Predicate Transformers for Local Description Units

A.A. Letichevsky, O.O. Letychevskyi

Glushkov Institute of Cybernetics,
Academy of Sciences of Ukraine
e-mail: et@cyfra.net, lit@iss.org.ua

ABSTRACT

Predicate transformers are symbolic functions used for computing transitions of system models with states represented by means of logic formulas. They are widely used for symbolic evaluation of programs, verification, abstract interpretation, and symbolic modeling of software. The models of software systems considered in this paper are represented by means of systems of local description units and their states are represented by means of first order logic formulas. Our goal is to develop an algorithm for computing the strongest predicate transformer for local description units considered as operators over formulas.

Keywords

Predicate transformers, symbolic verification, attributed transition systems, local description units

1. INTRODUCTION

Local description units are the main component of requirement specifications in insertion modeling paradigm. Insertion modeling is a trend that is developing over the last decade as an approach to a general theory of interaction of agents and environments in complex distributed multiagent systems. In recent years, insertion modeling has been applied to the development of systems for the verification of requirement specifications of distributed interacting systems [1,2,3,4,5]. The system VRS, developed in order from Motorola, has been successfully applied to verify the requirements and specifications in the field of telecommunication systems, embedded systems, and real-time systems. The mathematical foundation of insertion modeling is presented in [6]. The idea of local descriptions has been used in [5] where local descriptions were called basic protocols. Predicate transformer was defined axiomatically. Exact definition of predicate transformer used for VRS appeared later but only in [7] it was proved that predicate transformer of VRS really is the strongest one. The results of this paper generalize the results obtained in [7].

2. ATTRIBUTED TRANSITION SYSTEMS

Predicate transformers considered in this paper were developed for attributed environments with inserted agents. But actually our constructions do not depend on the structure of insertion models and can be reproduced for arbitrary transition systems with symbolic states. We call such systems attributed transition systems (or simply attributed systems) and attributed environments with inserted agents appear to be special cases.

Attributed systems are based on some *logic framework*. This framework includes a set of types (integer, real, enumerated, symbolic, behavioral, etc.), interpreted in some data domains. A framework includes symbols to denote constants of these domains, and a set of typed functions and predicate symbols. Some of these symbols are interpreted (e.g., arithmetic operations and inequalities, equality for all types, etc.). Uninterpreted function and predicate symbols are called *attributes*. Uninterpreted function symbols of arity 0 are called *simple attributes*, the other –*functional attributes* (uninterpreted predicate symbol is considered as a functional one with the binary values domain {0,1}). Function symbols are used to define data structures such as arrays, lists, trees, etc.

The *basic logic language* is built over a logic framework of attributed transition system. In this paper we shall consider only a first order language. *Attribute expression* is a simple attribute or an expression of the form $f(t_1,...,t_n)$, where f is a functional attribute of arity n, $t_1,...,t_n$ are already constructed attribute expressions or constants of suitable types. If all arguments of attribute expression are constants, then it is called a *constant expression*.

In general, the state of attributed system is the formula of basic language. Attributed transition systems are divided into two classes: the *concrete* and the *symbolic* ones.

A state of concrete attributed transition system is a formula of the form $t_1 = a_1 \wedge t_2 = a_2 \wedge ... \wedge t_n = a_n$, where $t_1,...,t_n$ are different constant attribute expressions, $a_1,...,a_n$ are constants. Typically, such a formula is represented as a partial mapping with domain $\{t_1,...,t_n\}$ and range equal to the set of all constants. It is natural to call this formula a memory state. Of course a formula must be type consistent (the type of the value of $f(t_1,...,t_n)$ must be equal to the type of the range of a function symbol *f*).

The states of symbolic attributed transition system are arbitrary formulas of basic language.

Sometimes it is useful to consider an extended notion of a concrete transition system with an infinite number of attribute expressions, which have concrete values. This corresponds to infinite conjunctions or a function with infinite domain. Such function is a mapping $\sigma : Attr \to D$ from the set *Attr* of all constant attribute expressions to the set of their values *D* (consistent with types). The mapping σ is naturally extended to the set of all expressions of a given system and to formulas of a given logic framework.

3. LOCAL DESCRIPTION UNITS

Local description units are used to specify the behavioral properties of attributed systems.

We shall consider local properties of a system represented as a concrete or symbolic attributed system. Local description unit of the attributed system is a formula $\forall x(\alpha \to < P > \beta)$, where *x* is a list of (typed) parameters, α and β – basic language formulas, *P* – process (finite behavior of the specified system). Formula α is called the *precondition*, and the formula β – the *postcondition* of local description unit. Both the conditions and the behavior of the unit may depend on parameters. Local description unit can be considered as a temporal logic formula that expresses the fact that, if (for suitable values of parameters) the state of a system satisfies the precondition, the behavior *P* can be activated and, after its successful termination, the new state will satisfy the postcondition.

Postconditions can contain assignments $f(x) := y$, where $f(x)$ is an attribute expression and *y* is an algebraic expression. This assignment is considered as a simple temporal logic statement which asserts that a new value of *f* in the point equal to the old value of attribute expression *x* is equal to the old value of algebraic expression *y*. Therefore, the local description unit $\forall z(\alpha \to < P > (f(x) := y) \wedge \beta)$ is equivalent to

$$\forall(u,v,z)(\alpha \wedge (x = u) \wedge (y = v) \to < P > (f(u) = v) \wedge \beta).$$

Local description units, which are used in the input language of VRS system are called *basic protocols*. They are the main units for expressing formal requirements to multiagent and distributed systems. Basic protocols are expressed in the basic language of the system VRS, and to represent the processes MSC diagrams are used in basic protocols. To study semantics underlying protocols several approaches were developed for the construction of such semantics. These approaches are described in [5] and in [8]. They are based on the notions of abstraction and implementation of a set of local descriptions.

There are two kinds of semantics: *big step* semantics and *short step semantics*. Both define a transition system on the set of concrete or symbolic states of attributed system. The set of actions of a big step semantics are local description units, in a short step semantics the set of actions consists with actions used in processes of local description units. Local description units define operators on a set of states of systems and they can be performed concurrently. In big step semantics this concurrency is hidden, in a short step semantics it is defined explicitly. In the sequel we shall consider only big step semantics.

To define big step semantics of local descriptions we use the notion of *predicate transformer*. Predicate transformer *pr* is a function that maps a formula of basic language to new formula. This function is used to define a *big step transition system* for symbolic attributed system as follows. Let $D = \forall x(\alpha \to < P > \beta)$ be a local description unit. Then

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

In the expression $pr(s \wedge \alpha, \beta)$ symbol *x* denotes a list of new simple attributes added to system, and after bounding this formula by existential quantifier it again is considered as a list of variables. Of course, if some collisions may appear, the symbols of the list *x* must be renamed.

The main requirement for the function *pr* is the relation $pr(s,\beta) \models \beta$.

A big step transition system is deterministic. It means that s' is a function of s and we shall use the notation $s' = B(s)$. We also require that predicate transformer function would be monotone and distributive.

4. PREDICATE TRANSFORMER

There are many functions *pr*, which satisfy the main requirement to predicate transformer. The weakest is simply $pr(s,\beta) = \beta$. It is not a good choice, because we lost completely the information about the state *s*. Following the Dijkstra methodology of predicate transformers [9] we would like to define the strongest predicate transformer like the strongest postcondition for a given precondition *s* after performing of postcondition β of a local description unit, considered as an operator over the formulas of basic language.

To refine the notion of the strongest predicate transformer let us consider the transition system with states of a symbolic attributed environment and postconditions as actions: $s \xrightarrow{\beta} s' \Leftrightarrow s' = pt(s,\beta)$. Let us compare the execution of a postconditions over a concrete and symbolic attribute environment. Symbolic attributed environment is used to simulate concrete environments. Each state of symbolic environment *s* "covers" the set of states of 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 refine this condition we must define transition relation on the set of concrete states. The state of concrete environment is a formula, so postconditions can be applied to concrete states as well, but the result in general is not concrete, so this semantics must be modified, to preserve concreteness. The natural definition for concrete environment is the following:

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

Here the condition $Ch(\sigma,\sigma',\beta)$ restricts the possible changes of the values of attribute expressions after transition from σ to σ' by means of operator β. In the case when β is assignment the definition of *Ch* is clear: only left hand side of assignment can change its value. To consider a general case we assume that $\beta = R \wedge C$, where *R* is a conjunction of assignments, and *C* is a formula of basic language. A formula *C* can contain quantifiers, so attribute expressions can contain variables. By definition 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 instead of variables into outermost occurrences of attribute expressions in *C*.

Let us assume that each postcondition is supplied by the set $Change(\beta)$ that includes the set of all attribute expressions obtained from the enumeration above. This set can contain more attribute expressions which are considered as changing in arbitrary way as a result of some hidden activity and interaction with external environment. Now predicate *Ch* can be defined as follows:

$$Ch(\sigma,\sigma',\beta) \Leftrightarrow \forall(t \in Attr)(t \notin \sigma(Change(\beta)) \Rightarrow \\ \Rightarrow \sigma(t) = \sigma'(t)).$$

So transitions of concrete systems are defined completely and we can define $pt(s,\beta)$ as the strongest condition s' that satisfies the simulation condition. For this purpose the inverse condition $\sigma' \models s' \Rightarrow \sigma \models s$ for some σ such that $\sigma \xrightarrow{\beta} \sigma'$ must be added. Finally, the strongest predicate transformer is defined as the condition satisfying the following two properties:

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

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

From this definition the existence of the strongest predicate transformer and its uniqueness is obvious. In the sequel we shall use the symbol *pr* for the strongest predicate transformer.

The strongest predicate transformer *pt* exists but the possibility to express it in basic language of symbolic attributed environment is not obvious. In [7] formula $pt(s,\beta)$ was defined as a first order formula of the basic language of VRS for the case when *s* uses only existential quantifiers and β is a quantifierless formula. Now we shall generalize this result for arbitrary first order formulas with some restrictions to terms of $Change\ (\beta)$.

We want to compute $pt(s,\beta)$ for the state *s* of symbolic attribute environment and postcondition $\beta = R \wedge C$ where *s* and *C* are first order formulas of basic language, and $R = (r_1 := t_1, r_2 := t_2, ...) = (r := t)$ is a parallel assignment. Four cases will be considered: case 1 corresponds to only simple attributes in the set $Change\ (\beta)$, case 2 restricts the set $Change\ (\beta)$ to attribute expressions without variables, case 3 allows variables bound by external universal quantifiers of *C*, and the case 4 allows only variables bounded by external universal quantifiers of formulas *s* and *C*. Let us consider a set $Unch\ (s,\beta)$ along with $Change\ (\beta)$. This set consists with outermost occurrences of attribute expressions in *s* which are not in $Change\ (\beta)$.

Let us represent the sets $Change\ (\beta)$ and $Unch\ (s,\beta)$ by lists $\mathbf{Q} = (q_1, q_2, ...)$ and $\mathbf{Z} = (z_1, z_2, ...)$. If an element of the list **Q** contains a variable, no substitution is needed. Only mark the occurrence of a variable by the quantifier that bounds it. Consider also a list $\mathbf{X} = (x_1, x_2, ...)$ of variables, set to one-to-one correspondence with the expressions from the list **Q**.

We shall use the following notation for substitution: $\varphi = subs(v : s(v) := t(v) \mid P(v))$. Here *v* is a list of variables for matching. The result $\varphi(E)$ of application of a substitution φ to expression *E* is obtained by simultaneous replacement of all outermost occurrences of expressions of the form $s(v)$ such that *v* satisfies condition $P(v)$ to $t(v)$. This substitution is equivalent to conditional rewriting rule applied with parallel outermost strategy. Substitution without matching is denoted as $subs(s_i := t_i \mid i \in I)$ and also refers to outermost occurrences. This substitution is considered as a set of unconditional rewriting rules applied with parallel outermost strategy. The following theorems can be proved.

Theorem 1. *If all attributes in the list* **Q** *have arity 0 then*

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

where $\varphi = subs(q_i = x_i \mid i = 1,2,...)$.

Theorem 2. *If attribute expressions of lists* **Q** *and* **Z** *do not contain variables then pt can be expressed by the first order formula.*

Let $M = \{(u,v) \mid f(u) \in \mathbf{Q}, f(v) \in \mathbf{Z}\}$.

Enumerating all subsets of the set **M** as $(J_1, J_2, ...)$ 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 $\varphi_i = \xi_i + \varphi$, where

$$\xi_i = subs(v : f(v) := x_j \mid f(v) \in \mathbf{Z}, (u,v) \in J_i, f(u) = q_j).$$

The strongest predicate transformer is defined by the following formulas:

$$pt(s,\beta) = \exists x(p_1 \wedge p_2 \wedge ...)$$

$$p_i = (E_i' \to s_i \wedge R_i \wedge C)$$

$$E_i' = \varphi_i(E_i), s_i = \varphi_i(s), R_i = (r(\varphi_i(u)) := \varphi_i(t))$$

This case is slightly more general than the result of [7].

Theorem 3. *Let* $C = \forall y C'$ *and be in a prenex normal form. Let all attribute expressions from the list* **Q** *have no variables except of the variables from the list y and attribute expressions from the list* **Z** *have no variables at all. Then*

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

$$p_i = (E_i' \to s_i \wedge R_i \wedge C')$$

Theorem 4. *Let* $s = \forall y s', C = \forall z C'$ *and both formulas be in a prenex normal form. Let all attribute expressions from the list* **Z** *have no variables except of the variables from the list y and all attribute expressions from the list* **Q** *have no variables except of the variables from the list z. Then*

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

$$p_i = (E_i' \to s_i' \wedge R_i \wedge C')$$

where all notations are the same as in the proof of theorem 2, but $s_i' = \varphi_i(s)$.

Transitions of local descriptions are computed as follows. Reduce the conjunction $s \wedge \alpha$ of the state of environment and the precondition of a local description $B = \forall x(\alpha \to < P > \beta)$ to prenex normal form. Let this normal form be $\exists zu$. Compute $v = pr(u,\beta)$ if *u* and β satisfy one of three conditions of theorems 1-4. The variables of a list *z* are considered as attributes. The result will be $B(s) = \exists x \exists z v$.

5. CONCLUSIONS

Symbolic verification methods for models described in the paper were implemented in IMS system developed in Glushkov institute of Cybernetics of the National Academy of Sciences of Ukraine. The use of universal quantifier in local descriptions allows description of system with unbounded number of agents and easy abstraction for complete verification.

REFERENCES

[1] Baranov, S., Jervis, C., Kotlyarov, V., Letichevsky, A., Weigert, T.: Leveraging UML to deliver correct telecom applications in UML for Real. In.: Lavagno, L., Martin, G., Selic, B. (eds) Design of Embedded Real-Time Systems, Kluwer Academic Publishers, pp. 323-342, 2003.

[2] Letichevsky, A., Weigert, T., Kapitonova,J. and Volkov, V.: Validation of Embedded Systems. In R. Zurawski, editor. The Embedded Systems Handbook, CRC Press, Miami, 2005.

[3] Letichevsky, A., Letichevsky, A.Jr., Kapitonova, J., Volkov, V., Baranov,S., Kotlyarov, V., Weigert, T.: Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications, ISSRE 2004, WITUL (Workshop on Integrated reliability with Telecommunications and UML Languages), Rennes, 4 November 2005.

[4] Letichevsky, A. Kapitonova,J. Letichevsky A. Jr., Volkov,V. Baranov, S. Kotlyarov,V. Weigert T. : Basic Protocols, Message Sequence Charts, and the

Verification of Requirements Specifications. Computer Networks, V. 47, pp.662-675, 2005.

[5] Letichevsky, A., Kapitonova,J., Volkov, V., Letichevsky, A.A.Jr., Baranov,S., Kotlyarov, V., and Weigert T. : System Specification with Basic Protocols. Cybernetics and System Analysis, V. 4, 2005.

[6] Letichevsky A.: 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.

[7] Letichevsky, A.A., Godlevsky, A.B., Letichevsky, A.A.Jr., Potienko S.V., Peschanenko, V.A.: The properties of predicate transformer of the VRS system. Cybernetics and System Analysis, V. 4, pp. 3-16, 2010.

[8] Letichevsky, A.A., Kapitonova, J.V., Kotlyarov, V.P., Letichevsky, A.A.Jr., Nikitchenko,N.S., Volkov,V.A., and Weigert,T.: Insertion modeling in distributed system design, Problems in Programming 4, pp. 13-38, Institute of Programming Systems, 2008.

[9] Edsger W. Dijkstra,: Guarded commands, nondeterminacy and formal derivation of programs, CACM August 1975, V. 18, N 8, pp. 453-457, 1975.