Volodymyr OVSYAK [1], Oleksandr OVSYAK [2]
[1] KIELCE UNIVERSITY OF TECHNOLOGY, Al. Tysiaclecia Panstwa Polskiego 7, 25-314 Kielce, Poland;
UKRAINIAN UNIVERSITY OF PRINTING, Pid Goloskom 19, 79-020 Lviv, Ukraine
[2] NATIONAL UNIVERSITY OF CULTURE AND ARTS, Shuchevitsha 5, 79-020 Lviv, Ukraine

The analysis of algorithm algebra formulae in *xml*-format

Prof. Ph.D. eng. Volodymyr OVSYAK

He specializes in theoretical and applied computer science, theory of algorithms, programming, information systems, mathematical modeling of systems, computer simulation and mathematical modeling. Currently he works as an professor in Kielce University of Technology

e-mail: ovsyak@rambler.ru

Ph.D. eng. Oleksandr OVSYAK

He specializes in theoretical and applied computer science, theory of algorithms, programming, information systems, mathematical modeling of systems, computer simulation and mathematical modeling. He works as an associate professor in National University of Culture and Arts in Lviv, Ukraine.

e-mail: ovsjak@ukr.net

Abstract

Analysis of formulae of algebra algorithms (AA) written in *xml* - format is described in the paper. A specific editor uses the *xml* - format for AA formulae writing to and reading from the computer memory. The *xml* – format contains operation types, operation orientations, operation uniterm separation, and AA operation uniterms. There are shown features of algorithm formula transforms, the result of which are 5 times shorter algorithms while saving all algorithm functionalities.

Keywords: editor, *xml* - format, algorithm formula, uniterm.

Analizy formuł algebry algorytmów w formacie *xml*

Streszczenie

Istniejąca algebra algorytmów (AA) zawiera specjalne znaki operacji, jakich nie ma wśród znaków matematycznych. Znaki mają skomplikowane formy. Znaki operacji mogą być stworzone z wykorzystaniem istniejących edytorów, takich jak na przykład Word. Jednak proces ich tworzenia jest bardzo skomplikowany i czasochłonny. Z tego powodu dla komputerowego edytowania formuł algebry algorytmów został stworzony specjalny edytor, którego główne okienko przedstawiono na rys.1. Dla zapisu formuł algebry algorytmów w pamięci komputera został stworzony specjalny format *xml*. Opisano format *xml*, służący do zapisu formuł algebry algorytmów do pamięci komputera. Zbudowano dwie formuły do analizy formatu xml, wykorzystane do identyfikacji i zapisu typów, orientacji i separatorów oraz unitermów operacji algebry algorytmów. Udowodniono, że te formuły umożliwiają wykonanie analizy formatu *xml*. Pokazano możliwości przekształcenia formuł algorytmów. W wyniku przekształceń możliwe jest 5 – krotne zmniejszenie liczby unitermów, przy zachowaniu wszystkich funkcjonalności formuły algorytmu.

Słowa kluczowe: edytor, format *xml*, formuła algorytmu, uniterm.

1. Introduction

Algorithm algebra (AA) gives means for algorithms description in the mathematic form. There are possible identical transforms of algorithm formulas as ones of mathematical expressions. The result of such transforms are reduction of algorithm formula uniterm number, time creation and fulfillment of program code saving, and simultaneous program code saving. Using the algorithm algebra [1, 2] for synthesis and algorithm formula transforms is illustrated by two examples of constructions written in *xml* – format. There is created the computer editor of algebra formulae, thus saving the computer memory. Elements of editor of AA formulae and the graphical interface there are presented in Chapter 2. Also *xml* - format of AA operations is described. Algorithm formulae that were created in such format can be saved in computer memory. In the Chapter 3 constructions of two analysis algorithms of *xml* - format are described. There are formed and proved theorems that synthesized algorithms describe the AA formulae analysis in *xml*-format. The *xml* – format contains operation types, operation orientations, operation uniterm separation, uniterms choosing, as well as the saving and editing this information. The identity of the two created algorithms is proved.

2. Elements of editor of algorithm formulae

The main window of algorithm algebra formula editor in Fig. 1 is shown. The editor simplifies the creation of AA formulae.

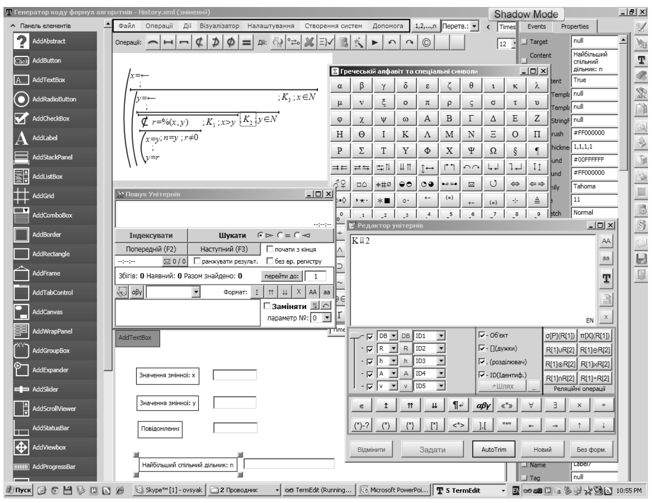

Fig. 1. The main menu of AA formula editor
Rys. 1. Okno główne edytora formuł algorytmów

Algorithm formulae that are synthesized by the editor are saved in computer memory as files with *xml* – format. For example, uniterm sequencing *F*(*x*,*y*) i *S*(*z*), of horizontal operation orientation, has the form $\overparen{F(x,\ y);\ S(z)}$. For uniterm separation the semicolon is used. In the computer memory this is saved in *xml*-format as:

```
<s sep="sem" ori="hor">
    <u>F(x, y)</u>
    <u>S(z)</u>
</s>
```

where in the first line is the identifier of operation sequencing (*s*), identifier of uniterm separation (*sep*=), separator itself (“*sem*”), orientation identifier (*ori*=) and its meaning (*hor*). In next lines the uniterm (*u*) indentifier, denoting uniterms (***F*(*x, y*)** or ***S*(*z*)**) is written. Sequencing operation of vertical orientation, with coma separator, has the form

$$\left(\begin{array}{l} \boldsymbol{F(x, y),} \\ \boldsymbol{S(z)} \end{array}\right.$$

The *xml*-format of the operation, has the form:

```
<s sep="com" ori="ver">
   <u>F(x, y)</u>
   <u>S(z)</u>
</s>
```

xml – formats of elimination (*e*) and paralleling (*p*) operations have similar descriptions. Note, the elimination operation has three uniterms, separated by semicolon.

Cycle operations, descripting the cyclic sequencing (*cs*) of horizontal (*hor*) and vertical (*ver*) orientation, containing cycle condition (*g-?*) and uniterm *H*, have the following forms:

```
<cs ori="hor">                and        <cs ori=”ver”>
    <u>g-?</u>                               <u>g-?</u>
    <u>H</u>                                 <u>H</u>
 </cs>                                   <cs>
```

Operations of cyclic elimination (*ce*) and paralleling (*cp*) have similar form.

3. Analysis models of *xml* - format

Theorem 1. If *F* is *xml* - format of algorithm formula, then algorithm formula (1) identifies operation type, orientation, uniterm separation and uniterms, and if there is an lack of any element then the error message is generated.

Proof. Elimination condition *(n≠$)-?* is checked. If variable *n* isn't empty then algorithm is described by *xml* – format. If variable is empty then an error message is formed. We get uniterm *Exc(“$_xml”)* with information (*$_xml*). To get this information automatically, from algorithm formulae *xml* - files the formula (1) is realized, where:

pu st(w∈@T)Txml(t∈@T,n∈@Nod) – algorithm formula (1) title;
. – the algorithm fulfilment end;
(n.Na.Eq(“u”)-? – checking if variable *n* includes *u*;
(n.Na.Eq(“s”))-? – checking if variable *n* includes *s*;
s∈@S – subsystem of working out of sequencing operation;
s.par=t – ascribing to variable *par* the value of variable *t*;
s.sep=@S.Sep.Sem – ascribing to variable *sep* the separator *Sem*;
pu – identifier of access;
st – identifier of statics;
(w∈@T) – ascribing *w* to subsystem of type *T*;
Txml – algorithm formula name;
n∈@Nod) – ascribing *n* to subsystem *Nod* (shortened name of standard subsystem *XmlNode* [3, 4]);
u∈@U – ascribing *u* to subsystem *U*;
u.par=t – ascribing *t* to *u.par*;
u.val=n.IT – ascribing *n.IT* to variable *val;*
w=u. – ascribing *u* to *w* and the end (.) of algorithm fulfillment;
(n.IT.Le>0)-? – calculation of sign number of variable (*n.IT.Le*), and checking if the number is greater than 0;

```
pu st (w∈@T)Txml(t∈@T, n∈@Nod)=

(u∈@U;
 ;
 (u.par=t
  ;
  [u.val=n.IT
   ;
   *
   ;
   (n.IT.Le>0)-?
  ;
  w=u.
                                                          (1)
(s∈@S; A; (n.Na.Eq(“s”))-?; (n.Na.Eq(“u”)-?; Exc(“$_xml”).; (n≠$)-?
 ;
 (s.par=t
  ;
  [s.sep=@S.Sep.Sem
    ;
    [s.sep=@S.Sep.Com
     ;
     Exc(„Błąd_s xml”).
     ;
    n.Attri[“sep”].Val.Eq(“com”)-?
   ;
  n.Attri[“sep”].Val.Eq(“sem”)-?
  ;
  ([s.ori=@S.Ori.Hor
    ;
    [s.ori=@S.Ori.Ver
     ;
     Exc(„Błąd_s_O xml”)
     ;
    n.Attri[“ori”].Val.Eq(“ver”)-?
    ;
   n.Attri[“ori”].Val.Eq(“hor”)-?
   ;
   (s.tA = Txml(s, n.CN[0])
    ;
    (s.tB = Txml(s, n.CN[1])
     ;
     w= s.
```

(n≠$)-? – checking if *xml* is not empty (*$*);
s.sep=@S.Sep.Com – ascribing *S.sep* to *Com* separator;
Exc(„Błąd_s xml”) – message outlet about an error in sequencing operation description;
n.Attri[“sep”].Val.Eq(“com”)-? – checking if coma (*com*) is a uniterm separator;
n.Attri[“sep”].Val.Eq(“sem”)-? – checking if semicolon (*sem)* is a uniterm separator;
s.ori=@S.Ori.Hor – ascribes horizontal (*Hor*) orientation identification sign to *s.ori*;
s.ori=@emS.Ori.Ver – ascribing vertical (*Ver*) orientation identification sign to *s.ori*;
Exc(„Błąd_s_O xml”) – message outlet about an error in operation orientation description;
n.Attri[“ori”].Val.Eq(“ver”)-? – checking if the orientation is vertical (*ver*);
n.Attri[“ori”].Val.Eq(“hor”)-? – checking if the orientation is horizontal (*hor*);
s.tA = Txml(s,n.CN[0]) – ascribing the first sequencing uniterm to *s.tA*;
s.tB = Txml(s, n.CN[1]) – ascribing the second sequencing uniterm to *s.tB*;
w= s. – ascribing to variable *w* the variable *s* value and the end (.) of algorithm description. Here:

```
A=  e∈@E; B ; (n.Na.Eq(“e”))-?
    ;
    e.par=t
    ;
        e.ori=@E.Ori.Hor
        ;
            e.ori=@E.Ori.Ver
            ;
            Exc(„Błąd_e xml”)
            ;
            n.Attri[“ori”].Val.Eq(“ver”)-?
        ;
        n.Attri[“ori”].Val.Eq(“hor”)-?
    ;
    e.tA = Txml(e, n.CN[0])
    ;
    e.tB = Txml(e, n.CN[1])
    ;
    cond=Txml(e, n.CN[2])
    ;
    w= e.
```

A – elimination formula. If the formula is not sequencing then the elimination operation exists. Elimination operation is identified by *e*;
e∈*@E* – type designation for variable *e*;
B – algorithm for checking if in the *xml* – format the paralleling operation exists. Elimination operation is identified by *p*;
p∈*@P* – type designation for variable *p*;
D – algorithm for checking if in the *xml* – format the cyclic sequencing operation exists. Cyclic sequencing operation is identified by *cs*;
cs∈*@CS* – type designation for variable *cs*;
I – algorithm for checking if in the *xml* – format the cyclic elimination exists. Cyclic eleimination operation is identified by *ce*;
ce∈*@CE* – type designation for variable *ce*;
G – algorithm for checking if in the *xml* – format the cyclic paralleling exists. Cyclic paralleling operation is identified by *cp*;
cp∈*@CP* – type designation for variable *cp*.

```
C=  p∈@P; D; (n.Na.Eq(“p”))-?
    ;
    p.par=t
    ;
        p.sep=@P.Sep.Sem
        ;
            p.sep=@P.Sep.Com
            ;
            Exc(„Błąd_p xml”)
            ;
            n.Attri[“sep”].Val.Eq(“com”)-?
        ;
        n.Attri[“sep”].Val.Eq(“sem”)-?
    ;
        p.ori=@P.Ori.Hor
        ;
            p.ori=@P.Ori.Ver
            ;
            Exc(„Błąd_p_1 xml”)
            ;
            n.Attri[“ori”].Val.Eq(“ver”)-?
        ;
        n.Attri[“ori”].Val.Eq(“hor”)-?
    ;
    p.tA = Txml(p, n.CN[0])
    ;
    p.tB = Txml(p, n.CN[1])
    ;
    w= p.
```

```
,  D=  cs∈@CS; I; (n.Na.Eq(“cs”))-?
       ;
       cs.par=t
       ;
           cs.ori=@CS.Ori.Hor
           ;
               cs.ori=@CS.Ori.Ver
               ;
               Exc(„Błąd_cs xml”)
               ;
               n.Attri[“ori”].Val.Eq(“ver”)-?
           ;
           n.Attri[“ori”].Val.Eq(“hor”)-?
       ;
       cs.tA = Txml(cs, n.CN[0])
       ;
       cs.tB = Txml(cs, n.CN[1])
       ;
       w= cs.
```

```
I=  ce∈@CE; G; (n.Na.Eq(“ce”))-?,
    ;
    ce.par=t
    ;
        ce.ori=@CE.Ori.Hor
        ;
            ce.ori=@CE.Ori.Ver
            ;
            Exc(„Błąd_cs xml”)
            ;
            n.Attri[“ori”].Val.Eq(“com”)-?
        ;
        n.Attri[“ori”].Val.Eq(“sem”)-?
    ;
    ce.tA = Txml(ce, n.CN[0])
    ;
    ce.tB = Txml(ce, n.CN[1])
    ;
    w= ce.
```

```
G=  cp∈@CP; Exc(„Błąd”); (n.Na.Eq(“cp”))-?
    ;
    cp.par=t
    ;
        cp.ori=@S.Ori.Hor
        ;
            cp.ori=@S.Ori.Ver
            ;
            Exc(„Błąd_cp xml”)
            ;
            n.Attri[“ori”].Val.Eq(“com”)-?
        ;
        n.Attri[“ori”].Val.Eq(“sem”)-?
    ;
    cp.tA = Txml(cp, n.CN[0])
    ;
    cp.tB = Txml(cp, n.CN[1])
    ;
    w= cp.
```

Theorem 2. If *F* is algorithm formula described by *xml*-format, then algorithm formula (2) describes the identification of AA operations, their orientation, uniterm separation, uniterm choosing and detection of errors in *xml* – format of the algorithm.

```
pu st (w∈@T)Txml(t∈@T, n∈@Nod)=
    u∈@U;
    ;
    u.par=t
    ;
        u.val=n.IT
        ;
        *
        ;
        (n.IT.Le>0)-?
    ;
    w=u.
                                                   ; n.Na.Eq(„u”)-?;Exc(“$_xml”).;(n≠$)-?
    ⊅i
        i∈@bi; Exc(„Błąd”). ; (n.Na.Eq(“i”))-?
        ;
        i.par=t
        ;
            *; j=1; (i∈Q3)-? ;
            ⊄j
                ⊅(k∈Q2)
                i.xj=@bi.yj.ak,j;ck; Exc(“Błąd_i”).; k-? ; (n.Attri[“xj”].Val.Eq(“ak,j”)-?
                ;
                cj
            ;
            ⊄z
                i.xz= Txml(i, n.CN[z]); *; ((i=e)|(z≠2)) - ?
                ;
                cz
            ;
            w=i.                                                                    (2)
```

where

$i \in Q_0 = \overparen{s; e; p; cs; ce; cp};$ $b_i \in Q_1 = \overparen{S; E; P; CS; CE; CP};$ $j, k \in Q_2 = \overparen{0; 1};$

$Q_3 = \overparen{s; p};$ $x_j \in Q_4 = \overparen{sep; ori};$ $y_j \in Q_5 = \overparen{Sep; Ori};$

$$a_{k,j} \in Q_6 = \begin{pmatrix} \overparen{Hor; Ver} \\ ; \\ \overparen{Sem; Com} \end{pmatrix}, \quad t_i = \begin{cases} tA, \text{ if } i \in Q_7 = \overparen{s, e, p}, \\ cond, \text{ if } i \in Q_8 = \overparen{cs, ce, cp}; \end{cases} \quad r_i = \begin{cases} tB, \text{ if } i \in Q_7, \\ t, \text{ if } i \in Q_8, \end{cases}$$

$$x_z = \begin{cases} t_i, \text{ if } z=0; \\ r_i, \text{ if } z=1; \\ cond, \text{ if } z=2, \end{cases} \quad z \in Q_9 = \overparen{0;1;2},$$

Proof. The proof is analogous to the one given for theorem 1. If variable *n* isn't empty then algorithm is described by *xml* – format. If variable is empty then an error message is formed. We get uniterm *Exc(“$_xml”)* with information (*$_xml*).

Let variable *i* has the meaning *s*. Then $b_i \in S$. Uniterm *i*∈*@b* or *Exc(„Błąd”).* is eliminated according to condition *(n.Na.Eq(“i”))-?*. Uniterms *s*∈*@S*, *Exc(„Błąd”).* and *(n.Na.Eq(“s”))-?* are similar as in formula (1).

Uniterm *i.par=t* after changing *i* to *s* has the form (*s.par=t*) and is the same as in formula (1). Elimination according to the condition *(i*∈Q_3*)-?* gives empty uniterm (*), that can be omitted as it doesn't changes the formula.

Variables *j* and *k* obtain the value 0. Variables x_0 , y_0, $a_{0,0}$ obtain meaning *sep, Sep, Sem*, that gives elimination according to the condition $(n.\underline{Attri}[\text{“}x_j\text{”}].\ \underline{Val}.\underline{Eq}(\text{“}a_{k,j}\text{”})\text{-?}$. The formula (3) is obtained:

$$\left[\begin{array}{l} s.sep=@S.Sep.Sem \\ ; \\ \left[\begin{array}{l} c_k \\ ; \\ \underline{Exc}(\text{“Błąd\_s xml”}). \\ ; \\ ((0+1)\in Q_2)\text{-?} \end{array}\right. \\ ; \\ (n.\underline{Attri}[\text{“sep”}].\underline{Val}.\underline{Eq}(\text{“Sem”})\text{-?} \end{array}\right. \tag{3}$$

Condition $(0\in Q_2)\text{-?}$ of cyclic elimination is fulfilled, and this is why the value of cycle variable increases by 1 and turns to cycle according to the variable *k*. We get expression for uniterm variables x_0 , y_0, $a_{1,0}$ of sequencing operations *sep, Sep, Com*. Substituting them in formula (3) we get the expression:

$$\left[\begin{array}{l} s.sep=@S.Sep.Sem \\ ; \\ \left[\begin{array}{l} s.sep=@S.Sep.Sem \\ ; \\ \left[\begin{array}{l} c_k \\ ; \\ \underline{Exc}(\text{“Błąd\_s”}). \\ ; \\ ((1+1)\in Q_2)\text{-?} \end{array}\right. \\ ; \\ (n.\underline{Attri}[\text{“sep”}].\underline{Val}.\underline{Eq}(\text{“Com”})\text{-?} \end{array}\right. \\ ; \\ (n.\underline{Attri}[\text{“sep”}].\underline{Val}.\underline{Eq}(\text{“Sem”})\text{-?} \end{array}\right.$$

In the last expression the condition $((1+1)\in Q_2)\text{-?}$ is not fulfilled, that is why elimination according to the condition can be changed into uniterm $\underline{Exc}(\text{“Błąd\_s”})$. The formula is obained:

$$\left[\begin{array}{l} s.sep=@S.Sep.Sem \\ ; \\ \left[\begin{array}{l} s.sep=@S.Sep.Com \\ ; \\ \underline{Exc}(\text{“Błąd\_s”}). \\ ; \\ (n.\underline{Attri}[\text{“sep”}].\underline{Val}.\underline{Eq}(\text{“Com”})\text{-?} \end{array}\right. \\ ; \\ (n.\underline{Attri}[\text{“sep”}].\underline{Val}.\underline{Eq}(\text{“Sem”})\text{-?} \end{array}\right. \tag{4}$$

Now the variable of cyclic sequencing *j* increases by 1, and operation variable of cyclic elimination *k* obtains first value 0. Then for variables x_1 , y_1, $a_{0,1}$ we get next meaning *ori, Ori, Hor,* and for $a_{1,1}$ we have *Ver.* Analogous as we got the expression (4), we create the formula:

$$\left[\begin{array}{l} s.ori=@S.Ori.Hor \\ ; \\ \left[\begin{array}{l} s.ori=@S.Ori.Ver \\ ; \\ \underline{Exc}(\text{“Błąd\_s”}). \\ ; \\ (n.\underline{Attri}[\text{“ori”}].\underline{Val}.\underline{Eq}(\text{“Ver”})\text{-?} \end{array}\right. \\ ; \\ (n.\underline{Attri}[\text{“ori”}].\underline{Val}.\underline{Eq}(\text{“Hor”})\text{-?} \end{array}\right.$$

Now variable *z* of cycle obtains the first value 0, and $x_z= t_i= tA$. Then condition $((s=e)|(0\neq 2))\ \text{-}\ ?$ of elimination is fulfilled. Elimination gives the uniterm $s.tA= Txml(i,\ n.\underline{CN[0]})$ and next the return to cycle according to the variable *z*, that obtains value 1, is realized. In the second iteration $x_z= r_i= tB$, and the condition of elimination $((s=e)|(1\neq 2))\ \text{-}\ ?$ is fulfilled, that gives uniterm $s.tB= Txml(s,\ n.\underline{CN[1]})$. In the third iteration $z=2$. Condition $((s=e)|(2\neq 2))\ \text{-}\ ?$ of elimination is not fulfilled. That's why we get from it empty uniterm that can be omitted.

From the last line of formula (2) we get $w=s$. So in this way we get the same formula as fragment of formula (1).

$$\left(\begin{array}{l} s.tA = Txml(s,\ n.\underline{CN[0]}) \\ ; \\ \left(\begin{array}{l} s.tB = Txml(s,\ n.\underline{CN[1]}) \\ ; \\ w= s. \end{array}\right. \end{array}\right.$$

That's proofs that formula (2) in *xml*-format identifies the sequencing operation parameters. The theorem is proved. Proofs for elimination, paralleling and cycle description operations can be made analogously.

4. Comparison of algorithm formulas

As it was proved formulas (1) and (2) describe the same analysis process of *xml*-description of algorithm formulas.

Let's compare algorithm formulas (1) and (2) taking into account the number of algorithm. Formula (1) contains 90 uniterms, while the formula (2) only 26 uniterms. Thus formula (2) contains 3.5 times less uniterms.

5. Conclusion

Algorithm description in the form of AA formulae makes it possible the fulfilment by algorithm transformed the identical results as original ones. This reduces costs that are necessary for algorithm realization.

6. References

[1] Owsiak W., Owsiak A.: Rozszerzenie algebry algorytmów. Pomiary Automatyka Kontrola. No 2, 2010, s. 184 – 188.

[2] Ovsyak A.V., Ovsyak V.K.: Modificirowana algebra algoritmov i instrumentalny sredstva obrabotki formul algebry algoritmiv. Upravlajuszczije systemy i maszyny. Nr 1, 2013. s. 27 – 36.

[3] Petzold C.: Programming Microsoft Windows with C#. 2002.

[4] MacDonald M.: Pro WPF in C#. 2008 Windows Presentation Foundation with .NET 3

otrzymano / received: 21.02.2014
przyjęto do druku / accepted: 01.04.2014 *artykuł recenzowany / revised paper*