Bibliography

[Markuszewski, 2011] T. Markuszewski. The model of the system for algorithm formula transformations, In: Measurement Automation and Monitoring, no 02/2011, pp. 201 – 204

[Owsiak, 2005] W.Owsiak, A.Owsiak, J.Owsiak. Teoria algorytmów abstrakcyjnych i modelowanie matematyczne systemów informacyjnych. – Opole: Politechnika Opolska, 2005. – 275 s.

Authors' Information

Magdalena Niziołek *– Opole University of Technology, Faculty of Electrical, Control and Computer Engineering, ul. Sosnkowskiego 5, 45-271 Opole, Poland; e-mail: m.niziolek@doktorant.po.opole.pl*

Major Fields of Scientific Research: programming, theory of algorithms,

Volodymyr Ovsyak *– Opole University of Technology, Opole, Poland and Ukrainian University of Printing, L'vov , Ukraine ; ovsyak@rambler.ru*

Major Fields of Scientific Research:Theoretical and applied computer science, theory of algorithms, programming, information systems, mathematical modeling.

THE COMPUTER PROCESS OF OPTIMIZATION ALGORITHMS FORMULAS

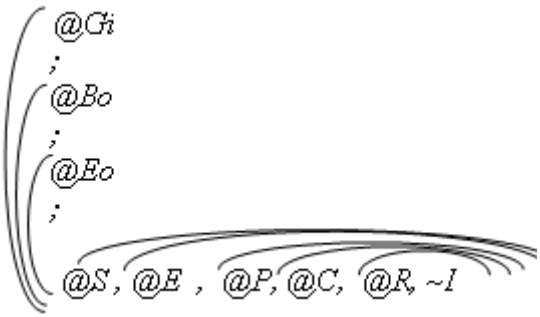
Tomasz Markuszewski

Abstract: *This paper presents a model of a computerized system to optimize formulas of algorithms. For this purpose the algebra of algorithms. The computer process optimization formula of algorithm is very complex, and were decomposed to simplify. The new model described in form of formulas algorithms, and contains variables, function uniterms. The variables are used for storing interim and final data used in the optimization process. Functional unierms are initiating variables, checking the possibility of optimization by introducing an additional condition, optimizing single operation algorithm algebra, and for all algorithms operations algebra with introducing an additional condition method. The built effect of the model is its simplicity. The primary benefit of implementation a model is to protect transformation formulas of algorithms.*

Keywords*: algebra, formula, algorithm, transformation, optimization, model, decomposition.*

Introduction

Algebra algorithms can write algorithms in the form of mathematical expressions called formulas [Ovsyak, 2005]. These formulas it's possible to transformed into a less complex (optimal) form by using the operations properties of the algebra algorithms. Appling algebra [Ovsyak, 2005, 2008] allows for the creation of a computerized system to optimize formulas algorithms [Markuszewski, 2010, Ovsyak, 2011]. The model of computerized optimization algorithms formulas is presented in the expresions algorithm algebra, and contains submodels: @*Gi* – generating indexes xml – description of the formula algorithm [Petzold, 2002], @*Bo* – supporting the optimization submodels: @*S* – sequencing operation; @*E* – elimination operation; @*P* – paralleling operation; @C – cyclic sequencing, cyclic elimination and cyclic paralleling operations; @*R* – inverting operation; @*Eo* – means the optimization subsystem by introducing an additional condition; ~*I* – subsystem that provides the necessary data. The order calls the subsystems of computer system optimization formulas algebra algorithms is presented in expression (1), and contained in @*Lo* subsystem



(1)

Many processes to optimize the operation of formulas according to the properties of algebra of algorithms, and optimization by introducing an additional condition, causing great difficulty in the induction of optimization. For this reason the model is very complex and complicated, it is necessary to simplify the decomposition.

Intuitive Explanation Of Operations

The created a model shown in expression (2) contains variables used to store data, *x* – processed xml – description of the formula algorithm, xi – variable of subsystem generating indexes (index or trackway is a string indicating the xml fragments description of the formula, a record (*xi @Gi*) – the creating variable *xi Gi* subsystem, ac – optimizes formulas by introducing an additional condition*, i1, i2* – provides access to data necessary for process optimization (the mark (~) indicates the sharing subsystem i1,i2), o – an array with variables subsystems optimization and created functional uniterms making the process of optimization, *loop* – contains information on whether the repeated optimization, *isAddO* – stores information on whether it is possible to optimize by the introduction of an additional condition methods, and function uniterms: *InitOp*() – setting the initial values of variables, *ChkAddC*() – checking the possibility of introducing an additional condition, *OneOp*() – making the optimization for single operation algorithm algebra and *Optima*() – performing a full optimization for all operations with optimization formulas by introducing an additional condition.

$$
@Lo = \left(\begin{array}{l}
x \in @\underline{xmld}\ ,\ xi \in @Gi \\
, \\
i1 \in \sim IOp1\ ,\ i2 \in \sim IOp2 \\
, \\
isO \in @\underline{bool}\ ,\ ac \in @Eo \\
, \\
o \in @\underline{object[\,]}\ ,\ loop \in @\underline{bool} \\
, \\
isAddO \in @\underline{bool} \\
, \\
InitOp()\ ,\ ChkAddC() \\
, \\
OneOp()\ ,\ Optima()
\end{array}\right) \qquad (2)
$$

where: *@xmld* – name of standard subsystem *XmlDocument* [Petzold, 2002, MacDonald, 2008], *@bool* – abbreviated name of standard subsystem bool .

The functional uniterm initialing variables

Function uniterm expressed in formula (3) contains the input parameters for entering, *ss* – the entire text *xml* – description of the algorithm formula (*@str* – means standard subsystem string [Ovsyak, 2008, MacDonald, 2008]) and *mode* – how to enter an additional condition (if the value is *true* then automatically otherwise new condition are set by user).

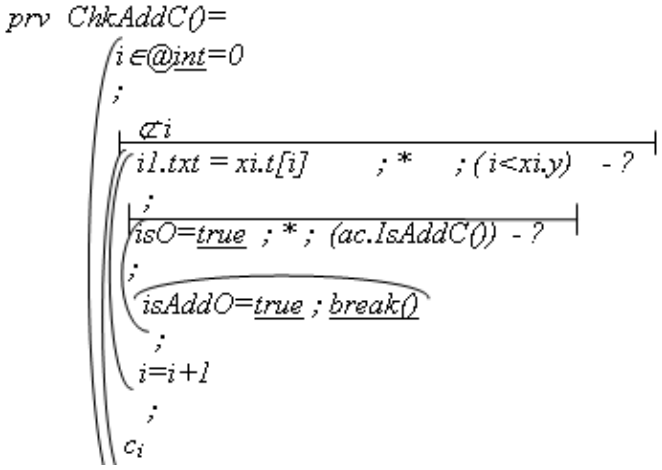
$$
prv\ \ InitOp(ss \in @\underline{str},\ mode \in @\underline{bool}) = \left(\begin{array}{l}
xi \in @Gi()\ ;\ o \in @\underline{object[5]} \\
; \\
o[0] \in @S()\ ,\ o[1] \in @E() \\
, \\
o[2] \in @P()\ ,\ o[3] \in @C() \\
, \\
o[4] \in @R()\ ,\ ac \in @Eo() \\
; \\
xi.Lox(x)\ ;\ xi.Cr() \\
; \\
ac.SetArrayU(ss)\ ,\ ac.auto=mode \\
, \\
isO=\underline{false}\ ,\ isAddO=\underline{false} \\
, \\
loop=\underline{false}\ ,\ i1=ac
\end{array}\right), \qquad (3)
$$

where *prv InitOp(ss∈ @str, mode∈ @bool)* – header of formula algorithm, *prv* – identifier of access private, *o* – an array (the standard subsystem *@object*) for storing the optimization subsystems: *o[0]∈@S()* – the creating of subsystem *@S* and put its into an array of objects – uniterm *o[0]* – means the first element in *o*, *o[1] ∈@E()* – the creating subsystem *@E* and put its into second position in array *o*, other systems are treated analogously, *ac* – subsystem optimization formulas by introducing an additional condition, *ac.auto=mode* – attributing to variable

auto value of input *mode* (automatically when have value *true* or user mode), *xi.Lox()* – load *xml* data, *xi.Cr()* – creating indexes for *xml* data, *ac.SetArrayU(ss)* – setting array available to optimize the conditions by introducing an additional condition (*ss* – input parameter that contains only text entire formula of algorithm), and the variable *loop* – used by the cyclic eliminations, to repeat the optimization until no further transformation is possible.

The functional uniterm checking the possibility of introducing condition

Function uniterm *ChkAddC* () represented by the formula (4)



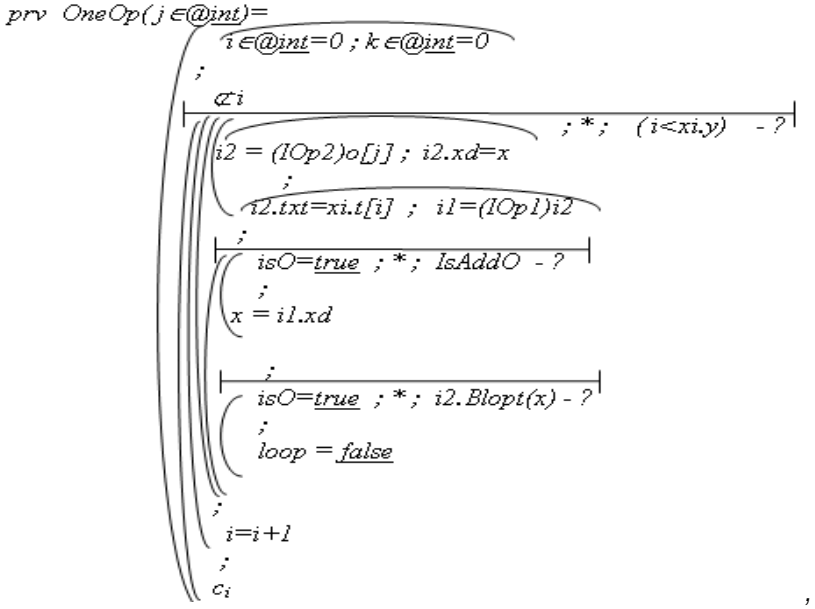
(4)

,

where *i* – the number of processed index, *isO* – store information on whether the optimization has been *xi.t[i]* – an array contains indexes *xml* – description of the formula, *xi.y* – the number of maximum index, *IsAddC()* – testing whether for a given *xi.t[i]* the index is loaded by providing uniterm *i1.txt* can be performed optimization by introduction an additional condition, *break()* – causes the exit from the cyclic elimination.

The functional uniterm for optimizing one operation

Function uniterm *OneOp()* expressed in the formula (5) performs the optimization of a single operation algorithms algebra.



(5)

,

where *j* – the input parameter (@*int* – standard subsystem int for storing integers) indicate the position in the array of objects *o[]* which optimization subsystem can be optimizing, *i,k* – represents the integers numbers, uniterms *i1, i2* – delivery indexes and the *xml* – description of the formulas algorithm to subsystems , *i2* = *(IOp2)o[j]* – loading the data from the uploaded uniterm *i2* subjects and this being situated in an array of objects *o[]*, *i2.xd* = *x* – means load *xml* – description of the algorithm formula variable *x* to *i2*, *x* = *i1.xd* – ask the value of processed *xml*– description of the algorithm, *i1* = *(IOp1)i2* – means that the uniterm *i2* giving to *i1* values (this is made possible by the inheritance mechanism), *i2.Blopt()* – functional uniterm that contains a call to all optimization properties single operation algebra algorithm, *xi.y* – the number of maximum index.

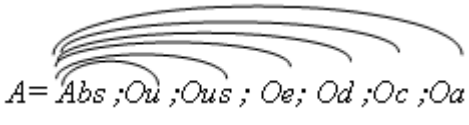
Function uniterm *i2.Blopt()* is predefined in optimization subsystems @*S,*@*E,*@*P,*@*C*, and @*R*, and uses variables *isO* – storing the information about whether there have been optimizations, *de* – an array used to store the order of operation an algebra inducing properties of algorithms and functional uniterm, *A* – contains sequences repeatedly nested sequence of function uniterms names optimization (an example illustrated by the formula (7)), and has function uniterm *de[i]()* – caller and the optimization of the properties of the operations of algebra algorithms, *ChkOp ()* – checking whether a formula optimization algorithms, *de.Co()* – returns the maximum and number of functional uniterms placed in an array *de* (elements of an array are numbered from zero).

Function uniterm ~*Op2.Blopt()* presented in formula (6) gives a value of variable *w*, which takes *true* value if made at least one optimization, or otherwise returns *false,* and the input parameter is *d*, witch update *x* – processed *xml* – description of the formula algorithm, Uniterm *pu* means the identifier of access public, and the end (.) of functional uniterm, * – empty uniterm, c_i, – means returns to cycle,

```
pu (w∈@bool) ~Op2.Blopt(d ∈@xmld) =
  (
    (
      x = d ; isO ∈@bool=false ; i ∈@int=0
      ;
      de[] ∈^Deleg()
      ;
      de=A
    ;
    ⊄ i
    |
      de[i]()                    ; w= isO. ; i<de.Co()-?
      ;
      |
        isO=true ; * ; ChkOp() - ?
      |
      ;
      i=i+1
      ;
      ci
```

(6)

Note that, uniterm ^*Deleg* means delegation uniterm (defined as: delegate [Petzold, 2002, MacDonald, 2008]), and uniterm *A* for example of eliminating operation show in formula (7)



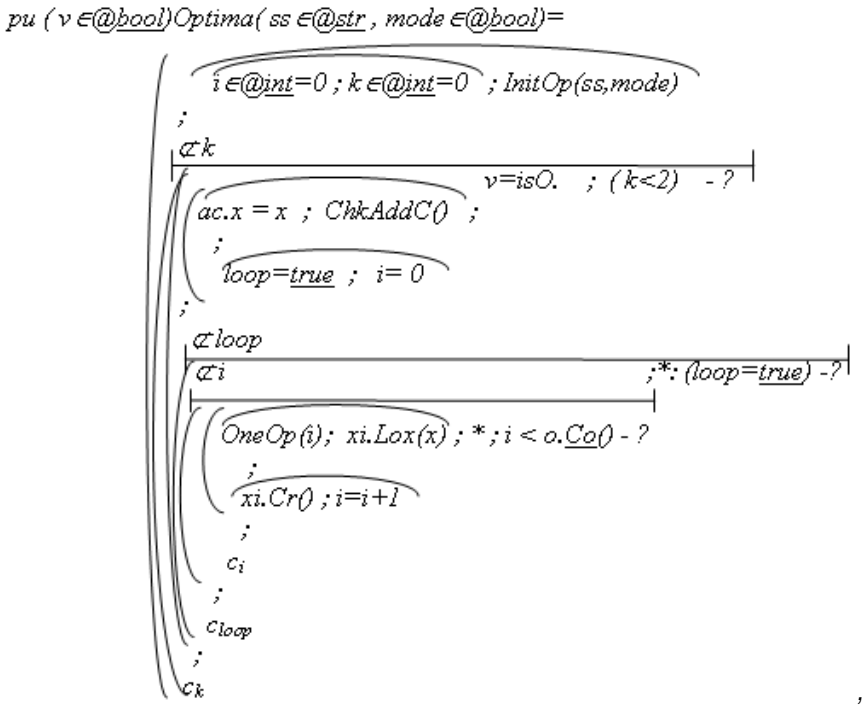
(7)

where *Abs* – name of function uniterm (defined in @*Bo* (supporting subsystem)) performing the optimization by property idempotency [Ovsyak, 2005], *Ou, Ous* – names of function uniterm absorption of uniterms, *Oe,Od,Oc,Oa* – means names of function uniterms, witch performing the optimization by distributiveness proprieties for operation of elimination algorithm algebra.

The names of optimization function uniterms, signed *Oxx* – where *xx* is the abbreviation of the property, which they use when they optimize (eg *Ous* where xx = us – pointing to the uniterms of absorption).

The functional uniterm for full optimizing

Function uniterm *Optima()* presented in formula (8) making the full optimization formulas algorithm



(8)

where *pu* (*v*□*bool)Optima(ss*□*@str, mode*□*@bool)* – header of formula algorithm, *pu* – – identifier of access public, *v* – output parameter, return information whether at least one optimization of the formula was made, *ss* – input parameter string describing the whole xml – formula of algorithm, *mode* – the mode of introduction of an additional condition (entered the user or automatically), *Lox*() – function uniterm is used to load the *xml* – the formula describing the algorithm, *Cr()* – function uniterm creating all indexes and placing to *xi.t* tables [Markuszewski, 2010], *o.Co()* – function uniterm that returns the maximum number object, (.) – the end of functional uniterm, * – empty uniterm, c_i, c_{loop}, c_k – means returns to cycles.

The variable *loop* "works" as long as it is not possible to further convert the formulas to the form of optimal algorithms. In order to ensure the iteration, which must be done twice, since the finding that it is impossible to

further optimization was introduced as a counter variable *k* in the elimination of cyclic. Note that, the function uniterm *OneOp()* change a state *loop* value of variable.

Conclusions

Made the decomposition @*Lo* subsystem simplified model, reduces complexity. Creating the computer system for optimization of formulas of abstract algorithms automates and save the optimization processes. Application uniterms array of objects and the elimination of cyclical loop ensure that the operations of algebra algorithms, placed in arrays will perform in accordance with the order placed in the array. An important extension of the elimination operation has been introduced in the paper, namely a multiconditional elimination has been offered. The extension contributes to the reduction of a number of elimination operations and it simplifies the algorithm minimization process, while improving the readability of algorithms.

An example illustrates the potential of the proposed theory and the underlying methodology for processing of the algorithms. Some other application examples are presented/reported in a complementary paper [Ovsyak, 2011].

Bibliography

[Ovsyak, 2005] V.K. Ovsyak. Theory of Abstract Algorithms and Mathematical Modelling of Information Systems (in Polish), Opole University of Technology Press, Opole, Poland, 2005.

[Ovsyak,2008] V.Ovsyak. Computation Models and Algebra of Algorithms. http://www.nbuv.gov.ua/Portal/natural/VNULP/ISM/2008_621/01.pdf

[Ovsyak et. all, 2011] A.V Ovsyak. Models of the process of an analysis of XML-formatted formulae of algorithms. Submitted to INFOS 2011, Rzeszów , Poland.

[Markuszewski, 2010] T. Markuszewski. A Computer System For Optimizing of Abstract Algorithms, WOFEX 2010, 8th–workshop Ostrawa 2010, pp. 352–357

[Owsiak et. all, 2010] W. Owsiak Synthesis Model Subsystem Serach to Access to Trackways Uniterms xml–Formulas of Algorithm, CSIT'2010, Lwów 2010. pp. 153

[Petzold, 2002] C. Petzold. Programming Microsoft Windows with C#. 2002.

[MacDonald, 2008] M. MacDonald. Pro WPF in C# 2008 Windows Presentetion Foundation with .NET 3.5.

Authors' Information

Tomasz Markuszewski – *Department of Electrical, Control and Computer Engineering, University of Technology, Box: 31, Sosnkowskiego, Opole 45-272, Poland, e-mail: kemotmark@wp.pl*

He specializes in theoretical and applied computer science, theory of algorithms, programming, information systems, mathematical modeling