{w})$ corresponds to the time passed while performing this transition),
4. $H(\mathbf{w}, \mathbf{w}')$ is a formula s.t. $V \models H(\mathbf{w}, \mathbf{w}')$ iff $\hat{V}(\mathbf{w}) = \hat{V}(\mathbf{w}')$ (encodes equality of states),
5. $L_k(j) = \bigvee_{i=0}^{k} T(\mathbf{w}_{k,j}, \mathbf{w}_{i,j})$ encodes a loop, i.e.,
   $V \models L_k(j)$ iff $loop((\hat{V}(\mathbf{w}_{0,j}), .., \hat{V}(\mathbf{w}_{k,j}))) \neq \emptyset$,
6. $I_{s_0}(\mathbf{w})$ is a formula s.t. $V \models I_{s_0}(\mathbf{w})$ iff $\hat{V}(\mathbf{w}) = s$ and $\hat{D}(\mathbf{w}) = 0$ (encodes the initial state),
7. $Z(\mathbf{w})$ is a formula s.t. $V \models Z(\mathbf{w})$ iff $\hat{D}(\mathbf{w}) = 0$ ($\mathbf{w}$ encodes the first state of a path),
8. $Le(\mathbf{w}, a)$, for $a \in \mathbb{N}$, is a formula such that $V \models Le(\mathbf{w}, a)$ iff $\hat{D}(\mathbf{w}) \leq a$.

Let M be a model and A be a finite subset of $\mathbb{N}$. The *unfolding of the transition relation* is defined as

$$\big[M\big]_k^A := \bigwedge_{j \in A} \bigwedge_{i=0}^{k-1} Td(\mathbf{w}_{i,j}, \mathbf{w}_{i+1,j}).$$

It is easy to see that $V \models \big[M\big]_k^A$ iff for each $j \in A$, $(\hat{V}(\mathbf{w}_{0,j}), \ldots, \hat{V}(\mathbf{w}_{k,j}))$ is a k-path in M. As the translation introduced in [16] was an essential improvement

over the original one of [15], we follow Zbrzezny's approach in our work. The improvement primarily consists in translating every subformula ψ of the formula α by using only $f_k(\psi)$ k-paths. We define the functions used to split the set of k-paths of submodel M'_k into parts sufficient to translate the subformulas of the considered temporal formula.

Following [16], let A and B be finite subsets of $\mathbb{N}$. By $A \prec B$ we denote that $x < y$ for all $x \in A$ and $y \in B$. Let $k, m, p \in \mathbb{N}$ and $m \leq |A|$, then:

1. $\hat{g}_L(A, m)$ is the subset B of A such that $|B| = m$ and $B \prec A \backslash B$,
2. $\hat{g}_R(A, m)$ denotes the subset B of A such that $|B| = m$ and $A \backslash B \prec B$,
3. $h_X(A)$ is the set $A \backslash \{min(A)\}$,
4. if $k+1$ divides $|A|-1$ then $h_G(A, k)$ is the sequence of sets $(B_0, \ldots, B_k)$ such that $\bigcup_{i=0}^{k} B_i = A \backslash \{min(A)\}$, $|B_i| = |B_j|$ and $B_i \prec B_j$ for every $0 \leq i < j \leq k$,
5. if k divides $|A|-1-p$, then $h_U(A, k, p)$ denotes the sequence of sets $(B_0, \ldots, B_k)$ such that $\bigcup_{i=0}^{k} B_i = A \backslash \{min(A)\}$, $B_i \prec B_j$ for every $0 \leq i < j \leq k$, $|B_0| = \ldots = |B_{k-1}|$ and $|B_k| = p$.

We also need a sequence element selector that is if $h_G(A, k) = (B_0, \ldots, B_k)$ then define $h_G(A, k)(i) = B_i$ for $0 \leq i \leq k$ and if $h_U(A, k, p) = (B_0, \ldots, B_k)$, define $h_U(A, k, p)(i) = B_i$ for $0 \leq i \leq k$. The functions $\hat{g}_L$ and $\hat{g}_R$ are used to divide the set of path indices into the two parts of the sizes sufficient to perform the independent translation of subformulas α and β of formula $\alpha \wedge \beta$. Similarly, h_G and h_U are used to divide the set of path indices into the sequences (hence the use of the selector) of subsets which are of the sizes sufficient to perform the translation of subformulas α and α together with β of, respectively, formulae $EG^{\leq \eta}\alpha$ and $E\alpha U^{\eta}\beta$. A more in-depth description can be found in [16].

Definition 10 (Translation of vRTECTL**).** *Let $\alpha, \beta \in$ vRTECTL, p – an atomic proposition, υ – a parameter valuation, η – a linear expression, $(m, n) \in \mathbb{N} \times \mathbb{N}$, and $A \subseteq \mathbb{N}$.*

- $\big[p\big]_k^{[m,n,A,\upsilon]} := p(\mathbf{w}_{m,n})$ *and* $\big[\neg p\big]_k^{[m,n,A,\upsilon]} := \neg p(\mathbf{w}_{m,n})$,
- $\big[\alpha \wedge \beta\big]_k^{[m,n,A,\upsilon]} := \big[\alpha\big]_k^{[m,n,\hat{g}_L(A,f_k(\alpha,\upsilon)),\upsilon]} \wedge \big[\beta\big]_k^{[m,n,\hat{g}_R(A,f_k(\beta,\upsilon)),\upsilon]}$,
- $\big[\alpha \vee \beta\big]_k^{[m,n,A,\upsilon]} := \big[\alpha\big]_k^{[m,n,\hat{g}_L(A,f_k(\alpha,\upsilon)),\upsilon]} \vee \big[\beta\big]_k^{[m,n,\hat{g}_L(A,f_k(\beta,\upsilon)),\upsilon]}$,
- $\big[EX\alpha\big]_k^{[m,n,A,\upsilon]} := H(\mathbf{w}_{m,n}, \mathbf{w}_{0,min(A)}) \wedge \big[\alpha\big]_k^{[1,min(A),h_X(A),\upsilon]}$,
- $\big[EG^{\leq \eta}\alpha\big]_k^{[m,n,A,\upsilon]} := H(\mathbf{w}_{m,n}, \mathbf{w}_{0,min(A)}) \wedge Z(\mathbf{w}_{0,min(A)}) \wedge$
  $\big(\big(Le(\mathbf{w}_{k,min(A)}, \upsilon(\eta)) \wedge (L_k(min(A)) \wedge \bigwedge_{j=0}^{k} \big[\alpha\big]_k^{[j,min(A),h_G(A,k)(j),\upsilon]} \big) \big)$
  $\vee \big(\neg Le(\mathbf{w}_{k,min(A)}, \upsilon(\eta)) \wedge$
  $\bigwedge_{j=0}^{k} (Le(\mathbf{w}_{j,min(A)}, \upsilon(\eta)) \Rightarrow \big[\alpha\big]_k^{[j,min(A),h_G(A,k)(j),\upsilon]}) \big) \big)$,
- $\big[E\alpha U^{\leq \eta}\beta\big]_k^{[m,n,A,\upsilon]} := H(\mathbf{w}_{m,n}, \mathbf{w}_{0,min(A)}) \wedge Z(\mathbf{w}_{0,min(A)}) \wedge$
  $\bigvee_{i=0}^{k} \big(Le(\mathbf{w}_{i,min(A)}, \upsilon(\eta)) \wedge (\big[\beta\big]_k^{[i,min(A),h_U(A,k,f_k(\beta,\upsilon))(k),\upsilon]} \wedge$
  $\bigwedge_{j=0}^{i-1} \big[\alpha\big]_k^{[j,min(A),h_U(A,k,f_k(\beta,\upsilon))(j),\upsilon]}) \big)$.

The above encoding is based on the bounded semantics for vRTECTL (see Definition 6). Notice that in case of the encoding of $EG^{\leq\eta}\alpha$ formula, we need to consider two cases. The first case deals with the situation when α should be checked along a k-path of length strictly greater than k, because the time elapsed at this k-path is lower than η. Therefore, we have to check α along the loop – hence we encode the loop condition using the propositional formula L_k. In the second case, when the time elapsed at the k-path is greater or equal η, α is checked along a path of length smaller or equal than the depth k of the unfolding of the model. Each such a finite path is then a prefix of some k-path. Both the cases are combined in the disjunction.

Definition 11 (Translation of PRTECTL**).** *Let* $\alpha \in$ PRTECTL, $A \subseteq \mathbb{N}$, $(m,n) \in \mathbb{N} \times \mathbb{N}$, *and* $c \in \mathbb{N}$. *If* α *contains no quantifiers and no free parameters, then:*

$$\big[\alpha\big]_k^{[m,n,A]} := \big[\alpha\big]_k^{[m,n,A,\upsilon]}, \text{ where } \upsilon \text{ is any parameter valuation.}$$

As in the above case $\alpha \in$ vRTECTL *and it contains no free parameters, the choice of* υ *is irrelevant.*

Let $d = min\{c, k \cdot \sigma_{max}\}$, *then:*

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

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

Let M_k be the k-model. If $\alpha \in$ PRTECTL, define $F_k(\alpha) := \{i \in \mathbb{N} \mid 1 \leq i \leq f_k(\alpha)\}$. The set F_k contains the indices of symbolic k-paths used to perform the translation. The formula $\big[M\big]_k^{F_k(\alpha)}$ encodes all the M_k submodels of the size not greater than needed to validate the truth of formula α, as indicated in Lemmas 4, 5 from [14].

Now we are ready to complete the translation of the problem of validity in vRTECTL and PRTECTL to the problem of satisfiability of propositional formulae. Let M_k be the k-model, $\alpha \in$ vRTECTL and υ be a parameter valuation. Denote

$$\big[M\big]_k^{\alpha,\upsilon} := \big[M\big]_k^{F_k(\alpha)} \wedge I_{s^0}(\mathbf{w}_{0,0}) \wedge \big[\alpha\big]_k^{[0,0,F_k(\alpha),\upsilon]}.$$

Similarly, let $\beta \in$ PRTECTL, then denote

$$\big[M\big]_k^{\beta} := \big[M\big]_k^{F_k(\beta)} \wedge I_{s^0}(\mathbf{w}_{0,0}) \wedge \big[\beta\big]_k^{[0,0,F_k(\beta)]}.$$

Theorems 2 and 3 from [14] ensure completeness and correctness of the translation.

4.3 Implementation for UML

In this section we present some details of a propositional encoding of systems specified in UML. In particular we show the symbolic encoding of the states and the transition relation, as well as some of the functions needed to translate PRTECTL formulae to SAT.

Given a UML system, as described in Sect. 2. Let CL be the set of all classes in the system, and let $Obj(c)$ be the set of all instances of a given class $c \in \mathrm{CL}$. By n denote the number of all objects in the system, by $\mathcal{S}_i$ – the set of all states of the i-th object (all states from the state machine instance assigned to the object), by $\mathcal{CS}_i \subseteq \mathcal{S}_i$ – the set of *completion sensitive* states of the i-th object (i.e., source states of completion transitions), by $\Gamma_i \subseteq \mathcal{S}_i$ – the set of time states, by $\mathcal{V}_i$ – the set of variables, and by q_i the event queue of the i-th object, where $i \in \{1, \ldots, n\}$. Let σ_{max} denotes the greatest upper bound of all time triggers present in the system, and let $Reg(\Gamma_i)$ denotes the set of state machine regions which are the direct ancestors of the time states from Γ_i.

A global state of a given system is a tuple of states of the objects. A single object state consists of a set of active states, a set of completed states (the states for which completion events have recently occurred), a contents of the event queue, a valuation of the variables, and a valuation of the clocks. A single variable and a single clock can be represented respectively by $int_{size} = \lceil \log_2(max_{int} + 1) \rceil + 1$[4] and $clock_{size} = \lceil \log_2(\sigma_{max} + 2) \rceil$ state variables. An event queue is represented by m-element cyclic buffer and three indices pointing respectively to the event to be processed next, the first unoccupied position in the queue (where it will be placed the next event coming), and the first deferred event. Thus an event queue can be represented by $size(q_i) = m \cdot b(i) + 3 \cdot \lceil \log_2 m + 1 \rceil$ state variables, where $b(i)$ denotes the number of variables needed to encode a single operation call. The global state of the system can be encoded by valuations of a vector of state variables $w_g = (w_g[1], \ldots, w_g[r])$, where:

$$r = \sum_{i=1}^{n} \Big(|\mathcal{S}_i| + |\mathcal{CS}_i| + size(q_i) + |\mathcal{V}_i| \cdot int_{size} + |Reg(\Gamma_i)| \cdot clock_{size} \Big).$$

Due to the fact that we need to count the time passed along a path, we augment the representation of states with an additional state variable vector $w_d = (w_d[1], \ldots, w_d[z])$ of the length $z = \lceil \log_2(k \cdot \sigma_{max} + 1) \rceil$, called a *global clock*. From now on the symbolic state $\mathbf{w}$ is a pair (w_g, w_d). The particular state variables are denoted by $\mathbf{w}[1], \ldots, \mathbf{w}[r]$ (the global state part) and $\mathbf{w}[r+1], \ldots, \mathbf{w}[r+z]$ (the global clock part).

Next, we give the encoding of the transition relation. We start with a set of helper formulae that encode enabling conditions and execution of transitions of types 1 - 6, described in Sect. 2. We define propositional formulae for transitions of types $1 \leq j \leq 5$ that encode their preconditions over the vector $\mathbf{w}$ for the object o: $EOj(o, \mathbf{w})$. We define also the propositional formulae encoding an execution of these transitions over the vectors $\mathbf{w}, \mathbf{w}'$ for the object o: $XOj(o, \mathbf{w}, \mathbf{w}')$ for $1 \leq j \leq 5$ and the formula encoding the time flow $X6(\mathbf{w}, \mathbf{w}')$.

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

[4] In order to keep the verification problem decidable the values of the variables are bounded to $\langle -(max_{int} + 1), max_{int} \rangle$.

encoded as (we set $XO(o, \mathbf{w}, \mathbf{w}') = f_1(o, \mathbf{w}, \mathbf{w}')$):

$$\begin{aligned} f_5(o, \mathbf{w}, \mathbf{w}') &= EO5(o, \mathbf{w}) \wedge XO5(o, \mathbf{w}, \mathbf{w}') \\ f_j(o, \mathbf{w}, \mathbf{w}') &= EOj(o, \mathbf{w}) \wedge XOj(o, \mathbf{w}, \mathbf{w}') \\ &\quad \vee \neg EOj(o, \mathbf{w}) \wedge f_{j+1}(o, \mathbf{w}, \mathbf{w}') \texttt{ for } j \in [1, 4] \end{aligned}$$

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

$$XC(c, \mathbf{w}, \mathbf{w}') = \bigvee_{o \in Obj(c)} XO(o, \mathbf{w}, \mathbf{w}')$$

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

$$\mathfrak{T}(\mathbf{w}, \mathbf{w}') = \bigvee_{c \in \mathrm{CL}} XC(c, \mathbf{w}, \mathbf{w}') \vee E6(\mathbf{w}) \wedge X6(\mathbf{w}, \mathbf{w}')$$

where $E6(\mathbf{w})$ encodes the enabling conditions of the time flow transition. Some details of the above encoding were given in [3, 4] and are omitted here.

Now, we provide the explanation of the functions used to propositional translation of PRTECTL, given in Sect. 4.2:

1. $p(\mathbf{w})$ - a propositional formula encoding the property p over the state variables from $\mathbf{w}$,
2. $T(\mathbf{w}, \mathbf{w}') = \mathfrak{T}(\mathbf{w}, \mathbf{w}')$,
3. $Td(\mathbf{w}, \mathbf{w}') = \big(\bigwedge_{j=r+1}^{r+z} \mathbf{w}[j] \iff \mathbf{w}'[j]\big) \wedge \big(\bigvee_{c \in \mathrm{CL}} XC(c, \mathbf{w}, \mathbf{w}')\big) \vee E6(\mathbf{w}) \wedge X6(\mathbf{w}, \mathbf{w}') \wedge incGC(\mathbf{w}, \mathbf{w}')$,
4. $H(\mathbf{w}, \mathbf{w}') = \bigwedge_{j=1}^{r} \mathbf{w}[j] \iff \mathbf{w}'[j]$,
5. $I_{s_0}(\mathbf{w}) = \bigwedge_{i=1}^{n} \Big(\bigwedge_{s \in Init_i} \big(act(s) \wedge cpl(s)\big) \bigwedge_{s \in (\mathcal{S}_i \setminus Init_i)} \big(\neg act(s) \wedge \neg cpl(s)\big) \Big)$,
6. $Z(\mathbf{w}) = \bigwedge_{j=r+1}^{r+z} \neg \mathbf{w}[j]$,

where $act(s)$ and $cpl(s)$ are the state variables that evaluate to *true* when the state s is respectively active and completed, and $incGC(\mathbf{w}, \mathbf{w}')$ is a propositional formula encoding the increasing of the global clock by the same value as all other clocks (that is encoded by $X6(\mathbf{w}, \mathbf{w}')$). The encoding of the propositional formula $Le(\mathbf{w}, a)$, as well as encoding of integers and arithmetic operators is described in detail in [17].

Encoding of the extensions of the UML subset. In this paragraph we discuss how the extensions of the considered UML subset affect the encoding of the transition relation. To this aim we sketch the encoding of the transitions of level 2 (firing the completion transitions, including the transitions crossing the join pseudostates), and discuss the encoding of the transitions of level 3 (firing the triggered transitions, including handling of the change events).

The propositional formula $ET2(t, \mathbf{w})$ encodes the enabling condition for a completion transition t, while the formula $EO2(i, \mathbf{w})$ encodes the enabling condition for all the completion transitions of i-th object (i.e., the set T_i^κ) in the global state represented by $\mathbf{w}$:

$$ET2(t, \mathbf{w}) = \big(\neg isJoin(t) \wedge cpl(s, \mathbf{w}) \wedge grd(t, \mathbf{w})\big) \vee \big(isJoin(t) \wedge \bigwedge_{s \in src(t)} act(s, \mathbf{w})\big)$$

$$EO2(i, \mathbf{w}) = \bigvee_{t \in T_i^\kappa} ET2(t, \mathbf{w})$$

where: $isJoin(t)$ is a boolean function returning **true** only if t is a transition crossing the join state, $src(t)$ is a set of source states of the transition t (it is a singleton when $\neg isJoin(t)$ holds), $s \in src(t)$, and $grd(t, \mathbf{w})$ is a propositional formula encoding the guard of the transition t.

In order to handle the join of transitions we have to consider the two cases of completion transitions. The first one is when we deal with single-source completion transitions, and the second is the case with the transitions crossing a join pseudostate. Then both the cases are combined in the disjunction.

We handle the transitions triggered by change events in a similar way. We distinguish three cases of transitions: (i) triggered by an event from a queue, (ii) triggered by a time event, and (iii) triggered by a change event.

5 Preliminary Experimental Results

In order to estimate the efficiency of our approach we performed several experiments using our prototype implementation – the tool BMC4UML. The tests have been performed using a PC equipped with Intel Core 2 Duo (2.4 GHz) and 2.3 GB RAM running Linux OS. The SAT-solver PrecoSAT [18] has been used for checking satisfiability of the propositional formulas.

k	**vars**	**clauses**	**BMC[s]**	**BMC[MB]**	**SAT[s]**	**SAT[MB]**	**Total[s]**
1	78350	236288	1,63	14,93	0,2	29	1,83
2	162341	493491	3,48	26,02	2,8	71	6,28
3	261588	797112	5,84	39,17	5,7	91	11,54
4	388920	1188181	8,73	56,05	9	157	17,73
5	527149	1611897	12,27	74,23	12,4	259	24,67
6	683434	2091077	16,02	94,98	21,7	298	37,72
Total			47,97		51,8		**99,77**

Table 1. Experimental results for $\neg\varphi_2(5)$ - satisfiable at the depth 6.

Consider the simple lift system specified by the diagrams in Fig. 1, 2, and 3. The system consists of four objects: *lift* – an instance of the class *Lift*, and three instances of the class *Button* – *b0*, *b1*, and *b2*. The *lift* object contains three public variables: *cf* – the current floor where there is an elevator, *df* – lift's

k	vars	clauses	BMC[s]	BMC[MB]	SAT[s]	SAT[MB]	Total[s]
1	78350	236288	1,61	14,93	0,2	29	1,81
2	162341	493491	3,47	26,02	2,8	71	6,27
...	...	...	...	...	...	...	...
11	1787418	5484210	44,15	241,16	55,2	684	99,35
12	2060309	6322621	51,19	277,39	63,2	738	114,39
13	2420715	7438503	60,13	325,34	94,6	1135	154,73
Total			**318,46**		**408,9**		**727,36**

Table 2. Experimental results for $\neg\varphi_1(5)$ - satisfiable at the depth 13.

destination floor, and *moving*, where $moving = 0$ indicates that the elevator stopped. Every button is assigned a different floor number.

In general the button objects can be "pressed" in a non-deterministic way. This is modeled by timed transitions - a button does nothing (a self-loop), or moves to the state *On*. When the button is in the *On* state, the system waits for the moment when the lift stops at a destination floor different from the button's floor, and then the elevator call is accepted by assigning the *lift.df* the appropriate value.

We tested the following property: *Always when the i-th button has been pressed, the elevator always arrives to the i-th floor no later than in m time units*. This property can be expressed as the following PRTCTL formula:

$$\varphi_i(m) = \exists_{x \leq m} AG(button_i \implies AF^{\leq x} floor_i),$$

where $button_i$ and $floor_i$ are the propositional variables that are true respectively when the object *bi* is in the *On* state and when the lift is at the i-th floor ($lift.cf = i$), for $i \in \{0, 1, 2\}$. However, using BMC we can handle existential properties only, so we proceed with the negation of the formula $\varphi_i(m)$:

$$\neg\varphi_i(m) = \forall_{x \leq m} EF(button_i \wedge EG^{\leq x} \neg floor_i)$$

We started our tests for the floor 2, and $m = 5$. The formula $\neg\varphi_2(5)$ holds at the depth 6, which means that $\varphi_2(5)$ is not true in the model. The results of the verification are presented in Table 1. Checking the formula $\neg\varphi_2(5)$ was quite quick. According the lift specification it is obvious that the lift can reach the second floor not earlier than after 6 time units.

k	vars	clauses	BMC[s]	BMC[MB]	SAT[s]	SAT[MB]	Total[s]
1	78314	236180	1,63	14,93	0,2	29	1,83
2	162287	493329	3,48	26,02	2,8	71	6,28
...	...	...	...	...	...	...	...
9	1297424	3979000	31,84	176,32	33,3	584	65,14
10	1532985	4702589	38,96	207,52	39,9	633	78,86
11	1787202	5483562	43,93	241,16	56,3	684	100,23
Total			**209,92**		**224,7**		**434,62**

Table 3. Experimental results for $\neg\varphi_0(5)$ - satisfiable at the depth 11.

k	vars	clauses	BMC[s]	BMC[MB]	SAT[s]	SAT[MB]	Total[s]
1	145242	440948	3,33	23,83	2,5	67	5,83
2	319007	974073	7,69	46,78	5,6	144	13,29
3	540300	1652064	13,31	76,04	12,2	261	25,51
4	839590	2572591	21,24	115,74	20,1	330	41,34
5	1176289	3605973	30,04	160,34	29,1	560	59,14
6	1567156	4805747	40,66	212,16	48,8	638	89,46
Total			**116,27**		**118,3**		**234,57**

Table 4. Experimental results for $\neg\psi_3(5)$ - satisfiable at the depth 6.

Next, we aimed at checking the properties $\neg\varphi_1(5)$ and $\neg\varphi_0(5)$. The experimental results are given in Table 2 and 3.

In all the cases above the formula has been translated using submodels of size 12. According to the function f_k we need 2 symbolic paths for the translation of the vRTECTL subformula $EF(button_i \wedge EG^{\leq x})$, and the resulting PRTECTL formula $\neg\varphi(i)$ is the conjunction of six such subformulas, each one assigning x a different value from 0 to 5. It seems that the number of symbolic paths under consideration is the main source of the computation complexity here.

This conclusion seems to be confirmed by the next experiment, where the property involving all the floors and all the buttons has been checked: *Whenever the button is pressed the lift will always arrive at the appropriate floor in m time units*. This is expressed by the following formula:

$$\psi_n(m) = \exists_{x \leq m} \bigwedge_{i=0}^{n-1} AG(button_i \implies AF^{\leq x} floor_i),$$

where n stands for the total number of floors in the system considered.

The results of verification of the formula $\neg\psi_3(5)$ are given in Table 4, while the summary results of verification of the formula $\neg\psi_n(5)$ for different values of n are presented in Table 5. The translation required to use 12 symbolic paths.

n	k	vars	clauses	BMC[s]	BMC[MB]	SAT[s]	SAT[MB]	Total[s]
2	11	2853832	8730338	337,65	380,99	386,8	1223	724,45
3	6	1567156	4805747	116,27	212,16	118,3	638	234,57
4	7	2904428	8953986	306,87	391,38	257,7	1237	564,57
5	8	4805263	14864423	543,59	646,88	515,9	2263	1059,49

Table 5. The summary results for the formula $\neg\psi_n(5)$, n – the total number of floors, k – the depth of a counterexample found.

6 Final Remarks

In this paper we have presented preliminary results of our implementation of parametric BMC for UML. It seems that the number of symbolic paths under consideration is the main source of the computation complexity here. Whereas

the number of paths returned by $f_k(\alpha)$ is *sufficient* for checking α, it is not always *necessary*. Especially this is the case for formulas that can be equivalently expressed as (parametric) linear-time temporal logic formulas. Therefore, our plan is to bound the number of paths used for the translation, and to introduce several optimizations that should significantly improve the efficiency of our method.

References

1. OMG: Unified Modeling Language. http://www.omg.org/spec/UML/2.1.2 (2007)
2. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1) (2001) 7–34
3. Niewiadomski, A., Penczek, W., Szreter, M.: A new approach to model checking of UML state machines. Fundam. Inform. **93**(1-3) (2009) 289–303
4. Niewiadomski, A., Penczek, W., Szreter, M.: Towards checking parametric reachability for UML state machines. LNCS **5947/2010** (2010) 319–330
5. Emerson, E.A., Trefler, R.: Parametric quantitative temporal reasoning. In: Proc. of the 14th Symp. on Logic in Computer Science (LICS'99), IEEE Computer Society (July 1999) 336–343
6. Lilius, J., Paltor, I.: vUML: A tool for verifying UML models. In: ASE. (1999) 255–258
7. Jussila, T., Dubrovin, J., Junttila, T., Latvala, T., Porres, I.: Model checking dynamic and hierarchical UML state machines. In: MoDeV^{2}a. (2006) 94–110
8. Knapp, A., Merz, S., Rauh, C.: Model checking - timed UML state machines and collaborations. In: FTRTFT. (2002) 395–416
9. Diethers, K., Goltz, U., Huhn, M.: Model checking UML statecharts with time. In: Proc. of the UML'02 workshop, TU München (2002) 35–52
10. Compton, K., Gurevich, Y., Huggins, J., Shen, W.: An automatic verification tool for UML. Technical Report CSE-TR-423-00, University of Michigan (2000)
11. Gutiérrez, M.E.B., Barrio-Solórzano, M., Quintero, C.E.C., de la Fuente, P.: UML automatic verification tool with formal methods. Electr. Notes Theor. Comput. Sci. **127**(4) (2005) 3–16
12. Dubrovin, J., Junttila, T.: Symbolic model checking of hierarchical UML state machines. Technical Report B23, HUT TCS, Espoo, Finland (December 2007)
13. Clarke, J.E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge, MA, USA (1999)
14. Knapik, M., Penczek, W., Półrola, A.: Bounded Parametric Verification for Time Petri Nets with Discrete-Time Semantics. In: Proceedings of CS&P'2009, http://csp2009.mimuw.edu.pl/csp09vol1_v2_numbered.pdf (2009) 277–290
15. Penczek, W., Woźna, B., Zbrzezny, A.: Bounded model checking for the universal fragment of CTL. Fundamenta Informaticae **51(1-2)** (2002) 135–156
16. Zbrzezny, A.: Improving the translation from ECTL to SAT. Fundamenta Informaticae **85(1-4)** (2008) 513–531
17. Zbrzezny, A.: A boolean encoding of arithmetic operations. In: Concurrency, Specification and Programming (CS&P'2008). Volume 3. (2008) 536–547
18. Biere, A.: Precosat. http://fmv.jku.at/precosat (2010)