Abstract: - This paper is an approach methodology of programming language, considerate like a tuple
$L = (\text{Sem}, \text{Syn}, l: \text{Sem} \rightarrow \text{Syn})$. 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
$L = (\text{Sem}, \text{Syn}, l: \text{Sem} \rightarrow \text{Syn})$
where $\text{Sem}$ is a $\Sigma$-algebra which is the language
semantic, $\text{Syn}$ is a $\Sigma$ word algebra which is the
language syntax, and $l$ is a partial mapping called the
language learning function. $l$ maps semantic
constructs in $\text{Sem}$ to their expressions as a syntactic
constructs in $\text{Syn}$ such that there exists a
complementary homomorphism $\varepsilon: \text{Syn} \rightarrow \text{Sem}$,
called the language evaluation function, that maps
expressions in $\text{Syn}$ to their semantic constructs in $\text{Sem}$.

When a programming language is defined in
this manner, the carrier sets in the semnatics $\text{Sem}$
contain the valid computations of the language and
the words in the syntax. $\text{Syn}$ are the expressions of
the computations. $l$ is a mechanism that allows a
programmer to learn to express computations in $\text{Sem}$
by a symbolic construct in $\text{Syn}$ such that for any
computation $c \in \text{Sem}$, $\varepsilon((l(c)) = c$.
The evaluation function is related to the ability to
express computation by the identity $l \circ \varepsilon = 1_{\text{Sem}}$
where $1_{\text{Sem}}$ is the identity map on $\text{Sem}$.

A language specification rule formally describes
elements of $\text{Syn}$, $\text{Sem}$ and the relationship between
them. Equations of the form $r: A_0 = t_0 A_1 t_1 A_2 t_2 \ldots A_n t_n$, $A_0$ is the left hand side
of the rule and is denoted by $\text{lhs}(r)$ and
t_0 A_1 t_1 A_2 t_2 \ldots A_n t_n is the right hand side of
the rule and is denoted by $\text{rhs}(r)$. 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 $W = N \cup T; e$ 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,
$\text{Syn}(R)$ is unique, while the semantic algebra, $\text{Sem}(R)$
may be a language user choice. The algebraic models
of language processing requires both $\text{Syn}(R)$ and
$\text{Sem}(R)$ to be specified by concrete notation.

Since $\text{Syn}(R)$ is unique, the major problem rests with the
notation used to express $\text{Sem}(R)$.

2 The algebraic model of the syntax

Definition 2: The syntax model of language
specified by $R$ is:
$\text{Syn} (R) = \left\{ [A_i] A_i \in N \right\} \left\{ [r] r \in R \right\}$,
where $[A_i] A_i \in N$ 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 $r \in R, r : A_0 = t_0, A_1 \ldots t_{n-1}, A_n$ is interpreted as an algebraic operation $\{r\} : [A_1] \times \ldots \times [A_n] \rightarrow [A_0]$ which constructs the elements $w_0 \in [A_0]$ from the elements $w_i \in [A_i]$, with $0 \leq i \leq n$, by the rule:
  \[ \{r\}(w_1, w_2, \ldots, w_n) = w_0 = t_0 w_1 t_1 w_2 \ldots t_{n-1} w_n t_n. \]

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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State = Ident “;” Expr</td>
<td>State = “if” Expr “then” State “else”</td>
</tr>
<tr>
<td>State = “while” Expr “do” State “od”</td>
<td>State = Bl</td>
</tr>
<tr>
<td>State1 = State</td>
<td>State1 = State “;” State1</td>
</tr>
<tr>
<td>IdentCt1 = Ident Type</td>
<td>IdentCt1 = IdentCt “;” IdentCt1</td>
</tr>
<tr>
<td>Bl = “begin” IdentCt1 “;” State1 “end”</td>
<td></td>
</tr>
</tbody>
</table>

The syntax interpretation (domain) of the parameters used in these rules is defined by the equations in Table 2:

- $[Expr]$ is set of expressions generated by identifiers and constants,
- $[Type]$ is set of given (predefined or defined) type names,
- $[State] = \{ident := \text{expr} | ident \in [Ident] \} \cup \{if \text{ then } e \text{ else } f | e, f \in [Expr] \}$,
- $[IdentCt] = \{ident \in [Ident] | t \in [Type] \}$,
- $[(\text{begin} \text{1} ; s) \text{ end}] = \{begin1; s1 \in [IdentCt1] \} \cup \{end1; s1 \in [IdentCt1] \}$.

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

\[ \text{Syn}(R) = \{ \{ [Ident] \} [Expr] [Type] [State] [State1] \} \cup \{ [IdentCt] [IdentCt1] [Bl] \} \cup \{ [r_1] [r_2] [r_3] [r_4] [r_5] [r_6] [r_7] [r_8] [r_9] [r_{10}] \}. \]

The syntax interpretation (domain) of the parameters used in these rules is defined by the equations in Table 2:

Fact 1: $\text{Syn}(R)$ is embedded in the semi group $W = \langle N \cup T, \circ, e \rangle$ by derived operations. A component relation, $\leq_{\text{Syn}}$, can now be defined by:
for all \( w,w' \in \text{Syn}(R) \), \( w \leq_{\text{Syn}} w' \) if \( w = w' \) or there is \( r \in R \) such that \( w' = [r]w_1,\ldots,w_k \) and \( w \leq_{\text{Syn}} w_i \) 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

\[
\text{Sem}(A) = \left\{ \left[ A_n \right] \mid A_n \in N, [r] \in R \right\},
\]

where \( [A_n] \) 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 \( r \in R, r : A_0 = t_0 A_0 t_1 \ldots t_n A_n t_n \) is interpreted as an algebraic operation \( \left[ [r] \right] : \left[ [A_0] \right] \times \ldots \times \left[ [A_n] \right] \rightarrow \left[ [A_0] \right] \) 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: \( \left[ [r] \right] c_1, c_2, \ldots, c_n = c_0 \).

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

\[
\left[ T_i \sigma \right] \xrightarrow{t} \left[ T_i \sigma \right] \xrightarrow{t} \ldots \xrightarrow{t} \left[ T_i \sigma \right] \xrightarrow{t} \left[ T_i \sigma \right],
\]

The initial state of a computation is a state whose variables and constants satisfy an initial condition:

\( \Theta : D \cup \{ \uparrow, \downarrow \} \rightarrow \{ \text{true, false} \} \), the final state of a computation in a state whose variables and constants satisfy a final condition:

\( \Phi : D \cup \{ \uparrow, \downarrow \} \rightarrow \{ \text{true, false} \} \).

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 \( \nu(t) = \{ [r] \}, t \in N \); state parameters \( \sigma \in N \) represent functions mapping sets of names (of variables and constants), \( D \), to values in the universe of their types, \( \sigma : D \rightarrow \{ [r] \}, t \in N \); mapping parameters \( \tau \in N \) represent state-transitions, mapping functions \( \sigma' : D' \rightarrow [r], t \in N \) into functions \( \sigma' : D' \rightarrow [r], t \in N \). These interpretations of the parameters in \( N \) can be unified as transitions, as follows:

\[
[p] = \left\{ \begin{array}{ll}
\left\langle p, \phi \right\rangle & \rightarrow \left\langle p, \phi \right\rangle, p = \text{type} \\
\left\langle t, \sigma : D \rightarrow \nu(t) \right\rangle & \rightarrow \left\langle t, \sigma : D \rightarrow \nu(t) \right\rangle, \\
\left\langle t, \sigma : D \rightarrow \nu(t) \right\rangle & \rightarrow \left\langle t, \sigma : D' \rightarrow \nu(t') \right\rangle, \\
p = \text{state} & \\
\left\langle t, \sigma : D \rightarrow \nu(t) \right\rangle & \rightarrow \left\langle t, \sigma' : D' \rightarrow \nu(t') \right\rangle, \\
p = \text{transformation} & \end{array} \right.
\]

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

\[
\text{Sem}(R) = \left\{ \left[ \text{Ident} \right] \left[ \text{Expr} \right] \left[ \text{Type} \right] \left[ \text{State} \right] \left[ B1 \right] \right\},
\]

where the operations \[ \left[ f \right] \] through \[ \left[ r_{10} \right] \] are in Table 5.

**Fact 2:** \[ \text{Sem}(R) \] is embedded in the semi group \( \langle \left[ A \right] \mid A \in N', \circ, \cdot \rangle \) by derived operations.

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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Semantic Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \left[ f \right] \left[ e \right] \left[ \text{Expr} \right] \rightarrow \left[ \text{State} \right] )</td>
<td>( \left( x \rightarrow \sigma (x), e \right) )</td>
</tr>
<tr>
<td>( \left[ f \right] \left[ e \right] \left[ \text{Expr} \right] \rightarrow \left[ \text{State} \right] )</td>
<td>( \left( \text{true}, \circ,\cdot\right) )</td>
</tr>
<tr>
<td>( \left[ f \right] \left[ \text{Type} \right] \left[ \text{State} \right] \rightarrow \left[ \text{State} \right] )</td>
<td>( \left( \circ,\cdot\right) )</td>
</tr>
<tr>
<td>( \left[ f \right] \left[ \text{Type} \right] \left[ \text{State} \right] \rightarrow \left[ \text{State} \right] )</td>
<td>( \left( \circ,\cdot\right) )</td>
</tr>
<tr>
<td>( \left[ f \right] \left[ \text{Type} \right] \left[ \text{State} \right] \rightarrow \left[ \text{State} \right] )</td>
<td>( \left( \circ,\cdot\right) )</td>
</tr>
</tbody>
</table>

We illustrate the construction of model algebra \( \text{Sem}(R) \) using the specification rules given in Table 1, assuming that \[ \left[ \text{Ident} \right] \] is the universe of names, \[ \left[ \text{Expr} \right] \] is the universe of values, and \[ \left[ \text{Type} \right] \] is the set of types. For \( x \in \left[ \text{Ident} \right] \), \( e \in \left[ \text{Expr} \right] \) and \( t \in \left[ \text{Type} \right] \), we denote with \( \tau_x \), \( \tau_e \) and \( \tau_t \), respectively. In addition, \( \left[ e \right] \in \left[ \text{Expr} \right] \) 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; \[ \left[ \text{State} \right] \] is shown in Table 4.

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

\[
\tau_1 = \left( T_1, D_1, \uparrow, w \right) \rightarrow \left( T'_1, D'_1, \downarrow \right),
\]

\[
\tau_2 = \left( T_2, D_2, \uparrow, w \right) \rightarrow \left( T'_2, D'_2, \downarrow \right),
\]

and the table 5 follows:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Semantic Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \left[ f \right] \left[ \text{Type} \right] \left[ \text{State} \right] \rightarrow \left[ \text{State} \right] )</td>
<td>( \left( \circ,\cdot\right) )</td>
</tr>
<tr>
<td>( \left[ f \right] \left[ \text{Type} \right] \left[ \text{State} \right] \rightarrow \left[ \text{State} \right] )</td>
<td>( \left( \circ,\cdot\right) )</td>
</tr>
<tr>
<td>( \left[ f \right] \left[ \text{Type} \right] \left[ \text{State} \right] \rightarrow \left[ \text{State} \right] )</td>
<td>( \left( \circ,\cdot\right) )</td>
</tr>
</tbody>
</table>
\( \tau_1 \leq_{\text{Sem}} \tau_2 \) if \( \tau_{w'} = E(\tau_w, o, t) \).

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

References: