ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1994

Seria: INFORMATYKA z. 26 Nr kol. 1255

Przemysław SZMAL

KILKA UWAG O SYSTEMACH STEROWANYCH PRZEPŁYWEM ARGUMENTÓW, KANONICZNYCH POSTACIACH ALGORYTMÓW I ICH CHRONOLOGIZACJI[1]

Streszczenie. W artykule uściślono definicję rozpatrywanego w literaturze uproszczonego systemu realizacji algorytmów sterowanego przepływem argumentów. Przedstawiono sposoby opisu struktury takiego systemu i określono warunki jego poprawnego działania. Pokazano, jak uporządkować postać kanoniczną algorytmu realizowanego przez system, by poszczególne operacje korzystały wyłącznie ze zmiennych ustawianych przez operacje o numerach niższych.

SOME REMARKS ON SYSTEMS DRIVEN BY THE FLOW OF ARGUMENTS, CANONICAL FORMS OF ALGORITHMS AND THEIR CHRONOLOGIZATION

Summary. In the paper an attempt is made to precise a definition of a simplified system for algorithm realization driven by the flow of arguments. Methods of how to describe the structure of such a system are presented. Conditions of its proper work are defined. Moreover, it is shown how the statements of a canonical form of an algorithm should be rearranged to make individual statements use variables which are set by lower-numbered operations exclusively.

[1] Praca zrealizowana w ramach Projektu Badawczego (Grantu) nr KBN 3 P406 011 04.

QUELQUES REMARQUES SUR LES SYSTEMES CONTROLES PAR LE FLUX DES ARGUMENTS, LES FORMES CANONIQUES DES ALGORITMES ET LEUR CHRONOLOGISATION

Résumé. Dans l'article on essaie de préciser une définition d'un système simplifié de la réalisation d'algorithmes contrôlé par le flux des arguments. On présente des méthodes de la description de la structure d'un tel système, ainsi que les conditions de son travail propre. On montre aussi comment peut-on ranger les phrases d'une forme canonique d'un algorithme pour les faire utiliser exclusivement les variables élaborées par les opérations aux numéros plus bas.

1. Wstęp

Pojęcie algorytmu, często używane i intuicyjnie dość oczywiste, w różnych pracach definiowane jest różnie. W [1] przez to pojęcie rozumiany jest skończony, uporządkowany zbiór operacji, które po wykonaniu na wprowadzonych danych dają rozwiązanie dowolnego zadania z określonej klasy zadań.

W literaturze dostrzega się fakt, że jednoznaczny opis algorytmu można uzyskać definiując układ (względnie maszynę lub system), w ramach którego ma się odbywać faktyczna realizacja algorytmu, definiując operacje elementarne tego systemu i korzystając z przyjętego języka opisu tych operacji [2, s.14]. Wielu autorów ten fakt przyjmuje za oczywisty i – nie podkreślając go specjalnie – rozważa algorytmy dedykowane dla konkretnego rodzaju układu wykonującego. Wspomniane różnice w definicji pojęcia algorytmu mają związek z tym, jaki układ wykonujący autor ma na myśli.

W [1] zwrócono uwagę na możliwość realizacji operacji wchodzących w skład algorytmów w systemach o dwóch zasadniczo różnych zasadach działania: w systemach sterowanych przepływem operacji, SSPO[2] i w systemach sterowanych przepływem argumentów, SSPA. Rozważa się tam wyidealizowany model SSPA; niektóre elementy jego charakterystyki są podane implicite. Model ten nazwiemy nieiteratywnym systemem sterowanym przepływem argumentów, NSSPA. Opiszemy ten model, wprowadzając niezbędne uściślenia.

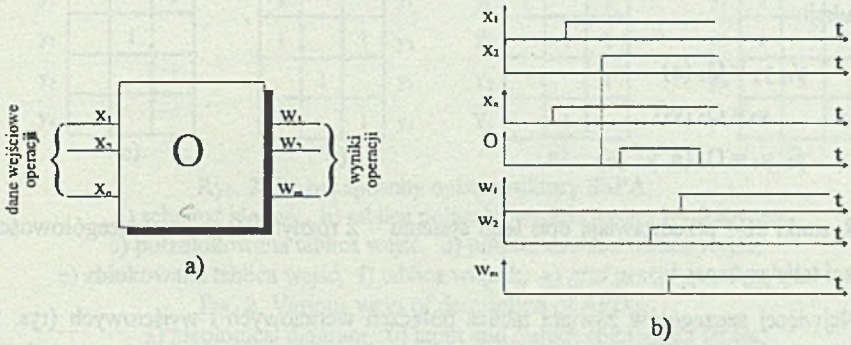
[2] Słowniczek wprowadzonych skrótów znajduje się na końcu artykułu.

2. Budowa systemu realizacji algorytmów sterowanego przepływem argumentów

W skład SSPA wchodzi pewna liczba procesorów odpowiedzialnych za wykonywanie poszczególnych operacji oraz infrastruktura połączeń.

Rysunek 1 przedstawia schematycznie procesor realizujący operację O na zespole argumentów – danych wejściowych $a_1, ..., a_n$ – i generującą zespół wyników $w_1, ..., w_m$. Poszczególne argumenty i wyniki są przekazywane za pośrednictwem osobnych linii wejściowych i wyjściowych.

Infrastrukturę połączeń stanowi zespół linii umożliwiających przekazywanie danych stałych oraz wyników generowanych przez poszczególne procesory na wejścia innych procesorów. Pojęcie "linia" jest umowne i oznacza dowolny środek umożliwiający przekazywanie danych – w konkretnym systemie może to być np. wydzielone połączenie galwaniczne, bufor w pamięci wykorzystany do składowania danej względnie identyfikator paczki danych przekazywanych za pośrednictwem sieci komputerowej.



Rys. 1. Procesor składowy SSPA: a) schemat ogólny, b) przebiegi czasowe
Fig. 1. AFDS member processor: a) a general view, b) time diagrams

System może zawierać mechanizmy umożliwiające buforowanie danych, powiązane z procesorami lub z liniami łączącymi. W tym artykule nie będziemy się interesować tymi mechanizmami.

Strukturę SSPA można przedstawiać na szereg sposobów.

Opis symboliczny układu to zbiór opisów poszczególnych operacji. Operacja realizowana przez i-ty procesor może być przedstawiona w postaci instrukcji przypisania:

$$Y_i = O_i (A_i), \tag{1}$$

gdzie:

O_i – operacja,

A_i – wektor argumentów,

Y_i – wektor wyników.

Argumenty i wyniki są identyfikowane z poszczególnymi liniami łączącymi.

Jeśli uwzględni się podział zbioru linii łączących **A** na zbiór linii stałych **D** i linii zmiennych **X**, można nieco zmodyfikować opis operacji:

$$Y_i = O_i (\mathbf{D}, X_i), \tag{2}$$

gdzie:

O_i – operacja,

D – wektor danych stałych,

X_i – wektor zmiennych wejściowych,

Y_i – wektor wyników (zmiennych wyjściowych).

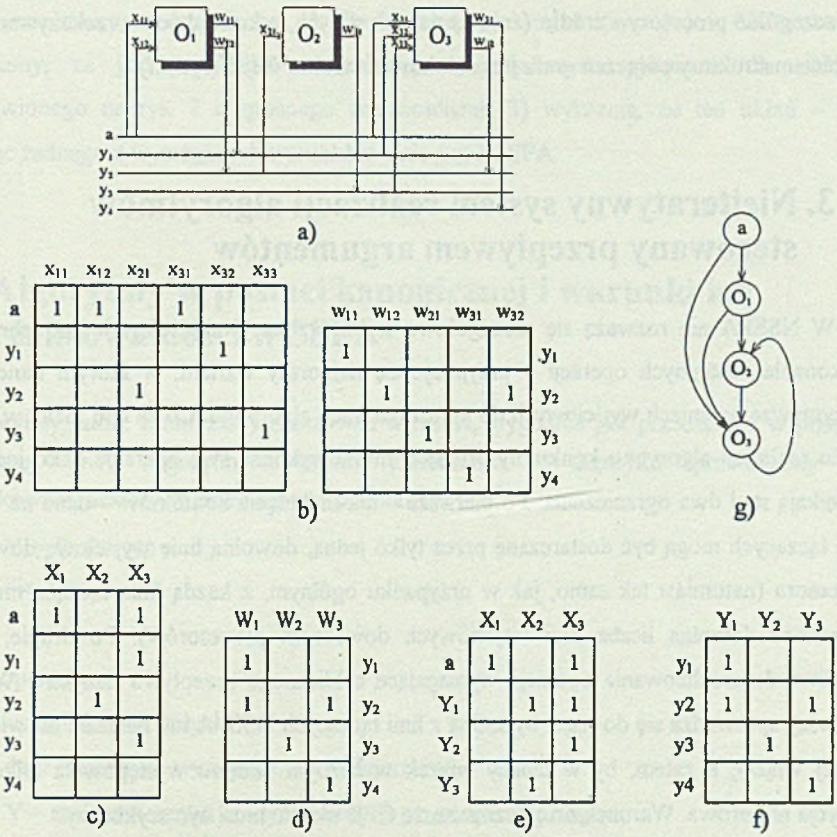
Przykładowo, układ przedstawiony na rys. 2a może być opisany za pomocą zbioru instrukcji:

$$\begin{aligned} y_1, y_2 &= O_1 (a) \\ y_3 &= O_2 (y_2) \\ y_2, y_4 &= O_3 (a, y_1, y_3) \end{aligned} \tag{3}$$

Rysunki 2b-f przedstawiają opis tego systemu – z różnym stopniem szczegółowości – w postaci tabelarycznej.

Najwięcej szczegółów zawiera tablica połączeń wejściowych i wyjściowych (rys. 2b) – pokazuje ona, z którymi liniami łączącymi połączone są poszczególne linie wejściowe i wyjściowe. Obecność połączenia symbolizuje wartość 1, brak połączenia – wartość 0 lub pole puste. Wartości te mogą być też interpretowane jako wartości logiczne.

Rysunki 2c i 2d przedstawiają półzblokowane tablice wejść i wyjść. Poszczególne wiersze obu tablic, jak w tablicach pełnych, odpowiadają liniom łączącym. Natomiast kolumny odpowiadające wszystkim wejściom poszczególnych procesorów zostały połączone, podobnie jak kolumny odpowiadające wyjściom.



Rys. 2. Różne sposoby opisu struktury SSPA:
a) schemat ideowy, b) tablice połączeń wejściowych i wyjściowych,
c) półzblokowana tablica wejść, d) półzblokowana tablica wyjść,
e) zblokowana tablica wejść, f) tablica wiązek, g) graf przepływu danych
Fig. 2. Various ways of description of AFDS:
a) ideological diagram, b) input and output connection tables,
c) semi-condensed input table, d) semi-condensed output table,
e) condensed input table, f) table of clusters, g) data-flow graph

W zblokowanej tablicy wyjść przedstawionej na rys. 2e poszczególne wiersze odpowiadają całym wiązkom linii łączących. Wiązka obejmuje wszystkie linie połączone z liniami wyjściowymi jednego procesora. Tablica wiązek na rys. 2f przedstawia, które linie łączące wchodzą w skład poszczególnych wiązek. Tablica ta jest identyczna z dokładnością do opisu kolumn – z półzblokowaną tablicą wyjść (por. rys. 2d).

Ostatnim z rozważanych w tym artykule sposobów opisu systemu jest (zblokowany) graf przepływu danych, GPD (por. rys. 2g). Jest to graf skierowany, w którym węzły reprezentują

poszczególne procesory i źródło (źródła) danych stałych, a krawędzie – przekazywane dane. Odbiciem struktury połączeń grafu jest zblokowana tablica wejść (rys. 2f).

3. Nieiteratywny system realizacji algorytmów sterowany przepływem argumentów

W NSSPA nie rozważa się szczegółowo trybu przekazywania danych potrzebnych do wykonania następnych operacji – przyjmuje się najgorszy wariant, w którym dane trzeba utrzymywać na liniach wyjściowych do końca realizacji algorytmu. Co za tym idzie, w jednym cyklu realizacji algorytmu konkretny procesor może wykonać swą operację tylko jeden raz. Wynikają stąd dwa ograniczenia. Po pierwsze – dla uniknięcia konfliktów – dane na każdą z linii łączących mogą być dostarczane przez tylko jedną, dowolną linię wyjściową dowolnego procesora (natomiast tak samo, jak w przypadku ogólnym, z każdą linią łączącą może być połączona dowolna liczba linii wejściowych dowolnych procesorów). Po drugie, nie są możliwe do zrealizowania operacje wymagające cyklicznego przepływu danych[3]. Warunek pierwszy sprowadza się do tego, by żadna z linii łączących NSSPA nie należała do więcej niż jednej wiązki, a zatem, by w tablicy wiązek w każdym wierszu występowała tylko jedna pozycja niezerowa. Warunek drugi oznacza, że GPD układu musi być acykliczny.

Rozważmy układ opisany zbiorem zależności:

$$
\begin{aligned}
y_1 &= O_1(a) \\
y_2, y_3 &= O_2(a, y_1) \\
y_4 &= O_3(y_2, y_5) \\
y_5 &= O_4(a, y_6) \\
y_6 &= O_5(a) \\
y_7 &= O_6(y_3, y_4, y_5)
\end{aligned}
\tag{4}
$$

[3] Model NSSPA można modyfikować. I tak, infrastruktura połączeń NSSPA umożliwia tylko przesył danych, bez ich gromadzenia. Wprowadzenie mechanizmu koordynującego przekazywanie danych między procesorami (łącznie np. z wyposażeniem linii lub procesorów w bufory kolejkowe) daje możliwość analizy realizacji w SSPA procesów cyklicznych lub takich, w których dane na jedną linię łączącą mogą być dostarczane przez kilka procesorów. Dopuszczenie możliwości przestrajania procesorów pozwala objąć analizą przypadek realizacji algorytmów przy ograniczonej liczbie dostępnych procesorów.

Jego strukturę przedstawia rys. 3. Badając tablicę wiązek (rys. 3c) i GPD układu stwierdzamy, że jest to NSSPA. Analogiczne sprawdzenia wykonane dla układu przedstawionego na rys. 2 i opisanego zależnościami (3) wykazują, że ten układ – nie spełniając żadnego z wymaganych warunków – nie jest NSSPA.

4. Algorytmy w postaci kanonicznej i warunki ich realizowalności w SSPA

Algorytmy, które zamierzamy realizować w SSPA, wygodnie jest przedstawić w postaci kanonicznej, PKA. Algorytm w postaci kanonicznej to algorytm sprowadzony do uporządkowanej sekwencji instrukcji przypisania [1]:

$$Y_i = O_i(\mathbf{D}, X_i) \qquad (5)$$
$$\text{dla } i = 1, \ldots, n$$

gdzie:

O_i – operacja,

D – wektor danych stałych,

Y – zbiór zmiennych występujących w algorytmie,

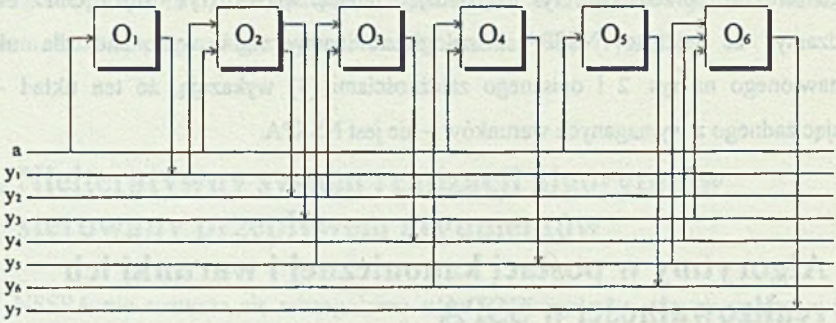
$Y_i \subset \mathbf{Y}$ – zbiór zmiennych modyfikowanych przez operację i-tą,

$X_i \subset \mathbf{Y}$ – zbiór zmiennych wejściowych wykorzystywanych przez operację i-tą,

n – liczba instrukcji w algorytmie.

Jak widać, zapis instrukcji składowych algorytmu jest identyczny z opisem operacji realizowanych przez procesor składowy SSPA (2). Wykorzystując zapis algorytmu w postaci kanonicznej można zatem skonstruować SSPA: dobrać niezbędną liczbę procesorów i odpowiednią infrastrukturę połączeń. Zwróćmy uwagę na to, że dla każdej zmiennej należy przewidzieć osobną linię łączącą.

W [1] do opisu algorytmu w postaci kanonicznej zaproponowano wykorzystanie tzw. macierzy zmiennych, MZ. W [1] definicję MZ zastąpiono przykładami, z których wynika, że MZ jest odpowiednikiem zblokowanej tablicy wejść, z pominięciem wiązki linii danych stałych lub – co na to samo wychodzi – odpowiednikiem półzblokowanej tablicy wejść dla przypadku, gdy każda z operacji ustawia jedną zmienną, być może złożoną, i gdy te zmienne są parami różne. MZ jest macierzą kwadratową, o n wierszach i n kolumnach.



a)

	X1	X2	X3	X4	X5	X6
a	1	1		1	1	
y1		1				
y2			1			
y3						1
y4						1
y5			1			1
y6				1		
y7						

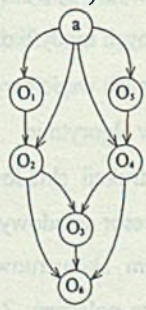
b)

	Y1	Y2	Y3	Y4	Y5	Y6
	W1	W2	W3	W4	W5	W6
y1	1					
y2		1				
y3		1				
y4			1			
y5				1		
y6					1	
y7						1

c)

	X1	X2	X3	X4	X5	X6
a	1	1		1	1	
Y1		1				
Y2			1			1
Y3						1
Y4			1			1
Y5				1		
Y6						

d)



e)

Rys. 3. Opis struktury przykładowego NSSPA:
a) schemat ideowy, b) półzblokowana tablica wejść, c) półzblokowana tablica wyjść i tablica wiązek, d) zblokowana tablica wejść, e) graf przepływu danych
Fig. 3. Description of sample NAFDS:
a) ideological diagram, b) semi-condensed input table, c) semi-condensed output table and table of clusters, d) condensed input table, e) data-flow graph

Na podstawie MZ można stwierdzić, czy między operacjami zachodzą zależności cykliczne [3]. Natomiast nie rozpatrując tablicy wiązek nie można określić, czy dane na liniach łączących SSPA pochodzą z oddzielnych źródeł. Bez dodatkowych założeń rozstrzygalny jest zatem tylko jeden z dwóch warunków pozwalających uznać system za NSSPA.

5. Chronologizowane postacie kanoniczne

W sytuacji, gdy decydujemy się na realizację algorytmu w SSPA, możemy zrezygnować z wymogu uporządkowania poszczególnych jego instrukcji, uwzględnionego w definicji PKA. Kolejność wykonywania operacji stanowi konsekwencję struktury połączeń systemu. Bezsprzecznie jednak odpowiednie uporządkowanie instrukcji może ułatwić analizę algorytmu i odpowiadającego mu systemu. Naturalna wydaje się propozycja uporządkowania – ponumerowania instrukcji tworzących PKA tak, by każda z nich korzystała wyłącznie z danych stałych oraz ze zmiennych wypracowywanych przez instrukcje o numerach niższych. Takie uporządkowanie jest możliwe – oczywiście – tylko wtedy, gdy między zmiennymi nie zachodzą zależności cykliczne.

Spróbujemy określić bardziej formalnie opisany warunek uporządkowania instrukcji. W tym celu zdefiniujemy kilka pomocniczych pojęć.

Wzorem [3] operację $Y_i = O_i$ (**D**, X_i) nazwiemy bezpośrednio zależną od (wyniku) operacji $Y_j = O_j$ (**D**, X_j) wtedy i tylko wtedy, gdy $X_i \cap Y_j \neq \emptyset$, tzn. gdy w zbiorze argumentów operacji O_i występuje jedna lub więcej zmiennych ustawianych przez operację O_j. W takiej sytuacji operację O_j nazwiemy operacją bezpośrednio wpływającą na operację (wynik operacji) O_i.

Przez ZOBZ (i) i ZOBW (i) oznaczymy odpowiednio zbiory operacji bezpośrednio zależnych i operacji bezpośrednio wpływających na operację O_i; zauważmy, że operacjom z ZOBW(i) odpowiadają niepuste pozycje w i-tej kolumnie MZ, a operacjom z ZOBZ(i) – niepuste pozycje w i-tym wierszu MZ.

Powiemy, że oznaczenie zmiennych w instrukcji $Y_i = O_i$ (**D**, X_i) jest nieinwersyjne, jeżeli ZOBW (i) nie zawiera żadnej operacji $Y_j = O_j$ (**D**, X_j) o numerze $j \geq i$. W przeciwnym przypadku oznaczenie zmiennych operacji O_i jest inwersyjne.

W algorytmie opisanym zależnościami (3) nieinwersyjne oznaczenia mają instrukcje odpowiadające operacjom O_1 i O_3. W algorytmie opisanym zależnościami (4) inwersyjne oznaczenia mają instrukcje odpowiadające operacjom O_3 i O_4.

Postać kanoniczną algorytmu, w której zmienne we wszystkich instrukcjach mają oznaczenie nieinwersyjne, będziemy nazywać postacią chronologizowaną, CPKA. Zauważmy, że macierz zmiennych CPKA jest macierzą trójkątną górną bez przekątnej głównej. Przekształcenie PKA w CPKA nazwiemy chronologizacją. Chronologizacja PKA w rzeczywistości stanowi odmianę sortowania topologicznego ([5, ss. 43-45]) zbioru instrukcji tworzących PKA. Od strony technicznej chronologizacja PKA sprowadza się do określenia sekwencji chronologizującej SC. SC zawiera numery operacji wchodzących w skład pierwotnej PKA, zapisane w takiej kolejności, w jakiej odpowiednie operacje mają być umieszczone w CPKA.

Podstawą do przekształcenia PKA w CPKA będzie ustalenie dla każdej operacji wchodzącej w skład PKA efektywnego poziomu operacji, EPO. EPO może być wyznaczany względem dwóch poziomów odniesienia: poziomu danych wejściowych algorytmu (EPOD) bądź poziomu wyników (EPOW). EPOD operacji O_i, oznaczany dalej przez EPOD(i), jest równy największej długości drogi prostej [4, s.78] w GPD algorytmu od węzła reprezentującego dane stałe do węzła odpowiadającego operacji O_i. EPOD wskazuje, ile kolejnych etapów przekształceń muszą w najgorszym przypadku przejść dane stałe, zanim przyjmą postać wyniku rozpatrywanej operacji. Analogicznie, EPOW operacji O_i jest równy największej długości drogi prostej w GPD algorytmu od węzła odpowiadającego operacji O_i do hipotetycznego węzła reprezentującego odbiorcę wyników algorytmu. EPOW określa, ile kolejnych etapów przekształceń muszą przejść w najgorszym przypadku dane wejściowe operacji, zanim przyjmą postać końcowych wyników algorytmu. W dalszym ciągu artykułu nasze rozważania ograniczymy do EPOD. Dostosowanie przedstawionych niżej definicji, algorytmów, przykładów i dyskusji do wariantu EPOW jest zadaniem elementarnym z uwagi na symetrię obu wariantów.

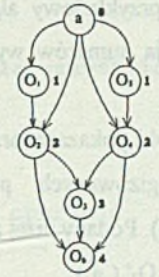
Dla węzłów odpowiadających źródłom (źródłu) danych stałych EPOD przyjmuje wartość 0. Jeśli rozpatrywany GPD jest acykliczny, to EPOD dla węzła odpowiadającego dowolnej operacji przyjmuje wartość równą maksymalnej wartości EPOD dla węzłów reprezentujących źródła jej poszczególnych argumentów, powiększoną o 1. Rysunek 4b przedstawia GPD z

rys.3e dla PKA opisanej zależnościami (4), z wyliczonymi wartościami EPOD wpisanymi obok poszczególnych węzłów.

Wartości EPOD można wyznaczyć na podstawie MZ układu. Służący temu celowi prosty algorytm, zapisany w pseudokodzie[4] jako procedura *wyznacz_EPOD*, przedstawiono w Dodatku. Algorytm jest zabezpieczony przed zapętleniem w przypadku, gdyby między zmiennymi zachodziły zależności cykliczne.

	X_1	X_2	X_3	X_4	X_5	X_6
Y_1		1				
Y_2			1			1
Y_3						1
Y_4			1			1
Y_5				1		
Y_6						

a)



b)

	X_1	X_5	X_4	X_2	X_3	X_6
Y_1				1		
Y_5			1			
Y_4					1	1
Y_2					1	1
Y_3						1
Y_6						
SC	1	5	4	2	3	6

c)

	X_5	X_4	X_1	X_2	X_3	X_6
Y_5		1				
Y_4					1	1
Y_1				1		
Y_2					1	1
Y_3						1
Y_6						
SC	5	4	1	2	3	6

d)

Rys. 4. Chronologizacja PKA danej zależnościami (4):
a) macierz zmiennych pierwotnej PKA, b) GPD algorytmu z zaznaczonymi wartościami EPOP, c) macierz zmiennych PKA chronologizowanej poziomowo, d) macierz zmiennych PKA chronologizowanej pionowo
Fig. 4. Chronologization of PKA given by relations (4):
a) variable matrix of primary algorithm canonical form,
b) DFG of the algorithm with EOL values shown,
c) VM of canonical form chronologized in level-based way,
d) VM of canonical form chronologized in pipe-line-based way

[4] Zastosowany pseudokod jest wzorowany na języku Pascal.

Znając wartości EPOD oraz zbiory ZOBZ i ZOBW poszczególnych operacji można przeprowadzić chronologizację PKA na kilka sposobów. Sposób pierwszy to chronologizacja poziomowa. PKA chronologizowana poziomowo charakteryzuje się tym, że występują w niej kolejno grupy operacji o jednakowych wartościach EPOD, od EPOD = 1 do wartości maksymalnej. Drugi sposób to chronologizacja pionowa, polegająca na umieszczeniu obok siebie – o ile to tylko możliwe – operacji bezpośrednio od siebie zależnych. Poniżej przedstawiamy przykładowy algorytm ustalający SC dla tego przypadku; przy tworzeniu CPKA sekwencja numerów wyprowadzona przez algorytm powinna być odczytywana od końca.

Na rys. 4c i 4d pokazano przykładowe sekwencje chronologizujące i macierze zmiennych PKA chronologizowanych poziomowo i pionowo, odpowiadające PKA opisanej zależnościami (4). Podany niżej zbiór zależności (6) uzyskano z (4) stosując SC z rys. 4d.

$$\begin{aligned} y_6 &= O_1'(a) \\ y_5 &= O_2'(a, y_6) \\ y_1 &= O_3'(a) \\ y_2, y_3 &= O_4'(a, y_1) \\ y_4 &= O_5'(y_2, y_5) \\ y_7 &= O_6'(y_3, y_4, y_5) \end{aligned} \tag{6}$$

Zwróćmy uwagę, że w (6) nie zmieniono oznaczeń zmiennych. Natomiast odpowiedniki operacji w obu postaciach algorytmu przedstawiono zbiorem zależności (7):

$$\begin{aligned} O_1' &= O_5 \\ O_2' &= O_4 \\ O_3' &= O_1 \\ O_4' &= O_2 \\ O_5' &= O_3 \\ O_6' &= O_6 \end{aligned} \tag{7}$$

Algorytm chronologizacji pionowej PKA

Dane: MZ postaci kanonicznej algorytmu

Wynik: Odwrócona sekwencja chronologizująca, zapewniająca przekształcenie danej PKA w PKA chronologizowaną pionowo

Metoda:

```
{ const  l_op;  -- liczba operacji w PKA  }
procedure chronologizacja( MZ : macierz_kw_bool );
  var
    ZWG : set of 1 .. l_op;  { zbiór numerów operacji "wolnych",
                               którym jeszcze nie przydzielono
                               nowego numeru }
    ob  : 1 .. l_op;         { operacja bieżąca }
begin
  wyznacz_EPOD( MZ );  { wyznaczenie EPOD dla wszystkich
                         operacji w PKA }
  ZWG := [1 .. l_op];
  while ZWG <> [] do
    begin
      { wyznaczenie/ustalenie numeru wolnej operacji
        o najwyższym EPOD : }
      ob  := nr_op_max_EPOD( ZWG );
      { przydzielenie numerów operacjom wpływającym na Oob :}
      przydziel_numery_w_poddrzewie( ob, ZWG );
    end;
end;     { chronologizacja }
```

Procedura rekurencyjna *przydziel_numery_w_poddrzewie* ma postać:

```
procedure przydziel_numery_w_poddrzewie
                 (ob : integer;  var ZWG : macierz_kw_bool );
{ ustalenie (odwróconej) sekwencji numerów operacji
  wpływających na operację bieżącą, ob }
  var
    ZWL : set of 1 .. l_op;  { lokalny zbiór numerów operacji
                               "wolnych" }
    op  : 1 .. l_op;         { operacja poprzednia }
begin
  writeln( ob );
  ZWG := ZWG - [ob];
  ZWL := ZOBW( ob ) * ZWG;
  while ZWL <> [] do
    begin
      { wybór operacji wpływającej na ob : }
      op  := nr_op_wpływ( ob, ZWL );
      ZWL := ZWL - [op];
      if ZOBZ( op ) * ZWG = [] then
        przydziel_numery_w_poddrzewie( op, ZWG );
        { numer operacji wolno przydzielić dopiero wtedy, gdy
          wszystkie operacje bezpośrednio od niej zależne już
          mają numery }
    end
end; { przydziel_numery_w_poddrzewie }
```

W przedstawionym algorytmie warto zwrócić uwagę na wywołania funkcji *nr_op_max_EPOD* i *nr_op_wpływ*. Zadaniem pierwszej z tych funkcji jest wybór "wolnej" operacji z najwyższą aktualnie wartością EPOD, zadaniem drugiej – wybór jednej spośród "wolnych" operacji bezpośrednio wpływających na operację aktualnie rozpatrywaną. W zależności od potrzeb może okazać się celowe ustalenie dodatkowych kryteriów wyboru operacji w obrębie każdej z funkcji, co wpłynie na charakter uzyskanej CPKA. Przykładowo, w funkcji *nr_op_wpływ* można wymusić wybór operacji o najniższej bądź najwyższej wartości EPOD. Na rys. 5 przedstawiono MZ kolejnego algorytmu, jego GPD i macierze zmiennych dla CPKA zbudowanych według SC uzyskanych z uwzględnieniem zaproponowanych obostrzeń.

Warto również zauważyć, że jeśli z procedury *przydziel_numery_w_poddrzewie* usunie się wszystkie instrukcje poza pierwszymi dwiema, to strywializowany algorytm będzie generował odwrócone sekwencje chronologizujące poziomowo.

SŁOWNICZEK SKRÓTÓW

CPKA – chronologizowana postać kanoniczna algorytmu

EPO – efektywny poziom operacji

EPOD – efektywny poziom operacji liczony względem danych wejściowych algorytmu

EPOW – efektywny poziom operacji liczony względem wyników algorytmu

GPD – graf przepływu danych

MZ – macierz zmiennych

NSSPA – nieiteratywny system (realizacji algorytmów) sterowany przepływem operacji

PKA – postać kanoniczna algorytmu

SC – sekwencja chronologizująca

SSPA – system (realizacji algorytmów) sterowany przepływem argumentów

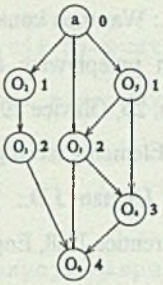
SSPO – system (realizacji algorytmów) sterowany przepływem operacji

ZOBW – zbiór operacji bezpośrednio wpływających (na rozpatrywaną operację)

ZOBZ – zbiór operacji bezpośrednio zależnych (od rozpatrywanej operacji)

	X_1	X_2	X_3	X_4	X_5	X_6
Y_1				1		1
Y_2			1			
Y_3						1
Y_4						1
Y_5	1			1		
Y_6						

a)



b)

	X_2	X_3	X_5	X_1	X_4	X_6
Y_2		1				
Y_3						1
Y_5				1	1	
Y_1					1	1
Y_4						1
Y_6						
SC	2	3	5	1	4	6

c)

	X_5	X_1	X_4	X_2	X_3	X_6
Y_5		1	1			
Y_1			1			1
Y_4						1
Y_2					1	
Y_3						1
Y_6						
SC	5	1	4	2	3	6

d)

Rys. 5. Zróżnicowanie PKA chronologizowanych pionowo:
a) macierz zmiennych pierwotnej PKA, b) GPD algorytmu,
c) MZ PKA chronologizowanej z preferencją dla operacji o wyższym EPOP,
d) MZ PKA chronologizowanej z preferencją dla operacji o niższym EPOP

Fig. 5. Variations of algorithm canonical forms chronologized in pipe-line-based way:
a) variable matrix of primary algorithm canonical form, b) DFG of the algorithm,
c) VM of canonical form chronologized with preference to operations of higher effective (source) level,
d) VM of canonical form chronologized with preference to operations of lower effective (source) level

LITERATURA

[1] Węgrzyn S.: Systemy sterowane przepływem operacji i systemy sterowane przepływem argumentów, Zeszyty Naukowe Politechniki Śląskiej, seria Informatyka, 24, Gliwice 1993.

[2] Goodman S.E., Hedetniemi, S.T.: Introduction to the design and analysis of algorithms (tłum. na język rosyjski: Wwiedienije w razrabotku i analiz algoritmow), "Mir", Moskwa 1981.

[3] Wyciślik M.: Warunek konieczny i wystarczający realizowalności algorytmu w systemach sterowanych przepływem argumentów, Zeszyty Naukowe Politechniki Śląskiej, seria Informatyka, 26, Gliwice 1994.

[4] Korzan B.: Elementy teorii grafów i sieci. Metody i zastosowania, WNT, Warszawa 1978.

[5] Aho A.V., Ullman J.D.: The theory of parsing, translation and compiling, Vol.1: "Parsing", Prentice-Hall, Englewood Cliffs 1972.

DODATEK. PROCEDURA WYZNACZAJĄCA EFEKTYWNY POZIOM OPERACJI W SYSTEMIE STEROWANYM PRZEPŁYWEM ARGUMENTÓW (SSPA)

```
{ const  l_op;  -- liczba operacji
  type wektor_int       = array [1 .. l_op] of integer;
       macierz_kw_bool = array [1..l_op, 1..l_op] of boolean;
}
procedure wyznacz_EPOD(MZ: macierz_kw_bool; var EPOD: wektor_int);
  var
    nr_op, nr_it : integer;
    m_wej, m_wyn : macierz_kw_bool;
begin
  { przygotowanie tablicy EPOD }
  for nr_op := 1 to l_op do  EPOD [nr_op] := 1;
  m_wyn := MZ;
  nr_it := 1;
  while (nr_it <= l_op) and korekta_tab_EPOD(m_wyn, EPOD) do
    begin
      m_wej := m_wyn;
      mnóż( m_wej, MZ, m_wyn);  { mnożenie macierzy }
      nr_it := nr_it + 1;
    end;
    { pętla jest wykonywana tyle razy, ile wynosi maksymalny
      stopień zależności zmiennych w MZ (por. [3]) }
end;  { wyznacz_EPOD }

function korekta_tab_EPOD(m_spr     : macierz_kw_bool;
                          var EPOD : wektor_int ) : boolean;
{ skorygowanie zawartosci tablicy efektywnych poziomów
  operacji EPOD; zwiększenie o 1 EPOD operacji i-tej, jeżeli
  w i-tej kolumnie m_spr przynajmniej jedna pozycja jest
  "niepusta", tzn. ma wartość true}
```

```
  var
    nw, nk : integer; { numer wiersza, numer kolumny }
    zm_rob : boolean; { flaga obecności pozycji niepustych
                        w kolumnie }
    wynik  : boolean;{ flaga obecności pozycji niepustych
                       w całej tablicy }
begin
  wynik := false;

  for nk := 1 to l_op do
    begin
      { ustalenie, czy w kolumnie nk jest pozycja niepusta }
      zm_rob := false;
      for nw := 1 to l_op do
        { zaznaczenie zależności zmiennej od wyniku poprzedniej
          fazy }
        zm_rob := zm_rob or m_spr [nw][nk];

      { odnotowanie zmiany w bieżącej kolumnie }
      if zm_rob then  EPOD [nk] := EPOD [nk] + 1;

      { odnotowanie zmiany dotyczącej całej tabeli }
      wynik := wynik or zm_rob;
    end;
  korekta_tab_EPOD := wynik;
end;  { korekta_tab_EPOD }

procedure mnóż( mnożna, mnożnik : macierz_kw_bool;
                var tab_wyn     : macierz_kw_bool );
{ wyznaczenie i umieszczenie w tab_wyn iloczynu macierzy
  mnożna i mnożnik; operacje są realizowane w algebrze Boole'a }

  var
    nw, nk, ib : integer;
    zm_poz     : boolean;  {nowa wartość bieżącej pozycji tabeli }
begin
  for nk := 1 to l_op do
    for nw := 1 to l_op do
      begin
        zm_poz := false;

        for ib := 1 to l_op do
          zm_poz := zm_poz or
                    (mnożna [nw, ib] and mnożnik [ib, nk]);

        tab_wyn [nw, nk] := zm_poz; { wpisz bieżący element }
      end;
end;  { mnóż }
```

Recenzent: Dr inż. Andrzej Wilk

Wpłynęło do Redakcji 2 marca 1994 r.

Abstract

Some recently published papers [1, 3] deal with argument-flow driven systems for algorithm realization (AFDS). However, not all essential properties of AFDS model used there are stated explicitly. Actually, an attempt was made to describe such a model, called non-iterative AFDS, shortly – NAFDS, in a more precise way.

NAFDS can be defined analytically as an ordered set of operations; they can have a form of assignment statements. It is also possible to use the form of ideological diagram (fig. 2a), input and output connections table (fig. 2b), semi-condensed input and output tables (fig. 2c, 2d), condensed input table and table of clusters (fig.2e, 2f), and data-flow graph (DFG) (fig. 2g). In Section 3 one shows that NAFDS can work when its DFG is acyclic, and individual rows of its table of clusters contain just one non-empty item.

A semi-condensed input table with removed row of constant data is equivalent to the variable matrix (VM) of some canonical form of the algorithm realized by the system. One can chronologize canonical forms – i.e. order them so that their statements use exclusively variables worked-out by the lower-numbered instructions. In Section 5 one shows how to perform chronologization, given the VM of a canonical form, using effective level (EOL) of individual operations and the sets of directly dependent and directly influencing operations. A level-based chronologization (operations ordered by ascending EOL) and pipe-line-based chronologization (directly dependent operations coming together) is considered. An algorithm – in the form of procedure *chronologizacja* – is proposed to elaborate a pipe-line-based chronologizing sequence (of operation numbers) of a canonical form. Examples of chronologized canonical forms are given.