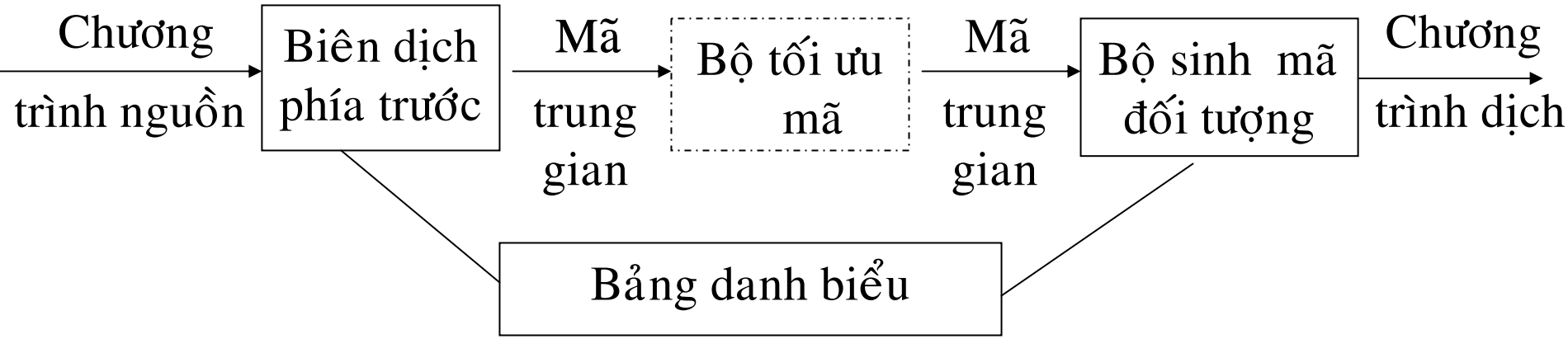
CHƯƠNG 9

SINH MÃ ĐỐI TƯỢNG



Hình 9.1. Vị trí của bộ sinh mã đối tượng

9.1. Các vấn đề thiết kế bộ sinh mã

Đầu vào của bộ sinh mã

Chương trình đích

Sự lựa chọn chỉ thị

Giả sử đối với phát biểu ba địa chỉ có dạng x := y + z với x, y, z tượng trưng cho các vị trí nhớ. Chúng ta có thể dịch sang chuỗi mã đối tượng:

MOV y, R_o /* cất y vào thanh ghi R_o */

ADD z, R_o /* cộng z vào nội dung R_o, kết quả chứa trong R_o */

MOV R_o, x /* cất nội dung R_o vào x */

Tuy nhiên việc sinh mã cho chuỗi các phát biểu sẽ dẫn đến sự dư thừa mã. Như thí dụ sau:

a := b + c; d := a + e

Chúng ta chuyển sang mã đối tượng:

(1) MOV b, R_0

(2) ADD c. R_0

(3) MOV R_0, a

(4) MOV a, R_0

(5) ADD e, R_0

(6) MOV R_0, d

Chỉ thị thứ tư là thừa.

Chất lượng mã được tạo ra, được xác định bằng tốc độ của mã và kích thước tập mã. Thí dụ:

MOV a, R_0

ADD # 1, R_0

MOV R_0, a

Cấp phát thanh ghi

Sự lựa chọn cho việc đánh giá thứ tự

9.2. Máy đích

Chúng ta sẽ dùng máy đích như là máy thanh ghi (register machine). Máy đích có mỗi từ gồm bốn byte và có n thanh ghi: R_0, R_1 ... R_{n-1}, có chỉ thị hai địa chỉ, với dạng tổng quát: op source, destination

Thí dụ một số chỉ thị:

MOV: chuyển trị của source đến destination

ADD: cộng nội dung source và destination

SUB: trừ nội dung source cho destination

Mode địa chỉ

Thí dụ:

	Mode	Dạng	Địa chỉ	Giá
1	Absolute	M	M	1
2	Register	R	R	0
3	indexed	c (R)	c + contents (R)	1

4	indirect register	*R	contents (R)	0
5	inderect indexed	*c (R)	contents (c + contents (R))	1
6	literal	# C	hằng C	1

Giá chỉ thị (instruction cost)

Giá chỉ thị được tính bằng một công giá kết hợp trong bảng mode địa chỉ nguồn và đích ở trên.

Qua các thí dụ trên chúng ta thấy muốn sinh mã tốt thì làm sao phải hạ giá của các chỉ thị.

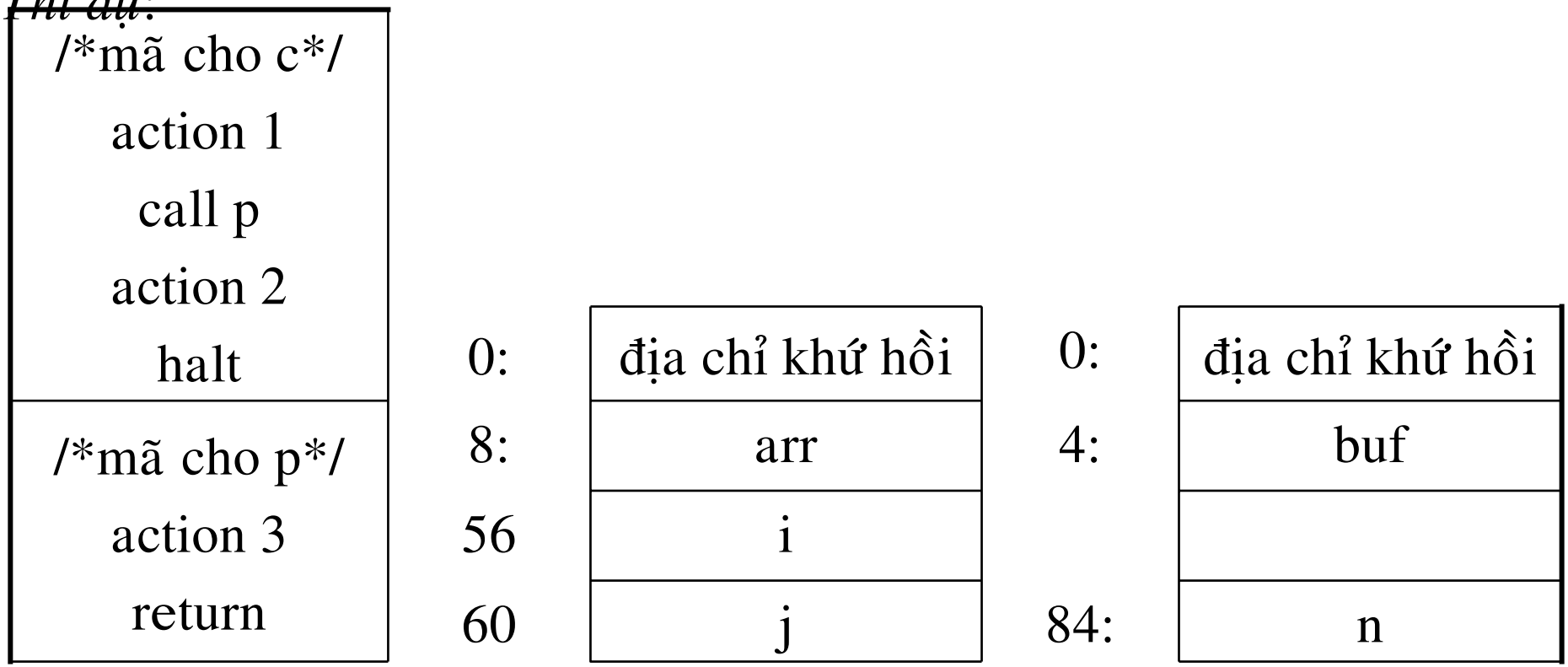
Sinh mã để quản lý các bản ghi hoạt động trong thời gian thực thi. Các mã quản lý này phải đáp ứng được hai kỹ thuật quản lý bộ nhớ tĩnh và cấp phát bộ nhớ theo cơ chế stack.

Việc cấp phát và giải tỏa vị trí nhớ cho bản ghi hoạt động là một phần trong chuỗi hành vi gọi và trở về của chương trình con.

1. call 2. return

3. halt 4. action /* tượng trưng cho các phát biểu khác */

Thí dụ:



Cấp phát tĩnh

Phát biểu call được hiện thực bằng hai mã đối tượng MOV và GOTO.

MOV # here + 20, callee.static - area

GOTO callee. code – rea

Thí dụ 9.1.

Mô phỏng 9.1. *Mã đối tượng cho chương trình con c và p*

	/* mã cho c */
100: action	
120: MOV 140, 364	/* cất địa chỉ khứ hồi 140 */
132: GOTO 200	/* gọi p */
140: action$_2$	
160: halt	
...	
	/* mã cho p */

200: action$_3$	
220: GOTO * 364	/* trở về địa chỉ được cất tại vị trí 364 */
	/* 300 - 364 cất bản ghi hoạt động của c */
300:	/* chứa địa chỉ khứ hồi */
304:	/* dữ liệu cục bộ của c */
	/* 364 - 451 chứa bản ghi hoạt động của p*/
364:	/* chứa địa chỉ khứ hồi */
368:	/* dữ liệu cục bộ của p */

Cấp phát theo cơ chế stack

Mã cho chương trình đầu tiên là mã khởi động stack, cất địa chỉ bắt đầu stack vào sp bằng chỉ thị MOV # stackstart, SP. Như vậy mã đối tượng cho chương trình con đầu tiên bao gồm:

```
MOV # stackstart, SP        /* khởi động stack */
đoạn mã cho chương trình con
HALT                        /* kết thức sự thực thi */
```

ADD # caller.recordsize, SP
MOV # here + 16, * SP /* lưu địa chỉ khứ hồi */
GOTO callee.code-area
Chuỗi trở về gồm hai chỉ thị:
GOTO *0 (SP) /* trở về chương trình gọi */
SUB # callee.recordsize, SP
Chỉ thị GOTO *0 (SP)

Thí dụ 9.2

/* mã cho s */
$action_1$
callq
$action_2$
halt
/* mã cho p */
$action_3$
return

/* mã cho q */
$action_4$
callp
$action_5$
callq
$action_6$
callq
return

Hình 9.3. Mã trung gian của chương trình ở mô phỏng 9.1

Mô phỏng 9.2. *Mã đối tượng cho mã trung gian ở (H.9.3)*

```
                        /* mã cho s */
100: MOV # 600, SP          /* khởi động stack */
108: action1
128: ADD # ssize, SP        /* chuỗi gọi bắt đầu */
136: MOV 152, * SP          /* cất địa chỉ khứ hồi */
144: GOTO 300               /* gọi q */
152: SUB # ssize, SP        /* giảm trị của SP một khoảng ssize */
160: action2
180: HALT
                        /* mã cho p */
200: action3
220: GOTO * 0(SP)           /* trở về chương trình gọi */
                            /* mã cho q */
300: action4                /* nhảy có điều kiện về 456 */
320: ADD # qsize, SP
328: MOV 344, * SP          /* cất địa chỉ khứ hồi */
336: GOTO 200               /* gọi P */
```

```
344: SUB # qsize, SP
352: action5
372: ADD # qsize, SP
380: MOV 396, * SP          /* cất địa chỉ khứ hồi */
388: GOTO 300               /* gọi q */
396: SUB # qsize, SP
304: action6
424: ADD # qsize, SP
432: MOV 440, * SP          /* cất địa chỉ khứ hồi */
440: GOTO 300               /* gọi q */
448: SUB # qsize, SP
456: GOTO *0 (SP)           /* trở về chương trình gọi */
...
600:                        /* địa chỉ bắt đầu của stack trung tâm */
```

Xác định địa chỉ cho tên danh biểu trong thời gian thực thi

Nếu chúng ta dùng cơ chế cấp phát tĩnh, vùng dữ liệu được cấp phát tại địa chỉ *static*, có phát biểu x := 0. Địa chỉ tương đối của x là 12. Vậy địa chỉ của x trong bộ nhớ là *static* + 12. Nếu *static* là 100. Địa chỉ của x trong bộ nhớ là 112. Phát biểu x := 0 được dịch sang mã đối tượng với địa chỉ tuyệt đối là: MOV # 0, 112.

Giả sử x là biến cục bộ của chương trình con hiện hành, thanh ghi R3 lưu giữ địa chỉ bắt đầu của bản ghi hoạt động thì chúng ta sẽ dịch phát biểu x := 0 sang mã trung gian.

$t_1 := 12 + R_3$

$* t_1 := 0$

Chuyển sang mã đối tượng:

MOV # 0, 12 (R_3)

Giá trị trong R_3 chỉ có thể được xác định trong thời gian thực thi.

9.3. Khối cơ bản và lưu đồ

Khối cơ bản (basic block)

t_1 := a * a;
t_2 := a * b;
t_3 := 2 * t2
t_4 := t1 + t2
t_5 := b * b;
t_6 := t4 + t5

Giải thuật 9.1. Phân chia các khối cơ bản

Nhập: các phát biểu ba địa chỉ

Xuất: danh sách các khối cơ bản với từng chuỗi các phát biểu ba địa chỉ cho từng khối.

Phương pháp:

1. Xác định tập các phát biểu dẫn đầu của các khối chúng ta dùng các quy tắc sau đây:
   i) Phát biểu đầu tiên là phát biểu dẫn đầu (leader) (từ đây ta sẽ dùng từ leader thay cho cụm từ tiếng *Việt phát biểu dẫn đầu*).

ii) Bất kỳ phát biểu nào là đích nhảy đến của phát biểu GOTO có điều kiện hoặc không điều kiện đều là leader.

iii) Bất kỳ phát biểu nào đi ngay sau phát biểu goto hoặc không điều kiện có điều kiện đều là leader.

2. Với mỗi leader thì khối cơ bản gồm có nó và tất cả các phát biểu nhưng không bao gồm một leader nào khác hay là lệnh kết thúc chương trình.

Thí dụ 9.3.

Mô phỏng 9.3. *Chương trình tích tích vectơ vô hướng*

```
begin
        prod := 0
            i := 1
        repeat
            prod := prod + a[1] * b[1];
            i := i + 1
        until i > 20
end
```

Mô phỏng 9.4. *Mã trung gian để tính tích vectơ vô hướng*

```
(1)     prod :=0
(2)     i := 1
(3)     t1 := 4 * i
(4)     t2 := a[t1]          /* tính a[i]
(5)     t3 := 4 * I
(6)     t4 := b[t3]
(7)     t5 := t2 * t4
(8)     t6 := prod + t5
(9)     prod := t6
(10)    t7 := i + 1
(11)    i := t7
(12)    if i <= 20 goto (3)
```

Sự luân chuyển trên các khối
Lưu chuyển bảo tồn cấu trúc

1. Loại bỏ các biểu thức con chung.

Thí dụ: a := b + c; b := a – d; c := b + c; d := a – d

Như vậy ta chuyển bốn phát biểu trên thành:

a := b + c; b := a – d; c := b + c; d := b

2. Loại bỏ mã chết

Giả sử x là chết, nếu xuất hiện phát biểu x := y + z trong khối cơ bản thì sẽ bị loại mà không làm thay đổi giá trị của khối.

3. Đặt tên lại biến tạm

t := b + c với t là biến tạm.

u := b + c mà u là biến tạm mới ta cũng phải thay t bằng u ở bất cứ chỗ nào xuất hiện t.

4. Hoán đổi phát biểu

t_1 := b + c

t_2 := x + y

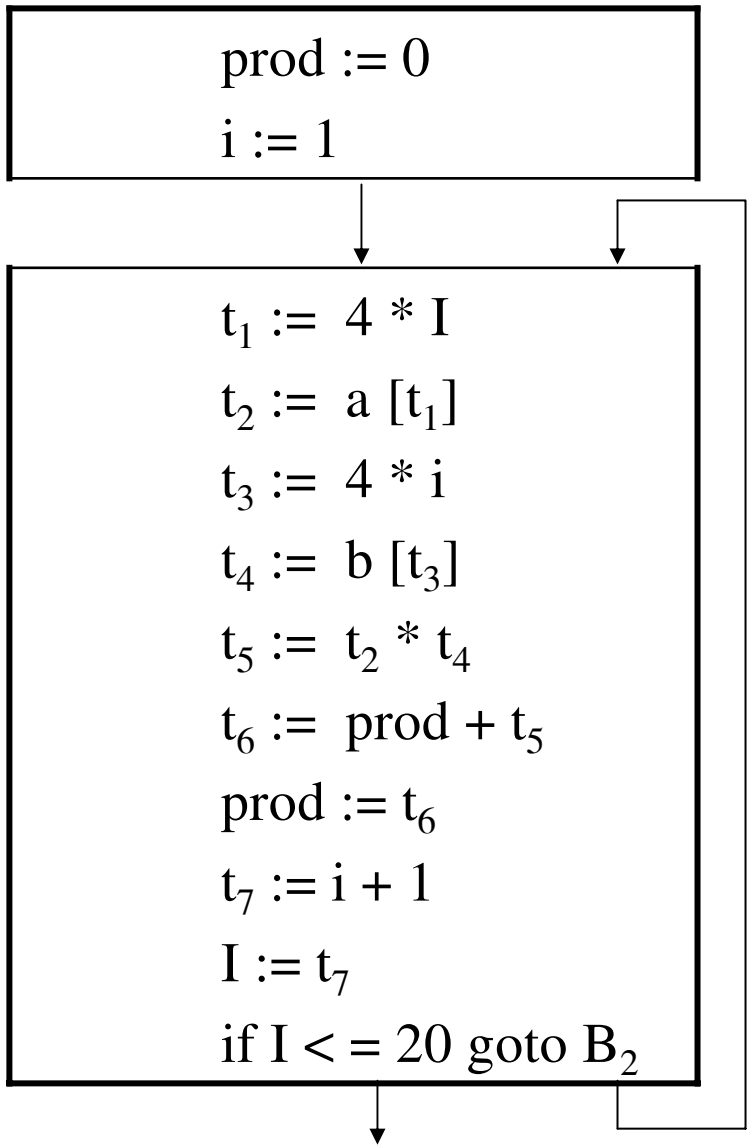
Có thể hoán đổi thứ tự hai phát biểu nếu x và y đều không phải t_1 đồng thời b và c đều không phải là t_2.

Chuyển đổi đại số

x := x + 0 hoặc x := x * 1, x := y ** 2, x := y * y

Lưu đồ (flow graph)

Đồ thị trực tiếp được gọi là lưu đồ. Các nút của lưu đồ là khối cơ bản. Một nút được gọi là khối điểm, nếu nó có chứa phát biểu đầu tiên của chương trình.



Hình 9.4. Lưu đồ của chương trình

9.4. Bộ sinh mã đơn giản

Bộ sinh mã này sẽ sinh mã đối tượng cho các mã trung gian ba địa chỉ.

Đặc tả thanh ghi và địa chỉ

1. Bộ đặc tả thanh ghi sẽ lưu giữ những gì tồn tại trong từng thanh ghi.
2. Bộ đặc tả địa chỉ sẽ lưu giữ các vị trí nhớ chứa trị của các danh biểu.

Giải thuật sinh mã đối tượng

Nhận vào chuỗi mã trung gian ba địa chỉ của một khối cơ bản. Với mỗi phát biểu ba địa chỉ có dạng x := y op z chúng ta thực thi các bước sau:

1. Gọi hàm getreg để xác định L.
2. Xác định địa chỉ đặc tả cho y.
3. Tạo chỉ thị OP z’,.
4. Nếu trị hiện tại của y và hoặc z sẽ không còn được dùng nữa.

Hàm getreg

1. Nếu y đang ở trong thanh ghi và y sẽ không được dùng nữa sau phát biểu x := y op z.
2. Ngược lại.
3. Nếu không có thanh ghi rỗng.
4. Nếu X sẽ không được dùng tiếp và cũng không thể tìm được một thanh ghi như đã nói ở bước 3.

Thí dụ 9.3. Ta có phát biểu gán d := (a – b) + (a – c) + (a – c)

Chuyển thành mã trung gian

t := a – b; u := a – c; v := t + u; d := v + u

Bảng 9.1. *Chuỗi mã đối tượng sinh ra cho thí dụ 9.3*

Mã trung gian	**Mã đối tượng**	**Giá**	**Bộ đặc tả thanh ghi**	**Bộ đặc tả địa chỉ**
t := a – b	MOV a, R_0	2	Thanh ghi rỗng, R_0 chứa t	t ở trong R_0
	SUB b, R_0	2		
u := a – c	MOV a, R_1	2	R_0 chứa t	t trong R_0
	SUB c, R_1	2	R_1 chứa u	u trong R_1
v := t + u	ADD R_1, R_0	1	R_0 chứa v	u trong R_1
			R_1 chứa u	v trong R_0
d := v + u	ADD R_1, R_0	1	R_0 chứa d	d trong R_0
	MOV R_0, d	2		d ở trong bộ nhớ

Sinh mã cho loại phát biểu khác

Bảng 9.2. *Chuỗi mã đối tượng cho phát biểu xác định chỉ số và gán.*

Phát biểu	(1) i trong thanh ghi R_1		(2) i trong bộ nhớ M_1		(3) i trên stack	
	mã	**giá**	**mã**	**giá**	**mã**	**giá**
a := b[1]	MOV b (R_i), R	2	MOV M_i , R MOV b(R), R	4	MOV S_i (A), R MOV b(R), R	4
a[1] := b	MOV b, a (R_1)	3	MOV M_i , R MOV b, a(R)	5	MOV S_i (A), R MOV b, a(R)	5

Bảng 9.3. *Mã đối tượng cho phép gán con trỏ*

Phát biểu	p trong thanh ghi R_p		p trong bộ nhớ M_p		p trong stack	+
	mã	**giá**	**mã**	**giá**	**mã**	**giá**
a := * p	MOV * R_p , a	2	MOV M_p , R MOV * R, R	3	MOV S_p(A), R MOV * R , R	4
* p := a	MOV a * R_p	2	MOV M_p , R MOV a , * R	4	MOV a , R MOV R , * S_p(A)	5

Sinh mã cho phát biểu điều kiện

if x < y goto z. Chỉ thị so sánh CMP. Thí dụ CMP x, y nếu x > y thì mã điều kiện sẽ được xác lập dương. Chỉ thị nhảy có điều kiện được thực thi nếu điều kiện được xác lập <, =, >, > =, < >, < = chúng ta dùng chỉ thị nhảy có điều kiện CJ < = z. Như vậy phát biểu điều kiện if x < y goto z được dịch sang mã máy như sau:

CMP x, y; CJ < z

9.5. Dag biểu diễn khối cơ bản

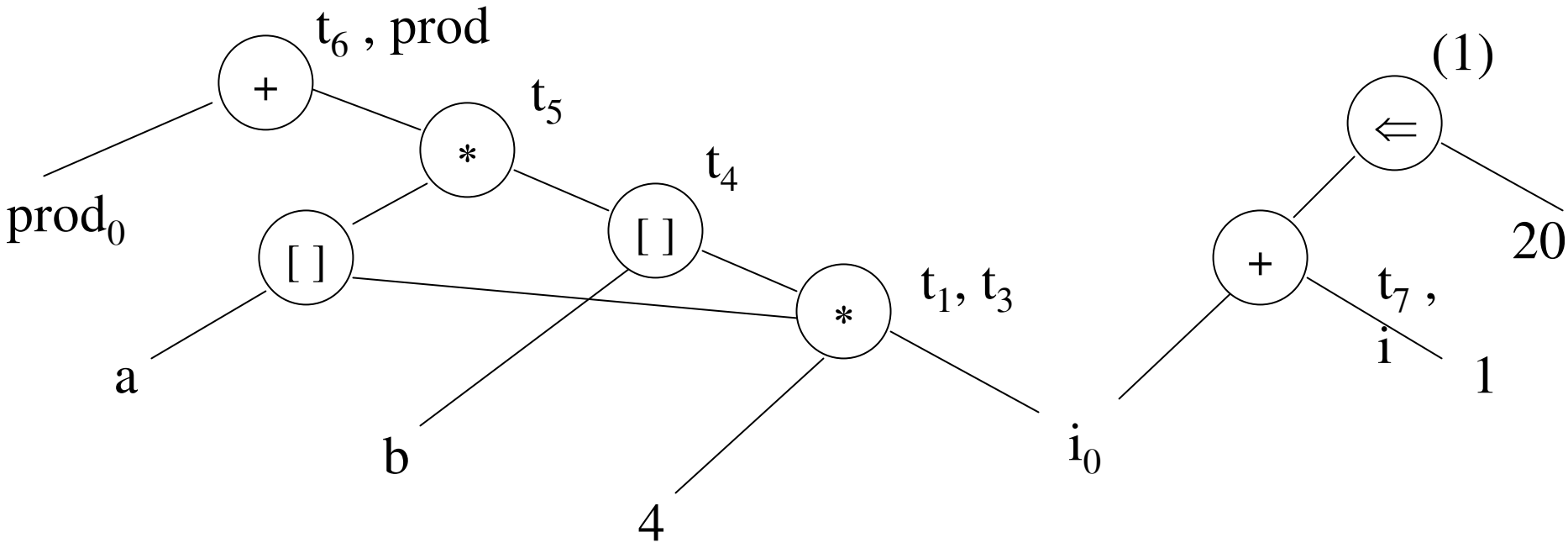
Dag là cấu trúc dữ liệu rất thích hợp để hiện thực việc chuyển đổi các khối cơ bản.

1. Các lá được đặt tên bằng các danh biểu duy nhất, hoặc là tên biến hoặc hằng số. Hầu hết các lá là r-value.
2. Các nút trung gian được đặt tên bằng ký hiệu phép toán.
3. Các nút cũng có thể là chuỗi các danh biểu cho trước. Lưu ý phải phân biệt sự khác nhau giữa lưu đồ và dag.

Thí dụ 9.4.

Mô phỏng 9.5. *Mã trung gian của khối B2*

(1)	$t_1 := 4 * i$
(2)	$t_2 := a\ [t_1]$
(3)	$t_3 := 4 * 1$
(4)	$t_4 := b\ [3]$
(5)	$t_5 := t_2 * t_4$
(6)	$t_6 := prod + t_5$
(7)	$prod := t_6$
(8)	$t_7 := i + 1$
(9)	$i := t_7$
(10)	if i < = 20 goto (1)



Hình 9.5. Dag cho khối B_2 ở mô phỏng 9.5

Xây dựng dag

Giải thuật 9.2. Xây dựng dag.

Nhập: khối cơ bản

Xuất: dag cho khối cơ bản, chứa các thông tin sau:

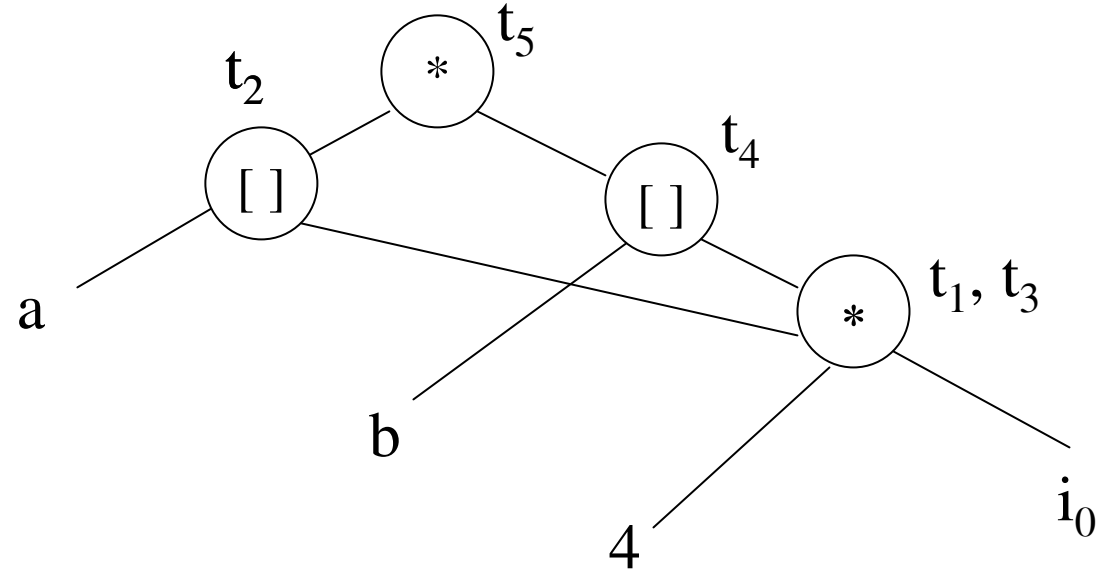
1. Tên cho từng nút.
2. Mỗi nút đều có danh sách các danh biểu gắn vào nó.

Phương pháp: Giả sử tồn tại hàm node indentifier, hàm này khi ta xây dựng dag, sẽ trả về nút mới nhất có liên quan với identifier.

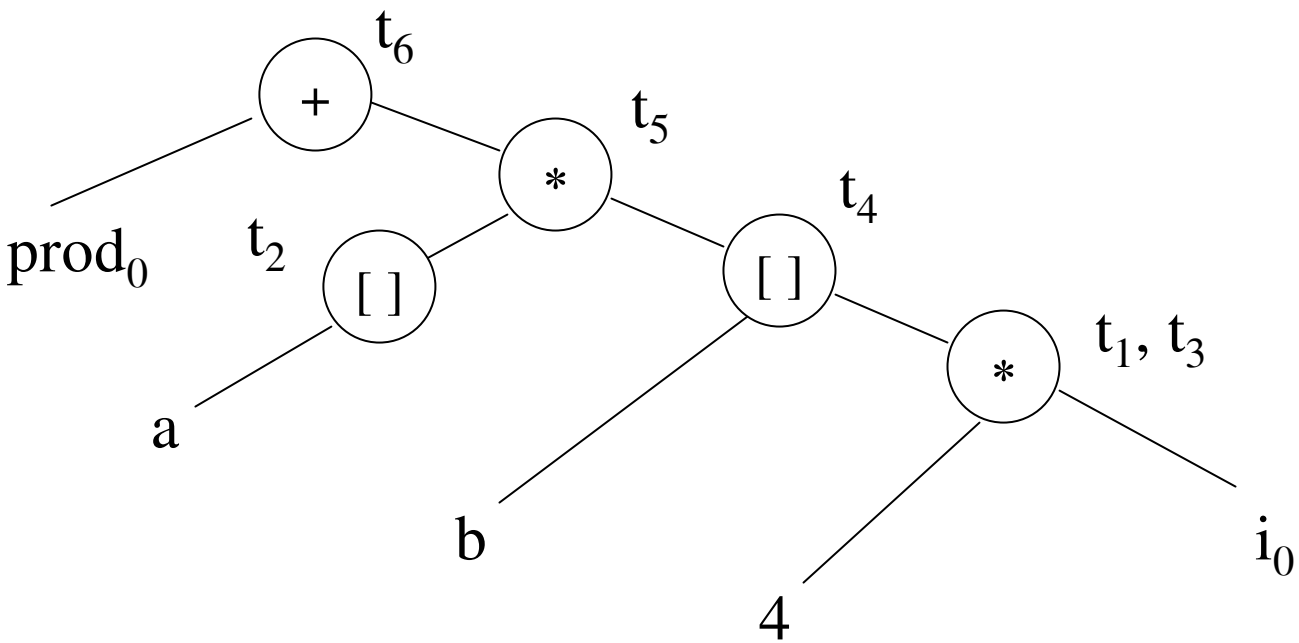
Các dạng phát biểu ba địa chỉ như sau (i) x := y op z, (ii) x := op y, (iii) x := y có trường hợp phát biểu điều kiện, thí dụ if i < = 20 goto ta coi là trường hợp (i) mà x không được định nghĩa.

1. Nếu node (y) không được định nghĩa, ta tạo lá có tên y và node (y) chính là nút đó. Trong trường hợp (i) nếu node (z) không được định nghĩa, ta tạo lá tên z và lá chính là node (z).
2. Trong trường hợp (i), xác định xem trên dag có nút nào có tên op mà con trái là node (y) và con phải là node (z). Trong trường hợp (ii) ta xác định xem có nút nào có tên op, mà nó chỉ có một con duy nhất là node (y). Trường hợp thứ (iii) thì đặt n là node (y).
3. Loại x ra khỏi danh sách biểu gắn vào nút node (x). Thêm x vào danh sách các danh biểu gắn vào nút được tìm ở bước (2) và đặt node (x) vào n.

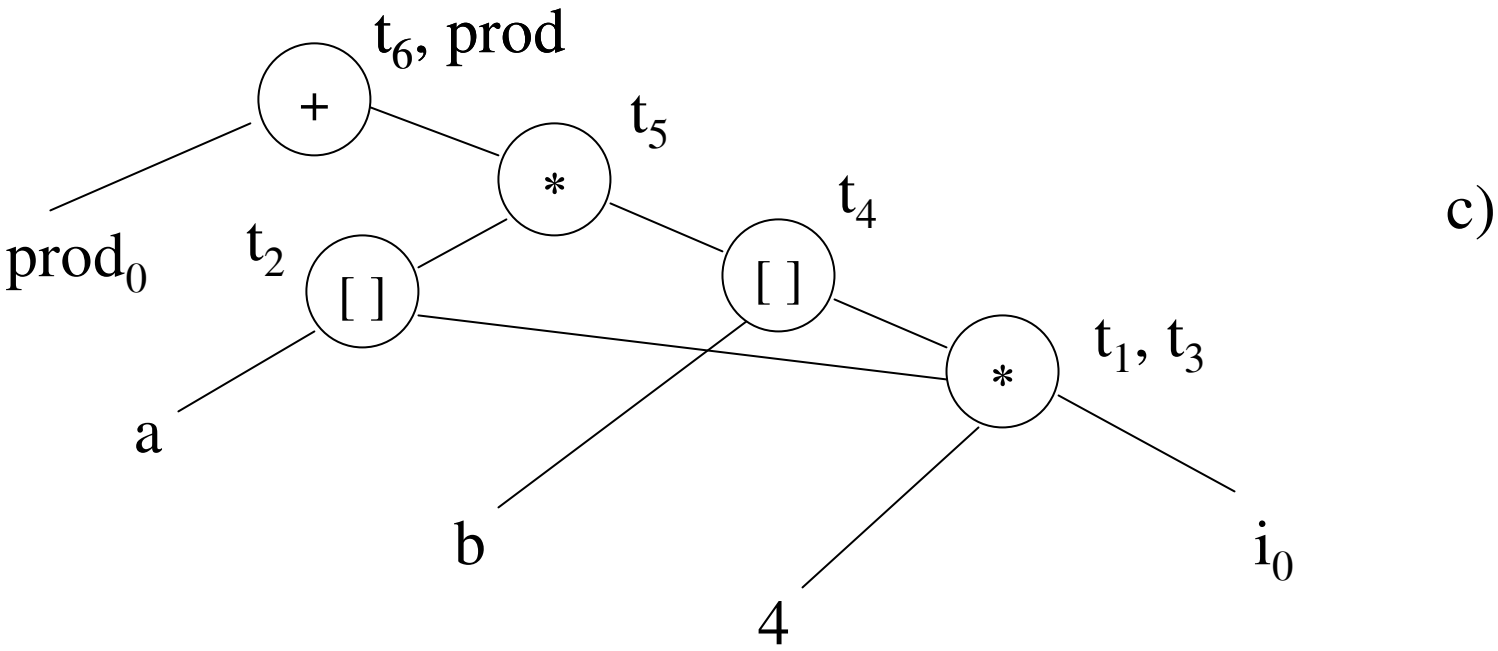
Thí dụ 9.5. Khối B_2 ở mô phỏng 9.5 của thí dụ 9.4.

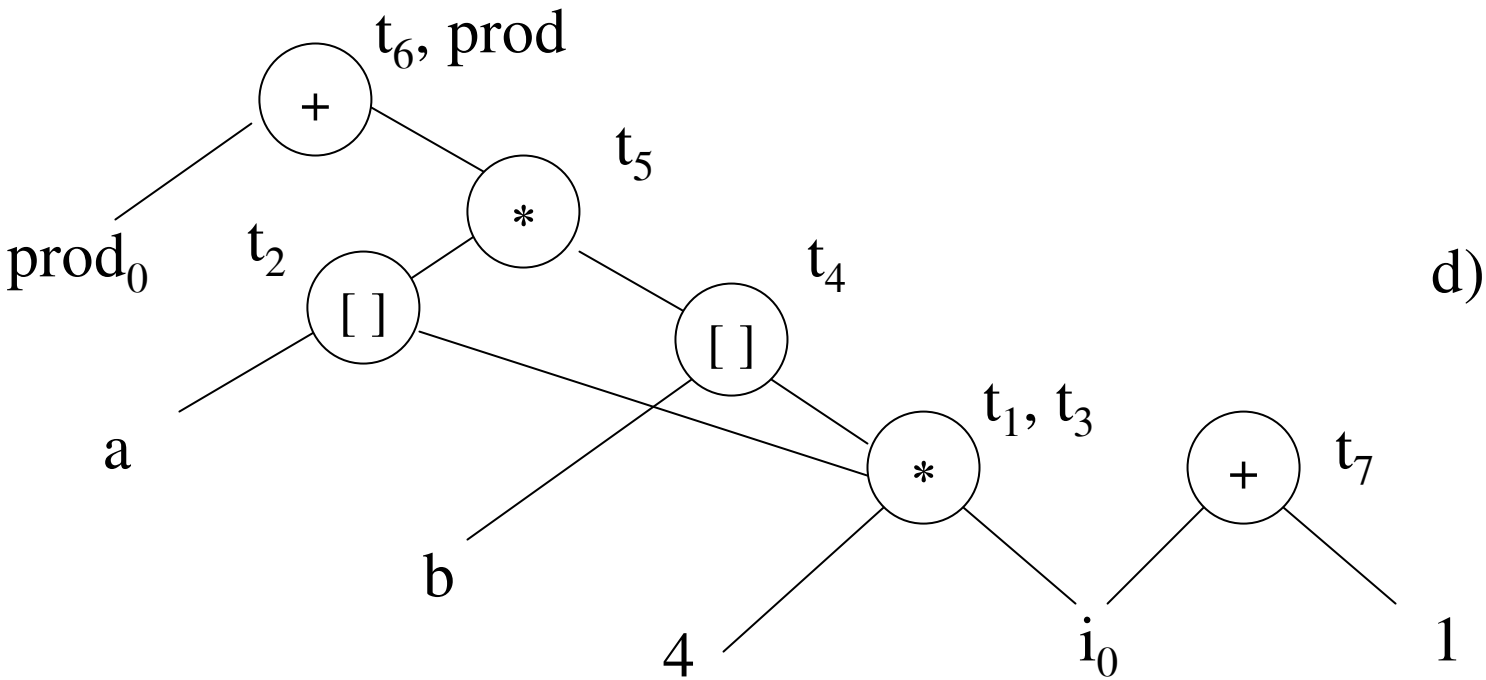


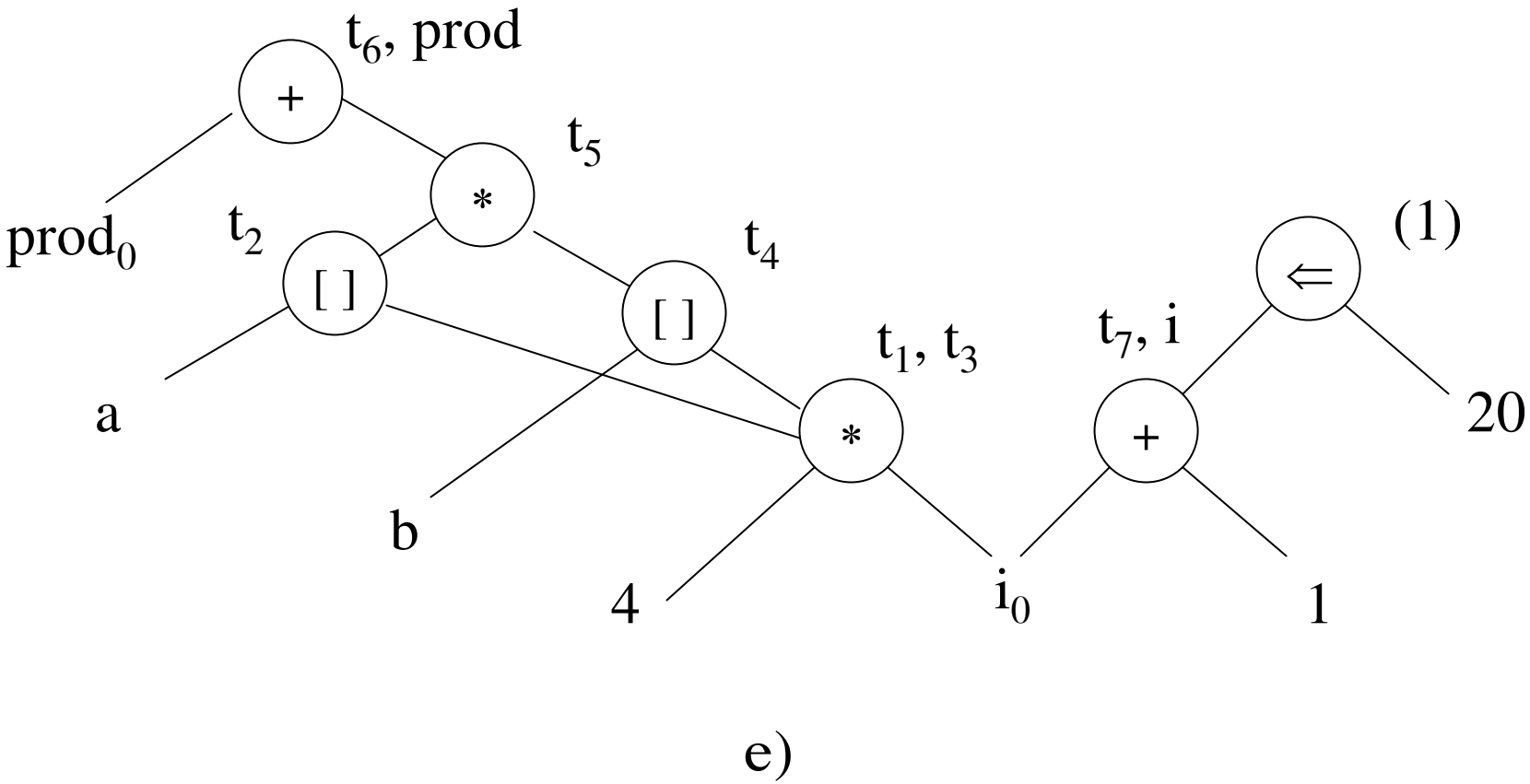
a)



b)


t_6, prod
+
t_5
*
t_4
c)
$prod_0$
t_2
[ ]
[ ]
t_1, t_3
*
a
b
i_0
4


t_6, prod
+
t_5
*
$prod_0$
t_2
t_4
d)
[ ]
[ ]
t_1, t_3
a
*
+
t_7
b
4
i_0
1



Hình 9.7. Các bước xây dựng dag của khối B_2 ở thí dụ 9.5.

Ứng dụng của dag

Ở thí dụ 9.5 chúng ta đã xây dựng dag, nó giúp cho việc tự động loại bỏ các biểu thức con giống nhau. Nó xác định những biến mà trị của chúng được sử dụng trong khối cơ bản. Dag còn giúp ta xác định những phát biểu mà trị của chúng được sử dụng ở ngoài phạm vi của khối cơ bản.

Thí dụ 9.6. Chúng ta sẽ xây dựng lại khối cơ bản từ dag của (H.9.7e).

Dãy, con trỏ và lệnh gọi chương trình con

Chúng ta khảo sát khối cơ bản sau đây:

```
x := a[i]
a[j] := y
z := a[i]
```

Giải thuật 9.2 thì a[i] sẽ trở thành biểu thức chung. Từ dag chúng ta tạo lại khối cơ bản của nó sẽ tối ưu và có dạng:

```
x := a[i]
z := x
a[j] := y
```

Nhưng khối (9.1) và (9.2) sẽ tính trị z khác nhau nếu j = I và y ≠ a[i]. Đối với con trỏ cũng xảy ra vấn đề khi ta có phát biểu gán * p := w. Nếu ta không biết p sẽ chỉ đến đối tượng nào, thì phải loại tất cả các nút có dạng trên.

Việc gọi chương trình con sẽ *giết* tất cả các nút bởi vì ta chưa biết gì về chương trình bị gọi, nên ta buộc phải giả sử rằng bất cứ biến nào cũng có thể bị thay đổi trị do hiệu ứng lề.

9.6. Tạo mã đối tượng từ dag

Sắp xếp lại thứ tự

$t_1 := a + c$

$t_2 := c + d$

$t_3 := e - t_2$

$t_4 := t_1 - t_3$

Ta có thể sắp xếp lại chuỗi mã trung gian sao cho việc tính toán t1 chỉ xảy ra ngay trước t4.

t2 := c + d t1 := a + b

t3 := e – t2 t4 := t1 – t3

Mô phỏng 9.6. *Mã đối tượng cho chuỗi phát biểu ở (H.9.8)*

```
MOV a, R0
ADD b, R0
MOV c, R1
ADD d, R1
MOV R0 , t1
MOV e, R0
SUB R1, R0
MOV t1, R1
SUB R0 , R1
MOV R1 , t4
```

Mô phỏng 9.7. *Chuỗi mã sau khi đã sắp xếp lại mã trung gian*

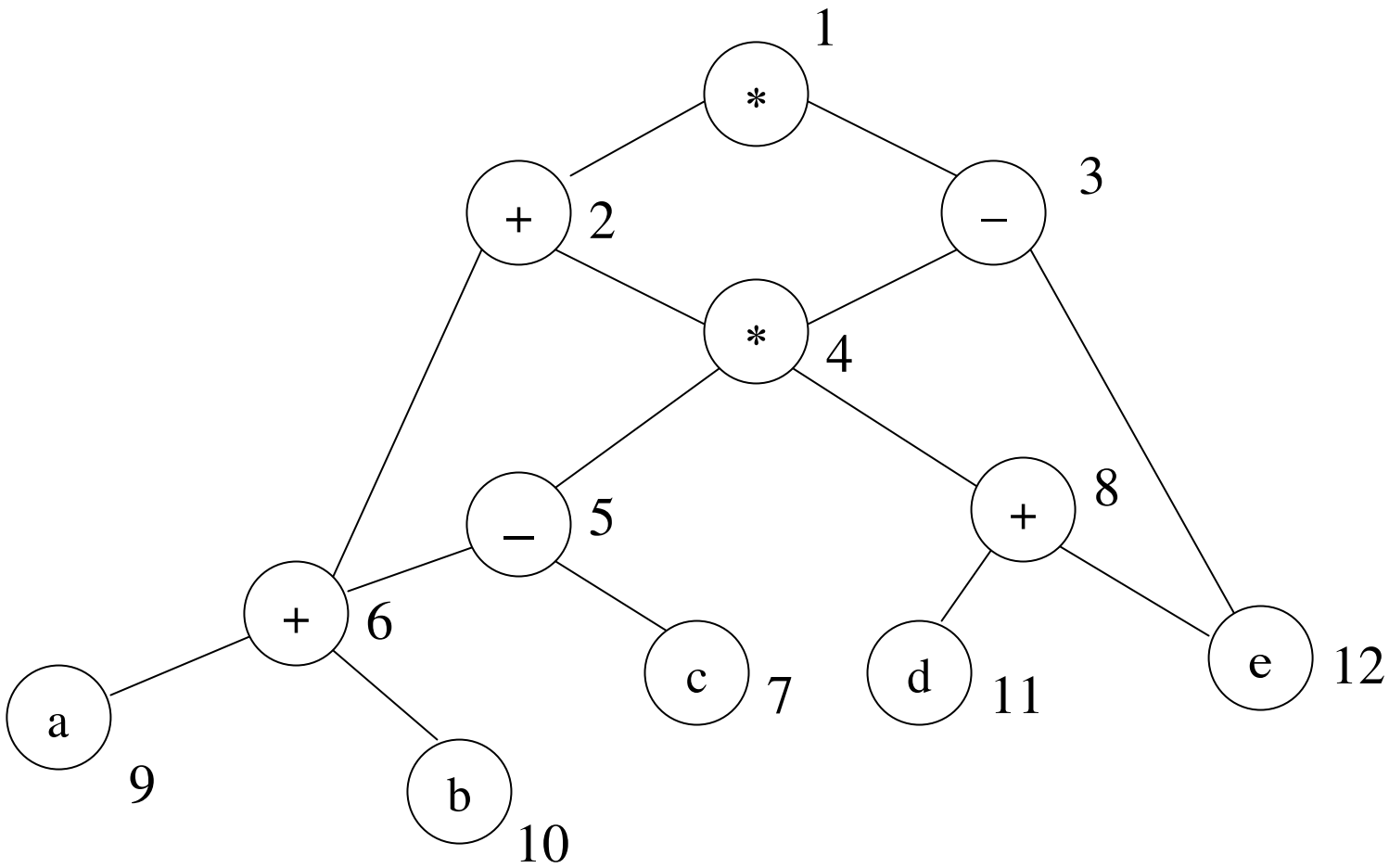
```
MOV c, R0
ADD d, R0
MOV e, R1
SUB R0 , R1
MOV a, R0
ADD b, R0
SUB R1 , R0
MOV R0 , t4
```

Heuristics dùng để sắp xếp dag

Mô phỏng 9.8. *Giải thuật sắp xếp các nút của dag*

(1) **While** nếu còn các nút trung gian chưa được liệt kê ra
do begin
(2) Chọn nút chưa liệt kê n; tất cả các nút cha mẹ của nó đã liệt kê
(3) liệt kê n;
(4) **While** con m ở tận cùng bên trái của n, có các cha mẹ đã liệt kê và nó không phải là nút lá **do begin**
(5) liệt kê m;
(6) n := m;
end;
end

Thí dụ 9.7. Giải thuật ở mô phỏng 9.8 để tạo sự sắp xếp của các mã trung gian ở (H.9.9).



Hình 9.9. Dag cho thí dụ

$t_8 := d + e$

$t_6 := a + b$

$t_5 := t_6 - c$

$t_4 := t_5 * t_8$

$t_3 := t_4 - e$

$t_2 := t_6 + t_4$

$t_1 := t_2 * t_3$

Sắp xếp tối ưu cho cây

Giải thuật có hai phần: phần đầu đánh tên cho các nút của cây, phần thứ hai của giải thuật miêu tả lộ trình trên cây. Mã đối tượng sẽ sinh ra trong quá trình thực hiện lộ trình trên cây.

- **Giải thuật xác định nhãn của nút trên cây**

Mô phỏng 9.9. *Giải thuật tính tên của nút*

$$\text{label}(n) = \begin{cases} \max(l_1, l_2) & \text{nếu } l_1 \neq l_2 \\ l_1 + 1 & \text{nếu } l_1 = l_2 \end{cases}$$

(1) **if** n là lá **then**

(2) **if** n là con tận cùng bên trái của nút cha của nó

then

(3) label (n) := 1

(4) **else** label (n) := 0

else begin /* n là nút trung gian */

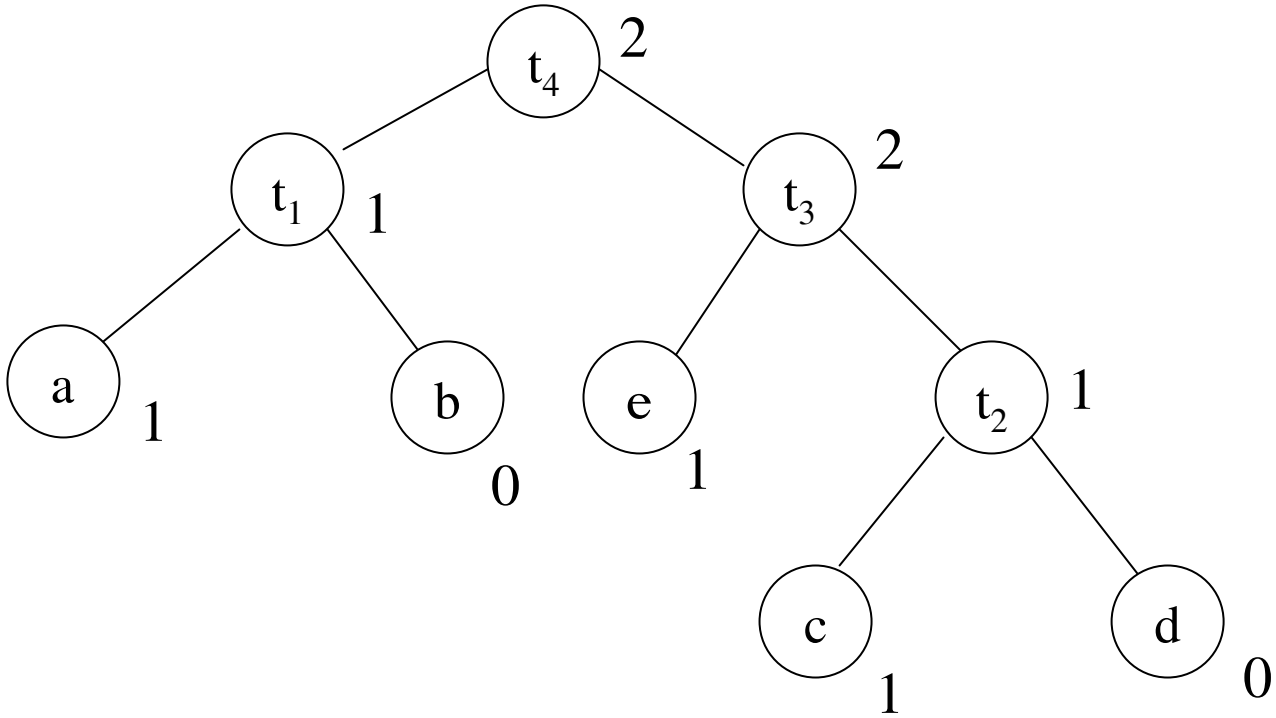
(5) giả sử $n_1, n_2, ..., n_k$ là con của nút n, được sắp theo thứ tự của tên, sao cho

$$\text{label}(n_1) > \text{label}(n_2) \geq ... \geq \text{label}(n_k)$$

(6) label (n) := $\max_{1 < i < k}$ (label $(n_i) + i - 1$)

end

Thí dụ 9.8. Chúng ta xét cây ở (H.9.8)



Hình 9.10. Cây được xác định tên

Sinh mã đối tượng từ cây có tên

Mô phỏng 9.10. *Giải thuật của thủ tục gencode*

procedure gencode (n);

begin

/* trường hợp 0 */

if n là lá bên trái biểu thị cho toán hạng **name**

and n là con tận cùng bên trái của nút cha của nó

then print 'MOV' || name || ',' || top (rstack)

else if n là nút trung gian với toán tử là op, con bên trái là n_1 và con bên phải là n_2 **then**

/* trường hợp thứ nhất */

if label (n_2) = 0 **then begin**

đặt name là toán hạng được biểu thị bằng n_2. **gencode** (n_1);

print op || name || ',' || top (rstack)

end

```
/* trường hợp thứ hai */
    else if 1 ≤ label (n1) < label (n2) and
                    label (n1) < r then begin
        swap (rstack);
        gencode (n2);
        R := pop (rstack);        /* n2 đã được tính, nằm trong R */
        gencode (n1);
        print op || R || ',' || top (rstack);
        push (rstack, R);
        swap (rstack)
    end
/* trường hợp thứ ba */
    else if 1 ≤ label (n2) ≤ label (n1) and label (n2) < r then begin
        gencode (n1);
        R := pop (rstack); /* n1 đã được tính, nằm trong thanh ghi
R*/
```

```
            gencode (n2);
            print op || top (rstack) || ‘,’ || R;
            push (rstack, R);
        end
    /* trường hợp thứ tư, cả hai tên ≥ r, r là số lượng tối đa của thanh ghi */
        else begin
            gencode (n2);
            T := pop (rstack);
            print ‘MOV’ || top (rstack) || ‘,’ || T;
            gencode (n1)
            push (tstack, T);
            print op || T || ‘,’ || top (rstack)
            end
        end;
```

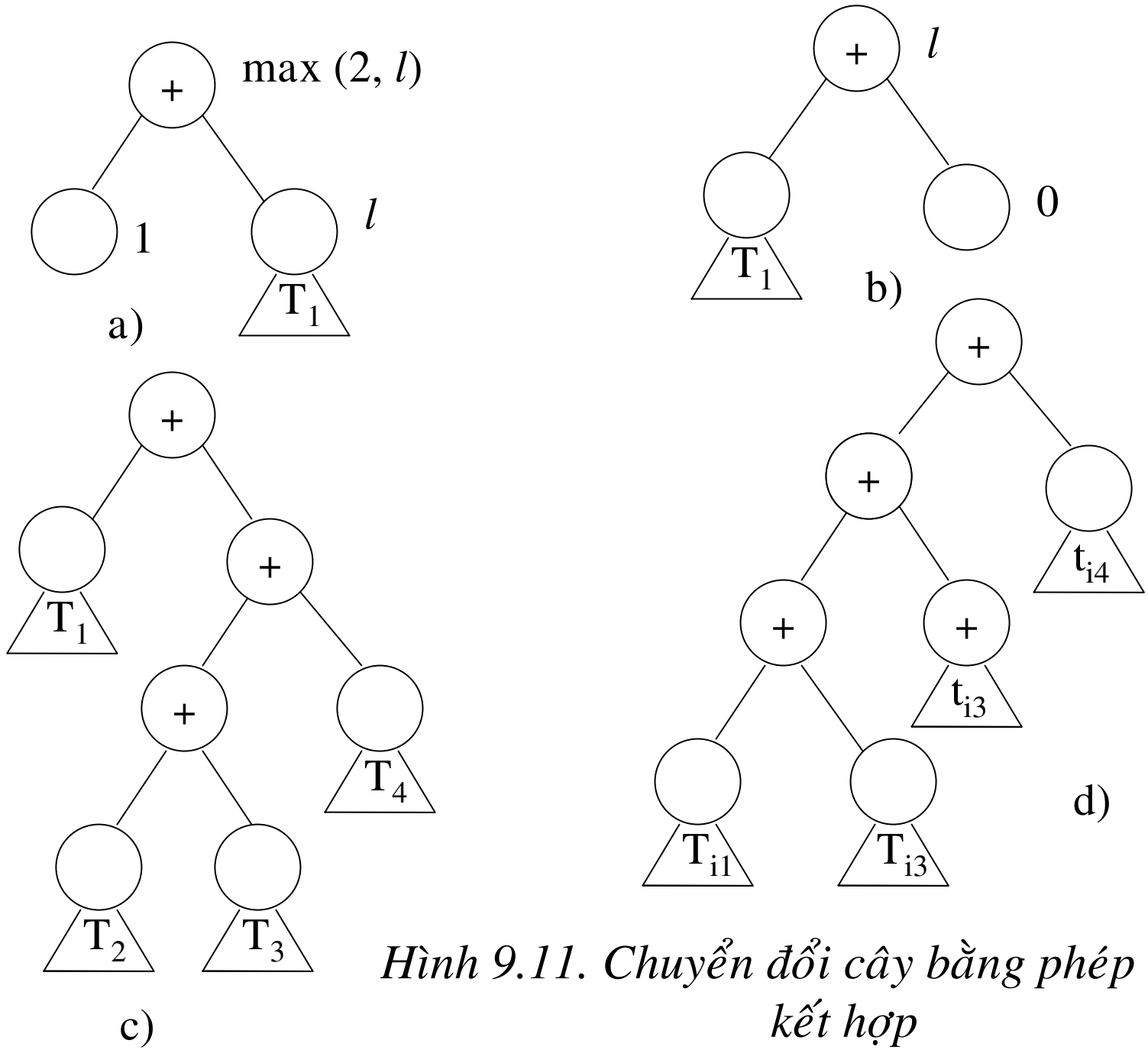
Mô phỏng 9.11. *Chuỗi các lệnh gọi thủ tục gencode và các lệnh print của các trường hợp*

gencode (t_4)	[R_1R_0]	/* trường hợp 2 */
gencode (t_3)	[R_0R_1]	/* trường hợp 3 */
gencode (e)	[R_0R_1]	/* trường hợp 0 */
print MOV e, R_1		
gencode (t_2)	[R_0)	/* trường hợp 1 */
gencode (c)	[R_0]	/* trường hợp 0 */
print MOV c, R_0		
print ADD d, R_0		
print SUB R_0 , R_1		
gencode (t_1)	[R_0]	/* trường hợp 1 */
gencode (a)	[R_0]	/* trường hợp 0 */
print MOV a, R_0		
print ADD b, R_0		
print SUB R_1 , R_0		

1. Tác vụ (toán tử phép toán) cho mỗi nút trung gian.
2. Cất mỗi nút lá là nút con tận cùng bên trái vào thanh ghi.
3. Lưu giữ cho từng nút cả hai con mà chúng có tên bằng hoặc nhiều hơn r.

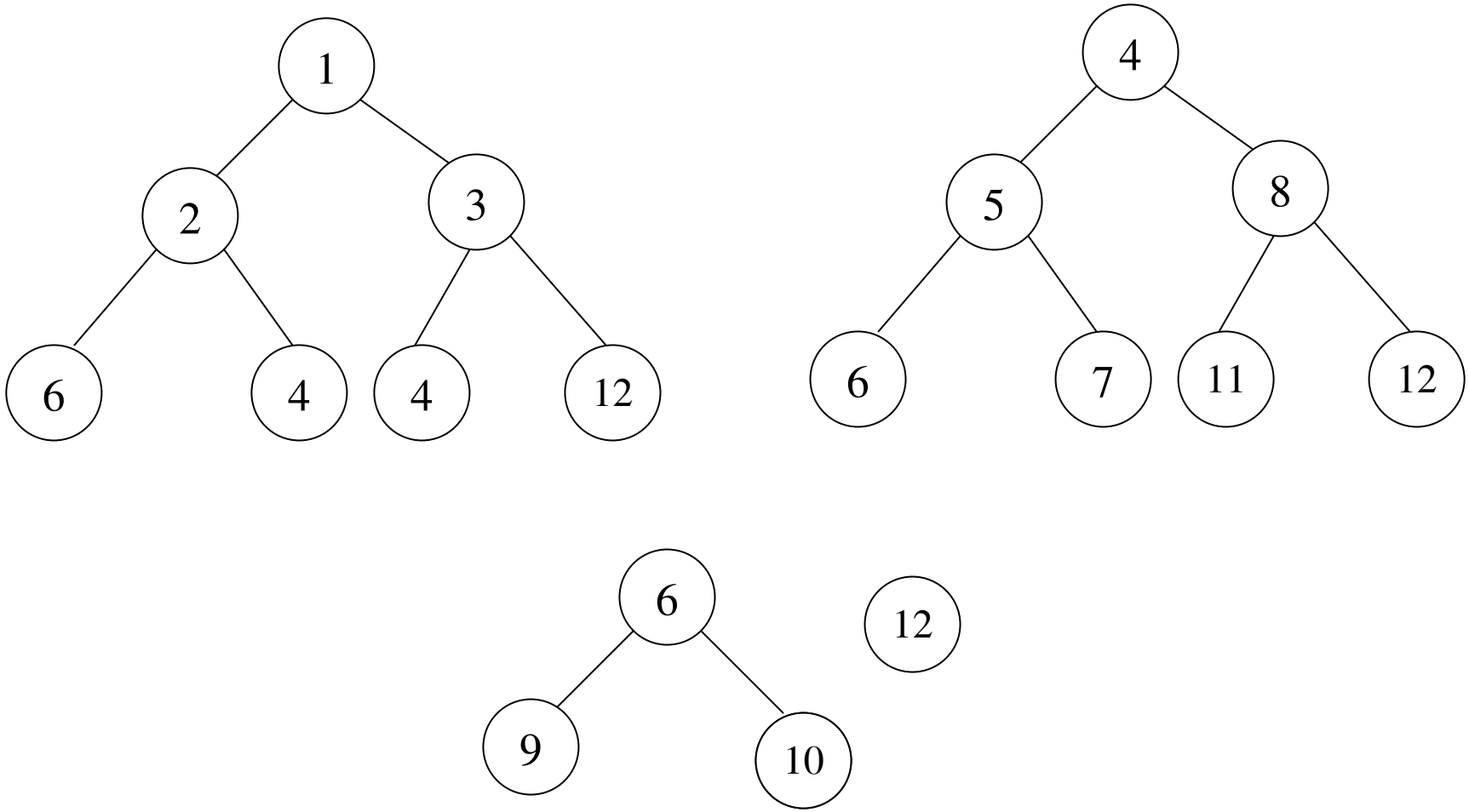
Các tác vụ với nhiều thanh ghi

Các tính chất đại số



Hình 9.11. Chuyển đổi cây bằng phép hoán vị, kết hợp

Các biểu thức chung



Hình 9.12. Phân chia thành các cây con