PHÂN TÍCH DÒNG ĐIỀU KHIỂN VÀ DÒNG DỮ LIỆU TRONG QUÁ TRÌNH TỐI ƯU MÃ CỦA TRÌNH BIÊN DỊCH

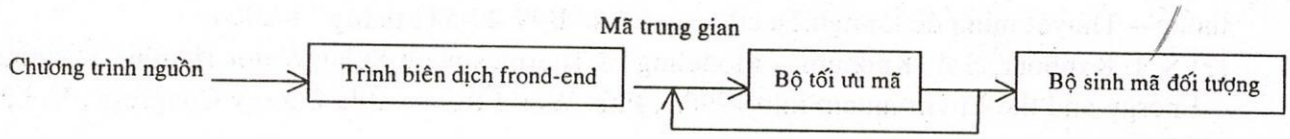
Phan Thị Tươi
Trường Đại học Bách Khoa – Đại học Quốc Gia TP.HCM
(Bài nhận ngày 06 tháng 2 năm 2002, hoàn chỉnh sửa chữa ngày 11 tháng 3 năm 2002)

TÓM TẮT: Để thực hiện việc cải tiến mã, bộ tối ưu mã phải thực hiện qua hai giai đoạn: phân tích dòng điều khiển, dòng dữ liệu và chuyển đổi mã. Trong bài báo này chúng tôi sẽ trình bày các vấn đề được xử lý trong giai đoạn phân tích cũng như trình bày các giải thuật đã được thực hiện và những kết quả nghiên cứu và triển khai.

1. GIỚI THIỆU:

1.1. Trình biên dịch tối ưu

Trình biên dịch sẽ sinh ra mã tốt nếu chương trình nguồn được viết tốt. Tuy nhiên điều này còn phụ thuộc vào trình độ người lập trình. Vì vậy để mã được tạo sinh tốt hơn người ta phải chủ động cài đặt một bộ phận cải tiến mã, được gọi là bộ tối ưu mã (code optimizer) trong trình biên dịch như ở hình H.1.1.



H 1.1 Vị trí của bộ tối ưu mã trong trình biên dịch

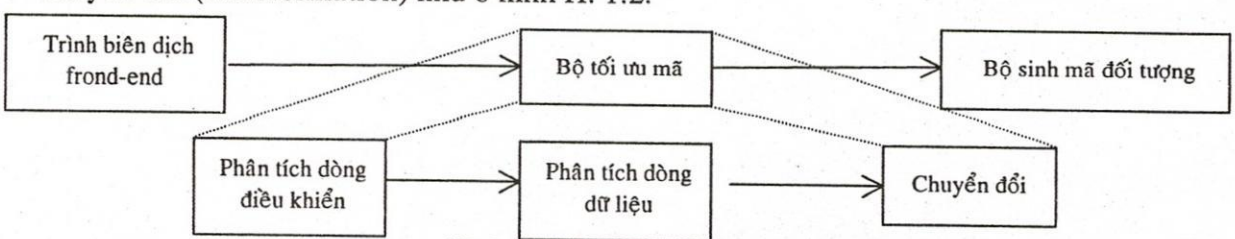
Trình biên dịch có bộ phận tối ưu mã được gọi là trình biên dịch tối ưu (optimizing compiler).

Mục tiêu của trình biên dịch tối ưu là chuyển đổi một chương trình P sang chương trình P' với cùng một hành vi nhập xuất nhưng chạy nhanh hơn hoặc chiếm ít bộ nhớ hơn (hoặc cả hai), cụ thể:

(i) Phải giảm số lệnh thực thi, (ii) Thay thế các lệnh phức tạp bằng các lệnh đơn giản, (iii) Tận dụng khả năng truy xuất cache và bộ nhớ, (iv) Tận dụng khả năng song song của phần cứng.

1.2. Bộ tối ưu mã

Bộ tối ưu mã, ở đây bài báo nói đến là bộ tối ưu mã trung gian. Mã trung gian được tối ưu qua hai giai đoạn: Giai đoạn 1 là phân tích dòng điều khiển (control flow analysis) và phân tích dòng dữ liệu (data flow analysis) sau đó ở giai đoạn 2 mã được đưa qua bộ phận chuyển đổi (transformation) như ở hình H. 1.2.



H. 1.2. Tổ chức của bộ tối ưu mã

1.3. Mã trung gian

Trong nhiều trình biên dịch, mã trung gian được sử dụng là dạng mã ba địa chỉ:

$$result = operand_1 \ op \ operand_2$$

Hiện nay chúng tôi đã xây dựng thành công phần mềm là bộ tối ưu mã trung gian [4]. Bộ tối ưu mã này đã thực hiện toàn bộ những giải thuật tối ưu như loại bỏ biểu thức con dùng chung trong một khối cơ bản, trong toàn bộ chương trình, di chuyển mã ra khỏi vòng lặp (tối ưu vòng lặp) và thay thế các câu lệnh phức tạp bằng các lệnh đơn giản trong chương trình. Để thực hiện các giải thuật chuyển đổi này, bộ tối ưu mã phải biết được cấu trúc điều khiển, dòng thông tin dữ liệu của chương trình cần tối ưu. Chính vì thế trong bài báo này tôi sẽ trình bày các vấn đề mà trong bước phân tích dòng điều khiển và phân tích dòng dữ liệu cần phải giải quyết cũng như các giải thuật tương ứng. Sau cùng bài báo sẽ trình bày một số minh họa về kết quả nghiên cứu.

2. PHÂN TÍCH DÒNG ĐIỀU KHIỂN

Trong giai đoạn này, bộ tối ưu mã xác định cấu trúc điều khiển của chương trình, cụ thể:

- mô tả các con đường thực thi có thể của chương trình
- xác định các vòng lặp
- xác định cấu trúc bên trong của một thủ tục
- xác định cấu trúc điều khiển giữa các thủ tục trong chương trình

Trước tiên bộ tối ưu mã trong thời gian này phải xác định các khối cơ bản (basic blocks) và đồ thị dòng điều khiển (control flow graph).

Giải thuật 2.1: Xác định khối cơ bản

Nhập: chuỗi phát biểu ba địa chỉ

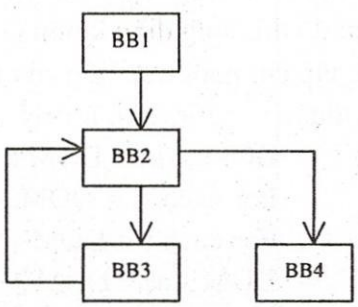
Xuất: danh sách các khối cơ bản và chuỗi phát biểu ba địa chỉ của từng khối

Phương pháp: xác định phát biểu dẫn đầu khối cơ bản đó là: điểm vào của chương trình, đích đến của phát biểu rẽ nhánh, các phát biểu ngay sau phát biểu rẽ nhánh.

Ví dụ 2.1: Xác định các khối cơ bản cho chương trình

```
(1)  read L          } BB1
(2)  n = 0           }
(3)  k = 0           }
(4)  m = 0
(5)  k = k + m       } BB2
(6)  c = k > L       }
(7)  if (c) go to (11) }
(8)  n = n + 1       } BB3
(9)  m = m + 1       }
(10) go to (5)       } BB4
(11) write n         }
```

H 2.1Xác định các khối cơ bản: BB1, BB2, BB3, BB4



H 2.2 Đồ thị dòng điều khiển của chương trình ở H2.1

2.2. Đồ thị dòng điều khiển

Đồ thị dòng điều khiển (CFG) của một chương trình là đồ thị có hướng G = (N, E), với N là tập các khối cơ bản, E là các cạnh có hướng thể hiện sự chuyển điều khiển giữa các khối cơ bản như ở hình H. 2.2.

2.3. Xác định các successor, predecessor của một khối cơ bản

Cho G = (N,E) và khối cơ bản b thuộc N:

- *Successor* của b, ký hiệu succ (b), là tập hợp các khối cơ bản, có thể đạt được từ b trên một cạnh, succ (b) = { n ∈ N | (b, n) ∈ E }
- *Predeccessor* của b, ký hiệu pred (b), là tập các khối cơ bản, mà có thể đạt được đến b trên một cạnh: pred (b) = {m ∈ N | (m, b) ∈ E }. Theo hình H 2.2. thì pred (BB2) = {BB1, BB3}, succ (BB2) = {BB3, BB4}
- Để thể hiện điểm đi vào chương trình và ra khỏi chương trình, hai nút Entry và Exit được thêm vào, là nút rỗng.

2.4. Nút chi phối (dominator)

- Một nút n ∈ N của G chi phối nút n', được ký hiệu n dom n' (hoặc n → n') nếu mọi con đường đi từ nút Entry của G đến n' đều phải đi qua n. Ký hiệu DOM (n) = { m | m → n }
- Nút n được gọi là chi phối đích thực n', ký hiệu n $\rightarrow_p$ n', nếu n → n' và n ≠ n' Ví dụ: Theo hình H.2.2. BB1 $\rightarrow_d$ BB2, BB1 $\rightarrow_p$ BB3
- Nút n được gọi là chi phối trực tiếp nút n', ký hiệu n $\rightarrow_d$ n' nếu n $\rightarrow_p$ n' và không tồn tại n''∈ N mà n $\rightarrow_p$ n'' $\rightarrow_p$ n'

Giải thuật 2.2: Xác định nút chi phối

Nhập: đồ thị dòng điều khiển

Xuất: tập các chi phối cho mỗi nút

Giải thuật: DOM (Entry) = Entry

DOM (v) = N ∀ v ∈ N - { Entry, Exit }

Changed = true

While (changed) do begin

Changed = false

For each v ∈ N - { Entry, Exit } do begin

Old DOM = DOM (v)

$$DOM(v) = \{v\} \cup \bigcap_{P \in pred(v)} DOM(p)$$

If (DOM (v) = old DOM) then changed = true; end; end;

Giải thuật 2.3 :Xác định các nút chi phối trực tiếp

Nhập: đồ thị dòng điều khiển G

Xuất: tập chi phối trực tiếp của mỗi nút

Giải thuật: for each n ∈ N - { Entry} do begin

DOM_d (n) = DOM (n) - { n}

For each s ∈ DOM_d(n) do

For each t ∈ DOM_d (n) - { s} do

DOM_d(n) = DOM_d (n) - { t}; end;

2.4.1. Hậu chi phối

Cho G = (N,E) là đồ thị dòng điều khiển và n, n'∈ N

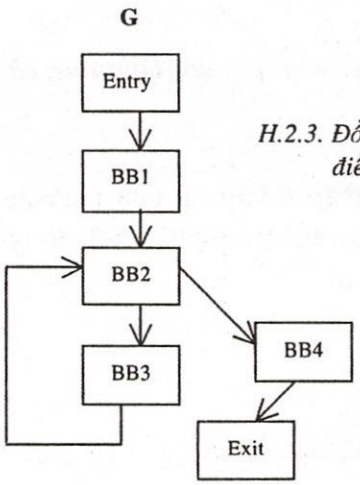
n được gọi là hậu chi phối bởi n', được ký hiệu là n ← n' nếu mọi đường từ n đến nút Entry đều có chứa n'.

2.4.2. Hậu chi phối đích thực

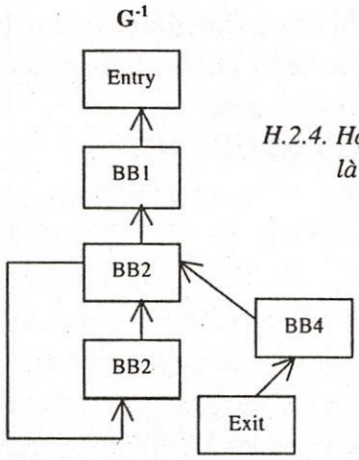
n được gọi là hậu chi phối đích thực bởi n' ký hiệu là n ← $_p$n', nếu tồn tại n ← n' và n ≠ n'

2.4.3. Hậu chi phối trực tiếp

Nút n được gọi là hậu chi phối trực tiếp bởi n', ký hiệu $n \leftarrow_d n'$ nếu tồn tại $n \leftarrow_p n'$ và không tồn tại $n'' \in N$ mà $n \leftarrow_p n'' \leftarrow_p n'$



H.2.3. Đồ thị dòng điều khiển G



H.2.4. Hậu chi phối trong G là chi phối trong G^{-1}

2.5. Vòng *lặp*

Xác định vòng lặp trong CFG cho các cú pháp như *do, for while, go to....*

Các yếu tố để xác định vòng lặp là:

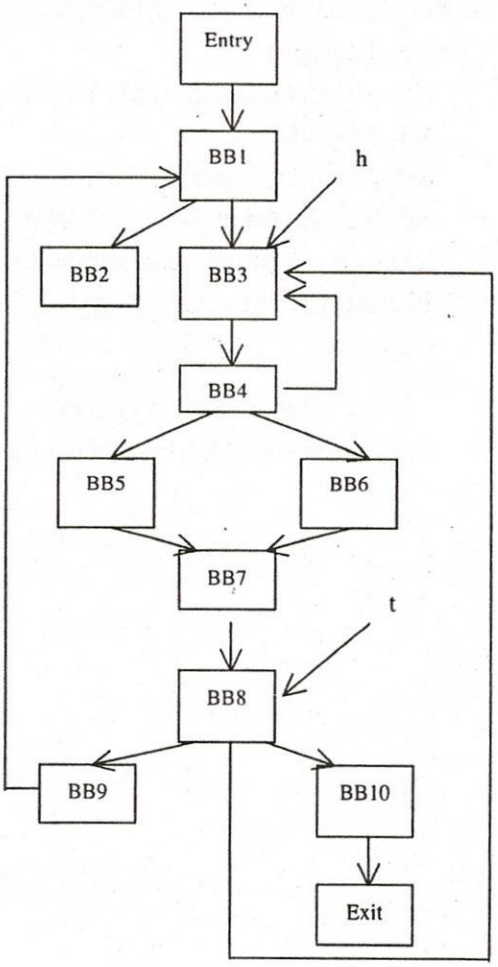
- vòng lặp phải có một điểm vào đơn, gọi là header, chi phối tất cả các nút còn lại trong vòng lặp
- phải có ít nhất một cách lặp, tức là phải có ít nhất một cạnh đi về header (gọi là tail).

Giải thuật 2.4: Xác định vòng lặp

Nhập: Đồ thị dòng G và một cạnh về từ t → h

Xuất: Vòng lặp bao gồm tất cả các nút của một vòng lặp tự nhiên t → h

Giải thuật: Stack S = Entry;
Set Loop = {h};
Insert – on- stack (t)
While (S không rỗng) do begin m = pop (S);
For each pred (m) do insert – on – stack (p); end;
Insert – on – stack (a) do begin if (a ∉ Loop) then begin
Loop = Loop ∪ a ; push – on – stack (a); end; end;



H.2.5. Ví dụ về xác định vòng lặp

2.5.1. Pre - header của vòng lặp

Trong quá trình tối ưu, ta cần phải chuyển mã ra ngoài vòng lặp. Vì vậy phải tạo pre – header cho vòng lặp, nó nằm trên header của vòng lặp để đánh dấu một câu lệnh nằm ngoài vòng lặp. Pre – header chính là chi phối trực tiếp của header vòng lặp.

2.5.2. Đồ thị dòng thu giảm (reducible flow graph)

Các cấu trúc điều khiển thường dùng như: *if – then – else, while – do, continue và break* là đồ thị dòng thu giảm

3. PHÂN TÍCH DÒNG DỮ LIỆU

Để thực hiện tối ưu chương trình, trình biên dịch cần thu thập thông tin của chương trình, truyền đến các khối cơ bản. Quá trình thu thập thông tin gọi là quá trình phân tích dòng dữ liệu. Những việc cần làm trong thời gian phân tích dòng dữ liệu là:

- Xác định dữ liệu sử dụng trong chương trình
- Sử dụng dữ liệu để quyết định khi nào cần tối ưu
- Trong một thủ tục phải xác định tính hiệu quả của câu lệnh
- Trong một chương trình thì xác định tính hiệu quả của dòng điều khiển

3.1. Điểm và đường

Trong khối cơ bản, điểm là vị trí giữa hai phát biểu, đường từ p1 đến pn là con đường đi từ điểm p1 đến điểm pn trong chương trình.

3.2. Đạt đến sự định nghĩa (reaching definition)

Nói x được định nghĩa nếu có hành vi gán trị cho x.

Một định nghĩa d của x được gọi là đạt đến điểm p nếu tồn tại con đường từ sau d đến p mà không xuất hiện một định nghĩa khác của x trên con đường đó.

3.3. Tập gen

Gen (b) là tập định nghĩa xuất hiện trong b và đạt đến điểm kết thúc của b.

3.4. Tập kill

Kill (b) là tập định nghĩa trong khối cơ bản khác b mà bị thay đổi trong b

3.5. Sự cân bằng dòng dữ liệu

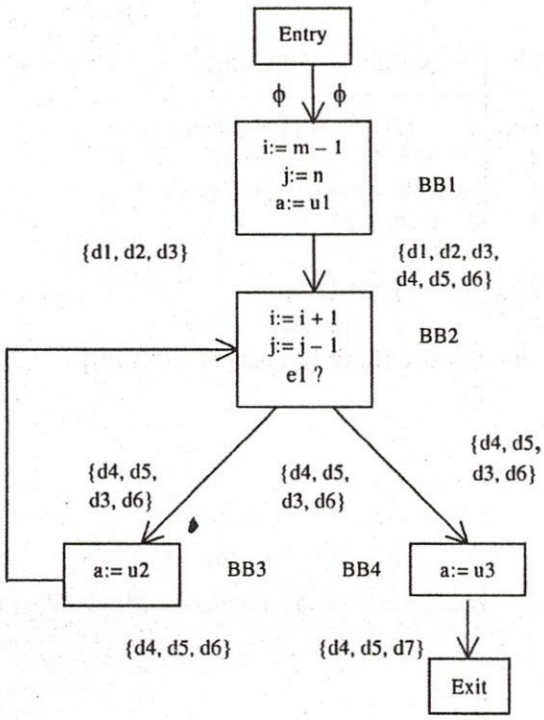
RDin (b): tập các định nghĩa đạt đến sự bắt đầu của b

RDout (b): tập các định nghĩa đạt đến sự kết thúc của b

Công thức: $RDin(b) = \bigcup_{i \in pred(b)} RDout(i)$

$$RDout(b) = Gen(b) \cup [RDin(b) - Kill(b)]$$

Giải thuật xác định các RDin và RDout ở [4]



D1: i:= m – 1
D2: j:= n
D3: a:= u1
D4: i:= I + 1
D5: j:= j – 1
D6: a:= u2
D7: a:= u3

BB	Gen (BB)	Kill (BB)
1	{d1, d2, d3}	{d4, d5, d6, d7}
2	{d4, d5}	{d1, d2}
3	{d6}	{d3, d7}
4	{d7}	{d3, d6}

RDin (b) = $\cup$ RDout (i)
$i \in$ pred (b)
RDout (b) = Gen (b) $\cup$ [RDin(b) – kill (b)]

H.2.6. Ví dụ về xác định RDin (b) và RDout (b)

3.6. Biến sống

Biến v gọi là sống tại điểm p trong chương trình nếu giá trị hiện tại của v sẽ được sử dụng trước khi v bị gán giá trị mới, hoặc trước khi chương trình kết thúc, ngược lại được gọi là biến chết. Ứng dụng quan trọng của biến sống là khi ra khỏi khối, ta không cần lưu giữ biến chết, như vậy ta có thể dùng các thanh ghi cho mục đích khác.

3.6.1 Tập Use:

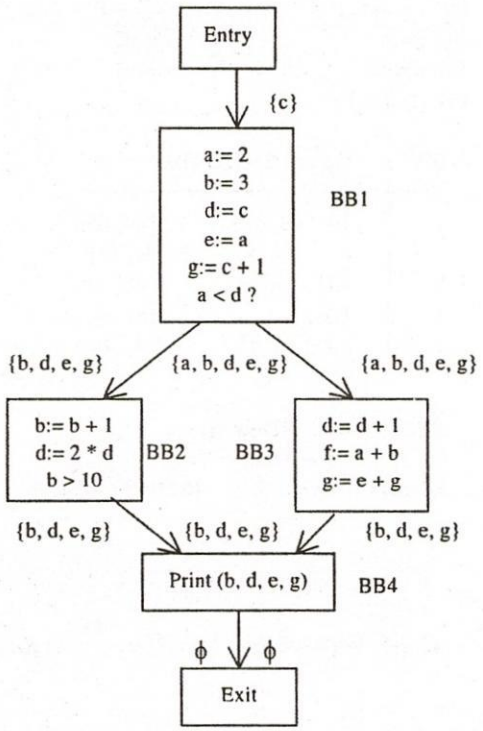
Use (b) là tập các biến được sử dụng trước khi được định nghĩa trong b.

3.6.2. Tập Def

Def (b) là tất cả các biến được định nghĩa trong b.

3.6.3. Sự cân bằng dòng dữ liệu

LVin (b): tập biến sống tại điểm bắt đầu của b
LVout (b): tập biến sống tại điểm kết thúc của b
Công thức
LVout (b) = $\cup$ LVin (i)
$i \in$ succ (b)
LVin (b) = Use (b) $\cup$ [LVout (b) – Def (b)]
Giải thuật xác định biến sống ở [4]



BB	Use(BB)	Def (BB)
1	{c}	{a, b, d, e, g}
2	{b, d}	{b, d}
3	{a, b, d, e, g}	{d, f, g}
4	{b, d, e, g}	ϕ

$$LVout(b) = \bigcup_{i \in succ(b)} LVin(i)$$

$$LVin(b) = Use(b) \cup [LVout(b) - Def(b)]$$

H.2.7. Ví dụ về xác định LVout (b) và LVin (b)

3.7. Biểu thức có sẵn (Available expression)

Một biểu thức x + y được gọi là có sẵn tại điểm p nếu mọi con đường từ nút khởi đầu đến p, hoặc là sau lần tính toán trước khi đạt đến p, không có tao vụ gán cho x và y.

Ứng dụng của biểu thức có sẵn là để tìm biểu thức con đường chung.

3.7.1. Tập Eval

Eval (b) là tập các biểu thức có sẵn được thực hiện trong b mà vẫn có sẵn tại điểm ra của b.

3.7.2.Tập Kill

Kill (b) là tập biểu thức có sẵn được thực hiện trong b mà vẫn có sẵn tại điểm ra của b.

3.7.3. Sự cân bằng dòng dữ liệu

AEin (b): tập biểu thức có sẵn tại điểm vào của b

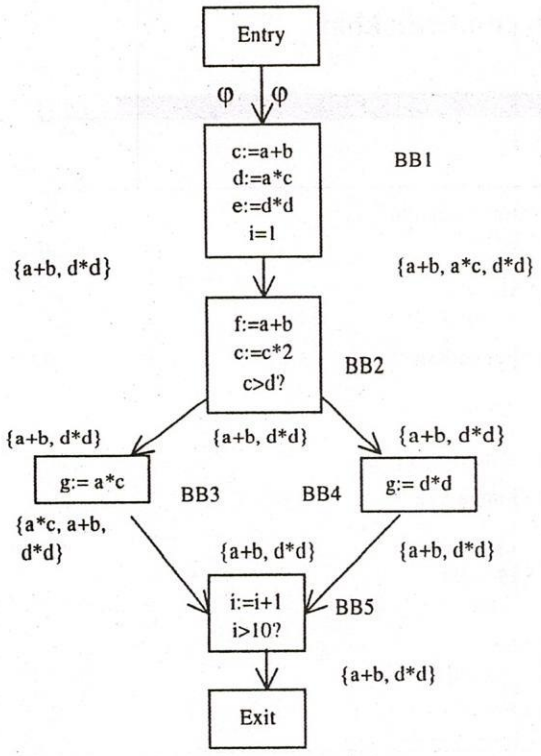
AEout(b): tập các biểu thức có sẵn đặt đến điểm kết thúc của b

Công thức:

$$AEin(b) = \bigcap_{i \in pred(b)} AEout(i)$$

$$AEout(b) = Eval \cup [AEin(b) - Kill(b)]$$

Giải thuật chi tiết tìm các biểu thức có sẵn tại điểm vào và ra của mỗi khối cơ bản được trình bày ở [4].



BB	Eval(BB)	Kill(BB)
1	{a+b, a*c, d*d}	{c*2,a*c, d*d, i+1}
2	{a+b}	{a*c}
3	{a*c}	
4	{d*d}	ϕ
5	ϕ	ϕ

U = {a+b, a*c, c*2, d*d, i+1}

$$AEin(b) = \bigcap_{i \in pred(b)} AEout(i)$$

$$AEout(b) = Eval(b) \cup [AEin(b) - Kill(b)]$$

H.2.8. Ví dụ về việc xác định biểu thức có sẵn

4. MINH HỌA CÁC KẾT QUẢ ĐẠT ĐƯỢC

Trong phần này chúng tôi xin điểm qua các giải thuật mà chúng tôi đã xây dựng để giải quyết các vấn đề trong phân tích dòng điều khiển và dòng dữ liệu.

4.1. Xác định khối cơ bản

a) Giải thuật xác định phát biểu dẫn đầu khối cơ bản
b) Giải thuật xác định khối cơ bản
c) Giải thuật xác định successor, predeccessor của một khối cơ bản

4.2. Giải thuật đồ thị dòng điều khiển (CFG)

4.3. Giải thuật các nút chi phối trong CFG

4.4. Giải thuật xác định vòng lặp

4.5. Xác định việc đạt đến sự định nghĩa

a) Giải thuật lưu giữ thông tin toàn cục
b) Giải thuật tìm tập gen cho khối cơ bản
c) Giải thuật tìm tập kill
d) Giải thuật tìm Rdin và Rdout

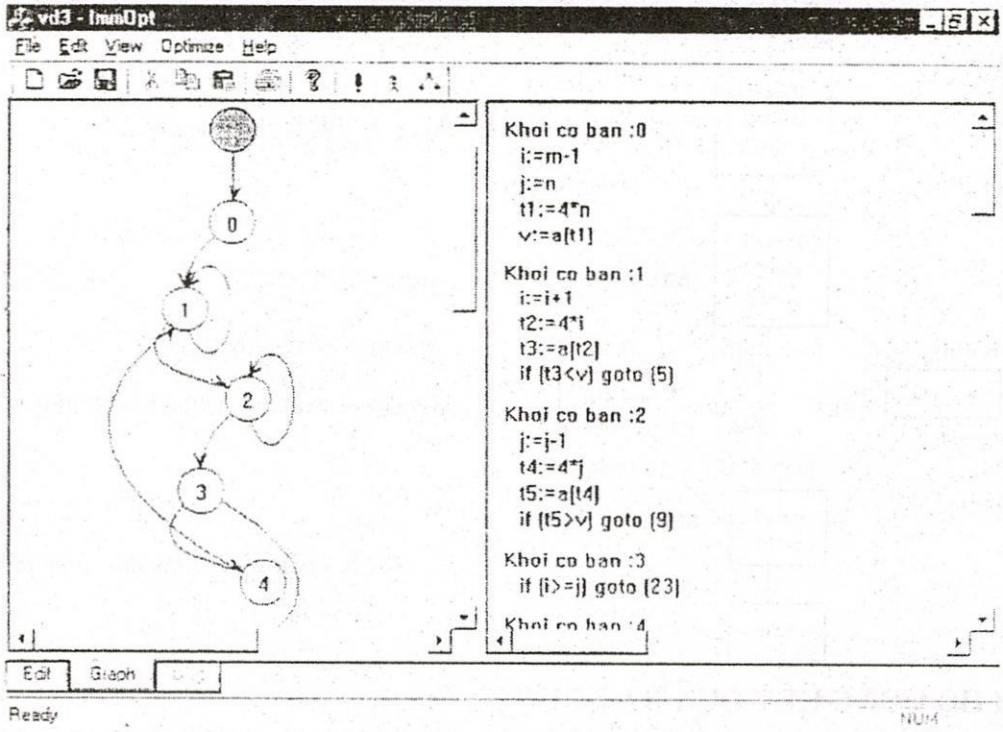
4.6. Tìm biến sống

a) Giải thuật xác định tập Use
b) Giải thuật xác định Def
c) Giải thuật xác định tập LVin, LVout

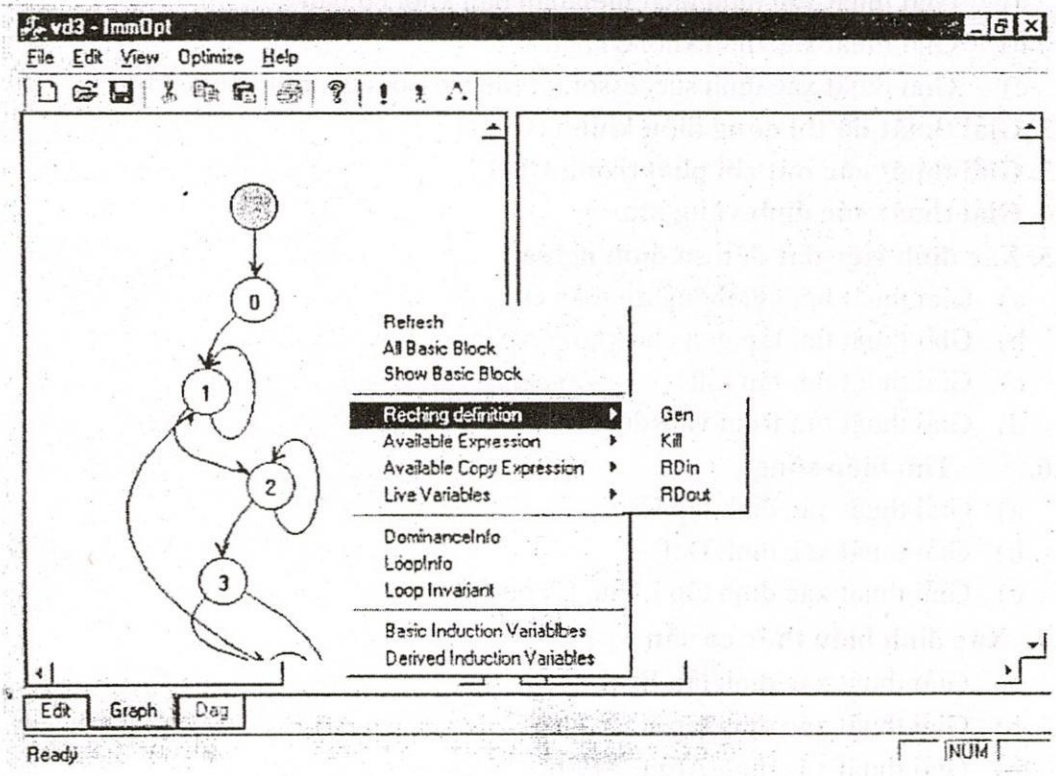
4.7. Xác định biểu thức có sẵn

a) Giải thuật xác định tập Eval
b) Giải thuật xác định tập Kill
c) Giải thuật xác định AEin, AEout

4.8. Một số minh họa về kết quả nghiên cứu triển khai

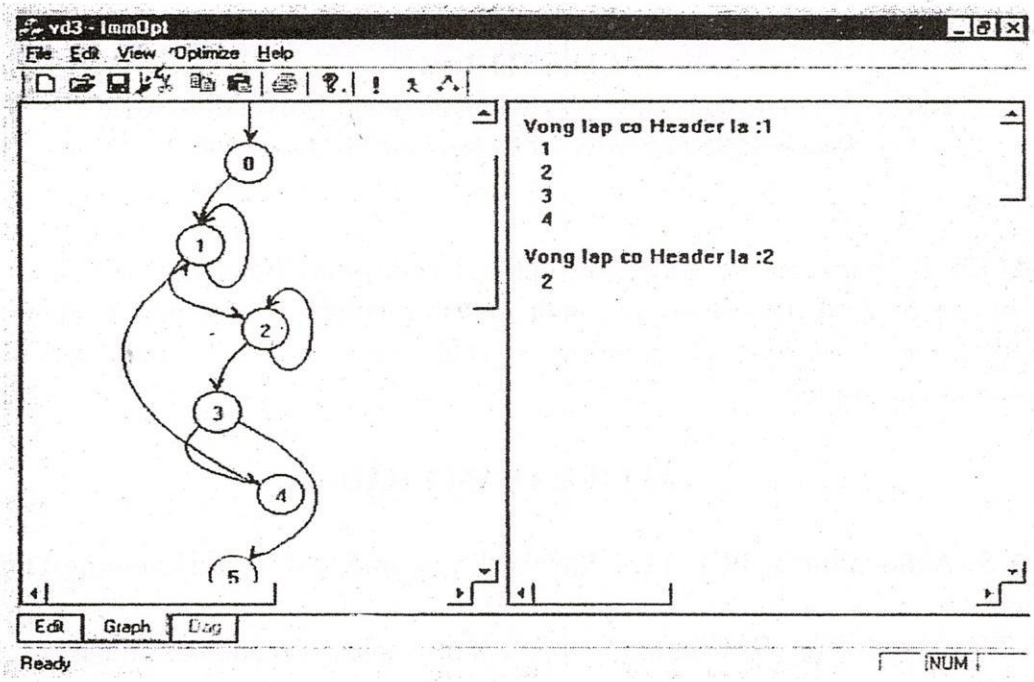


H.4.1 Khối cơ bản và dòng điều khiển



H.4.2. Chọn các chức năng

H.4.2. Chọn các chức năng



H.4.3. Xác định các vòng lặp

KẾT LUẬN

Hiện nay hầu hết các giải thuật tối ưu mã trung gian cho trình biên dịch của các sách kinh điển như [2] còn ở mức ý tưởng, phương pháp. Chúng tôi đã từ các ý tưởng đó, xây dựng chương trình. Chương trình này đã hiện thực toàn bộ các giải thuật hiện có. Trong quá trình hiện thực chương trình chúng tôi đã giải quyết một số trường hợp mà trong các tài liệu tham khảo chưa nói đến. Như xây dựng Dag cho phép toán gán trị cho biến có chỉ số, cũng như một số trong hàng loạt các vấn đề khác như vấn đề đệ quy trong quá trình giải quyết loại bỏ biểu thức con dùng chung và tối ưu vòng lặp. Những vấn đề này cũng chưa được trình bày trong các tài liệu hiện có. Qua thực nghiệm, chúng tôi còn nhận thấy rằng để kiểm tra tính hiệu quả về thời gian và bộ nhớ, chúng ta có thể xây dựng một máy tính trừu tượng để thực thi mã trung gian trước và sau khi tối ưu mã, vấn đề này sẽ được trình bày trong thời gian sắp tới. Nếu chương trình nhập tương đối lớn thì phải cần cải tiến một số giải thuật hiện có để tốc độ xử lý nhanh hơn.

ANALYSIS OF CONTROL AND DATA FLOW IN THE CODE OPTIMIZATION OF COMPILE PROCESS

Phan Thi Tuoi
University of Technology – VietNam National University-HCMC
(Received 06 February 2002, Revised 11 March 2002)

ABSTRACT: To improve the code, the optimal code must be done through 2 phases: analysis of control and data flow, and code transformation. In this paper, algorithm for identifying control structure of program and algorithm for collecting data used in optimization are presented.

TÀI LIỆU THAM KHẢO

[1] Tarek S. Abdelrahman, ECE 1724 Special Topic in Software Engineering Optimizing Compilers.
[2] Alfred V. Aho, Jeffrey D. Ullman, Compolers principles, Techniques and Tools, Printice Hall Inc. 1986.
[3] Phan Thị Tươi, Trình biên dịch, NXB Đại học Quốc gia TPHCM, 2001,
[4] Lê Hồng Sơn, GVHD: Phan Thị Tươi, Luận văn tốt nghiệp, Trường Đại học Bách Khoa – ĐHQG TPHCM, 01/2002.
[5] Steven S.Muchnick, Advanced Compiler Design Implementation, Morgan Kaufman Publishers, 1997.