Annales Univ. Sci. Budapest., Sect. Comp. 19 (2000) 211-224

INFIX AND POSTFIX NOTATIONS

A. and D. Popovici (Timişoara, Roumania)

Abstract. Let (A, Ω) be a universal algebra having finite operators domain. Evaluating the infix (postfix) expressions over (A, Ω) we obtain a characterization for the closure operator associated to this universal algebra. Using a recursive method it proves that the language of all infix (postfix) expressions is unambiguous. Finally, constructing a matrix grammar, it obtains an isomorphism between the languages of the two expressions types mentioned above. It is also presented an example in which we apply the most important results obtained in this paper.

1. Introduction

The compilers are necessary parts of any computer. The compiler construction is an important research domain in computer science. A compiler is, in fact, a program written in a certain language.

There exist situations in which the compiler transforms the source program in an internal form easier to process by the computer. For example, we can mention here the existence of a limited space of memory or of a source language too complicated.

Usually, in the most internal forms, the operators appear in the order in which they are to be applied on the operands, which has a big utility in code generation, subsequent analysis, and also in interpreting (that is the realization of a program which is able to execute the source program as it is represented in its internal form).

Among the more frequently used internal forms we mention here the Polish notation, quadruples and triples (to see for example [1], [3]). Also, for the code optimization are used blocks and a program graph (for example we can see in [5] how FORTRAN H on the IBM 360 uses such a scheme with quadruples), and, in several translator writing systems, syntax trees in a list-structured form (to see in [2] how the ALGOL W compiler uses them). Generally the compiler uses a mixture of several such internal forms and not only a single internal form exclusively. All the internal forms have in common the operators and the operands, the difference being in how these are connected.

In this paper we refer to one of the internal forms mentioned above, namely the Polish notation (postfix notation or reverse Polish string or suffix notation).

The Polish notation was introduced for the first time by Lukasiewicz in order to obtain the recognition of correct sentences in formal logics. Later, this technique proved to be useful for a larger class of context-free languages, those generated by operator grammars (to see [4]).

There is not a general way to approach the postfix concepts and techniques.

In this paper we propose ourselves to deal with the problem of writing some infix expressions in a unique postfix form.

2. Notations and preliminaries

The Polish notation is useful for the representation of arithmetical and logical expressions in order to simplify and specify the exact evaluation order of operators.

In both notations, the infix one (the usual notation in which the operators appear between the operands) and the postfix one (the operators can be found immediately after their operands), the operands appear in the same order, but in addition, in the last notation, no parentheses are needed.

We shall mention now a few methods to obtain the postfix version of an infix expression:

• in *the syntactical analysis method* we attach to each grammar rule which generates the infix expression a semantic routine, construct the derivation tree associated to this expression and successively reduce the simple left sentence; to each reduction we appeal the corresponding semantic routine thus generating the postfix form of the infix expression.

• In *the technique of parantheses* we associate to each operator a weight and we put between parentheses each operation (to be clear the order of application of the operators), we shall read the expression from left to right ignoring the left parentheses, putting the operands in a list and the operators in a stack, and, when we find a right parenthesis, the operator in the top of the stack goes to the list.

• *The technique of the operators stack* uses two linear lists (one for the infix expression, and the other for its postfix version) and a stack (where we put the operators). Reading the first list from left to right, we write the operands directly in the output string (the postfix form), and the operators go in and out from the stack according to some rules which depend on their associated weights.

For the method, which we shall present in this paper, we shall define recursively the postfix version of an infix expression.

Let (A,Ω) be a universal algebra having the domain of operators finite. For an operator $\omega : A^n \to A$, n is said to be *its arity* and is denoted by $\tau(\omega)$. We say that τ is *the type* of the universal algebra (A,Ω).

For $\omega \in \Omega$, $S_1, S_2, \cdots, S_{\tau(\omega)}$ symbols and Σ_ω an alphabet of symbols which describe the operator ω, we define $\tilde{\omega}(S_1, S_2, \cdots, S_{\tau(\omega)}) \in \Big(\{S_1, S_2, \cdots, S_{\tau(\omega)}\} \cup \cup \Sigma_\omega\Big)^+$. We shall suppose that each of the symbols $S_1, S_2, \cdots, S_{\tau(\omega)}$ appears exactly once, in this order, in $\tilde{\omega}(S_1, S_2, \cdots, S_{\tau(\omega)})$. This word will also contain the symbols of Σ_ω. For example, if $\tau(\omega) = 2$ we can define $\tilde{\omega}(S_1, S_2) = S_1 \omega S_2$.

The infix expressions over the universal algebra (A,Ω) can be presented recursively in the following way:

- if $a \in A$, a is an infix expression;
- if $\omega \in \Omega$, $e_1, e_2, \cdots, e_{\tau(\omega)}$ are infix expressions then $\Big(\tilde{\omega}(e_1, e_2, \cdots, e_{\tau(\omega)})\Big)$ is an infix expression.

Eliminating the parentheses and taking, for $\omega \in \Omega$ and $S_1, S_2, \cdots, S_{\tau(\omega)}$ symbols,

$$\tilde{\omega}(S_1, S_2, \cdots, S_{\tau(\omega)}) = S_1 S_2 \cdots S_{\tau(\omega)} \omega,$$

we shall obtain in the same way *the postfix expressions* over (A,Ω).

We can now define *the postfix version* of a given infix expression:

- if $a \in A$, a is the postfix version of the infix expression a;
- if $\omega \in \Omega$ and for each $k \in \{1, 2, \cdots, \tau(\omega)\}$, e_p^k is the postfix version of the infix expression e_i^k then $e_p^1 e_p^2 \cdots e_p^{\tau(\omega)} \omega$ is the postfix version of the infix expression $\Big(\tilde{\omega}(e_i^1, e_i^2, \cdots, e_i^{\tau(\omega)})\Big)$.

For an infix or postfix expression e we denote by $Op(e)$ the number of operators which are used in the description of the expression (at each operator we shall calculate its number of appearances).

3. Infix and postfix expressions

In the following we shall present other construction methods for infix and postfix expressions over the universal algebra (A, Ω).

Let

$$V_N = \{S\}, \ V_T = \{s\} \text{ and the grammars}$$
$$G_i = \Big(V_N, V_T \cup \{(,)\} \cup \Omega, S, \mathcal{P}_i\Big),$$
$$G_p = \Big(V_N, V_T \cup \Omega, S, \mathcal{P}_p\Big)$$

having the productions

$$\mathcal{P}_i = \Big\{S \to \Big(\tilde{\omega}(\underbrace{S, S, \cdots, S}_{\tau(\omega)})\Big), \ \omega \in \Omega\Big\} \cup \{S \to s\} \text{ and}$$
$$\mathcal{P}_p = \Big\{S \to \underbrace{SS \cdots S}_{\tau(\omega)}\omega, \ \omega \in \Omega\Big\} \cup \{S \to s\}.$$

The words of the language $L(G_i)$ are simply called *infix expressions,* and those of the language $L(G_p)$ *postfix expressions.*

For each subset $B \subset A$ we shall define the mapping

$$\sigma_B : V_T \cup \{(,)\} \cup \Omega \to \mathcal{P}(A) \cup \{(,)\} \cup \Omega,$$
$$\sigma_B(s) = B, \quad \sigma_B(x) = x \quad (x \neq s)$$

which can be extended to $\Sigma^+ = \Big(V_T \cup \{(,)\} \cup \Omega\Big)^+$ using the relation

$$\sigma_B(xy) = \sigma_B(x)\sigma_B(y), \quad x, y \in \Sigma^+.$$

Without difficulty we can observe that

Proposition 3.1. *$\sigma_A(L(G_i))$ (respectively $\sigma_A(L(G_p))$) represents the set of all infix (respectively postfix) expressions over the universal algebra (A, Ω).*

Remark 3.2. If F is a finite set of symbols (variables over A) with the help of the mapping

$$\sigma_F : V_T \cup \{(,)\} \cup \Omega \to \mathcal{P}(F) \cup \{(,)\} \cup \Omega,$$
$$\sigma_F(s) = F, \quad \sigma_F(x) = x \quad (x \neq s)$$

we can obtain all the infix or postfix expressions having at most *cardF* elements of A. More exactly

$$\bigcup_{\substack{B\subset A\\ cardB=n}} \sigma_B(L(G_i)) \text{ (respectively } \bigcup_{\substack{B\subseteq A\\ cardB=n}} \sigma_B(L(G_p)))$$

represents the set of all infix (postfix) expressions over (A,Ω) having at most n symbols from A.

We consider a sequence $(L^n(G_i))_{n\in\mathbf{N}}$ of languages recurrently defined by the relations

$$\begin{aligned} L^0(G_i) &=\{s\},\\ L^{n+1}(G_i) &=L^n(G_i)\cup\Big\{\Big(\tilde{\omega}(e_1,e_2,\cdots,e_{\tau(\omega)})\Big)\,|\\ &\qquad \omega\in\Omega,\ e_k\in L^n(G_i),\ k=\overline{1,\tau(\omega)}\Big\}. \end{aligned} \tag{1}$$

Denote by $Op^n(G_i)$ the set of all infix expressions $e_i\in L(G_i)$ with $Op(e_i)=n$. Obviously, with these notations,

$$L(G_i)=\bigcup_n Op^n(G_i).$$

Remark 3.3. If $e_i\in L^n(G_i)$, $n\neq 0$, then

$$Op(e_i)\le 1+m+\cdots+m^{(n-1)},\ \text{ where } m=\max_{\omega\in\Omega}\tau(\omega).$$

For this purpose, let $x_n=\max_{e_i\in L^n(G_i)} Op(e_i)$, $n\in\mathbf{N}^*$. Obviously $x_0=0$, $x_1=$ $=1$. Because for $e_i\in L^n(G_i)$ with $Op(e_i)=x_n$, $e_i'=\Big(\tilde{\omega}(\underbrace{e_i,e_i,\cdots,e_i}_{\tau(\omega)})\Big)$ has $Op(e_i')=x_{n+1}$. We deduce the recurrent relation

$$x_{n+1}=mx_n+1,\quad n\in\mathbf{N}.$$

The conclusion it obtains easily.

Proposition 3.4. $L(G_i)=\bigcup_{n\in\mathbf{N}} L^n(G_i)$.

Proof. Observe for the beginning that $Op^n(G_i)\subseteq L^n(G_i)$, $n\in\mathbf{N}$. We shall verify this property by induction. It is obvious that $Op^1(G_i)\subseteq L^1(G_i)$. In order to make the induction step observe that if $e\in Op^{n+1}(G_i)$, then there exists $\omega \in \Omega$, $e_1, e_2, \cdots, e_{\tau(\omega)}$ infix expressions such that $e =$ $= \left(\widetilde{\omega}(e_1, e_2, \cdots, e_{\tau(\omega)})\right)$. Since $Op(e_k) \le n$, $k = \overline{1, \tau(\omega)}$ the expressions e_k will be words in $L^n(G_i)$ which shows that $e \in L^{n+1}(G_i)$.

So, we obtained that

$$L(G_i) = \bigcup_{n \in \mathbf{N}} Op^n(G_i) \subset \bigcup_{n \in \mathbf{N}} L^n(G_i)$$

and the conclusion follows immediately.

Definition 3.5. *The closure operator* $J : \mathcal{P}(A) \to \mathcal{P}(A)$, associated to the universal algebra (A, Ω) of the type τ, can be defined by the relation

$$\mathcal{J}(B) = \bigcup_{n \in \mathbf{N}} \mathcal{J}^n(B) \quad B \subset A,$$

the sets $\mathcal{J}^n(B)$, $n \in \mathbf{N}$ being recurrently built with the help of the formula

$$\begin{aligned} &\mathcal{J}^0(B) = B, \\ (2) \qquad &\mathcal{J}^{n+1}(B) = \mathcal{J}^n(B) \cup \Big\{\omega(e_1, e_2, \cdots, e_{\tau(\omega)}) | \\ &\qquad \omega \in \Omega, \ e_k \in \mathcal{J}^n(B), \ k = \overline{1, \tau(\omega)}\Big\}, \quad n \in \mathbf{N}. \end{aligned}$$

For an infix or postfix expression e over the universal algebra (A, Ω) we shall denote by $Ev(e)$ the element of A obtained by the evaluation of the expression.

Proposition 3.6. *For each* $B \subset A$

$$\mathcal{J}(B) = Ev(\sigma_B(L(G_i))).$$

Proof. Using the result presented in the Proposition 3.4

$$L(G_i) = \bigcup_{n \in \mathbf{N}} L^n(G_i) \text{ we obtain } \sigma_B(L(G_i)) = \bigcup_{n \in \mathbf{N}} \sigma_B(L^n(G_i)), \quad B \subset A.$$

Acting inductively, with the help of the formulas (1) and (2), we find $\mathcal{J}^n(B) =$ $= Ev(\sigma_B(L^n(G_i)))$, $n \in \mathbf{N}$. Consequently

$$\mathcal{J}(B) = \bigcup_{n \in \mathbf{N}} \mathcal{J}^n(B) = \bigcup_{n \in \mathbf{N}} Ev(\sigma_B(L^n(G_i))) = Ev(\sigma_B(L(G_i))).$$

Remark 3.7. We can obtain for the grammar G_p, likewise, results similar to those presented for the grammar G_i.

4. The postfix notation

In this section we shall build an isomorphism between the languages of infix, respectively postfix expressions over a universal algebra (A, Ω). Let us recall (to see for example [7]) that a grammar is *unambiguous* if each generated word has a unique leftmost derivation. The language generated by it is also called unambiguous.

Lukasiewicz introduced the postfix notation in order to obtain for an expression a recognized form without delimiters. The property of G_p to be unambiguous proves that. For the sake of completion we give here an original proof. Also we verify this property for the language of all infix expressions.

Proposition 4.1. *$L(G_i)$ is unambiguous.*

Proof. We shall demonstrate by induction after $n \in \mathbf{N}$ the proposition

(3) $\quad e_i \in L(G_i)$ with $Op(e_i) \leq n$ has a unique leftmost derivation.

For the beginning let us observe that there exists exactly one expression e_i with $Op(e_i) = 0$, namely $e_i = s$, that for which the only leftmost derivation is $S \Rightarrow s$. Furthermore, if $Op(e_i) = 1$, e_i will have the form $e_i = \Big(\tilde{\omega}(\underbrace{s, s, \cdots, s}_{\tau(\omega)})\Big)$ for some $\omega \in \Omega$. Applying the production $S \Rightarrow s$ for the first leftmost symbol S, the only leftmost derivation is

$$S \Rightarrow \Big(\tilde{\omega}(\underbrace{S, S, \cdots, S}_{\tau(\omega)})\Big) \Rightarrow \Big(\tilde{\omega}(s, \underbrace{S, \cdots, S}_{\tau(\omega)-1})\Big) \overset{*}{\Rightarrow} \Big(\tilde{\omega}(\underbrace{s, s, \cdots, s}_{\tau(\omega)})\Big) = e_i.$$

Suppose now that (3) is true for a certain n. Let $e_i \in L(G_i)$ with $Op(e_i) = n+1$. The derivation of e_i will have the form

$$(4) \quad S \Rightarrow \Big(\tilde{\omega}(\underbrace{S, S, \cdots, S}_{\tau(\omega)})\Big) \overset{*}{\Rightarrow} \Big(\tilde{\omega}(e_1, e_2, \cdots, e_{\tau(\omega)})\Big) = e_i, \quad \text{for some } \omega \in \Omega,$$

e_k, $k = \overline{1, \tau(\omega)}$ being infix expressions with $Op(e_k) \leq n$. Using the induction hypothesis and applying for each e_k the corresponding leftmost derivation, (4) becomes a leftmost derivation for e_i. We shall see in the following that this is the only one.

Let us suppose that there exists another leftmost derivation for e_i,

(5)
$$S \Rightarrow \left(\tilde{\omega}'(\underbrace{S, S, \cdots, S}_{\tau(\omega)})\right) \overset{*}{\Rightarrow} \left(\tilde{\omega}'(e'_1, \underbrace{S, \cdots, S}_{\tau(\omega)-1})\right) \overset{*}{\Rightarrow} \left(\tilde{\omega}'(e'_1, e'_2, \cdots, e'_{\tau(\omega')})\right) = e_i.$$

We shall prove firstly that $\omega' = \omega$. Starting from the form of the expression $\tilde{\omega}(e_1, e_2, \cdots, e_{\tau(\omega)})$ we shall obtain $\tilde{\omega}(S_1, S_2, \cdots, S_{\tau(\omega)})$, S_k, $k = \overline{1, \tau(\omega)}$ being symbols, using the following recursive proceeding:

- because we do not know the arity of ω we shall consider, for the beginning, a number of symbols $S_1, S_2, \cdots$ equal with the length of the word $\tilde{\omega}(e_1, e_2, \cdots, e_{\tau(\omega)})$;
- put the symbols $S_1, S_2, \cdots$ in a stack, this order being that of going out;
- search, from left to right, for the first terminal belonging to the set $\{s, (\}$;
- if this terminal is s we replace it with the first symbol from the stack, symbol which will be taken out from this;
- if this terminal is a left parenthesis we shall replace the expression contained between this parenthesis and the corresponding right one, so a word of the form (e), with the first symbol from the stack which will be taken out from this;
- repeat this process beginning with the third step until we finish all the terminals of the word $\tilde{\omega}(e_1, e_2, \cdots, e_{\tau(\omega)})$.

On the place of the expression $\tilde{\omega}(e_1, e_2, \cdots, e_{\tau(\omega)})$ it obtains $\tilde{\omega}(S_1, S_2, \cdots, S_{\tau(\omega)})$. Since

$$\tilde{\omega}(S_1, S_2, \cdots, S_{\tau(\omega)}) \equiv \tilde{\omega}'(S'_1, S'_2, \cdots, S'_{\tau(\omega)})$$

$$\text{iff } \omega = \omega' \text{ and } S_k = S'_k, \quad k = \overline{1, \tau(\omega)} = \overline{1, \tau(\omega')}$$

it obtains the conclusion.

The derivation will have the form

$$S \Rightarrow \left(\tilde{\omega}(\underbrace{S, S, \cdots, S}_{\tau(\omega)})\right) \overset{*}{\Rightarrow} \left(\tilde{\omega}(e'_1, \underbrace{S, \cdots, S}_{\tau(\omega)-1})\right) \overset{*}{\Rightarrow} \left(\tilde{\omega}(e'_1, e'_2, \cdots, e'_{\tau(\omega)})\right) = e_i.$$

The equality $e_k = e'_k$, $k = \overline{1, \tau(\omega)}$ can be proved in a similar way with that described above. For example e_1 (and e'_1) represents the first terminal s found or the expression of the form (e) found between the first left parenthesis and the right corresponding one.

Finally observe that we cannot have two different derivations of the type (4) because, in this case, we obtain for at least one infix expression e_k, $k = \overline{1,\tau(\omega)}$ two different leftmost derivations. But, in this manner, we contradict the hypothesis.

Example 4.2. If in the infix expressions definition we do not use parentheses, the unambiguity of G_i, proved in the previous proposition, can be easily lost. An example is the following:

$$\Omega = \{+\},\ \tilde{+}(S_1,S_2) = S_1+S_2,\quad G_i = \Big(\{S\},\{s,+\},S,\{S\to s,\ S\to S+S\}\Big).$$

The word $s+s+s$ has two different leftmost derivations, namely

$$S \Rightarrow S+S \Rightarrow s+S \Rightarrow s+S+S \Rightarrow s+s+S \Rightarrow s+s+s,$$
$$S \Rightarrow S+S \Rightarrow S+S+S \Rightarrow s+S+S \Rightarrow s+s+S \Rightarrow s+s+s.$$

Proposition 4.3. *$L(G_p)$ is unambiguous.*

Proof. We shall act inductively, in a similar way with that presented in the Proposition 4.1 proving, for $n \in \mathbf{N}$, the proposition

(6) $\quad e_p \in L(G_p)$ with $Op(e_p) \le n$ has a unique leftmost derivation.

Insist only on the induction step. Suppose that (6) is true for a certain $n \in \mathbf{N}$. Let $e_p \in L(G_p)$ with $Op(e_p) = n+1$. A leftmost derivation of e_p will have the form

(7)
$$S \Rightarrow \underbrace{SS\cdots S}_{\tau(\omega)}\omega \overset{*}{\Rightarrow} e_1\underbrace{S\cdots S}_{\tau(\omega)-1}\omega \overset{*}{\Rightarrow} e_1e_2\cdots e_{\tau(\omega)}\omega = e_p,$$

for a certain $\omega \in \Omega$,

e_k, $k = \overline{1,\tau(\omega)}$ being postfix expressions with $Op(e_k) \le n$ obtained from a leftmost derivation.

In order to prove the uniqueness of the expressions $e_1, e_2, \cdots, e_{\tau(\omega)}$ which describe the postfix expression e_p in (7) we shall use, as in the Proposition 4.1, a recursive algorithm. We shall present in the following this algorithm using a Pascal sequence.

We shall read the word e_p from the right to left separating the expressions $e_{\tau(\omega)}, e_{\tau(\omega)-1}, \cdots, e_1$, for this taking into acount the arity of each operator found (the arity being specified by a function A). Each expression e_k, $k = \overline{1,\tau(\omega)}$, different from the *s* symbol, is built in the procedure *PROC* with the help of the variable *ex*.

```
procedure PROC (a:string; var ex:string);
var j:integer;
    op:char;
begin
    op := a;
    for j := 1 to A(op) do begin
                            i := i - 1;
                            a := st[i];
                            ex := a + ex;
                            if a <>'s' then PROC (a, ex);
                            end;
end;
begin
    write('The string to analyse:');
    readln(st);
    l :=length(st);
    i := l - 1;
    a := st[i];
    for k := A(st[l]) downto 1 do begin
                                  if a ='s' then e[k] :='s'
                                          else begin
                                              e[k] := a;
                                              ex := e[k];
                                              PROC (a, ex);
                                              e[k] := ex;
                                              end;
                                  i := i - 1;
                                  a = st[i];
                                  end;
    for k := 1 to A(st[l]) do writeln('e[',k,']=', e[k]);
end.
```

We cannot have two different derivations for e_p of the form (7) because, in this case, at least one expression e_k, $k = \overline{1, \tau(\omega)}$ will admit two different leftmost derivations. But in this way we contradict the induction hypothesis.

Theorem 4.4. *Every infix expression $e_i \in L(G_i)$ admits a unique postfix version $e_p \in L(G_p)$.*

Proof. Consider the matrix grammar

$$G_{i,p} = \Big(\{S, S_i, S_p\}, \{s, \&\} \cup \{(,)\} \cup \Omega, S, \mathcal{P}_{i,p}\Big)$$

with the productions

$$\mathcal{P}_{i,p} = \Big\{S \to S_i \& S_p,\ [S_i \to s, S_p \to s],$$
$$\Big[S_i \to \Big(\tilde{\omega}(\underbrace{S_i, S_i, \cdots, S_i}_{\tau(\omega)})\Big), S_p \to \underbrace{S_p S_p \cdots S_p}_{\tau(\omega)} \omega\Big]_{\omega \in \Omega}\Big\}.$$

The main properties of the matrix grammars can be found, for example, in [6]. Let $e_i \in L(G_i)$ be an infix expression such that $e_i \& e_p^i \in L(G_{i,p})$. Applying the productions which generated the expression e_i we obtain that there exists $e_p^i \in L(G_p)$. The productions of the form

$$\Big[S_i \to \Big(\tilde{\omega}(\underbrace{S_i, S_i, \cdots, S_i}_{\tau(\omega)})\Big), S_p \to \underbrace{S_p S_p \cdots S_p}_{\tau(\omega)} \omega\Big], \quad \omega \in \Omega$$

keep the recursive modality of the postfix version definition of an infix expression and consequently e_p^i is a postfix version for e_i.

Because the grammars G_i and G_p are unambiguous, for a word of the form $e_i \& e_p \in L(G_{i,p})$ there exists a unique leftmost derivation. Let us suppose now that e_p' is another postfix version for e_i. Obviously $e_i \& e_p' \in L(G_{i,p})$. Writing a leftmost derivation for $e_i \& e_p'$ we obtain a leftmost derivation for e_i. The last one being unique we obtain in fact that $e_p' = e_p$.

Corollary 4.5. *The mapping $\varphi : L(G_i) \to L(G_p)$, $\varphi(e_i) = e_p^i$, defined by the relation $e_i \& e_p^i \in L(G_{i,p})$, is an isomorphism.*

Proof. Using the Theorem 4.4 φ is well defined and injective. This mapping is also onto. Let us observe that if $e_p \in L(G_p)$, using a leftmost derivation for e_p we obtain a leftmost derivation for an expression $e_i \& e_p \in$ $\in L(G_{i,p})$. Obviously $e_p = e_p^i = \varphi(e_i)$ and the proof is finished.

If $\overline{e_i}$ is an infix expression over the universal algebra (A, Ω) replacing all the operands with the s symbol we obtain $e_i \in L(G_i)$.

Replacing now each s symbol of the expression $e_p^i = \varphi(e_i)$ with the elements of A used for $\overline{e_i}$, in the same order, we obtain an expression $\overline{e_p^i}$, the postfix version of $\overline{e_i}$. Associating this $\overline{e_p^i}$ to each $\overline{e_i}$ we obtain

Theorem 4.6. *There exists an isomorphism between the set of all infix expressions over a universal algebra with the domain of operators finite and the set of the postfix expressions.*

Finally we shall present an example in which we apply the results obtained above.

Example 4.7. Consider a universal algebra (A, Ω), A being a set of sequences (simple sequences like conditions, arithmetical and logic expressions, certain Pascal sequences and also the empty sequence), and the operators set

$$\Omega = \{i, w, b, r, f, =\}$$

corresponding respectively, to the instructions of the Pascal language *if* , *while, begin-end (the compound instruction), repeat, for* and *the instruction of assignation.*

If S_1, S_2, S_3 are symbols we have

$$\begin{aligned} \tilde{i}(S_1, S_2, S_3) &= if \;\; S_1 \;\; then \;\; S_2 \;\; else \;\; S_3; \\ \tilde{w}(S_1, S_2) &= while \;\; S_1 \;\; do \;\; S_2; \\ \tilde{b}(S_1) &= begin \;\; S_1 \;\; end; \\ \tilde{r}(S_1, S_2) &= repeat \;\; S_1 \;\; until \;\; S_2; \\ \tilde{f}(S_1, S_2, S_3) &= for \;\; S_1 \;\; to \;\; S_2 \;\; do \;\; S_3; \\ \tilde{=}(S_1, S_2) &= S_1 := S_2. \end{aligned}$$

Consider the following Pascal sequence:

```
for s1 to s2 do
        if s3 then while s4 do s5
             else if s6 then begin s7 end
                      else s8 := s9
```

which can be written as an infix expression by

$$(for\ s_1\ to\ s_2\ do\ (if\ s_3\ then\ (while\ s_4\ do\ s_5)\ else$$
$$(if\ s_6\ then\ (begin\ s_7\ end)\ else\ (s_8 := s_9)))).$$

The postfix version of this expression it obtains in the matrix grammar presented in the Theorem 4.4, using the following direct leftmost derivations:

$S \Rightarrow S_i\&S_p \Rightarrow (\ for\ S_i\ to\ S_i\ do\ S_i\ )\&S_pS_pS_pf \Rightarrow$

$\Rightarrow (\ for\ s\ to\ S_i\ do\ S_i\ )\&sS_pS_pf \Rightarrow (\ for\ s\ to\ s\ do\ S_i\ )\&ssS_pf \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ S_i\ then\ S_i\ else\ S_i\ ))\&ssS_pS_pS_pif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ S_i\ else\ S_i\ ))\&sssS_pS_pif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (while\ S_i\ do\ S_i)\ else\ S_i))\&sssS_pS_pwS_pif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (\ while\ s\ do\ S_i\ )\ else\ S_i\ ))\&ssssS_pwS_pif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (\ while\ s\ do\ s\ )\ else\ S_i\ ))\&ssssswS_pif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (while\ s\ do\ s)\ else\ (\ if\ S_i\ then\ S_i\ else\ S_i)))$
$\&ssssswS_pS_pS_piif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (\ while\ s\ do\ s)\ else\ (\ if\ s\ then\ S_i\ else\ S_i)))$
$\&ssssswsS_pS_piif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (\ while\ s\ do\ s\ )\ else\ (\ if\ s\ then$
$(\ begin\ S_i\ end\ )\ else\ S_i\ )))\&ssssswsS_pbS_piif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (\ while\ s\ do\ s\ )\ else\ (\ if\ s\ then$
$(\ begin\ s\ end\ )\ else\ S_i\ )))\&ssssswssbS_piif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (\ while\ s\ do\ s)\ else\ (\ if\ s\ then$
$(\ begin\ s\ end\ )\ else\ (\ S_i := S_i\ ))))\&ssssswssbS_pS_p = iif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (\ while\ s\ do\ s\ )\ else\ (\ if\ s\ then$
$(\ begin\ s\ end\ )\ else\ (\ s := S_i\ ))))\&ssssswssbsS_p = iif \Rightarrow$

$\Rightarrow (\ for\ s\ to\ s\ do\ (\ if\ s\ then\ (\ while\ s\ do\ s\ )\ else\ (\ if\ s\ then$
$(\ begin\ s\ end\ )\ else\ (\ s := s\ ))))\&ssssswssbss = iif.$

Replacing the s symbols it obtains the postfix version of the given expression:

$$s_1s_2s_3s_4s_5ws_6s_7bs_8s_9 = iif.$$

References

[1] **Aho A.V. and Ullman J.D.**, *Principles of compiler design,* Addison-Wesley, 1977.

[2] **Bauer H., Becker S. and Graham S.**, *ALGOL W implementation,* CS 98, Computer Science Dept., Stanford Univ., 1968.

[3] **Gries D.**, *Compiler construction for digital computers,* John Wiley & Sons Inc., New York, 1971.

[4] **Jebelean T.**, A theory on postfix notation, *SIAN,* West University of Timişoara, **18** (1984).

[5] **Lowry E. and Medlock C.**, Object code optimization, *Comm. of ACM,* **12** (1969), 13-22.

[6] **Păun Gh.**, *Gramatici matriciale,* Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1981.

[7] **Salomaa A.**, *Formal languages,* Academic Press, New York, 1973.

(Received December 11, 1997)

A. and D. Popovici
Department of Mathematics
West University of Timişoara
Bd. V. Pârvan 4.
1900 Timişoara, Roumania
dianap, danp@tim1.uvt.ro