**Tomasz MARKUSZEWSKI**
POLITECHNIKA OPOLSKA, WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI I INFORMATYKI,
ul. Gen. Sosnkowskiego 31, 45-272 Opole

# Model systemu do przekształceń formuł algorytmów

**Mgr inż. Tomasz MARKUSZEWSKI**

Studia ukończył na Wydziale Edukacji Techniczno Informatycznej Politechniki Opolskiej w roku 2005. Od 2007 roku jest słuchaczem na studiach doktoranckich w dyscyplinie Automatyka i Robotyka na Politechnice Opolskiej. Specjalizuje się w informatyce teoretycznej i stosowanej, teorii algorytmów, programowaniu, systemach i modelowaniu matematycznym.

*e-mail: kemotmark@wp.pl*

**Streszczenie**

W artykule przedstawiono budowę złożonego komputerowego systemu do przekształceń formuł algorytmów przy wykorzystaniu właściwości operacji algebry algorytmów. Budowany model systemu nazwano komputerowym systemem optymalizacji (KSO) i dokonano jego dekompozycji na dwóch poziomach, poziomie systemu i podsystemu. Na poziomie systemu w logiczny sposób wyodrębniono podsystemy pełniące oraz wspomagające zadania optymalizacji formuł algorytmów, a na poziomie podsystemu utworzono unitermy zmienne i funkcyjne. Dla opisania modelu systemu wykorzystano algebrę algorytmów. Fragment modelu systemu zaimplementowano w języku C#.

**Słowa kluczowe**: system, dekompozycja, podsystem, formuła, algorytm, model.

## The model of the system for algorithm formula transformations

**Abstract**

The model of algorithm algebra formulae optimization system is presented in the paper. The decomposition of the system into subsystems is performed using algebra algorithm expressions, and is described in terms of the expressions. The following algorithm algebra operation optimization subsystem models are formed: sequencing, elimination, parallelization, reversing, cycle operations, basic subsystem, subsystem introducing additional condition, subsystem rendering data accessible, and the one generating indices for *xml* code of algorithm formula. The basic subsystem model and the rendering data accessible one in the 3rd section there are described. Algorithm algebra expressions there are used consistently. Models are composed of variables and functional uniterms, which are also used by other subsystems. Variables are used for the data storage. Functional uniterms of the basic subsystem there are the following: extracting, substituting the *xml* code fragment, checking the separator kind in the uniterm, as well as the one checking if the uniterm is a number. Additionally there are included functional uniterms absorbing other uniterms, and the one informing on optimization process run. The basic and rendering data models are implemented in C#. In conclusions some advantages of both the algorithm formula optimization computer system and model decomposition into subsystem are given. Bibliography contains a set of intuitive and formal works describing the algorithms algebra.

**Keywords**: system, decomposition, subsystem, formula, algorithm, model.

## 1. Wprowadzenie

Metody opisywania algorytmów takie jak maszyna Posta [1], Turinga [2], Aho-Ullmana-Hopkrofta [3], Schönhage [4], Kolmogorowa [5] oraz funkcje rekursywne [6], algorytmy Markowa [7], nie dają możliwości formalnego przekształcenia algorytmów do postaci prostszych. Przekształcania algorytmów stają się możliwe przy wykorzystaniu algebry algorytmów [8, 9, 10]. Algebra algorytmów [8, 9] zawiera operacje sekwencjonowania, eliminowania, zrównoleglenia, rewersowania oraz operacje cykliczne, które wykonuje się na unitermach [8, 9]. Algorytmy przedstawiane są w postaci wyrażeń matematycznych zwanych formułami. Formuły algorytmów przy wykorzystaniu właściwości operacji algebry algorytmów mogą być przekształcane do prostszych postaci. Takie procesy przekształceń nazwano optymalizacją algorytmów. Zadanie optymalizacji może być wykonywane przez komputerowy system optymalizacji (KSO).

Stworzenie modelu KSO w całości jest bardzo skomplikowanym procesem. Przeprowadzenie dekompozycji modelu systemu na podsystemy znacznie ułatwia procesy syntezy modelu.

## 2. Dekompozycja systemu KSO

Wykorzystując fakt, że istnieje duża część wspólna dla właściwości operacji algebry algorytmów, która wymaga podobnego potraktowania, wyodrębniono podsystem *@Bo*. W podsystemie *@Bo* umieszczono te unitermy funkcyjne, które są udostępniane do podsystemów optymalizacji operacji eliminowania (*@E*), sekwencjonowania (*@S*), zrównoleglenia (*@P*), operacji cyklicznych (*@C*), rewersowania (*@R*). Do systemu wchodzą także podsystemy optymalizacji:

- podsystem optymalizacji przez wprowadzenie dodatkowego warunku (*@Eo*) [11],
- podsystem (*~I*) udostępniający dane w postaci *xml* (opis formuł algorytmów),
- podsystem wywołujący procesy optymalizacji (*@Lo*),
- podsystem tworzący indeksy (*@Gi*) struktury *xml* (opisów formuł).

Dekompozycja systemu na podsystemy opisana jest formułą (1).

$$KSO = \overbrace{\overbrace{\overbrace{\overbrace{\overbrace{\overbrace{\sim I, @Lo}^{}, \overbrace{@Gi, @Bo}^{}}^{}, \overbrace{@Eo, @S}^{}}^{}, \overbrace{@E, @P}^{}}^{}, \overbrace{@C, @R}^{}}^{} \quad (1)$$

Podsystem *~I* opisany jest formułą (2).

$$\begin{array}{l} pu\ \sim I = \\ \left( \begin{array}{l} pu \sim IOp1 = \\ \quad \left( \begin{array}{l} \{get\} xd \{set\} \in @\underline{xmld} \\ , \\ \{get\} txt \{set\} \in @\underline{str} \end{array} \right) \\ ; \\ pu \sim IOp2: \sim IOp1 = (w \in @\underline{bool}) Blopt(d \in @\underline{xmld}) \end{array} \right) \end{array} \quad (2)$$

Gdzie:

- *pu* – identyfikator powszechnego dostępu do podsystemu,
- *IOp1* – uniterm udostępniający odczyt (*{get}*) i nadający wartości (*{set}*) zmiennej *xd* standardowego typu *XmlDocument* [12], przechowującej opisu formuły algorytmu *xml*,
- *txt* – zmienna typu *string* [12], udostępniająca tekstowy indeks do opisu fragmentu *xml* formuły,
- identyfikator dziedziczenia podsystemów, oznaczony przez (:),
- *Blopt*() – uniterm funkcyjny z danymi wejściowymi (*d*) oraz przekazujący zmienną (*w*) typu *bool*, wykorzystywany do uruchomienia unitermów funkcyjnych optymalizacji konkretnej operacji.

Dekompozycja podsystemu dotyczy odpowiedniego podziału na zmienne oraz unitermy funkcyjne. Korzyścią z zastosowania podziału jest logiczne wyodrębnienie części podsystemu i podział na określone podzadania. Takie postępowanie prowadzi do zmniejszenia złożoności tworzenia modelu. Zmienne służą do przechowywania danych i mogą być typów podanych w tabeli 1. Za pomocą unitermów funkcyjnych wykonuje się zadania na danych przechowywanych w pamięci. Przynależność zmiennych do konkretnego typu oznacza się symbolem $\in$.

Tab. 1. Oznaczenia unitermów odpowiadające standardowym typom w języku C#
Tab. 1. Denotations used for the standard uniterm types in C #

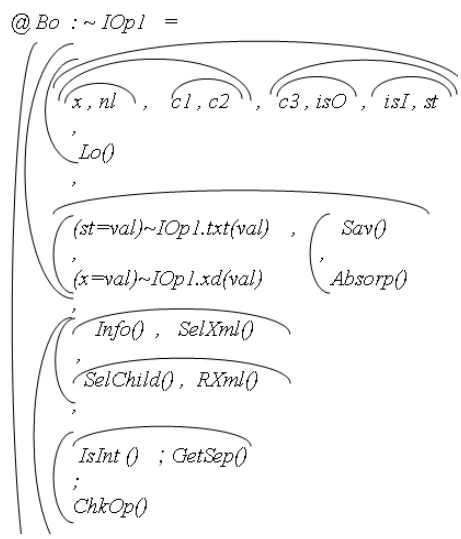
| Lp. | Uniterm | Typ w C# | Reprezentuje |
|---|---|---|---|
| 1 | @*int* | int | liczby całkowite |
| 2 | @*str* | string | teskst |
| 3 | @*bool* | bool | wartość logiczną |
| 4 | @*xmld* | XmlDocument | dokument XML |
| 5 | @*xmln* | XmlNode | pojedynczy węzeł XML |
| 6 | @*xmle* | XmlElement | pojedynczy element XML |
| 7 | @*xmlnl* | XmlNodeList | lista węzłów XML |

## 3. Dekompozycja podsystemu @*Bo*

W podsystemie *@Bo* wprowadza się:
- zmienną *x,* służącą do przechowywania opisu *xml* formuły algorytmu
- zmienną *nl* typu @*xmlnl*, dla przechowania listy wybranych węzłów,
- zmienne *c1,c2* i *c3* – do przechowywania kodu kolejnych unitermów wewnętrznych,
- *isO* – do przechowywania stanu procesu podczas optymalizacji
- *isI* –, do przechowania informacji o tym czy dokonano optymalizacji,
- *st* – do przechowania pojedynczego indeksu węzła,

oraz następujące unitermy funkcyjne:
- *Lo*() – służący do załadowania *xml* (opisu formuły algorytmu),
- *Sav*() – do zapisu formuły zoptymalizowanej,
- *Info*() – informujący o procesie optymalizacji,
- *SelXml*() – wybierający fragment formuły,
- *SelChild*() – ustawiający zmienne *c1,c2,c3*,
- *RXml*() – podmieniający fragment kodu *xml*,
- (*st=val*) – ustawiający zmienną *st*,
- *txt* – ustawiający wskaźnik do fragmentu kodu *xml*,
- *txt*(*val*) – pobierający zawartość zmiennej *st*,
- (*x=val*) – ustawiający zmienną *x*,
- *xd*(*val*) – pobierający z podsystemu wartość *val* dla *x*,
- *Absorp*() – optymalizujący formułę przez pochłonięcie unitermu,
- *IstInt*() – sprawdzający czy podany parametr wejściowy jest liczbą,
- *GetSep*() – zwracający rodzaj separatora, występującego wewnątrz unitermu,
- *ChkOp*() –sprawdzający czy wykonano optymalizację.

Postać dekompozycji podsystemu *@Bo* jest podana wzorem (3).




(3)

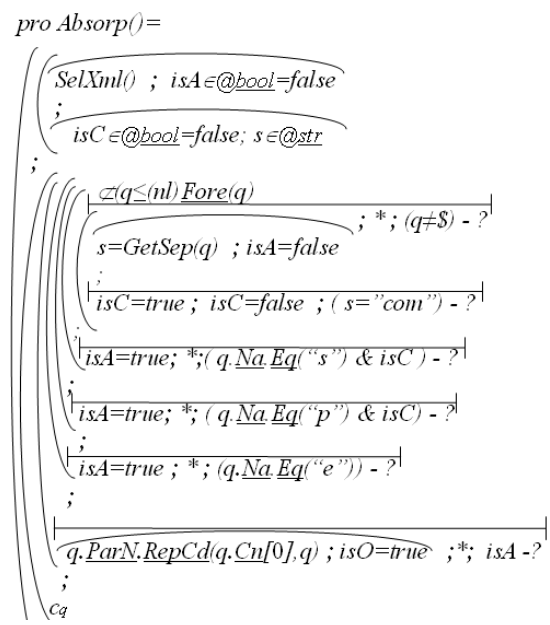
Do nazwy podsystemu *Lo*() dodano przedrostek *pro*, - oznaczający ograniczony dostęp w ramach podsystemu (*protected*) [12]. Do podstawienia pod zmienną *x* kodu *xml*, wykorzystano standardowy uniterm funkcyjny *Load*(*s*) (*s* - nazwa pliku *xml*). Do zapisu *xml* służy funkcyjny uniterm standardowy *Save*(*s*). Do podstawienia *xml* w zmiennej *x* służy uniterm funkcyjny *Lo*() (4), a do zapisania wyniku optymalizacji uniterm funkcyjny *Sav*(), przedstawiony wzorem (5).

$$pro\ Lo(s\in@str) = \overbrace{x \in @xmld() ; x.Load(s)} \tag{4}$$

$$pro\ Sav(s\in@str) = (x.Save(s)) \tag{5}$$

Uniterm funkcyjny *Absorp*() przedstawiony jest wzorem (6), gdzie:
- *q.Na.Eq("s")* – służy do porównania nazwy węzła *q* z nazwą węzła „*sequence*”,
- *q.Na.Eq("e")* – służy do porównania nazwy węzła *q* z nazwą węzła „*elimination*”,
- *q.Na.Eq("p")* - służy do porównania nazwy węzła *q* z nazwą węzła „*parallelisation*”,
- *q.ParN.RepCd())* - dokonuje podmiany węzła *q* na inny,

Wprowadzono zmienne:
- *isA,* przyjmującą wartość *true* jeśli możliwe jest pochłonięcie unitermu,
- *isC* – zawierającą informację czy separatorem wewnątrz operacji jest *comma*,
- (*nl*)*Fore*(*q*) – unitrm opisujący optymalizację operacji cyklicznego eliminowania,
- *$* – wartość zmiennej *null.*

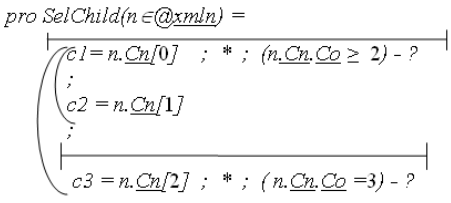



(6)

Uniterm funkcyjny *SelXml*() przedstawiony jest formułą (7)

$$pro\ SelXml() = \begin{pmatrix} isO = false \\ ; \\ nl = x.SelN(st) \end{pmatrix} \tag{7}$$

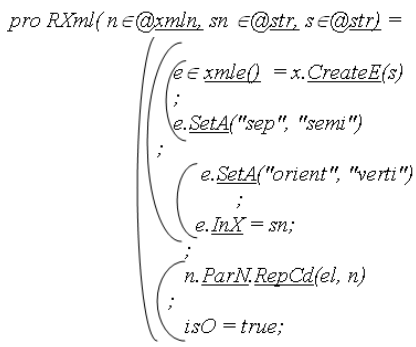
gdzie *SelNl*() opisuje wybór fragment kodu *xml* dla podanego indeksu *st*. Fragment jest identyfikowany przez nazwę i jest nazwany węzłem. Tworzy się tablicę *nl* węzłów.




(8)

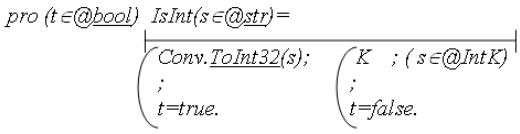
Formuła przedstawiona wzorem (8) opisuje uniterm funkcyjny *SelChild*(), zawierający unitermy *Cn[0], Cn[1], Cn[2]*, służące do kolejnego ustawiania zmiennych *c1, c2, c3* na pierwszy, drugi oraz trzeci podwęzeł węzła *n*, oraz uniterm funkcyjny *Cn.Co* do zwracania liczby podwęzłów węzła *n.*

Uniterm funkcyjny *RXml*(), przedstawiony wzorem (9), opisuje zamiany węzła *n,* podanego jako parametr, na nowy węzeł. Nazwa węzła nadrzędnego (*sn*) jest podawana w parametrze *s* i zawiera uniterm funkcyjny *RepCd*(), wywołany na poziomie węzła rodzica *ParN*. Zmienna *e* służy do przechowywania elementu *xml*. Uniterm funkcyjny *CreateE*() tworzy element *xml*. Uniterm funkcyjny *SetA()* ustawia atrybuty węzła *xml*, takie jak *separator* i *orientantion*.



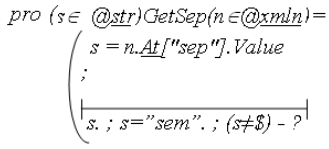

(9)

Uniterm funkcyjny *IsInt*(), opisany wyrażeniem (10), służy do sprawdzenia czy podany w parametrze *s* ciąg znaków jest liczbą. Jeśli tak, to zwracona jest wartość *true.* Zawiera wyjściowy parametr *t* z informacją o rezultatach konwersji typu danych. Uniterm funkcyjny *Conv.ToInt32*(*s*) konwertuje typ danych na typ liczb całkowitych. *K* oznacza komunikat o błędzie w danych.




(10)

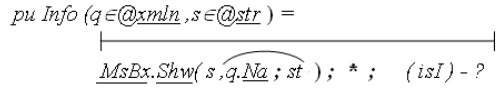
Unitem funkcyjny *GetSep*() służy do zwracania rodzaju separatora. Jest on opisany formułą (11), gdzie *n.At["sep"].Value* jest to uniterm funkcyjny pobierający separator (przecinek lub średnik) oraz zwracający zmienną wyjściową *s*, zwracającą separator.




(11)

Wszystkie zmienne i unitermy funkcyjne znajdujące się w podsystemie *@Bo* są wykorzystywane przez podsystemy optymalizacji *@E, @S, @P, @C, @R* [11].

Uniterm funkcyjny *Info*() (12) przeznaczony jest do wyświetlenia informacji o dokonanej optymalizacji przez wykorzystanie unitermu funkcyjnego *MsBx.Shw*().




(12)

Uniterm funkcyjny *ChkOp*() (13) zwraca informacje o wykonanej albo niewykonanej optymalizacji, gdzie *w* – parametr wyjściowy, któremu przypisuje się wartość zmiennej *isO*, zawierającej dane (*true,false*) o wykonanej optymalizacji.

$$pu\ (w=@bool\ )ChkOp() = (w=isO.) \tag{13}$$

## 4. Implementacja @*Bo* w języku C#

W tej części zamieszczono kod źródłowy programowej realizacji w języku C# podsystemów *~I* oraz *@Bo.*

Programowa realizacja unitermu funkcyjnego *~I*:

```
public interface IOp1
    {
        XmlDocument xd { get; set; }
        string txt { get; set; }
    }

public interface IOp2 : IOp1
    {
        bool Blopt(XmlDocument d);
    }
```

Programowa realizacja podsystemu @Bo:

```
class Bo: IOp1
    {
        public XmlDocument x;
        protected XmlNodeList nl;
        protected XmlNode c1, c2, c3;
        protected bool isO;
        public bool isI;
        protected string st;

         XmlDocument IOp1.xd
        {
          get{return x;}set{x = value;}
        }
        string IOp1.txt
        {
           get{return st;}set{st= value;}
        }
             protected:
void Absorp()
{
bool isA = false, IsC = false;
string s;
SelXml();
foreach (XmlNode q in nl)
  {   SelChild(xo);
      s = GetSep(xo);
      isA = false;
      if  (s.Equals("comma"))  isC  =  true;
      else isC = false;
      if(q.HasChildNodes&&
      q.ChildNodes.Count >= 2)
      if(c1.InnerXml == c2.InnerXml){
            if(q.Name.Equals("sequence")&&
       isC) isA = true;

     if(q.Name.Equals("parallelisation")
       &&isC ) isA = true;
```

```
       if(q.Name.Equals("elimination"))
       isA = true;
       if (isA)
       {
        q.ParentNode.ReplaceChild(c1,q);
         isO = true;
       }
}  }

void SelXml()
        {
          isO = false;
         nl = x.SelectNodes(st)
          }

void SelChild(XmlNode q)
{
  if(q.HasChildNodes&&
             q.ChildNodes.Count >= 2)
  {
    c1 = q.ChildNodes[0];
    c2 = q.ChildNodes[1];
    if (q.ChildNodes.Count == 3)
          c3 = q.ChildNodes[2];
  }
}

void Info(XmlNode q, string s)
{ if (isI)
      MessageBox.Show(s, q.Name
                          + "" + st);
}

void RXml(XmlNode q, string sn,
           string s)
{
       XmlElement e = x.CreateElement(s);
       e.SetAttribute("separator",
                 "semicolon");
       e.SetAttribute("orientation"
                    ,"vertical");
       e.InnerXml = sn;
       q.ParentNode.ReplaceChild(e, q);
       isO = true;
}

bool IsInt(string s)
{
      try {
           Convert.ToInt32(s);
           return true;
            } catch { return false; }
}

string GetSep(XmlNode n)
{
  string s;
  try{
  s = n.Attributes["separator"].Value;
      return s;
            }
catch{
                    s = "semicolon";
                    return s; }
 }

public bool ChkOp(){ return isO;}

public void Lo(string s)
       {
       x = new XmlDocument();
       x.Load(s);
       }

public void Sav(string s)
       {
       x.Save(s);
       }
}
```

## 5. Wnioski

Opisany model komputerowego systemu przekształceń formuł algorytmów opisuje procesy ich optymalizacji.

Przeprowadzona dekompozycja systemu na podsystemy zmniejsza złożoność modelu systemu i podsystemów.

Realizacja modelu umożliwia automatyczne wykonanie optymalizacji formuł algorytmów.

## 6. Literatura


[1] Post E.L.: Finite Combinatory Processes Formulation 1. Journal of Symbolic Logic, 1, pp. 103-105, 1936. Reprinted in The Undecidable, pp. 289ff.

[2] Turing A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. of London Mathematical Society, series 2, vol. 42 (1936-1937), pp. 230-265; correction, ibidem, vol. 43, pp. 544-546. Reprinted in [13 Davis M., pp. 155-222] and available online at http://www.abelard.org/turpap2/tp2-ie.asp

[3] Aho A.V., Hopcroft J.E., Ullman J.D.: The design and analysis of computer algorithms. Addison-Wesley Publishing Company, 1974.

[4] Schönhage A.: Universelle Turing Speicherung. In J. Dörr and G. Hotz, Editors, Automatentheorie und Formale Sprachen, Bibliogr. Institut, 1970, pp. 369-383.

[5] Kolmogorov A.N.: On the concept of algorithm (in Russian). Uspekhi Mat. Nauk 8:4 (1953), pp. 175–176; translated into English in Uspensky V.A., Semenov A.L.: Algorithms: Main Ideas and Applications, Kluwer, 1993.

[6] Church A.: An unsolvable problem of elementary number theory. American Journal of Mathematics, vol. 58 (1936), pp. 345-363.

[7] Markov A.A.: Theory of algorithms (in Russian). Editions of Academy of Sciences of the USSR, vol. 38, 1951, pp. 176-189; translated into English in American Mathematical Society Transactions, 1960, series 2, 15, pp. 1-14.

[8] Owsiak W., Owsiak A., Owsiak J.: Teoria algorytmów abstrakcyjnych i modelowanie matematyczne systemów informacyjnych. Wyd. Pol. Opolskiej, Opole, 2005.

[9] Owsiak W., Owsiak A.,: Rozszerzenie algebry algorytmów. Pomiary, Automatyka, Kontrola. 2-2010, Luty, vol. 56, pp. 184-188.

[10] Piaskowy A: Algebra algorytmów w przykładach. Pomiary, Automatyka, Kontrola. 2-2010 vol. 56 pp. 189-192.

[11] Markuszewski T.: Optymalizacja formuł algorytmów przez wprowadzenie warunku. Pomiary, Automatyka, Kontrola. Nr 02/2011, vol. 57.

[12] Perry S.C.: C# i .NET. Wyd. Helion, Gliwice, 2006.


*otrzymano / received: 19.11.2010*
*przyjęto do druku / accepted: 03.01.2011* *artykuł recenzowany*