STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

FUNCTIONAL AND RELATIONAL PROGRAMMING WITH PSP

Ilie PARPUCEA* and Bazil PÂRV**

Received February 26, 1994
AMS subject classification 68Q05

REZUMAT. - Programare funcţională şi relaţională cu PSP. Articolul prezintă PSP (Procesorul Simbolic Poisson), într-o manieră ce unifică programarea relaţională cu clauze Horn bazate pe predicate cu programarea funcţională bazată pe egalităţi Unificarea pleacă de la o logică minimală, ce posedă atât clauze Horn cât şi egalităţi, numită logica clauzelor Horn cu egalităţi În ipotezele teoremei Church-Roser, semantica operaţională a PSP constituie o logică completă Semantica se bazează pe unificarea a două abordări. una construită pe baza teoriei modelelor, care foloseşte relaţia de satisfacţie între modele şi instrucţiuni, şi una bazată pe teoria demonstrării, care foloseşte relaţia de partiţionare (entailment) între mulţimi şi instrucţiuni PSP posedă tipuri abstracte de date ce se pot defini de utilizator şi care pot fi considerate module generice (parametrizate) Cu ajutorul subsorturilor se pot introduce operatori polimorfici şi o relaţie de moştenire pe tipurile de date Toate aceste caracteristici concură la definirea riguroasă a semanticii cu ajutorul logicii substrat, ilustrată cu câteva exemple

1. Introduction. A main feature of the processor described in this paper, hereafter called PSP, is the practical way in which it unifies relational programming with functional one, by unifying the logics that underlie relational and functional programming, namely first order Horn clause logic and many-sorted equational logic, to get many-sorted first order Horn clause logic with equality [8] In addition, generic modules are available with a rigorous logical foundation, and PSP also has a subsort facility that greatly increases its expressive power

PSP is intended to operate with Poisson series, which are a well-known tool in

* "Babeş-Bolyai" University, Faculty of Economic Sciences, 3400 Cluj-Napoca, Romania

** "Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania

expressing celestial mechanics problems The motion of celestial bodies is described by means of differential equations, in which the right-hand-side terms are in fact Poisson series Usually, the solution of these differential equations cannot be obtained in exact form There are two alternatives numerical integration or analytic construction of an approximate solution (known as "theory of motion") First was used extensively, being a "classical" solution of motion problems The second alternative seems to be more attractive, because one can obtain the solution in analytical form, which provide a qualitative study of motion There are many analytical methods for constructing the approximate solution of differential equations, most of them known as "perturbation theory" methods [2, 14]

The advantages claimed for PSP includes simplicity, clarity, understandability, reusability and maintanability There is another requirement that we argue also be imposed on our symbolic processor every program should have an initial model [10, 12] An initial model is characterized, uniquely up to isomorphism, by the property that only what is provable is true, and everything else is false The initial model provides a foundation for database manipulations, since you know exactly is true

We have found that neither of the approaches, the model-theoretic and the proof-theoretic one, is by itself sufficient to axiomatize our PSP The model-theoretic approach focuses on the satisfaction relation

$$M \models \gamma$$

between a model M and a sentence γ, and the proof-theoretic one tries to axiomatize the entailment relation

$$\Gamma \vdash \gamma$$

between a set of sentences Γ and a sentence γ derivable from Γ The model-theoretic approach is exemplified by Barwise's axioms for abstract model theory [1] The framework of institutions, given by Goguen and Burstall [6, 7] also belongs to this approach The proof-theoretic approach has a long tradition, dating back to work of Tarski [15] on "consequence relations", and of Hertz and Gentzen on the entailment relation $\vdash$.

This paper proposes a practical approach that integrates the two above-mentioned ones (model-theoretic and proof-theoretic aspects) into a single axiomatization The axiomatization in question consists of an "entailment system", specifying an entailment relation $\vdash$, together with a "satisfaction system" (specifically, an institution in the Goguen-Burstall sense), specifying a satisfaction relation $\models$ [11] The entailment and satisfaction relations are then linked by a soundness axiom

The entailment relation $\vdash$ says nothing about the internal structure of a proof To have a satisfactory account of proofs, we use the additional concept of a proof calculus C for a L The same logic may have, of course, many different proof calculi When we wish to include a specific proof calculus as part of a logic, the resulting logic plus proof calculus is called logical system The axioms for a proof calculus C state that each signature in the logic L has an associated space of proofs, which is an object of an appropriate category. From such a space we can then extract an actual set of proofs supporting a given entailment $\Gamma \vdash \gamma$

In order to obtain some efficiency with respect to PSP, we use the more general concept of proof subcalculus, where proofs are restricted to some given class of axioms and conclusions are also restricted to some given class of sentences It is by systematically exploiting such restrictions that the structure of proofs can be simplified In this way, we can obtain efficient proof theories, which lead to the theoretical concept of variable operational semantics

2. The features of PSP Conceptual clarity and ease of understanding are facilitated by breaking a program into modules This in turn offers support for debugging and reusability When there are many modules, it is helpful to design the structure of module dependencies in an hierarchical manner Whenever one module (client module) uses data (state) or operations (services) declared in a second one (server module), the server must be explicitly imported to the client and also must be defined earlier in the program text A program obtained in this way has the abstract structure of an acyclic graph with modules as vertices and the module dependencies as edges

A PSP program is a sequence of modules (objects) Each module may define one or more new data sorts, together with associated operations that may create, select, interrogate, store, or modify data Such an module may use existing modules with their sorts of data and operations The module concept includes both data types in the programming language sense (that is, a domain of values of variables together with operations that access or modify those values) and algorithms

PSP has the following syntax for import

<**importing**> <mod_list>,

where **importing** is keyword and <mod_list> is a list o module names By convention, if a module M imports a module M', that imports a module M", then M" is also imported into M, that is, "importing" is a transitive relation

Usually, programming systems provide a number of built-in data types, for example numbers and identifiers PSP has the following built-in modules BOOL, NAT, INT, and RAT BOOL provides the expected syntax and semantics for Booleans NAT, INT, and RAT define natural, integer and rational numbers (the last ones from the integers)

There is much work on providing user-defined abstract data types in programming languages (e g [3, 4, 9]) The essential idea is to allow users to introduce models that define new sorts and their associated functions and give axioms in Horn clause logic with equality or rules of computation It can also be very helpful to have available subsorts and their associated predicates, as we will see later

Note that PSP keywords are written in **bold**, module names are all CAPITALS, while variable names begin with a capital letter and that relation, function and constant names are all lowercase Attributes can be given for operators, for example, **assoc**, **comm**, and **id** indicate that a binary operator is associative, commutative, and idempotent, respectively

PSP mix-fix notation allows any desired ordering of keywords and arguments for operators, this is declared by giving a syntactic form consisting of a string of keywords and underbar character "_" followed by a " ", followed by the arity as a string of sorts, followed by "->", followed by the value sort of the function Similar conventions are used for predicates An expression is considered well-formed in this scheme iff it has exactly one parse, the parser can interactively help the user to satisfy this condition

PSP operates with Poisson series, which are of the form

$$S = \sum_{i=0}^{\infty} C_i \, y_1^{j_1} y_2^{j_2} \quad y_m^{j_m} \, {}^{\sin}_{\cos}(k_1x_1+k_2x_2+ \quad +k_nx_n),$$

where C_i are numerical coefficients, y_1, y_2, $,y_m$ are monomial variables, x_1, x_2, $,x_n$ are trigonometric variables, j_1, j_2, $,j_m$ and k_1, k_2, $,k_n$ are exponents, and, respectively, coefficients, the summation index i covers the set of all possible combinations of the exponents j and coefficients k ($j \in \mathbf{Z}^m$, $k \in \mathbf{Z}^n$, $\mathbf{Z}$ being the set of integers)

In a concise form we write (1) as follows

$$S = \sum_{i=0}^{\infty} T_i,$$

in which T_i is a term of this series

$$T_i = C_i\, F_i\, P_i\,,$$

where the polynomial part P_i has the form

$$P_i = y_1^{j_1} y_2^{j_2} \;\; y_m^{j_m},$$

while the trigonometric part F_i is

$$F_i = {\sin \atop \cos}(k_1 x_1 + k_2 x_2 + \;\; + k_n x_n)$$

In practice, one does not operate with Poisson series, but with partial sums of these ones, called Poisson expressions, of the form

$$S = \sum_{i=0}^{N} T_i, \quad N \in N$$

The Poisson expression can be defined in an hierarchical way The complete specification of trigonometric and polynomial part of a Poisson term (Ttr, and Ppol, respectively) can be found in [13] Now we define the Poisson term as following

```
psp  TERM is
       importing Rat Ttr Ppol
       sorts Rat Ttr Ppol Term
       op
            _·_·_  Rat Ttr Ppol -> Term [assoc comm]
       _=_   Term -> Bool
```

```
        vars
              X  Rat
              Y  Ttr
              Z  Ppol
              X·Y·Z X'·Y'·Z'   Term
        eq
              0·Y·Z = 0
              X·0·Z = 0
              X·Y·0 = 0
              1/1·Y·Z = Y·Z
              X·1·Z = X·Z
              X·Y·1 = X·Y
              X·Y·Z = X'·Y'·Z'  - X = X', Y = Y', Z = Z'
  endpsp.
```

The above keyword **importing** indicates that the sorts, subsorts, predicates, functions, and axioms of the listed models are imported into the module being defined The equation

X·Y·Z = X'·Y'·Z' - X = X', Y = Y', Z = Z'

is a Horn clause with equality, where "=" represents equality predicate defined on types, respectively

In the same way, we define EXP, that is based upon TERM, and specify the Poisson expression, viewed as a list of terms, in which the symbol "," is separator

```
  psp EXP is
        importing Term
        sorts Term NeExp Exp
        subsorts Term < NeExp < Exp
     op
              _+_  Term Term -> Exp [assoc comm id 0]
        _-_  Term Term -> Exp [id 0]
        _*_  Term Term -> Exp [assoc comm id 1]
              _ _  Exp Exp -> Exp [assoc id nil]
              _==_  Exp Exp -> Bool
        head_  NeExp -> Term
        tail_  NeExp -> Exp
        empty?_  Exp -> Bool
     vars
        T  Term
```

E Exp

$N/M \cdot \left\{ \begin{matrix} \sin X_1 \\ \cos X_1 \end{matrix} \right\} \cdot Y_1$ Term

$P/Q \cdot \left\{ \begin{matrix} \sin X_1 \\ \cos X_1 \end{matrix} \right\} \cdot Y_1$ Term

$P/Q \cdot \left\{ \begin{matrix} \sin X_2 \\ \cos X_2 \end{matrix} \right\} \cdot Y_2$ Term

eq

$$N/M \cdot \left\{ \begin{matrix} \sin X_1 \\ \cos X_1 \end{matrix} \right\} \cdot Y_1 \pm P/Q \cdot \left\{ \begin{matrix} \sin X_1 \\ \cos X_1 \end{matrix} \right\} \cdot Y_1 =$$

$$= (N/M \pm P/Q) \cdot \left\{ \begin{matrix} \sin X_1 \\ \cos X_1 \end{matrix} \right\} \cdot Y_1$$

$$(N/M \cdot \cos X_1 \cdot Y_1) * (P/Q \cdot \sin X_2 \cdot Y_2) =$$

$$=((1/2 * N/M * P/Q) \cdot \sin(X_1+X_2) \cdot Y_1 \cdot Y_2\,,$$
$$(1/2 * N/M * P/Q) \cdot \sin(X_2-X_1) \cdot Y_1 \cdot Y_2)$$

$$(N/M \cdot \sin X_1 \cdot Y_1) * (P/Q \cdot \sin X_2 \cdot Y_2) =$$

$$=((1/2 * N/M * P/Q) \cdot \cos(X_1-X_2) \cdot Y_1 \cdot Y_2\,,$$
$$-(1/2 * N/M * P/Q) \cdot \cos(X_1+X_2) \cdot Y_1 \cdot Y_2)$$

$$(N/M \cdot \cos X_1 \cdot Y_1) * (P/Q \quad \cos X_2 \cdot Y_2) =$$

$$=((1/2 * N/M * P/Q) \cdot \cos(X_1+X_2) \cdot Y_1 \cdot Y_2\,,$$
$$(1/2 * N/M * P/Q) \cdot \cos(X_1-X_2) \cdot Y_1 \cdot Y_2)$$

head(T E) = T
tail(T E) = E
empty?_E = E == nil

endpsp.

In addition, we define two modules for differentiating and integrating of Poisson expressions

psp DERIV **is**
importing Exp
sorts Term Exp NeExp
subsorts Term < NeExp < Exp
op

$\frac{\partial}{\partial\_}\_$ Term Set --> Exp

$\frac{\partial}{\partial\_}\_$ Exp Set --> Exp

vars
E Exp
T Term

$$N/M \cdot \left\{ \begin{matrix} \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_l \cdot X_l) \\ \sin(N_1 \cdot Y_1 + N_2 \cdot Y_2 + \quad + N_l \cdot X_l) \end{matrix} \right\} \cdot Y_1^{M1} \cdot \quad \cdot Y_h^{Mh} \cdot \text{ Term}$$

eq

$$\frac{\partial}{\partial Y_k} \left(N/M \cdot \left\{ \begin{matrix} \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \\ \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \end{matrix} \right\} \cdot Y_1^{M1} \cdot \quad \cdot Y_k^{Mk} \cdot \quad \cdot Y_h^{Mh} \right) =$$

$$= (N^* M_k / M \cdot \left\{ \begin{matrix} \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \\ \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \end{matrix} \right\} \cdot Y_1^{M1} \cdot \quad \cdot Y_k^{Mk-1} \cdot \quad \cdot Y_h^{Mh},$$

$$\mp N^* N_k / M \cdot \left\{ \begin{matrix} \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \\ \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \end{matrix} \right\} \cdot Y_1^{M1} \cdot \ldots \cdot Y_k^{Mk} \quad \cdot Y_h^{Mh})$$

$$\frac{\partial}{\partial Y_k}(0) = 0$$

$$\frac{\partial}{\partial Y_k}(\text{nil}) = 0$$

$$\frac{\partial}{\partial Y_k}(E) = \frac{\partial}{\partial Y_k}(\text{head } E) + \frac{\partial}{\partial Y_k}(\text{tail } E)$$

endpsp.

In the specification of INTEG module given below, we use the following abreviations

$$I_p \;\equiv \int N/M \cdot \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$
$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{p} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \; dY_k \,,$$

and

$$J_p \;\equiv \int N/M \cdot \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$
$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{p} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \; dY_k \,,$$

where p Int, p≠-1 (the case p=-1 does not preserve the form of Poisson expressions, because the integration leads to logarithms)

psp INTEG **is**
importing Exp
sorts Term Exp NeExp
subsorts Term < NeExp < Exp
op
∫_d_ Term Set --> Exp
∫_d_ Exp Set --> Exp
vars
E Exp
T Term
P Nat
$N/M \cdot \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \;.+ N_l \cdot X_l) \cdot Y_1^{M_1} \cdot \quad \cdot Y_h^{M_h}$ Term
$N/M \cdot \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_l \cdot X_l) \cdot Y_1^{M_1} \cdot \quad \cdot Y_h^{M_h}$ Term
eq

$$I_0 = (-1/N_k * N/M) \cdot \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$
$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{0} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h}$$

$$I_1 = ((-1/N_k * N/M) \cdot \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$
$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{1} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \,,$$

$$(1/(N_k * N_k)) \cdot \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$
$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{0} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \;)$$

$$I_p = ((-1/N_k * N/M) \cdot \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$

$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{p} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \ ,$$

$$(N/M*_p/(N_k*N_k)) \cdot \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$

$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{p-1} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \ ,$$

$$-(N/M*p/N_k*(p-1)/N_k) \cdot I_{p-2}\ ) \ - p > 1$$

$$J_0 = (1/N_k*N/M) \cdot \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$

$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{0} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h}$$

$$J_1 = ((1/N_k*N/M) \cdot \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$

$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{1} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \ ,$$

$$(1/(N_k*N_k)) \cdot \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$

$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{0} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \ )$$

$$J_p = ((1/N_k*N/M) \cdot \sin(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$

$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{p} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \ ,$$

$$(N/M*p/(N_k*N_k)) \cdot \cos(N_1 \cdot X_1 + N_2 \cdot X_2 + \quad + N_k \cdot Y_k + \quad + N_l \cdot X_l) \cdot$$

$$\cdot Y_1^{M_1} \cdot \quad Y_{k-1}^{M_{k-1}} \cdot Y_k^{p-1} \cdot Y_{k+1}^{M_{k+1}} \cdot \quad \cdot Y_h^{M_h} \ ,$$

$$-(N/M*p/N_k*(p-1)/N_k) \cdot J_{p-2}\ ) \ - p > 1$$

$$\int 0 \, dX_k = 0$$
$$\int (\mathrm{nil}) \, dX_k = 0$$
$$\int E \, dX_k = \int \mathrm{head}(E) \, dX_k + \int \mathrm{tail}(E) \, dX_k$$

endpsp.

The NORMAL module provides a normal form of Poisson expressions

```
psp NORMAL is
importing Exp
sorts Term Exp
subsorts Term < Exp
op
  normalised_   Exp -> Bool
  normalising_   Exp -> Exp
vars
  T T'   Term
  E E'   Exp
eq
  normalised(nil) = True
  normalised(T) = True
  normalised(2T E)  - T = T', normalised(T' E)
  normalising(E T T' E') = normalising(E 2T E')  - T=T'
  normalising(E) = E  - normalised(E)
endpsp
```

The basic building blocks of parameterized programming are parameterized modules Parameterized programming is a powerful technique for the reliable reuse of software In this technique, modules are parameterized over very general interfaces that describe what properties of an environment are required for the module to work correctly.

Here is an example of a parameterized module, intitulated SUBST, over the theories SORT1 and SORT2 In our example, SUBST module provides the symbolic substitution operation

```
psp SUBST [S1    SORT1 , S2    SORT2] is
    importing Exp
    sorts Term Exp NeExp
    subsorts Term < NeExp < Exp
    op
      _sub_ _    Term S1 S2 -> Term
      -sub_ _    Exp S1 S2 -> Exp
    vars
      E   Exp
      T   Term
      X   S1
      Y   S2
```

```
  eq
    sub(0 X Y) = 0
          sub(nil X Y) = nil
          sub(T X Y) = sub(T X -> Y)
          sub(E X Y) = sub(head E X Y) + sub(tail E X Y)
endpsp.
```

where the meaning of expression sub(T X -> Y) is all occurences of the symbol X are replaced by the symbol Y in term T SORT1 and SORT2 are theories defined as follows

```
Th SORT1 is                          Th SORT2 is
  sorts Sor1                       sorts Sor2
endth.                               endth.
```

The following specification

```
view SUBS is (Sor1 as Rat
                        Sor2 as Set)
```

define a view called SUBS, mapping from the sorts of SORT1 and SORT2 to the other sorts already defined, that preserves the subsort relation, and a mapping from the operations of SORT1 and SORT2 to the operations of Rat and Set, preserving arity, value sort, and attributes

To actually use a parameterized module, it is necessary to instantiate it with an actual parameter The **Make** command applies a parameterized module to an actual one, by use of a **view** For example,

```
Make SUBSTITUTION is SUBST[SUBS] endm.
```

uses the view SUBS to instantiate the parameterized module SUBST with the actual parameters Rat and Set

In the same way, one can construct new PSP modules, which implements new operations on Poisson series, like power expansion (including exponents integer numbers or rational numbers of the form 1/M or M/2 with M nonzero integer), inverse of a Poisson series, binomial expansion and so on (see, for example, [2]) Also, on the basis of PSP we can realize new specialized modules, like Kepler or Taylor ones In Keplerian module, for example, the polynomial and trigonometric variables are the well-known elliptic elements For these elements, there are transformation rules, which can be considered, from our point of view, as rewriting rules The next level of abstraction consists of modules for constructing the approximate solution of differential equations up to an desired order One can construct different modules for each "perturbation method", each of them using operations defined in previous modules Using different methods applied to the same problem, one can compare the obtaining solutions, keeping in mind the fact that many of methods are assimptotically equivalent This can be another facility of theorem proving of PSP

3. Concluding remarks PSP is intended to be a symbolic processor, with features of theorem proving, dedicated to the study of the motion of celestial bodies From the implementation point of view, there are some modules that are not so efficient, this difficulty remains to be considered later Taking into account the built-in abstract data types, the denotational semantics of initial models, the operational semantics based on rewriting rules, PSP, considered as open system, can be helpful in other fields, too

REFERENCES

1 Barwise,K J Axioms for Abstract Model Theory Ann.Math Logic, 7, pp 221-265, 1974
2 Brumberg,V A Analytic Algorithms of Celestial Mechanics, Nauka, Moskow, 1980 (russ)
3 Futatsugi,K ,Goguen,J ,Jouannaud,J P ,Meseguer,J Princi-ples of OBJ2 In Proc 1985 Symp on Principles of Programming Languages, ACM, pp 52-66, 1985
4 Goguen,J Parameterized Programming Tech Rep CSLI-84-9, 1984
5 Goguen,J et al Introducting OBJ Oxford University Draft of January, 1993
6 Goguen,J ,Burstall,R Introducing Institutions In E Clarke and D Kozen (eds), Logics of Programs, Springer, LNCS vol 164, pp 221-256, 1984
7 Goguen,J ,Burstall,R Institutions Abstract Model Theory for Computer Science Tech Rep CSLI-85-30, Stanford University, 1985
8 Goguen,J ,Meseguer,J Models and Equality for Logical Programming In Proc TAPSOFT87, Springer, LNCS vol 250, pp 1-22, 1987
9 Goguen,J ,Tardo,J An Introduction to OBJ A Language for Writing and Testing Software Specifications In Specifications of Reliable Software, IEEE Press, pp 170-189, 1979
10 Goguen,J ,Thatcher J ,Wagner,E An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types, Tech Rep RC 6487, IBM Watson Research Center, 1976
11 Meseguer,J General Logics H-D Ebbinghaus et al (eds), Elsevier B V , 1989
12 Meseguer,J ,Goguen,J Initiality, Induction and Computabi-lity, in Algebraic Methods in Semantics (M Nivat and J C Reynolds eds), Cambridge University Press, 1985
13 Parpucea,I ,Pârv,B Algebraic Specification of PSP, Studia, XXXVII, No 3, pp 99-110, 1992
14 Rand,R ,Armbruster,D Perturbation Methods, Bifurcation Theory and Computer Algebra, Springer, 1987
15 Tarski,A One Some Fundamental Concepts of Metamathematics In Logic, Semantics, Metamathematics, Oxford University Press, pp 30-37, 1956