УДК 004.41,004.51

APS and Tools

A. Letichevsky, A. Letichevsky (jr), V. Peschanenko
Glushkov Institute of Cybernetics NAS Ukraine, Ukraine
Research Institute of Information Technologies, Kherson State University

This article is devoted to comparison of functional possibilities and efficiency of the most widespread systems of terms rewriting and the system of algebraic programming APS. Also the article describes the new APS tools, intended for writing of C++ code as per APS prototype.

Key words: *functionality, systems of terms rewriting, algebraic programming, prototype, software tools.*

У статті описані основні відмінності функціональних можливостей та ефективності найбільш розповсюджених систем переписування термів та системи алгебраїчного програмування APS. Також розглянуто основні засоби розробки програм в системі APS за його прототипом.

Ключові слова: *функціональність, система переписування термів, алгебраїчного програмування, прототип, засоби розробки програм.*

В статье описаны основные отличия функциональных возможностей и эффективности наиболее известных систем переписывания термов и системы алгебраического программирования APS. Также рассмотрены основные инструменты разработки программ APS по ее прототипу.

Ключевые слова: *функциональность, система переписывания термов, алгебраическое программирование, прототип, инструменты разработки программ.*

1. Introduction

APS - is the system of algebraic programming that has been developed in the mid-eighties in departments of the Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine [1]. Historically, APS is the first system which has started to use the technology of term rewriting in combination with user's defined strategies of rewriting. APS is the system which supports a run-time self-modification (these are possibilities that make APS more useful for different kinds of algorithms).

The reason of its week distribution was unstable economic situation in Ukraine and a number of deficiencies in the first versions of APS (memory leaks and a sluggishness of interpreter of APS). At this time the closest analogs of APS started to appear, such as: Stratego [2], Maude [3], Elan [4]. As a result the APS has been forgotten and the majority of designers use now other systems of terms rewriting.

Now the company LitSoft [4] is the official distributor of the algebraic programming system APS. It is possible to find all necessary information about the APS system on the site of this company. Also on the site it is possible to receive the free version of the present system (the binary files for Windows and Unix operating systems) along with the examples, which are carefully selected by the authors. Besides, on this site it is possible to try online working of the system, i.e. it is possible to write APLAN (language of APS system) code and to launch its applications in the Internet.

So, article is devoted comparison of functionality of known rewriting term systems with APS system.

2. Functional possibilities of rewriting term systems

Let's demonstrate the functional possibilities of each system (see Table 1). It is clear from the table that APS doesn't concede to well-known systems of terms rewriting as per all criterions.

Table 1. Functionalities of rewriting term systems

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 System of the Rewriting Rules	Support	Application Area	Commercial Products	Country
1	ELAN	arbitrary	-	-	+	1992	-	+,-	-	+		*	France
2	STRATEGO	arbitrary	+	-	-	1994	-	+,-	+	-	transformation systems including compilers, interpreters, static analyzers, domain-specific optimizers, code generators, source code refactorers, documentation generators, and document transformers.	*	Netherlands
3	MAUDE	7	+	-	-	1995	-	+,-	-	+	General Logics and Logical Frameworks, Specification Languages, Declarative Programming Languages, Semantics of Programming Languages and Models of Computation, Concurrent and Distributed Systems, Formal Tools and Formal Interoperability, Reflection and Metaprogramming, Object-Oriented Modeling and Programming, Real-Time Systems, Bio Informatics, Mobile Languages, Network Protocols and Active Networks	*	USA
4	APS	arbitrary	+	+	+	1987	**	***	+	+	Algebraic programming, Insertional modeling, program transformation, General Logics and Logical Frameworks, Specification Languages, Declarative Programming	VRS (Verification of Requirement Specification), TERM(School System of Computer Algebra)	Ukraine

* - 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.

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 in table 2.

Without doubts due to quite developed typification in the system, MAUDE has more benefits than other systems (the process of evaluation with integers numbers doesn't use some additional structures etc). In this connection, there is a quite limited number of rewriting strategies in the system that considerably complicates the algorithms realized in it.

Table 2. The results of launching of algorithms finding of Fibonacci n-number.

№	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

From the other side, APS was considered to be one of the slowest systems of term rewriting. Talking into consideration the results of capacity of rewriting, it can be said that APS is 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 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).

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 [6]. Thus, by means of this technology in the second version of APS system it was possible to be saved of this deficiency.

In the course of analysis of the source code of APS system and the experience of its usage we have come to the important conclusion - the slowest part of APS is the rewriting interpreter of APLAN language. This is a quite appropriate conclusion, as the APS supports the ideology of self-variation of the program written in APLAN language during program interpreting. As the interpreter of rewriting machine is written in C ++, the functions of rewriting machine shall work fast - system of rewriting rules with strategies will translate in C++ procedures, interpreter of rewriting machine will be optimized.

3. APS Tools

Application development process in APS passes the following stages: prototype development in APLAN language, analysis of this prototype and realization of the final version in C++. Therefore having saved of memory leaks in the system, we have prepared it for industrial usage. For industrial usage of APS system we have created a set of tools for reduction of terms of transference from the prototype of the program to its final version the usage of which together with APS system is going to reduce considerably the software development terms.

The tools of APS include:

1. The Language APLANC (**A**lgebraic **P**rogramming **Lan**guage **C**++).
2. The converter from APLAN language to APLANC language.

The Language APLANC represents a set of functions being realized in C++ which help working with internal structures of APS, as well as to realize the most frequently used procedures of APS, and also the functions of usage of the rewriting machine. Experience of usage of APLANC on a number of commercial products such as VRS (Verification of Requirements Specification) [7], MathLog (Mathematic Logic for Universities) [8] displays that its usage significantly simplifies a view of the source code, considerably reduces the development of terms and thus is simple enough in usage. The slow work of the program using APLANC in comparison with the program which doesn't use APLANC can be considered as a deficiency. However as we will demonstrate further this retardation is not significant.

So, the language APLANC includes:

1. Function **input_aplan**.
2. Functions **make_formula**, **make_hash_formula**.
3. Function **let**.
4. Functions **applr, appls, nbt, ntb**.

All enumerated above functions of C ++ are described in class FPL (Formula Processing Language). Therefore further in the article we will understand under the object fpl the class FPL exemplar. Let's examine in details each of the enumerated above functions.

3.1 Input_aplan function

The input_aplan function is used for import of data into the current interpreter in APLAN language syntaxes (it can be transferred from the parts of ar-module [9] into the arguments of this function). Syntaxes:

```
int input_aplan(const char *ccap);
```

The function returns 1 if the data have been successfully imported into the current interpreter, and 0 in the opposite case, in the line ccap the text is transferred in APLAN syntaxes. The result of fulfillment of this function for the interpreter will be the same as if this text would be described somewhere in ar-modules.

The next instance (Instance 1) demonstrates the import of rewriting rule *rsGCD* (search of greatest common devisor) into the interpreter and receipt of indicator to the top of the tree, that was describing this rewriting rule:

fpl.input_aplan ("NAME *rsGSD*; *rsGSD*:=rs(x,y)((0,x)=x,*isnum*(x)->((x,y)=(y *mod* x,x)))"); - import of data *fpl.find_name* ("*rsGCD*"); - search of created name (see [9] chapter "names").

At calling of the first function of the instance the name *rsGSD* will be created in the interpreter and a value for it (the right part of the sign :=) will be specified. The function *isnum* returns 1 if the tree x – is a number and 0 in the opposite case, it should be determined in C++. The possibility of calling at application of rewriting rules system to the tree, functions, written in C++ shall be foreseen. (see further).

3.2. Make_formula,make_hahs_fomrula functions

Functions *make_formula*, *make_hash_formula* are used for construction of trees in syntaxes of APLAN language. Unlike the previous function it returns a tree by incoming line, which will be completely identical at printing to the incoming line. That is, if we use the function arguments from the previous example (*input_aplan*), the line NAME *rsGCD* will be the left argument of the top «;». Syntaxes:

```
node_ptr make_formula(const char *ccap,int n,...);
node_ptr make_hash_formula(int hash,const char *ccap,int n,...);
```

Functions return the tops of constructed tree, the parameter *ccap* is a line according to which a tree will be constructed. In the line such sub lines as «()» can be met, instead of each incoming of such sub line from right to left the values listed after the parameter *n* – quality of substitutes will be substituted. The second function form the fist one differs by one parameter *hash* – that is hash number, which is used in order not to construct repeatedly a tree according to the line. Here *node_ptr* – is a class of smart indicators at the class of knots of trees of APS node. That is, class *node* inherits the class of counter of references, and class *node_ptr* is determined as *typedef CSmartPtr<node> node_ptr* [6].

The next instances (instance 2) show the work with these functions:

```
node_ptr x = make_formula("a-b",0);
node_ptr y = make_formula("c-d",0);
node_ptr z = make_formula("()+2+()",2,*x,*y);
```

The first and the second instances build trees without substitution, the third example builds a tree and instead of two incomings «()» substitutes the values. The following formula: *(a-b)+2+(x-d)* will be as a result in *x*.

3.3. Let function

The function *le t* is used for simultaneous check of the tree structure and is used for receipt of the necessary sub trees of this tree if this structure coincides. Syntaxes:

```
int let(node_ptr &t,const char *ccap);
int let(node_ptr &t,const char *ccap,nodes_ptr args);
int let(node_ptr &t,string shash,const char *ccap,nodes_ptr args);
```

t – is incoming tree, value *ccap* is the same as in function *make_formula*, structure *nodes_ptr* is a array of objects *node_ptr*, *args* – is a structure in which the necessary sub trees will be saved, *shash* (STL string) – is a hash key, which is required in order not to build new tree as per the line *ccap*.

Filling of *args* occurs in the following way: each occurrence of key name *ac_h* (APLANC here) in *ccap* line is a signal that it is necessary to save a new tree with *args*. If let is executable at the present tree, the key name yes will be equal to 1 and function returns 1, otherwise – 0 and function returns also 0. Besides, if *yes*=1, in *args* it will be saved so much corresponding sub trees as there were occurrences of the name *ac_h* in line *ccap*.

Let's examine an instance of application of let function in practice. (Instance 3):

```
 node_ptr x = make_formula("x+y+2+3",0);
nodes_ptr args;
if (let("hash_add_op",x,"_+_+ac_h+ac_h",args)){
  prn(args.size());
  prn(args[0]);
  prn(args[1]);
}
```

Symbol «_», which occurs in *let* function of the instance means that it is not necessary to save the corresponding tree. Function size returns the sizes of array. As a result of execution of this code, we will see on the screen a column of numbers 2,2,3.

3.4 applyr,applys,nbt,ntb functions

Functions *applyr*, *applys*, *nbt*, *ntb* are used for access to rewriting machine from C++ language. These functions are strategies which realize different modes of application of rewriting rules systems to the given term [9]. Syntaxes:

```
int applyr(node_ptr &t,node_ptr &s);
int applys(node_ptr &t,node_ptr &s);
int nbt(node_ptr &t,node_ptr &s);
int ntb(node_ptr &t,node_ptr &s);
```

where are *t* – is a tree, to which it is necessary to apply rewriting rules system, *s* – is a tree, which sets the rewriting rules system (i.e a tree structure should look like: rs(…)(…)) . More details about the correctness of describing of the rewriting rules system you can see here [9].

So, *applyr* – is a strategy which applies the rewriting rules system *s* to the source tree *t* one time. If the rewriting rules system has been applied the internal name *yes* will be equal to 1, otherwise – 0.

applys – is a strategy, which applies the rewriting rules system *s* to the source tree *t* until it is not applicable. If the system has been applied at least once, internal name *yes* will be equal to 1, otherwise – 0.

nbt – is strategy which realizes the bypass of the source tree bottom-up and applies the strategy *applys* in each knot of the tree. If the rewriting rules system has been applied at least once internal name *yes* will be equal to 1, otherwise – 0.

ntb- a strategy is determined in the same way as *nbt*, except the fact that it realizes bypass of the tree up-bottom.

Let's examine the instances of application of each strategy (example 4). For this purpose we will use the rewriting rules system *rsGCD* from the instance 1. So,

```
node_ptr nm_val = fpl.find_name("rsGCD").val();
node_ptr x = fpl.make_formula("12,16",0);
fpl.applyr(x,nm_val);
fpl.applys(x,nm_val);
node_ptr y = fpl.make_formula("4,12,16",0);
fpl.nbt(y,nm_val);
node_ptr z = fpl.make_formula("(4,12),16",0);
fpl.ntb(z,nm_val);
```

Let's examine in details each function of this example. Function *find_name* – returns the structure of APS – name (as we used *inpute_aplan* from the first instance, such name will be found). Next, the function *val* returns the meaning of such name, i.e. directly the rewriting rules system itself. Description of function *make_formula* was examined above. After the application of the strategy *applyr* it will be the following tree in *x*: *4,12* and after *applys* – 4. Next, with the help of the strategies *nbt*, *ntb* the greatest common divisor for the sequence of numbers can be found. The only difference for both of these strategies is incoming trees: for *nbt* – right-side (in APS by default the right- side trees are used, although this process is governed [10]), and for *ntb* – left-side ones.

For the right work of the rewriting rules system *rsGCD* described above, the canonizer of *mod* operation is required. In APLAN language this operation is determined with the help of type – mark [9]. In order to realize the canonizer of the mark to C++ it is necessary:

1. to realize the function with the following title int can_name(clew_ptr &cl,node_ptr &arg), where clew_ptr is a class of smart indicators for working class with the basic data structures of APS. Clew, arg – a tree occurs here at calling of this function, the main top of which is a mark with this canonizer.
2. to install this canonizer with the help of function set_info, class of marks of APS mark for the required mark.

For instance (instance 5):

```
 int add_can(clew_ptr &cl,node_ptr &arg){
...
}
...
fpl.find_mark("ADD").set_info(add_can);
```

Function *find_mark* effects search of the mark with name *ADD* (it is meant in this example that the mark will be always found). Function *set_info* sets canonizer for this mark.

Except the canonizers of marks it is necessary to foresee the possibility of calling of C++ functions at application of the rewriting rules system. For realization of such functions it is required:

1. to realize the function with the following title: *int func_name(clew_ptr &cl,node_ptr &nd)*. Calling of this function will work as per the rules of the interpreter of APLAN language [9].;
2. to add to the corresponding record to the structure *FuncsDecript* – array of pairs of view: *string name*, name c++ function.

For instance (instance 6):

```
int isnum(clew_ptr &cl,node_ptr &nd){
...
}
const stFuncsDecript FuncsDescripts [] =
{
...
 {"isnum", isnum}
};
```

4 Converter from APLAN language to APLANC language.

Conversion process of the programs written in terms rewriting systems to the language of their realization is quite difficult task. In many systems of terms rewriting listed above such converters exist and they support only very limited part of the language. It causes inefficiency of their usage in general case, as in quite big programs many codes have to be transferred manually in any case, because to write programs taking into consideration this language limitation only is also quite difficult task.

As the slow work of APS shows its worth at working with procedural and rewriting parts of APS, the aim of our converter is to build the source code of C++ program in such a way that at starting of the program the interpreter of APS wouldn't be used. That is we would like to build C++ code as per APS program or as per some of its part, which is able to use the APLANC language and optimized rewriting machine (or this code will be converted in a procedures in C++). But the realization of C++ functions which were used by the interpreter of APS program, is entrusted to the user. The speed of performance of procedures written on APLANC in 10 times exceeds speed of application of rewriting rules systems.

Actually it means the formation of some list of ready realized functions of C++ of APS system, which the user has to transfer himself from the source code of APS to the source code of the program.

At the present stage of the realization of the converter of APS we have made enough experiments in order to lay dawn the requirements to the APS converter. They are the following:

1. The source code of the system should depend only on FPL.
2. The source code of the system should use the language APLANC.
3. All necessary canonizers of marks should be described by the user.
4. Converter should generate the source code in such a way in order not to put the changes by the user, but in order the user could put the separate parts of this code into the existing program.

At present in order to finish the final version of converter prototype it is lack filling of different APLAN language operators. As to the difficulties, this work is not complicated as the major part of procedural possibilities of APS is ready.

Next we are to determine the several modes of source code writing in APS in order to understand, whether it is worth using APS tools for development and what kind of results we will receive.

So, in the process of transference from program prototype to its final version it is possible:

1. to build the source code manually, i.e. without the use of APS tools;
2. to use APLANC language without the converter;
3. to use the converter.

Let's examine theses modes of construction of program final version according to its prototype on the following instances:

1. Construction of disjunctive (DNF) and conjunctive (CNF) normal forms for the set expression.
2. Operations of insert (INSERT) and removal (REMOVE) of the rule into the rewriting rules system without its re-compiling in REM language [11] (description of this algorithm goes beyond the discussion of the present paper and will be described in the next publications of the APS authors).

Thus, the results of the experiment, being made by the APS authors are represented in the following table 3

Table 3. Experiment results

№	Algorithm's name	Lines of the code (psc)	Manual construction (man/hour)	APLANC without the converter (man/hour)	APS tools (man/hour)
1	DNF+CNF	60	3	1,5	0,5
2	REMOVE+INSERT	160	16	8	2

5 Conclusion

The system of algebraic programming APS exceeds the majority of criterions of well-known systems of terms rewriting. Among these criterions we can outline the most two important ones: the presence of procedural language (allows using simultaneously the paradigms of declarative and imperative languages) and the commercial using (using of system in real big commercial products and not only in different researches).

The using of tools designed for APS allows the transfer from writing of the prototype in the system itself easily and quickly to its final realization on C++ language.

More detailed information about the process of transfer from the prototype to the product will be described in the next publication of the authors: "APS: The transfer from the prototype to the product".

REFERENCES

1. Glushkov Institute of Cybernetics [http://www.icyb.kiev.ua].
2. Stratego [http://www.program-transformation.org/Stratego/WebHome].
3. Maude [http://maude.cs.uiuc.edu].
4. Elan [http://elan.loria.fr/].
5. LitSoft [http://www.litsoft.org].
6. Smart Pointers - What, Why, Which? [http://ootips.org/yonat/4dev/smart-pointers.html].
7. A.A. Letichevsky, Yu. V. Kapitonova, V.A. Volkov, A.A. Letichevsky jr, S.N. Baranov, V.P. Kotlyarov, T. Weigert Systems Specification by Basic Protocols [http://portal.acm.org/citation.cfm?id=1103712.1103717&coll=&dl=].
8. Pedagogical Software Mathematical Logic [http://www.ksu.ks.ua/downloads/LabRVPPZ/MathLog_eng.htm].
9. APS user manual [http://aps.ksu.ks.ua/files/APS.user%20manual.pdf].
10. V.S Peschanenko Using Algebraic Programming System APS for Algebra Teaching Support System in School // Control System and Machines (in russian). − 2006. − №4. − C. 86−94.
11. A.A. Letichevsky, V.V. Chomenko Rewriting Machine and Optimization of Terms Rewriting Strategies (in russian) [http://aps.ksu.ks.ua/files/8.zip].

Надійшла 29.07.2009.