Algebraic Model of Programming Languages

CRISTINA IOANA BRUMAR, EMIL M. POPA, FLORENTINA LAURA CACOVEAN, BOGDAN ALEXANDRU BRUMAR
Department of Computer Science
Lucian Blaga University of Sibiu, Faculty of Sciences
Str. Ion Ratiu 5-7, 550012, phone +40269216642, Sibiu
ROMANIA

Abstract: - This paper is an approach methodology of programming language, considerate like a tuple $L = \langle Sem, Syn, l: Sem \rightarrow Syn\rangle$. The definition of a programming language provides a working mechanism for the development of language processing tools. A language processing environment is a set of integrated tools that support the three tasks of computer language processing: specification, implementation and usage. The development the tools that belong to a language processing environment requires language specification mechanisms that define formally all components of a programming language, the syntax, the semantics, and their association, by the same specification rules. The results presented here have as initial point [7], [8], [9].

Key-Words: - model algebra, semantic, syntax, specification, domain, programming language

1 Introduction

Definition 1: Language L is defined as the tuple

$$L = \langle Sem, Syn, l: Sem \rightarrow Syn\rangle$$

where *Sem* is a Σ-algebra which is the language semantic, *Syn* is a Σ word algebra which is the language syntax, and *l* is a partial mapping called the language learning function. *l* maps semantic constructs in *Sem* to their expressions as a syntactic constructs in *Syn* such that there exists a complementary homomorphism $\varepsilon : Syn \rightarrow Sem$, called the language evaluation function, that maps expressions in *Syn* to their semantic constructs in *Sem*. When a programming language is defined in this manner, the carrier sets in the semnatics *Sem* contain the valid computations of the language and the words in the syntax. *Syn* are the expressions of the computations. *l* is a mechanism that allows a programmer to learn to express computations in *Sem* by a symbolic construct in *Syn* such that for any computation $c \in Sem$, $\varepsilon(l(c)) = c$.

The evaluation function is related to the ability to express computation by the identity $l \circ \varepsilon = 1_{Sem}$, where 1_{Sem} is the identity map on *Sem*.

A language specification rule formally describes elements of *Syn*, *Sem* and the relationship between them. Equations of the form $r: A_0 = t_0A_1t_1A_2t_2 \ldots A_nt_n$ called BNF notation [6] are used as specification rules, where A_i, $0 \le i \le n$, are parameters and t_i, $0 \le i \le n$, are fixed strings. For a given specification rule $r : A_0 = t_0 A_1 t_1 A_2 t_2 \ldots A_n t_n$, A_0 is the *left hand side* of the rule and is denoted by $lhs(r)$ and $t_0 A_1 t_1 A_2 t_2 \ldots t_{n-1} A_n t_n$ is the *right hand side* of the rule and is denoted by $rhs(r)$. We assume here that a programming language is specified by a finite set, *R*, *T* is a collection of fixed strings used by rules in *R* and $W = \langle N \cup T; \circ, e\rangle$ is the semigroup associative of words with unity *e* generated by concatenation $\circ$ over the alphabet $N \cup T$.

For a set of specification rules *R* the syntax algebra, *Syn(R)* is unique, while the semantic algebra, *Sem(R)* may be a language user choice. The algebraic models of language processing requires both *Syn(R)* and *Sem(R)* to be specified by concrete notation. Since $Syn(R)$ is unique, the major problem rests with the notation used to express $Sem(R)$.

2 The algebraic model of the syntax

Definition 2: The syntax model of language specified by *R* is:

$$Syn(R) = \langle \{[A_n] \mid A_n \in N\}, \{[r] \mid r \in R\} \rangle ,$$

where $[A_n], A_n \in N$ and $[r], r \in R$, are the syntax interpretations of the specification rules and are defined as follows:

- Each parameter $A_n \in N$ is interpreted as a family of well-formed expressions called syntactic domain, denoted by $[A_n]$.
- Each rule $r \in R, r : A_0 = t_0 A_1 t_1 ... t_{n-1} A_n t_n$ is interpreted as an algebraic operation $[r] : [A_1] \times ... \times [A_n] \to [A_0]$ which constructs the elements $w_0 \in [A_0]$ from the elements $w_i \in [A_i]$, with $0 \le i \le n$, by the rule:
  $[r](w_1, w_2, ..., w_n) = w_0 = t_0 w_1 t_1 w_2 ... t_{n-1} w_n t_n$.

The specification rules of a simple language are illustrated in Table 1:

```
State = Ident ":=" Expr
State = "if" Expr "then" State "else"
State "fi"
State = "while" Expr "do" State "od"
State = B1
State1 = State
State1 = State ";" State1
IdentCt1 = Ident Type
IdentCt1 = IdentCt
IdentCt1 = IdentCt ";" IdentCt1
B1 = "begin" IdentCt1 ";" State1 "end"
```

Table 1

The model of syntax algebra of the language specified by the rules in Table 1 is:

$$Syn(R) = \left\langle \begin{Bmatrix} [[Ident]], [[Expr]], [[Type]], [[State]], [[Slate1]], \\ [[IdentCt]], [[IdentCt1]], [[B1]] \end{Bmatrix}; \right.$$
$$\left. \{[[r_1]], [[r_2]], [[r_3]], [[r_4]], [[r_5]], [[r_6]], [[r_7]], [[r_8]], [[r_9]], [[r_{10}]]\} \right\rangle.$$

The syntax interpretation (domain) of the parameters used in these rules is defined by the equations in Table2:

$[Expr]$ = set of expressions generated by identifiers and constants;

$[Type]$ = set of given (predefined or defined) type names;

$$[State] = \{ident := \exp r \mid ident \in [Ident], \exp r \in [Expr]\}$$
$$\cup \{if\ e\ then\ s\ else\ s\ fi \mid e \in [Expr], s \in [State]\}$$
$$\cup \{while\ e\ do\ s\ od \mid e \in [Expr], s \in [State]\}$$
$$\cup \left\{begin\ identct1; state1\ end \,\middle|\, \begin{matrix} identct1 \in [IdentCt1], \\ state1 \in [State1] \end{matrix}\right\};$$
$$[State1] = [State] \cup \{h; t \mid h \in [State], t \in [State1]\};$$
$$[IdentCt] = \{it \mid i \in [Ident], t \in [Type]\};$$
$$[IdentCt1] = [IdentCt] \cup \{h; t \mid h \in [IdentCt], t \in [IdentCt1]\};$$
$$[Bl] = \{begin\ i1; s1\ end \mid i1 \in [IdentCt1], s1 \in [State1]\};$$

Table 2

where operations $[r_1]$ through $[r_{10}]$ are provided by the syntax interpretation of the specification rules defined by equations in Table 3.

Syntax interpretation of the rules in Table 1:

$[r_1]$: $[Ident] \times [Expr] \to [State]$;
$[r_2]$: $[Expr] \times [State] \times [State] \to [State]$;
$[r_3]$: $[Expr] \times [State] \to [State]$;
$[r_4]$: $[B1] \to [State]$;
$[r_5]$: $[State] \to [State1]$;
$[r_6]$: $[State] \times [State1] \to [State1]$;
$[r_7]$: $[Ident] \times [Type] \to [IdentCt]$;
$[r_8]$: $[IdentCt] \to [IdentCt1]$;
$[r_9]$: $[IdentCt] \times [IdentCt1] \to [IdentCt1]$;
$[r_{10}]$: $[IdentCt1] \times [State1] \to [B1]$;

$$[r_1](ident, \exp r) = ident := e$$
$$[r_2](e, s_1, s_2) = if\ e\ then\ s_1\ else\ s_2\ fi$$
$$[r_3](e, s) = while\ e\ do\ s\ od$$
$$[r_4](b) = b$$
$$[r_5](s) = s$$
$$[r_6](h, t) = h; t$$
$$[r_7](i, t) = i\,t$$
$$[r_8](i) = i$$
$$[r_9](h, t) = h; t$$
$$[r_{10}](i1, s1) = begin\ i1; s1\ end$$

Table 3

Fact 1: $Syn(R)$ is embedded in the semi group $W = \langle N \cup T; \circ, \varepsilon \rangle$ by derived operations. A component relation, $\le_{Syn}$, can now be defined by:

for all $w, w' \in Syn(R)$, $w \leq_{Syn} w'$ if $w = w'$ or there is $r \in R$ such that $w' = [r](w_1, ..., w_k)$ and $w \leq_{Syn} w_i$ for some i, $1 \leq i \leq k$.

3 The algebraic model of the semantics

Definition 3: The semantic algebra of language specified by R is
$Sem(R) = \langle \{[[A_n]] \mid A_n \in N\}, \{[[r]] \mid r \in R\} \rangle$, where
$[[A_n]], A_n \in N$ and $[[r]], r \in R$, are the semantics interpretations of the parameters and specification rules and are defined as follows:

- Each parameter $A_i \in N$ is interpreted as a family of computations called semantic domain, denoted by $[[A_i]]$;
- Each rule $r \in R, r: A_0 = t_0 A_1 t_1 ... t_{n-1} A_n t_n$ is interpreted as an algebraic operation $[[r]]: [[A_1]] \times ... \times [[A_n]] \to [[A_0]]$ which constructs the elements $c_0 \in [[A_0]]$ from the elements $c_i \in [[A_i]]$, with $0 \leq i \leq n$, by the rule: $[[r]](c_1, c_2, ..., c_n) = c_0$.

The computations expressed in this language are machine-independent. Each computation is however specified in terms of three generic elements [8]: an universe of discourse defined by a collection of types, a state defined as a mapping assigning values of types given in the universe to a finite set of terms (variables and constants), and a state transition that maps the state before computation into the state after computation. The universe of types contains the type computation process whose values are processes performing given computation. We assume that a computation operates on a set D of typed data variables and constants, that denote values in the universe of types, and a set C of control variables and constants that denote processes in the universe of types. To simplify presentation we use here only two control variables denoted by $\uparrow$ which identifies a computation process before its execution, and $\downarrow$ that identifies a computation process after its execution.

A machine independent computation is a sequence of transitions

$$\langle T_1, \sigma_1 \rangle \xrightarrow{\tau_1} \langle T_2, \sigma_2 \rangle \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_{i-1}} \langle T_i, \sigma_i \rangle \xrightarrow{\tau_i} \langle T_{i+1}, \sigma_{i+1} \rangle \xrightarrow{\tau_{i+1}} \ldots$$

The initial state of a computation is a state whose variables and constants satisfy an initial condition: $\Theta : D \cup \{\uparrow, \downarrow\} \to \{true, false\}$; the final state of a computation in a state whose variables and constants satisfy a final condition:
$\Phi : D \cup \{\uparrow, \downarrow\} \to \{true, false\}$.

To construct the algebraic model of the semantic we first observe that computational interpretation of parameters N used in R is one of: type, state and mapping. Type parameters $t \in N$ represent families of sets $\nu(t) = ([[t]]), t \in N$; state parameters $\sigma \in N$ represent functions mapping sets of names (of variables and constants), D, to values in the universe of their types, $\sigma : D \to ([[t]]), t \in N$; mapping parameters $\tau \in N$ represent state-transitions, mapping functions $\sigma_t : D_t \to [[t]], t \in N$ into functions $\sigma'_{t'} : D'_{t'} \to [[t']], t' \in N$. These interpretations of the parameters in N can be unified as transitions, as follows:

$$[[p]] = \left\{ \begin{array}{l} \langle p, \phi \rangle \xrightarrow{\iota} \langle p, \phi \rangle, \ p = type \\ \langle t, \sigma : D \to \nu(t) \rangle \xrightarrow{\iota} \langle t, \sigma : D \to \nu(t) \rangle, \\ p = state \\ \langle t, \sigma : D \to \nu(t) \rangle \xrightarrow{\tau} \langle t', \sigma' : D' \to \nu(t') \rangle, \\ p = transformation \end{array} \right\}$$

The rules $r \in R, r: A_0 = s_0 A_1 s_1 \ldots s_{n-1} A_n s_n$ are interpreted as computation steps. For the example given in Table 1 the semantic domain specification is shown in Table 4:

$$\llbracket State \rrbracket = \quad \forall w \in [St] \text{if } w = ident := e \text{then}$$
$$\llbracket State \rrbracket \cup (D, \uparrow ident := e) \xrightarrow{\tau_{:=}} (D', id := e \downarrow)$$
$$\text{where}\, \sigma'(x) = \sigma(x) if\ x \neq ident\ else \llbracket e \rrbracket$$
$$\forall w \in [St] \text{if } w = if\ e\ then\ s_1\ else\ s_2\ fi \text{ and } \llbracket e \rrbracket = true \text{ then}$$
$$\llbracket State \rrbracket \cup (D, \uparrow if\ e\ then\ s_1\ else\ s_2\ fi)$$
$$\xrightarrow{\tau_e} (D', if\ e\ then \uparrow s_1\ else\ s_2)$$
$$\circ (D', if\ e\ then \uparrow s_1\ else\ s_2\ fi)$$
$$\xrightarrow{\tau_{s1}} (D'', if\ e\ then\ s_1 \downarrow else\ s_2\ fi)$$
$$\circ (D'', if\ e\ then\ s_1 \downarrow else\ s_2\ fi)$$
$$\xrightarrow{\iota} (D'', if\ e\ then\ s_1\ else\ s_2\ fi \uparrow)$$
$$\forall w \in [St] \text{if } w = if\ e\ then\ s_1\ else\ s_2\ fi \text{ and } \llbracket e \rrbracket = false \text{ then}$$
$$\llbracket State \rrbracket \cup (D, \uparrow if\ e\ then\ s_1\ else\ s_2\ fi)$$
$$\xrightarrow{\tau_e} (D', if\ e\ then\ s_1\ else \uparrow s_2\ fi)$$
$$\circ (D', if\ e\ then\ s_1\ else \uparrow s_2\ fi)$$
$$\xrightarrow{\tau_{s2}} (D'', if\ e\ then\ s_1\ else\ s_2 \downarrow fi)$$
$$\circ (D', if\ e\ then\ s_1\ else\ s_2 \downarrow fi)$$
$$\xrightarrow{\iota} (D'', if\ e\ then\ s_1\ else\ s_2\ fi \uparrow)$$
$$\forall w \in [State] \text{if } w = while\ e\ do\ s\ od$$
$$\text{and } \llbracket e \rrbracket = true \text{ then } \llbracket State \rrbracket \cup (D, \uparrow while\ e\ do\ s\ od)$$
$$\xrightarrow{\tau_e} (D', while\ e\ do \uparrow s\ od)$$
$$\circ (D', while\ e\ do \uparrow s\ od)$$
$$\xrightarrow{\tau_s} (D'', while\ e\ do\ s \downarrow od)$$
$$\circ (D'', while\ e\ do\ s \downarrow od)$$
$$\xrightarrow{\iota} (D'', while\ e\ do\ s\ od \uparrow)$$
$$\forall w \in [State] \text{if } w = while\ e\ do\ s\ od \text{ and} \llbracket e \rrbracket = false \text{ then}$$
$$\llbracket State \rrbracket \cup (D, \uparrow while\ e\ do\ s\ od)$$
$$\xrightarrow{\tau_e} (D', while\ e\ do \downarrow s\ od)$$
$$\circ (D', while\ e\ do \downarrow s\ od) \xrightarrow{\iota} (D', while\ e\ do\ s\ od \uparrow)$$

Table 4

We illustrate the construction of model algebra *Sem(R)* using the specification rules given in Table 1, assuming that $\llbracket Ident \rrbracket$ is the universe of names, $\llbracket Expr \rrbracket$ is the universe of values, and $\llbracket Type \rrbracket$ is the set of types. For $x \in [Ident]$, $e \in [Expr]$ and $t \in [Type]$ we denote with τ_x, τ_e and τ_t, respectively, the corresponding transitions in $\llbracket Ident \rrbracket$, $\llbracket Expr \rrbracket$, $\llbracket Type \rrbracket$, respectively. In addition, $\llbracket e \rrbracket \in \llbracket Expr \rrbracket$ is the value of the expression e computed to current state, replacing each variable that occur in e with its value in the current state; $\llbracket State \rrbracket$ is shown in Table 4.

The semantic algebra of the language specified by the rules in Table 1 is:

$$Sem(R) = \left\langle \left\{ \begin{matrix} \llbracket Ident \rrbracket, \llbracket Expr \rrbracket, \llbracket Type \rrbracket, \llbracket State \rrbracket, \llbracket Slate1 \rrbracket, \\ \llbracket IdentCt \rrbracket, \llbracket IdentCt1 \rrbracket, \llbracket B1 \rrbracket \end{matrix} \right\}; \right.$$
$$\left. \{ \llbracket r_1 \rrbracket, \llbracket r_2 \rrbracket, \llbracket r_3 \rrbracket, \llbracket r_4 \rrbracket, \llbracket r_5 \rrbracket, \llbracket r_6 \rrbracket, \llbracket r_7 \rrbracket, \llbracket r_8 \rrbracket, \llbracket r_9 \rrbracket, \llbracket r_{10} \rrbracket, \circ, \iota \} \right\rangle,$$

where the operations $\llbracket r_1 \rrbracket$ through $\llbracket r_{10} \rrbracket$ are in Table 5.

Fact 2: *Sem(R)* is embedded in the semi group $\langle \{ \llbracket A \rrbracket \mid A \in N \}, \circ, \iota \rangle$ by derived operations.

The semantic interpretations of the rules in Table 1 are shown in next table:

$$\llbracket r_1 \rrbracket : \llbracket Ident \rrbracket \times \llbracket Expr \rrbracket \to \llbracket State \rrbracket;$$
$$\llbracket r_1 \rrbracket(\tau_x, \tau_e) = \langle x \to \sigma(x), \uparrow e \rangle$$
$$\xrightarrow{\tau_e} \langle x \to \llbracket e \rrbracket, e \downarrow \rangle$$
$$\llbracket r_2 \rrbracket : \llbracket Expr \rrbracket \times \llbracket State \rrbracket \times \llbracket State \rrbracket \to \llbracket State \rrbracket;$$
$$\llbracket r_2 \rrbracket(\tau_e, \tau_1, \tau_2) = \tau_1\ if\ \llbracket e \rrbracket = true\ else\ \tau_2$$
$$\llbracket r_3 \rrbracket : \llbracket Expr \rrbracket \times \llbracket State \rrbracket \to \llbracket State \rrbracket;$$
$$\llbracket r_3 \rrbracket(\tau_e, \tau) = \tau \circ \llbracket r_3 \rrbracket((\tau_e \circ \tau), \tau)$$
$$if\ \llbracket e \rrbracket = true\ else\ \tau_1$$
$$\llbracket r_4 \rrbracket : \llbracket B1 \rrbracket \to \llbracket State \rrbracket;$$
$$\llbracket r_4 \rrbracket(\tau) = \tau$$
$$\llbracket r_5 \rrbracket : \llbracket State \rrbracket \to \llbracket State \rrbracket;$$
$$\llbracket r_5 \rrbracket(\tau) = \tau$$
$$\llbracket r_6 \rrbracket : \llbracket State \rrbracket \times \llbracket State1 \rrbracket \to \llbracket State1 \rrbracket;$$
$$\llbracket r_6 \rrbracket(\tau_1, \tau_2) = \tau_1 \circ \tau_2$$
$$\llbracket r_7 \rrbracket : \llbracket Ident \rrbracket \times \llbracket Type \rrbracket \to \llbracket IdentCt \rrbracket;$$
$$\llbracket r_7 \rrbracket(\tau_x, \tau_t) = \langle x \to \sigma(x), \varepsilon \rangle$$
$$\xrightarrow{\tau_t} \langle x \to @_t, \varepsilon \rangle$$
$$\llbracket r_8 \rrbracket : \llbracket IdentCt \rrbracket \to \llbracket IdentCt1 \rrbracket;$$
$$\llbracket r_8 \rrbracket(\tau) = \tau$$
$$\llbracket r_9 \rrbracket : \llbracket IdentCt \rrbracket \times \llbracket IdentCt1 \rrbracket \to \llbracket IdentCt1 \rrbracket;$$
$$\llbracket r_9 \rrbracket(\tau_1, \tau_2) = \tau_1 \circ \tau_2$$
$$\llbracket r_{10} \rrbracket : \llbracket IdentCt1 \rrbracket \times \llbracket State1 \rrbracket \to \llbracket B1 \rrbracket;$$
$$\llbracket r_{10} \rrbracket(\tau_1, \tau_2) = \tau_1 \circ \tau_2$$

Table 5

A component relation, $\leq_{Sem}$, on *Sem(R)* can now be defined by: for all transitions

$$\tau_1 = (T_1, D_1, \uparrow w) \xrightarrow{\tau_w} (T'_1, D'_1, w \downarrow),$$
$$\tau_2 = (T_2, D_2, \uparrow w') \xrightarrow{\tau_{w'}} (T'_2, D'_2, w' \downarrow),$$

$\tau_1 \leq_{Sem} \tau_2$ if $\tau_{w'} = E(\tau_w, \circ, \iota)$.

Note that if $w' \in w[\ ]$ then $\tau_w \leq_{Sem} \tau_{w'}$. If we denoted by $w(\tau)$ the expression of the computation performed by a transition τ then we can see that $w(\tau_1) \leq_{Sem} w(\tau_2)$ implies $\tau_1 \leq_{Sem} \tau_2$ and vice-versa.

References:

[1] G. Birkhoff, *Lattice Theory*, American Mathematical Society, 1948.

[2] R. M. Burstall and P. J. Landin, Programs and their proofs: an algebraic approach, *Machine Intelligence*, 17-43, 1969.

[3] T. M. V. Jansen, Algebraic translations, correctness and algebraic compiler construction, *Theoretical Computer Science*, 25-56, 1998.

[4] U. Kastens and W. M. Waite, Modularity and reusability in attribute grammars, *Acta Informatica*, 601-627, 1994.

[5] P. Klint, *The evolution of implementation techniques in the asf - sdf meta-environment*. In ASF-SDF'95, CWI Amsterdam, 1995, University of Amsterdam, Programming Research Group, Proceedings of ASF-SDF'95 workshop, May 11-12, 1995.

[6] P. Naur, Revised report on the algorithmic language algol60, *Communications of the ACM*, 1-17, 1963.

[7] T. Rus, *Algebraic construction of compilers*, Theoretical Computer Science, 90: 271-308, 1991.

[8] T. Rus, Algebraic processing of programming languages, *Theoretical Computer Science*, 105-143, 1998.

[9] T. Rus, *Algebraic definition of programming languages*, Theoretical Computer Science, 207-223, 2001

[10] D. R. Smith, *Towards a classification approach to design*, In Algebraic Methodology and Software Technology, AMAST'96, Proceedings, volume 1101 of Lecture Notes in Computer Sciences, 1997.

[11] E. Van Wyk, *Semantic Processing by Macro Processors*, PhD thesis, The University of Iowa, Department of Computer Science, Iowa City, IA 52242, July 1998.