UDC 004.41,004.51

A.A. Letichevsky, O.A. Letichevskyi, V.S. Peschanenko

The Non-Deterministic Strategy of Rewriting

Описана система алгебраического программирования – первая система переписывания термов, разделившая системы переписывающих правил и стратегии, а также система моделирования – базовая для системы верификации формальных спецификаций.

The Algebraic Programming System is described – the first term rewriting system, which uses the rewriting rules system and strategies separately and to Insertion Modeling System IMS – a basic system for the Verification of a Formal Specification system.

Описано систему алгебраїчного програмування – першу систему переписування термів, яка розділила системи правил переписування і стратегії, та систему інсерційного моделювання – базову для системи верифікації формальних специфікацій.

Introduction. Algebraic Programming System APS [1] was developed by the departments 100, 105 of Glushkov Institute of Cybernetics of the National Academy of Science of Ukraine [2] in 1987. It was the first system of term rewriting which used the RRS and strategies separately. The last version of APS system was created with collaboration of Research Institute of Information Technologies of Kherson State University [3] in 2009.

Unlike traditional approach oriented to the usage of canonical RRS with "transparent" strategy of their application, in APS it is possible to combine arbitrary RRS with different strategies of rewriting. Such 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. One of its most important application is the Insertion Modeling System (IMS). The main differences between APS and ELAN [4], MAUDE [5], STRATEGO [6] are presented.

So, the article is devoted to a special rewriting strategy of APS system and its application for VFS system.

The section "APS and Other" describes the functional possibilities and differences between APS system and ELAN, MAUDE, STRATEGO systems. The section "APS Rewriting" is devoted to RRS representation in APS and some special algorithms of work with RRS. We describe our proposal for non-deterministic rewriting strategy (ND Strategy) in section "Non-Deterministic Strategy of Rewriting". In section "VFS system" we propose a short description of Semantics of Basic Protocols of VFS system. The section "Non-Deterministic Strategy Application" presents example of application of such special strategy for verification.

APS and Other

Let's demonstrate the comparison of functional possibilities of APS system [1] and ELAN [2], MAUDE [3], STRATEGO [4] systems.

Where EA, SA is transformation systems including the compilers, interpreters, static analyzers, domain-specific optimizers, code generators, source code refectories, documentation generators, and document transformers; MA is the general logics and logical frameworks, specification languages, declarative programming languages, semantics of programming languages and models of computation, distributed systems, formal tools and formal interoperability, reflection and metaprogramming, object-oriented modelling and programming, real-time systems, bio informatics, mobile languages, network protocols and active networks; AA – algebraic programming, insertional modelling, program transformation, general logics and logical frameworks, specification languages, declarative programming; ACP – VRS

(Verification of Requirement Specification), TERM (School System of Computer Algebra); * – can't find any information about concrete projects; ** – to the binary files and system commands; *** – C – version of the arbitrary paths of program; +,– means that the system supports compilation of some small sub-set of system's language.

Without doubts due to quite developed typification in the system, MAUDE has more benefits than other systems (the process of evaluation with integers numbers). In this connection, there is a quite limited number of rewriting strategies in the system that considerably complicates the algorithms realized in it.

It is clear from the table that APS doesn't concede to well-known systems of terms rewriting as per all criteria.

Let's compare the capacity of terms rewriting systems taking the example of finding of n – number of Fibonacci (in this case we are interested in total operation time of the program which is used for rewriting only).

We are going to perform test on DELL VOSTRO 1500 (CPU Intel Core duo 2.0, Memory 2 Gb, HDD 160 Gb). The results of launching of this program in different systems of term rewriting are presented:

From the other side, APS was considered to be one of the slowest systems of term rewriting. Taking into consideration the results of capacity of rewriting, it can be said that APS is a quite quick system of term rewriting (after the elimination of some deficiencies). Surely, it doesn't have compiler, but instead of it we propose a number of tools for convenience of programming in APLAN (Algebraic Programming LANguage, the language of APS system) as well as in C++. These tools take APS to the new more qualified level (the compilers of ELAN, MAUDE, Stratego don't support language possibilities completely).

T a b l e 2. The results of launching of algorithms finding of Fibonacci n-number

No	System names	Fibonacci number (in seconds)					
		15	20	21	22	23	24
1	Interpreter of ELAN	0	2	6	11,5	18,5	28
2	Interpreter of Stratego	0	3	7	12	21	34
3	Interpreter of MAUDE	0,004	0,04	0,068	0,072	0,104	0,236
4	Procedures of APS	0	1	1	3	4	7
5	Rewriting systems of APS	0	2	2	4	6	10

Let's examine in details the mentioned above deficiencies of APS system. The first version of APS system had the memory leaks. The “garb” operator for collection of waste in the program was realized in APS system but as practice proved this operator didn't delete memory completely. In a course of analysis of the source code the designers discovered places of memory leaks. It was precipitated out 7 bytes of memory at the procedure calling, described in APLAN language. However, the actual C++ code which performs the calling of these procedures does not contain obvious calling functions of memory selection. It led to the fact that the used memory increased very fast at execution of the program and some small instances simply were terminated due to lack of memory in computer. As a result we have taken the following decision to implement the technology Smart Pointers in APS [12]. Thus, by means of this technology in the second version of APS system it was possible to be saved of this deficiency.

T a b l e 1. Comparison of Functional Possibilities Between TRS's

No	Name	Strategies Number	None Typing Strategies and rules	Procedural Language	Possibilities of Language Extension	User Manual Publication	Connection to the External Modules	Compilation	Dynamical Creation of the RRS	Support	Application Area	Commercial Products	Country
1	ELAN	arbitrary	–	–	+	1992	–	–,+	–	+	EA	*	France
2	STRATEGO	arbitrary	+	–	–	1994	–	–,+	+	–	SA	*	Netherlands
3	MAUDE	7	+	–	–	1995	–	–,+	–	+	MA	*	USA
4	APS	arbitrary	+	+	+	1987	**	***	+	+	AA	ACP	Ukraine

APS Rewriting

General definition of syntax of RRS is the following:

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

(< list o f rules separated by "," >)

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

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

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

< variable >::=< identifier >

Each application of RRS 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 the rules have been written.

Each RRS in APS applies to a term $t \in T_\Omega(Z)$, where $T_\Omega(Z)$ is algebra of terms of some algebraic program (see subsection, Ω is a signature of operation of this algebra and Z is a generic set of terms with zero arities, with some strategy: *applr* applies RRS to a term ones, *appls* applies while it's possible etc.

Rewriting machine of APS realized strategy *applr* which is a base for all strategies in APS. APS used a special language for faster application of *REM* – *REM* (*RE*writing *M*achine) language[13]. Algebraic definition of *REM* language is presented by the next algebra.

Let $T_\Omega(Z)$ be the base algebra of terms for some algebraic program. Set Ω is an operation signature of this algebra, Z – is a generating set of terms of arity 0. *REM* programs produce many-sorted algebra $R = (R_k)_{k \in Z}$ above $T_\Omega(Z \cup V)$, where V – is a set of variables of this program, R_k – set of programs of rank k. The corresponding signature contains the next operation:

1. $+ : R_k \times R_k \to R_k, k > 0;$

2. $test(\omega((),...,())) : R_{n+m-1} \to R_n, n > 0, m > 0, \omega \in \Omega, m = ART(\omega)$ (function ART returns arity of term).

3. $match(t) : R_n \to R_{n+1}, n \geq 0, t \in T_\Omega(Z \cup V)$.

4. $rewrite(t) \in R_0, t \in T_\Omega(Z \cup V)$.

5. $If(u, rewrite(t)) \in R_0; u, t \in T_\Omega(Z \cup V)$.

6. $hash(t) : R_k \to R_k, k > 0, t \in S_{T_\Omega(Z \cup V)}$, where $S_{T_\Omega(Z \cup V)}$ is set of marks and term of arity 0 from $T_\Omega(Z \cup V)$.

Operations 4 and 5 have zero arity and they produce elements for algebra of *REM* language, as follows from definitions.

SSR which was successfully converted in representation of REM language is called *REM-program*.

Dynamical Adding and Removing of Rules from the RRS

The process of application of RRS to current term has a few stages: conversion to *REM-program* (one time for each system only), interpreting of *REM-program* by the kernel of rewriting machine. It means that if we have to update RRS (add, remove or update some rule), APS system has to rebuild it into *REM-program* each time. But this operation demands more time, especially for big RRS. So, in APS system we have realized two operators:

- *remove_rule_rs* – the function for removing of rule from RRS without rebuilding of it.
- *add_rule_rs* – the function for adding of rule to RRS without rebuilding, dynamical adding of rule in *REM* language.

More interesting function is *add_rule_rs*. This function adds new rule into RRS without its rebuilding. Its means that we should add new operation from signature of algebra which corresponds to insertion of rule into *REM-program*. We should create new RRS by the next ideas to this effect:

1. Using list of variables from current RRS for new RRS with one new rule.

2. Conversion of new RRS into *REM-program*.

3. Insertion of new *REM-program* into current RRS (there are two possibilities to add new rule into RSS: to make it the first and to make it the last).

4. Using already known size of previous RSS making of number of insertion rewriting rule (this number can be used by appls strategy).

The unification algorithm of two *REM-programs* uses the next ideas (*new_rs* – new *REM-program*, *old_rs* – old *REM-program*):

1. In any case new_rs will have the next template *match(new_rs_m)rewrite*(_) or *match*(*new_rs_m*) *If*(_,_).

2. If *old_rs* has the next template (in terms of APS) *hash*(_)_ then if hash contains the main mark of *new_rs_m* then we should call recursive to *hash*(*type*(*new_rs_m*)) and *new_rs* if it's possible or if not then we should use + operator from signature of algebra.

3. If *old_rs* has the next template _+_ then we should try to add in first argument and if it is not possible then to try to add in second argument.

4. If *old_rs* has the next template *match*(*old_rs_m*)*last_part* (it is possible only in one case: if body of *match* of *old_rs* and *new_rs* has equal main mark) then we should eliminate the *new_rs_m* from *old_rs_m* and call algorithm recursively with *last_part*.

5. At last, we should use *hash* operator if it's possible or + operator if it's not.

Non-Deterministic Strategy

A non-deterministic rewriting strategy is an enhancement of theory of Set Functions for Functional Logic Programming [14] by the means of fuzzy sets and its application of rewriting. We use the next conditions for realization of non-deterministic rewriting strategy:

1. After each successful application of some rule from RRS rewriting we continue with rules written below current.

2. Results of application of non-deterministic strategy of rewriting are separated by the special non-deterministic operation +.

Let's show how it differs from the *applr* strategy. The strategy *applr* is presented as a function with two arguments: term which should be rewritten and *REM-program*. The high level of realization *applr* strategy in APLAN language is *napplr*:

$$NAME \quad napplr, appl;$$
$$napplr := proc(t,p)loc(pr, Yes, s)($$
$$let(p, rs \quad pr \quad p);$$
$$s := (p, t \quad Nil, pr) + Nil;$$
$$appls(s, appl);$$
$$let(s, 1:s);$$
$$yes \to t := s$$
$$);$$

The main part of it is presented by RRS *appl*, which is applied interactively to a state of rewriting machine *s*. The initial state contains program *p*, initial term *t* joins with constant *Nil*, and array *pt* which consists of «_». It is a precondition which defines requirement to *applr*. The postcondition of it is the following: if some system *R* which corresponds to a program *p* is applicable to a term *t*, then after stopping $s = (1 : R(t))$. If not then $s = 0$ after stopping. The high level of realization of *applr* strategy in APLAN language is *appl*:

$$appl := rs(p,q,r,t,pr)($$
$$(1:t) + r = (1:t),$$
$$(p+q,t,pr) + r = (p,t,new(pr)) + (q,t,pr) + r,$$
$$(p,t,pr) + r = perform(p,t,pr) + r$$
$$);$$

The information about functions *appls*, *applr*, *napplr*, *appl*, *perform, new* is represented in [15].

For execution of these conditions we should rebuild the result *s* after application of *appl* and to add non-deterministic application of RRS *appl* to *r*. So, let's consider non-deterministic rewriting strategy *nds_napplr* in APLAN language:

$$NAMES \quad nds\_applr, elm\_colon, nds\_appl, nds\_appls;$$
$$nds\_napplr := proc(t,p)loc(pr, Yes, s)($$
$$let(p, rs \quad pr \quad p);$$
$$s := (p, t \quad Nil, pr) + Nil;$$
$$appls(s, nds\_appl);$$
$$elm\_colon(s);$$
$$yes \to t := s$$
$$);$$
$$elm\_colon := rs(x,y)($$
$$x + y = elm\_colon(x) + elm\_colon(y),$$
$$x : y = y$$
$$);$$

$nds\_appl := rs(p,q,r,t,pr)($
$(1:t)+r = nds\_appls(1:t,\mathrm{r}),$
$(p+q,t,pr)+r = (p,t,new(pr))+(q,t,pr)+r,$
$(p,t,pr)+\kappa = perform(p,t,pr)+r$
);
$nds\_apps := proc(nds\_res,other)($
$appls(other,\mathrm{nds\_appl});$
$return(nds\_res+other)$
);

The optimization of ND Strategy is a very important thing, because if RRS is bigger then checking all of non-deterministic choices will be considerably slower. So, the main question is whether such ND strategy is optimal or not?

Theorem of optimization of ND Strategy. All non-deterministic simultaneously impossible cases will be eliminated from a consideration of *REM-programs* on a one step of interpretation with the help of hash operator of signature Ω.

Let's consider simultaneously impossible cases on a one step interpretation: different mark of a term *t* of set of terms with arity 0. But all of those cases will be in *hash* operator (it follows from its definition). It means that we choose only one case from all simultaneously impossible cases for current step of interpretation. So, the current realization of ND Strategy is optimal.

Semantics of Basic Protocols of VFS system

Each basic protocol is a Hoare triple $\alpha \to\!\!\prec P \succ\!\!\beta$, 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 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 the 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 is successfully terminated, the postcondition must be true [11].

Non-Deterministic Rewriting Strategy Application for Verification

Let $B_n = \{\alpha_i \to\!\!\prec P_i \succ\!\!\beta_i \mid i \in (1,...,n)\}$ be a set of basic protocols of a project, a predicate transformer $Tr(\alpha,\beta)$ is a function defined on formulae of the base language returning a new formula such that $Tr(\alpha,\beta) \to e$, e – define formulae. A predicate transformer strengthens the postcondition of a basic protocol by adding residual properties from the precondition.

We can represent a process of one-step application of each protocol from set B_n to formulae *e* by the ND Strategy and the following RRS:

$S_a = rs(e,u)($
$sat(e \wedge \alpha_1) \to \mathrm{e}[\mathrm{P_1}] = Tr(e \wedge \alpha_1, \beta_1),$
…,
$sat(e \wedge \alpha_n) \to \mathrm{e}[\mathrm{P_n}] = Tr(e \wedge \alpha_n, \beta_n)$
);

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

Other good example for application of ND Strategy in verification is based on experiments for effective term hashing algorithm realization on APS.

Let $e_n = \{e_i \mid i \in (1,...,n)\}$ be a set of formulae, $B_n = \{\alpha_i \to\!\!\prec P_i \succ\!\!\beta_i \mid i \in (1,...,n)\}$ be a set of basic protocols of a project. How we could determine the set of basic protocols $B_i \in B_n$ from which we could get state e_i. To determine the set of basic protocols B_i we can build the next RRS which should be dynamically update after each step of application of basic protocol:

$B_a := rs(x)($
$e_0(x) = B_{i_0},$
…,
$e_n(x) = B_{i_n}$
);

Conclusion

The system of algebraic programming APS exceeds the majority of criteria of well-known TRS. Among these criteria we can outline the most two important ones: the presence of procedural lan-

guage (allows using simultaneously the paradigms of declarative and imperative languages) and the commercial usage (usage of system in real big commercial products and not only in different researches).

The non-deterministic rewriting strategy is optimal and together with *REM-program* updating functions can be applied in different arias of APS and IMS systems applications.

1. *Algebraic* Programing System Site. – access mode: http://apsystem.org.ua
2. *Department* of Theory of Digital Automatic Machines of Glushkov Institute of Cybernetics of National Academy of Science of Ukraine. – http:// icyb.kiev.ua
3. *Laboratory* for Development and Implementation Pedagogical Software of Research Institute of Information Technologies of Kherson State Univeristy. – http://riit.ksu.ks.ua/ index.php?q=en/node/88
4. *Elan* System Official Site. – http://elan. loria.fr
5. *Maude* System Official Site. – http:// maude.cs.uiuc.edu
6. *Stratego* System Official Site. – http:// www.program-transformation.org/ Stratego/WebHome
7. *Insertion* programming / V.A. Volkov, A.A. Letichevsky, Y.V. Kapitonova et al. // Cybernetics and System Analysis. – 2003. – N 3. – P. 19–32.
8. *Insertion* modeling in distributed system design / A. Letichevsky, Y. Kapitonova, V. Kotlyarov et al. // Problems in Programing. – 2008. – N 4. – P. 13–39.
9. *Semantics* of Timed MSC Language / A. Letichevsky, Y. Kapitonova, V. Kotlyarov et al. // Cybernetics and System Analysis. – 2002. – N 4. – P. 3–14.
10. *Evidence* algorithm and problems of representation and processing of computer mathematical knowledge / A. Letichevsky, A. *Degtyarev,* J. Kapitonova et al. // Proc. of the International Workshop on Logic and Complexity in Comp. Sci. – Workshop. – University Paris 12, Creteil, France. – 2001. – P. 159–167.
11. *Systems* specification by basic protocols / A. Letichevsky, J. Kapitonova, V. Volkov et al. // Cybernetics and System Analysis. – 2005. – N 4. – P. 3–21.
12. *Yonat* Sharon Smart pointers – what, why, which? – http://ootips.org/ yonat/4dev/smart-pointers. html
13. *Letichevsky A*., *Khomenko V*. A rewriting machine and optimization of strategies or term rewriting // Cybernetics and System Analysis. – 2002. – N 5. – P. 3–17.
14. *Hanus M*., *Antoy S*. Set functions for functional logic programming // Proc. of the 11th ACM SIGPLAN conf. on Principles and practice of declarative programming. – 2009. – P. 73–82.
15. *Letichevsky A.A*., *Kapitonova J.V*., *Konozenko S.V*. Computations in aps // Theor, Comp. Sci. – 1993. – N 119. – P. 145–171.

Поступила 26.06.2012

Тел. для справок: +38 095 324-1557 (Киев, Херсон)
E-mail: *vladim@ksu.ks.ua*, *vladimirius@gmail.com*

●