UDC 004.41,004.51

SIMPLE NON-DETERMINISTIC REWRITING IN VERIFICATION

A. Letichevsky, V. Peschanenko

Kherson State University, 40 Rokiv Zhovtnja str., 27, Kherson, 73000, Ukraine, phone: +38(0552)326707,
e-mail: vladimirius@gmail.com

Abstract. We discuss the non-deterministic rewriting in application for engine functions of Verification of Formal Specification (VFS) system in this paper. VFS – are tools to prove properties of systems described as formal specifications (basic protocols), such as the completeness (the system behavior has a possible continuation at each of its stages) and consistency (the system behavior is deterministic at each stage), safety (something bad will never happened), or the correspondence of the specified behavior to given scenarios. Together these tools constitute a powerful environment for the formal verification of formal specifications expressed through message sequence charts.

Introduction

VFS (Verification of Formal Specification) is based on the APS system [1] developed at the Glushkov Institute of Cybernetics. VFS is comprised of several implementation layers. On top is a level describing an environment specific for the subject domain under development which is tightly connected with the formalization of requirements. The second layer is the Action Language simulator [2] and is the basis for the key functionalities of VFS. This simulator is implemented in APLAN, the language based on rewriting logic. As the lowest layer the APLAN interpreter and its supporting libraries have been written in C/C++. The following tools have been realized within the VFS system.

- Creation and debugging of a specification formalized as a set of basic protocols.
- Verification of the formalized specifications.
- Generation of traces and scenarios.
- Proving of properties .

Basic protocols represent system requirements in the form of Hoare triples $\alpha \rightarrow < P > \beta$, where P is a process, and α and β are logical formulae (constituting pre and post-conditions of process P). Requirements stated as Hoare triples closely resemble the requirements used in engineering practice. (The main difference to the latter, of course, is the use of formal language instead of natural language.) Software designer typically specify system requirements as a set of possible behavior fragments expressing the system functionalities rather than developing specification in the form of complete scenarios.

The representation of requirements as basic protocols is based on the theory of interacting agents and environments [3]. In contrast to the major traditional theories of interaction, including CCS [4], CSP [5], or ACP [6, 7] which are based on an implicit and hence not formalized notion of an environment, the theory of interaction of agents and environments studies agents and environments as objects of different types. An environment may be considered as another agent, but sometimes an environment for a given agent is obtained from considering all other agents of the system acting in parallel with the given one. This theory was introduced in [8–10].

Verification of the set of basic protocols is performed by the Transition Consistency Checker. It processes the set of basic protocols and creates a verdict regarding the transition consistency and completeness of basic protocols.

The Trace Generator implements the generation of traces and scenarios from the set of basic protocols. Further, it detects deadlocks in the system of basic protocols, checks any safety conditions, and detects the reachability of different states of the system.

In this paper we present a new approach based on non-deterministic rewriting in application for generation of traces and scenarios of VFS. This approach has been implemented in the VFS system which has been realized in APS system.

The VFS system is based on algebraic and insertion programming [11, 12].

Section 2 introduces the basic notions of theory of interacting agents and environments[3], in Section 3 we present the formal definition of basic protocols and related concepts.

Section 4 introduces a new approach for realization of generating traces and scenarios in verification which based on non-deterministic rewriting. Section 5 is an application of non-deterministic rewriting for verification.

1. Agents and environments

We formalize system requirements in a language based on process algebra [7] enriched by the model of interaction of agents and environments [3]. *Agents* are labeled transition systems with states considered up to bisimilarity. They interact with each other by performing observable actions. The notion of an agent formalizes such diverse entities as software components, programs, users, clients, servers, or active components of distributed systems.

A state of an agent is defined by its behavior. Therefore, the equivalence of agents can be characterized in terms of the complete and continuous behavior algebra $F(A)$. This is an algebra with two sorts of elements — behaviors $u \in F(A)$, represented as finite or infinite labeled trees, and actions $a \in A$. As in basic process algebra, two operations are defined over $F(A)$: nondeterministic choice, which is an associative, commutative, and idempotent binary operation

$u+v$, where $u, v \in F(A)$ and *prefixing* $a.u \in F(A)$, where $a \in A$, $u \in F(A)$. The neutral element of non-deterministic choice is the deadlock element 0 (representing the impossible behavior). The empty behavior Δ performs no actions and denotes successful termination of the behavior of an agent. The algebra $F(A)$ is partially ordered by the approximation relation $\sqsubseteq$ with minimal element $\perp$. Both operations are continuous functions on the set of all behaviors over A. Completeness means that any directed set of behaviors has at least upper bound.

Each element u of the behavior algebra has a canonical representation

$$u = \sum_{i \in I} a_i.u_i + \varepsilon_u$$

defined up to commutatively and associativelynon-deterministic choice. In this representation, all $a_i.u_i$ are different behaviors, I is a finite or infinite set of indices, and $\varepsilon_u \in \{0, \Delta, \perp, \Delta + \perp\}$. We say that behavior v is *reachable* from u if $u = v + u'$ or (inductively) v is reachable from u_i for some $i \in I$. We call behavior u *divergent* if $\perp$ is reachable from u and convergent otherwise.

An *environment* E is an agent over a set of actions C together with an *insertion function* **Ins**(e, u) which we denote by $e[u]$. Its first argument e is a behavior of an environment, the second argument is a behavior of an agent over a set A of actions (of the agent) in a given state u. An insertion function is an arbitrary function continuous in both of its arguments. It yields a new behavior of the same environment.

We define an equivalence relation over agents which is, in general, weaker than bisimilarity. Two agents (in given states) u and v are insertion equivalent with respect to an environment E, written $u \sim_E v$, if for all $e \in E$, $e[u] = e[v]$. After the insertion of an agent into an environment, the new environment is ready to accept new agents to be inserted and multiple insertion allows to consider states of the form $e[u_1, u_2 \ , u_n]$ (shorthand for $(...((e[u_1])[u_2])...)[u_n]$).

To define an insertion function one defines labeled transitions on the set of states $e[u]$ of an environment with an inserted agent. We rely on rewriting (rewriting) rules

$$F(x)[G(y)]' \to d.F'(z)[G'(z)],$$

$$F(x)[G(y)]' \to F'(z)[G'(z)],$$

where $x = (x_1, x_2,...)$, $y = (y_1, y_2, ...)$, $z = (x_1, x_2, ..., y_1, y_2, ...)$, $x_1, x_2, ..., y_1, y_2, ...$ are action or behavior variables, $d \in C$, and F, G, F', G' are expressions in behavior algebra, that is, expressions built by non-deterministic choice and prefixing. The first kind of rule defines transitions of a type

$$\frac{e[u] \xrightarrow{*} e'[u'], e'[u'] \xrightarrow{d} e''[u'']}{e[u] \xrightarrow{d} e''[u'']}$$

where $\xrightarrow{*}$ denotes the transitive closure of unlabelled transition. Rewriting rules must be left linear with respect to their behavior variables, that is, no behavior variables may occur more than once in the left hand side. Further, rewriting rules for terminal and d

ivergent states must be added to ensure that the insertion function is continuous. Rewriting rules may define non-deterministic transition relations when two different left hand sides can be matched with the same state of an environment $e[u]$ (critical pairs are allowable).

We consider attributed transition systems, that is, systems with a mapping from states to attribute values. For attributed transition systems, the notion of bisimilarity must be slightly modified, and the behavior algebra should be considered together with an attribute mapping from behaviors to the attribute domain.

For more details about a theory of interacting agents and environments see [3, 12, 13].

2. System specification by means of basic protocols

2.1. Base language. Basic protocols are functionally definite fragments of system behavior. A system is defined as an attributed transition system, and the states of a system are observed by means of performed actions as well as attributes and variables defined on states changing their values in time. Properties of states are defined by means of formulae of some logic, which we refer to as the base language. Typically, the base language is first order, possibly with typed variables. The formulae of the *base language* may have attributes as the only free variables. Attributes may belong to functional types; to avoid higher order types, attributes may depend on parameters and functional expressions are restricted to first order expressions. The state of a system consists of the state of the environment which defines the values of attributes and the states of agents inserted into this environment if they are observable after insertion.

The choice of abstraction level of the base language is critical and depends on the problem domain and the state of development. Usually, a low level of abstraction with concrete states is used when the development of a system has been completed and the intention is to generate test cases from the requirements. A higher abstraction level is useful when one attempts to prove properties of the system requirements, where large collections of agents (processors in multiprocessor system, mobile phones, etc.) can be replaced by some formula expressing important properties of these collections.

2.2. Permutability relation. Before defining transitions, the set of actions C that can be performed by the system and observed by the external world needs to be defined. Actions are functional expressions of the base language and may depend on attributes. To generate the behavior of a system defined by basic protocols, we shall use a binary relation $a \leftrightarrow b$ on the set of actions called permutability relation. We assume that predicate $a \leftrightarrow b$ belongs to the base language, therefore its validity depends on the current state of a system and we can compute either semantic $s \models a \leftrightarrow b$ or syntactic inference $\alpha \vdash a \leftrightarrow b$, where α is a formula (or a set of formulae) of the base language. In the following, we shall assume that permutability relation does not depend on the state of a system; however, all main concepts can be extended to the general case.

We define permutability for the case $u \leftrightarrow b$ where u is a behavior over C and b is an action. This is the minimal relation such that:

1) for all actions b, $\Delta \leftrightarrow b, \overline{\perp \leftrightarrow b}, \overline{0 \leftrightarrow b}$;

2) $u + v \leftrightarrow b, u \leftrightarrow b \Leftrightarrow v \leftrightarrow b$;

3) $a.u \leftrightarrow b \Leftrightarrow a \leftrightarrow b \wedge u \leftrightarrow b$.

From this definition it follows that action b is permutable with behavior u if all actions reachable from u are permutable with b (action a is reachable from u if some v is reachable from u and $v \xrightarrow{\alpha} v'$), u is convergent, and 0 is not reachable from u.

Partial sequential composition of two behaviors. Let

$$u = \sum_{i \in I} a_i.u_i + e_u, v = \sum_{j \in J} a_j.u_j + \varepsilon_v$$

where $(\Delta; \varepsilon) = \varepsilon, (\perp; \varepsilon) = \perp, (0; \varepsilon) = 0$. Note that partial sequential composition is not continuous in the first argument, but it is continuous in the second one, and it is continuous in both when the first argument is finite and convergent.

If the permutability relation is always false, partial sequential composition coincides with sequential composition. If all actions are permutable, it coincides with interleaving parallel composition. Partially sequential composition originates from weak sequential composition introduced by Renier [14] for the definition of the operational semantics of MSC (Message Seauence Charts) and generalizes it further.

2.3. Basic protocols. Each basic protocol is a Hoare triple $\alpha \rightarrow < P > \beta$, where P is a process, α and β are precondition and postcondition of process P, respectively. α and β are represented by logical expressions of the base language and define conditions on the set of states of a system. A process of a basic protocol is a finite convergent process over the set C of environment actions, which may contain the set A of agent actions. We shall use the following notation for arbitrary basic protocols: ***pre***$(b) = \alpha$, ***post***$(b) = \beta$, and the process of B is denoted as P_b.

Each basic protocol defines properties of a system and can be understood as a statement of temporal logic: if the precondition is true then the process of a protocol can start, and after it has successfully terminated, the postcondition must be true.

2.4. Predicate transformers. Assume an assertion φ in the form of a formula of the base language means $\forall s(s \models \varphi)$ or $\vdash \varphi$ in a given theory.

A predicate transformer ***Tr***(α, β) is a function defined on formulae of the base language returning a new formula such that ***Tr***$(\alpha, \beta) \rightarrow \beta$. A predicate transformer strengthens the postcondition of a basic protocol by adding residual properties from the precondition.

2.5. Systems specified by basic protocols. A system is specified by its initial state and its properties. Let the initial state be described by a set of properties expressed in the base language, denoted as α_0. We then denote the behavior of a system generated by a set of basic protocols B and initial state satisfying α_0 by $S(B, \alpha_0)$. The system is usually not defined uniquely by the initial state α_0; rather, several protocols may be applicable in the initial state as the initial state α_0 may imply their precondition. Therefore, we can define the behavior of a system as the non-deterministic sum of behaviors starting in the initial state

$$S(B, \alpha_0) = \sum_{\alpha_0 \rightarrow \alpha} S_a.$$

The behavior S_α is the partially sequential composition of basic protocols from B. The first protocol is arbitrarily chosen from those basic protocols with a precondition satisfied by α. The set of all such conditions is $B(\alpha) = \{b \in B | \alpha \rightarrow$ ***pre***$(b)\}$. When the process of a basic protocol is completed, the postcondition of this basic protocol will be true. In fact, stronger set of conditions may be true, as the postcondition may not take all the aspects of the precondition into account. We consider the stronger condition given by the predicate transformer ***Tr***$(\alpha,$ ***post***$(b))$.

Consequentially, the behavior S_α is defined as

$$S_a = \sum_{b \in B(a)} P_b * (S_{Tr(\alpha, post(b))} + \Delta).$$

The summand Δ is added to generate not only infinite traces, but also finite ones. When the set $B(\alpha)$ is empty, $S_\alpha = 0$. Therefore, if Δ is absent, all finite traces, if any, terminate in the deadlock state 0.

A system is defined up to bisimilarity as a minimal fixed point of the above equations in the behavior algebra.

2.6. Scenarios. A system $S(B, \alpha_0)$ represents all possibilities of selecting basic protocols to construct behaviors. At times we are interested in a partial system description which can be obtained by restricting the choice of basic protocols. Behaviors obtained this way are called scenarios generated by basic protocols. To formalize this constuction we consider the set S_α of scenarios generated by the set of basic protocols B starting from initial condition α. This set is defined as a maximal set satisfying the following condition: If $S \in S_\alpha$, then there exists $b \in B$ such that $\alpha \rightarrow \boldsymbol{pre}(b)$ and $S = P_b * S' + S''$ where $S' \in S_{\boldsymbol{Tr}(\alpha, \boldsymbol{post}(b))}$ and $S'' \in S_\alpha$ or $S = P_b$.

If the set $B(\alpha)$ is not empty, then neither is the set S_α because it contains the *universal scenario* $S(B, \alpha)$ as well as $B(\alpha)$.

2.7. Parameterized basic protocols have the general form

$$\forall\ x\ (\alpha(x) \rightarrow\!< P(x) > \beta(x))$$

where $x = (x_1, \ldots, x_n)$. Parameterized basic protocols are used when there are infinitely many or at least a great number of similar basic protocols. Bound variables can be typed if the base language allows types. Substitution of constant (ground) values for x gives us the set $\boldsymbol{Inst}(B)$ of instantiated basic protocols; this set must be used instead of B in the definitions above.

3. Non-deterministic Rewriting

Let's consider APS[1] as a system for realization of non-deterministic rewriting. Unlike traditional approach oriented to the usage of canonical systems of rewriting rules with "transparent" strategy of their application, it is possible in APS to combine arbitrary s.r.r. with different strategies of rewriting. Such an approach essentially extends the possibilities of rewriting technique enlarging the flexibility and expressiveness of it. The APS integrates four main programming paradigms in the following way. The main part of the program can be written in the form of rewriting systems. Imperative and functional programming is used for the definition of strategies. Logic paradigm is realized on a base of rewriting using built-in unification procedure.

General definition of syntax of s.r.r. is the following:

<rewriting system> ::= rs(<list of variables separated by ",">)

(<list of rules separated by "," >)

<rule> ::= <simple rule> | <conditional rule>

<simple rule> ::= <algebraic expression> = <algebraic expression>

<conditional rule> ::= <condition> -> <simple rule>

<variable> ::= <identifier>

Each application of s.r.r. in APS satisfies the following conditions now:

1. One of the rules of the system is applied or arithmetic operation is performed at each step of rewriting.
2. The choice of a rule is made according to the sequence in which rules have been written.

For non-deterministic rewriting we should add new conditions:

3. After each successful application of some rule from s.r.r., rewriting is being continued with rules written below current.
4. Results of application of non-deterministic rewriting are separated by the special non-deterministic operation of behavior algebra *F(A)* (see section 2).

Such strategy for non-deterministic rewriting has been realized as ndcs (**n**on-**d**eterminist**c** **s**trategy) function in APS and successfully applied in the prototype of VRS system on APS.

4. Application of Non-Deterministic Rewriting in Verification

Let B be a set of basic protocols of a project, behavior $u = \sum a_i.u_i + \varepsilon_u$ where all $a_i.u_i$ are different behaviors, I is a finite or infinite set of indices, and $\varepsilon_u \in \{0, \Delta, \perp, \Delta + \perp\}$, $B_k = \forall(x,u)((a_k.u, \alpha(x)) \rightarrow\!< P(x) > (u, \beta(x)))$ where $B_k \in B, k \in I, a_k \in A$, E, E' are environments, e, e' are behaviors of environments (see section 2), and $e[a_k.u] \xrightarrow{B_k} e'[u]$.

We know that the behavior S_{a_k} is defined as

$$S_{a_k} = \sum_{b \in B(a_k)} P_b * (S_{Tr(\alpha(x), post(b))} + \Delta)$$

and non-deterministic rewriting gets all $b \in B(a_k)$. So, we can represent S_{a_k} as s.r.r. with all protocols from *B. W*e will get S_{a_k} after application of non-deterministic rewriting to state a_k.

Then,

$$S_a = rs(e,u)($$
$$satisfy(e \wedge \alpha(x)) \rightarrow (e[a_1.u] = P_1(x_1) * Tr(u, e \wedge \alpha(x_1), \beta(x_1))),$$
$$\ldots,$$
$$saatisfy(e \wedge \alpha(x)) \rightarrow (e[a_n.u] = P_n(x_n) * Tr(u, e \wedge \alpha(x_n), \beta(x_n)))$$
$$);$$

where n – a number of basic protocols in project, function *satisfy* checks satisfiability of conjunction of precondition and current environment.

And $S_{a_k} = ndcs(e[a_k.u], S_a)$.

Other good example for application of n.d.r. in VFS is based on experiments for effective term hashing algorithm realization on APS.

Let the initial state be described as a set of properties expressed in the base language, denoted as α_0. We then denote the behavior of a system generated by a set of basic protocols *B* and initial state satisfying α_0 by $S(B,\alpha_0)$. The system is usually not defined uniquely by the initial state α_0; rather, several protocols may be applicable in the initial state as the initial state α_0 may imply their precondition. Therefore, we can define the behavior of a system as the non-deterministic sum of behaviors starting in the initial state $S(B,\alpha_0) = \sum_{\alpha_0 \rightarrow \alpha} S_a$.

The behavior S_a is a partially sequential composition of basic protocols from *B*.

So, Let $\alpha = \{\alpha_0,\ldots,\alpha_n\}$ be a set of all possible states in behavior of a system $S(B,\alpha_0)$, n is a number of all possible states. How we could determine the set of basic protocols B_{α_i} from which we could get state α_i with a precision to names of variables? The answer for this question is n.d.r. in APS. Then we build the next s.r.r.:

$$B_a = rs(x)($$
$$\alpha_0 = B_{\alpha_0},$$
$$\ldots,$$
$$\alpha_n = B_{\alpha_n}$$
$$);$$

Where x is a set of variables from α, $B_{\alpha_i} \in B, i \in \{1,..,n\}$ is a basic protocol after application of which we get α_i. So, $B_{\alpha_i} = ndcs(\alpha_i, B_a)$.

Conclusions

So, the non-deterministic rewriting has being successfully applied in VFS system and together with optimized rewriting machine makes APS and VFS systems more powerful and extends the possibilities of their application in different arias.

1. *Algebraic* Programming System site [http://apsystem.org.ua]
2. *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 // Programs Systems, 4: 198: 228, 2008.
3. *Letichevsky A.A., Gilbert D.R.* A Model for Interaction of Agents and Environments. In: Selected papers from the 14th International Workshop on Recent Trends in Algebraic Development Techniques. Lecture Notes in Computer Science, 1827, 311–328, 1999.
4. *Milner R.* Communication and Concurrency. Prentice Hall, 1989.
5. *Hoare C.A.R.* Communicating Sequential Processes. Prentice Hall, 1985.
6. *Bergstra J.A., Klop J.W.* Process algebra for synchronous communication. Information and Control, 60(1): 109–137, 1984.
7. . *Bergstra J.A., Ponse A., Smolka S.A.*, Editors. Handbook of Process Algebra. Elsevier, 2001.
8. *Gilbert D.R., Letichevsky A.A* A universal interpreter for nondeterministic concurrent programming languages. In M. Gabbrielli (editor), Fifth Compulog network area meeting on language design and semantic analysis methods, Sep. 1996.
9. *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, ISSRE 2004, WITUL (Workshop on Integrated reliability with Telecommunications and UML Languages) , Rennes, 4 November 2005.
10. *Letichevsky A.A., Kapitonova J.V., Kotlyarov V.P., Volkov V.A., Letichevsky A.A.Jr, and Weigert T.* Semantics of Message Sequence Charts, SDL Forum, 2005.
11. Kapitonova J.V., A.A. Letichevsky, and S.V. Konozenko. Computations in APS. Theoretical Computer Science, 119:145–171, 1993.
12. *Letichevsky A.A., Kapitonova Y.V., Volkov V.A., Vyshemirsky V.V. and Letichevsky A.A.Jr.* Insertion Programming. Cybernetics and System Analysis, Kiev, 1, 2003.
13. *Letichevsky A.A. and Gilbert D.R.* A general theory of action languages. Cybernetics and System Analysis, (1):16–36, Feb. 1998.
14. *Reniers M.A.* Message Sequence Chart: Syntax and Semantics. Eindhoven, Eindhoven, University of Technology, 1998.

ПРОСТЕ НЕДЕТЕРМІНОВАНЕ ПЕРЕПИСУВАННЯ ПРИ ВЕРИФІКУВАННІ

А.А. Летичевський, В.С. Песчаненко

Херсонський державний університет, вул. 40 років Жовтня, 27, Херсон, 73000, Україна, тел: +38(0552)326707, e-mail: vladimirius@gmail.com

Анотація. У статті обговорюється недетерміноване переписування та його застосування до основних функцій системи верифікації формальних специфікацій (VFS). VFS – це набір інструментів для доказу властивостей систем, що описані у вигляді формальних специфікацій (базових протоколів), таких як повнота (поведінка системи продовжується з кожного з її станів), несуперечність (система має детерміновану поведінку у кожному зі станів), надійність (гарантія того, що все спрацює коректно), або відповідність до визначеної поведінки заданого сценарію. Разом ці засоби створюють потужне середовище для формальної верифікацій формальної специфікації Together these tools constitute a powerful environment for the formal verification of formal specifications expressed through message sequence charts.