

New predicate transformer for symbolic modeling

A.A. Letichevsky

Glushkov Institute of Cybernetics, Academy of Sciences of Ukraine, 40 Glushkova ave., Kyiv, Ukraine, 03187

O.A. Letychevskyi

Glushkov Institute of Cybernetics, Academy of Sciences of Ukraine, 40 Glushkova ave., Kyiv, Ukraine, 03187

V.S. Peschanenko

Kherson State University, 7, 40 rokiv Zhovtnya str., Kherson, Ukraine, 73000

T. Weigert

UniqueSoft LLC, 1530 E. Dundee Road, Suite 100, Palatine, Illinois 60074

Abstract

The paper presents a new predicate transformer for symbolic modeling. A predicate transformer is a symbolic function that computes the transitions of a system with states represented by means of first-order formulas. Systems under consideration are compositions of environments and agents inserted into these environments (insertion models), specified by means of basic protocols. A set of basic protocols represent the formal requirements of a system. Previous versions of predicate transformers allowed only existential quantifiers in formulas, while the predicate transformer presented here allows both existential and universal quantifiers, but with some restriction on formulas. The use of universal quantification is illustrated through the specification of the MESI protocol.

Key words: predicate transformer, insertion modeling, verification of requirement specification

Email addresses: let@cyfra.net (A.A. Letichevsky), lit@iss.org.ua (O.A. Letychevskyi), vladimirius@gmail.com (V.S. Peschanenko), thomas.weigert@uniquesoft.com (T. Weigert).

URL: <http://www.icyb.kiev.ua> (A.A. Letichevsky).

1. Introduction

The VRS (Verification of Requirements Specifications) system aims at the specification and verification of the requirements for large industrial systems (Baranov et al., 2003). The main difference of the verification techniques leveraged in VRS from model checking methods is twofold: Models with an infinite number of states are verified and a first order logic is used instead of the propositional calculus. We rely on a specification of system requirements by means of basic protocols (Letichevsky et al., 2005). System requirements are symbolically modeled where a predicate transformer is used for the definition of transitions of the system. We had earlier defined a predicate transformer for the case when only existential quantifiers appeared in the formulas representing the state of the system (Letichevsky et al., 2010). In this work we extend the definition of a predicate transformer to allow both existential and universal quantifiers. Some restrictions are imposed on attribute expressions using variables bound by universal quantifiers.

Predicate transformers are also used in other techniques such as static verification, invariant generation, and test generation. The usage of the enhanced predicate transformer is illustrated in specifying and verifying the MESI protocol.

1.1. System specifications

System specifications are represented in VRS by means of sets of basic protocols (Letichevsky et al., 2005). A basic protocols specification $\Sigma = \langle B, E \rangle$ of a system consists of a set of basic protocols B together with an environment description E . We consider systems represented as the insertion $E[u_1, u_2, \dots]$ of agents $u_1, u_2, \dots$ into an environment E . The insertion function $E[u] \rightarrow E'$ maps the state of the environment and the state of the agents into a new state of the environment (Letichevsky and Gilbert, 1998; Letichevsky, 2005). Environments and agents are transition systems considered up to bisimilarity or trace equivalence. In VRS, the environment description defines the structure of the environment while basic protocols define the insertion function.

1.2. Environment description

An environment description defines the signature of the language used to express pre- and postconditions of basic protocols and the interpretation of this signature. We refer to the language defined by the environment description as the *basic language* of a specification. We consider *attributed environments* which also define a set of attributes of the specified system (i.e., symbols that may take different values over time). Each attribute is of a simple or functional type. An *attribute expressions* f (for attributes with a simple type τ) or $f(t_1, t_2, \dots)$ (for attributes f with functional type $(\tau_1, \tau_2, \dots) \rightarrow \tau$) is called a *constant attribute expression* if it is a constant or all $t_1, t_2, \dots$ are constants. The environment description also defines the types, attributes, and behaviors of agents inserted into an environment as well as the set of possible initial states of agents and the environment.

In general, the states of an attributed environment consists of the formulas of basic language. Attributed environments are divided into *concrete* and *symbolic* environments.

The state of a concrete attributed environment is a formula of the form $t_1 = a_1 \wedge \dots \wedge t_n = a_n$, where $t_1, \dots, t_n$ are different constant attribute expressions and $a_1, \dots, a_n$ are constants. Typically, such a formula is represented as a partial mapping with domain $\{t_1, \dots, t_n\}$ and a range equal to the set of all constants. Sometimes it is useful to consider

an extended notion of a concrete environment with an infinite number of attribute expressions, all with concrete values. This corresponds to infinite conjunctions or a function with an infinite domain.

The state $s[m_1 : u_1, m_2 : u_2, \dots]$ of a symbolic environment can be identified with the formula $s \wedge [m_1 : u_1] \wedge [m_2 : u_2] \wedge \dots$, where $m_1, m_2, \dots$ are the names of agents and $[m : u]$ means that an agent named m is in the state u .

1.3. Basic protocols

The general form of a basic protocol is

$$\forall x(\alpha(x, r) \rightarrow \langle P(x, r) \rangle \beta(x, r)) \quad (1)$$

The typed variables $x_1, x_2, \dots$ in the list $x = (x_1, x_2, \dots)$ are considered as the *parameters* of the protocol. The formula $\alpha(x, r)$ is the *precondition*, $P(x, r)$ is the *process*, and $\beta(x, r)$ is the *postcondition* of the basic protocol. The list r is a list of attribute expressions. The precondition is a formula of the basic language, the process is an MSC diagram, and the postcondition is a formula of the basic language extended by assignment statements (considered as temporal logic formula).

In (Letichevsky et al., 2005) and (Letichevsky et al., 2008) several approaches to construct the semantics of basic protocol specifications were described. These approaches define a transition system on the set of concrete or symbolic states of the attributed environment and explicate the semantics either as *big step* semantics or *short step semantics*. The set of actions of a big step semantics are basic protocols while in a short step semantics the set of actions consists of the actions used in the processes of the basic protocols. Basic protocols define operators on the set of states of the environment which can be performed concurrently. In a big step semantics concurrency is hidden, while in a short step semantics it is defined explicitly. In the following, we shall consider a big step semantics only.

To define a big step semantics of a set of basic protocols we rely on the notion of a *predicate transformer*. A predicate transformer pr is a function that maps two formulas of the basic language to a new formula. This function is used to define a *big step transition system* for a symbolic attributed environment as follows. Let $B = \forall x(\alpha \rightarrow \langle P \rangle \beta)$ be a basic protocol. Then a step of the transition system $s \xrightarrow{B} s'$ is

$$s \xrightarrow{B} s' \Leftrightarrow ((s \wedge \exists x \alpha) \neq 0) \wedge (s' = \exists x(pr(s \wedge \alpha, \beta)) \neq 0)$$

In the expression $pr(s \wedge \alpha, \beta)$, the symbol x denotes a list of new simple attributes added to the environment, and after binding this formula by an existential quantifier it again is considered as a list of variables. If collisions appear, the symbols of the list x must be renamed. The main requirement for the function pr is that $pr(s, \beta) \models \beta$.

A big step transition system is deterministic which means that s' is a function of s which we shall denote by $s' = B(s)$. We also require that the predicate transformer is monotonic ($s \rightarrow s' \Rightarrow pr(s, \beta) \rightarrow pr(s', \beta)$) and distributive ($pr(s \vee s', \beta) = pr(s, \beta) \vee pr(s', \beta)$).

2. Example: MESI protocol

The well-known MESI protocol (Papamarcos and Patel, 1984) is used for the coherence control of the shared memory in a multiprocessor system with local cache memories for each processor (Sorin et al., 2011).

The cache memory of a processor consists of *lines* (data units of the cache memory). Each line is synchronized separately. The states of the lines are M, E, S, I (thus the name), with the following interpretation: M indicates that the content of a line is modified, E indicates that the data in a line is exclusive and coincides with the corresponding data in the main memory, S indicates that the data in this line may be stored in other lines, and I indicates that this line is invalid.

The model considered in this section is a simplified abstraction of a complex system consisting of bus, processors, shared memory, data, and programs. In this model, lines are considered as agents inserted into an environment containing the bus, processors and shared memory. The actions of a line are **Read** and **Write**. The level of abstraction of this model hides such details as the content of lines and the addresses for the **Read** and **Write** actions. The behavior of a line can be described by the following system of equations:

$$\begin{aligned} I &= \text{Read.}(S+E+M) + \text{Write.M} \\ S &= \text{Read.S} + \text{Write.M} \\ M &= \text{Read.M} + \text{Write.M} \\ E &= \text{Read.E} + \text{Write.E} \end{aligned}$$

For simplicity we assume that a cache memory consists of only one line associated with a processor.

Basic protocols allow us to specify a system with an undefined number of processors. The statement $[i : q]$ shall mean that the line i is in state q ; textually we will express this as $\text{Mesi}(i) = q$. We shall assume no specifics about the structure of the environment except the identity $E[i : I, u] = E[u]$, that is, the number of agent which are in the state I is infinite (or undefined). The following set of basic protocols defines this environment.

```

R1:Forall i(
  Forall j (Mesi(j) = I) -> <Read i> (Mesi(i) = E)
);
R2:Forall(i,k)(
  Forall j (j!=k -> (Mesi(j) = I)) & (Mesi(k) = E) & (i!=k)
  -> <Read(i)>
  (Mesi(k) = S & Mesi(i) = S)
);
R3: Forall (i,k) (
  Forall j ((j != i) -> Mesi(j) = S | / Mesi(j) = I) &
  (i!=k) & Mesi(k) = S & Mesi(i) = I
  -> <Read(i)>
  (Mesi(i) = S)
);
R4: Forall(i,k)(
  Forall j (j!=k -> (Mesi(j) = I)) & (Mesi(k) = M) & (i!=k)
  -> <Read(i)>
  (Mesi(k) = I & Mesi(i) = M)
);

```

```

R5: Forall i(
    ~(Mesi(i) = I) -> <Read i> 1
);
R6: Forall i(
    Mesi(i) = M -> <Write i> 1
);
R7: Forall i(
    Mesi(i) = E -> <Write i> Mesi(i) = M
);
R8: Forall i(
    Mesi(i) = S
    -> <Write i>
    Forall j (j!=i -> Mesi(j) = I) & Mesi(i) = M
);
R9: Forall i(
    Forall j (Mesi(j) = I)
    -> <Write i>
    Forall j (j!=i -> Mesi(j) = I) & Mesi(i) = M
);
R10: Forall i,k(
    Forall j (j != k & j != i -> Mesi(j) = I) &
    Mesi(i) = I & (Mesi(k) = E | / Mesi(k) = S)
    -> <Write i>
    Forall j (j!=i -> Mesi(j) = I) & Mesi(i) = M
);
R11: Forall i,k(
    Forall j (j != k & j != i -> Mesi(j) = I) &
    Mesi(i) = I & Mesi(k) = M
    -> <Write i>
    Forall j (j!=i -> Mesi(j) = I) & Mesi(i) = M
);

```

The basic protocols R1–R3 and R9–R11 describe the situation where a copy of the requested data is absent in the cache memory; the other protocols describe situations where there is a cache hit.

This example is interesting for several reasons.

- (1) It illustrates a method of describing an environment with an undefined number of agents. While real systems will contain only a finite number of agents, there is no practical difference between a very large number of agents and infinitely many. Reasoning over an undefined number of agents allows the specification to be independent of such specifics.
- (2) It illustrates the use of universal quantifiers.
- (3) There are many other environments exhibiting similar structure at some level of abstraction. (For example, telecommunication systems must operate correctly independent of the number of subscribers.)

3. Predicate transformer

Many functions pr satisfy the main requirements on a predicate transformer. The weakest such requirement is simply that $pr(s, \beta) = \beta$. However, this constraint completely drops the information about the state s . Following Dijkstra's methodology of predicate transformers (Dijkstra, 1975) we would like to define the strongest predicate transformer analogous to the strongest postcondition for a given precondition s after performing the postcondition β of a basic protocol, considered as an operator over the formulas of the basic language.

To refine the notion of the strongest predicate transformer, consider a transition system with states of a symbolic attributed environment and postconditions as actions: $s \xrightarrow{\beta} s' \Leftrightarrow s' = pt(s, \beta)$. Now compare the execution of a postcondition over both concrete and symbolic attributed environments. A symbolic attributed environment can simulate concrete attributed environments. Each state of the symbolic environment s 'covers' the set of states of a concrete environment. This set consists of the states σ such that $\sigma \models s$. The simulation condition can be formulated as

$$\sigma \xrightarrow{\beta} \sigma', s \xrightarrow{\beta} s', \sigma \models s \Rightarrow \sigma' \models s'$$

To establish this condition we define a transition relation on the set of concrete states. The states of a concrete environment are formulas, so postconditions can be applied to concrete states as well. However, the result in general is not concrete, so this semantics must be modified to preserve concreteness. The natural definition of a concrete environment is as follows:

$$\sigma \xrightarrow{\beta} \sigma' \Leftrightarrow (\sigma' \models \beta) \wedge Ch(\sigma, \sigma', \beta)$$

The condition $Ch(\sigma, \sigma', \beta)$ restricts the possible changes of the values of attribute expressions after a transition from σ to σ' by means of the operator β . When β is an assignment, the definition of Ch is clear: only the left hand side of the assignment can change its values. For the general case, we assume that $\beta = R \wedge C$ where R is a conjunction of assignments, and C is a formula of the basic language. By definition, only the following attribute expressions can change their values:

- (1) left hand sides of assignments,
- (2) outermost occurrences of attribute expressions in C which do not contain variables,
- (3) the results of substituting arbitrary constants for variables in the outermost occurrences of attribute expressions in C .

Let $change(\beta)$ denote the set of all attribute expressions obtained from β under this enumeration. Then $Ch(\sigma, \sigma', \beta) \Leftrightarrow x \notin change(\beta) \Rightarrow \sigma(x) = \sigma'(x)$.

Note that this definition depends on the syntactical representation of the formula β and the semantics of β will change when the representation of β will be changed in an equivalent way. To avoid this inconvenience, each postcondition β can be equipped with the set $Change(\beta)$ of attribute expressions which can change their values. If the attribute expression t contains variables, it must be represented in the form $\lambda x(t(x))$, where x is the list of variables occurring in t . This term represents the set $\{t(a) | a \in Dom(x)\}$. All attribute expressions from the set $change(\beta)$ must be included into the set $Change(\beta)$ but it can contain more attribute expressions.

We define $pt(s, \beta)$ as the strongest condition s' that satisfies the simulation condition. For this purpose, the inverse condition $\sigma' \models s' \Rightarrow \sigma \models s$ must be added. Finally the strongest predicate transformer is defined as follows:

$$(\sigma \xrightarrow{\beta} \sigma', s \xrightarrow{\beta} s') \Rightarrow (\sigma \models s \Leftrightarrow \sigma' \models s')$$

From this definition the existence of the strongest predicate transformer is obvious. In the following, pt will be used to refer to the strongest predicate transformer.

3.1. The strongest predicate transformer

While the strongest predicate transformer pt exists, how to express it in the basic language of a symbolic attributed environment is not obvious. In (Letichevsky et al., 2010) the formula $pt(s, \beta)$ was defined for those situations where s contains only existential quantifiers and β is a quantifier-free formula. We shall generalize this result for arbitrary first order formulas, with minor restrictions on the terms of $Change(\beta)$.

We want to compute $pt(s, \beta)$ for the state s of a symbolic attributed environment and postcondition $\beta = R \wedge C$, where s and C are first order formulas of the basic language, and $R = (r_1 := t_1, r_2 := t_2, \dots)$ is a parallel assignment, i.e., where the assignments to all attributes of the environment are performed at the same time (we shall write this as $R = r := t$). Four cases will be considered: In case 1, the set $Change(\beta)$ contains only simple attributes, case 2 restricts the set $Change(\beta)$ to attribute expressions without variables, case 3 allows variables bound by external universal quantifiers of C , and case 4 allows only variables bound by external universal quantifiers of the formula s in $Change(\beta)$. Consider a set $Unch(s, \beta)$ consisting of the outermost occurrences of attribute expressions in s which are not in $Change(\beta)$.

Represent the sets $Change(\beta)$ and $Unch(s, \beta)$ by the lists $\mathbf{Q} = (q_1, q_2, \dots)$ and $\mathbf{Z} = (z_1, z_2, \dots)$. Further, consider a list $\mathbf{X} = (x_1, x_2, \dots)$ of variables, corresponding one-to-one to the expressions in $\mathbf{Q}$. We shall use the following notation for substitution: $\varphi = subs(v : s(v) := t(v) | P(v))$ where v is a list of variables available for matching. The result $\varphi(E)$ of applying a substitution φ to expression E is obtained by simultaneous replacement of all outermost occurrences of expressions of the form $s(v)$ by $t(v)$ such that v satisfies the condition $P(v)$. Substitution without matching is denoted as $subs(s_i := t_i | i \in I)$ and also refers to outermost occurrences only.

The proof that pt is the strongest predicate transformer is based on the representation of this condition in set theoretical form. Let C_E be the set of concrete states of the environment E . Let $State(s) = \{\sigma \in C_E | \sigma \models s\}$, and $Nstate(S, \beta) = \{\sigma' | \exists (\sigma \in S) (\sigma \xrightarrow{\beta} \sigma')\}$, $S \subseteq C_E$. pt is the strongest predicate transformer if and only if the following statement holds:

$$State(pt(s, \beta)) = Nstate(State(s), \beta)$$

This statement consists of two inclusions: $State(pt(s, \beta)) \subseteq Nstate(State(s), \beta)$ and $Nstate(State(s), \beta) \subseteq State(pt(s, \beta))$. The former establishes the *consistency* of pt , the latter establishes its *completeness*. An equivalent form of this criteria is

$$\sigma' \models pt(s, \beta) \Leftrightarrow \exists (\sigma \in C_E) (\sigma \models s \wedge \sigma \xrightarrow{\beta} \sigma'), \sigma' \in C_E$$

Case 1. All attributes of the postcondition β have arity 0. In this case

$$pt(s, \beta) = \exists x(\varphi(s) \wedge (r = \varphi(t))) \wedge C$$

where $\varphi = \text{subs}(q_i = x_i | i = 1, 2, \dots)$.

Consistency. Suppose that $\sigma' \models pt(s, \beta)$. Then there exist constants $c_i, i = 1, 2, \dots$ such that $\sigma' \models \mu\varphi(s) \wedge (r = \mu\varphi(t)) \wedge C$, where $\mu = \text{subs}(x_i = c_i | i = 1, 2, \dots)$. Define σ so that $\sigma(q_i) = c_i, \sigma(r_i) = \mu\varphi(t_i)$ and $\sigma(g) = \sigma'(g), g \notin \text{Change}(\beta)$. Now $\sigma \models s$ and $\sigma \xrightarrow{\beta} \sigma'$.

Completeness. Let $\sigma \models s \wedge \sigma \xrightarrow{\beta} \sigma'$. C is valid, which follows from the main property of operator β . To prove $\exists x(\varphi(s) \wedge (r = \varphi(t)))$ it is sufficient to consider the values of $\text{Change}(\beta)$ in σ .

Case 2. Attributes of arbitrary arities are allowed but the following restriction holds: if $f(u)$ occurs in β , then the list u of its arguments does not contain variables, except for variables that are bound by external existential quantifiers of the formula s .

In constructing the formula $pt(s, \beta)$, the external quantifiers can be omitted and variables can be considered as additional simple attributes. After building the formula, the omitted quantifiers can be recovered.

Attribute expression of arities greater than 0 require that the following identity holds: if $u = v$ then $f(u) = f(v)$ (substitutionality of equality). It is obvious that if $f(u) \in \text{Change}(\beta), f(v) \in \text{Unch}(s, \beta)$ but since $(u = v)$ then $f(v)$ can be changed also and must be in $\text{Change}(\beta)$. We need to consider all cases of equalities and inequalities of arguments of attributed expressions.

Let $M = \{(u, v) | f(u) \in \text{Change}(\beta), f(v) \in \text{Unch}(s, \beta)\}$. Enumerating all subsets of M as $(J_1, J_2, \dots)$ we can enumerate all combinations of equalities and inequalities:

$$E_i = \bigwedge_{(u,v) \in J_i} (u = v) \wedge \bigwedge_{(u,v) \in M \setminus J_i} (u \neq v)$$

Let $\varphi_i = \text{subs}(v : f(v) := x_i | f(v) = q_i \vee \exists u((u, v) \in J_i))$. The strongest predicate transformer is defined by the following formulas:

$$\begin{aligned} pt(s, \beta) &= \exists x(p_1 \wedge p_2 \wedge \dots) \\ p_i &= (E'_i \rightarrow s_i \wedge R_i \wedge C) \\ E'_i &= \varphi_i(E_i), s_i = \varphi_i(s), R_i = (r(\varphi_i(u)) := \varphi_i(t)) \end{aligned}$$

Some of the formulas $p_1, p_2, \dots$ may not be satisfiable. In this case, the transition $s \xrightarrow{\beta} s'$ is undefined ($s' = 0$), i.e., the state s is a deadlock state. Also, some of $p_1, p_2, \dots$ may be identically true (for example $E_i = 0$), and they can be deleted from the conjunction. An implementation of the algorithm for computing pr must rely on heuristics to reduce the number of subsets of M to efficiently compute $p_1, p_2, \dots$. This case is slightly more general than the result of (Letichevsky et al., 2010).

Consistency. Let $\sigma' \models pt(s, \beta)$. Consider x such that all conditions $p_1, p_2, \dots$ are valid in σ' . The conditions E'_i are mutually exclusive and their disjunction is identically true. We can select i such that $\sigma' \models E'_i$ will be true. Therefore $\sigma' \models s_i \wedge R_i \wedge C$. Identifying attribute expressions according to E'_i we can consider them as simple attributes and continue as in case 1.

Completeness. Let $\sigma \models s \wedge \sigma \xrightarrow{\beta} \sigma'$. Again select i such that E'_i is valid in σ' . The identified attribute expressions can be considered as simple attributes and the proof continues as in case 1.

Case 3. Same as 2, but with a weaker restriction: if attribute expression $f(u)$ is in the list $\mathbf{Q}$, then the list of its arguments may contain occurrences of variables bound by external universal quantifiers of the formula C .

Let C be in prenex normal form and $C = \forall y C'$, where C' has no external universal quantifiers. The formula for pt then has the form

$$pt(s, \beta) = \exists x \forall y (p_1 \wedge p_2 \wedge \dots)$$

$$p_i = (E'_i \rightarrow s_i \wedge R_i \wedge C')$$

where the notation is as in case 2.

Consistency. Consider x such that $\forall y(\dots)$ is true. Taking arbitrary values for y , select i such that E'_i is true. Continue as in the previous case, considering y as a symbolic constant.

Completeness. Let $\sigma \models s \wedge \sigma \xrightarrow{\beta} \sigma'$, then $\sigma' \models \forall y C'$. Take arbitrary symbolic constant for y and continue as in the previous cases.

Case 4. Same as 2, but the restriction is weaker: if the attribute expression $f(u)$ is from the list $\mathbf{Z}$, then the list of its arguments may contain occurrences of variables bound by external universal quantifiers of the formula s .

Let s be in prenex normal form and let $s = \forall y s'$ where s' has no external universal quantifiers. The formula for pt then has the form

$$pt(s, \beta) = \exists x \forall y (p_1 \wedge p_2 \wedge \dots)$$

$$p_i = (E'_i \rightarrow s'_i \wedge R_i \wedge C')$$

where the notation is the same as in case 2, and $s'_i = \varphi_i(s')$.

The proofs of consistency and completeness are similar to the previous case.

3.2. Example: abstractions

Continuing with the MESI protocol, the initial state of MESI is the symbolic state $\forall i(\text{Mesi}(i) = I)$. The application of basic protocols generates an infinite number of states. Consider proving that the following property is a safety property:

```

Forall i(
  (Mesi(i) = E) & Forall(j:int)(j!=i -> (Mesi(j) = I)) | /
  (Mesi(i) = M) & Forall(j:int)(j!=i -> (Mesi(j) = I)) | /
  (Mesi(i) = S) & Exist k(
    (k!=i) & (Mesi(k)=S) & Forall j((Mesi(j) = I) | / (Mesi(j) = S))
  ) | /
  Forall(i)(Mesi(i)=I)
)

```

The formula is represented in the syntax of VRS. To reduce the system to a finite state system some abstraction is needed. This abstraction can be introduced in the definition of the insertion function by adding the following identity: $E[i : S, j : S, k : S] = E[i : S, j : S]$ if i, j, k are different. Usually i, j, k are bound by existential quantifiers, so it is not important which of them will be omitted. After this abstraction the system will be a finite state system, and safety can be proven after generating all states.

However, there is another way to prove the safety of coherence. There exists a backward predicate transformer pt^{-1} which computes the weakest precondition of basic protocols. The postconditions of basic protocols are used as preconditions for the backward predicate transformer. It can be proven that none of the postconditions are applied to the negation of coherence. Therefore, the initial state of MESI is not reachable under this assumption.

4. Conclusions

A new predicate transformer leveraged in the VRS system has been described. This predicate transformer extends the possibilities of symbolic modeling, but creates a new problem for the deductive system used in VRS: the lack of efficient simplification techniques that can work with the large formulas that appear in practice. Experimentations with our own SMT prover used for insertion modeling as well as with third party provers and solvers, for example z3 and CVC, are ongoing.

References

- S. Baranov, V. Kotlyarov, A. Letichevsky, and T. Weigert: Leveraging UML to Deliver Correct Telecom Applications, in L. Lavagno, G. Martin, and B. Selic (eds), UML for Real: Design of Embedded Real-Time Systems. Kluwer Academic Publ. (2003).
- E. Dijkstra: Guarded commands, nondeterminacy and formal derivation of programs, CACM, volume 18, number 8, pp. 453–457 (1975).
- A. Letichevsky: Algebra of behavior transformations and its applications. in V.B. Kudryavtsev and I.G. Rosenberg (eds.), Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry–V. 207, pp. 241–272, Springer (2005).
- A. Letichevsky and D. Gilbert: A general theory of action languages. Cybernetics and System Analysis, 1 (1998).
- A. Letichevsky, J. Kapitonova, V. Volkov, A. Letichevsky Jr., S. Baranov, V. Kotlyarov, and T. Weigert: System Specification with Basic Protocols. Cybernetics and System Analyses, 4 (2005).
- A. Letichevsky, J. Kapitonova, V. Kotlyarov, A. Letichevsky Jr., N. Nikitchenko, V. Volkov, and T. Weigert: Insertion modeling in distributed system design. Problems in Programming, pp. 13–38, 4 (2008).
- A. Letichevsky, A. Godlevsky, A. Letichevsky Jr., S. Potienko, V. Peschanenko: The properties of predicate transformer of the VRS system. Cybernetics and System Analyses, pp. 3–16, 4 (2010).
- M. Papamarcos and J. Patel: A low-overhead coherence solution for multiprocessors with private cache memories. Proceedings of the 11th annual international symposium on computer architecture, p. 348 (1984).
- D. Sorin, M. Hill, and D. Wood: A Primer on Memory Consistency and Cache Coherence, Synthesis Lectures on Computer Architectures, p. 197, Morgan and Claypool (2011)