Bài 11
Sinh mã trung gian

ONE LOVE. ONE FUTURE.

Nội dung

- Khái niệm về mã trung gian
- Mã ba địa chỉ
- Sinh mã cho lệnh gán
- Sinh mã cho các biểu thức logic
- Sinh mã cho các cấu trúc lập trình

Lợi ích của mã trung gian

- Mã trung gian độc lập với máy
- Việc sinh mã đích từ mã trung gian sẽ dễ dàng hơn sinh từ mã nguồn
- Dễ dàng tối ưu mã với mã trung gian

Mã trung gian

- Một chương trình với mã nguồn được chuyển sang chương trình tương đương trong ngôn ngữ trung gian bằng bộ sinh mã trung gian.
- Ngôn ngữ trung gian được người thiết kế trình biên dịch quyết định, có thể là:
  - Cây cú pháp: Cây phân tích cú pháp thu gọn do loại bỏ một số nút trong. Thể hiện được các cấu trúc cú pháp, nhưng rất phức tạp
  - Ký pháp Ba Lan sau (hậu tố): Chủ yếu dùng cho sinh mã biểu thức. Dạng hậu tố rất phù hợp khi phân phối bộ nhớ trên stack
  - Mã 3 địa chỉ: là phương pháp phổ biến nhất

Mã trung gian

- Được sản sinh dưới dạng một chương trình cho một máy trừu tượng, không cần biết kiến trúc của máy.
- Mã trung gian thường dùng : mã ba địa chỉ, tương tự mã assembly.
- Chương trình là một dãy các lệnh. Mỗi lệnh gồm tối đa 3 định danh.
- Tồn tại **nhiều nhất một toán tử** ở vế phải cộng thêm một toán tử gán
- x,y,z là các địa chỉ , tức là tên, hằng hay các tên trung gian do trình biên dịch sinh ra
  - Tên trung gian phải được sinh để thực hiện các phép toán trung gian
  - Các địa chỉ được thực hiện thường là con trỏ tới lối vào của nó trong bảng ký hiệu

Ví dụ: mã trung gian của lệnh gán a :=x + y * z

t1 := y*z

t2 := x+t1

a :=t2

Việc sinh mã trung gian được thực hiện trực tiếp từ định nghĩa tựa cú pháp thông qua tính toán thuộc tính code

Tập mã lệnh ba địa chỉ điển hình

- Mã 3 địa chỉ tương tự mã Assembly: lệnh có thể có nhãn, có những lệnh chuyển điều khiển cho các cấu trúc lập trình.

  1. *Lệnh gán* x := y *op* z.
  2. *Lệnh gán với phép toán 1 ngôi* : x := *op* y.
  3. *Lệnh sao chép*: x := y.
  4. *Lệnh nhảy không điều kiện*: *goto L, L là nhãn của một lệnh*
  5. *Lệnh nhảy có điều kiện x relop y goto L.*

Tập mã lệnh ba địa chỉ điển hình

6. *Lời gọi thủ tục param x và call p,n để gọi thủ tục p với n tham số . Return y là giá trị thủ tục trả về*

   param x_1
   param x_2
   . . .
   param x_n
   Call p,n

7. *Lệnh gán có chỉ số x:=y[i] hay x[i]:=y*

Sinh mã trực tiếp từ định nghĩa tựa cú pháp

- Thuộc tính tổng hợp S.code biểu diễn mã ba địa chỉ của lệnh
- Các tên trung gian được sinh ra để lưu kết quả tính toán trung gian
- Các biểu thức được liên hệ với hai thuộc tính tổng hợp
  - E.place chứa địa chỉ chứa giá trị của E
  - E.code mã ba địa chỉ để đánh giá E
- Hàm newtemp sinh ra các tên trung gian: t1, t2,. .
- Hàm gen sinh mã ba địa chỉ
- Trong thực tế, code được gửi vào file thay cho thuộc tính code

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

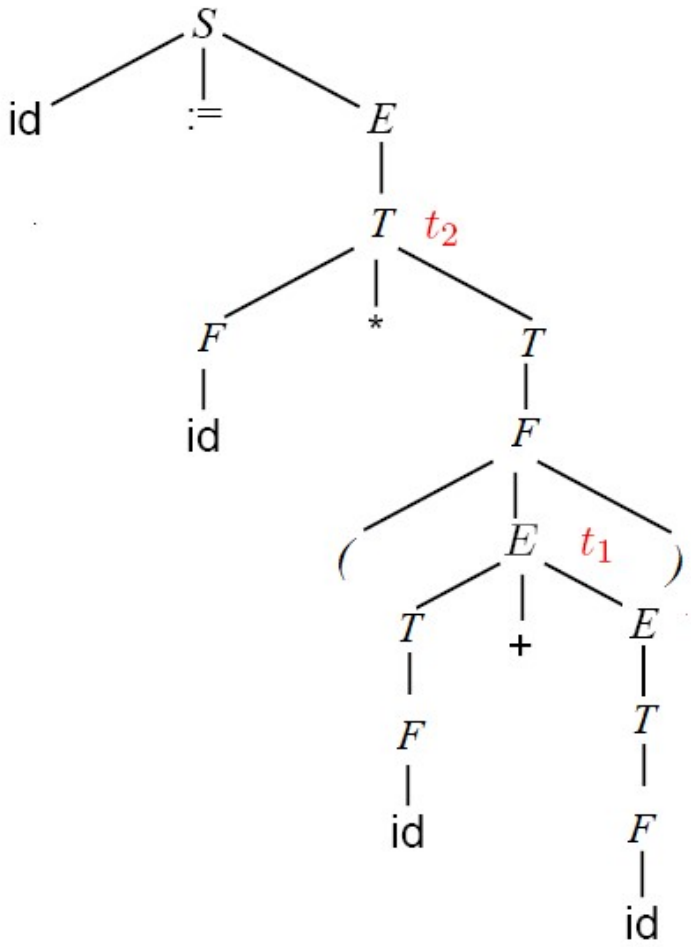
Sản xuất	Quy tắc ngữ nghĩa
$S \rightarrow$ id := E	{S.code = E.code || gen(id.place ':=' E.place)}
$E \rightarrow T+E_1$	{E.place = newtemp ; E.code = T.code || E_1.code || gen(E.place':='T.place'+'E_1.place) }
$E \rightarrow T$	{E.place = T.place ; E.code = T.code}
$T \rightarrow F^* T_1$	{T.place = newtemp ; T.code = F.code || T_1.code || gen(T.place':='F.place'*'T_1.place) }
$T \rightarrow F$	{T.place = F.place ; T.code = F.code}
$F \rightarrow (E)$	{F.place= E.place ; F.code = E.code}
$F \rightarrow$ id	{F.place = id.place ; F.code = '' }

Hàm ***newtemp*** trả về một dãy các tên khác nhau t_1, t_2… cho lời gọi kế tiếp.

E.*place:* là tên sẽ giữ giá trị của E

E.*code:* là dãy các câu lệnh 3 địa chỉ dùng để ước lượng E

Mã cho lệnh gán a := b * (c + d)



$$t_1 := c + d$$
$$t_2 := b * t_1$$
$$a := t_2$$

Cài đặt câu lệnh ba địa chỉ

- Khi biểu diễn mã ba địa chỉ, để tiết kiệm không gian nhớ, có thể dùng các dạng biểu diễn sau:
  - Bộ bốn
  - Bộ ba
  - Bộ ba gián tiếp

Bộ bốn (Quadruples)

t1 := y*z

t2 := x+t1

a :=t2

	op	*arg1*	*arg2*	*result*
(0)	*	y	z	t_1
(1)	+	x	t_1	t_2
(2)	:=	t_2		a

- Định danh trung gian phải được thêm vào bảng kí hiệu khi chúng được tạo ra.
- Cấu trúc giống như văn bản, dễ dàng tối ưu mã
- Dễ dàng truy cập biến trung gian thông qua bảng ký hiệu
- Việc tạo ra nhiều định danh trung gian làm tăng độ phức tạp tính toán (không gian, thời gian)

Bộ ba (Triples)

t1:= y * z
t2:= x + t_1
a:= t2

	op	*arg1*	*arg2*
(0)	*	y	z
(1)	+	x	(0)
(2)	:=	a	(1)

- Định danh trung gian không được thêm vào trong bảng kí hiệu mà sử dụng con trỏ đến kết quả tính toán ở bước trước.
- Rất khó thực hiện tối ưu mã, vì khi cập nhật một lệnh, phải cập nhật tất cả các lệnh liên quan.

Bộ ba gián tiếp

- Các phép tính được thực hiện riêng biệt.
- Con trỏ đến kết quả phép tính và được lưu trữ và sắp thứ tự.
- Tác dụng tương tự như biểu diễn bộ bốn nhưng cần ít không gian hơn.
- Các định danh trung gian được tạo một cách ngầm định và dễ sắp xếp lại mã khi tối ưu.

	op
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

	op	*arg1*	*arg2*
(14)	uminus	c	
(15)	*	b	(14)
(16)	uminus	c	
(17)	*	b	(16)
(18)	+	(15)	(17)
(19)	:=	a	(18)

Định nghĩa tựa cú pháp để sinh ra mã lệnh 3 địa chỉ cho lệnh gán

Sản xuất	**Quy tắc ngữ nghĩa**
S→ id:=E	{p:=lookup (id.name); if p <> nil then emit (p':='E.place) else error}
$E \rightarrow E_1 + E_2$	{E.place = newtemp; emit(E.place ':=' E_1.place '+' E_2.place)}
$E \rightarrow E_1 * E_2$	{E.place = newtemp; emit(E.place ':=' E_1.place '*' E_2.place)}
$E \rightarrow -E_1$	{E.place = newtemp; emit(E.place ':=' 'uminus E_1.place)}
$E \rightarrow (E_1)$	{E.place = E_1.place)}
E → id	{p:=lookup (id.name); if p <> nil then E.place =p else error}

Giải thích

- Hàm lookup tìm trong bảng kí hiệu xem có hay không một tên được cho bởi *id.name*. Việc tìm kiếm áp dụng luật phạm vi gần nhất.
- Thủ tục *emit* để đưa mã 3 địa chỉ vào một tệp text chứ không ghép vào thuộc tính code cho các kí hiệu không kết thúc.
- Trong quá trình dịch ra mã trung gian, tệp *output* chứa thuộc tính *code* của kí hiệu không kết thúc (biến) trong vế trái sản xuất được tạo ra bằng cách ghép thuộc tính *code* của kí hiệu không kết thúc trong vế phải theo đúng thứ tự xuất hiện.

Sinh mã biểu thức Logic

- Biểu thức logic được sinh bởi văn phạm sau:
  B→ B or B | B and B | not B | (B) | id relop id | true |false
- Trong đó:
  - or và and kết hợp trái
  - or có độ ưu tiên thấp nhất tiếp theo là and, và not
  - Những thông tin này có thể thêm vào bộ phân tích cú pháp dưới lên sử dụng quan hệ thứ bậc toán tử. Kết quả cho 1 cây phân tích cú pháp với các phép toán được thực hiện theo đúng thứ tự ưu tiên

Biểu diễn hằng logic bằng số

- Mã hóa true và false bằng các số và ước lượng một biểu thức boole tương tự như đối với biểu thức số học
- Có thể biểu diễn true là 1; false là 0
- Hoặc các số khác 0 là true, 0 là false

  Ví dụ: biểu thức a or b and not c
- Mã ba địa chỉ:

  t1 := not c
  t2 := b and t1
  t3 := a or t2
- Biểu thức quan hệ a<b tương đương lệnh điều kiện if a<b then 1 else 0. Mã 3 địa chỉ tương ứng:

  100: if a<b goto 103
  101: t:=0
  102: goto 104
  103: t:= 1
  104:

ĐNTCP dùng số để biểu diễn các giá trị logic

Sản xuất	Quy tắc ngữ nghĩa
$B \rightarrow B_1$ or B_2	B.place:= newtemp(); emit(B.place ':=' B1.place 'or' B2.place)
$B \rightarrow B_1$ and B_2	B.place:= newtemp(); emit(B.place ':=' B1.place 'and' B2.place)
$B \rightarrow$ not B_1	B.place := newtemp(); emit(B.place ':=' 'not' B1.place)
$B \rightarrow id_1$ relop id_2	B.place := newtemp(); emit('if' id1.place relop id2.place 'goto' nextstat + 3); emit(B.place':=''0') ;emit('goto' nextstat + 4) emit(B.place':=''1')
	B.place = newtemp(); emit (B.place ':=''1')
	B.place = newtemp(); emit (B.place ':=''0')

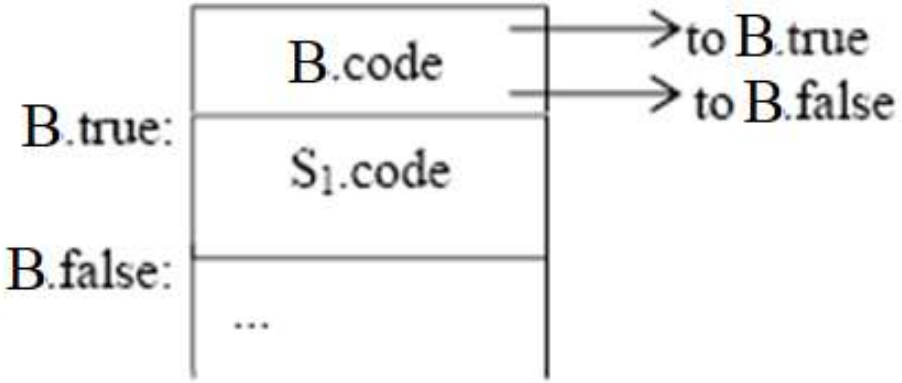
Nextstat cho biết nhãn của câu lệnh 3 địa chỉ đang được sinh

emit: đặt câu lênh 3 địa chỉ vào tệp , emit làm tăng nextstat sau khi thực hiện

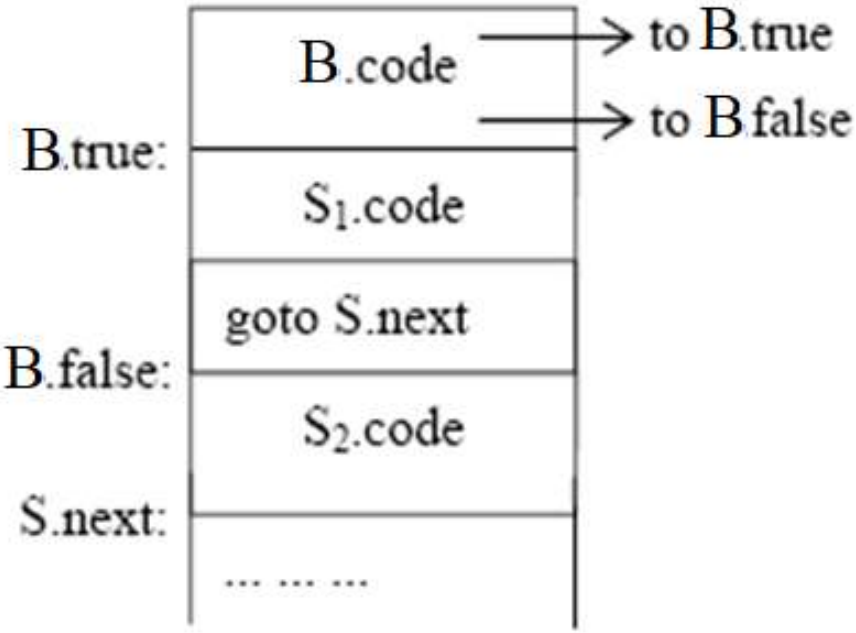
Sinh mã cho các cấu trúc lập trình

- Biểu diễn các giá trị của biểu thức Boole bằng biểu thức đã đến được trong một chương trình.
- Ví dụ: cho câu lệnh sau
- S→ if B then S_1 | iF B then S_1 else S_2 | while B do S_1
- Với mỗi biểu thức B chúng ta kết hợp với 2 nhãn:
  - B.true: nhãn của dòng điều khiển nếu B là true
  - B.false: nhãn của dòng điều khiển nếu B là false
  - S.code: mã lệnh 3 địa chỉ được sinh ra bởi S
  - S.next: là nhãn mã lệnh 3 địa chỉ đầu tiên sẽ thực hiện sau mã lệnh của S
  - S.begin: nhãn địa chỉ lệnh đầu tiên được sinh ra cho S là lệnh while

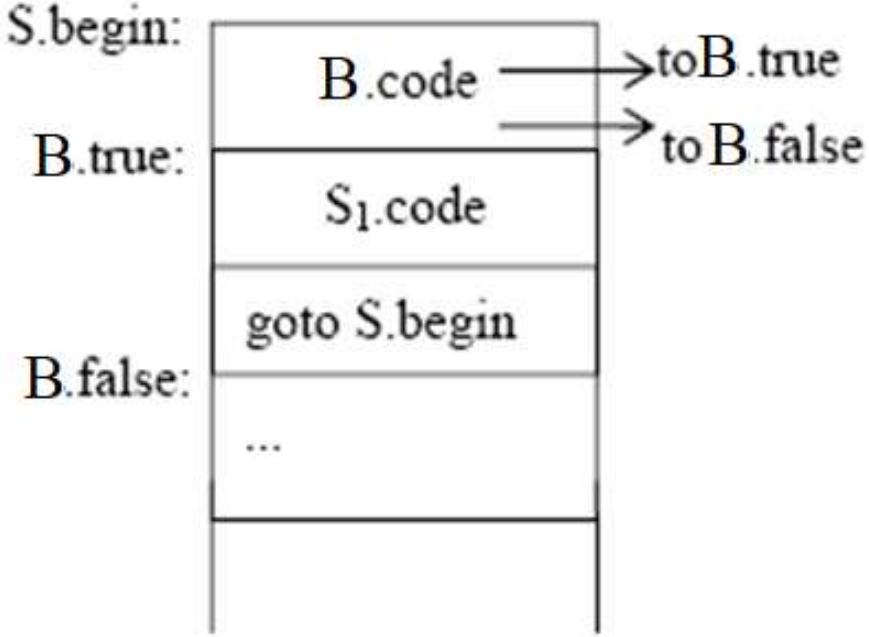
Mã lệnh của các lệnh if-then, if-then-else, while-do



(a) if -then



(b) if -then-else



(c) while-do

Định nghĩa tựa cú pháp cho các cấu trúc lập trình

Sản xuất	**Quy tắc ngữ nghĩa**
$P \rightarrow S$	S.code = newlabel() P.code= S.code||label (S.next)
$S \rightarrow$ assign	S.code = assign.code
$S \rightarrow$ if B then S_1 else S_2	B. true = newlabel() B.false = newlabel() S1.next = S2.next = S.next S.code = B.code||label (B.true)||S_1.code||gen ('goto' S.next)||label (B.false)||S2.code
$S \rightarrow$ while B do S_1	begin = newlabel() B,true = newlabel() B.false = S.next S_1.next = begin S.code = label (begin)||B.code||label (B.true)||S1.code||gen ('goto' begin)
$S \rightarrow S_1S_2$	S_1.next = newlabel() S_2.next = S.next S.code = S_1.code||label(S_1.next)||S_2.code

Dịch biểu thức logic trong các cấu trúc lập trình

- Nếu B có dạng: a<b thì mã lệnh sinh ra có dạng
  - If a<b then goto B.true else goto B.false
- Nếu B có dạng: B_1 or B_2 thì
  - Nếu B1 là true thì B cũng là true
  - Nếu B_1là false thì phải đánh giá B_2; B sẽ là true hay false phụ thuộc B_2
- Tương tự với B1 and B2

Dịch biểu thức logic trong các cấu trúc lập trình

Sản xuất	**Quy tắc ngữ nghĩa**
$B \rightarrow B_1$ or B_2	B_1. true =B. true; B_2. False = newlabel(); B_2. true = B. true; B_2. false = B.false; B.code= B_1.code||gen (label(B.false)|| B_2.code)
$B \rightarrow B_1$ and B_2	B_1. true = newlabel(); B_1.false =B.false; B_2. true = B. true; B_2.false=B.false; B. code= B_1.code||gen (label(B.true)||B_2.code)
$B \rightarrow$ not B_1	B1. true = B.false; B_1.false= B.true; B.code = B_1.code
$B \rightarrow id_1$ relop id_2	B1.code = gen (‘if’ id1.place relop id2.place ‘goto’ B.true) || gen (‘goto’ B.false)
$B \rightarrow$ true	B.code = gen (‘goto’ B. true)
$B \rightarrow$ false	B.code = gen (‘goto’ B. false)

Biểu thức logic ở dạng hỗn hợp

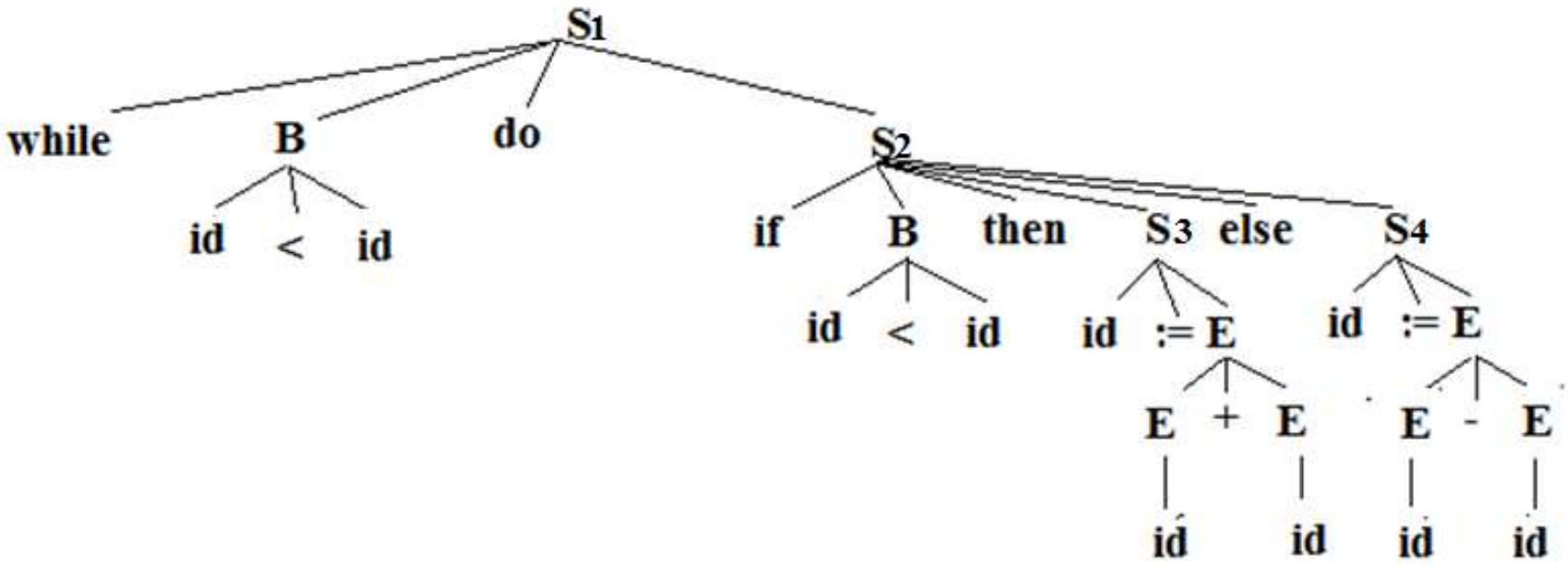
- Thực tế, các biểu thức logic thường chứa các biểu thức số học như trong (a+b)<c
- Ta dùng 2 ký hiệu không kết thúc: E cho biểu thức số học và B cho biểu thức logic

Ví dụ

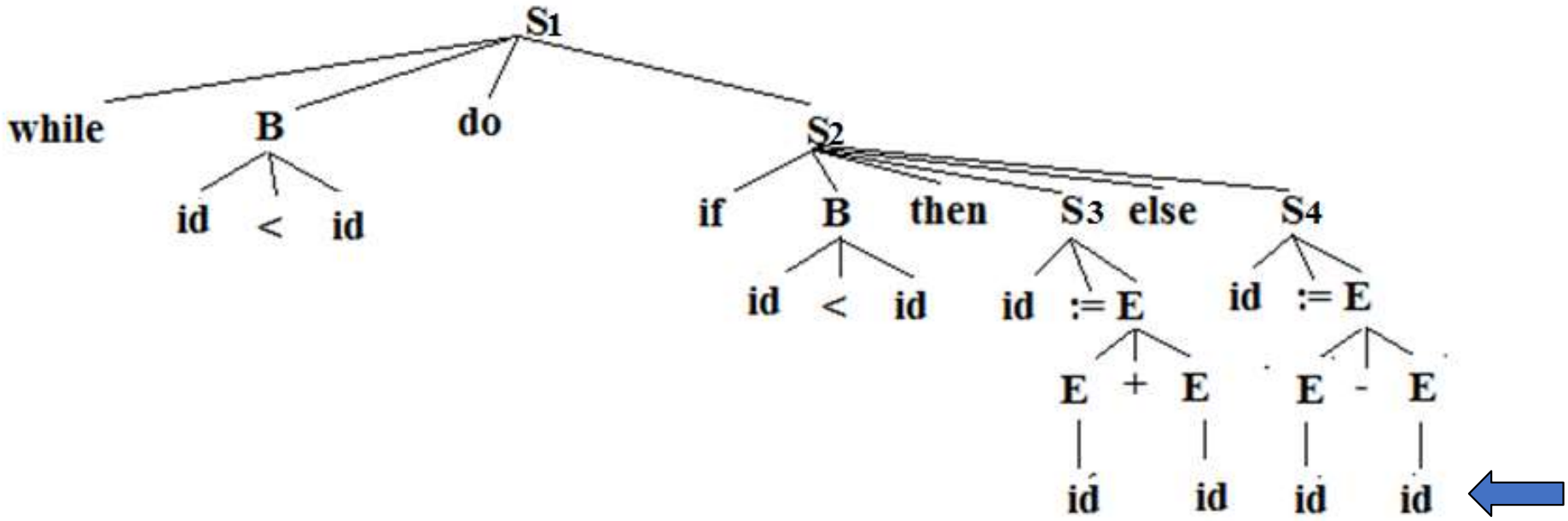
- Sinh mã trung gian cho lệnh sau:

```
while a < b do
if c < d then
x := y + z
else
x := y - z
```

Cây phân tích cú pháp của lệnh

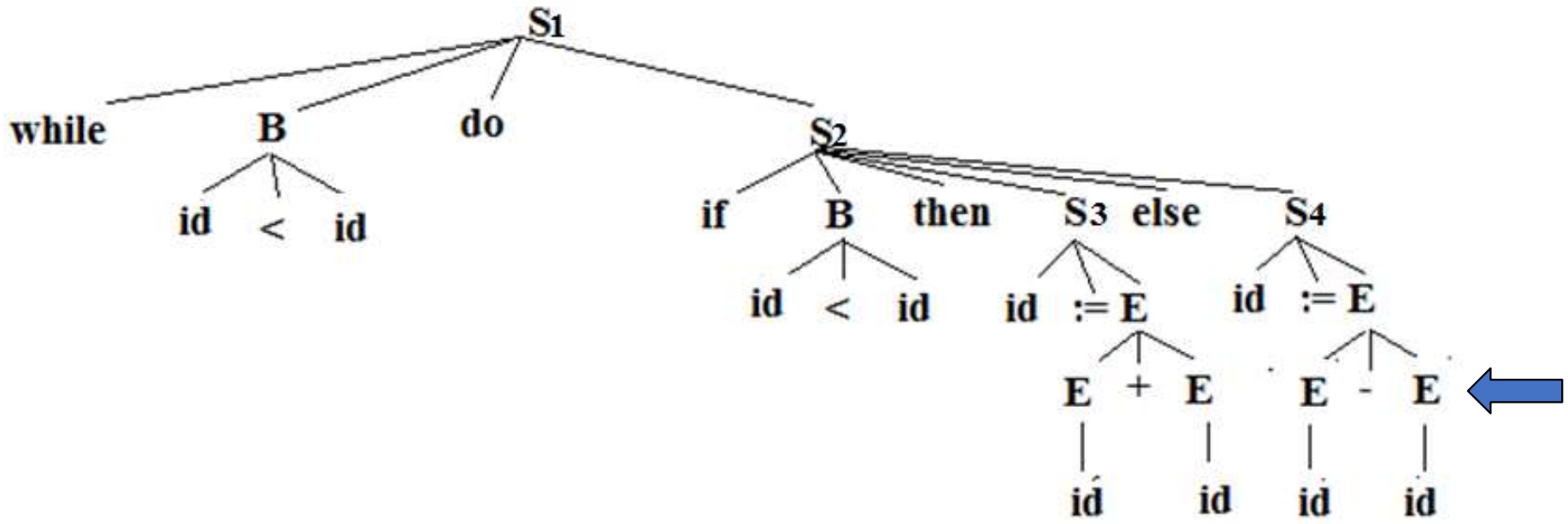


Sinh mã cho biểu thức số học



Thuộc tính
id.place =z

Sinh mã cho biểu thức số học



Quy tắc ngữ nghĩa

E.place = id.place

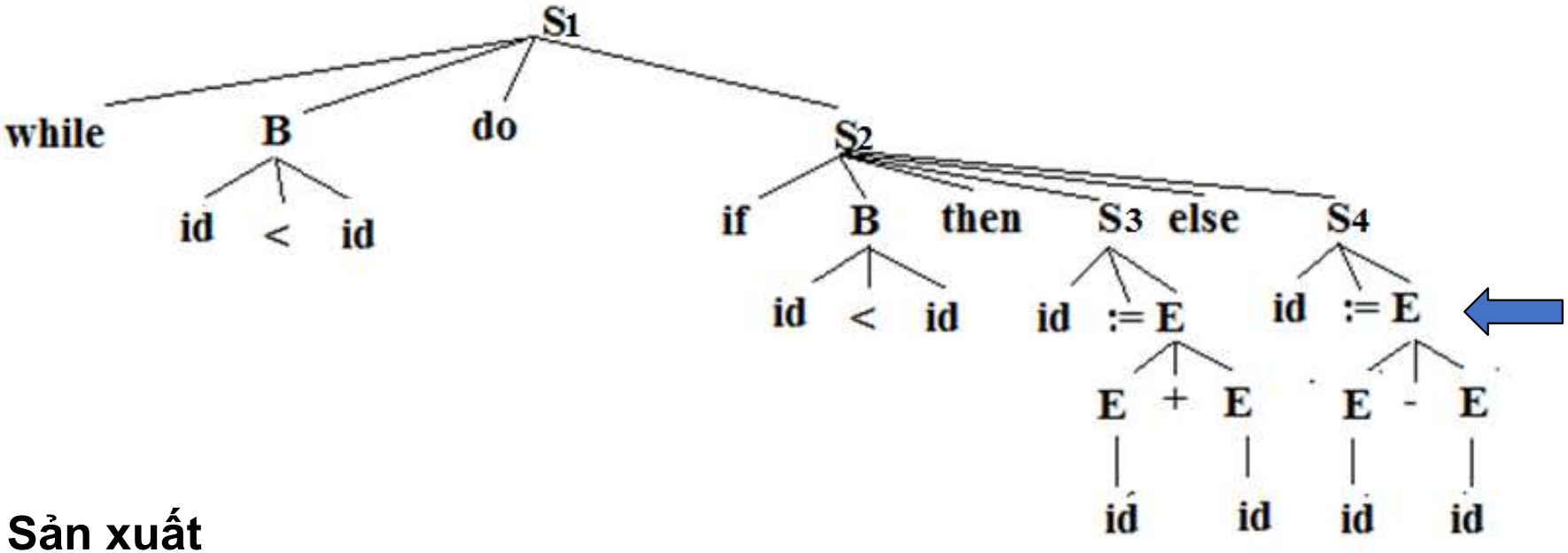
E.code=“”

Thuộc tính

E.place =z

E.code=“”

Sinh mã cho biểu thức số học



Sản xuất

$E \rightarrow E_1 - E_2$

Quy tắc ngữ nghĩa

E.place = newtemp() t1

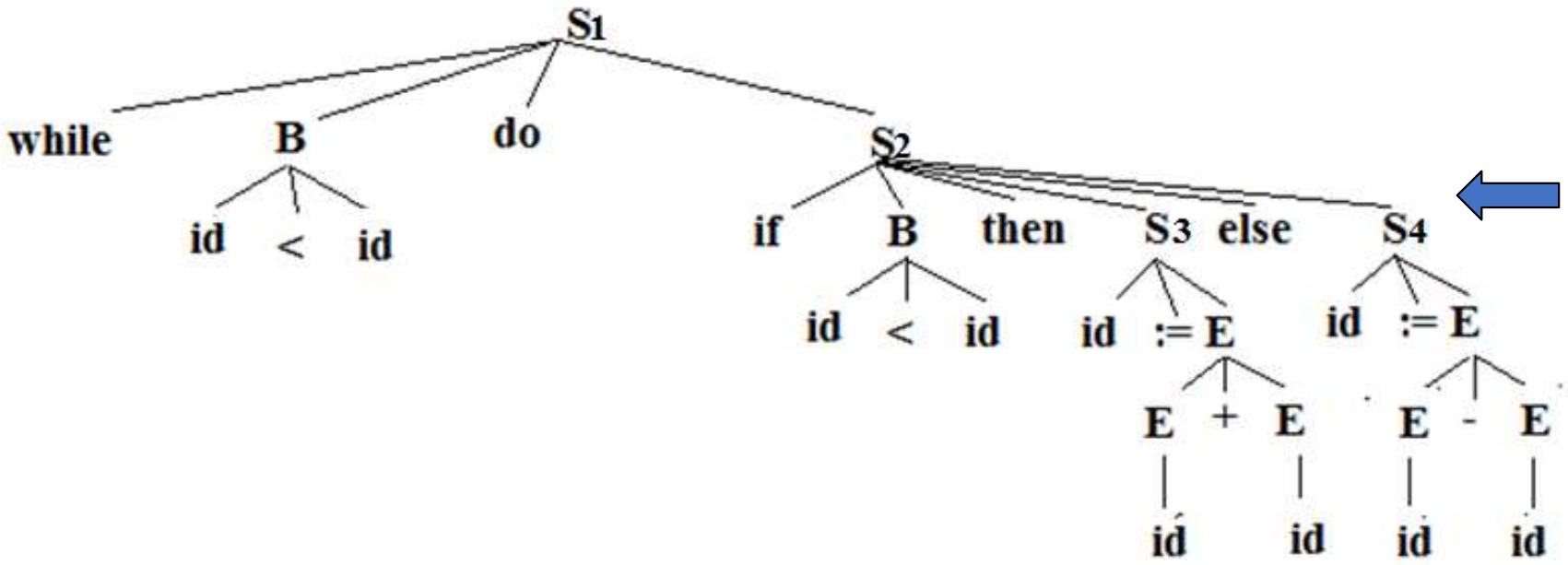
E.Code = E_1.code || E_2.code ||gen (E.place := E_1.place "-" E_2.place)

Thuộc tính

E.place = newtemp() t1

E.code= "t1:=y-z"

Sinh mã cho lệnh gán



Sản xuất	**Quy tắc ngữ nghĩa**
S → id := E	{S.*code* = E.*code* || *gen*(id.*place* ':=' E.*place*)}

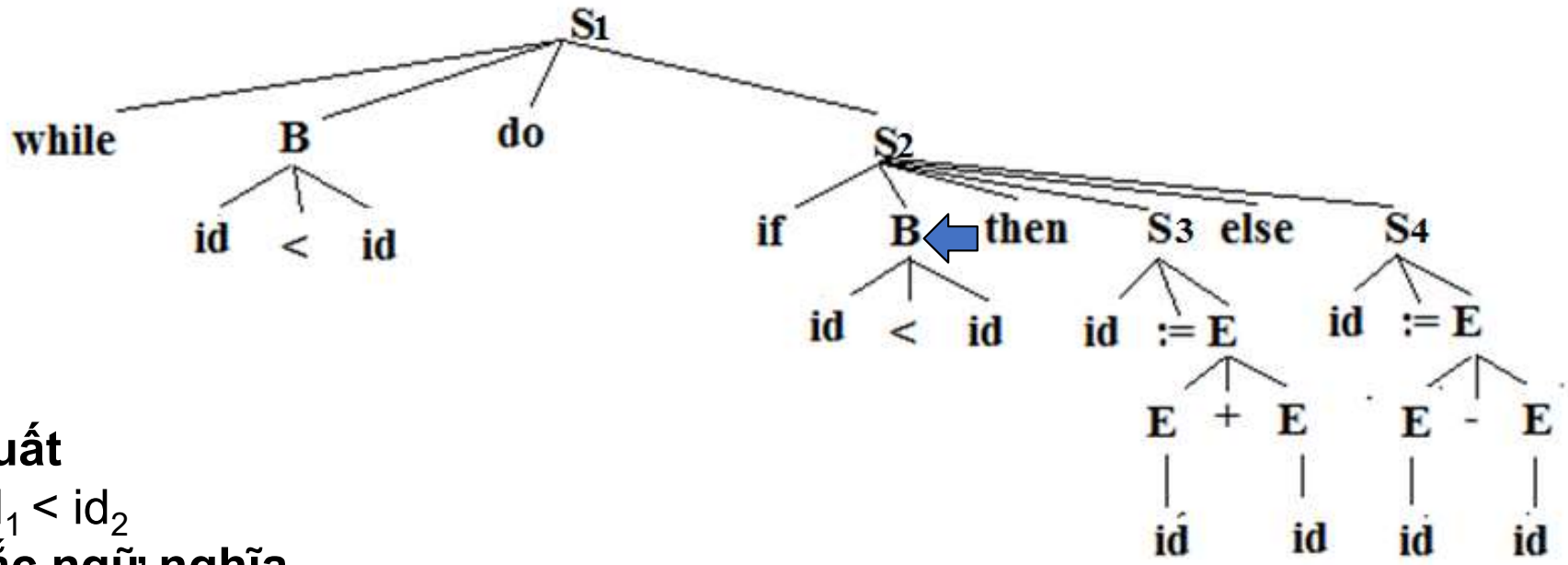
Thuộc tính

id.place = x

S.code= "t1:=y-z

x:=t1"

Sinh mã cho biểu thức logic



Sản xuất

$B \rightarrow id_1 < id_2$

Quy tắc ngữ nghĩa

B.code = gen ('if' id_1.place < id_2.place) 'goto' B. true || gen ('goto' B.false)

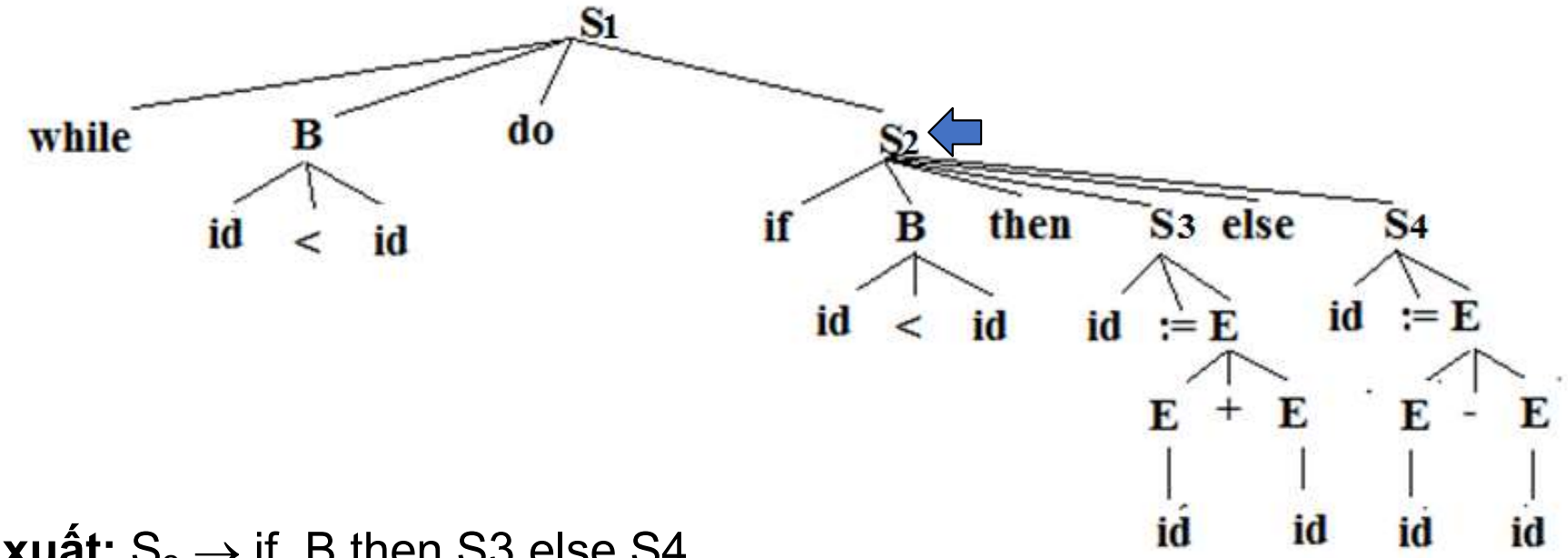
Thuộc tính

B.code

if c < d goto L_1

goto L_2

Sinh mã cho lệnh IF



Sản xuất: $S_2 \rightarrow$ if B then S3 else S4

Quy tắc ngữ nghĩa

B.true=newlabel()

B.false=newlabel()

S_3.next = S_4.next = S_2.next

S_2.code = B.code || label (B.true) ||

S_3.code || gen('goto' S_2.next ||

label(B.false) || S_4.code

Thuộc tính (B của lệnh IF)

B.true = newlabel() -> L_1

B.false= newlable() -> L_2

S_2.next = S_3.next= S_4.next

S_2.code

if c < d goto L_1

goto L_2

L_1: t2:= y + z

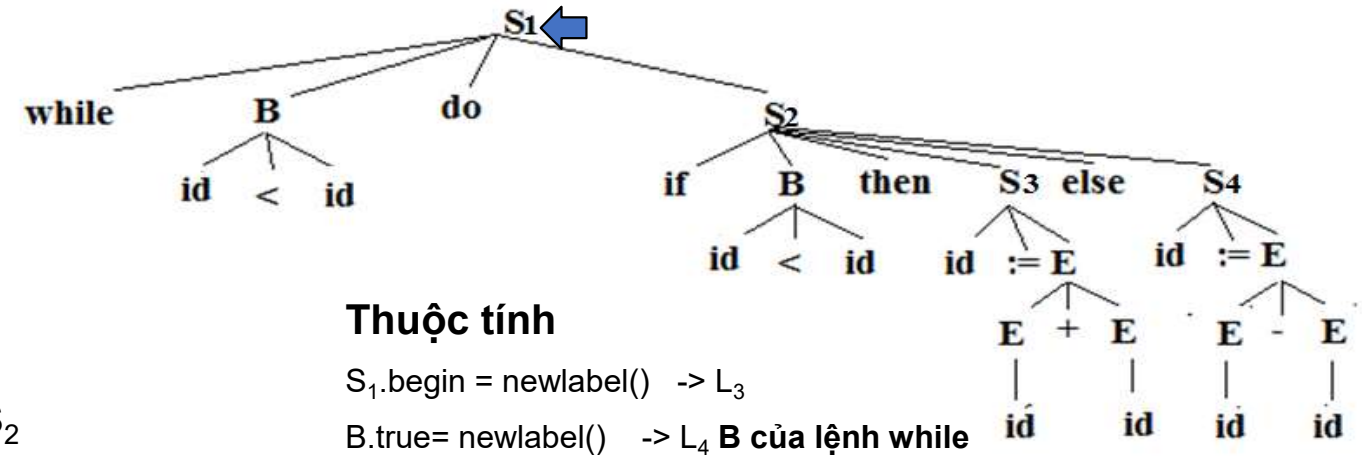
x := t2

goto S_2.next

L_2: t1 := y – z

x := t1

Sinh mã cho lệnh WHILE



Sản xuất:

$S_1 \rightarrow$ while B do S_2

Quy tắc ngữ nghĩa:

S_1.begin = newlabel()

B.True = newlabel()

B.False = S_1.next

S_1.code =

S_1.begin || B.code

|| B.true || S_2.code

||gen ('goto' S_1.begin)

Thuộc tính

S_1.begin = newlabel() -> L_3

B.true= newlabel() -> L_4 **B của lệnh while**

B.False = S.next

S_2.next = begin = L_3 => S_2.next = S_3.next = L_3

S.code

L_3: if a< b goto L_4

goto L_0

L_4: if c < d goto L_1

goto L_2

L_1: t1= y + z

x = t1

goto L_3

L_2:t2 = y – z

x = t1

goto L3

Nhãn L_0 sẽ được sinh khi phân tích sản xuất S $\rightarrow$ S_1S_2 và áp dụng các quy tắc ngữ nghĩa của nó