Multi Level Recursive Specifications for Context Free Grammars

VASILE CRACIUNEAN
Dept. of Computer Science and Economic Informatics
Univ. "Lucian Blaga"
Bd. Victoriei, No.10
Sibiu
Romania

CRISTINA ARON
Dept. of Computer Science and Economic Informatics
Univ. "Lucian Blaga"
Bd. Victoriei, No.10
Sibiu
Romania

RALF FABIAN
Dept. of Computer Science and Economic Informatics
Univ. "Lucian Blaga"
Bd. Victoriei, No.10
Sibiu
Romaina

DANIEL HUNYADI
Dept. of Computer Science and Economic Informatics
Univ. "Lucian Blaga"
Bd. Victoriei, No.10
Sibiu
Romania

Abstract: - The support for the material presented in this paper was waved around two fundamental concepts: on one hand, the programming language concept regarded as an object that should be formally specified, and on the other hand, the concept of heterogeneous algebraic structure HAS, regarded as a specification mechanism. Also programming languages are probably one of the most studied objectives of computing techniques; they raise problems for computers designers and users, as the concept of programming language has not reached yet the maturity necessary to a mathematical object. Its different aspects constitute a study objective since the beginning of computers age. But the study method is more engineering, without a mathematical base.

Key-Words: - Fix point, context free algebra, HAS hierarchy, algebraic model

1 Introduction

At the beginning we recall some basic concepts regarding sets and posets (partial ordered set: it is reflexive, anti-symmetric and transitive).

- y is an upper bound of a subset Z of a poset $(P,\le)$ iff $y \in P$ and, for all $z \in Z$, $z \le y$;
- y is the least element of Z iff $y \in Z$ and, for all $z \in Z$, $y \le z$;
- y is a maximal element of Z iff $y \in Z$ and there is no $z \in Z$ such that $z \ne y$ and $y \le z$.

The notion of lower bound, greatest element, and minimal element receive dual definitions (i.e. definitions obtained by replacing "$\le$" by "$\ge$"). Also

- y is the supremum / infimum, denoted by $\vee Z$ / $\wedge Z$, of Z iff y is an upper / lower bound of Z and y is the least / greatest of the upper / lower bounds of Z, and

Definition 1.1. A partial ordered set $(P,\le)$ is called domain if it has one least element and if any ascending sequence over *P* has an upper bound in *P*.

Definition 1.2. Let $(D_1,\le_1),...,(D_n,\le_n)$, $n>0$ be domains. Then the product domain of *n* domains is a domain $(D,\le)$, where:

$D = D_1 \times ... \times D_n$ and

$(x_1,x_2,...,x_n) \le (y_1,y_2,...,y_n)$ iff $x_i \le_i y_i$, $i = \overline{1,n}$, while $(x_1,x_2,...,x_n),(y_1,y_2,...,y_n) \in D_1 \times ... \times D_n$.

Definition 1.3. Let D be a domain. A recursive specification over D is a total function $\psi : D \to D$ such that $\psi(\perp) \le \psi^2(\perp) \le \psi^3(\perp) \le ....$

The sequence $\psi(\perp) \le \psi^2(\perp) \le \psi^3(\perp) \le ...$ is called Kleene sequence for ψ.

Lemma 1.1. If $(D,\le)$ is a domain and $\psi : (D,\le) \to (D,\le)$ is monotone then ψ is a recursive specification.

Definition 1.4. Let $(P,\le)$ be a partial ordered set and $\psi : P \to P$ a total function. A fixed point of ψ is an element $f \in P$ that verifies $\psi(f) = f$. The least fix point of ψ (if it exists) is the least element from the set of fix points.

Theorem 1.1. Let $(P,\le)$ be a poset, and $\psi : (P,\le) \to (P,\le)$ a monotone function. If there exists $f = \vee\{h \mid h \le \psi(h)\}$, then it is a fix point of ψ.

2 Fix points

2.1 Kleen's fix point theorem

Theorem 2.1.1. (Kleene's fix point theorem)
Let $(D,\le)$ be a domain and $\psi : (D,\le) \to (D,\le)$ a continuous function. Then Kleen's semantic $f_\psi = \overset{\infty}{\underset{k=1}{\vee}} \psi^k(\perp)$ is the least fix point of ψ.

Example 2.1.1.
We denote $Pfn(X,Y)$ the set of all partial functions $f: X \to Y$. Then $(Pfn(X,Y),\leq)$ is a domain, where the relation $\leq$ is defined as follows:
$f \leq g \Leftrightarrow DD(f) \subseteq DD(g)$ and
$g(x) = f(x)$, $(\forall)x \in DD(f)$.
Thus, if $f_1 \leq f_2 \leq ... \leq f_j \leq ...$ we define $\vee f_i$ as

$$DD(\vee f_i) = \bigcup_{i=0}^{\infty} DD(f_i)$$

$(\vee f_i)(x) = f_k(x)$, $(\forall)$ k such that $x \in DD(f_k)$.
The function $\vee f_i$ is well defined because if $x \in DD(f_j) \cap DD(f_k)$, then $f_j(x) = f_k(x)$.
In this circumstances it can easily been seen that the Kleene sequence $\psi(\perp) \leq \psi^2(\perp) \leq \psi^3(\perp) \leq ...$ is indeed an increasing sequence in $Pfn(X,Y)$, and Kleene's semantic

$$DD(\vee f_\psi) = \bigcup_{k=1}^{\infty} DD(\psi^k(\perp)),$$

$f_\psi(x) = \psi^k(\perp)(x)$, $(\forall)$ k such that $x \in DD(\psi^k(\perp))$
satisfies the relation $f_\psi = \vee\psi^i(\perp)$.

Remark: The relation $f \leq g$ shows as that g offers at least as much information than f does.

2.2 Fix point and formal languages

In this part we focus on context free grammars (type 2 grammars in Chomsky's classification).
We'll show, using an example, that the language generated by a context free grammar (CFG) can be obtained from the smallest fix point of a well chosen recursive specification.

Example 2.2.1.
Suppose the following simple CFG:
$G = (V_N, V_T, S, P)$ where
P= { $S \to A$
$S \to B$
$A \to aAb$
$A \to ab$
$B \to bBa$
$B \to ba$
}
$V_N = \{S, A, B\}$
$V_T = \{a, b\}$
and S is the grammar axiom (starting symbol).
First we rewrite the production rules in the following way:
$S \to A+B$
$A \to aAb+ab$
$B \to bBa+ba$
where "+" denotes the union.

We define the recursive specification:
$\psi: (2^{V_T^*})^3 \to (2^{V_T^*})^3$
$\psi(S, A, B) = (A+B, aAb+ab, bBa+ba)$
Now we apply Kleene's theorem to determine the leases fix point for the above chosen ψ. Computing the Kleene sequence we have:
$\psi^0(\perp,\perp,\perp) = (\varnothing,\varnothing,\varnothing)$
$\psi^1(\perp,\perp,\perp) = \psi(\varnothing,\varnothing,\varnothing) = \{\varnothing, ab, ba)\}$
$\psi^2(\perp,\perp,\perp) = \psi(\varnothing, ab, ba) = \{ab+ba, a^2b^2+ab, b^2a^2+ba\}$
It can be easily seen that an induction over m proofs that $\psi^m(\perp) = (\{a^jb^j \mid 1 \leq j \leq m\} \cup \{b^ja^j \mid 1 \leq j \leq m\}, \{a^jb^j \mid 1 \leq j \leq m\}, \{b^ja^j \mid 1 \leq j \leq m\})$
Thus the sequence $\psi^m(\perp)$ is indeed an increasing sequence and we have the
$\underset{m\geq 0}{\vee} \psi^m(\perp) = (L(G), \{a^jb^j \mid j \geq 1\}, \{b^ja^j \mid j \geq 1\})$ (here $L(G)$ means the language generated by the grammar G).
We notice here that $L(G)$ is the first component of the least fix point of ψ. More generally speaking, for $k = 1,2,3,...$ the k-th component of $\underset{m\geq 0}{\vee} \psi^m(\perp)$ is
$\{w \mid w \in X^* \text{ and } v_k \overset{*}{\Rightarrow} w\}$, where $v_k \in \{S, A, B\}$.
A closer look on the above relations will reveal some other information too. Let $\overset{j}{\Rightarrow}$ denote the power j of the relation $\Rightarrow$, i.e. $w \overset{j}{\Rightarrow} w'$ means that w derives (generates) w' in j steps if there exists a sequence $w_1, w_2, ..., w_j$ such that $w = w_1 \Rightarrow w_2 \Rightarrow ... \Rightarrow w_{j-1} \Rightarrow w_j = w'$, and $w \overset{0}{\Rightarrow} w'$ means that $w = w'$.
Hence, the k-th element of $\psi^m(\perp)$ is
$\{w \mid w \in X^* \text{ and } v_k \overset{j}{\Rightarrow} w', \text{ for } j \leq m\}$.
However, this representation is tricky.

3 Context free algebras and algebraic models

3.1. Context free algebras

We'll specify heterogeneous operations (ω) by mean of operation schemes (σ).
Definition 3.1.1. Operations schemes are triplets of the form $\sigma = < n, s_0s_1...s_n, b_1b_2...b_n, b >$ where:
n – is the operation's arity

$s_0 s_1 ... s_n = \omega$ – is the symbol of heterogeneous operation specified by σ distributed on the operands, which is also named state's word of operation.

- $b_1 b_2 ... b_n$ – is operation's domain
- b – is operation's range.
- $b = s_0 b_1 s_1 b_2 ... s_n b_n$

Definition 3.1.2. In a HAS hierarchy the algebraic structure of level *i,* HAS(*i*) is called a base relatively to the structure of level *i*+1, HAS(*i*+1) from the same hierarchy.

Denote the structure HAS(*i*) by $\boldsymbol{B} = <B, \Omega_B>$, where:

B – support set

Ω_B – the set of operations that define the structure $\boldsymbol{B}$

Then the structure HAS(*i*+1) in the same hierarchy is specified as follows:

$\boldsymbol{A} = < A = (A_b)_{b \in B}, \Sigma = (\Sigma_\omega)_{\omega \in \Omega_B}, F >$, where

$A = (A_b)_{b \in B}$ – is a family of sets, indexed by the support of base $\boldsymbol{B}$ (B). That means, for every element $b \in B$ consider a set in the family that forms the support of the structure $\boldsymbol{A}$.

$\Sigma = (\Sigma_\omega)_{\omega \in \Omega_B}$ is a family of operations schemes, each operation schemes specifies an operation from the base.

Each n-ary operation $\omega \in \Omega_B$ induces in the structure $\boldsymbol{A}$ a set of operations schemes defined by the relation:

$\Sigma_\omega = \{ < n, , b_1 b_2 ... b_n, b > \mid b = (b_1, b_2, ..., b_n) \}$

F – is the symbol of a function that associates to each scheme operation $\sigma \in \Sigma_\omega$, $\omega \in \Omega_B$ a heterogeneous operation specific to HAS(*i*+1)

So the operation schemes are inherited from the base, but the action of the operations is specific to the new defined structure, so it can't be inherited from the base.

$F(\sigma) : A_{b_1} \times A_{b_2} \times ... \times A_{b_n} \to A_b$

Notice that HAS(*i*) behaves at the support's level as a factor structure of HAS(*i*+1).

A HAS hierarchy begins always with an homogenous (universal) algebra as a level base (HAS(0)).

Any homogenous algebraic structure is a 0-level HAS.

We will now focus on the 2nd level of HAS hierarchy, corresponding to context free algebras.

HAS(2) – Free context algebras

$\boldsymbol{A} = < A = (A_b)_{b \in B_2}, \Sigma S^1, F^1 >$, where ΣS^1 is given as follows:

$\Sigma S^1 = \{ < n, w_1, w_2 > \mid \mid w_1 \mid = \mid w_2 \mid = n+1 \}$, $(w_1, w_2) \in F^0_{\sigma_5}$

So if

$\sigma = < n, s_0 s_1 ... s_n, i_1 i_2 ... i_n, i > \in \Sigma S^1$

then σ specifies a heterogeneous operation of the form

$F^0_{\sigma_1} : A_{i_1} \times A_{i_2} \times ... \times A_{i_n} \to A_i$

If $a_k \in A_{i_k}$, $k = \overline{1,n}$, then

$F^0_{\sigma_1}(a_1, a_2, ..., a_n) = s_0 a_1 s_1 a_2 ... s_{n-1} a_n s_n$

Example 3.1.3. Consider a context free grammar $G = (V_N, V_T, S, P)$ that has the productions of the form

$$A \to s_0 A_1 s_1 A_2 ... s_{n-1} A_n s_n ,$$

$s_0, s_1, ... s_{n-1}, s_n \in V_T^*$, $A_1, A_2, ..., A_n \in V_N$

A base given by G is $\boldsymbol{B}_G = < V_T, V_N, \Sigma S >$, where

$\Sigma S = \{ \sigma = < n, s_0, s_1, ..., s_n, A_1 A_2 ... A_n A > \mid$

$A \to s_0 A_1 s_1 A_2 ... s_{n-1} A_n s_n \in P \}$

and the context free algebra derived from grammar G and specified by the base $\boldsymbol{B}_G$ can be written as a triplet.

$\boldsymbol{A}_G = = < W(V_T, \Sigma S) > = W_A(V_T, \Sigma S)_{A \in V_N}, \Sigma S, F^1 >$,

where: $W_A(V_T, \Sigma S) = [A] = \{ x \in V_T^* \mid A \overset{*}{\Rightarrow} x \}$

If $\sigma = < n, s_0, s_1, ..., s_n, A_1 A_2 ... A_n A >$,

$F^0_{\sigma_1}(w_1 w_2 ... w_n) = s_0 w_1 s_1 w_2 ... s_{n-1} w_n s_n \in W_A(V_T, \Sigma S) =$

$= (w_1 w_2 ... w_n) \in (A_1 \times A_2 \times ... \times A_n)$

So, the context free language

$L(G) = \{ w \in V_T^* \mid \exists A \in V_T, A \overset{*}{\Rightarrow} w \}$ coincides with the support of context free algebra $\boldsymbol{A}_G$ specified by $\boldsymbol{B}_G$ derived from the grammar G.

3.2. Algebraic model of an abstract computing system

The context free algebras offer a general framework which is ideal to build a model of the abstract computing system. The recursive specification built above will be very useful; those specifications, according to Kleene's theorem, will give the semantic constructions of the types of computing or of abstract dates.

Consider the base $B = < S^+, I^+, \lambda >$ where S^+ şi I^+ are free monoids (semigroups) generated by the symbols concatenation operation in the symbols of S and I, and $\lambda \subseteq S^+ \times I^+$ is a finite relation between the elements of structures S^+ and I^+, $\lambda = \{ (x, y) \in S^* \times I^* \mid \lambda(x) = \lambda(y) \}$.

This base gives the operation structure: $\Sigma S = \{\tau = (n, s_0 s_1 ... s_n, i_1 i_2 ... i_n, i) \mid (s_0 s_1 ... s_n, i_1 i_2 ... i_n, i) \in \lambda\}$
The context free algebra is $\mathcal{A} =< A = (A_i)_{i \in I}, \Sigma S, F >$.
In order that this algebra has the role of a mechanism for the specification of a real computing system, the operations schemes set ΣS has to be provided with an operation of composing types $i \in I$, so that I is an extensible set by mean of such an operation.
This extensibility has to be regarded as a free generation of I, over a given set, by mean of that operation.
So, I has to be seen in regard to λ as a finite sequence of levels $I_0 \subseteq I_1 \subseteq ... \subseteq I_t = I$. Each I_j is a generators system for I_{j+1} in regard to composing operation. I_0 will be named the level of primitive (or pre-defined) types of the algebra A and I_j is the level of types defined in the terms of types from I_{j-1}.
The levels of types $I_0, I_1, ..., I_t$ are called internal hierarchy levels of the set I, and m is called the hierarchic order.

Definition 3.2.1. If $B =< S^+, I^+, \lambda >$ is the base that specifies the context free algebra $\mathcal{A} =< A = (A_i)_{i \in I}, \Sigma S, F >$ and has the property that the set of types, I, satisfies the classification on levels relation specified above, we say that B is a base with internal hierarchy.
The classification relation of set I will be also inherited by context free algebra of type A specified by B, that's why this algebra will be called internal classified algebra. its order of internal classification coincides with base B order.
Let be $B =< S^+, I^+, \lambda >$ and
$\Sigma S = \{\tau = (n, s_0 s_1 ... s_n, i_1 i_2 ... i_n, i) \mid (s_0 s_1 ... s_n, i_1 i_2 ... i_n, i) \in \lambda\}$
the set of operations defined above. In practice we have a finite set of operations $\Sigma \subseteq \Sigma S$,

$$\Sigma = \{\sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >\mid s_0 s_1 ... s_n \in S^+, \; i_1 i_2 ... i_n i \in I^+\}$$

We denote
by $d(\sigma) = \{i_1, i_2, ..., i_n\}$ – the selector of the domain of operation σ,
by $t(\sigma) = \{j\}$ – the selector of the range of operation σ,
by $o(\sigma) = s_0, s_1, ..., s_n$ – the symbol of operation specified
by σ distributed on operators and called the state word of that operation,
and by $n(\sigma)=n$ – the arity of the operation.
First we'll build the hierarchy of the sets of types defined by the operations from Σ as follows:

$I^0 = \{i \in I \mid \sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >\in \Sigma$ and $d(\sigma) = \varnothing\}$
$I^{n+1} = I^n \cup \{i \in I \mid \sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >\in \Sigma$ and $d(\sigma) \in I^i\}$
This hierarchy of sets can be computed by the following algorithm:

Algorithm 3.2.1
Given a set of operations $\Sigma \subseteq \Sigma S$

$$\Sigma = \{\sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >\mid s_0 s_1 ... s_n \in S^+ \text{ and } i_1 i_2 ... i_n i \in I^+\}$$

we give an algorithm that determines the hierarchy of types defined by Σ.
Input: The set of operations:

$$\Sigma = \{\sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >\mid s_0 s_1 ... s_n \in S^+ \text{ and } i_1 i_2 ... i_n i \in I^+\}$$

Output: The sets I^k of different types.
The method consists of building a sequence of sets $\{I^k \mid k \in \mathbb{N}\}$ with the property
$I^0 \subset I^1 \subset ... \subset I^{k-1} = I^k = ...$ As one can notice, I^k will contain all the types defined by the operations set Σ.
Algorithm
$n \leftarrow 0$;
$I^0 = \{i \in I \mid \sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >\in \Sigma$ and $d(\sigma) = \varnothing\}$
DO UNTIL ($I^n = I^{n-1}$)

$$I^{n+1} = I^n \cup \{i \in I \mid \sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i > \in \Sigma \text{ and } d(\sigma) \in I^i\}$$

$n \leftarrow n + 1$;
ENDDO
STOP

Proposition 3.2.1. For any finite set of operations $\Sigma \subseteq \Sigma S$

$$\Sigma = \{\sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >\mid s_0 s_1 ... s_n \in S^+, \; i_1 i_2 ... i_n i \in I^+\}$$

there is a finite number k of levels of hierarchical types.
The proof is obvious.
We build now an internal hierarchy of operations from $\Sigma \subseteq \Sigma S$:

$\Sigma^0 = \{\sigma \in \Sigma \mid d(\sigma) = \varnothing$ and $t(\sigma) \subseteq I^0\}$
$\Sigma^k = \{\sigma \in \Sigma \mid d(\sigma) \subseteq I^k$ and $t(\sigma) \subseteq I^{k-1}\}$
This hierarchy op operation can be determined by the following algorithm:

Algorithm 3.2.2.
Given a set of operations $\Sigma \subseteq \Sigma S$

$\Sigma = \{\sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >| s_0 s_1 ... s_n \in S^+$ and $i_1 i_2 ... i_n i \in I^+\}$

we give an algorithm that determines the internal hierarchy of those.

Input: The set of operations:

$\Sigma = \{\sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >| s_0 s_1 ... s_n \in S^+$ and $i_1 i_2 ... i_n i \in I^+\}$

Output: The sets Σ^k of types.

The method consists of building a sequence of sets $\{\Sigma^k \mid k \in \mathbb{N}\}$ with property

$\Sigma^0 \subset \Sigma^1 \subset ... \subset \Sigma^{k-1} = \Sigma^k = ...$ As it can be seen, Σ^k will contain all the operations of Σ.

Algorithm:

$k \leftarrow 0;$

$\Sigma^0 = \{\sigma \in \Sigma \mid d(\sigma) = \varnothing$ and $t(\sigma) \subseteq I^0\}$

DO UNTIL ($\Sigma^k = \Sigma^{k-1}$)

$\Sigma^k = \{\sigma \in \Sigma \mid d(\sigma) \subseteq I^k$ and $t(\sigma) \subseteq I^{k-1}\}$

$k \leftarrow k+1;$

ENDDO

STOP

Proposition 3.2.2. For any finite set of operations $\Sigma \subseteq \Sigma S$

$\Sigma = \{\sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i >| s_0 s_1 ... s_n \in S^+$ and $i_1 i_2 ... i_n i \in I^+\}$

there is a finite number k of hierarchical levels of operations $\Sigma^1 \subset \Sigma^2 \subset ... \subset \Sigma^{k-1} = \Sigma^k$

The proof is obvious.

Proposition 3.2.3. Let be the set $(2^{S^+}, \subseteq)$; then the product $((2^{S^+})^n, \subseteq_n)$, where

$(x_1, x_2, ..., x_n) \subseteq_n (y_1, y_2, ..., y_n) \Leftrightarrow x_i \subseteq_i y_i, i = \overline{1,n},$

$(x_1, x_2, ..., x_n), (y_1, y_2, ..., y_n) \in (2^{S^+})^n$

is a domain.

Proof: A partially ordered (regular) set $(P, \leq)$ is a domain if it has one smallest element and if any ascending sequence from P has an upper bound in P. The set $(2^{S^+}, \subseteq)$ is partially ordered, has one smallest element which is the empty set $\varnothing$. Also it is obvious that any ascending sequence from 2^{S^+} are an upper bound in 2^{S^+} and so $(2^{S^+}, \subseteq)$ is a domain. So the product $((2^{S^+})^n, \subseteq_n)$ is a domain.

For building the hierarchy of types we'll use recursive specifications adequate to each hierarchical level.

For each $I^k = \{i_{k_1}, i_{k_2}, ..., i_{k_r}\}$ consider the recursive specification

$\psi_k : (2^{S^+})^r \to (2^{S^+})^r$

$\psi_k(i_{k_1}, i_{k_2}, ..., i_{k_r}) = ([i_{k_1}], [i_{k_2}], ..., [i_{k_r}])$, where we denoted by

$[i_{k_j}] = \Sigma\{\alpha = s_0 i_1 s_1 i_2 ... s_n i_n s_{n+1} \mid \sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i > \in \Sigma$ and $d(\sigma) = i_{k_j}\}$

Proposition 3.2.4. The application $\psi_k : (2^{S^+})^r \to (2^{S^+})^r$ is a recursive specification.

Proof: If $(D, \leq)$ is a domain and $\psi : (D, \leq) \to (D, \leq)$ is monotone, then Ψ is a recursive specification. So we have to prove that $\psi_k : (2^{S^+})^r \to (2^{S^+})^r$ is monotone.

Let be $(x_1, x_2, ..., x_n) \subseteq_n (y_1, y_2, ..., y_n)$; $(x_1, x_2, ..., x_n), (y_1, y_2, ..., y_n) \in (2^{S^+})^n$. We shall prove that $\psi_k(x_1, x_2, ..., x_n) \subseteq_n \psi_k(y_1, y_2, ..., y_n)$

But $\psi_k(x_1, x_2, ..., x_n) = ([x_1], [x_2], ..., [x_n])$, where we denoted by

$[x_j] = \Sigma\{\alpha = s_0 x_1 s_1 x_2 ... s_n x_n s_{n+1} \mid \sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i > \in \Sigma$ and $d(\sigma) = i_j\}$

and $\psi_k(y_1, y_2, ..., y_n) = ([y_1], [y_2], ..., [y_n])$, where we denoted by

$[y_j] = \Sigma\{\alpha = s_0 y_1 s_1 y_2 ... s_n y_n s_{n+1} \mid \sigma =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n i > \in \Sigma$ and $d(\sigma) = i_j\}$

Considering the fact that if we have the sets A, B, C, then:

i) $A \subseteq B \Rightarrow A.C \subseteq B.C;$

ii) $A \subseteq B \Rightarrow C.A \subseteq C.B;$

iii) $A \subseteq B \Rightarrow A \cup C \subseteq B \cup C;$

iv) $A \subseteq B \Rightarrow C \cup A \subseteq C \cup B;$

it results immediately that $\psi_k(x_1, x_2, ..., x_n) \subseteq_n \psi_k(y_1, y_2, ..., y_n)$

According to the Kleene's fixed point theorem, it results that the Kleene semantic

$f_{\Psi_k} = \bigvee_{l=1}^{\infty} \psi_k^l(\perp, \perp, ..., \perp)$ is the smallest fixed point of Ψ.

If we denote by $\Psi_k^l(\perp, \perp, ..., \perp) = (L_{k_1}^l, L_{k_2}^l, ..., L_{k_r}^l)$ and by $f_{\psi_k} = (f_{\psi_k}^1, f_{\psi_k}^2, ..., f_{\psi_k}^r)$, then each $f_{\psi_k}^m = \bigvee_{l=1}^{\infty} L_{k_m}^l$, for $k = 1, 2, ..., r.$

Denote by $A_k^m = f_{\Psi_k}^m$ and define for each k the following heterogeneous algebra

$\mathcal{A}_k = (A_k = (A_k^m)_{m=1,2,...,r}, \Sigma^k, F_k)$

The process of internal inheritance of the base *B* in the algebra *A* is:

1. The set of primitive types of algebra $\mathcal{A}_0$ is defined by mean of null-ary operations: $\Sigma^0 = \{\tau \in \Sigma \mid d(\tau) = \varnothing\}$

In other words, each scheme of null-ary operation from Σ, $\tau =< o, s, i >$ specifies a type of computing object or a primitive data type in the set A_0^m, $m \in I$, specified by that scheme as follows:

if $\tau =< o, s, i >$, then $s \in A_0^m$ is a constant of type *m*, and A_0^m is the set of all computing objects or data of type *m*, or, in other words, A_0^m is the type of primitive data in $\mathcal{A}$.

2. Σ^1 is the set of schemes of operations defined in the terms of the schemes of operations from Σ^0. That means, if $\tau \in \Sigma^1$ then $d(\tau)$ selects as a domain only primitive types, the type defined by τ is a new type which we'll call the compose type of the types from $d(\tau)$, the composing operator is given by $o(\tau)$ and the n-arity is $n(\tau)$.

The operation of composing primitive types defined by Σ^1 can be seen under two aspects, which are: construction operations which represent the construction of some static structures and operations of constructing action types, or dynamical.

If $\tau =< n, s_0 s_1 ... s_n, i_1 i_2 ... i_n, j >$, then $i_1 i_2 ... i_n$ have to be seen as types of operators or null-ary computing processes, and *j* is the type of *n*-ary operation's result. The meaning of each operation τ in $\mathcal{A}_k$ is given by F_k.

Internal levels of algebra $\mathcal{A}$ are formed by the sub-types that can be specified using $A_0, A_1, \ldots, A_t$ and which have the following properties:

$A_0 \subset A_1 \subset ... \subset A_t$

$A_t = A$

A_{i-1} has the role of generating the sub-type A_i.

4 Conclusion

Algebraic modeling allows developing some computing models faithful and efficient from user's point of view. It allows computing models modularization, ensuring their reliability. Algebraic modeling offers mechanisms for generating syntax naturally associated to a semantic given by a model.

Algebraic modeling offers schemes for evaluating that are linked to the communicators for which the language program is specified.

Algebraic modeling that satisfies best the characteristics above is the one given by heterogeneous algebraic structures HAS. This concept appeared, on one hand, independently from the need of faithful modeling of programming languages, i.e. from demand of generalizing the concepts of universal algebras. On the other hand, HAS concept was introduced from necessity of associating a structure adequate to formal languages used traditionally in specifying conventional programming languages. Anyway, HAS proved to be a strong instrument, able to resolve many of the difficult problems of programming languages.

The relations of internal hierarchy offered a natural framework for developing universal algorithms mechanism of implementing programming languages specified as mentioned.

References:

[1] Dorin Andrica, Dorel I. Duca, Ioan Purdea, Ioana Pop, *Matematica de baza*, Ed. STUDIUM, Cluj-Napoca 2002.

[2] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Compilers*: *Principles, Techniques, and Tools*, Addison Wesley, 2001.

[3] Crăciunean V., *Translatoare şi compilatoare*, Ed. Alma Mater, Sibiu, 2002.

[4] Ralf Fabian, *Limbaje formale Teorie. Exemple. Probleme*, Lucian Blaga University Publishing, Sibiu, 2006.

[5] Emil M. Popa, *Generative Mechanisms for Economic Processes*, „Alma Mater" Printing House Sibiu, Romania 2003, vol.I, vol.II.

[6] Emil M. Popa, *Formal Syntax and Semantics of Programming Language*, „Alma Mater" Printing House Sibiu, Romania, 2004.

[7] Emil M. Popa, *Regular Expressions of Conditions for Processing Language Modelling*, Proceedings of the WSEAS International Conference MMACTEE '06, October 16-18 Bucharest, 2006.

[8] Gh. Păun, G. Rozenberg, A. Salomaa, *Current Trends in Theoretical Computer Science. The Challenge of the New Century*, Vol. I: Algorithms and Complexity, Vol. II: Formal Models and Semantics, World Scientific, Singapore, 2004.

[9] Gh. Păun, C. Martin-Vide, V. Mitrana, *Formal Language Theory and Applications*, Springer-Verlag, Berlin, 2004.

[10] Rajesh Kumar Gupta, *Formal Methods and Models for System Design*, Springer Publishing 2004.