ZESZYTY
NAUKOWE
POLITECHNIKI
ŚLĄSKIEJ

ZBIGNIEW CZECH

ALGORYTM ANALIZY REGIONÓW DLA PROBLEMÓW ANALIZY PRZEPŁYWU DANYCH

INFORMATYKA

Z. 6
GLIWICE
1984

POLITECHNIKA ŚLĄSKA

ZESZYTY NAUKOWE

Nr 796

ZBIGNIEW CZECH

ALGORYTM ANALIZY REGIONÓW DLA PROBLEMÓW ANALIZY PRZEPŁYWU DANYCH

GLIWICE 1984

OPRACOWANIE REDAKCYJNE

Alicja Nowacka

Wydano za zgodą
Rektora Politechniki Śląskiej

PL ISSN 0208-7286

Dział Wydawnictw Politechniki Śląskiej
ul. Kujawska 3, 44-100 Gliwice

Nakł. 150+85 Ark. wyd 6 Ark. druk. 6,75 Papier offset. kl. III. 70x100 70
Oddano do druku 27.02.1984 Podpis. do druku 10.05.1984 Druk ukończ. w maju 1984
Zam 346 84 U-23 Cena zł 60,—

Skład, fotokopie, druk i oprawę
wykonano w Zakładzie Graficznym Politechniki Śląskiej w Gliwicach

SPIS TREŚCI

1. WSTĘP

Jednym z zagadnień, jakie wyodrębniły się na przestrzeni ostatnich kilkunastu lat w dziedzinie oprogramowania komputerów, jest analiza przepływu danych. Obiektem takiej analizy są dowolne programy komputerowe, a informacje uzyskane po jej przeprowadzeniu mogą być bezpośrednio użyte do ulepszania programów wynikowych tworzonych w czasie kompilacji lub do kontroli poprawności (weryfikacji) programów. Wyniki analizy mogą być również pomocne przy uruchamianiu (tj. wykrywaniu i usuwaniu błędów), testowaniu, certyfikowaniu, modyfikowaniu oraz dokumentowaniu programów.

Przeprowadzenie analizy przepływu danych polega na rozwiązaniu kilku problemów. Informacje będące rozwiązaniem każdego z nich są otrzymywane w wyniku analizy struktury programu oraz sposobów i miejsc użycia danych w programie. Niniejsza rozprawa poświęcona jest efektywnym algorytmom rozwiązywania dwóch z tych problemów, a mianowicie: problemu definicji osiągających oraz problemu zmiennych aktywnych.

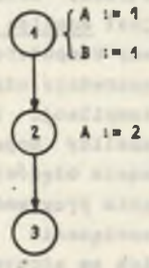
Dla potrzeb analizy przepływu danych wygodnie jest posługiwać się grafowym modelem analizowanego programu. W tym celu program dzielony jest na bloki podstawowe, tj. maksymalne ciągi instrukcji, których wykonanie przebiega sekwencyjnie, bez skoków. Z bloków podstawowych konstruowany jest graf przepływu sterowania lub krócej, graf przepływu programu. Wierzchołki grafu przepływu odpowiadają blokom podstawowym w programie, zaś skierowane krawędzie pomiędzy wierzchołkami odpowiadają przepływowi sterowania pomiędzy blokami podstawowymi. Jeden z wierzchołków grafów jest wyróżniony jako początkowy: jest to wierzchołek odpowiadający blokowi zawierającemu pierwszą instrukcję programu.

Przedstawimy obecnie w sposób nieformalny problemy analizy przepływu danych, które rozważa się w niniejszej rozprawie oraz krótko omówimy możliwości zastosowania informacji, jakie uzyskuje się w wyniku ich rozwiązania.

Problem definicji osiągających polega na wyznaczeniu zbioru definicji zmiennych, które mogą osiągnąć wejście każdego wierzchołka w grafie przepływu. Przez definicję zmiennej V rozumie się instrukcję przypisania wartości, np. V := e, gdzie e oznacza wartość lub instrukcję czytania danych, np. read(V). Mówimy, że definicja zmiennej V osiąga punkt p (np. wejście wierzchołka), jeśli w grafie przepływu istnieje droga od tej definicji do p, na której nie występuje żadna inna definicja V.

Przykład 1.1

Wejście wierzchołka 2 grafu przepływu z rys. 1.1 osiągają definicje A := 1 oraz B := 1, zaś wejście wierzchołka 3 - definicje A := 2 oraz B := 1. Definicja A := 1 nie osiąga wejścia wierzchołka 3, ponieważ na drodze prowadzącej od tej definicji do wejścia wierzchołka 3 występuje inna definicja tej samej zmiennej.□



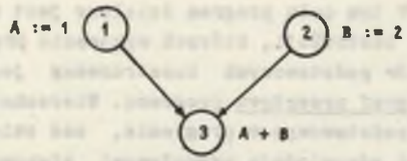
Rys. 1.1. Ilustracja osiągania przez definicje wierzchołków grafu przepływu

Zbiory definicji osiągających lub krócej zbiory rdef) mogą być wykorzystane w kilku przypadkach.

Znajomość zbiorów rdef umożliwia wykonanie składania stałych w czasie kompilacji programu, tj. zastąpienie wyrażeń występujących w programie przez ich wartości wtedy, gdy wartości te można obliczyć.

Przykład 1.2

Rozważmy wyrażenie A + B występujące w wierzchołku 3 fragmentu grafu przepływu z rys. 1.2 oraz załóżmy, że jedynymi definicjami A i B osiągającymi wierzchołek 3 są A := 1 i B := 2. (Zwróćmy uwagę, że informację taką możemy uzyskać dopiero po rozwiązaniu problemu definicji osiągających). Wówczas wyrażenie A + B w wierzchołku 3 może być zastąpione(w czasie kompilacji programu) przez stałą wartość równą 3.□



Rys. 1.2. Ilustracja składania stałych

Warto zauważyć, że tego typu metoda ulepszania programu - przez realizację obliczeń w czasie kompilacji, zamiast w czasie wykonywania programu - przynosi szczególnie duże zyski wtedy, gdy składane wyrażenia zawarte są w najczęściej wykonywanych fragmentach programu, np. w najbardziej wewnętrznych pętlach.

Zbiory rdef są użyteczne przy eliminacji zbędnych fragmentów programu. możemy bowiem na ich podstawie stwierdzić czy dowolna z definicji osiąga jakikolwiek blok podstawowy (wierzchołek grafu przepływu) programu. Jeżeli definicja zmiennej nie osiąga żadnego bloku w programie, a więc nie ma wpływu na żadne użycie zmiennej, to może być wyeliminowana[1].

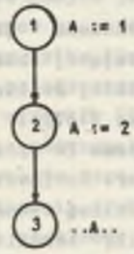
[1] Oprócz składania stałych oraz eliminacji zbędnych fragmentów programu istnieje kilka bardziej wyszukanych zastosowań zbiorów rdef w kompilatorach optymalizujących. Dla przykładu, mogą być one użyte do wykrywania, a następnie przemieszczania obliczeń niezmiennych w pętlach, poza pętle oraz być pomocne przy eliminowaniu zmiennej indukcyjnej (Aho i Ullman [1977]).

Trzecim przykładem zastosowania zbiorów rdef, związanym z diagnozą błędów występujących w programie, jest detekcja zmiennych o niezdefiniowanych wartościach. Wprowadzając dodatkową definicję dla każdej zmiennej V przed początkowy wierzchołek grafu przepływu możemy stwierdzić czy dowolna z tych definicji osiąga jakikolwiek wierzchołek zawierający użycie V. Jeżeli tak, to oznacza, że w czasie wykonywania programu może wystąpić użycie zmiennej V bez wcześniejszego nadania jej wartości. Tego typu informacje mogą być wykorzystane przez kompilator do drukowania odpowiednich ostrzeżeń przeznaczonych dla użytkownika.

Problem zmiennych aktywnych polega na wyznaczeniu zbiorów zmiennych, które są aktywne na wejściu i wyjściu każdego wierzchołka w grafie przepływu. (Zbiory te będziemy oznaczać w skrócie przez lvar.). Mówimy, że zmienna V jest aktywna w punkcie p (na przykład na wejściu wierzchołka), jeżeli istnieje droga w grafie przepływu nie zawierająca żadnej definicji zmiennej V, od punktu p do użycia bieżącej wartości V. Jeżeli droga taka nie istnieje, to zmienna V jest nieaktywna. Innymi słowy chcemy wiedzieć, dla zmiennej V i punktu p, czy wartość V w p może być użyta na którejś z dróg grafu przepływu wychodzących z p.

Przykład 1.3

W grafie przepływu z rys. 1.3 zmienna A jest nieaktywna na wyjściu wierzchołka 1 oraz na wejściu wierzchołka 2, natomiast jest aktywna na wyjściu wierzchołka 2 oraz na wejściu wierzchołka 3.□



Rys. 1.3. Ilustracja aktywności zmiennej (..A.. oznacza użycie zmiennej)

Zbiory lvar znajdują zastosowanie podczas generowania programów wynikowych w czasie kompilacji. Ich znajomość umożliwia prowadzenie efektywnej gospodarki rejestrami oraz pamięcią operacyjną.

Mianowicie, jeżeli wartość dowolnej zmiennej zostaje obliczona w rejestrze, a następnie użyta wewnątrz bloku, to zbędne jest chowanie tej wartości do pamięci operacyjnej (ściślej: zbędne jest generowanie w programie wynikowym rozkazów chowanie tej wartości) w przypadku, gdy zmienna ta jest nieaktywna na wyjściu bloku.

Jeżeli w trakcie przyznawania zmiennym rejestrów, wszystkie z nich są zajęte, a niezbędne jest przyznanie następnego rejestru, to powinniśmy użyć tego z nich, który przechowuje wartość nieaktywną, ponieważ wartość ta nie musi być pamiętana.

Zbiory lvar mogą być pomocne nie tylko przy prowadzeniu efektywnej gospodarki pamięcią, lecz również przy ulepszaniu programu poprzez przemieszczanie jego fragmentów realizujących obliczenia niezmienne w pętlach, poza pętle (Aho i Ullman [1977]).

Dokonamy teraz pewnych klasyfikacji związanych z analizą przepływu danych.

Jak już wspomniano, przeprowadzenie analizy przepływu danych polega na rozwiązaniu kilku problemów. Każdy z nich ma na celu uzyskanie określonych informacji dotyczących przepływu danych, biorąc za punkt wyjścia graf przepływu sterowania programu. W trakcie rozwiązywania problemów analizy przepływu danych, wierzchołki grafu są przetwarzane bądź "w przód" - tj. poczynając od wierzchołka początkowego, idąc następnie do dalszych wierzchołków zgodnie z przepływem sterowania w programie, bądź "w tył" - tj. poczynając od wierzchołka (lub wierzchołków), w którym wykonanie programu się kończy, idąc następnie do wierzchołka początkowego w kierunku przeciwnym do przepływu sterowania. Przyjmując za podstawę kolejność przetwarzania wierzchołków grafu, wyróżnia się problemy analizy przepływu danych klasy 1 (analiza "w przód") oraz klasy 2 (analiza "w tył"). Problem definicji osiągających należy do klasy 1, zaś problem zmiennych aktywnych - do klasy 2.

W zależności od tego, dla jakiego fragmentu programu przeprowadzana jest analiza przepływu danych, można wyróżnić trzy "poziomy" takiej analizy (Hecht [1977], Aho i Ullman [1977]), a mianowicie "poziom":

a) bloku podstawowego (analiza wewnątrzblokowa),
b) procedury (analiza wewnątrzproceduralna) oraz
c) programu (analiza międzyproceduralna).

Analiza wewnątrzblokowa jest także określana mianem lokalnej, podczas gdy wewnątrz- oraz międzyproceduralne analizy są określane mianem globalnych.

Jak nazwa wskazuje, analiza wewnątrzblokowa przeprowadzana jest dla bloku podstawowego, tj. sekwencji instrukcji nie zawierającej skoków. Jest ona stosunkowo łatwa do wykonania. Metody realizacji takiej analizy przedstawione są, np. w Cocke i Schwartz [1970], Aho i Ullman [1973].

Analiza wewnątrzproceduralna dotyczy fragmentu programu (w szczególności procedury) nie zawierającego wywołań innych procedur. Ponieważ w dalszej części rozprawy będziemy zajmować się wyłącznie analizą wewnątrzproceduralną, stan badań w tej dziedzinie omówimy za chwilę bardziej szczegółowo.

Przeprowadzając analizę międzyproceduralną, nie nakładamy na program będący przedmiotem analizy żadnych ograniczeń. Mogą więc w jego skład wchodzić procedury wywoływane bądź w części głównej programu, bądź z wnętrza innych procedur. Dopuszcza się również wywołania rekursywne. (Warto zaznaczyć, że podstawą analizy międzyproceduralnej, uwzględniającej wywołania procedur, są metody analizy wewnątrzproceduralnej). Studia nad analizą międzyproceduralną zostały zapoczątkowane przez Allena [1974], a następnie były kontynuowane przez kilku autorów, np. Rosen [1975a,b 1976], Hecht i Shaffer [1976], Lomet [1977], Barth [1978], Morel i Renvoise: w Muchnick i Jones [1981].

Omówimy obecnie w sposób syntetyczny stan badań nad efektywnymi algorytmami przeprowadzania analizy wewnątrzproceduralnej oraz przedstawimy w skrócie rezultaty uzyskane w niniejszej rozprawie.

Do najprostszych algorytmów rozwiązywania problemów wewnątrzproceduralnej analizy przepływu danych należą algorytmy iteracyjne. Ich ogólna zasada polega na cyklicznym wizytowaniu wierzchołków grafu przepływu oraz uzyskiwaniu coraz lepszych przybliżeń rozwiązań danego problemu. Prezentację oraz analizę algorytmów iteracyjnych można znaleźć m.in. w następujących pracach: Ullman [1973], Hecht i Ullman [1975], Kennedy [1976]. (W pkt. 4.2 niniejszej rozprawy przedstawiono algorytm iteracyjny rozwiązywania problemu zmiennych aktywnych). Pesymistyczna złożoność czasowa algorytmów iteracyjnych jest $O(n^2)$, gdzie n oznacza liczbę wierzchołków grafu przepływu. Podstawową zaletą algorytmów jest ich prostota (łatwość implementacji), natomiast za wadę można uznać fakt, że w algorytmach tych nie korzysta się z jakiejkolwiek reprezentacji struktury przepływu sterowania w programie, która zwykle jest pomocna przy stosowaniu wyników analizy przepływu danych, np. dla celów optymalizacji.

Do algorytmów, które uwzględniają strukturę przepływu sterowania w programie należy algorytm analizy interwałów podany przez Allena i Cocke'a (Allen [1970, 1971], Cocke [1970], Allen i Cocke [1976]). (W pkt. 3.2 rozprawy przedstawiono algorytm analizy interwałów dla problemu definicji osiągających). W algorytmie istotną rolę odgrywa interwał, zdefiniowany jako podgraf grafu przepływu o szczególnych własnościach. Pozwala on dzielić graf przepływu na fragmenty, dla których rozwiązanie problemu analizy przepływu danych jest prostsze, niż dla całego grafu. Wyniki rozwiązań częściowych są następnie scalane. Algorytm analizy interwałów może być stosowany tylko dla grafów redukowalnych (tj. takich, które można sprowadzić do pojedynczego wierzchołka za pomocą odpowiednich transformacji). Badania wykazały (Knuth [1971]), że ponad 95% programów spotykanych w praktyce jest redukowalnych. Pesymistyczna złożoność czasowa algorytmu analizy interwałów jest $O(n^2)$. Allen i Cocke jako pierwsi zastosowali algorytm do rozwiązania problemów klasy 1, natomiast Kennedy zastosował go do rozwiązania problemów klasy 2 (Kennedy [1971, 1976]).

Dla celów teoretycznych oraz w poszukiwaniu szybszych algorytmów, Ullman wprowadził dwie transformacje, T1 i T2, grafów przepływu programów (Ullman [1973]). Transformacja T1 przekształca cykl trywialny grafu w pojedynczy wierzchołek, zaś transformacja T2 pozwala zastąpić sekwencję dwóch wierzchołków pojedynczym wierzchołkiem w przypadku, gdy pierwszy wierzchołek sekwencji jest jedynym poprzednikiem drugiego z nich. Korzystając z transformacji T1 i T2 oraz zrównoważonych drzew "2-3", Ullman przedstawił algorytm rozwiązywania problemów klasy 1 o złożoności pesymistycznej O(nlogn) (Ullman [1973]). Jak dotychczas nie wiadomo czy algorytm ten może być zaadaptowany do rozwiązywania problemów klasy 2.

Jednym z rodzajów algorytmów iteracyjnych jest algorytm oparty na pojęciu listy wierzchołków, podany przez Kennedy'ego (Kennedy [1975]). Lista wierzchołków zawiera wszystkie wierzchołki grafu przepływu w takim uporządkowaniu, że jednokrotne jej przetworzenie zapewnia uzyskanie rozwiązania danego problemu analizy przepływu danych. Podstawową trudnością jest znalezienie takiego uporządkowania. Aho i Ullman pokazali że dla redukowalnego grafu przepływu można znaleźć listę wierzchołków o długości O(nlogn) w czasie O(nlogn) (Aho i Ullman [1976]). Algorytm analizy przepływu danych korzystający z listy wierzchołków, o złożoności O(nlogn), może być stosowany zarówno dla problemów klasy 1, jak i klasy 2. Wadą algorytmu jest konieczność rozwiązywania dwóch problemów, tj. problemu znalezienia listy wierzchołków, a następnie właściwego problemu analizy. Algorytm nie korzysta również z reprezentacji struktury przepływu sterowania w programie.

Graham i Wegman przedstawili algorytm o złożoności O(nlogn), w którym za pomocą transformacji T_1, T_2, T_3 (podobnych do transformacji T1 i T2 Ullmana) dokonuje się kompresji dróg grafu przepływu, co pozwala zredukować go do pojedynczego wierzchołka (Graham i Wegman [1976]). W czasie redukcji rozwiązuje się zadany problem analizy przepływu danych(klasy 1 lub 2) dla poszczególnych fragmentów grafu. Algorytm cechuje się złożonością liniową dla szerokiej podklasy redukowalnych grafów przepływu. Chociaż w algorytmie korzysta się z pewnej struktury przepływu sterowania w programie, to jest ona jednak niewygodna dla zastosowania wyników analizy, ponieważ przy jej konstruowaniu brane są pod uwagę przede wszystkim krawędzie grafu, a nie wierzchołki.

W niniejszej rozprawie przedstawiono oryginalny algorytm, nazwany algorytmem analizy regionów, przeznaczony dla przeprowadzania wewnątrzproceduralnej analizy przepływu danych. Za pomocą algorytmu można rozwiązywać zarówno problemy klasy 1 jak i klasy 2, dla programów o redukowalnych grafach przepływu. Pokazano, że pesymistyczna złożoność algorytmu jest $O(n^2)$. Dla kilku konkretnych grafów przepływu dokonano porównań algorytmu pod względem złożoności z dwoma, znanymi algorytmami o tej samej złożoności pesymistycznej, a mianowicie: z algorytmem iteracyjnym oraz algorytmem analizy interwałów. Porównania wykazały, że w zależności od rodzaju grafu przepływu, przewagę (tj. mniejszą liczbę operacji elementarnych niezbędnych dla rozwiązania problemu) wykazywał bądź algorytm analizy regionów, bądź algorytm użyty do porównania. Wydaje się, że proponowany algorytm jest łatwiejszy do zaimplementowania, niż algorytm analizy interwałów, natomiast trudniejszy w stosunku do algorytmu iteracyjnego. W algorytmie analizy regionów korzysta się z grafowej reprezentacji przepływu sterowania w programie, w szczególności z reprezentacji jego podgrafów zwanych regionami.

Rozprawa składa się z pięciu rozdziałów oraz dwóch dodatków. W rozdziale 1 omówiono w sposób ogólny problemy analizy przepływu danych będące przedmiotem rozważań, podano przykłady zastosowań informacji uzyskiwanych w wyniku rozwiązania tych problemów, przedstawiono w sposób syntetyczny stan badań nad efektywnymi algorytmami przeprowadzania analizy wewnątrzproceduralnej oraz omówiono w skrócie rezultaty uzyskane w przedstawionej rozprawie. Rozdział 2 zawiera definicje pojęć podstawowych niezbędnych dla prezentacji algorytmów analizy przepływu danych oraz oceny ich złożoności. Rozdział 3, poświęcony problemowi definicji osiągających, podzielony jest na cztery punkty. Punkt 3.1 zawiera sformułowanie problemu. W punkcie 3.2 przedstawiono algorytm analizy interwałów (dla problemu definicji osiągających) oraz wyznaczono jego złożoność. W punkcie 3.3 zaprezentowano algorytm analizy regionów, udowodniono jego poprawność oraz określono jego złożoność pesymistyczną. W punkcie 3.4 wspomniane wyżej algorytmy zostały porównane w oparciu o rodzinę samopowielających się redukowalnych grafów przepływu. Konstrukcja rozdziału 4, w którym dyskutowany jest problem zmiennych aktywnych, jest podobna do konstrukcji rozdziału 3, z tą różnicą, że algorytm analizy regionów został porównany z algorytmem iteracyjnym w wersji "round-robin". Rozdział 5 stanowi zakończenie rozprawy. W Dodatku A przedstawiono uzupełniającą dyskusję dotyczącą wyznaczania złożoności pesymistycznej algorytmu. Dodatek B zawiera wyniki weryfikacji algorytmu analizy regionów uzyskane po jego zaimplementowaniu w języku Ada.

Chciałbym w tym miejscu serdecznie podziękować Prof. Zdzisławowi Pawlakowi za uwagi, które pozwoliły ulepszyć tekst rozprawy. Jestem wdzięczny Prof. Stefanowi Węgrzynowi za możliwość przeprowadzenia części niniejszych badań w ramach problemu węzłowego, którego jest koordynatorem. Dziękuję władzom Wydziału Informatyki Uniwersytetu York, z Dr Ian C. Wand na czele, za umożliwienie mi dokonania implementacji algorytmu w systemie VAX - 11/780 UNIX. Składam podziękowania Dr Mark R. Manning oraz John A. Murdie, współautorom kompilatora York Ada, za wprowadzenie mnie w tajniki jego konstrukcji oraz użytkowania. Dziękuję również Urszuli Biadacz za staranne przygotowanie maszynopisu niniejszej rozprawy.

2. POJĘCIA PODSTAWOWE

Niniejszy rozdział zawiera omówienie pojęć niezbędnych dla przedstawienia algorytmów analizy przepływu danych oraz oceny ich złożoności. W pkt. 2.1 zaprezentowano pojęcia ogólne dotyczące grafowego modelu, jaki używany jest przy analizie przepływu sterowania oraz danych w programach. W pkt. 2.2 zebrano pojęcia, które głównie będą stosowane przy prezentacji algorytmu analizy interwałów. Pkt. 2.3 poświęcony jest pojęciom związanym z algorytmem analizy regionów, zaś w pkt. 2.4 omówiono pojęcia potrzebne do analizy algorytmu iteracyjnego w wersji "round-robin".

2.1. Graf przepływu

Przez blok podstawowy programu rozumiemy maksymalną, liniową sekwencję instrukcji, mającą jeden punkt wejściowy (pierwsza wykonywana instrukcja) oraz jeden punkt wyjściowy (ostatnia wykonywana instrukcja). Blok podstawowy może więc składać się jedynie z instrukcji bezwarunkowych, tj. nie zawierających warunków powodujących przejścia do różnych punktów programu.

Grafem przepływu sterowania programu (lub grafem przepływu) nazywamy trójkę $G = (N, \Gamma, s)$, gdzie:

1. N jest skończonym zbiorem wierzchołków, odpowiadających blokom podstawowym programu.
2. $\Gamma : N \to 2^N$ jest funkcją bezpośrednich następników, która przyporządkowuje każdemu wierzchołkowi $x \in N$, podzbiór wierzchołków z N będących bezpośrednimi następnikami x.
3. Wierzchołek $s \in N$ jest wierzchołkiem początkowym, od którego rozpoczyna się każde wykonanie programu.

Przykład 2.1

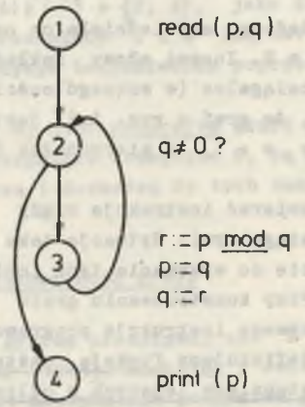
Na rys. 2.1(a) przedstawiono program wyznaczania największego wspólnego dzielnika liczb p i q. W programie tym można wyodrębnić cztery bloki podstawowe, na które składają się następujące instrukcje:

blok 1 - read(p, q)
blok 2 - badanie, czy $q \neq 0$?
blok 3 - r := p mod q
p := q
q := r
blok 4 - print(p).

(a)

```
begin
  --p i q, gdzie p>0 i q≥0, są zmiennymi całkowitymi
  read (p,q);
  while q≠0 do
      r:=p mod q;
      p:=q;
      q:=r
  endwhile;
  print (p)
end
```

(b)



Rys. 2.1. Program wyznaczania największego wspólnego dzielnika liczb p i q (a) oraz jego graf przepływu (b). (Wierzchołki grafu przepływu są ponumerowane w dowolny sposób. Drugi i trzeci wiersz programu zawierają komentarze. Będziemy przyjmować, że komentarz w programach rozpoczyna się dwoma znakami myślnika (niekoniecznie umieszczonymi na początku wiersza) oraz kończy się wraz z końcem wiersza)

Rys. 2.1(b) przedstawia graf przepływu sterowania $G = (N, \Gamma, s)$ tego programu, gdzie N, Γ i s określone są następująco: $N = \{1, 2, 3, 4\}$; $\Gamma 1 = \{2\}$, $\Gamma 2 = \{3, 4\}$, $\Gamma 3 = \{2\}$, $\Gamma 4 = \emptyset$; $s = 1$. (Dla uproszczenia, wartość funkcji Γ dla argumentu x będziemy oznaczać Γx, a nie jak tradycyjnie $\Gamma(x)$).□

Uporządkowana para wierzchołków (x, y) taka, że $y \in \Gamma x$ nazywana jest krawędzią grafu przepływu. Mówimy, że krawędź (x, y) dochodzi do wierz-

chołka y i wychodzi z wierzchołka x oraz, że x jest bezpośrednim poprzednikiem y, a y jest bezpośrednim następnikiem x. Nieformalnie, jeżeli w G istnieje krawędź (x,y), to po wykonaniu bloku reprezentowanego przez x, jest możliwym, że sterowanie zostanie przekazane do bloku reprezentowanego przez y.

Zauważmy, że graf przepływu programu jest grafem skierowanym, ponieważ każda jego krawędź ma określony kierunek.

Przez drogę od x_1 do x_k rozumiemy sekwencję wierzchołków $(x_1, x_2, \ldots, x_k)$ gdzie każdy x_i jest bezpośrednim poprzednikiem x_{i+1}, dla $1 \leqslant i \leqslant k-1$. Długością drogi nazywamy liczbę krawędzi występujących na tej drodze, tak więc droga $(x_1, x_2, \ldots, x_k)$ ma długość $k-1$. Cykl lub pętle[1] zdefiniujemy jako drogę $(x_1, x_2, \ldots, x_k)$, gdzie $x_1 = x_k$, zaś cykl trywialny - jako cykl o długości 1.

Przykład 2.2

Graf z rys. 2.1(b) zawiera cykl (2, 3, 2) o długości 2, natomiast nie zawiera cykli trywialnych.□

Definiując graf przepływu żądamy, aby istniała co najmniej jedna droga od s do każdego wierzchołka w N. Innymi słowy zakładamy, że w grafie nie występują wierzchołki nieosiągalne (w szczególności izolowane).

Przykładowo, gdyby przyjąć, że graf z rys. 1.2 jest grafem przepływu (a nie jego fragmentem), gdzie s = 1, to wierzchołek 2 byłby nieosiągalny.

Program komputerowy może zawierać instrukcje nigdy niewykonywane (odpowiadające wierzchołkom nieosiągalnym). Sytuacja taka występuje wtedy, gdy warunek powodujący przejście do wykonania tych instrukcji nigdy nie jest spełniony (np. 1 > 2). Przy konstruowaniu grafu przepływu, wierzchołki reprezentujące niewykonywane instrukcje programu zostają pominięte.

Korzystając z funkcji Γ zdefiniujemy funkcję następników leżących w odległości 2 - Γ^2, funkcję następników leżących w odległości 3 - Γ^3, ..., funkcję następników - Γ^* oraz funkcję bezpośrednich poprzedników - Γ^{-1}, w następujący sposób:

$$\Gamma^2 x = \Gamma(\Gamma x)$$
$$\Gamma^3 x = \Gamma(\Gamma^2 x) = \Gamma(\Gamma(\Gamma x))$$
$$\ldots$$
$$\Gamma^* x = \{x\} \cup \Gamma x \cup \Gamma^2 x \cup \ldots$$
$$\Gamma^{-1} x = \{y \mid x \in \Gamma y\}$$

Dla wszystkich zdefiniowanych w ten sposób funkcji możemy napisać

$$\Gamma^{\alpha}: N \to 2^N, \quad \text{gdzie } \alpha \in \{2, 3, \ldots, *, -1\}.$$

[1] W teorii grafów pętlę definiuje się jako cykl trywialny. W niniejszej rozprawie pętlę zdefiniowano jako cykl, ponieważ, jak się wydaje, taka definicja jest bliższa powszechnemu rozumieniu pętli w informatyce.

Z podanych definicji wynika, że funkcja Γ^2 wyznacza dla każdego wierzchołka x grafu przepływu podzbiór wierzchołków tego grafu, do których dochodzą drogi o długości 2 wychodzące z x. Podobnie jest dla funkcji Γ^3, z tym że długość dróg prowadzących od wierzchołka x do wierzchołków mu przyporządkowanych wynosi 3. Łatwo widać, że funkcja Γ^* przyporządkowuje każdemu wierzchołkowi x grafu przepływu, podzbiór wierzchołków tego grafu, do których dochodzą drogi wychodzące z x (są to wierzchołki które można "osiągnąć" z x drogami o dowolnych długościach). Wreszcie funkcja Γ^{-1} wyznacza dla każdego wierzchołka x grafu przepływu, podzbiór wierzchołków tego grafu będących bezpośrednimi poprzednikami x.

Przykład 2.3

Wyznaczmy przykładowo wartości funkcji $\Gamma^2 1$, $\Gamma^3 2$, $\Gamma^* 3$ i $\Gamma^{-1} 4$ dla grafu przepływu z rys. 2.1(b). Mamy: $\Gamma^2 1 = \{3, 4\}$, ponieważ z wierzchołka 1 prowadzą drogi o długości 2 tylko do wierzchołków 3 i 4 (są to drogi (1, 2, 3) oraz (1, 2, 4)); $\Gamma^3 2 = \{3, 4\}$, ponieważ istnieją drogi (2, 3, 2, 3) oraz (2, 3, 2, 4); $\Gamma^* 3 = \{2, 4\}$, jako że z wierzchołka 3 prowadzą drogi tylko do wierzchołków 2 i 4 (są to: (3, 2) oraz (3, 2, 4)); $\Gamma^{-1} 4 = \{2\}$, ponieważ jedynym bezpośrednim poprzednikiem wierzchołka 4 jest wierzchołek 2.□

Mówimy, że graf G_1 jest podgrafem grafu przepływu G, jeżeli wszystkie wierzchołki oraz wszystkie krawędzie G_1 są zawarte w G oraz wszystkie krawędzie w G_1 wychodzą i dochodzą do tych samych wierzchołków co w G.

2.2. Interwał. Redukowalność grafu

Niech G będzie grafem przepływu, zaś h wierzchołkiem w G. Interwał o głowie h (Schaefer [1973], Muchnick i Jones [1981], oznaczony przez I(h) (lub krótko I) jest maksymalnym podgrafem grafu G o własnościach:

1. $h \in I$.
2. Jeśli $x \in I$, to $x \in \Gamma^* h$.
3. Podgraf $I - \{h\}$ nie zawiera cykli.
4. Jeśli $x \in I - \{h\}$, to $\Gamma^{-1} x \subset I$.

Z definicji wynika, że interwał I(h) jest maksymalnym podgrafem o jednym wejściu, będącym głową h oraz mającym tę własność, że wszystkie cykle w nim występujące zawierają h.

Do wyodrębnienia interwału w grafie przepływu $G = (N, \Gamma, s)$, przy zadanym wierzchołku h, można użyć następującego algorytmu (Allen i Cocce [1976]):

Algorytm 2.1: Wyodrębnianie interwału w grafie przepływu.
Dane: Graf przepływu G z zadanym wierzchołkiem h.
Wyniki: Interwał I(h).

Metoda:

begin

Przyjmij I(h) równy $\{h\}$,

while m jest wierzchołkiem jeszcze nie zawartym w I(h) and

$m \neq s$ and

wszystkie poprzedniki m są zawarte w I(h) do

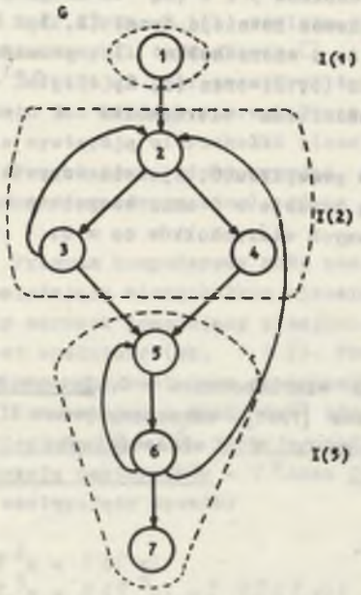
Dołącz m do I(h)

endwhile

end □

Przykład 2.4

Graf przepływu z rys. 2.2 ma trzy interwały: $I(1) = \{1\}$, $I(2) = \{2,3,4\}$ oraz $I(5) = \{5,6,7\}$.□



Rys. 2.2. Graf przepływu G oraz jego interwały: $I(1) = \{1\}$, $I(2) = \{2, 3, 4\}$ oraz $I(5) = \{5, 6, 7\}$

Zauważmy, że kolejność w jakiej poszczególne wierzchołki są dołączane do interwału I(h) w czasie wykonywania Algorytmu 2.1, jest istotna. Kolejność ta jest określana mianem porządku interwałowego. Porządek interwałowy nie jest jedyny, zależy on bowiem od tego, jak wybierane są kolejne wierzchołki m dołączane do interwału.

Przykład 2.5

Interwał I(2) grafu przepływu z rys. 2.2 ma dwa porządki interwałowe a mianowicie: (2, 3, 4) i (2, 4, 3). Rzeczywiście, w trakcie realizacji Algorytmu 2.1, po przyjęciu I(2) równego $\{2\}$, jako następny może być dołączony do interwału I(2) zarówno wierzchołek 3, jak i 4.□

Należy dodać, że Algorytm 2.1 wyznaczając jeden z porządków interwałowych, zachowuje zawsze częściowy porządek zdefiniowany przez relację następstwa wierzchołków w grafie (Hecht [1977]).

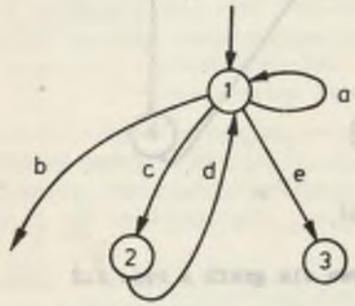
Rozważmy interwał I(h) w grafie przepływu G. Krawędzie wychodzące z wierzchołków I(h), jak również wierzchołki I(h), będziemy klasyfikować na kilka sposobów.

Niech E_I będzie zbiorem krawędzi wychodzących z wierzchołków I(h), E_I^b - zbiorem krawędzi wychodzących z wierzchołków I(h) i dochodzących do h (krawędzie zamykające I(h)), E_I^f - zbiorem krawędzi wychodzących z wierzchołków I(h) i dochodzących do wierzchołków z I(h) - $\{h\}$ (krawędzie w przód interwału I(h)), E_I^o - zbiorem krawędzi wychodzących z wierzchołków I(h) i dochodzących do wierzchołków leżących na zewnątrz interwału (krawędzie wyjściowe I(h)), E_I^ϕ - zbiorem krawędzi wychodzących z wierzchołków I(h) i dochodzących do wierzchołków I(h) nie posiadających następników, E_I^{b1} - zbiorem krawędzi wychodzących z wierzchołków I(h) i dochodzących do wierzchołków I(h), których wszystkie krawędzie wychodzące są krawędziami zamykającymi I(h).

Niech N_I będzie zbiorem wierzchołków I(h), N_I^f - zbiorem wierzchołków w I, z których wychodzi co najmniej jedna krawędź w przód, N_I^ϕ - zbiorem wierzchołków w I nie posiadających następników, N_I^{b1} - zbiorem wierzchołków w I, z których wychodzą jedynie krawędzie zamykające I.

Przykład 2.6

Zdefiniowane w ten sposób zbiory, dla interwału I z rys. 2.3, są następujące: $E_I = \{a, b, c, d, e\}$, $E_I^h = \{a, b, c, e\}$, $E_I^b = \{a, d\}$, $E_I^f = \{c, e\}$, $E_I^o = \{b\}$, $E_I^\phi = \{e\}$, $E_I^{b1} = \{c\}$, $N_I = \{1, 2, 3\}$, $N_I^f = \{1\}$, $N_I^\phi = \{3\}$ oraz $N_I^{b1} = \{2\}$.□



Rys. 2.3. Interwał $I(1) = \{1, 2, 3\}$

Przyjmijmy, że liczby krawędzi (wierzchołków) w zbiorach E_I, E_I^h, ..., E_I^{b1} (N_I, N_I^f, ..., N_I^{b1}) są odpowiednio równe e_I, e_I^h, ..., e_I^{b1} (n_I, n_I^f, ..., n_I^{b1}).

Można wykazać, że każdy graf przepływu da się w sposób jednoznaczny podzielić na rozłączne interwały (Allen i Cocke [1976], Hecht [1977]).

Jeżeli G jest grafem przepływu, to pochodny graf przepływu dla G (Allen [1970], Hecht i Ullman [1975]), oznaczony I(G)[1], jest zdefiniowany następująco:

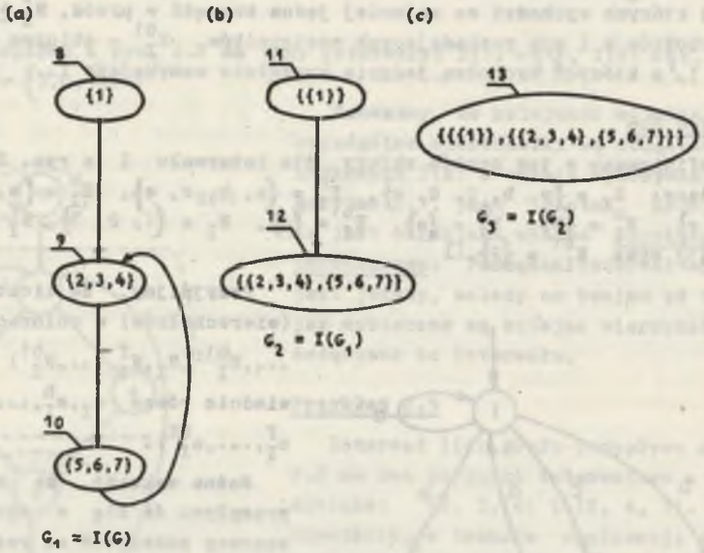
1. Wierzchołki I(G) są interwałami uzyskanymi z (jednoznacznego) podziału G.
2. Jeśli J, K są interwałami, to $K \in \Gamma_I J$ wtedy i tylko wtedy, gdy istnieją wierzchołki $n_J \in J$ oraz $n_K \in K$ takie, że $n_K \in \Gamma n_J$. Zauważmy, że n_K musi być głową K.
3. Początkowy wierzchołek I(G) jest równy I(s).

[1] Zwróćmy uwagę, że używamy identycznej notacji dla oznaczenia grafu pochodnego, I(G), oraz interwału, np. o głowie x, I(x).

Jak wynika z definicji, pochodny graf przepływu dla G otrzymujemy dzieląc graf G na interwały i traktując każdy z nich jako pojedynczy wierzchołek.

Przykład 2.7

Rys. 2.4(a) przedstawia pochodny graf przepływu dla grafu G z rys. 2.2.□



Rys. 2.4. Pochodne grafy przepływu dla grafu z rys. 2.2

Mówimy, że sekwencja $(G_0, G_1, \ldots, G_m)$ jest sekwencją grafów pochodnych dla G, jeśli $G = G_0, G_{i+1} = I(G_i), G_{m-1} \neq G_m$ oraz $I(G_m) = G_m$. Graf G_i zwany jest grafem pochodnym rzędu i, zaś G_m - grafem granicznym.

Przykład 2.8

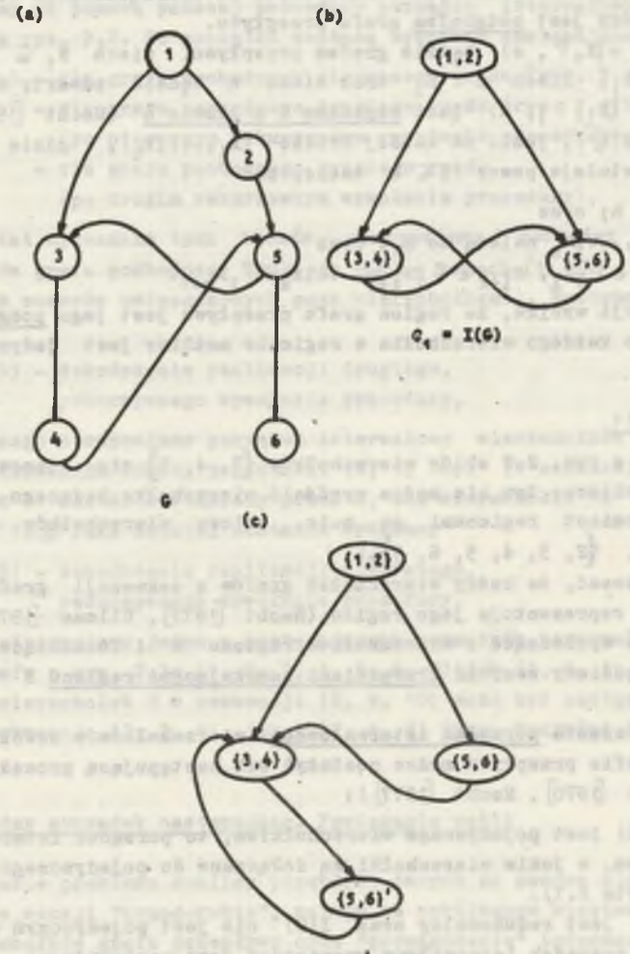
Sekwencja grafów (G, G_1, G_2, G_3) przedstawionych na rys. 2.2 i 2.4 jest sekwencją grafów pochodnych dla G. Graf G_3 jest grafem granicznym, ponieważ $I(G_3) = G_3$.□

Mówimy, że graf przepływu jest redukowalny (ang. reducible flow graph) wtedy i tylko wtedy, gdy jego graf graniczny jest grafem trywialnym, tj. pojedynczym wierzchołkiem bez krawędzi; w przeciwnym razie graf jest nieredukowalny.

Przykład 2.9

Graf G z rys. 2.2 jest redukowalny, ponieważ jego graf graniczny G_3 (rys. 2.4(c)) składa się z pojedynczego wierzchołka. Jest to więc graf trywialny. Na rys. 2.5(a) przedstawiono graf nieredukowalny, bowiem jego graf graniczny G_1 (rys. 2.5(b)) nie jest grafem trywialnym.□

Istnieje metoda, zwana metodą rozszczepiania wierzchołków (ang. node-splitting), za pomocą której można każdy nieredukowalny graf przepływu przekształcić w redukowalny (Aho i Ullman [1973], Schaefer [1973]).



Rys. 2.5. Nieredukowalny graf przepływu (a), jego graf graniczny (b) oraz przykład zastosowania metody rozszczepiania wierzchołków (c)

Przykład 2.10

Rozszczepiając wierzchołek oznaczony $\{5, 6\}$ grafu G_1 (rys. 2.5(b)), uzyskujemy równoważny graf G_1 (rys. 2.5(c)), który jest redukowalny.□

2.3. Region

Podamy obecnie definicję regionu oraz rekursywną procedurę wyznaczania porządku interwałowego wierzchołków w dowolnym, redukowalnym grafie przepływu. Procedura ta może być również użyta do uporządkowania wierzchołków regionu, który jest podgrafem grafu przepływu.

Niech $G = (N, \Gamma, s)$ będzie grafem przepływu, niech $N_1 \subseteq N$, niech $\Gamma_1 x = \Gamma x \cap N_1$, niech $x \in N_1$ oraz niech h będzie zawarty w N_1. Mówimy, że $R = (N_1, \Gamma_1, h)$ jest regionem G o głowie h (Hecht [1977], Hecht i Ullman [1974]), jeśli na każdej drodze $(x_1, \ldots, x_k)$, gdzie $x_1 = s$ i $x_k \in N_1$, istnieje pewne $i \leqslant k$ takie, że:

(a) $x_i = h$; oraz
(b) $x_{i+1}, \ldots, x_k$ należą do N_1; oraz
(c) $x_{i+1} \in \Gamma_1 x_i$, $x_{i+2} \in \Gamma_1 x_{i+1}, \ldots, x_k \in \Gamma_1 x_{k-1}$.

Z definicji wynika, że region grafu przepływu jest jego podgrafem oraz że dostęp do każdego wierzchołka w regionie możliwy jest jedynie poprzez głowę.

Przykład 2.11

W grafie z rys. 2.2 zbiór wierzchołków $\{3, 4, 5\}$ nie stanowi regionu, ponieważ w zbiorze tym nie można wyróżnić wierzchołka będącego głową regionu. Natomiast regionami są m.in. zbiory wierzchołków $\{2, 3, 4\}$, $\{1, 2, 3, 4\}$, $\{2, 3, 4, 5, 6, 7\}$.□

Można wykazać, że każdy wierzchołek grafów z sekwencji grafów pochodnych dla G, reprezentuje jego region (Hecht [1977], Ullman [1973]).

Krawędzie wychodzące z wierzchołków regionu R i dochodzące do jego głowy h będziemy nazywać krawędziami zamykającymi regionu R (ang. latches).

Dla znalezienia porządku interwałowego wierzchołków w dowolnym, redukowalnym grafie przepływu można posłużyć się następującą procedurą rekursywną (Allen [1970], Hecht [1977]):

1. Jeśli I(G) jest pojedynczym wierzchołkiem, to porządek interwałowy jest porządkiem, w jakim wierzchołki są dołączane do pojedynczego interwału G (Algorytm 2.1).
2. Jeśli G jest redukowalny oraz I(G) nie jest pojedynczym wierzchołkiem, to porządek interwałowy wyznaczany jest następująco:

(a) Znaleźć porządek interwałowy dla I(G).
(b) W porządku (a) zastąpić każdy wierzchołek I(G) wierzchołkami grafu G składającymi się na odpowiedni interwał przyjmując, że każdy z interwałów jest redukowalnym grafem przepływu o wierzchołku początkowym będącym głową interwału.

Należy zaznaczyć, że podobnie jak porządek interwałowy wierzchołków interwału, również porządek interwałowy wierzchołków dowolnego, redukowalnego grafu przepływu nie jest jedyny.

Przykład 2.12

Wyznaczmy za pomocą podanej procedury porządek interwałowy wierzchołków grafu z rys. 2.2. Na początku zostaną wykonane następujące jej kroki:

krok 2(a) - dla grafu pochodnego pierwszego rzędu (rys. 2.4(a)),
krok 2(a) - dla grafu pochodnego drugiego rzędu (rys. 2.4(b)) (po pierwszym rekursywnym wywołaniu procedury),
krok 1 - dla grafu pochodnego drugiego rzędu (po drugim rekursywnym wywołaniu procedury).

Jako rezultat wykonania tych kroków otrzymujemy porządek interwałowy wierzchołków grafu pochodnego drugiego rzędu w postaci: (11, 12). (Używamy tu dodatkowych numerów umieszczonych poza wierzchołkami). Następnie zostanie wykonany

krok 2(b) - dokończenie realizacji drugiego, rekursywnego wywołania procedury,

w wyniku czego otrzymujemy porządek interwałowy wierzchołków grafu pochodnego pierwszego rzędu, o postaci: (8, 9, 10). (W sekwencji (11, 12), wierzchołek 11 został zastąpiony przez 8, zaś wierzchołek 12 przez sekwencję (9, 10)). Jako ostatni zostanie wykonany

krok 2(b) - dokończenie realizacji pierwszego, rekursywnego wywołania procedury,

po którym otrzymujemy jeden z następujących porządków interwałowych wierzchołków grafu z rys. 2.2: (1, 2, 3, 4, 5, 6, 7) lub (1, 2, 4, 3, 5, 6, 7). Istotnie, wierzchołek 9 w sekwencji (8, 9, 10) może być zastąpiony zarówno przez sekwencję (2, 3, 4), jak i (2, 4, 3) (por. Przykład 2.5).□

2.4. Odwrotny porządek następujący. Powiązanie pętli

Rozwiązanie problemu analizy przepływu danych za pomocą algorytmu iteracyjnego w wersji "round-robin", polega na cyklicznym wizytowaniu kolejnych wierzchołków grafu przepływu oraz "propagowaniu" informacji pomiędzy nimi do momentu, gdy ich przepływ ustabilizuje się. Można wykazać, że minimalną złożoność takiego algorytmu uzyskuje się wtedy, gdy wierzchołki

grafu są odwiedzane w tzw. odwrotnym porządku następującym. W niniejszym punkcie podamy algorytm, który dla zadanego grafu przepływu konstruuje drzewo rozpinające w głąb, wyznaczając jednocześnie odwrotny porządek następujący wierzchołków grafu. Podane w dalszej kolejności definicje krawędzi w tył oraz powiązania pętli, będą pomocne przy określaniu złożoności algorytmu iteracyjnego.

Acyklicznym grafem skierowanym nazywamy graf skierowany nie zawierający cykli (Aho, Hopcroft i Ullman [1976]).

Przez drzewo skierowane, lub w skrócie drzewo (Aho, Hopcroft i Ullman [1976]), rozumiemy acykliczny graf skierowany o następujących własnościach:

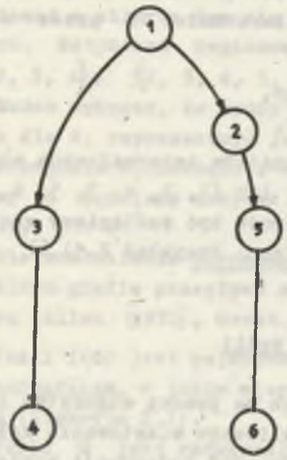
1. W grafie występuje dokładnie jeden wierzchołek, zwany korzeniem, do którego nie dochodzi żadna krawędź.
2. Do każdego wierzchołka w grafie, za wyjątkiem korzenia, dochodzi dokładnie jedna krawędź.
3. Dla każdego wierzchołka w grafie istnieje droga (łatwo wykazać, że jedyna) prowadząca od korzenia do tego wierzchołka.

Drzewem rozpinającym grafu przepływu $G = (N, \Gamma, s)$ nazywamy drzewo o korzeniu s zawierające wszystkie wierzchołki grafu G.

Przykład 2.13

Rys. 2.6 przedstawia drzewo rozpinające grafu przepływu z rys.2.5(a)□

Łatwo wykazać, że w ogólnym przypadku dla zadanego grafu przepływu istnieje więcej niż jedno drzewo rozpinające.



Rys. 2.6. Drzewo rozpinające grafu przepływu z rys. 2.5(a)

Jednym z rodzajów drzew rozpinających jest drzewo rozpinające w głąb (ang. depth-first spanning tree, DFST). Konstruuje się go stosując przeszukiwanie grafu w głąb, które przebiega według następujących zasad. Wybieramy oraz wizytujemy wierzchołek początkowy v. Następnie wybieramy dowolną krawędź (v, w) wychodzącą z v oraz wizytujemy w. Ogólnie, załóżmy że x jest ostatnim wizytowanym wierzchołkiem. Przeszukiwanie jest kontynuowane przez wybór nierozpatrywanej jeszcze krawędzi (x, y) wychodzącej z x. Jeśli y był już uprzednio wizytowany, to wybieramy inną, nową krawędź wychodzącą z x. Jeśli wierzchołek y nie był jeszcze wizytowany, wówczas wizytujemy go, a następnie rozpoczynamy przeszukiwanie od nowa traktując y jako wierzchołek początkowy. Po przeszukaniu wszystkich dróg wychodzących z y, wracamy do

wierzchołka x, tj. do wierzchołka, z którego po raz pierwszy osiągnęliśmy y. Kontynuujemy proces wybierania nie rozpatrywanych jeszcze krawędzi wychodzących z x do momentu, gdy ich lista się wyczerpie. Omówiony sposób wizytowania wierzchołków grafu zwany jest przeszukiwaniem w głąb, ponieważ dokonując tego przeszukiwania idziemy "w głąb" grafu tak długo jak to jest możliwe.

Podamy obecnie algorytm, który dla zadanego grafu przepływu znajduje drzewo rozpinające w głąb. Algorytm ten wyznacza również porządek wierzchołków grafu, zwany odwrotnym porządkiem następującym (ang. rPostorder).

Algorytm 2.2: Znajdowanie drzewa rozpinającego w głąb oraz odwrotnego porządku następującego wierzchołków grafu.

Dane: Graf przepływu $G = (N, \Gamma, s)$, o wierzchołkach ponumerowanych od 1 do n w dowolny sposób, reprezentowany przez listy następników podane dla poszczególnych wierzchołków.

Wyniki: 1. Drzewo DFST dla grafu G.
2. Odwrotny porządek następujący wierzchołków od 1 do n w tablicy rPOSTORDER, będący odwrotnym porządkiem wierzchołków osiąganych jako ostatnie podczas przeszukiwania w głąb grafu G.

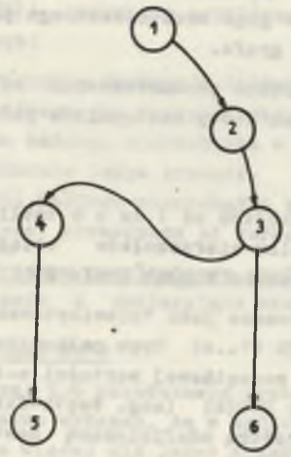
Metoda: Początkowo wszystkie wierzchołki są oznaczone jako "niewizytowane". Używana jest globalna tablica rPOSTORDER (1..n) typu całkowitego oraz globalna zmienna całkowita, i, o początkowej wartości n. Algorytm składa się z wywołania procedury DFS(s) (ang. depth-first search), gdzie DFS jest rekursywną procedurą zdefiniowaną następująco:

```
recursive procedure DFS(x);
  Oznacz x jako "wizytowany";
  while Γx ≠ 0 do
    Wybierz i usuń wierzchołek y z Γx;
    if y jest oznaczony jako "niewizytowany" then
      Dołącz krawędź (x, y) do DFST;
      call DFS(y)
    endif
  endwhile;
  rPOSTORDER(x) := i;
  i := i - 1
return □
```

Podobnie jak dla drzew rozpinających, w ogólnym przypadku można znaleźć dla zadanego grafu przepływu więcej niż jedno drzewo rozpinające w głąb.

Przykład 2.14

Rys. 2.7 przedstawia drzewo rozpinające w głąb grafu przepływu z rys. 2.5(a) uzyskane w wyniku działania Algorytmu 2.2. Wierzchołki drzewa są ponumerowane w odwrotnym porządku następującym. (Warto zwrócić uwagę, że drzewo rozpinające grafu przepływu z rys. 2.5(a) podane w przykładzie 2.13 nie jest drzewem rozpinającym w głąb).□



Rys. 2.7. Drzewo rozpinające w głąb oraz odwrotny porządek następujący wierzchołków grafu przepływu z rys. 2.4(a)

Krawędzią w tył redukowalnego grafu G (Kennedy [1976]) będziemy nazywać krawędź zamykającą dowolnego interwału w sekwencji grafów pochodnych dla G.

Przykład 2.15

Graf z rys. 2.2 ma trzy krawędzie w tył a mianowicie: krawędź (3, 2) jako krawędź zamykającą interwału I(2); krawędź (6,5), jako krawędź zamykającą interwału I(5) oraz krawędź (6, 2), jako krawędź zamykającą interwału $\{\{2, 3, 4\}, \{5, 6, 7\}\}$ w grafie pochodnym G_1 (rys. 2.4(a)).□

Przez powiązanie pętli. d, redukowalnego grafu G (ang. loopconnectedness) będziemy rozumieć maksymalną liczbę krawędzi w tył na dowolnej drodze bez cykli w G (Kam i Ullman [1977], Hecht [1977]).

Przykład 2.16

Dla grafu z rys. 3.5 (patrz rozdz. 3) mamy d = 3. Przyjmując, że wierzchołki grafu są ponumerowane "od góry do dołu" jako 1, 2, 3 i 4, znajdujemy, że na drodze (4, 3, 2, 1) występują 3 krawędzie w tył. Natomiast powiązanie pętli w grafie z rys. 2.2 wynosi 1. Przykładowe drogi w tym grafie, zawierające krawędzie w tył (w liczbie nie przekraczającej 1) i nie zawierające cykli, to: (3, 2), (6, 5), (4, 5, 6, 2).□

3. PROBLEM DEFINICJI OSIĄGAJĄCYCH

3.1. Sformułowanie problemu

Przez definicję zmiennej rozumiemy instrukcję, która może modyfikować wartość zmiennej[1], np. instrukcję przypisania lub czytania danych. Mówimy, że droga w grafie przepływu jest wolna definicyjnie (ang. definition-clear) ze względu na zmienną V lub V-wolna (ang. V-clear), jeśli żadna definicja V nie występuje na tej drodze. Definicja d zmiennej V w wierzchołku x osiąga wejście wierzchołka y, wtedy i tylko wtedy, gdy d występuje w wierzchołku x oraz istnieje V-wolna droga od d do wejścia wierzchołka y.

Rozwiązanie problemu definicji osiągających polega na wyznaczeniu zbioru definicji zmiennych, rdefen(x), które mogą osiągnąć wejście każdego wierzchołka x w grafie przepływu G. Dla danego wierzchołka x w oryginalnym lub pochodnym grafie przepływu, zbiór rdefen(x) (dla definicji zmiennej V) może być wyznaczony za pomocą zbiorów określonych następująco:

(a) gen(x,y), zdefiniowany dla każdego następnika y wierzchołka x, jest zbiorem lokalnie generowanych definicji na wyjściu wierzchołka x prowadzącego do wierzchołka y; tj. zbiór takich definicji w wierzchołku x, dla których istnieje V-wolna droga od d do wyjścia wierzchołka x prowadzącego do wierzchołka y;

(b) trans(x,y), zdefiniowany dla każdego następnika y wierzchołka x, jest zbiorem definicji, które są transmitowane (lub zachowywane) przez wierzchołek x; tj. zbiór definicji, dla których istnieje V-wolna droga od wejścia wierzchołka x do wejścia wierzchołka y.

Uwagi

Ponieważ wierzchołek x w grafie pochodnym reprezentuje interwał który może mieć więcej niż jeden wierzchołek wyjściowy (tzn. wierzchołek interwału o następniku nie zawartym w tym interwale), to zbiory gen i trans są przyporządkowane odpowiednim krawędziom wychodzącym z wierzchołka x. Dla oryginalnego grafu przepływu G występuje zawsze tylko jeden zbiór gen

[1] Będziemy przyjmować, że np. instrukcja przypisania A := 1 wykonywana dla zmiennej A o aktualnej wartości równej 1, jest również definicją zmiennej, mimo, że wartość zmiennej po wykonaniu instrukcji nie ulega w istocie modyfikacji.

i trans dla każdego wierzchołka $x \in N$. Zbiór gen zawiera ostatnią definicję zmiennej V w wierzchołku x lub jest pusty, gdy żadna definicja V nie występuje w wierzchołku x. Zbiór trans zawiera wszystkie definicje zmiennej V występujące w analizowanym programie w przypadku, gdy żadna z nich nie występuje w wierzchołku x; w przeciwnym razie zbiór ten jest pusty. □

Twierdzenie 3.1. Niech $G = (N, \Gamma, s)$ będzie grafem przepływu. Dla każdego $x \in N$

$$\mathrm{rdefen}(x) = \bigcup_{k \in \Gamma^{-1}x} (((\mathrm{rdefen}(k) \cap \mathrm{trans}(k,x)) \cup \mathrm{gen}(k,x)). \qquad (3.1)$$

Dowód. Definicja d zmiennej V osiąga wejście wierzchołka x, jeśli d osiąga wejście jakiegoś poprzednika wierzchołka x, poprzez który istnieje V-wolna droga do wejścia x, tj. $d \in$ rdefen(k) $\cap$ trans(k,x) dla jakiegoś $k \in \Gamma^{-1}x$ lub, jeśli d występuje w jakimś poprzedniku wierzchołka x i leży na V-wolnej drodze prowadzącej do wejścia x, tj. $d \in$ gen(k,x) dla jakiegoś $k \in \Gamma^{-1}x$. Łącząc te możliwości otrzymujemy (3.1). □

Z Twierdzenia 3.1 wynika, że zbiór definicji osiągających wejście dowolnego wierzchołka x w grafie przepływu jest sumą zbiorów definicji osiągających wyjścia wszystkich poprzedników tego wierzchołka. Zbiór definicji osiągających wyjście wierzchołka k (dowolnego poprzednika x) wyznaczamy znajdując przecięcie zbiorów rdefen(k) (definicje osiągające wejście k) oraz trans(k,x) (definicje transmitowane przez k), a następnie dodając do tego przecięcia zbiór gen(k, x) (definicje lokalnie generowane na wyjściu k).

Rozwiązanie problemu definicji osiągających jest w istocie rozwiązaniem n równań (3.1), z n niewiadomymi rdefen, gdzie n jest liczbą wierzchołków w grafie przepływu. Można pokazać, że w ogólnym przypadku rozwiązanie powyższego układu równań nie jest jedyne oraz, że dla problemu definicji osiągających szukamy rozwiązania minimalnego, co oznacza, że wartości rdefen tego rozwiązania są podzbiorami wartości każdego innego rozwiązania (Aho i Ullman [1977]).

W algorytmach, które przedstawimy zostaną również obliczone zbiory rdefen(x,y) definicji osiągających każde wyjście wierzchołka x[1]. Łatwo udowodnić, że dla dowolnych $x \in N$ oraz $y \in \Gamma x$,

$$\mathrm{rdefex}(x,y) = (\mathrm{rdefen}(x) \cap \mathrm{trans}(x,y)) \cup \mathrm{gen}(x,y).$$

[1] W istocie, z praktycznego punktu widzenia, tylko zbiory rdefen są użyteczne.

3.2. Algorytm analizy interwałów

W niniejszym punkcie przedstawimy algorytm analizy interwałów dla problemu definicji osiągających, podany przez Allena i Cocke'a [1976] oraz określimy jego złożoność czasową.

W pozostałej części rozprawy, wszystkie zbiory użyte w algorytmach będą reprezentowane przez wektory bitów (lub krótko wektory), w których każdemu bitowi odpowiada pojedyncza definicja analizowanego programu. W celu odróżnienia zbiorów od ich wektorowych reprezentacji będziemy używać dużych liter dla oznaczenia tych ostatnich. Na przykład RDEFEN(x) będzie oznaczać wektor bitów dla zbioru rdefen(x). TRANS(x,y) będzie oznaczać wektor bitów dla zbioru trans(x,y). itd.

Złożoność algorytmu wyznaczymy poprzez zliczenie operacji elementarnych wykonywanych w trakcie realizacji algorytmu. Za operacje elementarne przyjmiemy operacje boolowskie, takie jak OR, AND i NOT, wykonywane na wektorach bitów reprezentujących zbiory.

Algorytm analizy regionów składa się z dwóch przebiegów.

Przebieg 1

Celem tego przebiegu jest obliczenie, dla każdego interwału w sekwencji grafów pochodnych, zbiorów trans i gen oraz początkowej estymaty zbioru rdefen(1), zawierającego, dla każdej głowy interwału, te definicje interwału, które mogą osiągnąć jego głowę. Podany niżej algorytm oblicza wektorowe reprezentacje wymienionych zbiorów dla poszczególnych wierzchołków interwału.

Algorytm 3.1A: Oblicza dla interwału I wektory bitów RDEFEN(1), TRANS(I, J) oraz GEN(I, J).

Dane:
1. Wierzchołki interwału I ponumerowane od 1 do n_I w porządku interwałowym.
2. Informacje o następnikach i poprzednikach dla każdego wierzchołka interwału.
3. Zbiór interwałów J będących następnikami I (następniki w grafie pochodnym), wraz z ich głowami h_J.
4. TRANS(j,k) oraz GEN(j,k), $1 \leqslant j \leqslant n_I$, k przebiega po wszystkich następnikach j.

Zmienne pomocnicze:

1. TPATH(j,k), $1 \leqslant j \leqslant n_I$, k przebiega po wszystkich następnikach j, które nie są głową I. TPATH będzie wektorem bitów reprezentującym zbiór definicji, które są "transmitowane" wzdłuż jakiejś drogi od wejścia interwału poprzez wierzchołek j do wyjścia z j prowadzącego do k.
2. GPATH(j,k), $1 \leqslant j \leqslant n_I$, k przebiega po wszystkich następnikach j. GPATH będzie wektorem bitów reprezentującym zbiór definicji w interwale, które mogą osiągnąć wyjście j prowadzące do k.

Wyniki: 1. RDEFEN(1) - wektor bitów reprezentujący zbiór definicji, które mogą osiągnąć głowę interwału z wnętrza interwału.

2. TRANS(I,J) oraz GEN(I,J) dla każdego następnika J interwału I w grafie pochodnym[1].

Metoda[2]:

```
begin
  for wszystkich k ∈ Γ1 do
    TPATH(1,k) := TRANS(1,k);
    GPATH(1,k) := GEN(1,k)
  endfor;
  if I jest interwałem pierwszego rzędu then
    for j from 2 to n_I do
      if Γ j ≠ ∅ then
        -- x jest dowolnym wierzchołkiem z Γ j.
        -- T i G są chwilowymi wektorami bitów.
        A1: G := (∨ GPATH(k,j) ∧ TRANS(j,x)) ∨ GEN(j,x);
                k∈Γ^-1 j
        for wszystkich k ∈ Γ j do GPATH(j,k) := G endfor;
        if Γ j - {1} ≠ ∅ then
          A2: T := ∨ TPATH(k,j) ∧ TRANS(j,x);
                k∈Γ^-1 j
          for wszystkich k ∈ Γ j - {1} do
            TPATH(j,k): = T
          endfor
        endif
      endif
    endfor
  else
    -- Dla interwału w pochodnym grafie przepływu.
    for j from 2 to n_I do
      if Γ j ≠ ∅ then
        A3: G := ∨ GPATH(k,j);
                k∈Γ^-1 j
        for wszystkich k∈Γ j do
          A4: GPATH(j,k) := (G∧ TRANS(j,k)) ∨ GEN(j,k)
        endfor;
```

[1] W Allen i Cocke [1976] zbiory odpowiadające wektorom RDEFEN, TRANS GEN TPATH i GPATH są oznaczone odpowiednio przez R, PB, DB, P i D.

[2] W celu zmniejszenia złożoności, sposób przetwarzania interwałów pierwszego rzędu został nieco zmodyfikowany w porównaniu z Allen i Cocke [1976].


```
        if Γ j - {1} ≠ ∅ then
          A5: T := ∨ TPATH(k,j);
                k∈Γ^-1 j
          for wszystkich k ∈ Γ j - {1} do
            A6: TPATH(j,k) := T ∧ TRANS(j,k)
          endfor
        endif
      endif
    endfor
  endif;
  A7: RDEFEN(1) := ∨ GPATH(k,1);  -- Jeśli I nie ma krawędzi
                k ∈ I ∩ Γ^-1 1    -- zamykających, to wówczas
                                  -- RDEFEN(1) := wektor zerowy.
  -- Obliczenie wektorów TRANS i GEN dla I.
  for każdego J ∈ Γ I o głowie h_J do
    A8: TRANS(I,J) := ∨ TPATH(k,h_J);
                k∈Γ^-1 h_J ∩ I
    A9: GEN(I,J) := (RDEFEN(1) ∧ TRANS(I,J)) ∨ ∨ GPATH(k,h_J)
                                              k∈Γ^-1 h_J ∩ I
  endfor
end □
```

Niech $n_{\Gamma I}$ jest liczbą następników interwału I w grafie pochodnym.

Twierdzenie 3.2. Wykonanie Algorytmu 3.1A wymaga zrealizowania dla każdego interwału pierwszego rzędu

$$2e_I^f + 4e_I^o - 2e_I^\emptyset - e_I^{b1} + n_I - n_I^\emptyset - 2n_{\Gamma I} - 1 + \max(0, e_I^b - 1)$$

operacji na wektorach bitów.

Dowód. Obliczając G (krok A1) należy wykonać dla każdego wierzchołka różnego od głowy, z niepustym zbiorem następników, o jedną operację na wektorach bitów mniej niż liczba krawędzi dochodzących do tego wierzchołka (dla zsumowania GPATH) plus dwie operacje. Liczba wierzchołków różnych od głowy, z niepustym zbiorem następników, jest $n_I - n_I^\emptyset - 1$, zaś liczba krawędzi dochodzących do tych wierzchołków jest $e_I^f - e_I^\emptyset$, tak więc obliczając G należy wykonać

$$[e_I^f - e_I^\emptyset - (n_I - n_I^\emptyset - 1)] + 2(n_I - n_I^\emptyset - 1) = e_I^f - e_I^\emptyset + n_I - n_I^\emptyset - 1$$

operacji. Obliczenie T (krok A2) wymaga zrealizowania, dla każdego wierzchołka j różnego od głowy, z niepustym zbiorem następników $\Gamma j - \{1\}$, o jedną operację mniej niż liczba krawędzi dochodzących do tego wierzchołka (dla zsumowania TPATH) plus jedna operacja. Liczba wierzchołków

różnych od głowy, z niepustym zbiorem następników $\Gamma j - \{1\}$, jest $n_I - n_I^{\phi} - n_I^{b1} - 1$, zaś liczba krawędzi dochodzących do tych wierzchołków jest $e_I^f - e_I^{\phi} - e_I^{b1}$, tak więc obliczenie T wymaga realizacji

$$\left[e_I^f - e_I^{\phi} - e_I^{b1} - (n_I - n_I^{\phi} - n_I^{b1} - 1)\right] + (n_I - n_I^{\phi} - n_I^{b1} - 1) = e_I^f - e_I^{\phi} - e_I^{b1}$$

operacji. Dla obliczenia RDEFEN(1) (krok A7) niezbędne jest wykonanie o jedną operację mniej niż liczba krawędzi zamykających w I, tj. w sumie max $(0, e_I^b - 1)$[1] operacji. Dla każdego interwału J będącego następnikiem I, obliczenie TRANS(I,J) (krok A8) wymaga wykonania liczby operacji równej liczbie krawędzi wychodzących z I i dochodzących do J minus jeden, albo w sumie $e_I^o - n_{\Gamma I}$ operacji. Aby obliczyć GEN(I,J) (krok A9) dla każdego interwału J trzeba wykonać dwie operacje na wektorach bitów oraz o jedną operację mniej niż liczba krawędzi wychodzących z I i dochodzących do J, a więc łącznie

$$(2e_I^o) + (e_I^o - n_{\Gamma I}) = 3e_I^o - n_{\Gamma I}$$

operacji. Sumując operacje w krokach A1, A2, A7, A8 i A9 otrzymujemy

$$(e_I^f - e_I^{\phi} + n_I - n_I^{\phi} - 1) + (e_I^f - e_I^{\phi} - e_I^{b1}) + \left[\max(0, e_I^b - 1)\right] +$$

$$+ (e_I^o - n_{\Gamma I}) + (3e_I^o - n_{\Gamma I}) = 2e_I^f + 4e_I^o - 2e_I^{\phi} - e_I^{b1} +$$

$$+ n_I - n_I^{\phi} - 2n_{\Gamma I} - 1 + \max(0, e_I^b - 1).\square$$

<u>Twierdzenie 3.3</u>. Wykonanie Algorytmu 3.1A wymaga zrealizowania dla każdego interwału w pochodnym grafie przepływu

$$3e_I + 2e_I^f + 4e_I^o - 3e_I^h - 2e_I^{\phi} - e_I^b - e_I^{b1} - 2n_I + 2n_I^{\phi} + n_I^{b1} - 2n_{\Gamma I} +$$

$$+ 2 + \max(0, e_I^b - 1)$$

operacji na wektorach bitów.

<u>Dowód</u>. Obliczenie G (krok A3) wymaga wykonania, dla każdego wierzchołka różnego od głowy interwału, z niepustym zbiorem następników, o jedną

[1] Niezbędne jest zastosowanie funkcji <u>max</u>, ponieważ liczba operacji musi być nieujemna w przypadku, gdy $e_I^b = 0$.

operację na wektorach bitów mniej niż liczba krawędzi dochodzących do tego wierzchołka, a więc w sumie

$$e_I^f - e_I^{\phi} - (n_I - n_I^{\phi} - 1) = e_I^f - e_I^{\phi} - n_I + n_I^{\phi} + 1$$

operacji. Aby obliczyć GPATH (krok A4) niezbędne jest wykonanie dwóch operacji dla każdej krawędzi wychodzącej z wierzchołka interwału I, z wyjątkiem krawędzi wychodzących z głowy interwału, tj. w sumie $2(e_I - e_I^h)$ operacji. W każdym wierzchołku j różnym od głowy, z niepustym zbiorem następników $\Gamma j - \{1\}$, obliczenie wartości T (krok A5) wymaga wykonania o jedną operację na wektorach bitów mniej niż liczba krawędzi dochodzących do tego wierzchołka, a więc sumarycznie

$$e_I^f - e_I^{\phi} - e_I^{b1} - (n_I - n_I^{\phi} - n_I^{b1} - 1) = e_I^f - e_I^{\phi} - e_I^{b1} - n_I + n_I^{\phi} + n_I^{b1} + 1$$

operacji. Aby obliczyć TPATH (krok A6) należy zrealizować jedną operację dla każdej krawędzi wychodzącej z wierzchołka interwału I, z wyjątkiem krawędzi wychodzących z głowy interwału oraz krawędzi zamykających interwału, tj. w sumie $e_I - e_I^h - e_I^b$ operacji. Dodając operacje z kroków A3, A4, A5, A6, A7, A8 i A9 (liczby operacji w krokach A7,, A8 i A9 są takie same jak w dowodzie Twierdzenia 3.2) otrzymujemy:

$$(e_I^f - e_I^{\phi} - n_I + n_I^{\phi} + 1) + 2(e_I - e_I^h) + (e_I^f - e_I^{\phi} - e_I^{b1} - n_I + n_I^{\phi} +$$

$$+ n_I^{b1} + 1) + (e_I - e_I^h - e_I^b) + [\max(0, e_I^b - 1)] + (e_I^i - n_{\Gamma I}) +$$

$$+ (3e_I^o - n_{\Gamma I}) = 3e_I + 2e_I^f + 4e_I^o - 3e_I^h - 2e_I^{\phi} - e_I^b - e_I^{b1} - 2n_I +$$

$$+ 2n_I^{\phi} + n_I^{b1} - 2n_{\Gamma I} + 2 + \max(0, e_I^b - 1).\square$$

<u>Przebieg 2</u>

W przebiegu 2 wektory RDEFEN(1), TRANS(I,J) oraz GEN(I,J) obliczone w przebiegu 1 zostaną użyte do obliczenia wektorów RDEFEN dla każdego wierzchołka w grafie przepływu programu. Niech $G_m = (N_m, \Gamma_m, s_m)$ jest granicznym grafem przepływu sekwencji redukcji składającym się z pojedynczego wierzchołka s_m bez krawędzi. Pomiędzy przebiegami 1 i 2, wektor RDEFEN(s_m) jest inicjalizowany bądź wektorem zerowym (RDEFEN(s_m) := wektor zerowy), bądź wektorem reprezentującym zbiór definicji, o których wiadomo, że osiągają program z zewnątrz. Algorytm, który przy danych wektorach RDEFEN(1), TRANS(I,J) i GEN(I,J) dla interwału I oraz wszystkich jego następników J, oblicza wektory RDEFEN dla każdego wierzchołka interwału I, jest następujący.

Algorytm 3.1B: Oblicza wektory bitów RDEFEN(j), dla każdego wierzchołka interwału.

Dane: 1. Wierzchołki interwału I ponumerowane od 1 do n_I w interwałowym porządku.
2. RDEFEN(1) oraz RDEFEN(I).
3. TRANS(j,k), $1 \leqslant j \leqslant n_I$, k przebiega po wszystkich następnikach j nie będących głową I.
4. GEN(j,k), $1 \leqslant j \leqslant n_I$, k przebiega po wszystkich następnikach j.

Zmienne pomocnicze:
RDEFEX(j,k), $1 \leqslant j \leqslant n_I$, k przebiega po wszystkich następnikach j nie będących głową I[1].

Wyniki: RDEFEN(j), $1 \leqslant j \leqslant n_I$.

Metoda[2]

```
begin
  -- Inicjalizacja.
  B1: RDEFEN(1) := RDEFEN(1) ∨ RDEFEN(I);
  if I jest interwałem pierwszego rzędu then
    -- Dla głowy interwału.
    if Γ1 ∩ I - {1} nie jest zbiorem pustym then
      -- x jest dowolnym wierzchołkiem z Γ1.
      -- T jest zmienną chwilową.
      B2: T := (RDEFEN(1) ∧ TRANS(1,x)) ∨ GEN(1,x);
      for wszystkich k ∈ Γ1 ∩ I - {1} do RDEFEX(1,k) := T endfor
    endif;
    -- Dla pozostałych wierzchołków.
    for j from 2 to n_I do
      B3: RDEFEN(j) := ∨ RDEFEX(k,j);
                       k∈Γ^-1 j
      if Γj ∩ I - {1} nie jest zbiorem pustym then
        B4: T := (RDEFEN(j) ∧ TRANS(j,x)) ∨ GEN(j,x);
        for wszystkich k ∈ Γj ∩ I - {1} do RDEFEX(j,k) := T endfor
      endif
    endfor
  else
    -- Dla interwału w pochodnym grafie przepływu.
    -- Dla głowy interwału.
    for wszystkich k ∈Γ1 ∩ I - {1} do
```

[1] W Allen i Cocke [1976], zbiory odpowiadające wektorom RDEFEX są oznaczone przez A.

[2] Jak w algorytmie 3.1A, w celu zmniejszenia złożoności, sposób przetwarzania interwałów pierwszego rzędu został nieco zmodyfikowany w porównaniu z Allen i Cocke [1976].


```
      B5: RDEFEX(1,k) := (RDEFEN(1) ∧ TRANS(1,k)) ∨ GEN(1,k)
    endfor;
    -- Dla pozostałych wierzchołków.
    for j from 2 to n_I do
      B6: RDEFEN(j) := ∨ RDEFEX(k,j);
                       k∈Γ^-1 j
      for wszystkich k ∈ Γj ∩ I - {1} do
        B7: RDEFEX(j,k) := (RDEFEN(j) ∧ TRANS(j,k)) ∨ GEN(j,k)
      endfor
    endfor
  endif
end □
```

Twierdzenie 3.4. Wykonanie Algorytmu 3.1B wymaga zrealizowania dla każdego interwału pierwszego rzędu

$$e_I^f + 2n_I^f - n_I + 2$$

operacji na wektorach bitów.

Dowód. Obliczenie wektorów RDEFEN wymaga wykonania jednej operacji dla głowy interwału (krok B1) oraz o jedną operację mniej niż liczba krawędzi dochodzących do wierzchołka, dla wierzchołków różnych od głowy (krok B3); w sumie daje to

$$(1) + \left[e_I^f - (n_I - 1)\right] = e_I^f - n_I + 2$$

operacji. Dla każdego wierzchołka, z którego wychodzi co najmniej jedna krawędź w przód, wykonuje się dwie operacje, aby obliczyć T (kroki B2 i B4), a więc w sumie $2\,n_I^f$ operacji. Sumując liczby tych operacji mamy

$$(e_I^f - n_I + 2) + (2n_I^f) = e_I^f + 2n_I^f - n_I + 2. \square$$

Twierdzenie 3.5. Wykonanie Algorytmu 3.1B wymaga zrealizowania dla każdego interwału w pochodnym grafie przepływu

$$3e_I^f - n_I + 2$$

operacji na wektorach bitów.

Dowód. Liczba operacji niezbędna do obliczenia wektorów RDEFEN dla każdego interwału w pochodnych grafach przepływu (kroki B1 i B6) jest taka sama, jak liczba operacji dla interwału pierwszego rzędu, tj. $e_I^f - n_I + 2$ (patrz dowód Twierdzenia 3.4). Aby obliczyć wektory RDEFEX (kroki B5 i B7 należy wykonać dwie operacje dla każdej krawędzi w przód interwału, a więc w sumie $2\,e_I^f$ operacji. Po dodaniu tych operacji otrzymujemy

$$(e_I^f - n_I + 2) + (2e_I^f) = 3e_I^f - n_I + 2.\square$$

Kompletny algorytm analizy interwałów dla rozwiązania problemu definicji osiągających wygląda następująco:

Algorytm 3.1C: Algorytm analizy interwałów dla problemu definicji osiągających.

Dane: 1. Redukowalny graf przepływu $G = (N, \Gamma, s)$.
2. Sekwencja grafów pochodnych $(G = G_0, G_1, \ldots, G_m)$ dla G, gdzie G_m jest trywialnym grafem przepływu, $G_i = (N_i, \Gamma_i, s_i)$.
3. TRANS(j,k) i GEN(j,k) dla każdego $j \in N$ oraz $k \in \Gamma j$.
4. Zawartości interwałów we wszystkich grafach pochodnych podane w porządku interwałowym.

Wyniki: RDEFEN(j) dla każdego $j \in N$.

Metoda:

```
begin
  -- Przebieg 1: Przetwarzając sekwencję grafów pochodnych poczynając od
  -- interwałów wewnętrznych do zewnętrznych, stosować Algorytm 3.1A.
  for i from 1 to m do
     C1: for każdego wierzchołka I w G_i do -- I jest interwałem.
         Stosować Algorytm 3.1A do obliczenia wektorów RDEFEN, TRANS oraz
         GEN dla I.
     endfor
  endfor;
  RDEFEN(s_m) := wektor zerowy;
  -- Przebieg 2: Przetwarzając sekwencję grafów pochodnych poczynając
  -- od interwałów zewnętrznych do wewnętrznych, stosować Algorytm
  -- 3.1B.
  for i from m to 1 by -1 do
     C2: for każdego wierzchołka I w G_i do
         Stosować Algorytm 3.1B do obliczenia wektorów RDEFEN
         dla wierzchołków zawartych w I.
     endfor
  endfor
end □
```

Twierdzenie 3.6. Wykonanie Algorytmu 3.1C wymaga realizacji

$$3e_0^f + 4e_0^o - 2e_0^\phi - e_0^{b1} + 2n_0^f - n_0^\phi - 3n_1 + \sum_{i=1}^{m-1} (e_i + 5e_i^f + 4e_i^o - 3e_i^h -$$

$$- 2e_i^\phi - e_i^b - e_i^{b1} + n_i + 2n_i^\phi + n_i^{b1} + 4 + \sum_{i=1}^{m} \sum_{I \in N_i} \max(0, e_I^b - 1)$$

operacji na wektorach bitów.

Dowód. Rozważmy graf pochodny $G_i = (N_i, \Gamma_i, s_i)$. Każdy wierzchołek w tym grafie reprezentuje pewien interwał I. Wierzchołki zawarte w I są wierzchołkami grafu G_{i-1}. Niech e_I^f jest liczbą krawędzi w przód wychodzących z wierzchołków I. Wówczas

$$\sum_{I \in N_i} e_I^f = e_{i-1}^f$$

ponieważ wszystkie wierzchołki $I \in N_i$ "zawierają" wszystkie wierzchołki grafu N_{i-1}. Podobne równości sumacyjne są spełnione dla e_I, e_I^o, $n_I, \ldots$, itp. jeśli $n_{\Gamma I}$ jest liczbą wierzchołków $J \in N_i$ będących następnikami I to $\sum_{I \in N_i} n_{\Gamma I} = e_i$; prawdziwą jest również oczywista równość $\sum_{I \in N_i} 1 = n_i$.

Przypomnijmy, że na podstawie Twierdzeń 3.2 i 3.3, wykonanie Algorytmu 3.1A wymaga realizacji

$$2e_I^f + 4e_I^o - 2e_I^\phi - e_I^{b1} + n_I - n_I^\phi - 2n_{\Gamma I} - 1 + \max(0, e_I^b - 1)$$

operacji na wektorach bitów, dla każdego interwału pierwszego rzędu oraz

$$3e_I + 2e_I^f + 4e_I^o - 3e_I^h - 2e_I^\phi - e_I^b - e_I^{b1} - 2n_I + 2n_I^\phi + n_I^{b1} - 2n_{\Gamma I} + 2 +$$

$$+ \max(0, e_I^b - 1)$$

operacji dla każdego interwału w pochodnych grafach przepływu. W kroku C1 Algorytm 3.1A jest wywoływany raz dla każdego $I \in N_i$, a więc aby wykonać ten krok trzeba zrealizować

$$\sum_{I \in N_1} \left[2e_I^f + 4e_I^o - 2e_I^o - e_I^{b1} + n_I - n_I^\phi - 2n_{\Gamma I} - 1 + \max(0, e_I^b - 1)\right] =$$

$$= 2e_0^f + 4e_0^o - 2e_0^\phi - e_0^{b1} + n_0 - n_0^\phi - 2e_1 - n_1 + \sum_{I \in N_1} \max(0, e_I^b - 1)$$

operacji dla grafu pierwszego rzędu (i = 1) oraz

$$\sum_{I \in N_i} \left[3e_I + 2e_I^f + 4e_I^o - 3e_I^h - 2e_I^\phi - e_I^b - e_I^{b1} - 2n_I + 2n_I^\phi + n_I^{b1} -\right.$$

$$- 2n_{\Gamma I} + 2 + \max(0, e_I^b - 1)] = 3e_{i-1} + 2e_{i-1}^f + 4e_{i-1}^o - 3e_{i-1}^h -$$

$$- 2e_{i-1}^{\emptyset} - e_{i-1}^b - e_{i-1}^{b1} - 2n_{i-1} + 2n_{i-1}^{\emptyset} + n_{i-1}^{b1} - 2e_i + 2n_i +$$

$$+ \sum_{I \in N_i} \max(0, e_I^b - 1)$$

operacji dla każdego grafu pochodnego $(m \leqslant i \leqslant 2)$. Ponieważ krok C1 jest wykonywany raz dla każdego i, $1 \leqslant i \leqslant m-1$, to w pierwszym przebiegu trzeba zrealizować

$$2e_o^f + 4e_o^o - 2e_o^{\emptyset} - e_o^{b1} + n_o - n_o^{\emptyset} - 2e_1 - n_1 + \sum_{I \in N_1} \max(0, e_I^b - 1) +$$

$$+ \sum_{i=2}^{m} (3e_{i-1} + 2e_{i-1}^f + 4e_{i-1}^o - 3e_{i-1}^h - 2e_{i-1}^{\emptyset} - e_{i-1}^b - e_{i-1}^{b1} -$$

$$- 2n_{i-1} + 2n_{i-1}^{\emptyset} + n_{i-1}^{b1}) + \sum_{i=1}^{m} (-2e_i + 2n_i) + \sum_{i=2}^{m} \sum_{I \in N_i} \max(0, e_I^b - 1)$$

operacji. Modyfikując wskaźnik sumowania drugiej sumy, łącząc wyrazy oraz uwzględniając $e_m = 0$, $n_m = 1$ i $- 2e_1 - n_1 + \sum_{i=2}^{m} (-2e_i + 2n_i) = -3n_1 + \sum_{i=1}^{m-1} (-2e_i + 2n_i) + 2$ otrzymujemy

$$2e_o^f + 4e_o^o - 2e_o^{\emptyset} - e_o^{b1} + n_o - n_o^{\emptyset} - 3n_1 + \sum_{i=1}^{m-1} (e_i + 2e_i^f + 4e_i^o - 3e_i^h -$$

$$- 2e_i^{\emptyset} - e_i^b - e_i^{b1} + 2n_i^{\emptyset} + n_i^{b1}) + 2 + \sum_{i=1}^{m} \sum_{I \in N_i} \max(0, e_I^b - 1),$$

co jest liczbą operacji niezbędnych dla wykonania pierwszego przebiegu.

Zgodnie z Twierdzeniami 3.4 i 3.5, każde wykonanie Algorytmu 3.1B w kroku C2 wymaga realizacji $e_I^f + 2n_I^f - n_I + 2$ operacji dla interwału pierwszego rzędu oraz $3e_I^f - n_I + 2$ operacji dla każdego interwału w pochodnych grafach przepływu. Ponieważ Algorytm 3.1B jest wywoływany w kroku C2 jednokrotnie dla każdego $I \in N_1$, to dla wykonania tego kroku trzeba zrealizować

$$\sum_{I \in N_1} (e_I^f + 2n_I^f - n_I + 2) = e_o^f + 2n_o^f - n_o + 2n_1$$

operacji dla grafu pierwszego rzędu (i = 1) oraz

$$\sum_{I \in N_i} (3e_I^f - n_I + 2) = 3e_{i-1}^f - n_{i-1} + 2n_i$$

operacji dla każdego grafu pochodnego $(m \geqslant i \geqslant 2)$. Krok C2 powtarzany jest jednokrotnie dla każdego i, $m \geqslant i \geqslant 1$, a więc wykonanie drugiego przebiegu wymaga zrealizowania

$$e_o^f + 2n_o^f - n_o + 2n_1 + \sum_{i=2}^{m} (3e_{i-1}^f - n_{i-1} + 2n_i) = e_o^f + 2n_o^f - n_o +$$

$$+ \sum_{i=1}^{m-1} (3e_i^f + n_i) + 2$$

operacji na wektorach bitów. Sumując liczby operacji dla pierwszego i drugiego przebiegu otrzymujemy

$$3e_o^f + 4e_o^o - 2e_o^{\emptyset} - e_o^{b1} + 2n_o^f - n_o^{\emptyset} - 3n_1 + \sum_{i=1}^{m-1} (e_i + 5e_i^f + 4e_i^o - 3e_i^h -$$

$$- 2e_i^{\emptyset} - e_i^b - e_i^{b1} + n_i + 2n_i^{\emptyset} + n_i^{b1}) + 4 + \sum_{i=1}^{m} \sum_{I \in N_i} \max(0, e_I^b - 1),$$

co jest poszukiwanym rezultatem.□

3.3. Algorytm analizy regionów

W niniejszym punkcie wszystkie zbiory informacji o przepływie danych będą obliczane jedynie dla oryginalnego grafu przepływu G oraz wszystkie z nich będą przyporządkowane wierzchołkom, a nie krawędziom. Tak więc, zbiory trans, gen oraz rdefex będą oznaczone przez trans(x), gen(x) oraz rdefex(x) dla $x \in N$. Analogicznie jak w Twierdzeniu 3.1 mamy

$$\text{rdefen}(x) = \bigcup_{k \in \Gamma^{-1}x} ((\text{rdefen}(k) \cap \text{trans}(k)) \cup \text{gen}(k))$$

oraz

$$\text{rdefex}(x) = (\text{rdefen}(x) \cap \text{trans}(x)) \cup \text{gen}(x).$$

Wprowadźmy dla każdego wierzchołka x w interwale, zbiory definicji pathen(x) oraz pathex(x), które są "transmitowane" wzdłuż pewnej drogi od wejścia interwału do, odpowiednio, wejścia i wyjścia wierzchołka x.

Algorytm analizy regionów dla problemu definicji osiągających (Czech [1981 , 1983, 1984] składa się z dwóch faz. Celem pierwszej fazy, w której przetwarzane są minimalne regiony, tj. interwały G, jest obliczenie które definicje występujące wewnątrz interwału mogą osiągnąć wejście i wyjście każdego wierzchołka tego interwału. Podczas pierwszej fazy pomijane są "wpływy" krawędzi zamykających oraz części grafu leżącej na zewnątrz interwałów. Dodatkowo, w pierwszej fazie wyznacza się takie definicje, które w przypadku osiągnięcia głowy interwału będą osiągać wejścia i wyjścia poszczególnych wierzchołków w interwale. W drugiej fazie przechodząc sekwencję grafów pochodnych $(G = G_0, G_1, \ldots, G_m)$ dla G, przetwarzane są kolejno coraz to większe regiony, z których każdy może zawierać kilka mniejszych regionów G uprzednio przetwarzanych. W fazie tej wyznacza się te definicje, które mogą osiągać każdy wierzchołek regionu G z wnętrza tego regionu wzdłuż krawędzi zamykających oraz z części grafu leżącej na zewnątrz regionu.

Oto bardziej szczegółowe opisy algorytmów pierwszej oraz drugiej fazy.

Faza 1. W pierwszej fazie przetwarzane są w porządku interwałowym wierzchołki każdego interwału (minimalnego regionu) G oraz wyznaczane są wstępne estymaty[1] zbiorów rdefen(j) oraz rdefex(j), jak również zbiory pathen(j) oraz pathex(j), dla każdego wierzchołka w interwale. Zbiory te obliczane są następująco:

(i) dla głowy h każdego interwału

$$\mathrm{rdefen}(h) = \emptyset$$
$$\mathrm{rdefex}(h) = \mathrm{gen}(h)$$
$$\mathrm{pathen}(h) = D^{2)}$$
$$\mathrm{pathex}(h) = \mathrm{trans}(h)$$

(ii) dla każdego wierzchołka j interwału różnego od głowy

$$\mathrm{rdefen}(j) = \bigcup_{k \in \Gamma^{-1} j} \mathrm{rdefex}(k)$$
$$\mathrm{rdefex}(j) = \mathrm{rdefen}(j) \cap \mathrm{trans}(j) \cup \mathrm{gen}(j)$$
$$\mathrm{pathen}(j) = \bigcup_{k \in \Gamma^{-1} j} \mathrm{pathex}(k)$$
$$\mathrm{pathex}(j) = \mathrm{pathen}(j) \cap \mathrm{trans}(j)$$

Zauważmy, że obliczenia fazy 1 rozpoczynają się przy założeniu, że zbiór rdefen(h) jest pusty, albo inaczej, że żadna definicja nie osiąga (głowy) interwału. W fazie 2 będą uzyskiwane coraz lepsze przybliżenia zbiorów rdefen przez uwzględnienie "wpływów" coraz większych regionów (kontekstów interwału), a zatem zostanie w ten sposób znalezione rozwiązanie minimalne.

[1] Są to wstępne estymaty, ponieważ ignoruje się na razie "wpływy" krawędzi zamykających oraz części grafu leżącej poza interwałami.

[2] D oznacza zbiór wszystkich definicji w analizowanym programie.

Następujący algorytm oblicza wyżej wymienione zbiory dla każdego wierzchołka interwału używając ich wektorowych reprezentacji.

Algorytm 3.2A: Oblicza wektory PATHEN i PATHEX oraz wstępne estymaty wektorów RDEFEN i RDEFEX.

Dane: 1. Wierzchołki interwału I ponumerowane od 1 do n_I w porządku interwałowym.
2. Informacje o poprzednikach i następnikach każdego wierzchołka interwału.
3. Wektory TRANS(j) i GEN(j), $1 \leq j \leq n_I$.

Wyniki: Wektory PATHEN(j) i PATHEX(j) oraz wstępne estymaty wektorów RDEFEN(j) i RDEFEX(j), $1 \leq j \leq n_I$.

Metoda:

```
begin
  -- Dla głowy interwału
  A1: RDEFEN(1) := wektor zerowy;
  A2: PATHEN(1) := wektor jednostkowy;
  if Γ 1 nie jest zbiorem pustym then
     A3: RDEFEX(1) := GEN(1);
     A4: PATHEX(1) := TRANS(1)
  endif;
  -- Dla każdego wierzchołka różnego od głowy.
  for j from 2 to n_I do
     A5: RDEFEN(j) := ⋁_{k∈Γ^{-1}j} RDEFEX(k);
     A6: PATHEN(j) := ⋁_{k∈Γ^{-1}j} PATHEX(k);
     if Γ j nie jest zbiorem pustym then
        A7: RDEFEX(j) := (RDEFEN(j) ∧ TRANS(j)) ∨ GEN(j);
        A8: PATHEX(j) := PATHEN(j) ∧ TRANS(j)
     endif
  endfor
end □
```

Twierdzenie 3.7. Algorytm 3.2A jest częściowo poprawny oraz jego obliczenie dochodzi do punktu końcowego.

Dowód. Dochodzenie obliczenia algorytmu do punktu końcowego wynika ze skończoności n_I. Poprawność zostanie pokazana za pomocą czterech lematów. Zapis "d ∈ WEKTORBITÓW" będzie oznaczał, że "na pozycji odpowiadającej definicji d w wektorze bitów WEKTORBITÓW jest 1".

Lemat 3.7.1. Po wykonaniu Algorytmu 3.2A, definicja d ∈ RDEFEN(j), $2 \leq j \leq n_I$, wtedy i tylko wtedy, gdy d występuje w pewnym wierzchołku m, $1 \leq m \leq j-1$ oraz istnieje V-wolna droga od d w wierzchołku m do (wejścia) j.

Dowód. Przeprowadzimy dowód indukcyjny względem j.

Podstawa: j = 2. Jedynym poprzednikiem wierzchołka 2 musi być wierzchołek 1. Stąd, na podstawię kroków A3 i A5, RDEFEN(2) = RDEFEX(1) = GEN(1). Lecz $d \in$ GEN(1) wtedy i tylko wtedy, gdy d występuje w wierzchołku 1 oraz istnieje V-wolna droga od d do wyjścia wierzchołka 1, tj. do wejścia wierzchołka 2.

Krok indukcyjny. Załóżmy, że lemat jest prawdziwy dla wszystkich $1 \leqslant j \leqslant p-1$, gdzie $p \leqslant n_I$. Na podstawie kroku A5, $d \in$ RDEFEN(p) wtedy i tylko wtedy, gdy $d \in$ RDEFEX(k) dla pewnego bezpośredniego poprzednika k wierzchołka p. Lecz to jest prawdziwe (krok A7) wtedy i tylko wtedy, gdy $d \in$ GEN(k) lub, gdy $d \in$ RDEFEN(k) i $d \in$ TRANS(k). W pierwszym przypadku, $d \in$ GEN(k) wtedy i tylko wtedy, gdy d występuje w wierzchołku k, $1 \leqslant k \leqslant p-1$ oraz istnieje V-wolna droga od d do wyjścia k, tj. do wejścia p. W drugim przypadku $d \in$ RDEFEN(p) wtedy i tylko wtedy, gdy $d \in$ RDEFEN(k), co oznacza, że d występuje w pewnym wierzchołku m, $1 \leqslant m < k-1$, dla którego istnieje V-wolna droga od d w m do k (na podstawie hipotezy indukcyjnej) oraz istnieje V-wolna droga przez k. To jest jednak równoważne istnieniu pewnego wierzchołka m, $1 \leqslant m \leqslant p$, zawierającego definicję d, dla którego występuje V-wolna droga od d w m do p. (Z uwagi na to, że k jest bezpośrednim poprzednikiem p i wierzchołki są ponumerowane w porządku interwałowym, zachodzi $1 \leqslant m \leqslant k-1 < k \leqslant p$).□

Lemat 3.7.2. Po wykonaniu Algorytmu 3.2A, definicja $d \in$ RDEFEX(j), $2 \leqslant j \leqslant n_I$, wtedy i tylko wtedy, gdy d występuje w pewnym wierzchołku m, $1 \leqslant m \leqslant j-1$ oraz istnieje V-wolna droga od d w wierzchołku m do wyjścia j.

Dowód. Ponieważ dowód jest podobny do dowodu Lematu 3.7.1, więc pomijamy go.□

Lemat 3.7.3. Po wykonaniu Algorytmu 3.2A, definicja $d \in$ PATHEN(j), $2 \leqslant j \leqslant n_I$, wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia interwału do (wejścia) j.

Dowód. Przeprowadzamy indukcję względem j.

Podstawa. j = 2. Jedynym poprzednikiem wierzchołka 2 musi być głowa interwału - wierzchołek 1. Dlatego, na podstawie kroków A4 i A6, PATHEN(2) = PATHEX(1) = TRANS(1). Lecz $d \in$ TRANS(1) wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia głowy interwału, które jest identyczne z wejściem interwału, do wierzchołka 2.

Krok indukcyjny. Przyjmijmy, że lemat jest prawdziwy dla $1 \leqslant j \leqslant p-1$, gdzie $p \leqslant n_I$. Na podstawie kroku A6, $d \in$ PATHEN(p) wtedy i tylko wtedy gdy $d \in$ PATHEX(k) dla pewnego poprzednika k wierzchołka p. Zachodzi to jednak wtedy i tylko wtedy, gdy $d \in$ PATHEN(k) i $d \in$ TRANS(k) (krok A8). Ponieważ wierzchołki dane są w porządku interwałowym, to $k \leqslant p-1$. Tak więc, $d \in$ PATHEN(p) wtedy i tylko wtedy, gdy dla pewnego poprzednika k wierzchołka p istnieje V-wolna droga od wejścia interwału do k (na podstawie hipotezy indukcyjnej) oraz istnieje V-wolna droga przez wierzchołek k do p. To zaś jest równoważne istnieniu V-wolnej drogi od wejścia interwału do p.□

Lemat 3.7.4. Po wykonaniu Algorytmu 3.2A, definicja $d \in$ PATHEX(j), $2 \leqslant j \leqslant n_I$, wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia interwału do wyjścia j.

Dowód. Dowód jest analogiczny, jak dla Lematu 3.7.3, więc pomijamy go.□

Lematy 3.7.1 - 3.7.4. dowodzą Twierdzenia 3.7.□ .

Twierdzenie 3.8. Wykonanie Algorytmu 3.2A wymaga realizacji

$$2e_I^f + n_I - 3n_I^\phi - 1$$

operacji na wektorach bitów.

Dowód. Aby obliczyć wektory RDEFEN i PATHEN (kroki A5 i A6) należy wykonać dla każdego wierzchołka różnego od głowy interwału o jedną operację na wektorach bitów mniej, niż liczba krawędzi dochodzących do tego wierzchołka. Liczba krawędzi dochodzących do wierzchołków interwału różnych od głowy wynosi e_I^f, a więc do obliczenia RDEFEN i PATHEN niezbędne jest wykonanie $2(e_I^f - n_I + 1)$ operacji. W celu obliczenia wektorów RDEFEX i PATHEX (kroki A7 i A8) należy wykonać trzy operacje dla każdego wierzchołka różnego od głowy interwału, z niepustym zbiorem następników, tj. w sumie $3(n_I - n_I^\phi - 1)$ operacji. Dodając powyższe liczby operacji otrzymujemy

$$[2(e_I^f - n_I + 1)] + [3(n_I - n_I^\phi - 1)] = 2e_I^f + n_I - 3n_I^\phi - 1.\boxdot$$

Faza 2. Podczas drugiej fazy przetwarzana jest sekwencja grafów pochodnych $(G = G_0, G_1, \ldots, G_m)$ dla grafu $G = (N, \Gamma, s)$. Zbiory rdefen(j) i rdefex(j) są modyfikowane przez uwzględnienie tych definicji, które mogą osiągnąć wierzchołek j z wnętrza coraz większego regionu wzdłuż krawędzi zamykających oraz z części grafu leżącej na zewnątrz regionu. Niech R_J jest regionem G reprezentowanym przez wierzchołek J w pewnym grafie pochodnym. Załóżmy, że wierzchołki R_J są ponumerowane od 1 do n_J w porządku interwałowym. W drugiej fazie analizowane są wierzchołki każdego interwału w grafie pochodnym i dla wierzchołka J, który nie jest głową interwału, przetwarzane są wierzchołki regionu R_J w następujący sposób.

(i) jeśli wierzchołek $h \in R_J$ jest głową interwału pierwszego rzędu $I \subseteq R_J$, to

$$\text{rdefen}(h) = t = \bigcup_{k \in \Gamma^{-1}h} \text{rdefex}(k)$$

$$\text{rdefex}(h) = \text{rdefex}(h) \cup (t \cap \text{pathex}(h))$$

(ii) dla każdego wierzchołka $j \in I$ różnego do głowy
rdefen(j) = rdefen(j) $\cup$ (t $\cap$ pathen(j))
rdefex(j) = rdefex(j) $\cup$ (t $\cap$ pathex(j))

Zaprezentujemy obecnie algorytm, który dla danych starych wartości wektorów RDEFEN i RDEFEX oraz wektorów PATHEN i PATHEX, oblicza nowe wartości wektorów RDEFEN i RDEFEX dla każdego wierzchołka w regionie R_J.

Algorytm 3.2B: Uaktualnia wektory RDEFEN i RDEFEX.

Dane: 1. Region R_J grafu G reprezentowany przez wierzchołek J w pewnym pochodnym grafie przepływu. Wierzchołki R_J są ponumerowane od 1 do n_J w porządku interwałowym.
2. Wektory PATHEN(j) i PATHEX(j), $1 \leqslant j \leqslant n_J$.
3. Stare wartości wektorów RDEFEN(j) i RDEFEX(j), $1 \leqslant j \leqslant n_J$.

Wyniki: Nowe wartości wektorów RDEFEN(j) i RDEFEX(j), $1 \leqslant j \leqslant n_J$.

Metoda:

```
begin
  for j from 1 to n_J do
    if j jest głową then
      -- T jest chwilowym wektorem bitów.
      B1: RDEFEN(j) := T := ∨ RDEFEX(k);
                            k∈Γ⁻¹j
      if Γ j nie jest zbiorem pustym then
        B2: RDEFEX(j) := RDEFEX(j) ∨ (T ∧ PATHEX(j))
      endif
    else -- Dla każdego wierzchołka różnego od głowy.
      B3: RDEFEN(j) := RDEFEN(j) ∨ (T ∧ PATHEN(j));
      if Γ j nie jest zbiorem pustym then
        B4: RDEFEX(j) := RDEFEX(j) ∨ (T ∧ PATHEX(j))
      endif
    endif
  endfor
end □
```

Twierdzenie 3.9. Algorytm 3.2B jest częściowo poprawny oraz jego obliczenie dochodzi do punktu końcowego.

Dowód. Dochodzenie obliczenia algorytmu do punktu końcowego wynika ze skończoności R_J. Poprawności dowodzą następujące dwa lematy.

Lemat 3.9.1. Po wykonaniu Algorytmu 3.2B, definicja d jest w nowym (uaktualnionym) wektorze RDEFEN(j), $1 \leqslant j \leqslant n_J$, wtedy i tylko wtedy, gdy d była w starym wektorze RDEFEN(j) (sprzed akutalizacji) lub, gdy $d \in$ RDEFEN(1) oraz istnieje V-wolna droga od wierzchołka 1 do j.

Dowód. Niech przedrostki nowy- oraz stary- wyróżniają odpowiednio nowe oraz stare wartości wektorów bitów.

Przypadek 1. $d \in$ staryRDEFEN(j). Dla każdego wierzchołka różnego od głowy interwału, wektor nowyRDEFEN(j) jest obliczany przez dodanie logiczne wyrażenia $T \wedge$ PATHEN(j) do wektora staryRDEFEN(j) (krok B3). Tak więc wektor nowyRDEFEN(j) zawiera zawsze wszystkie definicje wektora staryRDEFEN(j) oraz dodatkowe definicje wynikające z dodania wyrażenia. Wektor nowyRDEFEN(j), dla każdej głowy interwału, jest obliczany przez zsumowanie wektorów RDEFEX(k), $k \in \Gamma^{-1}j$ (krok B1), z których każdy był obliczony w podobny sposób, jak opisano powyżej, podczas poprzednich wykonań Algorytmu 3.2B.

Przypadek 2. $d \in$ RDEFEN(1) oraz istnieje V-wolna droga od wierzchołka 1 do j. Przeprowadzamy dowód indukcyjny względem numeru interwału pierwszego rzędu w regionie R_J. Niech $h_1 = 1, h_2, \ldots, h_m$ są głowami interwałów w sekwencji wierzchołków regionu $1, 2, \ldots, n_J$. Zauważmy, że z uwagi na sposób w jaki Algorytm 3.2B jest wywoływany w fazie 2, wszystkie interwały w R_J z wyjątkiem pierwszego, tj. wierzchołki $h_2, h_2+1, \ldots, n_J$, były przetwarzane przez Algorytm 3.2B podczas analizy grafu przepływu niższego rzędu.

Podstawa. Rozważmy wierzchołki pierwszego interwału w R_J. Dla głowy interwału, wierzchołka 1, lemat jest oczywisty. Na podstawie kroku B3, $d \in$ RDEFEN(j) dla każdego j, $2 \leqslant j \leqslant h_2-1$, wtedy, gdy $d \in T$ i $d \in$ PATHEN(j). Ponieważ T jest sumą wektorów RDEFEX(k) dla wszystkich poprzedników k wierzchołka 1 (krok B1), to relacja $d \in T$ jest równoważna relacji $d \in$ RDEFEN(1). Na podstawie definicji, $d \in$ PATHEN(j) wtedy i tylko wtedy, gdy istnieje V-wolna droga od wierzchołka 1 do j.

Krok indukcyjny. Załóżmy, że lemat jest prawdziwy dla wierzchołków wszystkich interwałów $i \leqslant p-1$. Rozważmy interwał p. Dla głowy interwału, $d \in$ RDEFEN(h_p) wtedy, gdy $d \in$ RDEFEX(x) dla pewnego poprzednika k głowy h_p (krok B1), gdzie $1 \leqslant k \leqslant h_p-1$. (Możemy pominąć poprzedniki $k \notin (1, 2, \ldots, h_p-1)$ ponieważ ich "wpływ" na wektory RDEFEN został uwzględniony podczas przetwarzania grafów przepływu niższego rzędu). Na podstawie hipotezy indukcyjnej, $d \in$ RDEFEN(1) oraz istnieje V-wolna droga od wierzchołka 1 do k. Wynika stąd, że $d \in T_x$ = RDEFEN(h_x) oraz, że istnieje V-wolna droga od wierzchołka 1 do h_x, gdzie x jest interwałem zawierającym wierzchołek k, zaś T_x jest wartością chwilowego wektora bitów otrzymaną po przetworzeniu wierzchołka h_x. Na podstawie kroku B4 (lub B2, jeśli k jest głową h_x), $d \in$ RDEFEX(k) wtedy i tylko wtedy, gdy $d \in T_x$ i $d \in$ PATHEX(k). Relacje te są jednocześnie prawdziwe wtedy, gdy $d \in$ RDEFEN(1) oraz istnieje V-wolna droga od wierzchołka 1 do wyjścia wierzchołka k, tj. do wejścia wierzchołka p. Rozważmy teraz wierzchołki interwału p różne od głowy. Dla każdego wierzchołka j, $h_p + 1 \leqslant j \leqslant h_{p+1} - 1$, $d \in T_{h_p}$ wtedy, gdy $d \in$ RDEFEN(1) oraz istnieje V-wolna droga od wierzchołka 1 do h_p. Warunki te wraz z relacją $d \in$ PATHEN(j) są równoważne występowaniu V-wolnej drogi od wierzchołka 1 do h_p. □

Lemat 3.9.2. Po wykonaniu Algorytmu 3.2B, definicja d jest w nowym wektorze RDEFEX(j), $1 \leqslant j \leqslant n_J$, wtedy i tylko wtedy, gdy d była w starym wektorze RDEFEX(j) lub, gdy $d \in$ RDEFEN(1) oraz istnieje V-wolna droga od wierzchołka 1 do wyjścia wierzchołka j.

Dowód. W dowodzie można przeprowadzić podobne rozumowanie jak dla Lematu 3.9.2.□

Lematy 3.9.1 i 3.9.2 dowodzą Twierdzenia 3.9.□

Określimy obecnie złożoność Algorytmu 3.2B. Niech n_J jest liczbą wierzchołków w R_J, n_J^{ϕ} - liczbą wierzchołków w R_J z pustym zbiorem następników, n_J^h - liczbą głów interwałów pierwszego rzędu w R_J oraz n_J^{ph} - liczbą ich bezpośrednich poprzedników.

Twierdzenie 3.10. Wykonanie Algorytmu 3.2B wymaga realizacji

$$4n_J - 3n_J^h - 2n_J^{\phi} + n_J^{ph}$$

operacji na wektorach bitów.

Dowód. Aby obliczyć wektor RDEFEN = T (krok B1) należy wykonać dla każdej głowy interwału o jedną operację na wektorach bitów mniej niż liczba krawędzi dochodzących do głowy interwału. Liczba krawędzi dochodzących do głów interwałów w regionie R_J wynosi n_J^{ph}, a więc aby obliczyć wektory RDEFEN = T dla wszystkich głów w R_J należy wykonać $n_J^{ph} - n_J^h$ operacji. Do obliczenia wektora RDEFEN (krok B3) niezbędne jest zrealizowanie dwóch operacji dla każdego wierzchołka różnego od głowy, co daje w sumie $2(n_J - n_J^h)$ operacji. W końcu, obliczenie wektora RDEFEX (kroki B2 i B4) wymaga zrealizowania dwóch operacji dla każdego wierzchołka w R_J, z niepustym zbiorem następników, bądź sumarycznie $2(n_J - n_J^{\phi})$ operacji. Tak więc wykonanie Algorytmu 3.2B wymaga zrealizowania

$$(n_J^{ph} - n_J^h) + [2(n_J - n_J^h)] + [2(n_J - n_J^{\phi})] = 4n_J - 3n_J^h - 2n_J^{\phi} + n_J^{ph}$$

operacji na wektorach bitów.□

Kompletny algorytm analizy regionów dla rozwiązania problemu definicji osiągających jest następujący.

Algorytm 3.2C: Algorytm analizy regionów dla problemu definicji osiągających.

Dane: 1. Redukowalny graf przepływu $G = (N, \Gamma, s)$.
2. Sekwencja grafów pochodnych $(G = G_0, G_1, \ldots, G_m)$ dla G, gdzie G_m jest trywialnym grafem przepływu.
$G_i = (N_i, \Gamma_i, s_i)$.
3. Zawartości każdego regionu w grafie przepływu G oraz każdego interwału w grafach pochodnych podane w porządku interwałowym.
4. Wektory TRANS(j) i GEN(j) dla każdego $j \in N$.

Wyniki: Wektory RDEFEN(j) dla każdego $j \in N$.

Metoda:

```
begin
  -- Faza 1: Przetworzyć interwały pierwszego rzędu w G.
  for każdego wierzchołka J w G1 do
      Zastosować Algorytm 3.2A do obliczenia wektorów RDEFEN, RDEFEX,
      PATHEN i PATHEX dla regionu RJ (lub inaczej, dla interwału I = RJ).
  endfor;
  -- Faza 2: Przetwarzając sekwencję grafów pochodnych poczynając od re-
  -- gionów wewnętrznych do zewnętrznych, stosować Algorytm 3.2B.
  for i from 1 to m-1 do
      C1: for każdego wierzchołka J różnego od głowy interwału Gi, w po-
              rządku interwałowym do
          Zastosować Algorytm 3.2B do uaktualnienia wektorów RDEFEN i
          RDEFEX dla regionu RJ.
      endfor
  endfor;
  C2: Zastosować Algorytm 3.2B do ostatecznego uaktualnienia wektorów
      RDEFEN i RDEFEX dla regionu Rsm = G.
end □
```

Twierdzenie 3.11. Algorytm 3.2C jest częściowo poprawny oraz jego obliczenie dochodzi do punktu końcowego.

Dowód: Wszystkie pętle Algorytmu 3.2C są wykonywane skończoną liczbę razy, a więc dochodzenie obliczenia algorytmu do punktu końcowego jest ewidentne. Poprawność wynika z poprawności Algorytmów 3.2A i 3.2B wywoływanych w trakcie wykonywania Algorytmu 3.2C oraz z nastpujących obserwacji:

1. Ilekroć Algorytm 3.2B jest stosowany dla regionu R_J w G (krok C1), zbiory definicji, które osiągają każdy wierzchołek R_J, przy uwzględnieniu "wpływów" wszystkich krawędzi, z wyjątkiem krawędzi zamykających regionu R_J, zostały już obliczone. Wynika to stąd, że "podregiony" R_J były przetwarzane w porządku interwałowym w trakcie przetwarzania grafu niższego rzędu.

2. Algorytm 3.2B wyznacza zbiory definicji osiągające każdy wierzchołek w regionie R_J biorąc pod uwagę krawędzie zamykające R_J.

3. Ponieważ ostatnim przetwarzanym regionem jest graf przepływu G (krok C2), to Algorytm 3.2C oblicza osiągalność definicji dla grafu G.□

Twierdzenie 3.12. Wykonanie Algorytmu 3.2C wymaga realizacji co najwyżej

$$2e_o^f + n_o - 3n_o^\phi - n_1 + \sum_{i=1}^{m} \sum_{J \in N_i} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph})$$

operacji na wektorach bitów[1].

Dowód. Na podstawie Twierdzenia 3.8, każde wykonanie Algorytmu 3.2A w fazie 1 wymaga zrealizowanie $2e_I^f + n_I - 3n_I^\phi - 1$ operacji na wektorach bitów. Ponieważ w fazie 1, Algorytm 3.2A wywoływany jest jednokrotnie dla każdego interwału (regionu) $I \in N_1$, to aby wykonać fazę 1 należy zrealizować

$$\sum_{i \in N_1} (2e_I^f + n_I - 3n_I^\phi - 1) = 2e_o^f + n_o - 3n_o^\phi - n_1$$

operacji. Zgodnie z Twierdzeniem 3.10, dla każdorazowego wykonania Algorytmu 3.2B niezbędne jest zrealizowanie $4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}$ operacji. W kroku C1 Algorytm 3.2B wywoływany jest co najwyżej raz dla każdego regionu $J \in N_i$, a więc w trakcie wykonywania kroku C1 realizowanych jest co najwyżej

$$\sum_{J \in N_i} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph})$$

operacji. Ponieważ krok C1 jest powtarzany raz dla każdego i, $1 \leqslant i \leqslant m-1$ zaś w trakcie realizacji kroku C2 wykonuje się co najwyżej

$$\sum_{J \in N_m} (4n_J - 2n_J^h - 2n_J^\phi + n_J^{ph})$$

operacji, to aby wykonać fazę 2 trzeba wykonać

$$\sum_{i=1}^{m-1} \sum_{J \in N_i} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}) + \sum_{J \in N_m} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}) =$$

$$= \sum_{i=1}^{m} \sum_{J \in N_i} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph})$$

[1] e_o^f, n_o, n_o^ϕ oznaczają liczby odpowiednich krawędzi lub wierzchołków w oryginalnym grafie przepływu G, zaś n_1 oznacza liczbę wierzchołków w grafie pochodnym pierwszego rzędu.

operacji. Stąd, maksymalna liczba operacji wymagana dla wykonania algorytmu analizy Regionów (Algorytmu 3.2C) jest

$$T_R = 2e_o^f + n_o - 3n_o^\phi - n_1 + \sum_{i=1}^{m} \sum_{J \in N_i} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}).\square$$

Twierdzenie 3.13. Dla grafu przepływu o n wierzchołkach, wykonanie algorytmu analizy regionów dla problemu definicji osiągających (Algorytmu 3.2C) wymaga realizacji w najgorszym przypadku co najwyżej $2n^2 + 2n + 7$ operacji na wektorach bitów.

Dowód. Na podstawie Twierdzenia 3.12

$$T_R = 2e^f + n - 3n^\phi - n_1 + \sum_{i=1}^{m} \sum_{J \in N_i} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}) \leqslant 2e^f + n -$$

$$- n_1 + \sum_{i=1}^{m} \sum_{J \in N_i} (4n_J - 3n_J^h + n_J^{ph}).$$

(Dla uproszczenia przyjmujemy oznaczenia $e_o^f = e^f$, $n_o = n$,..., itd.) Wyrazimy najpierw T_R jako funkcję m, n_1 i n, a następnie znajdziemy maksimum tej funkcji względem m i n_1. Pomocne przy tym będą następujące oszacowania. Liczba krawędzi w przód grafu G, e^f, jest co najmniej równa liczbie "wewnętrznych" wierzchołków w interwałach G, tj. równa $n-n_1$, ponieważ co najmniej jedna krawędź w przód dochodzi do każdego wewnętrznego wierzchołka w interwale. Określimy teraz oszacowanie od góry wielkości e^f. Z równania $e = e^f + e^o + e^b$ mamy $e^f = e - (e^o + e^b)$. Ponieważ co najmniej dwie krawędzie wyjściowe lub zamykające muszą dochodzić do każdej głowy interwału, z wyjątkiem wierzchołka początkowego G, to $e^o + e^b \geqslant$ $\geqslant 2(n_1-1)$. Ostatecznie, oszacowania e^f są następujące:

$$n-n_1 \leqslant e^f \leqslant e-2(n_1-1) \leqslant 2n-2n_1+2. \tag{3.2}$$

(Zakładamy, że dla rzadkiego grafu przepływu prawdziwa jest nierówność $e \leqslant 2n$). Maksymalna długość sekwencji grafów pochodnych dla G jest równa liczbie interwałów pierwszego rzędu w G, tj. $m \leqslant n_1$. Ponieważ liczba interwałów pierwszego rzędu w G jest równa co najwyżej n-1, mamy

$$1 \leqslant m \leqslant n_1 \leqslant n-1. \tag{3.3}$$

Liczba wierzchołków G przetwarzanych przez Algorytm 3.2B podczas analizy i-tego grafu pochodnego jest równa co najwyżej n-m+i. Wynika to stąd, że liczba regionów G nie przetwarzanych przez Algorytm 3.2B (zauważmy, że

regiony takie są reprezentowane przez głowy interwałów w grafach pochodnych) jest równa 0 dla granicznego (trywialnego) grafu przepływu ($i = m$), jest równa co najmniej 1 dla grafu pochodnego rzędu ($m-1$), co najmniej 2 dla grafu pochodnego rzędu ($m-2$), itd. Ponieważ każdy nieprzetwarzany region składa się z co najmniej jednego wierzchołka, to

$$\sum_{J \in N_i} n_J \leqslant n - (m-i) = n - m + i.$$

Rozważmy oszacowanie od dołu wielkości $\sum_{J \in N_i} n_J^h$, gdzie n_J^h oznacza liczbę głów interwałów pierwszego rzędu w regionie R_J. W najgorszym przypadku, sekwencja grafów pochodnych dla G jest taka, że każdy graf pochodny rzędu ($i+1$) ma o jeden wierzchołek mniej, niż graf pochodny rzędu i. W tym przypadku, tylko jeden region G jest przetwarzany przez Algorytm 3.2B podczas analizy każdego grafu pochodnego oraz dla $i = 1$, przetwarzany region zawiera dokładnie jedną głowę interwału, dla $i = 2$ - dokładnie dwie głowy interwałów, itd. Stąd, w najgorszym przypadku

$$\sum_{J \in N_i} n_J^h \geqslant i.$$

Określmy oszacowanie od góry wielkości $\sum_{J \in N_i} n_J^{ph}$, gdzie n_J^{ph} jest liczbą bezpośrednich poprzedników głów interwałów w regionie R_J. Przed chwilą stwierdziliśmy, że liczba regionów G nie przetwarzanych przez Algorytm 3.2B jest równa 0 dla granicznego grafu przepływu, jest równa co najmniej 1 dla grafu pochodnego rzędu ($m-1$), co najmniej 2 dla grafu pochodnego rzędu ($m-2$), itd. Zakładając, że każdy nieprzetwarzany region zawiera co najmniej jedną głowę interwału oraz że każda głowa interwału, z wyjątkiem wierzchołka początkowego G, ma co najmniej dwa bezpośrednie poprzedniki, możemy przyjąć, że maksymalna liczba bezpośrednich poprzedników głów interwałów, które są przetwarzane przez Algorytm 3.2B, jest równa liczbie wszystkich bezpośrednich poprzedników głów interwałów w G minus 2 dla grafu pochodnego rzędu ($m-2$), minus 4 dla grafu pochodnego rzędu ($m-3$), itd. Uwzględniając dodatkowo (3.2), mamy dla $1 \leqslant i \leqslant m-2$

$$\sum_{J \in N_i} n_J^{ph} \leqslant e - e^f - 2(m-1-i) \leqslant 2n - (n-n_1) - 2m + 2i + 2 =$$
$$= n + n_1 - 2m + 2i + 2$$

oraz dla $m-1 \leqslant i \leqslant m$

$$\sum_{J \in N_i} n_J^{ph} \leqslant e - e^f \leqslant 2n - (n-n_1) = n + n_1.$$

Biorąc pod uwagę wszystkie uprzednie oszacowania otrzymujemy

$$T_R \leqslant 2e^f + n - n_1 + \sum_{i=1}^{m} \sum_{J \in N_i} (4n_J - 3n_J^h + n_J^{ph}) \leqslant 2(2n - 2n_1 + 2) +$$

$$+ n - n_1 + \sum_{i=1}^{m-2} [4(n-m+i) - 3i + n + n_1 - 2m + 2i + 2] +$$

$$+ \sum_{i=m-1}^{m} [4(n-m+i) - 3i + n + n_1] = 5n - 5n_1 + 4 +$$

$$+ \sum_{i=1}^{m-2} (5n - 6m + n_1 + 2 + 3i) + \sum_{i=m-1}^{m} (5n - 4m + n_1 + i) =$$

$$= 5n - 5n_1 + 4 + (5n - 6m - n_1 + 2)(m-2) + 3\,\frac{1+m-2}{2}\,(m-2) +$$

$$+ 2(5n - 4m + n_1) + \frac{m-1+m}{2}\,2 = -4{,}5\,m^2 + (n_1+5n+3{,}5)m - 5n_1 + 5n + 2 \tag{3.4}$$

Traktując funkcję T_R jako ciągłą znajdziemy jej maksimum ze względu na zmienne m i n_1 spełniające ograniczenia nierównościowe (3.3). Niech $x' = [m, n_1]$ (symbol "prim" oznacza transpozycję wektorów i macierzy), $T(x) = -T_R(x)$, $g_1(x) = 1-m$, $g_2(x) = m-n_1$, $g_3(x) = n_1-n+1$ oraz $g'(x) = [g_1(x), g_2(x), g_3(x)]$. Mamy do czynienia z następującym problemem programowania nieliniowego (Mangasarian [1969], Gabasow i Kirillowa [1975]): Znaleźć takie $\bar{x}$, jeśli istnieje, że[1)]

$$T(\bar{x}) = \min_{x \in X} T(x), \quad \bar{x} \in X = \{x \mid x \in R^2,\ g_i(x) \leqslant 0,\ i = 1,2,3\}. \tag{3.5}$$

Zbiór X zwany jest <u>zbiorem punktów dopuszczalnych</u>, $\bar{x}$ - <u>punktem minimum</u>, zaś $T(\bar{x})$ - <u>wartością minimum</u>. <u>Funkcja Lagrange'a</u> związana z tym problemem zdefiniowana jest następująco[2)]:

---

1) R jest zbiorem liczb rzeczywistych, zaś R^2 jest <u>produktem</u> zdefiniowanym jako $R^2 = R \times R = \{(x,y) \mid x \in R,\ y \in R\}$. R^2 wyznacza więc płaszczyznę Euklidesową (zbiór wszystkich uporządkowanych par liczb rzeczywistych).

2) Dla dwóch wektorów $u' = [u_1, u_2, \ldots, u_n]$ i $z' = [z_1, z_2, \ldots, z_n]$ definiujemy ich iloczyn skalarny jak zwykle jako $u'z = \sum_{i=1}^{n} u_i z_i$. Ponadto $u \leqslant z$ oznacza, że zachodzi $u_i \leqslant z_i$ dla $i = 1,2,\ldots,n$.

$$L(x,y) = T(x) + y'g(x)$$

gdzie $y' = [y_1, y_2, y_3]$ jest wektorem nieujemnych mnożników Lagrange'a. Warunkiem koniecznym na to, aby $\bar{x}$ było punktem minimum problemu (3.5) jest istnienie pewnego $\bar{y}$ takiego, że $\bar{x}$ oraz $\bar{y}$ spełniają warunki[1]:

$$\nabla_x L(\bar{x},\bar{y}) = 0, \quad \bar{y}'g(\bar{x}) = 0, \quad g(\bar{x}) \leqslant 0 \quad \text{i} \quad \bar{y} \geqslant 0.$$

W naszym przypadku warunki te mają następującą postać:

$$9m - n_1 - 5n - 3{,}5 - y_1 + y_2 = 0,$$

$$-m + 5 - y_2 + y_3 = 0,$$

$$y_1(1-m) = 0, \quad y_2(m-n_1) = 0, \quad y_3(n_1-n+1) = 0, \tag{3.6}$$

$$1 - m \leqslant 0, \quad m-n_1 \leqslant 0, \quad n_1 - n + 1 \leqslant 0 \quad \text{oraz}$$

$$y_i \geqslant 0, \quad i = 1,2,3.$$

Istnieją cztery punkty $(\hat{x},\hat{y})$, zwane <u>punktami stacjonarnymi</u>, które spełniają warunki (3.6). Charakterystyka tych punktów została przedstawiona w tablicy 3.1. Można pokazać, że punkt $\hat{x}^4$ nie spełnia warunku wystarczającego optymalności (patrz Dodatek A), nie jest więc punktem minimum dla problemu (3.5)[2]. Z pozostałych punktów, $\hat{x}^3$ wyznacza górną granicę funkcji T_R. W dalszym ciągu udowodnimy, że punkt ten spełnia warunek wystarczający optymalności oraz określimy maksimum T_R, tj. minimum T. (W Dodatku A przedstawiono uzupełniającą dyskusję pozostałych punktów stacjonarnych).

Warunkiem wystarczającym na to, aby punkt stacjonarny x był punktem minimum dla problemu (3.5) (Gabasow i Kirillowa [1975]), tj. $\hat{x} = \bar{x}$ jest aby następująca forma kwadratowa była dodatnio określona[3]

---

[1] Niech $f(x,y)$ jest funkcją zdefiniowaną na zbiorze $Y \subset R^n \times R^k$ i różniczkowalną w punkcie $(\tilde{x},\tilde{y}) \in Y$. Definiujemy

$$\nabla_x f(\tilde{x},\tilde{y}) = \left[\frac{\partial f(\tilde{x},\tilde{y})}{\partial x_1}, \frac{\partial f(\tilde{x},\tilde{y})}{\partial x_2}, \ldots, \frac{\partial f(\tilde{x},\tilde{y})}{\partial x_n}\right].$$

[2] Dla uproszczenia, punkt stacjonarny $(\hat{x}^i,\hat{y}^i)$ oznaczamy przez $\hat{x}^i$.

[3] Niech $f(x,y)$ jest funkcją zdefiniowaną na zbiorze $Y \subset R^n \times R^k$ i dwukrotnie różniczkowalną w punkcie $(\tilde{x},\tilde{y}) \in Y$. Macierz $\nabla^2_{xx} f(\tilde{x},\tilde{y})$ o rozmiarach $n \times n$, zdefiniowana jako

$$\left[\nabla^2_{xx} f(\tilde{x},\tilde{y})\right]_{ij} = \frac{\partial^2 f(\tilde{x},\tilde{y})}{\partial x_i \, \partial x_j} \quad i,j = 1,2,\ldots,n$$

zwana jest <u>hessianem (macierzą Hessego) funkcji f w punkcie $(\tilde{x},\tilde{y})$</u>.

Tablica 3.1

Charakterystyka punktów spełniających warunki konieczne minimum dla problemu (3.5)

Nr punktu	$\hat{x}$: $\hat{m}$	$\hat{x}$: $\hat{n}_1$	Mnożniki Lagrange'a: $\hat{y}_1$	$\hat{y}_2$	$\hat{y}_3$	Aktywne ograniczenia	Dodatkowe warunki na n[1]	$T_R(\hat{x})$
1	n-1	n-1	0	-3n+11.5	-2n+5.5	$m-n_1 \leqslant 0$, $n_1-n+1 \leqslant 0$	$2 \leqslant n \leqslant 2.75$	$1.5n^2 + 5.5n$
2	$\frac{5n-1.5}{7}$	$\frac{5n-1.5}{7}$	0	$\frac{36.5-5n}{7}$	0	$m-n_1 \leqslant 0$	$2.75 \leqslant n \leqslant 7.3$	$\frac{25n^2+55n+30.25}{14}$
3	$\frac{6n+2.5}{9}$	n-1	0	0	$\frac{6n-42.5}{9}$	$n_1-n+1 \leqslant 0$	$7.083 \leqslant n$	$\frac{36n^2+30n+132.25}{18}$
4	5	-5n+41.5	0	0	0	-	$7.083 \leqslant n \leqslant 7.3$	30n-93

[1] Wynikają one z ograniczeń $1 \leqslant \hat{m} \leqslant \hat{n}_1 \leqslant n-1$ oraz $\hat{y}_1 \geqslant 0$, $\hat{y}_2 \geqslant 0$, $\hat{y}_3 \geqslant 0$.

$$z' \nabla^2_{xx} L(\bar{x},\bar{y}) z > 0 \tag{3.7}$$

na hiperpłaszczyźnie[1)]

$$\nabla g_i(\bar{x}) z = 0, \quad i = i_1, i_2, \ldots, i_k, \quad \|z\| \neq 0 \tag{3.8}$$

gdzie $i_1, i_2, \ldots, i_k$ są numerami aktywnych ograniczeń w punkcie $\bar{x}$[2)].

Jedynym aktywnym ograniczeniem dla punktu $\hat{x}^3$ jest $n_1 - n - 1 \leqslant 0$, stąd na podstawie (3.8) mamy

$$[0, \quad -1] \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = 0 \quad \text{oraz} \quad z_2 = 0$$

Uwzględniając ten rezultat, wyznaczamy formę kwadratową (3.7):

$$[z_1, z_2] \begin{bmatrix} 9 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = 9z_1^2 - 2z_1 z_2 = 9z_1^2.$$

Forma ta jest dodatnio określona dla każdego $z_1 \neq 0$, a więc warunek wystarczający minimum jest spełniony.

Aby otrzymać wartość maksymalną funkcji T_R podstawiamy

$$\hat{m}^3 = \bar{m}^3 = \frac{6n+2.5}{9} \quad \text{oraz} \quad \hat{n}_1^3 = \bar{n}_1^3 = n-1$$

do (3.4):

$$T_R = 4.5\left(\frac{6n+2.5}{9}\right)^2 + (n-1+5n+3.5)\frac{6n+2.5}{9} - 5n+5+5n+2 = \frac{(6n+2.5)^2}{18} + 7 =$$

$$= \frac{36n^2+30n+132.25}{18}.$$

Dla $n > 1$ zachodzi

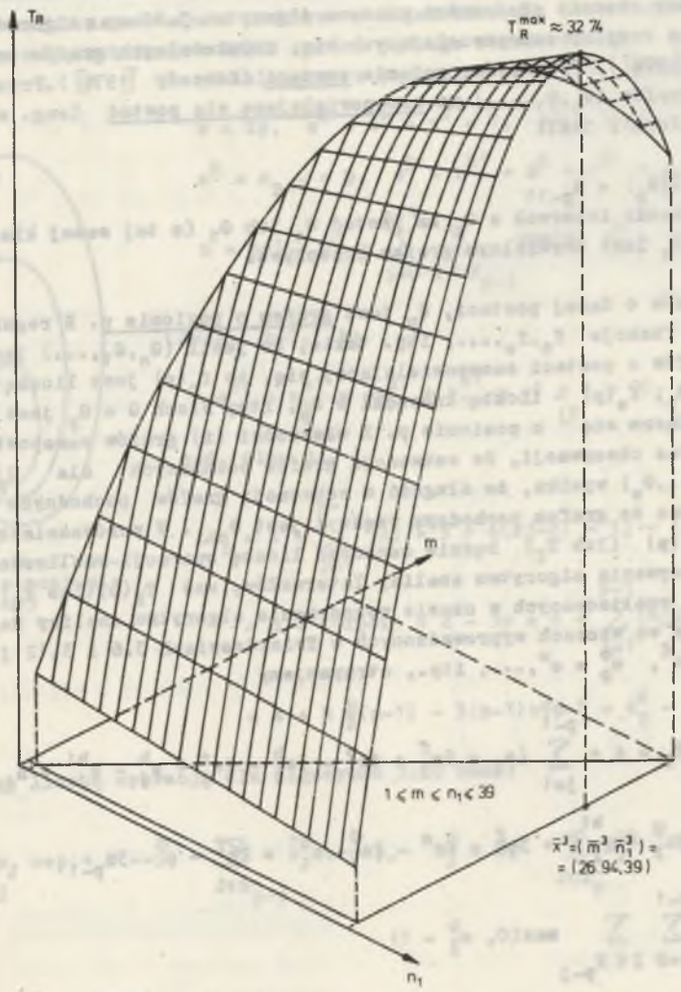
$$T_R = \frac{36n^2 + 30n + 132.25}{18} < 2n^2 + 2n + 7. \square$$

Na rys. 3.1 przedstawiono kształt funkcji T_R dla $n = 40$.

---

1) Norma $\|z\|$ wektora $z \in R^n$ jest zdefiniowana jako $\|z\|^2 = z'z$.

2) Ograniczenie $g_i(x) \leqslant 0$ jest <u>aktywne</u> w punkcie $\bar{x}$, jeśli $g_i(\bar{x}) = 0$; ograniczenie $g_j(x) \leqslant 0$ jest <u>nieaktywne</u> w punkcie $\bar{x}$, jeśli $g_j(\bar{x}) < 0$.



Rys. 3.1. Postać funkcji T_R dla $n = 40$

<u>Wniosek 3.1</u>. Pesymistyczna złożoność czasowa Algorytmu 3.2C jest $O(n^2)$.

<u>Dowód</u>. Wniosek wynika bezpośrednio z Twierdzenia 3.13.□

3.4. Porównanie algorytmów

Porównamy obecnie złożoności czasowe Algorytmu 3.1C oraz Algorytmu 3.2C w oparciu o rodziny samopowielających się, redukowalnych grafów przepływu (Kennedy [1976]). Wprowadźmy pojęcie <u>postaci</u> (Kennedy [1976]). Przeliczalna klasa grafów $(G_0, G_1, \ldots)$ ma <u>samopowielającą się postać</u> (ang. self-replicating form), jeśli

(i) $I(G_p) = G_{p-1}$,
(ii) każdy interwał w G_p ma postać G_0 lub G_1 (z tej samej klasy),
(iii) G_0 jest trywialnym grafem przepływu.

Dla grafów o danej postaci, G_p jest <u>grafem o poziomie p</u>. Z reguły można znaleźć funkcje $f_n, f_e, \ldots$, itp. takie, że jeśli $(G_0, G_1, \ldots)$ jest sekwencją grafów o postaci samopowielającej się, to $f_n(p)$ jest liczbą wierzchołków w G_p, $f_e(p)$ - liczbą krawędzi w G_p, itp. Niech $G = G_p$ jest grafem sampowielającym się[1] o poziomie p. Z własności (i) grafów samopowielających się oraz obserwacji, że sekwencją grafów pochodnych dla G_p jest $(G_p, G_{p-1}, \ldots, G_0)$ wynika, że długość n sekwencji grafów pochodnych dla G_p wynosi p oraz że grafem pochodnym rzędu i jest G_{p-i}. W porównaniach algorytmów, $T_I(p)$ (lub T_I) będzie oznaczać liczbę operacji realizowanych w czasie wykonywania algorytmu analizy Interwałów, zaś $T_R(p)$ (lub T_R) liczbę operacji realizowanych w czasie wykonywania algorytmu analizy Regionów. Podstawiając we wzorach wyprowadzonych w Twierdzeniach 3.6 i 3.12 $j = p-i$ oraz $e_p^f = e^f$, $e_p^o = e^o, \ldots$, itp., otrzymujemy

$$T_I(p) = T_I = 4 + \sum_{j=1}^{p-1} (e_j + 5e_j^f + 4e_j^o - 3e_j^h - 2e_j^\phi - e_j^b - e_j^{b1} + n_j +$$

$$+ 2n_j^\phi + n_j^{b1}) + 3e^f + 4e^o - 2e^\phi - e^{b1} + 2n^f - n^\phi - 3n_{p-1} +$$

$$+ \sum_{j=0}^{p-1} \sum_{I \in N_{p-j}} \max(0, e_I^b - 1)$$

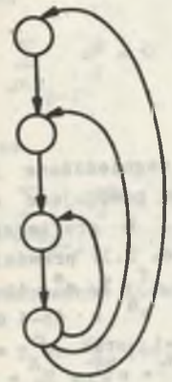
[1] Dla uproszczenia będziemy używać określenia "graf samopowielający się" zamiast określenia ściślejszego, lecz dłuższego: "graf z klasy grafów o postaci samopowielającej się".

oraz

$$T_R(p) = T_R = \sum_{j=0}^{p-1} \sum_{J \in N_{p-j}} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}) + 2e^f + n - 3n^\phi - n_{p-1}.$$

3.4.1. Graf muszelkowy (ang. seashell)

Graf muszelkowy powstaje w praktyce z zagnieżdżonych instrukcji iteracyjnych nie mających innych wyjść oprócz jednego standardowego. Dla grafu muszelkowego o poziomie p, $p > 1$, (rys. 3.2),



Rys. 3.2. Graf muszelkowy o poziomie p = 3

$$e = 2p, \quad e^f = e^b = n^f = 1, \quad e^o = 2p-2,$$

$$e^h = n_{p-1} = p, \quad e^\phi = e^{b1} = n^\phi = n^{b1} = 0,$$

$$n = p+1, \quad T_I' = \sum_{j=0}^{p-1} \sum_{I \in N_{p-j}} \max(0, e_I^b - 1) = 0.$$

(Jeśli p=1, to

$$-e_p^{b1} + n_p^{b1} = -e_1^{b1} + n_1^{b1} = -1 + 1 = 0).$$

tąd, dla $p > 1$

$$T_I = 4 + \sum_{j=1}^{p-1} [2j + 5 + 4(2j-2) - 3j - 1 + j + 1] +$$

$$+ 3 + 4(2p-2) + 2 - 3p = 4 + \sum_{j=1}^{p-1} (8j-3) + 5p - 3 =$$

$$= 4 + 8\tfrac{p}{2}(p-1) - 3(p-1) + 5p - 3 = 4p^2 - 2p + 4.$$

Obliczając liczbę operacji dla Algorytmu 3.2C mamy:

$$\sum_{J \in N_{p-j}} n_J = p + 1 - j, \quad \sum_{J \in N_{p-j}} n_J^h = p-j, \quad n_J^\phi = 0, \quad \sum_{J \in N_p} n_J^{ph} = 2p-1$$

oraz

$$\sum_{J \in N_{p-j}} n_J^{ph} = 2(p-j) \quad \text{dla} \quad j > 0,$$

a więc

$$T_R = \sum_{J\in N_p} (4n_J - 3n_J^h + n_J^{ph}) + \sum_{j=1}^{p-1} \sum_{J\in N_{p-j}} (4n_J - 3n_J^h + n_J^{ph}) + 2 + p + 1 - p =$$

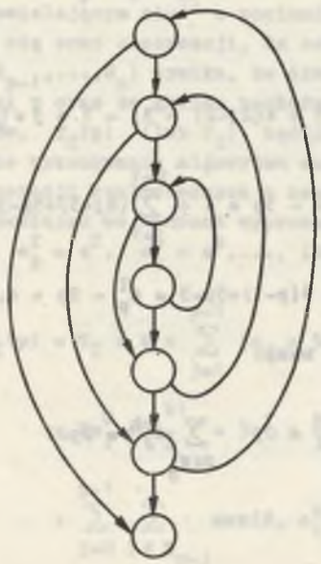
$$= 4(p+1) - 3p + 2p - 1 + \sum_{j=1}^{p-1} [4(p+1-j) - 3(p-j) + 2(p-j)] + 3 =$$

$$= 3p + 6 + \sum_{j=1}^{p-1} (3p+4-3j) = 3p + 6 + (3p+4)(p-1) - 3\frac{p}{2}(p-1) =$$

$$1.5\,p^2 + 5.5\,p + 2.$$

3.4.2. Graf spiralny (ang. spiral)

Graf spiralny powstaje dla programów zawierających zagnieżdżone instrukcje iteracyjne, w których występują badania warunków powodujące dodatkowe wyjścia z pętli. W grafie spiralnym o poziomie p (rys. 3.3) prawdziwe są zależności, $e = 4p$, $e^f = 3$, $e^o = 4p-4$, $e^h = 2$ dla $p=1$ oraz $e^h = 4p-4$ dla $p > 1$, $e^\phi = 2$, dla $p=1$ oraz $e^\phi = 0$ dla $p > 1$, $e^b = 1$, $e^{b1} = n^{b1} = T_I' = 0$, $n = 2p+1$, $n^f = 2$, $n^\phi = 1$ dla $p=1$ oraz $n^\phi = 0$ dla $p > 1$, $n_{p-1} = 2p-1$. Stąd, dla $p > 1$



Rys. 3.3. Graf spiralny o poziomie p = 3

$$T_I = 4 + \sum_{j=1}^{1} [4j + 15 + 4(4j-4) - 6 -$$

$$- 4 - 1 + 2j + 1 + 2] + \sum_{j=2}^{p-1} [4j+15+$$

$$+ 4(4j-4) - 3(4j-4) - 1 + 2j + 1] +$$

$$+ 9 + 4(4p-4) + 4 - 3(2p-1) =$$

$$= 4 + 13 + \sum_{j=2}^{p-1} (10j+11) + 10p =$$

$$= 5p^2 + 16p - 15.$$

Analizując Algorytm 3.2C mamy zależności

$$\sum_{J\in N_p} n_J = 2p+1 \quad \text{oraz} \quad \sum_{J\in N_{p-j}} n_J = 2(p+1-j) \quad \text{dla} \quad j > 0, \quad \sum_{J\in N_p} n_J^h = 2p-1$$

oraz

$$\sum_{j\in N_{p-1}} n_J^h = 2(p-j) \quad \text{dla} \quad j > 0, \quad \sum_{J\in N_{p-j}} n_J^\phi = 1 \quad \text{dla} \quad j < 2$$

oraz

$$\sum_{J\in N_{p-j}} n_J^\phi = 0 \quad \text{dla} \quad j \geqslant 2, \quad \sum_{J\in N_p} n_J^{ph} = 4p-3 \quad \text{dla} \quad j = 0$$

oraz

$$\sum_{J\in N_{p-j}} n_J^{ph} = 4(p-j) \quad \text{dla} \quad j > 0.$$

Oszacowanie liczby operacji dla Algorytmu 3.2C jest ($p > 1$).

$$T_R = \sum_{J\in N_p} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}) + \sum_{J\in N_{p-1}} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}) +$$

$$+ \sum_{j=2}^{p-1} \sum_{J\in N_{p-j}} (4n_J - 3n_J^h - 2n_J^\phi + n_J^{ph}) + 6 + 2p + 1 - (2p-1) =$$

$$= [4(2p+1) - 3(2p-1) - 2 + 4p - 3] + [8p - 6(p-1) - 2 + 4(p-1)] +$$

$$+ \sum_{j=2}^{p-1} [8(p+1-j) - 6(p-j) + 4(p-j)] + 8 = 12p + 10 + \sum_{j=2}^{p-1} (6p+8-6j) =$$

$$= 12p + 10 + (6p+8)(p-2) - 6\frac{p+1}{2}(p-2) = 3p^2 + 11p.$$

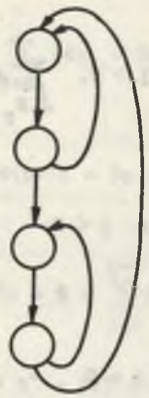
3.4.3. Graf binarny Hechta-Ullmana

Dla grafu tego o poziomie p, $p > 1$, (rys. 3.4), mamy następujące zależności:

$$e = 2^{p+1} - 2, \quad e^f = e^h = e^b = n^f = n_{p-1} = 2^{p-1}, \quad e^o = 2^p - 2,$$

$$e^\phi = e^{b1} = n^\phi = n^{b1} = T_I' = 0, \quad n = 2^p. \quad (\text{Jeśli } p=1, \text{ to } -e_p^{b1} + n_p^{b1} =$$

$$= -e_1^{b1} + n_1^{b1} = -1 + 1 = 0).$$



Rys. 3.4. Graf binarny Hechta-Ullmana o poziomie p = 2

Tak więc, dla Algorytmu 3.1C otrzymujemy (p > 1)

$$T_I = 4 + \sum_{j=1}^{p-1}\left[2^{j+1} - 2 + 5\cdot 2^{j-1} + 4(2^j-2) - 3\cdot 2^{j-1} - 2^{j-1} + 2^j\right] + 3\cdot 2^{p-1} + 4(2^p-2) + 2\cdot 2^{p-1} - 3\cdot 2^{p-1} = 4 + \sum_{j=1}^{p-1}(15\cdot 2^{j-1}-10) + 10\cdot 2^{p-1} - 8 = 4 + 15\sum_{j=1}^{p-1} 2^{j-1} - 10(p-1) + 10\cdot 2^{p-1} - 8 = 15\sum_{j=0}^{p-2} 2^j + 10\cdot 2^{p-1} - 10p + 6 = 15(2^{p-1}-1) + 10\cdot 2^{p-1} - 10p + 6 = 25\cdot 2^{p-1} - 10p - 9.$$

Ponieważ dla Algorytmu 2.3C mamy

$$\sum_{J\in N_p} n_J = 2^p \quad \text{oraz} \quad \sum_{J\in N_{p-j}} n_J = 2^{p-1} \quad \text{dla } j > 0, \quad \sum_{J\in N_p} n_J^h + 2^{p-1}$$

oraz

$$\sum_{J\in N_{p-j}} n_J^h = 2^{p-2} \quad \text{dla } j > 0, \quad n_J^\phi = 0, \quad \sum_{J\in N_p} n_J^{ph} = 3\cdot 2^{p-1} - 2$$

oraz

$$\sum_{J\in N_{p-j}} n_J^{ph} = 3\cdot 2^{p-2} - 2^{j-1} \quad \text{dla } j > 0,$$

więc

$$T_R = \sum_{J\in N_p}(4n_J - 3n_J^h + n_J^{ph}) + \sum_{j=1}^{p-1}\sum_{J\in N_{p-j}}(4n_J - 3n_J^h + n_J^{ph}) + 2\cdot 2^{p-1} + 2^p - 2^{p-1} = 4\cdot 2^p - 3\cdot 2^{p-1} + 3\cdot 2^{p-1} - 2 + \sum_{j=1}^{p-1}(4\cdot 2^{p-1} - 3\cdot 2^{p-2} + 3\cdot 2^{p-2} - 2^{j-1}) + 3\cdot 2^{p-1} = 11\cdot 2^{p-1} - 2 + \sum_{j=1}^{p-1}(4\cdot 2^{p-1} - 2^{j-1}) =$$

$$= 11\cdot 2^{p-1} - 2 + 4(p-1)2^{p-1} - (2^{p-1}-1) = p\cdot 2^{p+1} + 3\cdot 2^p - 1.$$

3.4.4. Graf podwójnie wiązany (ang. double-chain)

Graf podwójnie wiązany powstaje z ciągu zagnieżdżonych instrukcji iteracyjnych while. Dla takiego grafu o poziomie p, p > 1, (rys. 3.5), mamy:

$$e = 2p, \quad e^f = e^b = n^f = 1, \quad e^o = 2p-2,$$

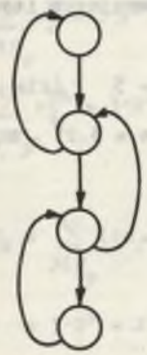
$$e^h = 2p-1, \quad e^\phi = e^{b1} = n^\phi = n^{b1} = T_I' = 0,$$

$$n = p+1, \quad n_{p-1} = p.$$

(Jeśli p=1, to $-e_p^{b1} + n_p^{b1} = -e_1^{b1} + n_1^{b1} = -1+1 = 0$).

Stąd, dla p > 1

$$T_I = 4 + \sum_{j=1}^{p-1}[2j + 5 + 4(2j-2) - 3(2j-1) - 1 + j + 1] + 3 + 4(2p-2) + 2 - 3p = 4 + \sum_{j=1}^{p-1} 5j + 5p - 3 = 2.5p^2 + 2.5p + 1.$$



Rys. 3.5. Graf podwójnie wiązany o poziomie p = 3

Dla Algorytmu 3.2C zachodzi:

$$\sum_{J\in N_{p-j}} n_J = p+1-j, \quad \sum_{J\in N_{p-j}} n_J^h = p-j, \quad n_J^\phi = 0, \quad \sum_{J\in N_p} n_J^{ph} = 2p-1$$

oraz

$$\sum_{J\in N_{p-j}} n_J^{ph} = 2(p-j) \quad \text{dla } j > 0,$$

więc

$$T_R = \sum_{J\in N_p}(4n_J - 3n_J^h + n_J^{ph}) + \sum_{j=1}^{p-1}\sum_{J\in N_{p-j}}(4n_J - 3n_J^h + n_J^{ph}) + 2+p+1-p =$$

$$= 4(p+1) - 3p + 2p-1 + \sum_{j=1}^{p-1}[4(p+1-j) - 3(p-j) + 2(p-j)] + 3 =$$

$$= 3p + 6 + \sum_{j=1}^{p-1}(3p+4-3j) = 3p + 6 + (3p+4)(p-1) - 3\frac{p}{2}(p-1) =$$

$$= 1.5p^2 + 5.5p + 2.$$

3.4.5. Graf rombowy (ang. diamond)

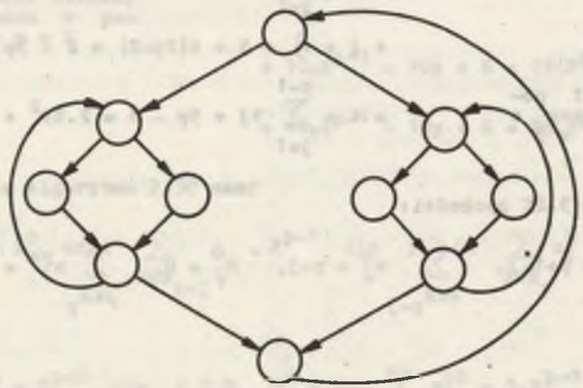
Graf rombowy, którego postać pokazana jest na rys. 3.6, powstaje z programu zawierającego instrukcje alternatywy if - then - else występujące wewnątrz instrukcji iteracyjnych, które z kolei są zagnieżdżone w instrukcjach alternatywy występujących w instrukcjach iteracyjnych itd. Dla grafu rombowego o poziomie p, $p > 1$, mamy

$$e = 5.2^p-5,\quad e^f = 2^{p+1},\quad e^o = 5.2^{p-1} - 5,\quad e^h = 3.2^p - 5 \quad (\text{dla } p=1,$$

$$e^h = 2),\quad e^\phi = e^{b1} = n^\phi = n^{b1} = T_I' = 0,\quad e^b = 2^{p-1},\quad n = 3.2^p - 2,$$

$$n^f = 3.2^{p-1},\quad n_{p-1} = 3.2^{p-1} - 2.$$

(Jeśli $p=1$, to $-e_p^{b1} + n_p^{b1} = -e_1^{b1} + n_1^{b1} = -2 + 1 = -1$).



Rys. 3.6. Graf rombowy o poziomie p = 2

Stąd przy $p > 1$, otrzymujemy

$$T_I = 4 - 3 - 1 + \sum_{j=1}^{p-1} \Big[5.2^j - 5 + 5.2^{j+1} + 4(5.2^{j-1} - 5) - 3(3.2^j - 5) -$$

$$- 2^{j-1} + 3.2^j - 2\Big] + 3.2^{p+1} + 4(5.2^{p-1} - 5) + 6.2^{p-1} - 3(3.2^{p-1}-2) =$$

$$= \sum_{j=1}^{p-1} (37.2^{j-1} -12) + 29.2^{p-1} - 14 = 37(2^{p-1} - 1) - 12(p-1) +$$

$$+ 29.2^{p-1} - 14 = 33.2^p - 12\,p - 39.$$

Ponieważ dla Algorytmu 3.2C spełnione są następujące zależności

$$\sum_{J\in N_p} n_J = 3\cdot 2^p-2 \quad \text{oraz} \quad \sum_{J\in N_{p-j}} n_J = 3(2^p-2^{j-1}) \quad \text{dla } j > 0,$$

$$\sum_{J\in N_p} n_J^h = 3\cdot 2^{p-1} - 2 \quad \text{oraz} \quad \sum_{J\in N_{p-j}} n_J^h = 3(2^{p-1} - 2^{j-1}) \quad \text{dla } j>0,\ n_J^\phi=0,$$

$$\sum_{J\in N_p} n_J^{ph} = 3\cdot 2^p - 5 \quad \text{oraz} \quad \sum_{J\in N_{p-j}} n_J^{ph} = 3(2^p-2^j) \quad \text{dla } j > 0, \quad \text{to}$$

$$T_R = \sum_{J\in N_p} (4n_J - 3n_J^h + n_J^{ph}) + \sum_{j=1}^{p-1} \sum_{J\in N_{p-j}} (4n_J - 3n_J^h + n_J^{ph}) + 2\cdot 2^{p+1} +$$

$$+ 3\cdot 2^p - 2 - 3\cdot 2^{p-1} + 2 = 4(3\cdot 2^p-2) - 3(3\cdot 2^{p-1}-2) + 3\cdot 2^p - 5 +$$

$$+ \sum_{j=1}^{p-1} \Big[12(2^p-2^{j-1}) - 9(2^{p-1} - 2^{j-1}) + 3(2^p-2^j)\Big] + 11\cdot 2^{p-1} =$$

$$= 16\cdot 2^p - 7 + \sum_{j=1}^{p-1} (21\cdot 2^{p-1} - 9\cdot 2^{j-1}) = 16\cdot 2^p - 7 + 21(p-1)2^{p-1} -$$

$$- 9(2^{p-1} - 1) = 21p\cdot 2^{p-1} + 2^p + 2.$$

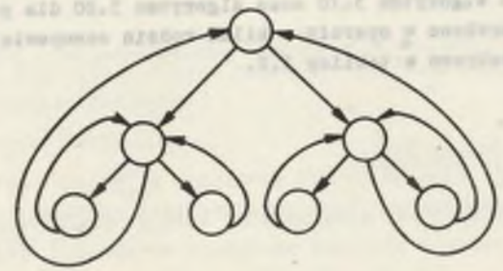
3.4.6. Graf niezwykły (ang. fan)

Graf niezwykły (rys. 3.7) powstaje przy programowaniu niestrukturalnym z użyciem instrukcji go to. Dla grafu niezwykłego o poziomie p, mamy:

$$e = 2^{p+2} - 4,\quad e^f = e^b = e^{b1} = n^{b1} = 2^p,\quad e^o = 2^{p+1} - 4,\quad e^h = 3\cdot 2^p-4,$$

$$e^\phi = n^\phi = 0,\quad n = 2^{p+1} - 1,\quad n^f = 2^{p-1},\quad n_{p-1} = 2^p - 1,$$

$$\sum_{I\in N_{p-j}} \max(0,\ e_I^b - 1) = 2^j, \quad \text{tak więc}$$



Rys. 3.7. Graf niezwykły o poziomie p = 2

$$T_I = 4 + \sum_{j=1}^{p-1} \left[2^{j+2} - 4 + 5.2^j + 4(2^{j+1} - 4) - 3(3.2^j - 4) - 2^j - 2^j + \right.$$

$$\left. + 2^{j+1} - 1 + 2^j\right] + 3.2^p + 4(2^{p+1} - 4) - 2^p + 2.2^{p-1} - 3(2^p-1) +$$

$$+ \sum_{j=0}^{p-1} 2^j = 4 + \sum_{j=1}^{p-1} (9.2^j - 9) + 8.2^p - 13 + 2^p - 1 = 9.2^p - 10 +$$

$$+ 18(2^{p-1} - 1) - 9(p-1) = 18.2^p - 9p - 19.$$

Dla Algorytmu 3.2C spełnione są zależności

$$\sum_{J\in N_{p-j}} n_J = 2^{p+1} - 2^j, \quad \sum_{J\in N_{p-j}} n_J^h = 2^p - 2^j, \quad n_J^{\emptyset} = 0, \quad \sum_{J\in N_p} n_J^{ph} = 3.2^p-4$$

oraz

$$\sum_{J\in N_{p-j}} n_J^{ph} = 3(2^p-2^j) \quad \text{dla} \quad j > 0,$$

a zatem

$$T_R = \sum_{J\in N_p} (4n_J - 3n_J^h + n_J^{ph}) + \sum_{j=1}^{p-1} \sum_{J\in N_{p-j}} (4n_J - 3n_J^h + n_J^{ph}) + 2.2^p +$$

$$+ 2^{p+1} - 1 - 2^p + 1 = 4(2^{p+1}-1) - 3(2^p-1) + 3.2^p - 4 +$$

$$+ \sum_{j=1}^{p-1} \left[4(2^{p+1} - 2^j) - 3(2^p-2^j) + 3(2^p-2^j)\right] + 3.2^p = 11.2^p - 5 +$$

$$+ 4 \sum_{j=1}^{p-1} (2^{p+1}-2^j) = 11.2^p - 5 + 4(p-1)2^{p+1} - 8(2^{p-1}-1) =$$

$$= p.2^{p+3} - 2^p + 3.$$

Wyniki porównań algorytmu 3.1C oraz Algorytmu 3.2C dla problemu definicji osiągających uzyskane w oparciu o kilka rodzin samopowielających się grafów przepływu zebrano w tablicy 3.2.

Tablica 3.2

Wyniki porównań algorytmów analizy interwałów oraz analizy regionów dla problemu definicji osiągających

Klasa G_p	$T_I(p)$	$T_R(p)$	Przebieg asymptotyczny $r(p) = T_R(p)/T_I(p)$
Muszelkowy	$4p^2 - 2p + 4$	$1.5p^2 + 5.5p + 2$	$r(p) \sim \frac{3}{8}$
Spiralny	$5p^2 + 16p - 15$	$3p^2 + 11p$	$r(p) \sim \frac{3}{5}$
Binarny Hechta--Ullmana	$25.2^{p-1} - 10p - 9$	$p.2^{p+1} + 3.2^p - 1$	$r(p) = O(p)$
Podwójnie wiązany	$2.5p^2 + 2.5p + 1$	$1.5p^2 + 5.5p + 2$	$r(p) \sim \frac{3}{5}$
Rombowy	$33.2^p - 12p - 39$	$21p.2^{p-1} + 2^p + 2$	$r(p) = O(p)$
Niezwykły	$18.2^p - 9p - 19$	$p.2^{p+3} - 2^p + 3$	$r(p) = O(p)$

4. PROBLEM ZMIENNYCH AKTYWNYCH

4.1. Sformułowanie problemu

Mówimy, że zmienna V jest użyta, wtedy i tylko wtedy, gdy występuje do niej odwołanie bez zmiany jej wartości, tak jak w instrukcji write, lub gdy jest ona argumentem wyrażenia. Zmienna V jest aktywna w punkcie p grafu przepływu, jeżeli istnieje V-wolna droga od p do punktu użycia V. Jeśli taka droga nie istnieje, to zmienna V w punkcie p jest nieaktywna. Oznacza to, że V jest aktywna, jeśli jej bieżąca wartość może być użyta przed ponownym zdefiniowaniem V.

Rozwiązanie problemu zmiennych aktywnych polega na wyznaczeniu zbiorów lvaren(x) (lvarex(x)) zawierających zmienne, które są aktywne na wejściu (wyjściu) każdego wierzchołka x grafu przepływu G[1]. Zbiory lvaren(x) i lvarex(x) można wyznaczyć za pomocą dwóch zbiorów zawierających informacje o charakterze lokalnym:

(a) inside(x), zdefiniowany dla każdego wierzchołka x w G, jest zbiorem zmiennych V mających lokalnie eksponowane użycia (ang. locally exposed uses) w wierzchołku x, tj. zbiorem takich zmiennych, dla których istnieje V-wolna droga od wejścia x do użycia V wewnątrz x,

(b) trans(x,y), zdefiniowany jak uprzednio (patrz pkt. 3.1).

Twierdzenie 4.1. Niech $G = (N, \Gamma, s)$ będzie grafem przepływu. Dla każdego $x \in N$

$$\mathrm{lvaren}(x) = \mathrm{inside}(x) \cup \mathrm{lvarex}(x)$$

$$\mathrm{lvarex}(x) = \bigcup_{y \in \Gamma x} (\mathrm{trans}(x,y) \cap \mathrm{lvaren}(y))$$

Dowód. Patrz Kennedy [1976].□

Uwaga

Dla problemu zmiennych aktywnych poszukiwane jest rozwiązanie minimalne.□

4.2. Algorytm "round-robin"

W niniejszym punkcie przedstawimy wersję "round-robin" iteracyjnego algorytmu rozwiązania problemu zmiennych aktywnych. Algorytm rozpoczynając obliczenia od wektorów INSIDE(j), $1 \leqslant j \leqslant n$, jako początkowej estymaty szukanej informacji, "propaguje" ją następnie do wierzchołków, wizytując je cyklicznie do momentu, gdy przepływ informacji ustabilizuje się. Ponieważ do reprezentowania zmiennych aktywnych używane są wektory bitów, to wykonywanie algorytmu kończy się w momencie, gdy kolejna iteracja nie zmienia stanu żadnego bitu w żadnym wektorze bitów.

[1] W praktyce, oba zbiory: lvaren(x) i lvarex(x) są użyteczne.

Algorytm 4.1: Wersja "round-robin" iteracyjnego algorytmu dla problemu zmiennych aktywnych.

Dane: 1. Graf przepływu $G = (N, \Gamma, s)$. Wierzchołki G są ponumerowane od 1 do n w porządku "rPostorder".
2. Wektory bitów TRANS(j) oraz INSIDE(j) dla każdego j, $1 \leqslant j \leqslant n$.

Wyniki: Wektory bitów LVAREN(j) dla każdego j, $1 \leqslant j \leqslant n$.

Metoda:

```
begin
  -- Faza 1: Dokonać inicjalizacji.
  for j from 1 to n do
     LVAREN(j) := INSIDE(j)
  endfor;
  -- Faza 2: Iterować.
  CHANGE := true; -- CHANGE jest zmienną boolowską.
  while CHANGE do
     CHANGE := false;
     for j from n to 1 by -1 do -- W porządku "Postorder".
       -- NEW jest pomocniczym wektorem bitów.
        E1: NEW := ((∨_{k∈Γj} LVAREN(k)) ∧ TRANS(j)) ∨ INSIDE(j);
        if NEW ≠ LVAREN(j) then
           LVAREN(j) := NEW;
           CHANGE := true
        endif
     endfor
  endwhile
end □
```

Twierdzenie 4.2. Algorytm 4.1 jest częściowo poprawny oraz jego obliczenie dochodzi do punktu końcowego.

Dowód. Patrz Ullman [1973].□

Lemat 4.3.1. Instrukcja while Algorytmu 4.1 wykonywana jest co najwyżej d + 2 razy dla dowolnego redukowalnego grafu przepływu G, gdzie d jest powiązaniem pętli w G[1].

[1] Należy podkreślić jeszcze raz, że wierzchołki G są wizytowane w porządku "Postorder".

Dowód. Patrz Hecht i Ullman [1975] .□

Twierdzenie 4.3. W najgorszym przypadku, wykonanie Algorytmu 4.1 wymaga realizacji $(d + 2)(e_0 + n_0)$ operacji na wektorach bitów.

Dowód. Dla każdego wierzchołka G, obliczenie $T = \bigvee_{k \in \Gamma j} LVAREN(k)$ w kroku E1 wymaga wykonania o jedną operację na wektorach bitów mniej niż liczba krawędzi wychodzących z tego wierzchołka, co daje w sumie $e_0 - n_0$ operacji. Ponadto w każdym wierzchołku niezbędna jest realizacja dwóch operacji do obliczenia $(T \wedge TRANS(j)) \vee INSIDE(j)$. Na podstawie Lematu 4.3.1, krok E1 jest powtarzany co najwyżej $d + 2$ razy, a więc maksymalna liczba operacji na wektorach bitów niezbędnych do wykonania Algorytmu 4.1 wynosi

$$(d + 2)(e_0 - n_0 + 2n_0) = (d + 2)(e_0 + n_0).\square$$

4.3. Algorytm analizy regionów

Algorytm analizy regionów dla problemu zmiennych aktywnych (Czech [1982a b]) składa się z dwóch faz. Zadaniem pierwszej fazy, w której przetwarza się wierzchołki minimalnych regionów (tj. interwałów G) w odwrotnym porządku interwałowym, jest określenie dla każdego wierzchołka x w interwale I zmiennych, dla których występują drogi bez definicji tych zmiennych od wierzchołka x do użyć zmiennych w dalszych wierzchołkach porządku interwałowego. Tak więc na razie pomija się "wpływy" krawędzi zamykających interwałów oraz fragmentów grafu leżących na zewnątrz interwałów. W pierwszej fazie określa się także dla każdego wierzchołka x interwału I czy występują drogi bez definicji od wejścia wierzchołka x do wejść głów interwałów będących następnikami I oraz do wejścia głowy interwału I.

W drugiej fazie algorytmu przegląda się sekwencję grafów pochodnych $(G = G_0, G_1, \ldots, G_m)$ dla G i poczynając od interwałów przetwarza się kolejno coraz większe regiony G. Dla każdego wierzchołka x regionu R wyznacza się zmienne, dla których występują drogi bez definicji (być może wzdłuż krawędzi w tył) od wierzchołka x do użyć zmiennych wewnątrz lub na zewnątrz regionu R.

Faza 1. W pierwszej fazie przetwarza się wierzchołki każdego interwału grafu G w odwrotnym porządku interwałowym określając wstępne estymaty zbiorów[1] lvaren(j)[2], jak również zbiory path(j,r) dla każdego wierzchołka j interwału. Zbiory path(j,r) zawierają wszystkie zmienne, dla których występują drogi bez definicji od wejścia wierzchołka j do wejścia wierzchołka r, gdzie r oznacza głowy interwałów będących następnikami interwału aktualnie przetwarzanego oraz głowę tego interwału. Oto algorytm, który oblicza wymienione zbiory używając ich reprezentacji w postaci wektorów bitów.

[1] Są to estymaty wstępne, ponieważ pomija się, na razie, "wpływy" krawędzi zamykających oraz fragmentu grafu leżącego na zewnątrz interwału.

[2] W algorytmach, które przedstawimy, będą obliczane jedynie zbiory lvaren(j). Zbiory lvarex(j) mogą być łatwo wyznaczone na podstawie lvaren(j) z użyciem jednego z wzorów z Twierdzenia 4.1.

Algorytm 4.2A: Oblicza wektory bitów PATH oraz wstępne estymaty wektorów bitów LVAREN.

Dane:
1. Interwał I z wierzchołkami ponumerowanymi od 1 do n_I w porządku interwałowym.
2. Informacje o następnikach dla każdego wierzchołka z I.
3. Zbiór interwałów M będących następnikami I (następniki w grafie pochodnym, tj. $M \in \Gamma I$), wraz z ich głowami h_M.
4. Wektory bitów INSIDE(j) i TRANS(j), $1 \leqslant j \leqslant n_I$.

Wyniki:
1. Wstępne estymaty wektorów bitów LVAREN(j), $1 \leqslant j \leqslant n_I$.
2. Wektory bitów PATH(j,r), $j \neq r$, gdzie r oznacza wszystkie takie głowy następników M interwału I oraz głowę I, dla których istnieje droga od wejścia wierzchołka j do wejścia głowy r.

Metoda:

```
begin
  -- Dla ostatniego wierzchołka porządku interwałowego I.
  A1: LVAREN(n_I) := INSIDE(n_I);
  for wszystkich k∈Γn_I do
    A2: PATH(n_I,k) := TRANS(n_I)
  endfor
  -- Dla każdego wierzchołka j∈I różnego od głowy, w odwrotnym porząd-
  -- ku interwałowym.
  for j from n_I - 1 to 1 by - 1 do
    if Γ j ∩ I - {1} ≠ ∅ then
      -- T jest pomocniczym wektorem bitów.
      A3: T := ⋁_{k∈Γj∩I-{1}} LVAREN(k);
      A4: LVAREN(j) := INSIDE(j) ∨ (TRANS(j) ∧ T)
    endif;
    for wszystkich k∈Γ j ∩ ({h_M} ∪ {1}) do
      if j ≠ k then  -- Pominąć pętle (1,1).
        A5: PATH(j,k) := TRANS(j)
      endif
    endfor;
    for wszystkich r∈Γ* j ∩ (({h_M} ∪ {1}) - Γ j) do
      if j ≠ r then -- Pominąć cykle (1,x_1,x_2,...,1).
```

-- Niech $U = \{k\}$ będzie zbiorem wierzchołków k w I takich,
-- że każdy z nich jest bezpośrednim następnikiem, różnym od
-- głowy, wierzchołka j, tzn., $k \in \Gamma j \cap I - \{1\}$, oraz ist-
-- nieje droga od wejścia wierzchołka k do wejścia wierz-
-- chołka r, tzn., $\Gamma^* k \cap \{r\} \neq \emptyset$.

A6: $T := \bigvee_{k \in U} PATH(k,r)$;

A7: $PATH(j,r) := T \wedge TRANS(j)$

endif

endfor

endfor

end □

Twierdzenie 4.4. Algorytm 4.2A jest częściowo poprawny oraz jego obliczenie dochodzi do punktu końcowego.

Dowód. Dochodzenie obliczenia algorytmu do punktu końcowego wynika ze skończoności n_I. Poprawności algorytmu dowodzą dwa lematy podane poniżej. W dalszym ciągu będziemy pisać "$V \in$ WEKTORBITÓW" mając na myśli, że "na pozycji dla V w wektorze bitów WEKTORBITÓW jest 1".

Lemat 4.4.1. Po wykonaniu Algorytmu 4.2A, $V \in LVAREN(j)$, $n_I \geqslant j \geqslant 1$, wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia wierzchołka j do użycia V wewnątrz któregoś z wierzchołków $j, j-1, \ldots, n_I$ interwału I.

Dowód. Przeprowadzamy indukcję według p, $1 \leqslant p \leqslant n_I$, gdzie $p = n_I - j + 1$ określa numery wierzchołków kolejno przetwarzanych przez Algorytm 4.2A.

Podstawa. $p = 1$, tj. $j = n_I$. Na podstawie kroku A1, $V \in LVAREN(n_I)$ wtedy i tylko wtedy, gdy $V \in INSIDE(n_I)$. Lecz $V \in INSIDE(n_I)$ wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia n_I do użycia V wewnątrz n_I.

Krok indukcyjny. Załóżmy, że lemat jest prawdziwy dla $1 \leqslant p \leqslant s-1$, gdzie $s \leqslant n_I$. Na podstawie kroku A3, $V \in LVAREN(s)$ wtedy i tylko wtedy, gdy a) $V \in INSIDE(s)$ lub b) $V \in TRANS(s) \wedge T$. W przypadku a), $V \in INSIDE(s)$ wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia s do użycia V wewnątrz s, a więc lemat jest prawdziwy. W przypadku b), $V \in TRANS(s) \wedge T$ wtedy i tylko wtedy, gdy istnieje V-wolna droga przez s, tj., $V \in TRANS(s)$ oraz istnieje V-wolna droga od wejścia pewnego wierzchołka k, $1 \leqslant k \leqslant s-1$, do użycia V wewnątrz I, tj. $V \in T$ (na podstawie hipotezy indukcyjnej) lecz to oznacza, że istnieje V-wolna droga od wejścia s do użycia V wewnątrz pewnego wierzchołka k, $1 \leqslant k \leqslant s-1$.□

Lemat 4.4.2. Po wykonaniu Algorytmu 4.2A, $V \in PATH(j,r)$, gdzie $j \in I$ oraz r jest głową M interwału będącego następnikiem I lub głową interwału I, wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia wierzchołka j do wejścia głowy r.

Dowód. Rozważmy wszystkie wierzchołki interwału I takie, że istnieje droga od każdego z nich do pewnej głowy r. Załóżmy, że zmienna p numeruje te wierzchołki od 1 do s, $s \leqslant n_I$, w odwrotnym porządku interwałowym. Dowiedziemy poprawności lematu przez indukcję według p.

Podstawa. $p = 1$. Wierzchołek ten musi być bezpośrednim poprzednikiem głowy r, a zatem w trakcie wykonywania Algorytmu 4.2A wykonuje się dla niego krok A5. Stąd, $V \in PATH(p,r)$ wtedy i tylko wtedy, gdy $V \in TRANS(p)$, a to jest równoważne występowaniu V-wolnej drogi od wejścia p do wyjścia p, tj. do wejścia r.

Krok indukcyjny. Załóżmy, że hipoteza indukcyjna jest prawdziwa dla wierzchołków p, $1 \leqslant p \leqslant q-1$, $q \leqslant s$, i rozważmy wierzchołek q. Należy przeanalizować dwa przypadki.

Przypadek 1. Wierzchołek q jest bezpośrednim poprzednikiem głowy r. W tym przypadku lemat jest prawdziwy w oparciu o krok A5.

Przypadek 2. Wierzchołek q nie jest bezpośrednim poprzednikiem głowy r. Wówczas, na podstawie kroku A7, $V \in PATH(q,r)$ wtedy i tylko wtedy, gdy $V \in TRANS(q)$ i $V \in T$. W oparciu o hipotezę indukcyjną oraz krok A6, $V \in T$ wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia pewnego następnika q do głowy r. Z tego faktu oraz definicji $TRANS(q)$ wynika, że $V \in PATH(q,r)$ wtedy i tylko wtedy, gdy istnieje V-wolna droga od wejścia wierzchołka q do wejścia głowy r.□□

Twierdzenie 4.5. Wykonanie Algorytmu 4.2A wymaga zrealizowania co najwyżej

$$e_I^f + n_I^f + e_{I-\{h\}}^{fh} + \sum_{r \in \Gamma I} e_{Ir}^f$$

operacji na wektorach bitów, gdzie $e_{I-\{h\}}^{fh}$ jest liczbą krawędzi w przód interwału I położonych na drogach prowadzących od wierzchołków I różnych od jego głowy h, do głowy h, zaś e_{Ir}^f jest liczbą krawędzi w przód interwału I położonych na drogach prowadzących od wierzchołków I do głów r interwałów będących następnikami I.

Dowód. Niech $n_{I-\{h\}}^{fh}$ oraz n_{Ir}^f oznaczają liczby wierzchołków I, z których wychodzą odpowiednio krawędzie $e_{I-\{h\}}^{fh}$ oraz e_{Ir}^f. Obliczenie T dla każdego wierzchołka I (krok A3) wymaga o jedną operację na wektorach bitów mniej niż liczba krawędzi w przód wychodzących z tego wierzchołka. Liczba krawędzi w przód interwału I wynosi e_I^f, a więc do obliczenia T wymaganych jest $e_I^f - n_I^f$ operacji. Dla każdego wierzchołka I, z którego wychodzi co najmniej jedna krawędź w przód wykonuje się dwie operacje (krok A4), co daje w sumie $2n_I^f$ operacji.

Obliczenie T w kroku A6 wymaga:

a) dla każdego wierzchołka I różnego od głowy, o jedną operację na wektorach bitów mniej niż liczba krawędzi w przód wychodzących z tego wierzchołka i leżących na drogach prowadzących do głowy h interwału I oraz

b) dla każdego wierzchołka I, o jedną operację na wektorach bitów mniej niż liczba krawędzi w przód wychodzących z tego wierzchołka i prowadzących do głów interwałów będących następnikami I.

Tak więc, wykonanie kroku A6 dla wszystkich wierzchołków interwału I wymaga zrealizowania

$$e^{fh}_{I-\{h\}} - n^{fh}_{I-\{h\}} + \sum_{r\in\Gamma I} (e^{f}_{Ir} - n^{f}_{Ir})$$

operacji na wektorach bitów. Krok A7, o jednej operacji na wektorach bitów, wykonywany jest dla każdego wierzchołka I różnego od głowy z przynajmniej jedną krawędzią w przód położoną na drodze prowadzącej do głowy I oraz dla każdego wierzchołka I z przynajmniej jedną krawędzią w przód położoną na drodze prowadzącej do głowy dowolnego interwału będącego następnikiem I. Stąd wykonanie kroku A7 wymaga realizacji

$$n^{fh}_{I-\{h\}} + \sum_{r\in\Gamma I} n^{f}_{Ir}$$

operacji na wektorach bitów. Sumując powyższe składniki otrzymujemy

$$(e^{f}_{I} - n^{f}_{I}) + 2n^{f}_{I} + \left[e^{fh}_{I-\{h\}} - n^{fh}_{I-\{h\}} + \sum_{r\in\Gamma I} e^{f}_{Ir} - n^{f}_{Ir})\right] +$$

$$+ (n^{fh}_{I-\{h\}} + \sum_{r\in\Gamma I} n^{f}_{Ir}) = e^{f}_{I} + n^{f}_{I} + e^{fh}_{I-\{h\}} + \sum_{r\in\Gamma I} e^{f}_{Ir}. \square$$

Faza 2. W drugiej fazie, przeglądając sekwencję grafów pochodnych ($G = G_0, G_1, \ldots, G_m$) dla G, przetwarza się kolejno coraz większe regiony G. Dla wszystkich wierzchołków j w każdym regionie uaktualnia się zbiory lvaren(j) zawierające zmienne, dla których istnieją drogi bez definicji od wejścia wierzchołka j do użyć zmiennych wewnątrz lub na zewnątrz regionu.

Niech R_J będzie regionem G reprezentowanym przez wierzchołek J w pewnym grafie pochodnym. Przyjmijmy, że wierzchołki R_J są dane w porządku interwałowym. W drugiej fazie przegląda się interwały oraz ich wierzchołki w grafie pochodnym, w odwrotnym porządku interwałowym, przetwarzając odpowiedni region R_J dla każdego wierzchołka J, z którego wychodzi co najmniej jedna krawędź w przód. Każdy interwał pierwszego rzędu I w regionie R_J analizowany jest oddzielnie, zaś interwały są wybierane w odwrotnym porządku interwałowym. Najpierw określane są wpływy krawędzi wyjściowych I[1] na zbiory lvaren(j), $j \in I$, a następnie wpływy krawędzi zamykających I.

Następujący algorytm uaktualnia zbiory lvaren(j) używając ich wektorowych reprezentacji.

Algorytm 4.2B: Uaktualnia wektory bitów LVAREN.

Dane: 1. Region R_J grafu G reprezentowany przez wierzchołek J w pewnym grafie pochodnym. Wierzchołki R_J są ponumerowane od 1 do n_J w porządku interwałowym.

2. Wektory PATH(j,r), $1 \leqslant j \leqslant n_J$, gdzie r oznacza głowy interwałów pierwszego rzędu w R_J osiągalne z wierzchołka j.

3. Stare wartości wektorów LVAREN(j), $1 \leqslant j \leqslant n_J$.

Wyniki: Nowe wartości wektorów LVAREN(j), $1 \leqslant j \leqslant n_J$.

Metoda:

```
begin
  for każdego interwału pierwszego rzędu I w regionie R_J w odwrotnym po-
      rządku interwałowym do
  for wszystkich wierzchołków j ∈ I takich, że Γ*j ∩ {h_M} ≠ ∅ do
      -- Zbiór {h_M} zawiera głowy interwałów bedących bezpośrednimi na-
      -- stępnikami I.
      for wszystkich r ∈ Γ*j ∩ {h_M} do
          B1: LVAREN(j) := LVAREN(j) ∨ (PATH(j,r) ∧ LVAREN(r))
      endfor
  endfor;
  for wszystkich wierzchołków j ∈ I-{h_I} takich, że h_I ∈ Γ* j do
      -- h_I jest głową I.
      B2: LVAREN(j)  := LVAREN(j) ∨ (PATH(j,h_I) ∧ LVAREN(h_I))
  endfor
  endfor
end □
```

Twierdzenie 4.6. Algorytm 4.2B jest częściowo poprawny oraz jego obliczenie dochodzi do punktu końcowego.

Dowód. Dochodzenie obliczeń algorytmu do punktu końcowego jest ewidentne, jako że liczby powtórzeń wszystkich instrukcji iteracyjnych algorytmu są skończone. Poprawność algorytmu udowodnimy przez pokazanie, że zmienna V jest w nowym wektorze LVAREN(j), $1 \leqslant j \leqslant n_J$, wtedy i tylko

[1] Wśród nich są krawędzie w tył regionu R_J.

wtedy, gdy V była w starym wektorze LVAREN(j)[1], lub gdy istnieje V-wolna droga od wierzchołka j do użycia V wewnątrz lub na zewnątrz regionu R_J. Niech przedrostki nowy- oraz stary- wyróżniają odpowiednio nowe oraz stare wartości wektorów bitów.

Przypadek 1. $V \in$ stary LVAREN(j). Każdy wektor nowyLVAREN(j) jest obliczany przez dodanie logiczne wyrażenia PATH(j,r) $\wedge$ LVAREN(t) (krok B1), lub /i PATH(j,h_I) $\wedge$ LVAREN(h_I) (krok B2) do wektora staryLVAREN(j). Dlatego wektor nowyLVAREN(j) będzie zawierał wszystkie zmienne wektora staryLVAREN(j) oraz ewentualnie inne zmienne jako rezultat dodania wymienionych wyrażeń.

Przypadek 2. Istnieje V-wolna droga od wejścia wierzchołka j do użycia V wewnątrz lub na zewnątrz regionu R_J. Udowodnimy poprawność twierdzenia dla tego przypadku przez indukcję według numeru interwału pierwszego rzędu w regionie R_J. Niech $I_1, I_2, \ldots, I_s$ będzie sekwencją interwałów w R_J danych w odwrotnym porządku interwałowym.

Podstawa. Rozważmy wierzchołki pierwszego przetwarzanego przez Algorytm 4.2B interwału w R_J, tj., I_1. Interwały będące następnikami I_1 to albo interwał I_s w regionie R_J albo interwały leżące poza regionem. Na podstawie kroku B1, $V \in$ LVAREN(j), gdy $V \in$ PATH(j,r) oraz $V \in$ LVAREN(r). Jeśli r jest głową I_s to warunki te są równoważne istnieniu V-wolnej drogi od wejścia wierzchołka j do głowy I_s i stąd do użycia V wewnątrz interwału I_s, tzn. wewnątrz regionu R_J. Podobnie, jeśli $V \in$ LVAREN(j) oraz r jest głową interwału leżącego poza regionem R_J, to musi istnieć V-wolna droga od wejścia j do użycia V na zewnątrz regionu R_J. Rozważając krok B2 wnioskujemy, że $V \in$ LVAREN(j), jeśli $V \in$ LVAREN(h_{I_1}) oraz $V \in$ PATH(j,h_{I_1}). Ponieważ wektor LVAREN(h_{I_1}) został obliczony w kroku B1, V należy do LVAREN(h_{I_1}), jeśli istnieje V-wolna droga od wejścia h_{I_1} do użycia V wewnątrz lub na zewnątrz regionu R_J. Biorąc ponadto pod uwagę warunek $V \in$ PATH(j,h_{I_1}) wnioskujemy ostatecznie, że po wykonaniu kroku B2, $V \in$ LVAREN(j), jeśli istnieje V-wolna droga od wejścia wierzchołka j poprzez głowę h_{I_1} do użycia V wewnątrz lub na zewnątru regionu R_J.

Krok indukcyjny. Załóżmy, że twierdzenie jest prawdziwe dla wierzchołków wszystkich interwałów $I_1, I_2, \ldots, I_{p-1}$, gdzie $p-1 < s$ oraz rozważmy interwał I_p. Na podstawie kroku B1, $V \in$ LVAREN(j) dla każdego $j \in I_p$, jeśli $V \in$ PATH(j,r) i $V \in$ LVAREN(r), gdzie r może być:

a) głową jednego z interwałów $I_1, I_2, \ldots, I_{p-1}$, albo
b) głową interwału I_s, albo
c) głową interwału położonego poza regionem R_J.

[1] Oznacza to, że istnieje V-wolna droga od wejścia wierzchołka j do użycia V wewnątrz mniejszego regionu $R_{J'} \subset R_J$.

Przypadek a) r jest głową jednego z interwałów $I_1, I_2, \ldots, I_{p-1}$. $V \in$ LVAREN(j), $j \in I_p$, jeśli istnieje V-wolna droga od wejścia wierzchołka j do głowy pewnego dalszego interwału w regionie R_J ($V \in$ PATH(j,r)) i stąd do użycia V wewnątrz lub na zewnątrz regionu R_J ($V \in$ LVAREN(r), na podstawie hipotezy indukcyjnej).

Przypadek b) r jest głową interwału I_s. $V \in$ LVAREN(j), $j \in I_p$, jeśli istnieje V-wolna droga od wejścia wierzchołka j do głowy I_s ($V \in$ PATH(j,h_{I_s})) i stąd do użycia V wewnątrz I_s ($V \in$ LVAREN(h_{I_s})), tj. wewnątrz R_J.

Przypadek c) r jest głową interwału I_o położonego poza regionem R_J. $V \in$ LVAREN(j), $j \in I_p$, jeśli istnieje V-wolna droga od wejścia wierzchołka j do głowy I_o ($V \in$ PATH(j,h_{I_o})) i stąd do użycia V na zewnątrz regionu R_J ($V \in$ LVAREN(h_{I_o})).

Argumenty dla tych trzech podprzypadków oraz kroku B2 są podobne jak powyżej, z tą różnicą, że V-wolna droga od wierzchołka j w interwale I do użycia V wewnątrz lub na zewnątrz regionu R_J będzie przechodzić przez głowę I.□

Twierdzenie 4.7. Wykonanie Algorytmu 4.2B wymaga realizacji

$$2 \sum_{I \in R_J} (n^h_{I-\{h\}} + \sum_{r \in \Gamma I} n_{Ir})$$

operacji na wektorach bitów.

Dowód. Krok B1 z dwoma operacjami na wektorach bitów jest wykonywany dla każdego wierzchołka I oraz każdej głowy r interwału będącego następnikiem I takich, że istnieje droga od tego wierzchołka do głowy r. Stąd, wykonanie kroku B1 dla całego interwału I wymaga $2 \sum_{r \in \Gamma I} n_{Ir}$ operacji na wektorach bitów. Dwie operacje wymagane są w kroku B2 dla każdego wierzchołka I różnego od głowy, od którego istnieje droga do głowy wzdłuż krawędzi zamykających I, a więc w sumie $2n^h_{I-\{h\}}$ operacji. Ponieważ kroki B1 i B2 są powtarzane dla każdego interwału I regionu R_J, wykonanie Algorytmu 4.2B wymaga realizacji

$$2 \sum_{I \in R_J} (n^h_{I-\{h\}} + \sum_{r \in \Gamma I} n_{Ir})$$

operacji na wektorach bitów.□

Oto kompletny algorytm analizy regionów dla problemu zmiennych aktywnych.

Algorytm 4.2C: Algorytm analizy regionów dla problemu zmiennych aktywnych

Dane: 1. Redukowalny graf przepływu $G = (N, \Gamma, s)$.

2. Sekwencja grafów pochodnych $(G = G_0, G_1, \ldots, G_m)$ dla G, gdzie G_m jest grafem trywialnym. $G_i = (N_i, \Gamma_i, s_i)$.

3. Listy zawartości w porządku interwałowym każdego regionu w grafie G oraz każdego interwału w sekwencji grafów pochodnych.

4. Wektory bitów INSIDE(j) oraz TRANS(j) dla każdego $j \in N$.

Wyniki: Wektory bitów LVAREN(j) dla każdego $j \in N$.

Metoda:

```
begin
  -- Faza 1: Przetworzyć interwały pierwszego rzędu w G.
  for każdego wierzchołka J w G1 do
     Zastosować Algorytm 4.2A do obliczenia wektorów LVAREN i PATH dla
     regionu RJ (lub po prostu dla interwału I = RJ).
  endfor;
  -- Faza 2: Przetwarzając sekwencję grafów pochodnych poczynając od re-
  -- gionów wewnętrznych do zewnętrznych, stosować Algorytm 4.2B.
  for i from 1 to m-1 do
     C1: for każdego wierzchołka J o przynajmniej jednej krawędzi nie w
         tył w Gi, w odwrotnym porządku interwałowym do
         Zastosować Algorytm 4.2B do uaktualnienia wektorów LVAREN dla re-
         gionu RJ.
     endfor
  endfor;
  C2: Zastosować Algorytm 4.2B do ostatecznego uaktualnienia wektorów
      LVAREN dla regionu Rsm = G.
end □
```

Twierdzenie 4.8. Algorytm 4.2C jest częściowo poprawny oraz jego obliczenie dochodzi do punktu końcowego.

Dowód. Ponieważ liczby powtórzeń wszystkich instrukcji iteracyjnych Algorytmu 4.2C są skończone, dochodzenie obliczenia algorytmu do punktu końcowego jest oczywiste. Poprawność algorytmu wynika z poprawności Algorytmów 4.2A i 4.2B wykonywanych podczas realizacji Algorytmu 4.2C oraz z następujących spostrzeżeń.

1. Ilekroć Algorytm 4.2B jest stosowany dla regionu R_J w G (krok C1), zbiór zmiennych, dla których istnieją drogi bez definicji od głowy R_J do użyć tych zmiennych wewnątrz R_J został już obliczony. Wynika to stąd, że "podregiony" R_J były przetwarzane w odwrotnym porządku interwałowym w trakcie przetwarzania grafu niższego rzędu.

2. Algorytm 4.2B określa dla każdego wierzchołka w regionie R_J zbiory zmiennych aktywnych biorąc pod uwagę krawędzie zamykające R_J oraz wszystkie użycia zmiennych wewnątrz R_J.

3. Ponieważ ostatnim przetwarzanym regionem jest graf przepływu G(krok C2), Algorytm 4.2C oblicza zbiory zmiennych aktywnych biorąc pod uwagę krawędzie zamykające G oraz wszystkie użycia zmiennych w G.□

Twierdzenie 4.9. Wykonanie Algorytmu 4.2C wymaga realizacji

$$e_o^f + n_o^f + e_o^{fh} + \sum_{I \in N_1} \sum_{r \in \Gamma I} e_{Ir}^f + 2 \sum_{i=1}^{m} \sum_{J \in N_i} \sum_{I \in R_J} (n_{I-\{h\}}^h + \sum_{r \in \Gamma I} n_{Ir})$$

operacji na wektorach bitów[1].

Dowód. Na podstawie Twierdzenia 4.5 każde wykonanie Algorytmu 4.2A w fazie 1 wymaga $e_I^f + n_I^f + e_{I-\{h\}}^{fh} + \sum_{r \in \Gamma I} e_{Ir}^f$ operacji. Ponieważ w fazie 1, Algorytm 4.2A wywoływany jest jednokrotnie dla każdego regionu (interwału) I, $I \in N_1$, to wykonanie fazy 1 wymaga realizacji

$$\sum_{I \in N_1} (e_I^f + n_I^f + e_{I-\{h\}}^{fh} + \sum_{r \in \Gamma I} e_{Ir}^f = e_o^f + n_o^f + e_o^{fh} + \sum_{I \in N_1} \sum_{r \in \Gamma I} e_{Ir}^f$$

operacji na wektorach bitów. Na podstawie Twierdzenia 4.7 podczas każdego wykonania Algorytmu 4.2B w fazie 2 realizowanych jest

$$2 \sum_{I \in R_J} (n_{I-\{h\}}^h + \sum_{r \in \Gamma I} n_{Ir})$$

operacji. W kroku C1 Algorytm 4.2B wykonywany jest co najwyżej raz dla każdego regionu $J \in N_i$, tak więc wykonanie kroku C1 wymaga co najwyżej

$$2 \sum_{J \in N_i} \sum_{I \in R_J} (n_{I-\{h\}}^h + \sum_{r \in \Gamma I} n_{Ir})$$

operacji. Ponieważ krok C1 jest powtarzany raz dla każdego i, $1 \leqslant i \leqslant m-1$ oraz wykonanie kroku C2 wymaga

$$2 \sum_{J \in N_m} \sum_{I \in R_J} (n_{I-\{h\}}^h + \sum_{r \in \Gamma I} n_{Ir})$$

---

[1] e_o^f, n_o^f i e_o^{fh} oznaczają liczby odpowiednich krawędzi lub wierzchołków w oryginalnym grafie przepływu G.

operacji, więc faza 2 wymaga co najwyżej

$$2\sum_{i=1}^{m-1}\sum_{J\in N_i}\sum_{I\in R_J}(n^h_{I-\{h\}}+\sum_{r\in\Gamma I}n_{Ir})+2\sum_{J\in N_m}\sum_{I\in R_J}(n^h_{I-\{h\}}+\sum_{r\in\Gamma I}n_{Ir})=$$

$$=2\sum_{i=1}^{m}\sum_{J\in N_i}\sum_{I\in R_J}(n^h_{I-\{h\}}+\sum_{r\in\Gamma I}n_{Ir})$$

operacji na wektorach bitów. Stąd maksymalna liczba operacji wykonywanych podczas realizacji Algorytmu 4.2C jest równa

$$e^f_o+n^f_o+e^{fh}_o+\sum_{I\in N_1}\sum_{r\in\Gamma I}e^f_{Ir}+2\sum_{i=1}^{m}\sum_{J\in N_i}\sum_{I\in R_J}(n^h_{I-\{h\}}+\sum_{r\in\Gamma I}n_{Ir}).\square$$

<u>Twierdzenie 4.10</u>. Dla grafu przepływu o n wierzchołkach, wykonanie algorytmu analizy regionów dla problemu zmiennych aktywnych (Algorytmu 4.2C) wymaga realizacji w najgorszym przypadku co najwyżej $6.5n^2 - 2.5n + 1$ operacji na wektorach bitów.

<u>Dowód</u>. Na podstawie Twierdzenia 4.9[1)]

$$T_R=e^f+n^f+e^{fh}+\sum_{I\in N_1}\sum_{r\in\Gamma I}e^f_{Ir}+2\sum_{I=1}^{m}\sum_{J\in N_i}\sum_{I\in R_J}(n^h_{I-\{h\}}+\sum_{r\in\Gamma I}n_{Ir}).$$

Zgodnie z (3.2) mamy $e^f \leqslant 2n - 2n_1 + 2$. Ponieważ $n^f \leqslant e^f$ to

$$e^f+n^f\leqslant 2(2n-2n_1+2)=4n-4n_1+4. \tag{4.1}$$

W celu wyznaczenia oszacowania od góry wielkości

$$e^{fh}+\sum_{I\in N_1}\sum_{r\in\Gamma I}e^f_{Ir}=\sum_{I\in N_1}(e^{fh}_{I-\{h\}}+\sum_{r\in\Gamma I}e^f_{Ir})$$

znajdziemy najpierw oszacowanie wyrażenia

$$\mathcal{E}=e^{fh}_{I-\{h\}}+\sum_{r\in\Gamma I}e^f_{Ir}=e^f_{Io_1}+e^f_{Io_2}+\dots+e^f_{I(o_{n_I-1})} \tag{4.2}$$

1) W dalszym ciągu, T_R będzie oznaczać złożoność algorytmu analizy Regionów dla problemu zmiennych aktywnych. Jak poprzednio, dla uproszczenia oznaczamy $e^f_o = e^f$, $n^f_o = n^f$,..., itd.

gdzie $o_j \in \{h\}\cup\Gamma I$, $1 \leqslant j \leqslant n_I-1$. Wyrażenie to osiąga wartość maksymalną wtedy, gdy zbiór ΓI zawiera maksymalną liczbę głów interwałów będących następnikami I (rys. 4.1). Łatwo pokazać, że maksymalne liczby krawędzi w przód w interwale I położonych na drogach prowadzących od wierzchołków I do poszczególnych wyjść o_j, są równe odpowiednio: $e^f_{Io_1} = n_I - 1$, $e^f_{Io_2} = n_I - 1$, $e^f_{Io_3} = n_I - 2$,..., $e^f_{I(o_{n_I-1})} = 1$, gdzie n_I jest liczbą wierzchołków w I. (Zakładamy, że każde wyjście o_j prowadzi do głowy interwału będącego następnikiem I lub do głowy interwału I). Tak więc, oszacowanie od góry wyrażenia $\mathcal{E}$ jest

$$\mathcal{E}=e^{fh}_{I-\{h\}}+\sum_{r\in\Gamma I}e^f_{Ir}=e^f_{Io_1}+e^f_{Io_2}+\dots$$

$$+e^f_{I(o_{n_I-1})}\leqslant(n_I-1)+(n_I-1)+(n_I-2)+$$

$$+\dots+1=\frac{n_I^2}{2}+\frac{n_I}{2}-1.$$

Rys. 4.1. Interwał I, dla którego wyrażenie (4.2) ma wartość maksymalną $o_j \in \{h\}\cup\Gamma I$, $1 \leqslant j \leqslant n_I-1$, n_I jest głową h interwału I

Rozważmy oszacowanie od góry wyrażenia

$$\mathcal{E}_1=\sum_{I\in N_1}(e^{fh}_{I-\{h\}}+\sum_{r\in\Gamma I}e^f_{Ir})=$$

$$=\sum_{k=1}^{n_1}(\frac{n_{I_k}^2}{2}+\frac{n_{I_k}}{2}-1),$$

gdzie $n_1 = \# N_1$ jest liczbą wierzchołków w grafie pochodnym pierwszego rzędu, tj. liczbą interwałów pierwszego rzędu w oryginalnym grafie przepływu G. Nietrudno udowodnić, że $\mathcal{E}_1$ osiąga maksimum, gdy graf przepływu G o n wierzchołkach zostaje podzielony na równe interwały[1)]. Wówczas $n_{I_k} = \frac{n}{n_1}$ dla każdego k, $1 \leqslant k \leqslant n_1$, a maksimum $\mathcal{E}_1$ wynosi

1) Wynika to z rozwiązania następującego problemu programowania nieliniowego

$$\max_{n_{I_k}}\sum_{k=1}^{n_1}(\frac{n_{I_k}^2}{2}+\frac{n_{I_k}}{2}-1),\ \text{gdzie}\ n_{I_1}+n_{I_2}+\dots+n_{I_{n_1}}=\sum_{k=1}^{n_1}n_{I_k}=n.$$

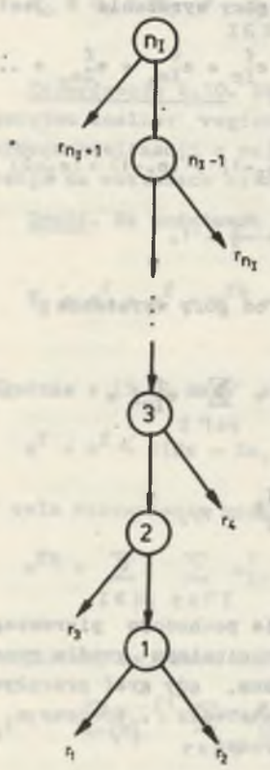
$$\mathcal{E}_1 = \sum_{k=1}^{n_1} \left(-\frac{n_{I_k}^2}{2} + \frac{n_{I_k}}{2} - 1\right) \leqslant \frac{n^2}{2n_1} + \frac{n}{2} - n_1. \tag{4.3}$$

Maksymalna liczba wierzchołków różnych od głów interwałów w grafie przepływu G, dla których istnieją drogi od tych wierzchołków do głów interwałów, w których wierzchołki te są zawarte, wynosi $n-n_1$, tak więc

$$\sum_{J \in N_1} \sum_{I \in R_J} n^h_{I-\{h\}} \leqslant n-n_1. \tag{4.4}$$

Wyznaczymy obecnie oszacowanie od góry wielkości

$$\mathcal{N} = \sum_{r \in \Gamma I} n_{Ir} = n_{Ir_1} + n_{Ir_2} + \dots + n_{I(r_{n_I+1})}. \tag{4.5}$$



Rys. 4.2. Interwał I, dla którego wyrażenie (4.5) ma wartość maksymalną. $\Gamma I = \{r_1, r_2, \dots, r_{n_I+1}\}$, n_I jest głową interwału I

Wyrażenie to osiąga wartość maksymalną dla interwału z rys. 4.2. Maksymalne liczby wierzchołków, z których istnieją drogi do poszczególnych głów r_j, $1 \leqslant j \leqslant n_I + 1$, interwałów będących następnikami I, są następujące: $n_{Ir_1} = n_I$, $n_{Ir_2} = n_I$, $n_{Ir_3} = n_I-1, \dots$, $n_{I(r_{n_I+1})} = 1$.

Stąd oszacowanie od góry wielkości $\mathcal{N}$ wynosi

$$\mathcal{N} = \sum_{r \in \Gamma I} n_{Ir} = n_{Ir_1} + n_{Ir_2} + \dots + n_{I(r_{n_I+1})} \leqslant n_I + n_I + (n_I-1) + \dots + 1 = \frac{n_I^2}{2} + \frac{3n_I}{2}.$$

Wyrażenie

$$\mathcal{N}_1 = \sum_{J \in N_1} \sum_{I \in R_J} \sum_{r \in \Gamma I} n_{Ir} = \sum_{k=1}^{n_1} \left(\frac{n_{k_I}^2}{2} + \frac{3n_{I_k}}{2}\right)$$

osiąga wartość maksymalną w przypadku, gdy graf przepływu G o n wierzchołkach zostaje podzielony na równe interwały pierwszego rzędu, tj. $n_{I_k} = \frac{n}{n_1}$ dla każdego k, $1 \leqslant k \leqslant n_1$. Stąd oszacowanie od góry wyrażenia $\mathcal{N}_1$ wynosi

$$\mathcal{N}_1 = \sum_{k=1}^{n_1} \left(\frac{n_{I_k}^2}{2} + \frac{3n_{I_k}}{2}\right) \leqslant \frac{n^2}{2n_1} + \frac{3n}{2}. \tag{4.6}$$

Używając oszacowań (4.1), (4.3), (4.4) i (4.6) otrzymujemy

$$T_R = (e^f + n^f) + \left(e^{fh} + \sum_{I \in N_1} \sum_{r \in \Gamma I} e^f_{iR}\right) + 2 \sum_{i=1}^{m} \sum_{J \in N_1} \sum_{I \in R_J} \left(n^h_{I-\{h\}} + \sum_{r \in \Gamma I} n_{Ir}\right) \leqslant (4n-4n_1+4) + \left(\frac{n^2}{2n_1} + \frac{n}{2} - n_1\right) + 2 \sum_{i=1}^{m} \left[(n-n_1) + \left(\frac{n^2}{2n_1} + \frac{3n}{2}\right)\right] = 4.5n - 5n_1 + 4 + \frac{n^2}{2n_1} + 2m(n-n_1) + \frac{m}{n_1} n^2 + 3\,mn.$$

Ponieważ $1 \leqslant m \leqslant n_1 \leqslant n-1$, więc ostatecznie

$$T_R = 4.5n - 5n_1 + 4 + \frac{n^2}{2n_1} + 2m(n-n_1) + \frac{m}{n_1} n^2 + 3\,mn \leqslant 4.5n - 5 + 4 + \frac{n^2}{2} + 2(n-1)(n-1) + n^2 + 3(n-1)n = 6.5n^2 - 2.5n + 1. \square$$

<u>Wniosek 4.1</u>. Pesymistyczna złożoność czasowa Algorytmu 4.2C jest $O(n^2)$

<u>Dowód</u>. Wniosek wynika bezpośrednio z Twierdzenia 4.10. $\square$

4.4. Porównanie algorytmów

Niniejszy punkt poświęcony jest porównaniu czasowych złożoności Algorytmu 4.1 i Algorytmu 4.2C dla problemu zmiennych aktywnych w oparciu o kilka rodzin samopowielających się, redukowalnych grafów przepływu. $T_I(p)$ (lub krótko T_I) będzie oznaczać liczbę operacji niezbędnych do wykonania wersji "round-robin" algorytmu Iteracyjnego, zaś $T_R(p)$ (lub krótko T_R, jak poprzednio) - liczbę operacji wymaganych do realizacji algorytmu analizy regionów. Podstawiając $j = p-i$ oraz $e_p = e$, $e^f_p = e^f, \dots$, itp. we wzorach udowodnionych w Twierdzeniach 4.3 i 4.9 otrzymujemy

$$T_I(p) = T_I = (d+2)(e+n)$$

oraz

$$T_R(p) = T_R = 2\sum_{j=0}^{p-1}\sum_{J\in N_{p-j}}\sum_{I\in R_J}(n^h_{I-\{h\}} + \sum_{r\in\Gamma I} n_{Ir}) + \sum_{I\in N_1}\sum_{r\in\Gamma I} e^f_{Ir} +$$

$$+ e^f + n^f + e^{fh}.$$

Dla uproszczenia notacji wyróżnimy następujące funkcje

$$T_1(p) = T_1 = \sum_{J\in N_p}\sum_{I\in R_J}(n^h_{I-\{h\}} + \sum_{r\in\Gamma I} n_{Ir}), \quad \text{dla} \quad j = 0$$

$$T_2(j) = T_2 = \sum_{J\in N_{p-j}}\sum_{I\in R_J}(n^h_{I-\{h\}} + \sum_{r\in\Gamma I} n_{Ir}), \text{ dla } \quad j > 0$$

oraz

$$T_3(p) = T_3 = \sum_{I\in N_1}\sum_{r\in\Gamma I} e^f_{Ir}.$$

Używając tych funkcji we wzorze dla $T_R(p)$ otrzymujemy

$$T_R(p) = 2T_1(p) + 2\sum_{j=1}^{p-1} T_2(j) + T_3(p) + e^f + n^f + e^{fh}.$$

4.4.1. Graf muszelkowy

Dla grafu muszelkowego o poziomie p, $p > 1$, (rys. 3.2), $d = 1$, $e = 2p$, $n = p+1$, $T_1 = 3p-2$, $T_2 = j$, $T_3 = p-1$, $e^f = n^f = 1$ oraz $e^{fh} = 0$. Tak więc otrzymujemy

$$T_I = (1+2)(2p+p+1) = 9p+3$$

oraz

$$T_R = 2(3p-2)+2\sum_{j=1}^{p-1} j+p-1+1+1 = 7p-3+2\,\frac{1+p-1}{2}(p-1) = p^2+6p-3.$$

4.4.2. Graf podwójnie wiązany

Dla grafu podwójnie wiązanego o poziomie p, $p > 1$, (rys. 3.5) mamy $d = p$, $e = 2p$, $n = p+1$, $T_1 = 2p$, $T_2 = 2j-1$, $T_3 = 1$, $e^f = n^f = 1$ i $e^{fh} = 0$.

Stąd otrzymujemy

$$T_I = (p+2)(2p+p+1) = 3p^2+7p+2$$

oraz

$$T_R = 2.2p+2\sum_{j=1}^{p-1}(2j-1)+1+1+1 = 4p+3+4\,\frac{1+p-1}{2}(p-1)-2(p-1) = 2p^2+5.$$

.4.3. Graf niezwykły

Dla tego grafu o poziomie p, (rys. 3.7) $d = p$, $e = 2^{p+2}-4$, $n = 2^{p+1}-1$, $T_1 = 4.2^p-4$, $T_2 = 3.2^j-4$, $T_3 = 2^p$, $e^f = 2^p$, $n^f = 2^{p-1}$ oraz $e^{fh} = 0$. Stąd mamy

$$T_I = (p+2)(2^{p+2}-4+2^{p+1}-1) = 3p.2^{p+1}+3.2^{p+2}-5p-10$$

oraz

$$T_R = 2(4.2^p-4)+2\sum_{j=1}^{p-1}(3.2^j-4)+2^p+2^p+2^{p-1} = 21.2^{p-1}-8+6(2^p-2)-8(p-1) =$$

$$= 33.2^{p-1}-8p-12.$$

W tablicy 4.1 zebrano wyniki porównań algorytmów dla problemu zmiennych aktywnych.

Tablica 4.1

Wyniki porównań algorytmów "round-robin" oraz analizy regionów dla problemu zmiennych aktywnych

Klasa G_p	$T_I(p)$	$T_R(p)$	Przebieg asymptotyczny $r(p)=T_R(p)/T_I(p)$
Muszelkowy	$9p + 3$	$p^2 + 6p - 3$	$r(p) = O(p)$
Podwójnie wiązany	$3p^2 + 7p + 2$	$2p^2 + 5$	$r(p) \sim \frac{2}{3}$
Niezwykły	$3p \cdot 2^{p+1}+3.2^{p+2}-$ $- 5p-10$	$33.2^{p-1}-8p-12$	$r(p) = O(\frac{1}{p})$

5. ZAKOŃCZENIE

W rozprawie zaproponowano nowy algorytm, zwany algorytmem analizy regionów, przeznaczony do rozwiązywania problemów analizy przepływu danych dla redukowalnych grafów przepływu.

Udowodniono częściową poprawność algorytmu oraz dochodzenie jego obliczeń do punktu końcowego dla problemów definicji osiągających oraz zmiennych aktywnych. Przeprowadzono praktyczną weryfikację algorytmu dla problemu definicji osiągających implementując algorytm w języku Ada oraz dokonując obliczeń dla kilku przykładowych grafów przepływu.

Pokazano, że pesymistyczna złożoność czasowa algorytmu jest $O(n^2)$ dla obu problemów oraz dokonano porównań algorytmu pod względem złożoności czasowej z dwoma dobrze znanymi algorytmami rozwiązywania wymienionych problemów. Porównania te przeprowadzono dla kilku rodzin samopowielających się, redukowalnych grafów przepływu.

Dla problemu definicji osiągających wybrano do porównań algorytm Allena-Cocke'a. Ponieważ praca Allena i Cocke'a [1976] nie zawiera dokładnej analizy złożoności czasowej algorytmu, niezbędne było jej wyznaczenie (Twierdzenie 3.6). Wyniki porównań (zebrane w tablicy 3.2) nie rozstrzygają, który z algorytmów jest lepszy, pokazują one bowiem, że algorytm analizy regionów wymaga zrealizowania asymptotycznie mniejszej liczby operacji na wektorach bitów dla grafów: muszelkowego, spiralnego i podwójnie wiązanego, zaś większej liczby operacji dla grafów: binarnego Hechta-Ullmana, rombowego oraz niezwykłego. Wydaje się, że algorytm analizy regionów jest konkurencyjny względem złożoności czasowej w stosunku do algorytmu Allena-Cocke'a, ponieważ ten ostatni wyznacza pomocnicze wektory bitów TRANS(I, J), GEN(I,J) oraz RDEFEN(1), dla każdego wierzchołka każdego grafu w sekwencji grafów pochodnych, podczas gdy algorytm analizy regionów oblicza pomocnicze wektory bitów PATHEN(j) i PATHEX(j) tylko dla każdego wierzchołka w oryginalnym grafie przepływu. Istotny z punktu widzenia praktycznej efektywności algorytmów jest również fakt, że jakkolwiek oba algorytmy analizują poszczególne grafy w sekwencji grafów pochodnych, to algorytm analizy regionów przetwarza sekwencję jednokrotnie, zaś algorytm analizy interwałów - dwukrotnie, za pierwszym razem przechodząc sekwencję od grafu pierwszego rzędu do grafu granicznego, a następnie, za drugim razem, w porządku odwrotnym.

Algorytm analizy regionów dla problemu zmiennych aktywnych został porównany z wersją "round-robin" algorytmu iteracyjnego (Hecht i Ullman [1975]). Również w tym przypadku wyniki porównań (tablica 4.1) nie są

rozstrzygające, ponieważ pokazują, że w algorytmie analizy regionów wykonuje się asymptotycznie mniej operacji niż w algorytmie iteracyjnym, dla grafów: podwójnie wiązanego i niezwykłego, zaś więcej operacji dla grafu muszelkowego. Należy podkreślić, że realizacja programowa (postać kodu) algorytmu "round-robin" jest prostsza aniżeli algorytmu analizy regionów. Ponadto algorytm iteracyjny przetwarza nieredukowalne grafy przepływu bez dodatkowych nakładów, nie badając nawet czy są one nieredukowalne.

LITERATURA

[1] Adams J.M., Phelan J.M., Stark R.H.: A note on the Hecht-Ullman characterization of nonreducible flow graphs, SIAM J. Comput. 3,3 (Sept.) 1974, 222-223.

[2] Aho A.V., Hopcroft J.E., Ullman J.D.: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass. 1976.

[3] Aho A.V., Ullman J.D.: The Theory of Parsing, Translation, and Compiling, Vol. II: Compiling, Prentice-Hall, Englewood Cliffs, N.J., 1973, Rozdz. 11.

[4] Aho A.V., Ullman J.D.: Node listings for reducible flow graphs, J. Computer System Sci. 13, 1976, 286-299.

[5] Aho A.V., Ullman J.D.: Principles of Compiler Design, Addison-Wesley, Reading, Mass. 1977.

[6] Alagić S., Arbib M.A.: Projektowanie Programów Poprawnych i Dobrze Zbudowanych, WNT, Warszawa 1982.

[7] Allen F.E.: Program optimization, w Ann. Review in Automatic Programming 5, Pergamon Press, N.Y., 1969, 239-307.

[8] Allen F.E.: Control flow analysis, ACM SIGPLAN Notices 5, 7 (July), 1970, 1-19.

[9] Allen F.E.: A basis for program optimization, Proc. IFIP Congr. 71, North-Holland, Amsterdam, Holland, 1971, 385-390.

[10] Allen F.E.: Interprocedural data flow analysis, Proc. IFIP Congr. 74 North-Holland, Amsterdam, Holland, August 1974, 398-402.

[11] Allen F.E., Cocke J.: A program data flow analysis procedure, Comm. ACM 19, 3 (March) 1976, 137-147.

[12] Arrow K.J. Hurwicz L., Uzawa H.: Studies in linear and non-linear programming, Stanford Univ. Press, Standord, Calif. 1958.

[13] Banachowski L., Kreczmar A.: Elementy Analizy Algorytmów, WNT, Warszawa 1982.

[14] Barth J.M.: A practical interprocedural data flow analysis algorithm Comm. ACM 21,9 (Sept.), 1978, 724-736.

[15] Beatty J.C.: An algorithm for tracing live variables based on a straightened program graph, Intern. J. Computer Math. 5, Sec. A, 1975, 97-108.

[16] Biswas S., Bhattacharjee G.P.: A comparison of some algorithms for live variable analysis, Intern. J. Computer Math. 8, Sec. A, 1980, 121-134.

[17] Briggs J.S., Forsyth C.H., Johnson C.W., Manning M.R. Murdie J.A., Pyle I.C., Runciman C., Wand I.C., Williams A.J.: Ada workbench compiler project 1982, York Computer Science Report No 59, University of York, 1983.

[18] Cocke J.: Global common subexpressions elimination, ACM SIGPLAN Notices 5, 7(July) 1970, 20-24.

[19] Cocke J., Kennedy K.: Profitability computations on program flow graph, Comp. and Maths. with Appls. 2, 1976, 145-159.

[20] Cocke J., Schwartz J.T.: Programming Languages and their Compilers, second rev. version, Courant Institute of Mathematical Sciences, New York Univ., N.Y., April, 1970. Rozdz. 6.

[21] Czech Z.: A region analysis algorithm for the reaching definitions problem, w Optymalizacja i Automatyczna Korekta Programów Komputerowych, Politechnika Śląska, Gliwice październik 1981, 25-79.

[22] Czech Z.: A region analysis algorithm for the live variables problem Politechnika Śląska, Gliwice 1982a (maszynopis), s. 32.

[23] Czech Z.: Algorytm analizy regionów dla problemu przepływu danych, w Optymalizacji i Automatyczna Korekta Programów Komputerowych, Politechnika Śląska, Gliwice październik 1982b, 39-77.

[24] Czech Z.: A region analysis algorithm for the reaching definitions problem, (maszynopis), s. 29 (złożone do publikacji) 1983.

[25] Czech Z.: A comparison of the worst-case complexities of two algorithms for the reaching definitions problem, Fundamenta Informaticae (aktualnie publikowane) 1984.

[26] Earnest C.: Some topics in code optimization, J. ACM 21, 1 (Jan.) 1974, 76-102.

[27] Earnest C., Balke K.G., Anderson J.: Analysis of graphs by ordering of nodes, J. ACM 19, 1 (Jan.) 1972, 23-42.

[28] Findeisen W., Szymanowski J., Wierzbicki A.: Teoria i Metody Obliczeniowe Optymalizacji, PWN, Warszawa 1977.

[29] Fong A.C., Ullman J.D.: Finding the depth of a flow graph, J. Computer System Sci. 15 1977, 300-309.

[30] Fosdick L.D., Osterweil L.J.: Data flow analysis in software reliability, Computing Surveys 8,3 (Sept.) 1976, 305-330.

[31] Gabasow R., Kirillowa F.M.: Metody Optimizacji, Izd. BGU im. W.I.Lenina, Mińsk 1975.

[32] Goodman S.E., Hedetniemi S.T.: Introduction to the Design and Analysis of Algorithms, McGraw-Hill, N.Y. 1977.

[33] Grabowski W.: Programowanie Matematyczne, PWE, Warszawa 1980.

[34] Graham S.L., Wegman M.: A fast and usually linear algorithm for global flow analysis, J. ACM 23,1 (Jan.) 1976, 172-202.

[35] Gries D.: Compiler Construction for Digital Computers, John Wiley, 1971, N.Y.

[36] Harrison W.H.: Compiler analysis of the value ranges for variables, IEEE Trans. Software Eng. SE-3,3 (May), 1977, 243-250.

[37] Hecht M.S.: Flow Analysis of Computer Programs, North-Holland, N.Y. 1977.

[38] Hecht M.S., Shaffer J.B.: A modest quad improver for SIMPL-T, Dept. of Comp. Sci., Univ. of Maryland, College Park, Md. 1976.

[39] Hecht M.S., Ullman J.D.: Analysis of a simple algorithm for global data flow problems, w Conf. Rec. ACM Symp. on Principles of Programming Languages, Boston, Mass., Oct., 1973, 207-217.

[40] Hecht M.S., Ullman J.D.: Characterizations of reducible flow graphs, J. ACM 21, 3 (July) 1974, 367-375.

[41] Hecht M.S., Ullman J.D.: A simple algorithm for global data flow analysis problems, SIAM J. Comput. 4, 4 (Dec.), 1975, 519-532.

[42] Holley L.H., Rosen B.K.: Qualified data flow problems, IEEE Trans. Software Eng. SE-7, 1 (Jan.), 1981, 60-78.

[43] Kam J.B., Ullman J.D.: Global data flow analysis and iterative algorithms, J. ACM 23, 1 (Jan.), 1976, 158-171.

[44] Kam J.B., Ullman J.D.: Monotone data flow analysis frameworks, Acta Informatica 7, 1977, 305-317.

[45] Kennedy K.: A global flow analysis algorithm, Intern. J. Computer Math. 3, Sec. A, 1971, 5-15.

[46] Kennedy K.: Node listing applied to data flow analysis, Conf. Rec. 2nd ACM Symp. on Principles of Programming Languages, Palo Alto, CA, Jan., 1975, 10-21.

[47] Kennedy K.: A comparison of two algorithms for global data flow analysis, SIAM J. Comput. 5, 1 (March.), 1976, 158-180.

[48] Kennedy K.: Use-definition chains with applications, Computer Languages 3, 1978, 163-179.

[49] Kildall G.A.: A unified approach to global program optimization, w Conf. Rec. ACM Symp. on Principles of Programming Languages, Boston, Mass., Oct. 1973, 194-206.

[50] Knuth D.E.: An empirical study of FORTRAN programs; Software-Practice and Experience 1, 2, 1971, 105-133.

[51] Kou L.T.: On live-dead analysis for global data flow problems, J. ACM 24, 3 (July) 1977, 473-483.

[52] Lipski W.: Kombinatoryka dla Programistów, WNT, Warszawa 1982.

[53] Lomet D.B.: Data flow analysis in the presence of procedure calls, IBM J. Res. Develop. 21, Nov. 1977, 559-571.

[54] Mangasarian O.L.: Nonlinear Programming, McGraw-Hill, N.Y., 1969.

[55] Morel E., Renvoise C.: Global optimization by suppression of partial redundancies, Comm. ACM 22, 2 (Feb.) 1970, 96-103.

[56] Muchnick S.S., Jones N.D., Eds. Program Flow Analysis: Theory and Applications, Prentice-Hall, Englewood Cliffs, N.J. 1981.

[57] Murdie J.A.: Functional specification of the York Ada workbench compiler, York Computer Science Report No 54, University of York, 1982.

[58] Myers E.W.: A precise inter-procedural data flow algorithm, w Conf. Rec. 8th ACM Symp. on Principles of Programming Languages, Wiliamsburg, Virginia, Jan., 1981, 219-230.

[59] Nielson F.: A denotational framework for data flow analysis, Acta Informatica 18, 1982, 265-287.

[60] Rabin M.O.: Complexity of computations, Comm. ACM 20, 9 (Sept.)1977, 625-633.

[61] Ramanathan J., Kennedy K.: Pathlistings applied to data flow analysis, Acta Informatica 16, 1981, 253-273.

[62] Reinhard W.: Computation and use of data flow information in optimizing compilers, Acta Informatica 12, 1979, 209-225.

[63] Rosen B.K.: Data flow analysis for recursive PL/I programs, Res.Rep. RC 5211, IBM T.J. Watson Res. Ctr., Yorktown Heihgts, N.Y., Jan. 1975a.

[64] Rosen B.K.: High level data flow analysis, Part 1 (Classical structured programming), Res. Rep. RC 5598, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., August 1975b.

[65] Rosen B.K.: High level data flow analysis, Part 2 (Escapes and jumpes), Res. Rep. RC 5744, IBM T.J. Watson Res. Ctr., Yorktown Heights N.Y., April 1976.

[66] Rosen B.K.: Arcs in graphs are not pairs of nodes, SIGACT News 9, 3, 1977a, 25-27.

[67] Rosen B.K.: High-level data flow analysis, Comm. ACM 20, 10 (Oct.), 1977b, 712-724.

[68] Rosen B.K.: Data flow analysis for procedural languages, J. ACM 26, 2 (April) 1979, 322-344.

[69] Schaefer M.: A Mathematical Theory of Global Program Optimization, Prentice-Hall, Englewood Cliffs, N.J. 1973.

[70] Sharir M.: Structural analysis: A new approach to flow analysis in optimizing compilers, Computer Languages 5, 1980, 141-153.

[71] Szamkołowicz L.: Teoria Grafów Skończonych, Ossolineum, Wrocław 1971.

[72] Taniguchi K., Kasami T.: An O(n) algorithm for computing the set of available expressions of D-charts, Acta Informatica 6, 1976, 361-364.

[73] Tarjan R.E.: Edge-disjoint spanning trees and depth-first search, Acta Informatica 6, 1976, 171-185.

[74] Ullman J.D.: Fast algorithms for the elimination of common subexpressions, Acta Informatica 2, 1973, 191-213.

[75] Ullman J.D.: A survey of data flow analysis techniques, Proc. 2nd USA-Japan Comp. Conf., AFIPS Press, Montvale, N.J. 1975, 335-342.

[76] Wadia A.B.: Generation of node lists using segment analysis, Computer Languages, 5, 1980, 115-129.

[77] Warren J.S., Jr.: Three simple node list algorithms, Computer Languages 3, 1978, 115-126.

[78] Wulf W., at al.: The Design of an Optimizing Compiler, American Elsevier, N.Y. 1975.

Dodatek A

Lemat A.1. Punkt $\hat{x}^4$ (tablica 3.1) nie jest punktem minimum dla problemu (3.5).

Dowód. Gdyby punkt $\hat{x}^4$ był punktem minimum dla problemu (3.5), to musiałby on być punktem lokalnego minimum funkcji $T(x)$, ponieważ żadne z ograniczeń (3.3) nie jest aktywne w tym punkcie. Lecz, aby stacjonarny punkt $\hat{x}$ funkcji $T(x)$ był lokalnym minimum, to muszą być jednocześnie spełnione następujące nierówności[1]:

$$\frac{\partial^2 T(\hat{x})}{\partial \hat{x}^2} > 0, \qquad |\nabla^2 T(\hat{x})| > 0.$$

Tylko pierwsza z tych nierówności spełniona jest w punkcie $\hat{x}^4$:

$$\frac{\partial^2 T(\hat{x}^4)}{\partial (\hat{x}^4)^2} = 9 > 0, \qquad \begin{vmatrix} 9 & -1 \\ -1 & 0 \end{vmatrix} = -1 < 0.\square$$

Lemat A2. Punkt $\hat{x}^2$ jest punktem minimum dla problemu (3.5), a więc $\hat{x}^2 = \bar{x}^2$.

Dowód. Ponieważ punkt $\hat{x}^2$ jest punktem stacjonarnym funkcji $L(x,y)$, to należy pokazać, że spełniony jest wystarczający warunek optymalności. Jedynym aktywnym ograniczeniem jest $n_1 - m < 0$, a więc z równań hiperpłaszczyzny (3.8) mamy

$$[-1,\ 1]\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = -z_1 + z_2 = 0 \Rightarrow z_2 = z_1.$$

Korzystając z tego rezultatu przy określaniu formy kwadratowej otrzymujemy

$$[z_1,\ z_2]\begin{bmatrix} 9 & -1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = 9z_1^2 - 2z_1z_2 = 7z_1^2.$$

Forma ta jest dodatnio określona dla każdego $z_1 \neq 0$, a więc wystarczający warunek optymalności jest spełniony.$\square$

---

[1] $|\nabla^2 T(\hat{x})|$ oznacza wyznacznik macierzy $\nabla^2 T(\hat{x})$.

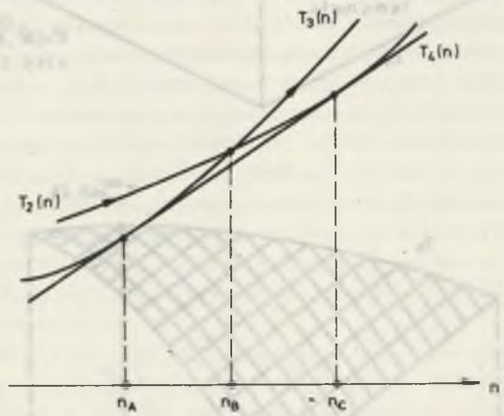
Należy zauważyć, że dla dowolnego n z przedziału $7.083 \approx n_A \leqslant n \leqslant n_C = 7.3$ wystarczający warunek optymalności jest spełniony nie tylko dla punktu $\hat{x}^2$, lecz również dla punktu $\hat{x}^3$. W celu stwierdzenia, który z punktów wyznacza maksimum T_R (lub minimum T), należy porównać wartości funkcji $T_R(\hat{x}^2)$ i $T_R(\hat{x}^3)$ w tym przedziale. Niech $T_i(n)$ oznacza $T_R(\hat{x}^i)$ dla $i = 2,3,4$. Prawdziwe są następujące nierówności:

$$T_4(n_A) = T_3(n_A) < T_2(n_A)$$

$$T_4(n_B) < T_2(n_B) = T_3(n_B)$$

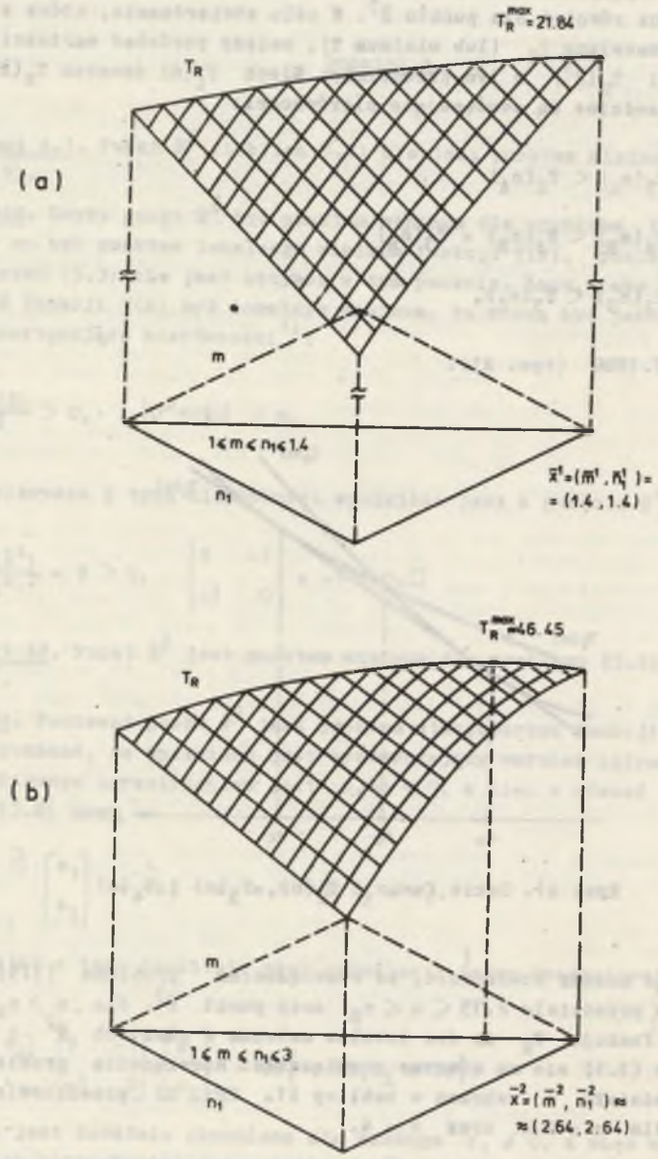
$$T_4(n_C) = T_2(n_C) < T_3(n_C),$$

gdzie $n_B \approx 7.1886$ (rys. A1).



Rys. A1. Szkic funkcji $T_2(n)$, $T_3(n)$ i $T_4(n)$

Reasumując możemy stwierdzić, że rozwiązaniem problemu (3.5) jest punkt $\hat{x}^2$ w przedziale $2.75 \leqslant n \leqslant n_B$ oraz punkt $\hat{x}^3$ dla $n > n_B$. Jeśli $n = n_B$, to funkcja T_R ma dwa lokalne maksima w punktach $\hat{x}^2$ i $\hat{x}^3$, a więc problem (3.5) nie ma wówczas rozwiązania. Rozwiązania problemu dla każdego przedziału n zebrano w tablicy A1. Rys. A2 przedstawia postać funkcji T_R dla $n = 2.4$ oraz $n = 4$.



Rys. A2. Postać funkcji T_R dla (a) n = 2.4 oraz (b) n = 4

Tablica A1

Rozwiązanie problemu (3.5)

	$\bar{m}$	$\bar{n}_1$	Przedział ważności rozwiązania	$T_R(\bar{x})$
$\bar{x}^1$	n - 1.	n - 1	$2 \leqslant n \leqslant 2.75$	$1.5n^2 + 5.5n$
$\bar{x}^2$	$\frac{5n - 1.5}{7}$	$\frac{5n - 1.5}{7}$	$2.75 \leqslant n < 7.1886$	$\frac{25n^2+55n+30.25}{14}$
$\bar{x}^3$	$\frac{6n + 2.5}{9}$	n - 1	$7.1886 < n$	$\frac{36n^2+30n+132.25}{18}$

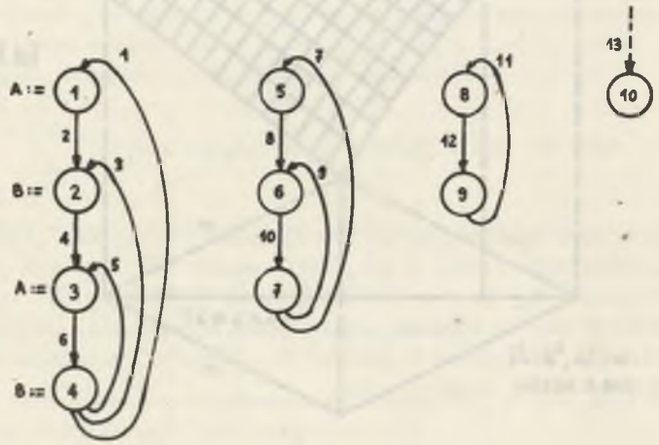
Podziękowania. Dziękuję Doc. J. Klamce za dyskusję nad rozwiązaniami problemu (3.5).

Dodatek B

Niniejszy dodatek zawiera wyniki weryfikacji algorytmu analizy regionów dla problemu definicji osiągających otrzymane po zaimplementowaniu algorytmu w języku Ada z użyciem kompilatora York Ada (Briggs at al. [1983], Murdie [1982]) w systemie VAX 11/780 UNIX. Celem weryfikacji było stwierdzenie poprawności działania algorytmu na kilku przykładowych grafach przepływu oraz potwierdzenie prawidłowości wyznaczonej teoretycznie złożoności czasowej algorytmu. Obliczenia wykonanego programu przeprowadzono dla grafów: muszelkowego o poziomach 2 i 3, podwójnie wiązanego o poziomie 3 oraz spiralnego o poziomie 2. We wszystkich przypadkach otrzymane wektory definicji osiągających były prawidłowe. Również liczby operacji wykonanych w programie w celu obliczenia danych wynikowych były zgodne z wzorami określającymi złożoność algorytmu dla poszczególnych grafów.

Zanim przedstawimy tekst programu oraz wyniki jego obliczeń, omówimy w skrócie struktury danych, jakie są używane w programie.

Wszystkie wektory bitów, takie jak RDEFEN, RDEFEX, GEN, TRANS itp., zostały zamodelowane z użyciem jednowymiarowych, statycznych tablic o wartościach boolowskich (bitowi 1 odpowiada wartość TRUE (w skrócie T), a bitowi 0 - wartość FALSE (w skrócie F)). Dla przykładu, w grafie mu-



Rys. B1. Sekwencja grafów pochodnych dla grafu muszelkowego o poziomie p = 3

szelkowym o poziomie 3 (rys. B1), obliczony wektor definicji osiągających wejście wierzchołka 2 był równy RDEFEN(2) = (TFTT). Oznacza to, że wierzchołek ten osiąga definicja zmiennej A występująca w wierzchołku 1, definicja zmiennej A występująca w wierzchołku 3 oraz definicja zmiennej B występująca w wierzchołku 4. Przyjęto, że numery definicji są takie same, jak numery bloków, w których one występują. Zgodnie z tą numeracją poszczególnym definicjom odpowiadają kolejne wartości boolowskie w tablicach.

Jednowymiarowa, dynamiczna tablica NODES_DATA, której elementami są rekordy (type NODE_DATA) składające się z wektorów TRANS i GEN, zawiera informacje o tym, jakie definicje są transmitowane oraz generowane w poszczególnych blokach podstawowych analizowanego programu.

Jednowymiarowa, dynamiczna tablica GRAPHS zawiera numery pierwszej i ostatniej krawędzi dla każdego grafu w sekwencji grafów pochodnych. Przyjęto, że zarówno wierzchołki, jak i krawędzie grafów są ponumerowane (niezależnie od siebie) kolejnymi liczbami naturalnymi poczynając od wierzchołków (krawędzi) grafu pierwszego rzędu, a kończąc na wierzchołku (krawędzi[1]) grafu granicznego (por. rys. B1).

Jednowymiarowa, dynamiczna tablica rekordów EDGES_TABLE zawiera informacje o krawędziach grafu pierwszego rzędu oraz grafów pochodnych. Każdej krawędzi odpowiada pojedynczy rekord występujący w tablicy EDGES_TABLE na pozycji wyznaczonej przez numer tej krawędzi. W skład informacji o krawędzi wchodzą numery wierzchołków grafu, z których krawędź, odpowiednio wychodzi (FROM_NODE) i dochodzi (TO_NODE), wartości boolowskie określające czy wierzchołek, do którego krawędź dochodzi jest głową interwału (HEAD) oraz czy wierzchołek ten ma następniki w grafie (SUCC). Ponadto, dla każdego wierzchołka TO_NODE w grafach pochodnych podane są numery krawędzi początkowej i końcowej regionu w grafie pierwszego rzędu, reprezentowanego przez ten wierzchołek.

Tablica EDGES_TABLE odgrywa podstawową rolę w programie. W fazach 1 i 2 przetwarzane są kolejne krawędzie tej tablicy, przy czym w czasie przetwarzania krawędzi w grafach pochodnych występuje ponowne przetwarzanie krawędzi odpowiednich regionów w grafie pierwszego rzędu. Dla poprawności działania programu istotne jest właściwe uporządkowanie krawędzi w tablicy. Musi być ono zgodne z uporządkowaniem interwałowym wierzchołków w poszczególnych grafach. Ponadto krawędzie wejściowe każdego wierzchołka muszą bezpośrednio poprzedzać jego krawędzie wyjściowe.

Dodatkowe informacje o strukturach danych zawarte są w komentarzach zamieszczonych w programie. Tekst programu jest następujący:

[1] W celu uproszczenia programu przyjęto, że do wierzchołka grafu granicznego dochodzi (w rzeczywistości nieistniejąca) pojedyncza krawędź.

```
with TEXT_IO;
procedure MAIN is
  use TEXT_IO; use INTEGER_IO;
  -- Declaration of static input data.
  LEVEL_P_OF_GRAPH: INTEGER;
  NO_OF_DEFS       : constant INTEGER := 4; -- Number of definitions
                                            -- in a program[1].
  NO_F_NODES       : INTEGER; -- Number of nodes in the 1st order graph.
  type VECTOR_OF_BITS is array (1 .. NO_OF_DEFS) of BOOLEAN;
  type NODE_DATA is -- For each 1st order graph node.
    record
      TRANS: VECTOR_OF_BITS: -- A bit in the vector is TRUE if defi-
                             -- nition is transmitted through the node.
      GEN  : VECTOR_OF_BITS; -- A bit in the vector is TRUE if a defi-
                             -- nition occurs in the mode.
    end record;
  NO_OF_GRAPHS: INTEGER; -- Number of graphs in the derived sequence.
  type GRAPH is -- For each graph.
    record
      FIRST_EDGE: INTEGER; -- Index of the first edge.
      LAST_EDGE:  INTEGER; -- Index of the last edge.
    end record;
  NO_OF_EDGES: INTEGER; -- Number of edges in the 1st order and
                        -- derived graphs.
  type CORRES_REGION is
    record
      INITIAL_EDGE: INTEGER; -- Indices for the initial and final
      FINAL_EDGE  : INTEGER; -- edges of the corresponding region in
                             -- the first order graph.
    end record;
  type EDGE is -- For each edte in the 1st order or derived graph.
    record
      FROM_NODE: INTEGER; -- The node it comes from.
      TO_NODE  : INTEGER; -- The node it comes to.
      HEAD: BOOLEAN; -- TRUE if TO_NODE is an interval head.
      SUCC: BOOLEAN; -- TRUE if TO_NODE has successor(s).
      COR_REG: CORRES_REGION;
    end record;
```

[1] Ponieważ w aktualnej wersji kompilatora York Ada (listopad 1983) nie zostały jeszcze zaimplementowane tablice dynamiczne zawarte w rekordach, niezbędne było zadeklarowanie liczby definicji w programie NO_OF_DEFS - jako stałej. Stałą tę należy zmodyfikować przed wykonaniem programu dla grafu o innej liczbie definicji.


```
  -- Declaration of static local data.
  SIGMA_RDEFEX, SIGMA_PATHEX: VECTOR_OF_BITS;
  T: constant BOOLEAN := TRUE;
  F: constant BOOLEAN := FALSE;
  -- Declaration of static output data.
  NO_OF_OP: INTEGER; -- Number of bit vector operations executed.
  -- Declaration of local IO procedures.
  procedure READ_BOOL(B: out BOOLEAN)is
    C: CHARACTER;
  begin
    GET(C);
    if C = 'T' then
      B := TRUE;
    else
      B := FALSE;
    end if;
  end READ_BOOL;
  procedure WRITE_BOOL(B: in BOOLEAN) is
  begin
    if B then
      PUT('T');
    else
      PUT('F');
    end if;
  end WRITE_BOLL;
  -- End of static data declarations
  -- and local procedure declarations.
begin
  -- Reading input data.
  GET(LEVEL_P_OF_GRAPH);
  GET(NO_OF_NODES);
  GET(NO_OF_GRAPHS);
  GET(NO_OF_EDGES);
  declare
    -- Dynamic input data.
    NODES_DATA : array (1 .. NO_OF_NODES) of NODE_DATA;
    GRAPHS     : array (1 .. NO_OF_GRAPHS) of GRAPH;
    EDGES_TABLE: array (1 .. NO_OF_EDGES) of EDGE;
    -- Dynamic local data.
    PATHEN, PATHEX: array (1 .. NO_OF_NODES) of VECTOR_OF_BITS;
```

```
  -- Dynamic output data.
  RDEFEN, RDEFEX: array (1 .. NO_OF_NODES) of VECTOR_OF_BITS;
begin
  for N in 1 .. NO_OF_NODES loop
    for D in 1 .. NO_OF_DEFS loop
      READ_BOOL(NODES_DATA(N).TRANS(D));
    end loop;
    for D in 1 .. NO_OF_DEFS loop
      READ_BOOL(NODES_DATA(N).GEN(D));
    end loop;
  end loop;
  for G in 1 .. NO_OF_GRAPHS loop
    GET(GRAPHS(G).FIRST_EDGE);
    GET(GRAPHS(G).LAST_EDGE);
  end loop;
  for E in 1 .. NO_OF_EDGES loop
    GET(EDGES_TABLE(E).FROM_NODE);
    GET(EDGES_TABLE(E).TO_NODE);
    READ_BOOL(EDGES_TABLE(E).HEAD);
    READ_BOOL(EDGES_TABLE(E).SUCC);
    GET(EDGES_TABLE(E).COR_REG.INITIAL_EDGE);
    GET(EDGES_TABLE(E).COR_REG.FINAL_EDGE);
  end loop;
  -- Writing input data.
  NEW_LINE;
  PUT("-- INPUT DATA --"); NEW_LINE(2);
  PUT("LEVEL_P_OF_GRAPH is");
  PUT(LEVEL_P_OF_GRAPH, 3); NEW_LINE;
  PUT("NO_OF_DEFS is ");
  PUT(NO_OF_DEFS, 3); NEW_LINE;
  PUT("NO_OF_NODES is ");
  PUT(NO_OF_NODES, 3); NEW_LINE(2);
  for N in 1 .. NO_OF_NODES loop
    PUT("NODES_DATA("); PUT(N, 3); PUT(") is"); NEW_LINE;
    PUT("   TRANS: (");
    for D in 1 .. NO_OF_DEFS loop
      WRITE_BOOL(NODES_DATA(N).TRANS(D));
    end loop;
    PUT(")"); NEW_LINE; PUT("   GEN  : (");
    for D in 1 .. NO_OF_DEFS loop
      WRITE_BOOL(NODES_DATA(N).GEN(D));
    end loop;
    PUT(")"); NEW_LINE;
```



```
  end loop;
  NEW_LINE;
  PUT("NO_OF_GRAPHS is ");
  PUT(NO_OF_GRAPHS, 3); NEW_LINE(2);
  for G in 1 .. NO_OF_GRAPHS loop
    PUT("GRAPHS("); PUT(G, 3); PUT(") is"); NEW_LINE;
    PUT("   FIRST_EDGE: ");
    PUT(GRAPHS(G).FIRST_EDGE, 3); NEW_LINE;
    PUT("   LAST_EDGE : ");
    PUT(GRAPHS(G).LAST_EDGE, 3); NEW_LINE;
  end loop;
  NEW_LINE;
  PUT("NO_OF_EDGES is ");
  PUT(NO_OF_EDGES, 3); NEW LINE(2);
  for E in 1 .. NO_OF_EDGES loop
    PUT("EDGES_TABLE("); PUT(E, 3); PUT(") is"); NEW_LINE;
    PUT("   FROM_NODE: ");
    PUT(EDGES_TABLE(E).FROM_NODE, 3); NEW_LINE;
    PUT("   TO_NODE : ");
    PUT(EDGES_TABLE(E).TO_NODE, 3); NEW_LINE;
    PUT("   HEAD    : ");
    WRITE_BOOL(EDGES_TABLE(E).HEAD); NEW_LINE;
    PUT("   SUCC    : ");
    WRITE_BOOL(EDGES_TABLE(E).SUCC); NEW_LINE;
    PUT("      INITIAL_EDGE: ");
    PUT(EDGES_TABLE(E).COR_REG.INITIAL_EDGE, 3); NEW_LINE;
    PUT("      FINAL_EDGE  : ")
    PUT(EDGES_TABLE(E).COR_REG.FINAL_EDGE, 3); NEW_LINE;
  end loop;
  NO_OF_OP := 0;
  --Phase 1:  Processes the first order intervals (minimum regions)
  --         of the program flow graph.
  for D in 1 .. NO_OF_DEFS loop
    SIGMA_RDEFEX(D) := F;
    SIGMA-PATHEX(D) := F;
  end loop;
  for EDGE_1 in GRAPHS(1).FIRST_EDGE .. GRAPHS(1).LAST_EDGE loop
    -- For each edge in the first order graph.
    -- EDGE_1 denotes the current edge in the first order graph.
    declare
    FROM: INTEGER renames EDGES_TABLE(EDGE_1).FROM NODE;
    TO  : INTEGER renames EDGES_TABLE(EDGE_1).TO_NODE;
  begin
    if not EDGES TABLE(EDGE_1).HEAD then
      <<A5_1>> SIGMA_RDEFEX := SIGMA_RDEFEX or RDEFEX(FROM);
      <<A6_1>> SIGMA_PATHEX := SIGMA_PATHEX or PATHEX(FROM);
```

```
    NO_OF_OP := NO_OF_OP + 2;
  end if;
  if EDGE_1 = GRAPHS(1).LAST_EDGE or else
     TO /= EDGES_TABLE(EDGE_1 + 1).TO_NODE then
    if EDGES_TABLE(EDGE_1).HEAD then
      -- For the header node.
      <<A1_A2>> for D in 1 .. NO_OF_DEFS loop
                  RDEFEN(TO)(D) := F;
                  PATHEN(TO)(D) := T;
                end loop;
      if EDGES_TABLE(EDGE_1).SUCC then
        <<A3>> RDEFEX(TO) := NODES_DATA(TO).GEN;
        <<A4>> PATHEX(TO) := NODES_DATA(TO).TRANS;
        end if;
      else
        -- For each node other than the header.
        NO_OF_OP := NO_OF_OP - 2;
        <<A5>> RDEFEN(TO) := SIGMA_RDEFEX;
        <<A6>> PATHEN(TO) := SIGMA_PATHEX;
        if EDGES_TABLE(EDGE_1).SUCC then
          <<A7>> RDEFEX(TO) := (RDEFEN(TO) and NODES_DATA(TO).TRANS)
                               or NODES_DATA(TO).GEN;
          <<A8>> PATHEX(TO) := PATHEN(TO) and NODES_DATA(TO).TRANS;
          NO_OF_OP:= NO_OF_OP + 3;
        end if;
        for D in 1 .. NO_OF_DEFS loop
          SIGMA_RDEFEX(D) := F;
          SIGMA_PATHEX(D) := F;
        end loop;
      end if;
    end if;
  end;
end loop;
-- The end of Phase 1.
-- Phase 2: Iterates through the derived sequence of the program flow
--          graph in inner-to-outer order.
for G in 2 .. NO_OF_GRAPHS loop
  for EDGE_D in GRAPHS(G).FIRST_EDGE .. GRAPHS(G).LAST_EDGE loop
    -- EDGE_D denotes the current edge in the derived graphs.
    if not EDGES_TABLE(EDGE_D).HEAD and
       (EDGE_D = GRAPHS(G).LAST_EDGE or else
       EDGES_TABLE(EDGE_D).TO_NODE /=
       EDGES_TABLE)EDGE_D + 1).TO_NODE) then
       for EDGE_1 in EDGES_TABLE(EDGE_D).COR_REG.INITIAL_EDGE ..
                     EDGES_TABLE(EDGE_D).COR_REG.FINAL_EDGE loop
         -- For each edge in the corresponding region.
         declare
           TO: INTEGER renames EDGES_TABLE(EDGE_1).TO_NODE;
```



```
         begin
           if EDGES_TABLE(EDGE_1).HEAD then
             <<B1_1>> SIGMA_RDEFEX := SIGMA_RDEFEX or
                      RDEFEX(EDGES_TABLE(EDGE_1).FROM_NODE);
             NO_OF_OP := NO_OF_OP + 1;
           end if;
           if EDGE_1 = EDGES TABLE(EDGE_D).COR_REG.FINAL_EDGE or else
              TO /= EDGES_TABLE(EDGE_1 + 1).TO_NODE then
             if EDGES_TABLE(EDGE_1).HEAD then
               -- For the header node.
               NO_OF_OP := NO_OF_OP - 1;
               <<B1>> RDEFEN(TO) := SIGMA_RDEFEX;
               if EDGES_TABLE(EDGE_1).SUCC then
                 <<B2>> RDEFEX(TO) := RDEFEX(TO) or
                                      (SIGMA_RDEFEX and PATHEX(TO));
                 NO_OF_OP := NO_OF_OP + 2;
               end if;
               if EDGE_1 = EDGES_TABLE(EDGE_D).COR_REG.FINAL_EDGE
                 or else EDGES_TABLE(EDGE_1 + 1).HEAD then
                 for D in 1 .. NO_OF_DEFS loop
                   SIGMA_RDEFEX(D) := F;
                 end loop;
               end if;
             else
               -- For each node other than the header.
               <<B3>> RDEFEN(TO) := RDEFEN(TO) or
                                    (SIGMA_RDEFEX and PATHEN(TO));
               NO_OF_OP := NO_OF_OP + 2;
               if EDGES_TABLE(EDGE_1).SUCC then
                 <<B4>> RDEFEX(TO) := RDEFEX(TO) or
                                      (SIGMA RDEFEX and PATHEX(TO));
                 NO_OF_OP := NO_OF_OP + 2;
                 if EDGE_1 = EDGES_TABLE(EDGE_D).COR_REG.FINAL_EDGE
                   or else EDGES_TABLE(EDGE_1 + 1).HEAD then
                   for D in 1 .. NO_OF_DEFS loop
                     SIGMA RDEFEX(D) := F;
                   end loop;
                 end if;
               end if;
             end if;
           end if;
         end;
       end loop;
       -- The end of processing a region.
     end if;
```

```
    end loop;
    -- The end of processing a graph.
  end loop;
  -- The end of processing all graphs.
  --The end of Phase 2.
  -- Writing output data.
  NEW_LINE;
  PUT("-- OUTPUT DATA --"); NEW_LINE(2);
  for N in 1 .. NO_OF_NODES loop
    PUT("RDEFEN("); PUT(N, 3); PUT(") is (");
    for D in 1 .. NO_OF_DEFS loop
      WRITE_BOOL(RDEFEN(N)(D));
    end loop;
    PUT(")"); NEW_LINE;
    PUT("RDEFEX("); PUT(N, 3); PUT(") is (");
    for D in 1 .. NO_OF_DEFS loop
      WRITE_BOOL(RDEFEX(N)(D));
    end loop;
    PUT(")"); NEW_LINE;
  end loop;
  NEW_LINE; put ("NO_OF_OP is ");
  PUR(NO_OF_OP, 4);
  NEW_LINE;
 end;
end MAIN;
```

Jak już wspomniano, dokonano obliczeń programu dla czterech przykładowych grafów przepływu sterowania. Poniżej zamieszczamy uzyskane wyniki obliczeń, przy czym tylko dla grafu muszelkowego o poziomie 3 są one pełne, tj. zawierają również wydruki wprowadzonych danych wejściowych. Numerację wierzchołków i krawędzi w grafie przedstawia rys. B1.

Dla oszczędności miejsca, pozostałe wyniki zamieszczono w postaci skrótowej. Numeracja wierzchołków i krawędzi grafów w tych przypadkach była podobna jak na rys. B1. Przyjęto również, że we wszystkich wierzchołkach grafu pierwszego rzędu o numerach nieparzystych występuje definicja zmiennej A, zaś w wierzchołkach o numerach parzystych - definicja zmiennej B.

Wyniki obliczeń dla grafu muszelkowego o poziomie 3:

```
-- INPUT DATA --
LEVEL_P_OF_GRAPH is 3
NO_OF_DEFS is 4
NO_OF_NODES is 4
```



```
NODES_DATA(1) is
   TRANS: (FTFT)
   GEN  : (TFFF)
NODES_DATA(2) is
   TRANS: (TFTF)
   GEN  : (FTFF)
NODES_DATA(3) is
   TRANS: (FTFT)
   GEN  : (FFTF)
NODES_DATA(4) is
   TRANS: (TFTF)
   GEN  : (FFFT)
NO_OF_GRAPHS is 4
GRAPHS(1) is
   FIRST_EDGE: 1
   LAST_EDGE : 6
GRAPHS(2) is
   FIRST_EDGE: 7
   LAST_EDGE: 10
GRAPHS(3) is
   FIRST_EDGE: 11
   LAST_EDGE : 12
GRAPHS(4) is
   FIRST_EDGE: 13
   LAST_EDGE : 13
NO_OF_EDGES is 13
EDGES_TABLE(1) is
   FROM_NODE: 4
   TO_NODE  : 1
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 0
      FINAL_EDGE  : 0
   EDGES_TABLE(2) is
   FROM_NODE: 1
   TO_NODE  : 2
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 0
      FINAL_EDGE  : 0
```

```
EDGES_TABLE(3) is
   FROM_NODE: 4
   TO_NODE  : 2
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 0
      FINAL_EDGE  : 0
EDGES_TABLE(4) is
   FROM_NODE: 2
   TO_NODE  : 3
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 0
      FINAL_EDGE  : 0
EDGES_TABLE(5) is
   FROM_NODE: 4
   TO_NODE  : 3
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 0
      FINAL_EDGE  : 0
EDGES_TABLE(6) is
   FROM_NODE: 3
   TO_NODE  : 4
   HEAD     : F
   SUCC     : T
      INITIAL_EDGE: 0
      FINAL_EDGE  : 0
EDGES_TABLE(7) is
   FROM_NODE: 7
   TO_NODE  : 5
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 1
      FINAL_EDGE  : 1
EDGES_TABLE(8) is
   FROM_NODE: 5
   TO_NODE  : 6
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 2
      FINAL_EDGE  : 3
```



```
EDGES TABLE(9) is
   FROM_NODE: 7
   TO_NODE  : 6
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 2
      FINAL_EDGE  : 3
EDGES_TABLE(10) is
   FROM_NODE: 6
   TO_NODE  : 7
   HEAD     : F
   SUCC     : T
      INITIAL_EDGE: 4
      FINAL_EDGE  : 6
EDGES_TABLE (11) is
   FROM_NODE: 9
   TO_NODE  : 8
   HEAD     : T
   SUCC     : T
      INITIAL_EDGE: 1
      FINAL_EDGE  : 1
EDGES_TABLE (12) is
   FROM_NODE: 8
   TO_NODE  : 9
   HEAD     : F
   SUCC     : T
      INITIAL_EDGE: 2
      FINAL_EDGE  : 6
EDGES_TABLE(13) is
   FROM_NODE: 0
   TO_NODE  : 10
   HEAD     : F
   SUCC     : T
      INITIAL_EDGE: 1
      FINAL_EDGE  : 6

-- OUTPUT DATA --

RDEFEN(1) is (FFTT)
RDEFEX(1) is (TFFT)
RDEFEN(2) is (TFTT)
RDEFEX(2) is (TTTF)
RDEFEN(3) is (TTTT)
RDEFEX(3) is (FTTT)
RDEFEN(4) is (FTTT)
RDEFEX(4) is (FFTT)
```

NO_OF_OP is 32

Zwróćmy uwagę, że liczba operacji (na wektorach) wykonanych w programie jest zgodna z liczbą operacji wyznaczoną teoretycznie. Mamy bowiem: $1.5p^2 + 5.5p + 2$ dla $p = 3$ wynosi 32 (por. tablica 3.2).

Wyniki obliczeń dla grafu muszelkowego o poziomie 2:

```
-- OUTPUT DATA --
RDEFEN(1) is (FTT)
RDEFEX(1) is (TTF)
RDEFEN(2) is (TTT)
RDEFEX(2) is (TTT)
RDEFEN(3) is (TTT)
RDEFEX(3) is (FTT)
NO_OF_OP is 19
```

Mamy: $1.5p^2 + 1.5p + 2$ dla $p = 2$ wynosi 19.

Wyniki obliczeń dla grafu podwójnie wiązanego o poziomie 3:

```
-- OUTPUT DATA --
RDEFEN(1) is (TTTF)
RDEFEX(1) is (TTFF)
RDEFEN(2) is (TTTT)
RDEFEX(2) is (TTTF)
RDEFEN(3) is (TTTT)
RDEFEX(3) is (FTTT)
RDEFEN(4) is (FTTT)
RDEFEX(4) is (FFTT)
NO_OF_OP is 32
```

Mamy: $1.5p^2 + 5.5p + 2$ dla $p = 3$ wynosi 32.

Wyniki obliczeń dla grafu spiralnego o poziomie 2:

```
-- OUTPUT DATA --
RDEFEN(1) is (TFTTF)
RDEFEX(1) is (TFFTF)
RDEFEN(2) is (TTTTF)
RDEFEX(2) is (TTTFF)
RDEFEN(3) is (TTTFF)
RDEFEX(3) is (FTTFF)
RDEFEN(4) is (TTTFF)
RDEFEX(4) is (TFTTF)
```



```
RDEFEN(5) is (TFTTF)
RDEFEX(5) is (FFFFF)
NO_OF_OP is 34
```

Mamy: $3p^2 + 11p$ dla $p = 2$ równe jest 34.

ALGORYTM ANALIZY REGIONÓW DLA PROBLEMÓW ANALIZY PRZEPŁYWU DANYCH

S t r e s z c z e n i e

Zaproponowano nowy algorytm, zwany algorytmem analizy regionów, przeznaczony do rozwiązywania problemów analizy przepływu danych dla redukowalnych grafów przepływu. Udowodniono częściową poprawność algorytmu oraz dochodzenie jego obliczeń do punktu końcowego dla problemów definicji osiągających oraz zmiennych aktywnych. Pokazano, że pesymistyczna złożoność czasowa algorytmu jest $O(n^2)$ dla obu problemów. Ponadto dokonano porównań algorytmu pod względem złożoności czasowej z dwoma dobrze znanymi algorytmami rozwiązywania wyżej wymienionych problemów. Porównania te przeprowadzono w oparciu o kilka rodzin samopowielających się, redukowalnych grafów przepływu. Prezentowany algorytm został porównany z algorytmem Allena-Cocke'a dla problemu definicji osiągających (Allen, F.E., Cocke J. [1976] A program data flow analysis procedure, Comm. ACM 19, 3, March, 137-147) oraz z wersją "round-robin" algorytmu iteracyjnego dla problemu zmiennych aktywnych (Hecht, M.S., Ullman, J.D. [1975] A simple algorithm for global data flow analysis problems, SIAM J. Comput. 4,4, Dec., 519--532). W obu przypadkach wyniki porównań nie są rozstrzygające, ponieważ pokazują, że algorytm analizy regionów wymaga zrealizowania asymptotycznie mniejszej liczby operacji na wektorach bitów, niż odpowiedni porównywany algorytm, dla pewnych rodzin redukowalnych grafów przepływu oraz większej liczby operacji dla innych rodzin grafów.

АЛГОРИТМ АНАЛИЗА РАЙОНОВ ДЛЯ ПРОБЛЕМ АНАЛИЗА СБОРА ДАННЫХ

Р е з ю м е

Предложен новый алгоритм, названный алгоритмом анализа районов, предназначенный для решения проблем анализа сбора данных для сводимых графов управления. Доказана частичная корректность алгоритма и достижение расчётов с применением этого алгоритма окончательной точки для проблем определений достигающих и переменных активных. Показано, что песимистическая временная сложность алгоритма есть $O(n^2)$ для обеих проблем. Кроме того произведено сравнение алгоритма с точки зрения временой сложности с двумя хорошо знакомыми алгоритмами решения выше упомянутых проблем. Эти сравнения были сделаны на базисе нескольких семейств самомножительных сводимых графов сбора. Представленный алгоритм сравнивался с алгоритмом Аллена-Кука для проблемы определений достигающих и с версией "роунд-робин" алгоритма итеррационного для проблемы переменных активных. В обеих случаях результаты сравнений являются непоказательными, так как они показывают, что алгоритм анализа районов требует реализации асимптотически меньшего числа операций на векторах битов, чем соответствующий сравниваемый алгоритм для некоторых семейств сводимых графов сбора и большого числа операций для других семейств графов.

THE REGION ANALYSIS ALGORITHM FOR DATA FLOW ANALYSIS PROBLEM

S u m m a r y

A new algorithm, called the region analysis algorithm, for global data flow analysis problems on reducible flow graphs has been proposed. The termination and partial correctness of the algorithm for the reaching definitions and live variables problems have been proved. It has been shown that the algorithm has the worst-case time bound of $O(n^2)$ bit vector operations for both problems. In addition it has been compared with respect to the time complexity with two well-known algorithms for two problems mentioned above. The comparisons have been carried out on some self-replicating families of reducible flow graphs. The algorithm to be presented has been compared with the Allen-Cocke algorithm for the reaching definitions problem (Allen, F.E., and Cocke, J. [1976]. A program data flow analysis procedure, Comm. ACM 19, 3, March, 137-147), and with the round-robin version of the iterative algorithm for the live variables problem (Hecht, M.S., and Ullman, J.D. [1975].A simple algorithm for global data flow analysis problems, SIAM J. Comput. 4, 4, Dec., 519--532). In both cases the results of comparisons are inconclusive, since they show that the region analysis algorithm requires asymptotically fewer bit vector operations than the respective algorithm being compared for some families of reducible flow graphs, and it requires more operations for other families of graphs.

Cena zł 60,–

WYDAWNICTWA NAUKOWE I DYDAKTYCZNE POLITECHNIKI ŚLĄSKIEJ MOŻNA NABYĆ W NASTĘPUJĄCYCH PLACÓWKACH:

44-100 Gliwice — Księgarnia nr 096, ul. Konstytucji 14 b
44-100 Gliwice — Spółdzielnia Studencka, ul. Wrocławska 4 a
40-950 Katowice — Księgarnia nr 015, ul. Żwirki i Wigury 33
40-096 Katowice — Księgarnia nr 005, ul. 3 Maja 12
41-900 Bytom — Księgarnia nr 048, Pl. Kościuszki 10
41-500 Chorzów — Księgarnia nr 063, ul. Wolności 22
41-300 Dąbrowa Górnicza — Księgarnia nr 081, ul. ZBoWiD-u 2
47-400 Racibórz — Księgarnia nr 148, ul. Odrzańska 1
44-200 Rybnik — Księgarnia nr 162, Rynek 1
41-200 Sosnowiec — Księgarnia nr 181, ul. Zwycięstwa 7
41-800 Zabrze — Księgarnia nr 230, ul. Wolności 288
00-901 Warszawa — Ośrodek Rozpowszechniania Wydawnictw Naukowych PAN — Pałac Kultury i Nauki

Wszystkie wydawnictwa naukowe i dydaktyczne zamawiać można poprzez Składnicę Księgarską w Warszawie, ul. Mazowiecka 9.