Specializations and Symbolic Modeling

Vladimir Peschanenko[1], Anton Guba[2] and Constantin Shushpanov[3]

[1] Kherson State University, 27, 40 rokiv Zhovtnya str., Kherson, 73000 Ukraine,

vladimirius@gmail.com

[2] Glushkov Institute of Cybernetics of NAS of Ukraine, 40, Glushkova ave., Kyiv, 03680,

antonguba@ukr.net

[3] LLC «Information Software Systems», 15, Bozhenko str., Kyiv, 03680 Ukraine,

costa@iss.org.ua

Abstract. We present the technique that allows splitting first-order logic formulae into parts which helps to use the special algorithms of satisfiability checking and predicate transformer, which are the specializations. We describe the mathematical description of the algorithm of the constructing specializations and a few particular approaches to them, which speed up modeling of industrial models. We prove the correctness of satisfiability and predicate transformer functions. We consider forward and backward applicability of basic protocols during symbolic modeling and verification We introduce the examples for each specialization. We provide the experiments with typical real examples.

Keywords. Symbolic modeling, satisfiability, predicate transformer

Key terms. FormalMethod, MathematicalModeling, SoftwareComponent, VerificationProcess

1 Introduction

The technique of symbolic verification of requirement specifications of software systems has shown good results in automatic detection of reachability of deadlocks and violation of user-defined properties [1]. In previous works [2-4] symbolic models of systems being transition systems with symbolic states represented by formulae of first order logic were considered. A relation of transitions between the formulae is determined and marked by basic protocols, which are considered as actions, performed in the system. A basic protocol is a formula of dynamic logic $\forall x(\alpha(x,a) \rightarrow < P(x,a) > \beta(x,a))$ and it describes local properties of the system in terms of pre- and postconditions α and β. Both are formulae of first order multisorted logic interpreted on a data domain, P is a process, represented by means of MSC diagram and describes the reaction of a system triggered by the precondition, x is a set of typed data variables, and a is a set of environment attributes. The general theory of basic protocols is presented in [5].

A transition is considered as an operator in the space of postcondition formulae. As the operator transforms one formula to another, in [6] a term "predicate transformer" is used. Thus, to compute transitions between the states of such models basic protocols are interpreted as predicate transformers: for given symbolic state of the system and given basic protocol the direct predicate transformer generates the next symbolic state as its strongest postcondition, and the backward predicate transformer generates the previous symbolic state as its weakest precondition. These concepts have been implemented in VRS (Verification of Requirement Specifications) system [7] and IMS (Insertion Modeling System) system [8].

An amount of papers with novel and very efficient techniques for computing satisfiability using SAT/SMT has been published in the last years, and some very efficient SMT tools are now available (e.g., BarceLogic [9], CVCLite/CVC/CVC4 [10,11,12], DLSAT [13], haRVey [14], MathSAT [15], SDSAT [16], TSAT++ [17], UCLID [18], Yices [19], Verifun [20], Zapato [21], Z3 [22]). An amount of benchmarks, mostly derived from verification problems, is available at the SMT-LIB [23]. Workshops devoted to SMT and official competitions on SMT tools are run yearly.

All these tools could be configured with the help of many parameters, which means the usage of some techniques, tactics, heuristics or not, in order to gain in performance. In the paper [24] the algorithm configuration problem is stated as follows: given an algorithm, a set of parameters for the algorithm, and a set of input data, found parameter values under which the algorithm achieves the best possible performance on the input data. It gives a possibility of automated tuning of algorithm for obtaining performance on formulae of some theory.

Usually during modeling of real projects we deal with complex environment states and simple formulae of basic protocols (pre- and postconditions). It means that we should check the satisfiability of the conjunction of the environment state and the precondition formula and transform this whole big formula with the help of predicate transformer [6]. Obviously, the manipulation with whole formulae is not required for most of cases.

For example, let $i, j : \text{int}, f : \text{int} \to \text{int}$ be attributes and $f(i) > 0 \wedge f(0) < 5 \wedge$ $\wedge j > 0$ be an environment state, and $1 \to j := j+1$ be a basic protocol. Let's apply this basic protocol to the environment state. First, the satisfiability of conjunction of basic protocol precondition and environment state should be checked: $f(i) > 0 \wedge f(0) < 5 \wedge j > 0$. This checking should use the notion of functional symbols: $(i = 0) \to (f(i) = f(0))$. After that we should apply basic protocol postcondition to conjunction of environment state and precondition (see section Application of Basic Protocol):

$$\exists(v : \text{int})(f(i) > 0 \wedge f(0) < 5 \wedge v > 0 \wedge (j = v+1)) \Rightarrow$$
$$\Rightarrow f(i) > 0 \wedge f(0) < 5 \wedge \exists(v : \text{int})(v > 0 \wedge (j = v+1)) \Rightarrow$$
$$\Rightarrow f(i) > 0 \wedge f(0) < 5 \wedge j > 1$$

It is known that basic protocol changes attribute j only (see section about predicate transformers). It means that we could apply basic protocol to small part of environment state that depends on j, but not to whole environment state formula. In this example it could be $j > 0$ only. If there are no predicates in projects ,which could compare values of attribute j with values of other attributes, then we could use some special theories for manipulating with such formulae. In this example numeral intervals could be used for representation of values of attribute j. We call such special theories *Specialization of sat*, *pt functions* according to our general algorithm.

So, the main goal of this paper is to present a mathematical description of algorithm of constructing specializations and a few particular approaches to specialization which speed up modeling of industrial models. This paper is a continuation of the [25], where only concrete values as a kind of specialization were described.

In the Section 2 we describe the process of forward application of a basic protocol with the help of the satisfiability and the forward predicate transformer. In the Section 3 we present an applicability of basic protocols using satisfiability and backward predicate transformer. The specializations by memory usage and functional symbols are proposed in the Section 4. The results of experiments are discussed in the Section 5. In the Section 6 we summarize advantages of usage of the specializations and what could be done in the nearest future.

2 Forward Application of Basic Protocol

Let *S(a)* be an environment state, $\forall x(\alpha(x,a) \to< P(x,a) > \beta(x,a))$ be a basic protocol, where x – parameters of basic protocol, a – attributes of model, $D(x,a) = E(a) \wedge \alpha(x,a)$ – conjunction of environment state and precondition of basic protocol.

At the first step of application of basic protocol satisfiability of conjunction of environment state and precondition of basic protocol is checked: $sat(D(x,a))$. If the formula is unsatisfiable, then basic protocol is not applicable to environment state *S(a)*. If not, then process $P(x,a)$ is run and after forward predicate transformer is applied: $pt(D(x,a), \beta(x,a))$. The process of $P(x,a)$ is not considered in the paper, because the specialization tries to speed up the functions *sat* and *pt*.

2.1 Satisfiability

The checking formula satisfiability function *sat* is based on the Shostak method, adapted to combination of numerical, symbolic and enumerated data types. If all of the attribute expressions (simple attributes and functional symbols with parameters) that are free in the formula *S* are simple, then for satisfiability checking it is sufficient to prove validity of the closed formula $\exists(a,x)D(x,a)$, where a is a set of all simple attributes which occur in *S*, x is a set of parameters of basic protocol. For attribute expressions with parameters (including access functions to the elements of arrays), the Ackermann reduction of the uninterpreted functional symbols is used, where attribute expression is an attribute or functional symbol with parameters.

The Shostak method consists of the following. An expression of the form $f(x)$ is called as *Functional Expression*, if *f* is an attribute and *x* is a list of its parameters. At first, superpositions of functional expressions are eliminated by successive substitution of every internal occurrence of $f(x)$ by a new variable y, bounded by existential quantifier and added to the formula $y = f(x)$. For example, formula $P(f(g(x)))$ is replaced by formula $\exists y(y = g(x) \wedge P(f(y)))$. After all such replacements there will not be complex functional expressions in the formula. Further, for every attribute expression *f* of functional type all its occurrences $f(x_1),...,f(x_n)$ with the different parameters $x_1,...,x_n$ are considered. Occurrence $f(x_i)$ is replaced by variable y_i, bounded by existential quantifier and substitutive equations $(x_i = x_j) \rightarrow (y_i = y_j)$ are added. Now in the formula there are only simple attributes, and a method considered in [26] is used.

2.2 Forward Predicate Transformer

In general case, the post-condition looks like $\beta(x,a) = R(x,a) \wedge C(x,a)$, where $R = (r_1 := t_1 \wedge r_2 := t_2 \wedge ...)$ is a conjunction of assignments and *C(x,a)* is a formula part of post-condition.

We will consider three sets of functional expressions (we consider attributes as a functional expression with 0 arity): *r*, *s* and *z*. Set $r = (r_1, r_2,...)$ consists of the left parts of assignment, and also of other functional expressions that recursively depend on the left parts. In other words, *r* consists of the left parts of assignments and, if some functional expression *f* is included into this set, then all functional expressions in which *f* occurs are also included in *r*. Set $s = (s_1, s_2,...)$ consists of functional expressions which have external occurrences (not in arguments of such functional expressions) in formula part *C* of post-condition, but do not coincide with expressions from the set *r*. Finally, set $z = (z_1, z_2,...)$ consists of functional expressions which have external occurrences in formula *D* in right parts of assignments and in internal occurrences (in arguments if functional expressions) of the functional expressions of formula part *C* of post-condition and left parts of assignments, but these assignments are not included in two other sets (including parameter of basic protocol). Now, considering formulae, from which a post-condition and formula *D* are constructed as functions of external occurrences of elements of these sets, we get a presentation of post-condition in the following form:

$$B(r,s,z) = (r_1(r,s,z) := t_1(r,s,z) \wedge r_2(r,s,z) := t_2(r,s,z) \wedge ...) \wedge C(r,s,z)\,,$$

Predicate transformer is determined by the following formula:

$\mathrm{pt}(D(r,s,z), \beta(r,s,z)) = q_1 \vee q_2 \vee ...$,where

$q_i = \exists(u,v)(D(u,v,z) \wedge R(u,v,z) \wedge E_i(u,v,z) \wedge C(r,s,z))$,

$R(u,v,z) = (r_1(u,v,z) = t_1(u,v,z)) \wedge (r_2(u,v,z) = t_2(u,v,z)) \wedge ...)$, Formula $R(u,v,z)$ is a quantifier-free part of the assignment formula. Set of the variables *u(v)* represents new variables for each attribute expression from *r(s)* set. The *pt* substitutes attributes from *r(s)* set to variables from *u(v)* set in corresponded part of formula.

Each of disjunctive members q_i corresponds to one of possible means of identification of functional expressions occurring in formulae $\beta(x,a)$, and $E_i(u,v,z)$ is a set of equalities and inequalities corresponding to such identification.

To describe the construction of $E_i(u,v,z)$ we will consider the set *M* of all pairs of functional expressions in the form $(f(k), f(l)), k = (k_1, k_2, ...), l = (l_1, l_2, ...)$, where $f(k)$ is chosen from set *z*, and $f(l)$ – from sets *r* and *s*. These functional expressions shall be equal if their arguments were equal before application of basic protocol.

Let's choose arbitrary subset $N \subseteq M$ (including an empty set for every pair $(f(k), f(l)) \in N$ we will consider conjunction of equalities $k = l, (k_1 = l_1 \wedge k_2 = l_2 \wedge ...)$. We will unite all such conjunctions in one and will add to it conjunctive negations of all equalities, which correspond to pairs which are not included into the set *N*. We will denote the obtained formula as $G_i(r,s,z)$. If this formula is satisfiable, then the choice is successful. Now obviously, $f(k)$ is not independent and shall change the value because $G_i(r,s,z)$ is true. Thus, $f(k)$ shall change the value in the same way as $f(l)$. Set $E_i(r,s,z) = G_i(r,s,z) \wedge H_i(z,u,v)$ where $H_i(z,u,v)$ is a conjunction of equalities $f(k) = w$ if a variable *w* corresponds to $f(l)$. Thus, if $f(k)$ coincides with several functional expressions, it is not important what variable is chosen (transitivity of equality) [6].

3 Backward Application of Basic Protocol

Let $S(a)$ be an environment state after the application of the basic protocol $\forall x(\alpha(x,a) \rightarrow < P(x,a) > \beta(x,a))$, where *x* is parameters of basic protocol, *a* – attributes of model, $\beta(x,a) = R(x,a) \wedge C(x,a)$, where $R = (r_1 := t_1 \wedge r_2 := t_2 \wedge ...)$ is a conjunction of assignments and *C* is a formula part of post-condition, $D(x,a) = S(a) \wedge C(x,a)$ is a conjunction of environment state and formula part of postcondition of basic protocol.

3.1 Satisfiability

At first step of application of basic protocol in backward mode satisfiability of conjunction of environment state and formula part of postcondition of basic protocol is checked: $sat(D(x,a))$. If the formula is unsatisfiable, then the basic protocol is not applicable to environment state $S(a)$.If not, then process $P(x,a)$ is run and after a backward predicate transformer is applied: $pt^{-1}(D(x,a), \beta(x,a))$.

3.2 Backward Predicate Transformer

A backward predicate transformer considers three sets of functional expressions *r*, *s* and *z* (as forward too). A postcondition of the basic protocols is represented by the following formula:

$$B(r,s,z) = (r_1(r,s,z) := t_1(r,s,z) \wedge r_2(r,s,z) := t_2(r,s,z) \wedge ...) \wedge C(r,s,z)$$

A backward predicate transformer is determined by the following formula:

$\mathrm{pt}^{-1}(D(r,s,z), \beta(r,s,z)) = q_1^{-1} \vee q_2^{-1} \vee ...$, where

$q_i^{-1} = \exists(u,v)(D(u,v,z) \wedge R'(u,r,s,z) \wedge E_i(u,v,z)) \wedge \alpha(r,s,z)$,

$R'(u,r,s,z) = (u_1(r,s,z) = t_1(r,s,z)) \wedge (u_2 = t_2(r,s,z)) \wedge ...), u = \{u_1, u_2, ...\}$,

Each of disjunctive members q_i corresponds to one of possible identification of functional expressions, occurring in formulae $\beta(x,a)$ and environment state *S(a)*, where $E_i(u,v,z)$ are sets of equalities and inequalities corresponding to such identification. Formula $E_i(u,v,z)$ is built in the same way as in forward predicate transformer [27].

4 Specialization

We propose to use two types of specializations:
1. Specialization by memory usage
2. Specialization by functional symbol

4.1 Specialization by memory usage

Let a_1, a_2 be sets of attributes from initial environment state and $a_1 \cap a_2 = \varnothing \wedge a_1 \cup a_2 = a$, $S(a) = S_1(a_1) \wedge S_2(a_2)$ is environment state, $B(x,a) = \forall x(\alpha_1(x_1,a_1) \wedge \alpha_2(x_2,a_2) \rightarrow < P(x,a) > \beta_1(x_1,a_1) \wedge \beta_2(x_2,a_2))$ is basic protocol, where $x_1 \cap x_2 = \varnothing \wedge x_1 \cup x_2 = x$.

If $B'(x,a) = \forall x(\underset{i}{\vee} \alpha_i(x,a) \rightarrow < P(x,a) > \beta(x,a))$ then $sat(S(a) \wedge (\underset{i}{\vee} \alpha_i(x,a))) =$ $= \underset{i}{\vee} sat(S(a) \wedge \alpha_i(x,a))$ and $pt(S(a) \wedge (\underset{i}{\vee} \alpha_i(x,a)), \beta(x,a)) = \underset{i}{\vee} pt(S(a) \wedge$ $\wedge \alpha_i(x,a), \beta(x,a))$. So, in the next text we consider basic protocol as $B(x,a)$ only.

4.2 Theorem 1

$$sat(S_1(a_1) \wedge \alpha_1(x_1,a_1) \wedge S_2(a_2) \wedge \alpha_2(x_2,a_2)) =$$
$$= sat(S_1(a_1) \wedge \alpha_1(x_1,a_1)) \wedge sat(S_2(a_2) \wedge \alpha_2(x_2,a_2))$$

Proving.
Function *sat* builds closed formula. So,

$$sat(S_1(a_1) \wedge \alpha_1(x_1, a_1) \wedge S_2(a_2) \wedge \alpha_2(x_2, a_2)) =$$
$$= \exists(v_1, v_2, x_1, x_2)(S_1(a_1) \wedge \alpha_1(x_1, a_1) \wedge S_2(a_2) \wedge \alpha_2(x_2, a_2))$$

where v_1, v_2 are variables generated for attribute expression which depend on attributes a_1, a_2. It is known that $a_1 \cap a_2 = \varnothing \wedge a_1 \cup a_2 = a \wedge x_1 \cap x_2 = \varnothing \wedge$ $\wedge x_1 \cup x_2 = x$. It means that scope of quantifiers could be narrowed:

$$\exists(v_1, v_2, x_1, x_2)(S_1(a_1) \wedge \alpha_1(x_1, a_1) \wedge S_2(a_2) \wedge \alpha_2(x_2, a_2)) =$$
$$= \exists(v_1, x_1)(S_1(a_1) \wedge \alpha_1(x_1, a_1)) \wedge \exists(v_2, x_2)(S_2(a_2) \wedge \alpha_2(x_2, a_2)) = \mathrm{i}$$
$$= sat(S_1(a_1) \wedge \alpha_1(x_1, a_1)) \wedge sat(S_2(a_2) \wedge \alpha_2(x_2, a_2))$$

Theorem is proved.

This theorem means the following:

1. If $S(a) = S_1(a_1) \wedge S_2(a_2)$ and $\alpha(a, x) = \alpha_1(x_1, a_1)$ and $S(a)$ is satisfiable, then it is enough to check satisfiability of conjunction of $S_1(a_1) \wedge \alpha_1(x_1, a_1)$ for satisfiability checking of $S(a) \wedge \alpha(x, a)$. Checking of satisfiability of $S_2(a_2)$ is not required.
2. Checking of each part $sat(S_i(a_i) \wedge \alpha_i(x_i, a_i))$ could be done concurrently.

This case could be easily generalized to $a_1, ..., a_n$ case, because if it is possible to build subsets $a_i^1, a_i^2 \in a_i \wedge a_i^1 \cap a_i^2 = \varnothing \wedge a_i^1 \cup a_i^2 = a_i$ and to spilt an environment state and basic protocol accordingly to the *theorem 1,* then $sat(\underset{i}{\wedge} S_i(a_i) \wedge \alpha_i(x_i, a_i)) = \underset{i}{\wedge} sat(S_i(a_i) \wedge \alpha_i(x_i, a_i))$. So, after if we say about such pair of two sets $a_i^1, a_i^2 \in a_i \wedge a_i^1 \cap a_i^2 = \varnothing \wedge a_i^1 \cup a_i^2 = a_i$, then we understand that it could be applicable and for *n* sets.

Let's see how forward and backward predicate transformer can be applied.

4.3 Theorem 2

For forward application of basic protocol it is true that:

$$pt(S_1(a_1) \wedge \alpha_1(x_1, a_1) \wedge S_2(a_2) \wedge \alpha_2(x_2, a_2), \beta_1(x_1, a_1) \wedge \beta_2(x_2, a_2)) =$$
$$= pt(S_1(a_1) \wedge \alpha_1(x_1, a_1), \beta_1(x_1, a_1)) \wedge pt(S_2(a_2) \wedge \alpha_2(x_2, a_2), \beta_2(x_2, a_2))$$

Proving.

pt function builds sets r, s, z from postcondition $\beta_1(x_1, a_1) \wedge \beta_2(x_2, a_2)$ and formula $S_1(a_1) \wedge \alpha_1(x_1, a_1) \wedge S_2(a_2) \wedge \alpha_2(x_2, a_2)$, where r is a set of attribute expressions from left parts of assignments of postcondition, s is a set of attribute expressions from formula part of postcondition, z is a set of other attribute expressions from formula and postcondition. We know that sets of attribute expressions from pairs $S_1(a_1) \wedge \alpha_1(x_1, a_1)$, $\beta_1(x_1, a_1)$ and $S_2(a_2) \wedge \alpha_2(x_2, a_2)$, $\beta_2(x_2, a_2)$ are not intersected. It means that we could split each set r, s, z on subsets

$r = r_1 \cup r_2, s = s_1 \cup s_2, z = z_1 \cup z_2$ and $r_1 \cap r_2 = \varnothing$, $s_1 \cap s_2 = \varnothing$, $z_1 \cap z_2 = \varnothing$, because $a_1 \cap a_2 = \varnothing$. Let's write formula which is built by *pt* function.

Let $D(a,x) = D_1(x_1,a_1) \wedge D_2(x_2,a_2), D_1(x_1,a_1) = S_1(a_1) \wedge \alpha_1(x_1,a_1)$, $D_2 = S_2(a_2) \wedge \alpha_2(x_2,a_2)$ and $\beta_1(x_1,a_1) = R_1(r_1,s_1,z_1) \vee C_1(r_1,s_1,z_1)$, $\beta_2(x_2,a_2) = R_2(r_2,s_2,z_2) \vee C_2(r_2,s_2,z_2)$. So, general formula of predicate transformer is the following:

$$\underset{i}{\vee} q_i = \exists(u,v)(D(u,v,z) \wedge R(u,v,z) \wedge (\underset{i}{\vee} E_i(u,v,z)) \wedge C(r,s,z))$$

where $R(u,v,z) = R_1(u_1,v_1,z_1) \wedge R_2(u_2,v_2,z_2)$, $C(r,s,z) = C_1(r_1,s_1,z_1) \wedge$ $\wedge C_2(r_2,s_2,z_2)$. because $\beta(a,x) = \beta_1(x_1,a_1) \wedge \beta_2(x_2,a_2)$.

Let's write in details how to obtain $\underset{i}{\vee} E_i(u,v,z)$. It is known that $r_1 \cap r_2 = \varnothing$, $s_1 \cap s_2 = \varnothing$, $z_1 \cap z_2 = \varnothing$. To build such disjunction we should take into account all pairs of functional attribute expressions from sets *r,s* and *z*. It means that each such pair should be in set of attribute $(r_1 \wedge s_1; z_1)$ or $(r_2 \wedge s_2; z_2)$. So, $\underset{i}{\vee} E_i(u,v,z) = (\underset{i_1}{\vee} E_{i_1}(u_1,v_1,z_1)) \wedge (\underset{i_2}{\vee} E_{i_2}(u_2,v_2,z_2))$

Let's consider formula of predicate transformer:

$$\begin{aligned}
&pt(S_1(a_1) \wedge \alpha_1(x_1,a_1) \wedge S_2(a_2) \wedge \alpha_1(x_2,a_2), \beta_1(x_1,a_1) \wedge \beta_2(x_2,a_2)) = \\
&\underset{i}{\vee} q_i = \exists(u,v)(D(u,v,z) \wedge R(u,v,z) \wedge (\underset{i}{\vee} E_i(u,v,z)) \wedge C(r,s,z)) = \\
&= \exists(u_1,u_2,u_1,v_2)(D_1(u_1,v_1,z_1) \wedge D_2(u_1,v_1,z_1) \wedge \\
&\wedge R_i(u_1,v_1,z_1) \wedge R_i(u_2,v_2,z_2) \wedge \\
&\wedge (\underset{i_1}{\vee} E_{i_1}(u_1,v_1,z_1)) \wedge (\underset{i_2}{\vee} E_{i_2}(u_2,v_2,z_2)) \wedge \\
&\wedge C_1(r_1,s_1,z_1) \wedge C_2(r_2,s_2,z_2)) = \\
&= \exists(u_1,v_1)(D_1(u_1,v_1,z_1) \wedge R(u_1,v_1,z_1) \wedge (\underset{i_1}{\vee} E_{i_1}(u_1,v_1,z_1)) \wedge \\
&\wedge C_1(r_1,s_1,z_1)) \wedge \exists(u_2,v_2)(D_2(u_2,v_2,z_2) \wedge R(u_2,v_2,z_2) \wedge \\
&\wedge (\underset{i_2}{\vee} E_{i_2}(u_2,v_2,z_2)) \wedge C_2(r_2,s_2,z_2)) \wedge \ldots = \\
&= pt(D_1(x_1,a_1), \beta_1(x_1,a_1)) \wedge pt(D_2(x_2,a_2), \beta_2(x_2,a_2))
\end{aligned}$$

Theorem is proved.

4.4 Theorem 3

For backward mode it is true that:

$$pt^{-1}(S_1(a_1) \wedge C_1(r_1,s_1,z_1) \wedge S_2(a_2) \wedge C_2(r_2,s_2,z_2),$$
$$\beta_1(x_1,a_1) \wedge \beta_2(x_2,a_2)) = pt^{-1}(S_1(a_1) \wedge C_1(r_1,s_1,z_1), \beta_1(x_1,a_1)) \wedge$$
$$pt(S_2(a_2) \wedge C_2(r_2,s_2,z_2), \beta_2(x_2,a_2))$$

Proving.

$R(u,v,z) = R(u_1,v_1,z_1) \wedge R(u_2,v_2,z_2)$, $C(r,s,z) = C_1(r_1,s_1,z_1) \wedge C_2(r_2,s_2,z_2)$ because $\beta(r_a,x) = \beta_1(a_1,x_1) \wedge \beta_2(a_2,x_2)$. $\underset{i}{\vee} E_i(u,v,z) = (\underset{i_1}{\vee} E_{i_1}(u_1,v_1,z_1)) \wedge \wedge(\underset{i_2}{\vee} E_{i_2}(u_2,v_2,z_2))$ from previous theorem.

$$\text{pt}^{-1}(S(a) \wedge C(r,s,z), \beta(r,s,z)) = \underset{i}{\vee} q_i^{-1} =$$
$$= \underset{i}{\vee} \exists(u,v)(S(a) \wedge C(r,s,z) \wedge R'(u,r,s,z) \wedge E_i(u,v,z)) \wedge \alpha(r,s,z) =$$
$$= \exists(u_1,u_2,v_1,v_2)(S_1(r_1,s_1,z_1) \wedge S_2(r_2,s_2,z_2) \wedge$$
$$\wedge C_1(r_1,s_1,z_1) \wedge C_2(r_2,s_2,z_2) \wedge$$
$$\wedge (\underset{i_1}{\vee} E_{i_1}(u_1,v_1,z_1)) \wedge (\underset{i_2}{\vee} E_{i_2}(u_2,v_2,z_2))) \wedge$$
$$\wedge \alpha_1(r_1,s_1,z_1) \wedge \alpha_2(r_2,s_2,z_2) =$$
$$= \exists(u_1,v_1)(S_1(r_1,s_1,z_1) \wedge C_1(r_1,s_1,z_1) \wedge (\underset{i_1}{\vee} E_{i_1}(u_1,v_1,z_1))) \wedge \alpha_1(r_1,s_1,z_1) \wedge$$
$$\wedge \exists(u_2,v_2)(S_2(r_2,s_2,z_2) \wedge C_2(r_2,s_2,z_2) \wedge (\underset{i_2}{\vee} E_{i_2}(u_2,v_2,z_2))) \wedge \alpha_2(r_2,s_2,z_2) =$$
$$= pt^{-1}(S_1(a_1) \wedge C_1(r_1,s_1,z_1), \beta_1(x_1,a_1)) \wedge pt(S_2(a_2) \wedge C_2(r_2,s_2,z_2), \beta_2(x_2,a_2))$$

Theorem is proved.

Theorem 2 and theorem 3 mean that:

1. Functions *pt, pt^{-1}* could be applied separately and concurrently.
2. If postcondition contains $\beta_1(x_1,a_1)$ only, then
$$pt(S_1(a_1) \wedge \alpha_1(x_1,a_1) \wedge S_2(a_2) \wedge \alpha_2(x_2,a_2), \beta_1(x_1,a_1)) =$$
$$= S_2(a_2) \wedge \alpha_2(x_2,a_2) \wedge pt(S_1(a_1) \wedge \alpha_1(x_1,a_1), \beta_1(x_1,a_1))$$
$$pt^{-1}(S_1(a_1) \wedge C_1(r_1,s_1,z_1) \wedge S_2(a_2) \wedge C_2(r_2,s_2,z_2), \beta_1(x_1,a_1)) =$$
$$= S_2(a_2) \wedge C_2(r_2,s_2,z_2) \wedge pt^{-1}(S_1(a_1) \wedge C_1(r_1,s_1,z_1), \beta_1(x_1,a_1))$$

So, functions $sat(D_i(x_i,a_i)), pt(D_i(x_i,a_i), \beta_i(x_i,a_i))$ are called specialization, because we could use some special theories for implementation of it.

5 Examples of Usage of Specializations

5.1 Examples of Specializations by Memory Usage

Example 1. Concrete values. Let $S(a) = (i = 2) \wedge S(a/i)$ be an environment state where $i:\text{int}$ and a/i is a set of all attributes in model except *i*, $b = \forall x((i > 0) \rightarrow <> (i := i + 1))$. For application of such basic protocol we should check satisfiability of the next formula: $sat((i = 2) \wedge (i > 0)) = 1$, and the postcondition should be applied to $(i = 2)$: $\exists v((v = 2) \wedge (v > 0) \wedge (i = v + 1)) \Rightarrow (i = 3)$. For such examples direct *C++* translation could be used instead of using some special theories, and it will work much faster because it doesn't require any additional checking, just direct translation into *C++* code and compilation of it.

Example 2. Let $S(a) = (i < 2) \wedge S(a/i)$ be environment state where $i:\text{int}$ and a/i is a set of all attributes in model except *i*, $b = \forall x((i > 0) \rightarrow <> (i := i + 1))$. For application of such basic protocol we should check satisfiability of the next formula: $sat((i < 2) \wedge (i > 0)) = 1$, and the postcondition should be applied to $(i < 2)$: $\exists v((v < 2) \wedge (v > 0) \wedge (i = v + 1)) \Rightarrow (i > 1) \wedge (i < 3)$. For such examples numerical intervals could be used. So, $S(a) = (i \in (-\infty;2)) \wedge S(a/i)$, $b = \forall x((i \in (0;+\infty)) \rightarrow <> (i := i + 1))$. Satisfiability checking looks like just crossing of two numerical intervals: $i \in (-\infty;2) \cap (0;+\infty) \Rightarrow i \in (0;2) \Rightarrow i \in [1;1]$ for integer. Application of *pt* creates the following formula: $\exists v((v \in [1;1]) \wedge (i = v + 1)) \Rightarrow$ $\Rightarrow i - 1 \in [1;1] \Rightarrow i \in [2;2]$. This approach will work faster than general satisfiability checking and quantifiers eliminations. Such approach could be used for all numeric and enumerated types.

5.2 Examples of Specializations by Functional Symbol

It is not always possible to represent environment state and basic protocols in the following way: $S(a) = S_1(a_1) \wedge S_2(a_2)$, and $B(a,x) = \forall x(\alpha_1(x_1,a_1) \wedge$ $\wedge \alpha_2(x_2,a_2) \rightarrow < P(x,a) > \beta_1(x_1,a_1) \wedge \beta_2(x_2,a_2))$ where $a_1 \cap a_2 = \varnothing$, $a_1 \cup a_2 = a$, $x_1 \cap x_2 = \varnothing \wedge x_1 \cup x_2 = x$. One of such situation occurs when a value of functional attribute expression and its parameter has different types and belongs to the different subsets a_i. For example, if functional attribute: $i, j:\text{int}, f:\text{int} \rightarrow T$ is defined where $T \in (c_1, c_2, c_3)$ is enumerated type with three enumerated constants: c_1, c_2, c_3, then formula $(f(i) = c_1) \wedge i > 0$ could be represented with specializations as follows: $(f : v_1 = f(i)) \wedge (v_1 = c_1) \wedge (i > 0)$. Let $b = 1 \rightarrow <> (f(j) := c_2)$ be a basic protocol. Its specialized representation is: $b = 1 \rightarrow <> (f : v_1 = f(j)) \wedge (v_1 := c_2) \wedge 1$. It is required to merge such data structures for *pt* function which should consider all pairs of functional attribute expression from sets *r,s* and *z*:

$(f: v_1 = f(i)) \wedge (f: v_1 = f(j)) \Rightarrow (f: v_1 = f(i), v_2 = f(j))$. After that basic protocol should be transformed in the following form: $b = 1 \rightarrow<> (f: v_2 = f(j)) \wedge (v_2 := c_2) \wedge 1$. It is required to take into account two possible combinations: $(i = j) \vee \neg(i = j)$. So, we obtain:

$$\begin{aligned}
& pt((f: v_1 = f(i), v_2 = f(j)) \wedge (v_1 = c_1) \wedge i > 0, v_2 := c_2) = \\
& = (f: v_1 = f(i), v_2 = f(j)) \wedge \\
& \wedge \exists v((i = j) \wedge (v = c_1) \wedge (v_2 = c_2)) \wedge \\
& \wedge \exists v(\neg(i = j) \wedge (v_1 = c_1) \wedge (v_2 = c_2)) \wedge i > 0 \Rightarrow \\
& \Rightarrow (f: v_1 = f(i), v_2 = f(j)) \wedge (v_2 = c_2) \wedge i > 0 \wedge (i = j) \vee \\
& \vee (f: v_1 = f(i), v_2 = f(j)) \wedge (v_1 = c_1) \wedge (v_2 = c_2) \wedge i > 0 \wedge \neg(i = j) \Rightarrow \\
& \Rightarrow (f: v_1 = f(i)) \wedge (v_1 = c_2) \wedge i > 0 \wedge (i = j) \vee \\
& \vee (f: v_1 = f(i), v_2 = f(j)) \wedge (v_1 = c_1) \wedge (v_2 = c_2) \wedge i > 0 \wedge \neg(i = j)
\end{aligned}$$

Let $S(a) = F(f_1, f_2, ..., v_1, v_2, a_1, a_2) \wedge S_1(a_1) \wedge S_2(a_2)$ be an environment state where $f_1 \neq f_2 \neq ...$ are names of functional symbols, v_1, v_2 are variables for each functional attribute expression from sets a_1, a_2 correspondently, and

$$\begin{aligned}
& F(f_1, f_2, ..., v_1, v_2, a_1, a_2) = \\
& = (f_1 : v_1^1 = f_1(t_1^1, t_1^2, ...), v_1^2 = f_1(t_1^1, t_1^2, ...), ..., \\
& f_2 : v_2^1 = f_2(t_2^1, t_2^2, ...), v_2^2 = f_2^2(t_2^1, t_2^2, ...), ..., ...)
\end{aligned}$$

where $v_1^1, v_1^2 \in a_{f_1}, v_2^1, v_2^2 \in a_{f_2}, ...$ are variables of type of functional names $f_1, f_2, ...$ for each attribute expression, a_{fi} is set of attribute, such as $f_i \in a_j$, $t_i^j \in a_i^i \in \{a_i\}$, … - corresponded arguments for each functional with the same name are in one specialization, and Shostak's method could be applied for each right part of equation in *F*.

Let $S(a) = F(f_1, f_2, ..., v_1, v_2, a_1, a_2) \wedge S_1(a_1) \wedge S_2(a_2)$. and $b(a) = \forall x(F_b(f_1, f_2, ..., v_1, v_2, a_1, a_2, x_1, x_2) \wedge \alpha_1(v_1, x_1, a_1) \wedge$ $\wedge \alpha_2(v_2, x_2, a_2) \rightarrow< P(a, x) > \beta_1(v_1.x_1, a_1) \wedge \beta_2(v_2, x_2, a_2))$

5.3 Theorem 4

$$\begin{aligned}
& sat(S(a) \wedge \alpha(x, a)) = sat(\bigwedge_{(i,k,l)} ((f_i(t_i^1, t_i^2, ...) = f_i(t_i'^1, t_i'^2, ...)) \rightarrow (v_i'^k = v_i'^l)) \wedge \\
& \wedge S_1(v_1', a_1) \wedge \alpha_1(v_1', x_1, a_1) \wedge S_2(v_2', a_2) \wedge \alpha_2(v_2', x_2, a_2)) = \\
& = \bigvee_i sat(q_i \wedge S_i(v_i', a_i) \wedge \alpha_i(v_i', a_i, x_i))
\end{aligned}$$

where $f_i(t_i^1, t_i^2, ...) = f_i(t_i'^1, t_i'^2, ...)$ is equality of arguments of functional attribute expressions.

Proving

Let's define $F'(f_1,f_2,...,v_1',v_2',a_1,a_2,x_1,x_2) = F(f_1,f_2,...,v_1^1,v_2^1,a_1,a_2) \cup$ $\cup F_b(f_1,f_2,...,v_1^2,v_2^2,a_1,a_2,x_1,x_2)$. We combine all equations with the same name of functional symbol f_i and renaming variables names after such union for equations from basic protocol. After that we obtain sets of variables v_1',v_2' and new basic proto-
col $b(a) = \forall x(F_b(f_1,f_2,...,v_1',v_2',a_1,a_2,x_1,x_2) \wedge \alpha_1(v_1',x_1,a_1) \wedge$
$\wedge \alpha_2(v_2',x_2,a_2) \wedge ... \to < P(a,x) > \alpha_1(v_1',x_1,a_1) \wedge \alpha_2(v_2',x_2,a_2))$.

For satisfiable checking we should add corresponded implication for each pair of equation from $F'(f_1,f_2,...,v_1',v_2',a_1,a_2,x_1,x_2)$ with the same name of functional symbol f_i.

$$\underset{(i,k,l)}{\wedge}((f_i(t_i^1,t_i^2,...) = f_i^l(t_i'^1,t_i'^2,...)) \to (v_i'^k = v_i'^l)) =$$
$$= \underset{i,k,l}{\wedge}(\neg(f_i(t_i^1,t_i^2,...) = f_i(t_i'^1,t_i'^2,...)) \vee (v_i'^k = v_i'^l))$$

Each left and right parts of equation and negation of equations are in the same specialization. It means that we could build here a disjunction of conjunction. Each conjunct in such disjunction is q_i which will be in one form of our specialization. So, it means that we could check satisfiability in the following form $\underset{i}{\vee} sat(q_i \wedge S_i(v_i',a_i) \wedge \alpha_i'(v_i',a_i,x_i))$.

Theorem is proved.

5.4 Theorem 5

$$pt(S(a) \wedge \alpha(x,a), \beta(x,a)) =$$
$$= \underset{i}{\vee}(pt'(E_1^i(v_1',x_1,a_1), S_1(v_1',a_1) \wedge \alpha_1(v_1',x_1,a_1), \beta_1(x_1,a_1))) \wedge$$
$$\wedge\, pt'(E_2^i(v_2',x_2,a_2), S_2(v_2',a_2) \wedge \alpha_2(v_2',x_2,a_2), \beta_2(x_2,a_2))))$$

where

$$pt'(E_j^i(v_j',x_j,a_j), S_j(v_j',a_j) \wedge \alpha_j(v_j',x_j,a_j), \beta_j(x_j,a_j)) = \underset{k}{\vee} q_k',$$
$$q_k' = \exists(u,v)(S_i(u,v,z) \wedge \alpha_i(u,v,z) \wedge R(u,v,z) \wedge$$
$$\wedge E_i^j(u,v,z) \wedge E_k(u,v,z) \wedge C(r,s,z)$$

Proving.

The sets v_1',v_2' are built in the same way as in *theorem 3*. Let's consider a general formula of predicate transformer:

$$\mathrm{pt}(D(r,s,z), \beta(r,s,z)) = \underset{i}{\vee} \exists(u,v)(D(u,v,z) \wedge R(u,v,z) \wedge E_i(u,v,z) \wedge C(r,s,z).$$

Coefficient $\underset{i}{\vee} E_i(u,v,z)$ looks like disjunction of conjunction of all possible matchings with functional attribute expressions from sets *r,s* and *z*. So, we can present it as conjunction of two disjunctions: $\underset{i}{\vee} E_i(u,v,z) = (\underset{k}{\vee} E_k(u,v,z)) \wedge (\underset{l}{\vee} E_l(u,v,z))$ where $\underset{k}{\vee} E_k(u,v,z)$ is disjunction for matching of functional attribute expression where parameters and its value are from different sets of a_j. $\underset{l}{\vee} E_l(u,v,z)$ is a disjunction of matching of other functional attribute expression. Each conjunct of such disjunction could be considered as a conjunction which depends on different sets of memory a_j. It means that disjunction of conjunction $\underset{k}{\vee} E_k(u,v,z)$ could be prepared early before calling of some *pt* function without corresponded substitution of *x,y*. So, $\underset{k}{\vee} E_k(u,v,z) = \underset{k}{\vee} E_1^k(v_1',x_1,a_1) \wedge E_2^k(v_2',x_2,a_2)$. Disjunction $\underset{l}{\vee} E_l(u,v,z)$ could be presented in the same way. So, the theorem is proved.

5.5 Theorem 6

$$\begin{aligned} & pt^{-1}(S(a) \wedge C(r,s,z), \beta(x,a)) = \\ & = \underset{i}{\vee}(pt'^{-1}(E_1^i(v_1',x_1,a_1), S_1(v_1',a_1) \wedge C_1(v_1',x_1,a_1), \beta_1(x_1,a_1))) \wedge \\ & \wedge pt'^{-1}(E_2^i(v_2',x_2,a_2), S_2(v_2',a_2) \wedge C_2(v_2',x_2,a_2), \beta_2(x_2,a_2)))) \end{aligned}$$

where

$$\begin{aligned} & pt'^{-1}(E_j^i(v_j',x_j,a_j), S_j(v_j',a_j) \wedge C_j(v_j',x_j,a_j), \beta_j(x_j,a_j)) = \underset{k}{\vee} q_k', \\ & q_k' \exists(u,v)(S_j(v_j',a_j) \wedge C_j(v_j',x_j,a_j) \wedge R'(u,r,s,z) \wedge \\ & \wedge E_i^j(u,v,z) \wedge E_k(u,v,z)) \wedge \alpha_j(r,s,z) \end{aligned}$$

This theorem could be proved in the same mode as theorem 4.

6 Experiments

In this section we present some results from our test suites. All experiments are devided into several groups. We compare the time of modeling of the satisfiability and the predicate transformer, presented in the Section 2, and these algorithms with the specializations.

The first group of experiments refers to specialization by memory usage. Projects contain formulae in which some attributes have only concrete values. Let us present one typical real example. This example has a functional attribute of symbolic type with integer parameters, simple enumerated and simple integer attributes. All of these integer attributes initialize with concrete values and have concrete values at all times during trace generation (basic protocols do not change those to symbolic ones). Other attributes are symbolic. We provide a specialization for attributes, which are always concrete. The difference of modeling time for this example and for this one specialized by concrete values is more than in 3 times. Of course, the speedup depends on project: more concrete attributes we have, more speedup we shall obtain. In [25] it was shown that speedup could be in thousands times.

The second group of experiments refers also to specialization by memory usage, but not to concrete values. Examples from this group have enumerated attributes and integer attributes. Some of the integer attributes memories are intersected, some of them are independent. First of all, we provide the splitting of formulae into two parts according to attribute types: enumerated part and integer part. For the enumerated part we use *bitsets*, for integer – common Pressburger algorithm. Speedup was about 5-7%. After we specialize an integer part. We consider the attributes which memory is independent and obtain speedup in 10 times.

So, the results of comparison of modeling time using general satisfiability functions and functions with specialization are given.

Table 6. Results of experiments

Group of tests	**General algorithm**	**With specializations**
1	930 sec	300 sec (memory usage/concrete values)
2	300 sec	280 sec (splitting by types)
3	300 sec	33 sec (memory usage/independent memory)

7 Conclusions

Symbolic modeling is a powerful technique for the automated reachability of deadlocks and violations of user-defined properties. The main complexity of the reachability problem is in the complexity of satisfiability and predicate transformer functions. There are a lot of SMT-based techniques which speed up the satisfiability of formulae that satisfy some particular theory. We propose a technique that allows to speedup classical symbolic modeling when formulae could be splitted in several parts and used some special theories for manipulations with them, which are called specializations. The mathematical description of the algorithm for constructing specializations is provided and the correctness of such specializations is proved.

Specializations by memory usage and functional symbols are considered and examples for each are given.

The nearest plans are the investigation of additional kinds of specialization, because the more specializations we have, the more speedup we obtain.

References

1. Symbolic Modeling, http://en.wikipedia.org/wiki/Model_checking
2. Letichevsky, A., Gilbert, D.: A Model for Interaction of Agents and Environments. In: Bert, D., Choppy, C., Moses, P. (eds.) Recent Trends in Algebraic Development Techniques. LNCS 1827, pp. 311–328. Springer Verlag, Berlin Heidelberg (1999)

3. Letichevsky, A.: Algebra of Behavior Transformations and its Applications. In: Kudryavtsev, V. B., Rosenberg, I. G. (eds.) Structural Theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry, vol. 207, pp. 241–272. Springer Verlag, Berlin Heidelberg (2005)
4. Letichevsky A., Kapitonova J., Kotlyarov V., Letichevsky Jr., A., Nikitchenko N., Volkov, V., Weigert T.: Insertion Modeling in Distributed System Design. Problems of Programming, (4), 13–39 (2008)
5. Letichevsky, A., Kapitonova, J., Volkov, V., Letichevsky Jr., A., Baranov, S., Kotlyarov, V., Weigert, T.: System Specification with Basic Protocols. Cybernetics and System Analysis, (4), 3–21 (2005)
6. Letichevsky, A. A., Godlevsky, A. B., Letichevsky Jr., A. A., Potienko, S. V., Peschanenko, V. S.: Properties of Predicate Transformer of VRS System. Cybernetics and System Analyses, (4), 3–16 (2010)
7. Letichevsky, A., Kapitonova, J., Letichevsky Jr., A., Volkov, V., Baranov, S., Kotlyarov, V., Weigert, T.: Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications, In: ISSRE 2004, WITUL (Workshop on Integrated reliability with Telecommunications and UML Languages) , Rennes, 4 November (2005)
8. Letichevsky, A., Letychevskyi, O., Peschanenko, V.: Insertion Modeling System. In: Clarke, E.M., Virbitskaite, I., Voronkov, A. (eds.) PSI 2011. LNCS 7162, pp. 262–274, Springer Verlag, Berlin Heidelberg (2011)
9. Bofill, M., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: The Barcelogic SMT Solver. In: Gupta, Aarti and Malik, Sharad (eds.) CAV 2008. LNCS 5123, pp. 294–298, Springer Verlag, Berlin Heidelberg (2008)
10. Barrett, C., Berezin, S.: CVC Lite: A New Implementation of the Cooperating Validity Checker. In: Rajeev, A., Peled, D.A. (eds.) CAV '04. LNCS 3114, pp. 515–518, Springer Verlag, Berlin Heidelberg (2004)
11. Barrett, C., Tinelli, C.: CVC3. In: W. Damm and H. Hermanns (eds.) CAV '07. LNCS 4590, pp. 298–302, Springer Verlag, Berlin Heidelberg (2007)
12. Barrett, C., Conway, C. L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV'11. LNCS 6806, pp. 171–177, Springer Verlag, Berlin Heidelberg (2011)
13. Cotton, S., Asarin, E., Maler, O., Niebert, P.: Some Progress in Satisfiability Checking for Difference Logic. In: Proc. FORMATS-FTRTFT (2004)
14. Déharbe, D., Ranise, S.: Bdd-Driven First-Order Satisfiability Procedures (extended version). Research report 4630, LORIA (2002)
15. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., van Rossum, P., Schulz, S., Sebastiani, R.: An Incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In: Halbwachs, Lenore (eds.) TACAS'05. LNCS 3440, pp. 317–333, Springer Verlag, Berlin Heidelberg (2005)
16. Ganai, M. K., Talupur, M., Gupta, A.: SDSAT: Tight Integration of Small Domain Encoding and Lazy Approaches in a Separation Logic Solver. In: H. Hermanns, J. Palsberg. (eds.) TACAS 2006. LNCS 3920, pp. 135–150. Springer Verlag, Berlin Heidelberg (2006)
17. Audemard, G., Bertoli, P. G., Cimatti, A., Kornilowicz, A., Sebastiani, R.: A SAT based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In: A. Voronkov (ed.) CADE 2002. LNCS (LNAI) 2392, pp. 195–210. Springer Verlag, Berlin Heidelberg (2002)
18. Bryant, R.E., Lahiri, S.K., Seshia, S.A.: Modeling and Verifying Systems using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions.. In: Brinksma

K., Larsen G. (eds) CAV’04. LNCS 2404, pp. 78–92, Springer Verlag, Berlin Heidelberg (2002)

19. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In T. Ball and R.B. Jones, (eds.) CAV’06. LNCS 4144, pp. 81–94, Springer Verlag, Berlin Heidelberg (2006)
20. Walther, C., Schweitzer, S.: About veriFun. In: F. Baader (eds.) CADE’03. LNCS 2741, pp. 322–327, Springer Verlag, Berlin Heidelberg (2003)
21. Ball, T., Cook, B., Lahiri, S.K., Zhang, L.: Zapato: Automatic Theorem Proving for Predicate Abstraction Refinement. In: Alur, R. A., Peled D. A. (eds.) CAV'04. LNCS 3114, pp. 457–461. Springer Verlag, Berlin Heidelberg (2004)
22. de Moura, L., Bjørner, N.: Z3: An Eficient SMT Solver. In: C. R. Ramakrishnan, J. Rehof (eds.) TACAS'08, LNCS 4963, pp. 337–340. Springer Verlag, Berlin Heidelberg (2004)
23. Barrett, C., de Moura, L., Ranise, S., Stump, A., Tinelli, C.: The SMT-LIB Initiative and the Rise of SMT. In: Barner S., Harris I. (eds.) HVC 2010. LNCS 6504, pp. 3–3, Springer Verlag, Berlin Heidelberg (2010)
24. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stuetzle, T.: ParamILS: an Automatic Algorithm Configuration Framework. JAIR, 36, 267–306 (2009)
25. Peschanenko, V. S., Guba, A. A., Shushpanov, C. I.: Mixed Concrete-Symbolic Predicate Transformer. Bulletin of Taras Shevchenko National University of Kyiv, Series Physics & Mathematics, 2 (2013) (in press)
26. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability Modulo Theories. Frontiers in Artificial Intelligence and Applications, 185, 825–885 (2009)
27. Godlevsky, A. B.: Predicate Transformers in the Context of Symbolic Modeling of Transition Systems. Cybernetics and System Analysis, 4, 91–99 (2010)