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

Optymalizacja formuł algorytmów przez wprowadzenie warunku

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 modelu podsystemu optymalizacji formuł algebry algorytmów metodą wprowadzenia dodatkowego warunku. Podsystem opisany został w postaci formuły algebry algorytmów. Przeprowadzono dekompozycje podsystemu na unitermy funkcyjne i zmienne. Opisano modele unitermów funkcyjnych i przedstawiono fragment implementacji podsystemu w języku C#.

Słowa kluczowe: algorytm, warunek, dekompozycja, uniterm, algebra algorytmów, optymalizacja formuł.

Algorithm formula optimization by the condition use

Abstract

The subsystem model designed for algorithm algebra formula optimization by the introducing the additional condition there is described in the paper. The theory description and the example of an application are shown in the section 2. Next the subsystem model decomposition into variables and functional uniterms, by the introducing the additional condition is realized. Variables are used for the storage of data needed for algorithm formula transformations. The construction of functional algorithms is described using algorithm algebra expressions. Functional algorithms perform the following tasks: initiate variables, set and release elements of the table retaining engaged conditions, and choose first free condition. The subsystem model contains functional uniterms used for the checking of the ability to transformation , as well as perform the optimization itself. The model allows to introduce the additional condition by two ways, automatically or manually. In automatic mode the system itself set variables, while in manual mode the user chooses conditions by the keyboard. The implementation of the subsystem model performing the optimization by the introducing the additional condition is shown in C# program language. In conclusions some advantages of the using of the algorithm formula optimization computer system are pointed out. Sources bounded with algorithm algebra are given in the bibliography.

Keywords: algorithm, formula, condition, decomposition, uniterm, uniterm function, formula optimization.

1. Wprowadzenie

Formuły algorytmów, czyli wyrażenia matematyczne opisujące algorytmy, mogą być przekształcane do prostszych postaci. Przekształcenia formuł algorytmów wykonuje się łatwo przy wykorzystaniu algebry algorytmów [1, 2, 3]. Jedną z metod transformacji formuł algorytmów jest wprowadzenie dodatkowego warunku [1]. Zadanie transformacji formuł do równoważnych prostszych postaci przy zastosowaniu algebry algorytmów, nazwano optymalizacją formuł algorytmów. Optymalizacja formuł algorytmów może zostać zrealizowana przez komputerowy system optymalizacji (KSO). Budowa modelu KSO opisana w pracy [4] zawiera podsystem automatyzacji, którego istota polega na wprowadzaniu dodatkowego warunku (*@Eo*).

W artykule opisano model podsystemu *@Eo*, który powstał w celu realizacji optymalizacji formuł algorytmów metodą wprowadzania dodatkowego warunku [1].

2. Metoda wprowadzenia dodatkowego warunku [1]

Dodatkowe warunki wprowadza się w przypadku optymalizacji takich operacji eliminowania, w których sekwencje różnią się unitermami początkowymi i końcowymi, ale mają jednakowe unitermy wewnętrzne lub jednakowe algorytmy. Na przykład, formuła (1) może zostać przekształcona do postaci przedstawionej wzorem (2), gdzie *F(x), R(y), S(t), P(x,y), K(z)* – oznaczają ogólnie unitermy funkcyjne, *u* – uniterm warunkowy.

$$\left(\begin{array}{l}F(x)\ ;\\ ;\\ R(y)\\ ;\\ S(t)\end{array}\right)\left|\begin{array}{l}P(x,y)\ ;\ u\text{-}?\\ ;\\ R(y)\\ ;\\ K(z)\end{array}\right. \tag{1}$$

Analizując sekwencje eliminowania za pomocą warunku *u*, należy zwrócić uwagę na to, że tylko uniterm *R(y)* występuje w obu sekwencjach. Optymalizowanie algorytmu, ze względu na ten uniterm, można zrobić dodając dodatkowy warunek u_1. Postać zoptymalizowanej formuły ze względu na uniterm *R(y)* przedstawia wzór (2).

$$\left(\begin{array}{l}\left(\begin{array}{l}F(x)\ ;\\ ;\\ u_1=1\end{array}\right)\left|\begin{array}{l}P(x,y)\ ;\ u\text{-}?\\ ;\\ u_1=0\end{array}\right.\\ ;\\ \left(\begin{array}{l}R(y)\\ ;\\ S(t)\ ;\ K(z)\ ;\ u_1\text{-}?\end{array}\right.\end{array}\right. \tag{2}$$

3. Dekompozycja podsystemu

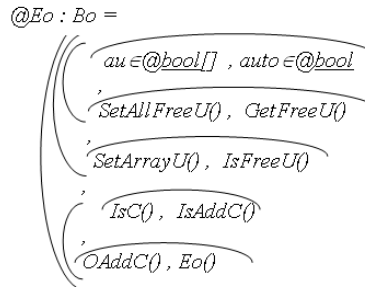
Model podsystemu *@Eo* jest skomplikowany, co utrudnia jego budowę w całości. Podsystem *@Eo* powstał na bazie podsystemu *@Bo* [4], co zostało zapisane w formule algorytmu znakiem dwukropka we wzorze (3). Dwukropek oznacza, że podsystem *@Eo* zawiera składowe podsystemu *@Bo* [4]. W podsystemie zawarto zmienne niezbędne do realizacji optymalizacji:

- *au*$\in$*@bool[]* –reprezentuje tablice (*[]* – identyfikator tablicy z nazwą *au,* zawierającej elementy typu *bool* [5]),
- *auto*$\in$*@bool* – przechowuje tryb wprowadzania dodatkowego warunku (automatyczny albo ręczny),

oraz unitermy funkcyjne wykonujące proces optymalizacji:

- *SetAllFreeU*() – ustawia elementy tablicy *au* na *false*,
- *GetFreeU*() – zwraca indeks wprowadzanego warunku,
- *SetArrayU*() – ustawia tablice *au*,
- *IsFreeU*() – sprawdza dostępność warunku,
- *IsC*() oraz *IsAddC* – sprawdzają możliwość wprowadzenia dodatkowego warunku,
- *OAddC*() – dokonuje optymalizacji metodą wprowadzenia dodatkowego warunku,

- *Eo*() – inicjuje początkowe wartości zmiennych.



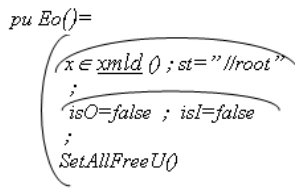
(3)

4. Zmienne

Unitermy zmiennych służą do przechowywania danych. Podsystem *@Bo* udostępnia zmienną *x* przechowującą *xml*, opisującą formułę algorytmu. Zmienna *st* jest pojedynczym indeksem węzła *xml*, który ma być poddany analizie. W celu przeprowadzenia optymalizacji formuł algorytmów metodą wprowadzenia dodatkowego warunku konieczne jest wprowadzenie zmiennej *au* jako tablicy, służącej do przechowywania informacji o dostępności wprowadzanego warunku u_n, gdzie *n* oznacza liczbę naturalną z zakresu od 0 do *m*. Tablica *au* stanowi mapę dostępności warunku u_n Jeśli wartością elementu tablicy jest *false* to warunek może zostać użyty. Zmienna *auto*∈*@bool* przechowuje informacje o sposobie wprowadzania dodatkowego warunku. Gdy zmienna ta przyjmuje wartość *true* to wprowadzanie warunku odbywa się automatycznie, jeśli zaś *false,* to użytkownik systemu nadaje samodzielnie dodatkowy warunek.

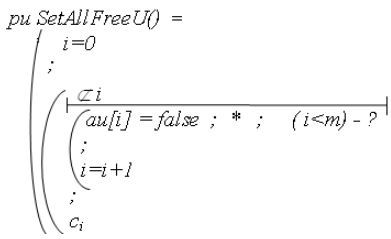
5. Unitermy funkcyjne

Unitermy funkcyjne mają za zadanie wykonanie działań na zmiennych i mogą, ale nie muszą, zwracać dane. Wprowadzono uniterm funkcyjny, zaprezentowany wzorem (4), inicjujący ustawienie początkowych wartości zmiennych podsystemu *@Eo* i zawierający zmienne udostępnianie z podsystemu *@Bo*, a mianowicie: zmienną *isO*∈*@bool,* przechowującą informacje o dokonaniu optymalizacji, *isI*∈*@bool*, informującą o przebiegu optymalizacji.



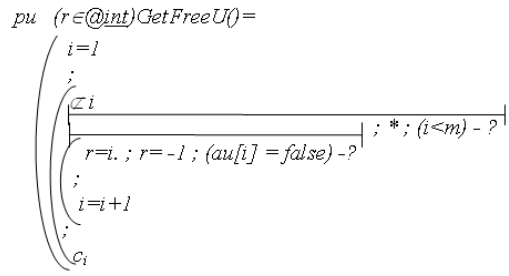
(4)

Uniterm funkcyjny *SetAllFreeU()* opisuje wykonanie ustawienia wszystkich komórek tablicy na wartość *false* i jest podany formułą (5), gdzie *i*∈*@int* - numer komórki tablicy *au*.



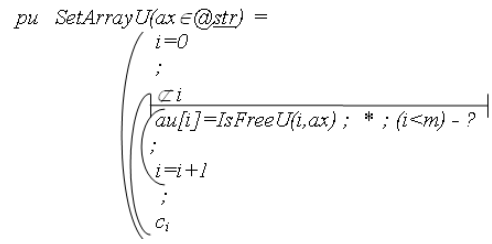
(5)

Do zwrócenia pierwszego wolnego warunku służy uniterm funkcyjny *GetFreeU*(), przedstawiony wzorem (6). Jeśli wszystkie elementy tabicy *au* są zajęte (tzn. mają ustawioną wartość *true*), to zwracana jest liczba –1, a jeśli nie, to zwraca pierwszy wolny warunek. Uniterm funkcyjny *GetFreeU*() zawiera uniterm zwracający zmienną *r,* będącą indeksem pierwszego elementu *false* oraz zmienną *i* standardowego typu *int* [5].



(6)

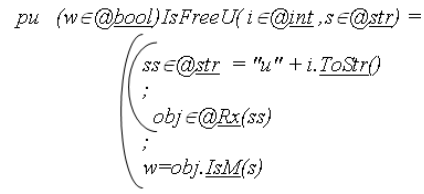
Do ustawienia tablicy *au* służy uniterm funkcyjny *SetArrayU*(), opisany formułą (7) i zawierający parametr wejściowy *ax*, który przedstawia kod *xml* oraz uniterm funkcyjny *IsFreeU*(), podany wyrażeniem (8), sprawdzający czy *i* – ty warunek jest dostępny.



(7)

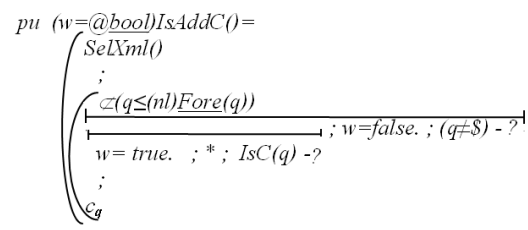
Uniterm funkcyjny *IsFreeU*() zwraca *false* wtedy, kiedy warunek nie jest użyty w całej formule algorytmu i zawiera parametry wejściowe:

- *i* – indeks warunku,
- *s* –kod *xml*,
- *w*∈*@bool* – parametr wyjściowy, zwracający informacje czy można użyć *i-ty* warunek;
- *i.ToStr*() – uniterm funkcyjny, konwertujący indeks warunku na ciąg tekstowy,
- *@Rx(ss)* – podsystem związany z wyrażeniami regularnymi i tworzący zmienną *obj*;
- *IsM*(s) – uniterm funkcyjny sprawdzający czy w zmiennej *s* występuje fragment ciągu tekstowego *ss* (zawraca wartość logiczną *true* zmiennej *w*, jeśli ciąg zadanego warunku *ss* znajduje się w zmiennej *s*).



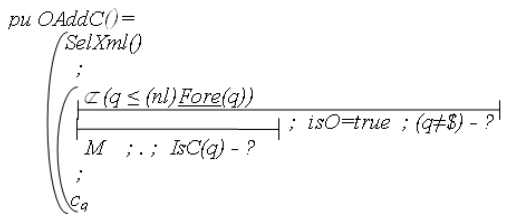
(8)

Uniterm funkcyjny *IsAddC()* jest opisany formułą (9). Zwraca wartość *true* w parametrze wyjściowym *w,* jeśli możliwa jest optymalizacja dla podanego indeksu *st* oraz zawiera wywołanie unitermu funkcyjnego *SelXml()* – pobierającego dane do tablicy wezłów *nl,* w operacji cyklicznego eliminowania algebry algorytmów. Uniterm funkcyjny (*nl)Fore(q)* ustanawia ilość elementów tablicy *nl.*



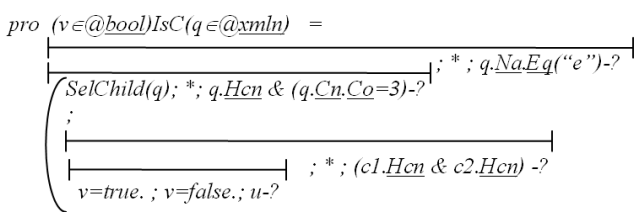
(9)

Uniterm funkcyjny *OAddC*() dokonuje przekształcenia formuły algorytmu na postać optymalną przez wprowadzenie dodatkowego warunku. Wyrażony jest formułą (10), gdzie *q*– zmienna do przechowywania fragmentu kodu *xml*, *IsC*() – uniterm funkcyjny przedstawiony formułą (11), przeznaczony dla sprawdzania czy dla podanego pramteru można zastosować metodę wprowadzania dodatkowego warunku.



(10)

M – uniterm funkcyjny tworzący zoptymalizowaną postać formuły algorytmu. Jest przedstawiony wzorem (12).



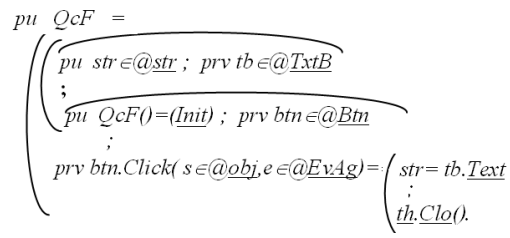
(11)

Uniterm funkcyjny *IsC()* zawiera:
- *v* – zmienna wyjściowa zawierająca informacje o możliwości wprowadzenia dodatkowego warunku,
- *q* - parametr wejściowy kodu *xml*, określający fragment kodu formuły algorytmu,
- $u = u_0 \& u_1 \& u_2 \& u_3 \& u_4 \& u_5 \& u_6 \& u_7$ - uniterm warunku, gdzie znak *&* oznacza iloczyn logiczny, oraz:
- u_1 = *c1.Na.Eq("seq")* – zmienna oznaczająca czy nazwa pierwszego podwezła *q* równa jest *sequence,*
- u_2 = *c2.Na.Eq("seq")* – zmienna oznaczająca czy nazwa drugiego podwezła *q* jest równa *sequence,*
- u_3 = *c1.Cd*[0].*Hcn* – zmienna oznaczająca, czy pierwszy podwęzeł węzła *c1* zawiera inne podwęzły,
- u_4 = *c2.Cd*[0].*Hcn* – zmienna oznaczająca, czy pierwszy węzeł *c2* zawiera inne podwęzły,
- u_5 = *c1.Cd*[0].*Cd*[1].*InX* = *c2.Cd*[0].*Cd*[1].*InX* – zmienna oznaczająca, czy porównane wartości tekstowe zawarte w zmiennej standardowej *InnerXml* oznaczone, jako *InX*, dla węzłów *c1* i *c2* odpowiednich podwęzłów, są równe,
- u_6 = *c1.Cd*[0].*InX* ≠ *c2.Cd*[0].*InX*, – zmienna oznaczająca, czy wartości tekstowe drugich podwęzłów węzłów *c1 i c2* różnią się od siebie,
- u_7 = *c1.Cd*[1].*InX* ≠ *c2.Cd*[1].*InX* – zmienna oznaczająca, czy wartości tekstowe drugich podwęzłów węzłów *c1* i *c2* różnią się od siebie wartościami (bez nazwy podwęzłów).

Wprowadzenie dodatkowego warunku możliwe jest tylko wtedy gdy wszystkie sprawdzane warunki w funkcyjnym unitermie *IsC*() przyjmą wartość *true*.

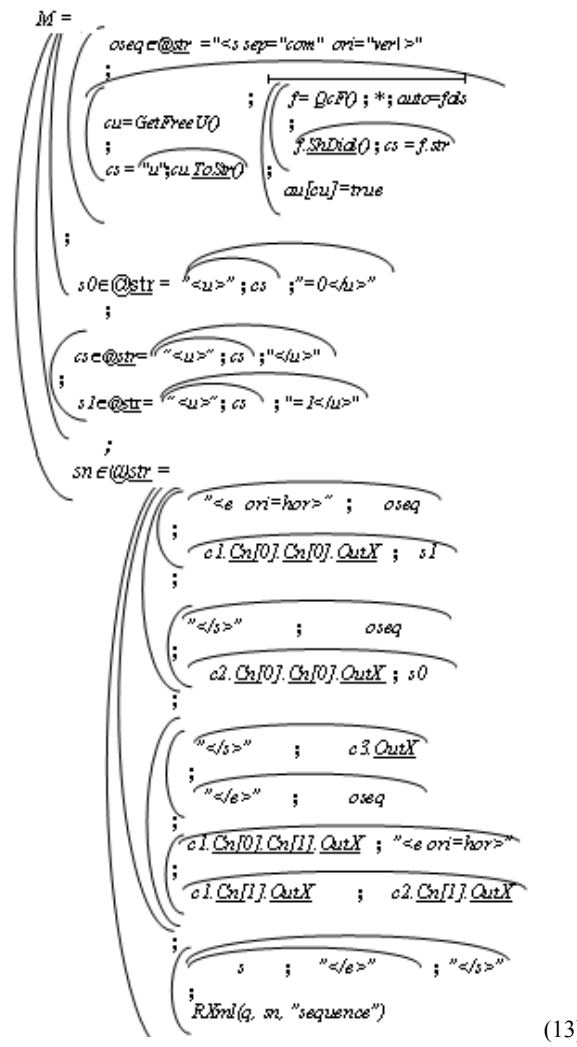
Uniterm funkcyjny *IsAddC*() służy do sprawdzenia wszystkich węzłów wybranych przez *SelXml()*. Uniterm funkcyjny oznaczony jako *c1.Cn*[0] odpowiada pierwszemu podwęzłowi węzła *c1*, zaś *c1.Cn*[1] odpowiada drugiemu podwęzłowi wezła *c1*. Uniterm *OAddC()* dokonuje optymalizacji. Uniterm standardowej zmiennej *.InX* oznacza tekst formuły zapisanej w kodzie *xml* bez nazwy węzła, który ją zawiera.

Budowany system zakłada wprowadzenie przez użytkownika nowego warunku, automatyczne lub ręczne. W tym celu utworzono funkcyjny uniterm graficzny (*QcF)*, przedstawiony formułą (12). *QcF()* zawiera zmienną *str* przechowującą tekst warunku *str*. Zmienna *str* typu *string* przechowuje tekst. Zmienna *tb* jest obiektem standardowego typu *TextBox*. Służy ona do wprowadzenia warunku (reprezentuje okienko tekstowe). Obiekt *btn* zatwierdza warunek (reprezentuje go przycisk typu *Button)*. *Init* – uniterm funkcyjny *InitializeComponent()*, inicjujący interfejs graficzny. *s* - parametr wejściowy standardowego typu *object, e* - parametr wejściowy standardowego typu *EvenArg, th.Clo* – uniterm funkcyjny *this.Close*(), zamykający uniterm funkcyjny. Uniterm *M* zawiera zmienną *oseq*, zawierającą ciąg tekstowy „*<sequence separator =\"comma\" orientation=\"veritical\">*”, który otwiera węzeł *xml.*



(12)

Utworzono obiekt *f* okna graficznego *QcF()*, do ręcznego wprowadzenia nazwy warunku. *f.ShDial* – wyświetla okienko graficzne, *f.str* – zmienna zawierająca wprowadzony przez użytkownika warunek.



(13)

Uniterm funkcyjny *M* zawiera wartości tekstowe:
- ”<*u*>”, oznaczającą „<*uniterm*>” otwierający,
- ”</u>” oznaczającą tekstowy znacznik zamykający węzeł ”</uniterm>”,
- ”<*e ori=hor*>” oznaczającą „<*elimination orientation =”\ hozizontal\”>”*
- ”<*s*>” – uniterm tekstowy otwierający węzeł sekwencji,

- ”*</s>*” – uniterm tekstu zamykający węzeł operacji „*</sequence >*”,
- *OutX* – (*OuterXml*[4]) zmienna zwracająca kod *xml,* łącznie z nazwą węzła,
- *RXml*() – unitem funkcyjny podmieniający węzeł *q* na ciąg *sn,* o nazwie „*sequence*”.

6. Implementacja programowa

Zaprojektowany i opisany formułami algebry algorytmów podsystem *@Eo* zaimplementowano w języku C# w klasie o tej samej nazwie.

```
  class Eo : Bo
 {    public:
       bool[] au = new bool[100];
      bool auto = false;
public Eo()
        {    x = new XmlDocument();
             st = "//root";
             isO = false;
             isI = true; SetAllFreeU();}
public:
      void SetAllFreeU()
       {for(int i=0;i<99;i++) au[i] = false;
                     au[0] = true; }
int GetFreeU()
{      for (int i = 1; i < 99; i++)
       if (au[i] == false) return i;
        return -1;
}
void SetArrayU(string txtXmlAll)
{      for(int i = 0; i < 99; i++)
       au[i] = IsFreeU(i, txtXmlAll);}
bool IsFreeU(int i,string sa)
{      string ss = "u"+i.ToString();
       Regex obj = new Regex(@ss);
       return obj.IsMatch(sa);
}
protected bool IsC(XmlNode q)
{
if (q.Name.Equals("elimination"))
  {
if(q.HasChildNodes && q.ChildNodes.Count == 3)
       {
SelChild(q);
if(c1.HasChildNodes&&c2.HasChildNodes)
if(c1.Name.Equals("sequence")&&
c2.Name.Equals("sequence")&&
c1.ChildNodes[0].HasChildNodes&&
c2.ChildNodes[0].HasChildNodes&&
c1.ChildNodes[0].Name.Equals("sequence")&&
c2.ChildNodes[0].Name.Equals("sequence")
&&(
c1.ChildNodes[0].ChildNodes[1].InnerXml==c2.
Child-
Nodes[0].ChildNodes[1].InnerXml)&&(c1.ChildN
odes[0].InnerXml!=
c2.ChildNodes[0].InnerXml)&&(
c1.ChildNodes[1].InnerXml!=
c2.ChildNodes[1].InnerXml){return true;}
       }
 return false;
}
bool IsAddC()
{
 SelXml();
 foreach (XmlNode q in nodes)
 if (IsC(q))  return true;
            return false;
 }
void OAddC()
{
int cu;
string oseq, s, cs, s0, s1;
oseq="<sequence separator=\"comma\"”+
      "orientation=\"vertical\">";
SelXml();
foreach (XmlNode q in nl)
  {
 if(IsC(q))
 {   cu = GetFreeU();
     cs = "u" + cu.ToString();
     if (!auto)
       {
       QcF f = new QcF(); f.ShowDialog();
       cs = f.str; au[cu] = true;
       }
s = "<uniterm>"+cs+"</uniterm>";
s0="<uniterm>"+cs+"=0</uniterm>";
s1="<uniterm>"+cs+ "=1</uniterm>";
string sn ="<elimination "+
"orientation     =\"horizontal     \">"+oseq+
c1.ChildNodes[0].ChildNodes[0].OuterXml
+s1+"</sequence>"+oseq
+c2.ChildNodes[0].ChildNodes[0].OuterXml  +s0
+"</sequence>" + c3.OuterXml +
"</elimination>"+oseq+
c1.ChildNodes[0].ChildNodes[1].OuterXml+
<elimination  orienttion=\"horizontal\">"  +
c1.ChildNodes[1].OuterXml +
c2.ChildNodes[1].OuterXml+cnd+
</elimination>"+"</sequence>";
   RXml(q, sn, "sequence");
       isO = true;
    } //koniec warunku IsC
 }}}}
```

7. Wnioski

Z systemu KSO wyodrębniono podsystem *@Eo* w celu optymalizacji formuł algorytmów metodą wprowadzania dodatkowego warunku.

Dokonana dekompozycja podsystemu *@Eo* ułatwia trudne zadanie zbudowania modelu systemu optymalizacji formuł metodą wprowadzenia dodatkowego warunku.

Przez automatyzację procesów optymalizacji formuł algorytmów uzyskano oszczędność czasu przy implementacji formuł algorytmów.

8. Literatura

[1] Owsiak W., Owsiak A., Owsiak J.: Teoria algorytmów abstrakcyjnych. Modelowanie matematyczne systemów informacyjnych. Wyd. Pol. Opolskiej, Opole, 2005.
[2] Owsiak W., Owsiak A.: Rozszerzenie algebry algorytmów. Pomiary Automatyka Kontrola. nr 2, 2010, vol. 56, str. 184-188.
[3] Piaskowy A.: Algebra algorytmów w przykładach. Pomiary Automatyka Kontrola. nr 2, 2010, vol. 56, str.189-192.
[4] Markuszewski T.: Model systemu do przekształceń formuł algorytmów. Pomiary Automatyka Kontrola, nr 2, vol 57.
[5] Perry S.C.: C# i .NET. Wyd. Helion, Gliwice, 2006.

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