



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài 11

Sinh mã trung gian

Nội dung

- Mã ba địa chỉ
- Sinh mã cho lệnh gán
- Sinh mã cho các biểu thức logic
- Sinh mã cho các cấu trúc lập trình

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
 - Ký pháp Ba Lan sau (hậu tố)
 - Mã 3 địa chỉ ...

Mã trung gian

- Được sản sinh dưới dạng một chương trình cho một máy trừu tượng
- 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

Mã trung gian của $t2 = x + y * z$

- $t1 := y * z$
- $t2 := x + t1$

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

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 ĐNTCP

- Thuộc tính tổng hợp S.code biểu diễn mã ba địa chỉ của lệnh
- Các tên trung gian được sinh ra cho các 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

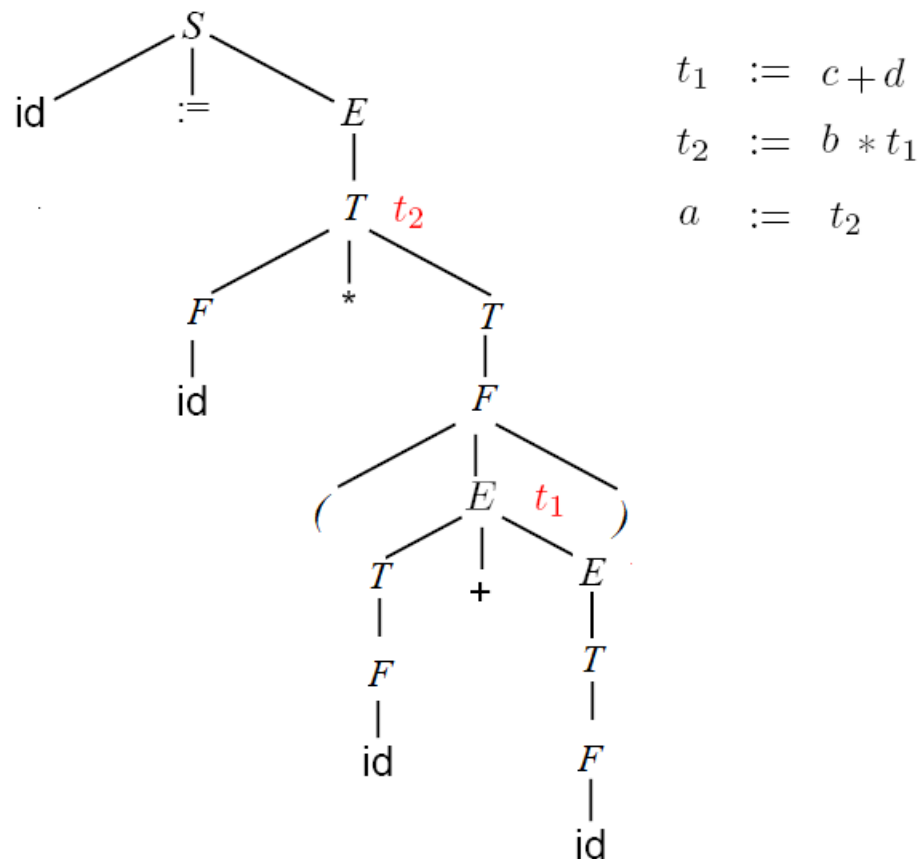
Sản xuất	Quy tắc ngữ nghĩa
$S \rightarrow id := E$	$\{S.code = E.code \parallel gen(id.place ':=' E.place)\}$
$E \rightarrow T + E_1$	$\{E.place = newtemp ; E.code = T.code \parallel E_1.code \parallel gen(E.place ':=' T.place '+' E_1.place) \}$
$E \rightarrow T$	$\{E.place = T.place ; E.code = T.code\}$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

Hàm ***newtemp*** trả về một dãy các tên khác nhau t_1, t_2, \dots 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 3 địa chỉ

Bộ bốn (Quadruples)

$t_1 := c + d$

$t_2 := b * t_1$

$a := t_2$

	<i>op</i>	<i>arg1</i>	<i>arg2</i>	<i>result</i>
(0)	+	c	d	t_1
(1)	*	b	t_1	t_2
(2)	:=	t_2		a

Tên tạm phải được thêm vào bảng kí hiệu khi chúng được tạo ra.

Cài đặt câu lệnh 3 địa chỉ

- Bộ ba (Triples)

$t_1 := c + d$

$t_2 := b * t_1$

$a := t_2$

	<i>op</i>	<i>arg1</i>	<i>arg2</i>
(0)	+	c	d
(1)	*	b	(0)
(2)	assign	a	(1)

Tên tạm không được thêm vào trong bảng kí hiệu.

Các dạng khác của câu lệnh 3 địa chỉ

- Ví dụ:
 $x[i] := y$ $x := y[i]$
- Sử dụng 2 cấu trúc bộ ba

	<i>op</i>	<i>arg1</i>	<i>arg2</i>
(0)	[]	x	i
(1)	:=	(0)	y

	<i>op</i>	<i>arg1</i>	<i>arg2</i>
(0)	[]	y	i
(1)	:=	x	(0)

Cài đặt câu lệnh 3 địa chỉ

- Bộ 3 gián tiếp: sử dụng một danh sách các con trỏ các bộ 3

	<i>op</i>		<i>op</i>	<i>arg1</i>	<i>arg2</i>
(0)	(14)	(14)	uminus	c	
(1)	(15)	(15)	*	b	(14)
(2)	(16)	(16)	uminus	c	
(3)	(17)	(17)	*	b	(16)
(4)	(18)	(18)	+	(15)	(17)
(5)	(19)	(19)	assign	a	(18)

ĐNTCP để sinh ra mã lệnh 3 địa chỉ cho lệnh gán

$S \rightarrow id := E$ $\{ p := \text{lookup}(id.name);$
 $\text{if } p \neq \text{nil then emit}(p := E.place) \text{ else error} \}$

$E \rightarrow E_1 + E_2$ $\{ E.place := \text{newtemp};$
 $\text{emit}(E.place := E_1.place + E_2.place) \}$

$E \rightarrow E_1 * E_2$ $\{ E.place := \text{newtemp};$
 $\text{emit}(E.place := E_1.place * E_2.place) \}$

$E \rightarrow - E_1$ $\{ E.place := \text{newtemp};$
 $\text{emit}(E.place := \text{'unimus' } E_1.place) \}$

$E \rightarrow (E_1)$ $\{ E.place := E_1.place \}$

$E \rightarrow id$ $\{ p := \text{lookup}(id.name);$
 $\text{if } p \neq \text{nil then } E.place := p \text{ else error} \}$

Tên trong bảng kí hiệu

- Hàm lookup sẽ tìm trong bảng kí hiệu xem có hay không một tên được cho bởi *id.name*. Nếu có thì trả về con trỏ của ô, nếu không thì trả về nil.
- Thủ tục emit để đưa mã 3 địa chỉ vào một tập tin output chứ không xây dựng thuộc tính code cho các kí hiệu chưa kết thúc như gen. Quá trình dịch thực hiện bằng cách đưa ra một tập tin output nếu thuộc tính code của kí hiệu không kết thúc trong vế trái sản xuất được tạo ra bằng cách nối thuộc tính code của kí hiệu không kết thúc trong vế phải theo đúng thứ tự xuất hiện của các kí hiệu chưa kết thúc ở vế phải.

Tên trong bảng kí hiệu

- Xét sản xuất $D \rightarrow \text{proc id; ND}_1; S$
- Các tên trong lệnh gán sinh ra bởi kí hiệu không kết thúc S sẽ được khai báo trong chương trình con này hoặc trong chương trình chứa nó.
- Khi khai báo tới một tên thì trước hết hàm lookup sẽ tìm xem tên đó có trong bảng kí hiệu hiện hành hay không, nếu không thì dùng con trỏ trong header của bảng để tìm bảng kí hiệu bao nó và tìm trong đó, nếu không tìm thấy trong tất cả các mức thì lookup trả về nil.

Địa chỉ hóa các phần tử của mảng

- Các phần tử của mảng có thể truy xuất nhanh nếu chúng được lưu trữ trong một khối ô nhớ kế tiếp nhau. Trong mảng một chiều, nếu kích thước của một phần tử là w thì địa chỉ tương đối phần tử thứ i của mảng A được tính theo công thức:
- $A[i] = \text{base} + (i - \text{low}) * w$
- Trong đó:
 - Low: cận dưới tập chỉ số
 - Base: địa chỉ tương đối của ô nhớ cấp phát cho mảng (địa chỉ tương đối của $A[\text{low}]$)
- Tương đương $A[i] = i * w + (\text{base} - \text{low} * w)$
- Trong đó:
 - $c = \text{base} - \text{low} * w$ có thể được tính tại thời gian dịch và lưu trong bảng kí hiệu
- $\Rightarrow A[i] = i * w + c$

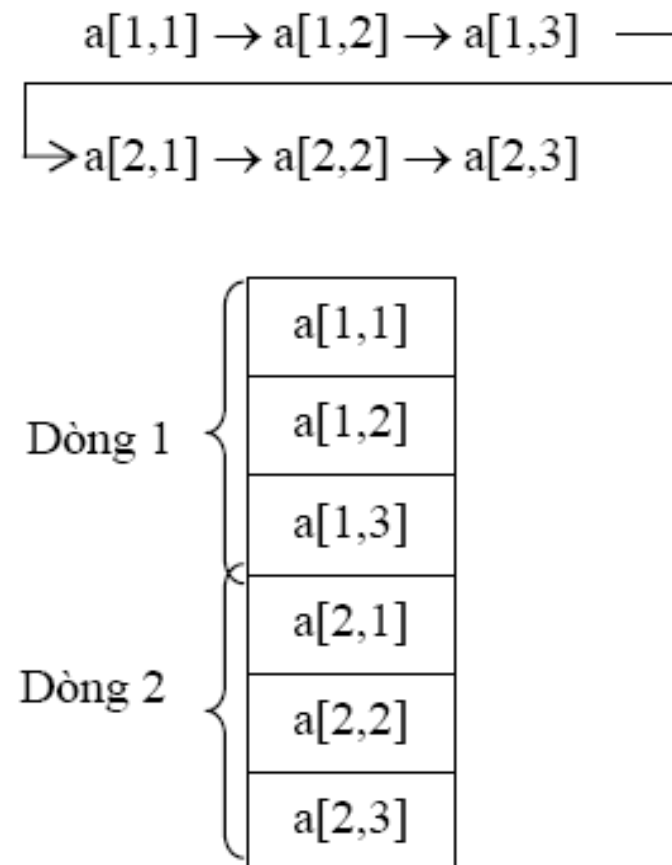
Địa chỉ hóa các phần tử của mảng 2 chiều

- Theo dòng

Địa chỉ tương đối của $A[i_1, i_2] =$
 $\text{base} + ((i_1 - \text{low}_1) * n_2 + i_2 - \text{low}_2) * w$

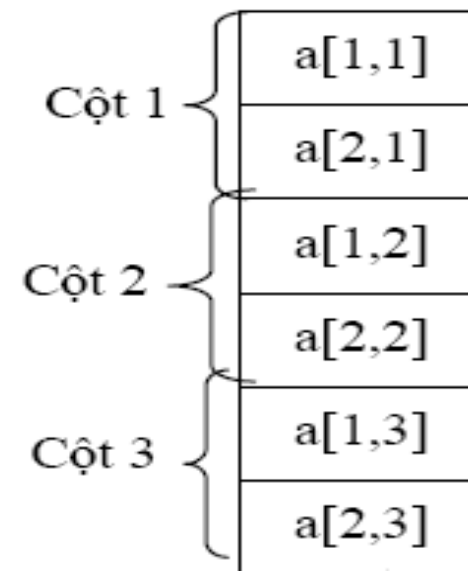
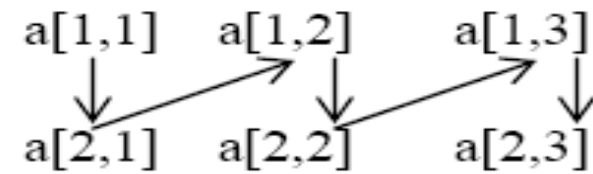
$\text{low}_1, \text{low}_2$: cận dưới cho i_1 và i_2

n_2 : số lượng các giá trị mà i_2 có thể nhận. Nếu high_2 là cận trên của i_2 thì $n_2 = \text{high}_2 - \text{low}_2 + 1$



Địa chỉ hóa các phần tử của mảng 2 chiều

- Theo cột



Sinh mã biểu thức Logic

- Biểu thức logic được sinh bởi văn phạm sau:
 $B \rightarrow B \text{ or } B \mid B \text{ and } B \mid \text{not } B \mid (B) \mid \text{id relop id} \mid \text{true} \mid \text{false}$
- 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 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 $a \text{ or } b \text{ and not } c$

- Mã 3 địa chỉ:
 $t1 = \text{not } c$
 $t2 = b \text{ and } t1$
 $t3 = a \text{ or } t2$
- 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

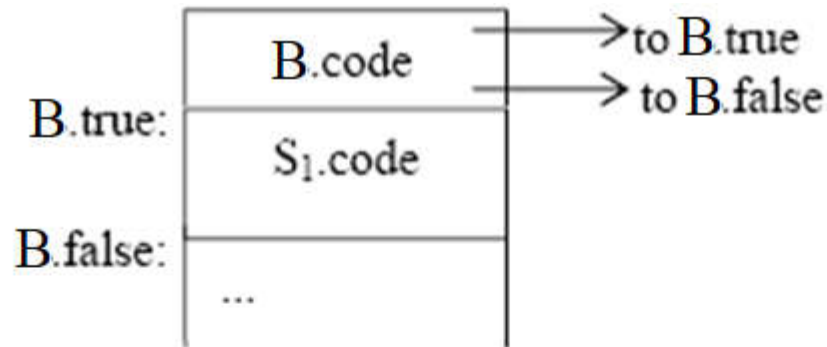
Sản xuất	Quy tắc ngữ nghĩa
$B \rightarrow B_1 \text{ or } B_2$	B.place := newtemp(); emit(B.place ':= ' B1.place 'or' B2.place)
$B \rightarrow B_1 \text{ and } B_2$	B.place := newtemp(); emit(B.place ':= ' B1.place 'and' B2.place)
$B \rightarrow \text{not } B_1$	B.place := newtemp(); emit(B.place ':= ' 'not' B1.place)
$B \rightarrow \text{id}_1 \text{ relop id}_2$ Emit: đặt câu lệnh 3 địa chỉ vào tập tin, emit làm tăng nextstat sau khi thực hiện	B.place := newtemp(); emit('if' id1.place relop id2.place 'goto' nextstat + 3); emit(B.place := "0") ;emit('goto' nextstat + 2) emit(B.place := "1")
$B \rightarrow \text{true}$	B.place = newtemp(); emit (B.place ':= "1")
$B \rightarrow \text{false}$	B.place = newtemp(); emit (B.place ':= "0")

Nextstat cho biết nhãn của câu lệnh 3 địa chỉ tiếp theo.

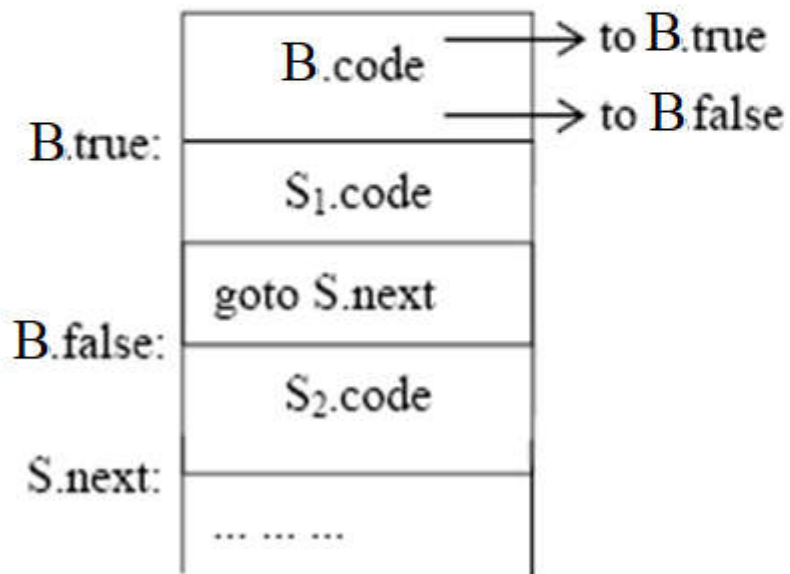
Sinh mã cho các cấu trúc lập trình

- 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
- $S \rightarrow \text{if } B \text{ then } S_1 \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \mid \text{while } B \text{ do } S_1$
- 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

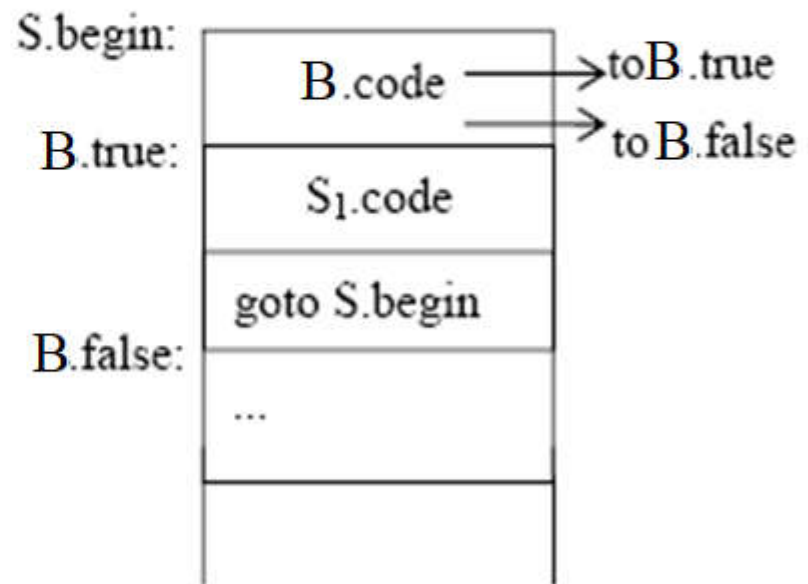
Mã lệnh của các lệnh if-then, if-then-else, while-do



(a) if -then



(b) if -then-else



(c) while-do

ĐNTCP cho các cấu trúc lập trình

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow \text{assign}$	$S.code = \text{assign.code}$
$S \rightarrow \text{if } B \text{ then } S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$S \rightarrow \text{if } B \text{ then } S_1 \text{ else } S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\parallel label(B.true) \parallel S_1.code$ $\parallel gen('goto' S.next)$ $\parallel label(B.false) \parallel S_2.code$
$S \rightarrow \text{while } B \text{ do } S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \parallel B.code$ $\parallel label(B.true) \parallel S_1.code$ $\parallel gen('goto' begin)$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$

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: B1 or B2 thì
 - Nếu B1 là true thì B cũng là true
 - Nếu B1 là false thì phải đánh giá B2; B sẽ là true hay false phụ thuộc B2
- Tương tự với B1 and B2

Dịch biểu thức logic trong các cấu trúc lập trình

Sản xuất	Quy tắc ngữ nghĩa
$B \rightarrow B_1 \text{ or } B_2$	$B_1.\text{true} := B.\text{true}; B_1.\text{false} := \text{newlable};$ $B_2.\text{true} := B.\text{true}; B_2.\text{false} := B.\text{false};$ $B.\text{code} := B_1.\text{code} \parallel \text{gen}(B.\text{false}: \parallel B_2.\text{code})$
$B \rightarrow B_1 \text{ and } B_2$	$B_1.\text{true} := \text{newlable}; B_1.\text{false} := B.\text{false};$ $B_2.\text{true} := B.\text{true}; B_2.\text{false} := B.\text{false};$ $B.\text{code} := B_1.\text{code} \parallel \text{gen}(B.\text{true}: \parallel B_2.\text{code})$
$B \rightarrow \text{not } B_1$	$B_1.\text{true} := B.\text{false}; B_1.\text{false} := B.\text{true};$ $B.\text{code} := B_1.\text{code}$
$B \rightarrow (B_1)$	$B_1.\text{true} := B.\text{true}; B_1.\text{false} := B.\text{false};$ $B.\text{code} := B_1.\text{code}$
$B \rightarrow \text{id}_1 \text{ relop } \text{id}_2$	$B.\text{code} := \text{gen}(\text{'if' id1.place relop id2.place 'goto' B.true});$ $\parallel \text{gen}(\text{'goto' B.false})$
$B \rightarrow \text{true}$	$B.\text{Code} := \text{gen}(\text{'goto' 'E.true'})$
$B \rightarrow \text{false}$	$B.\text{Code} := \text{gen}(\text{'goto' 'E.true'})$

Biểu thức logic ở dạng hỗn hợp

- Thực tế, các biểu thức logic thường chứa các biểu thức số học như trong $(a+b)<c$
- Để đơn giản, ta vẫn dùng 2 ký hiệu không kết thúc E cho biểu thức số học và B cho biểu thức logic
- Nếu dùng chung một lý hiệu không kết thúc, cần lưu thêm thuộc tính kind để chỉ biểu thức là số học hay logic

Ví dụ

- Sinh mã trung gian cho lệnh sau:

while $a < b$ do

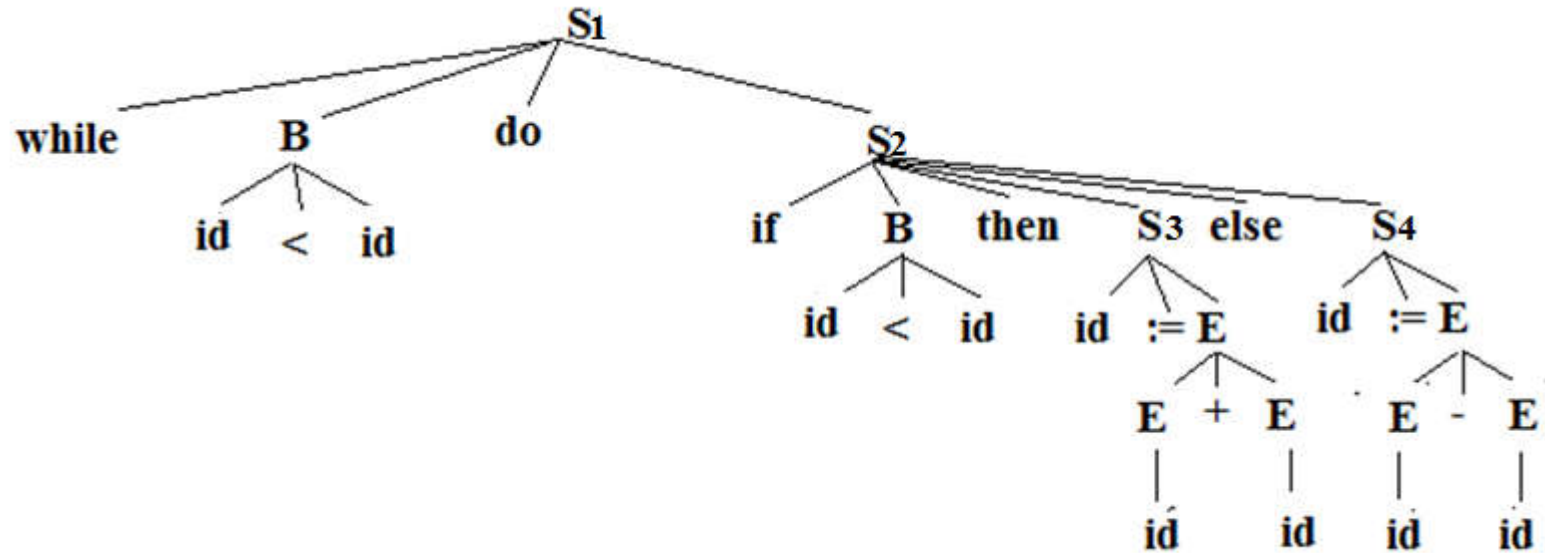
if $c < d$ then

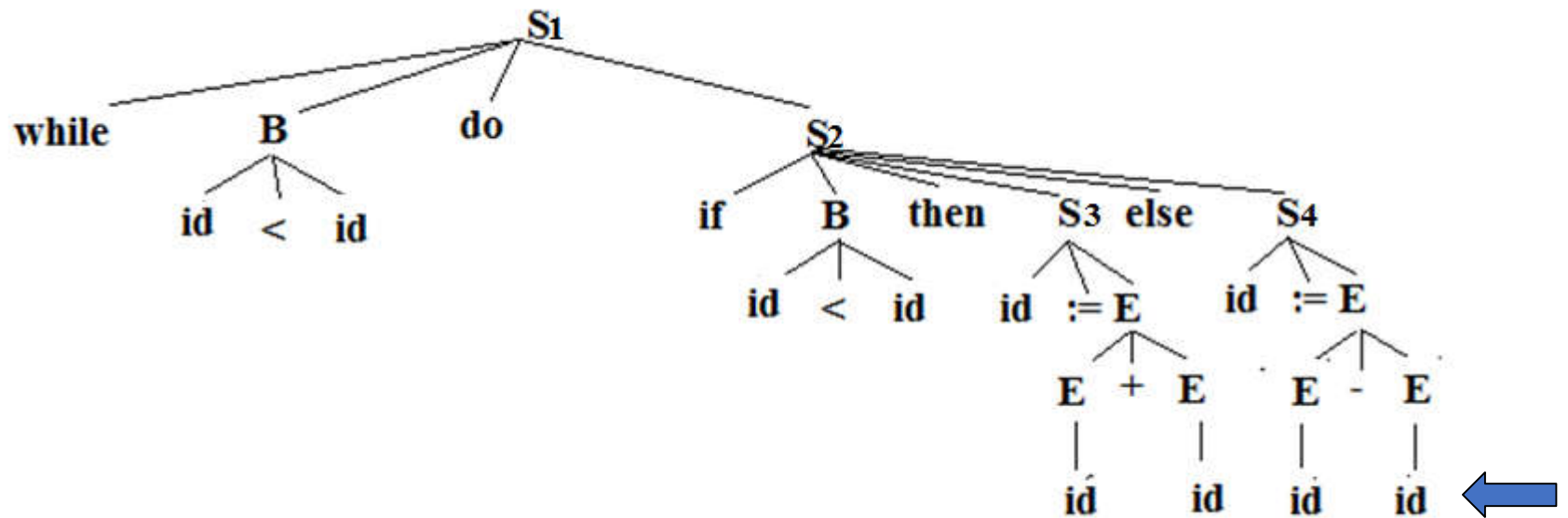
$x := y + z$

else

$x := y - z$

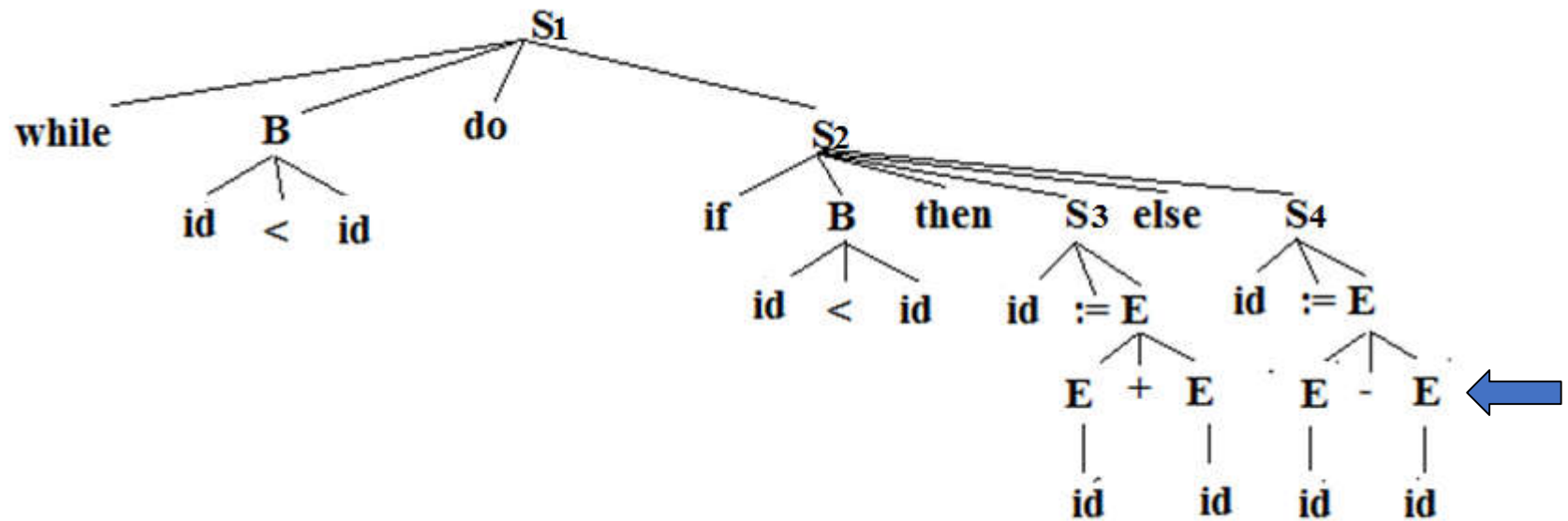
Cây PT cú pháp của lệnh





Thuộc tính

id.place = z



Quy tắc ngữ nghĩa

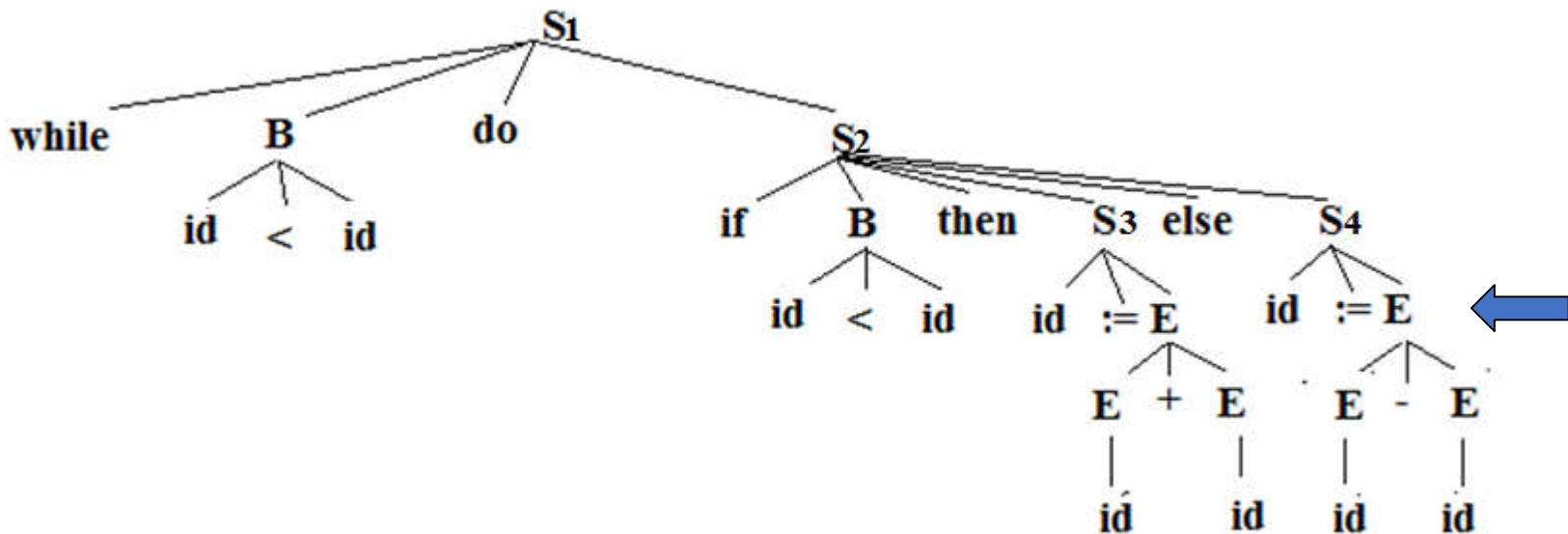
E.place = id.place

E.code=""

Thuộc tính

E.place = z

E.code=""



SX:

$E \rightarrow E_1 - E_2$

Quy tắc ngữ nghĩa

$E.place = newtemp() \quad t1$

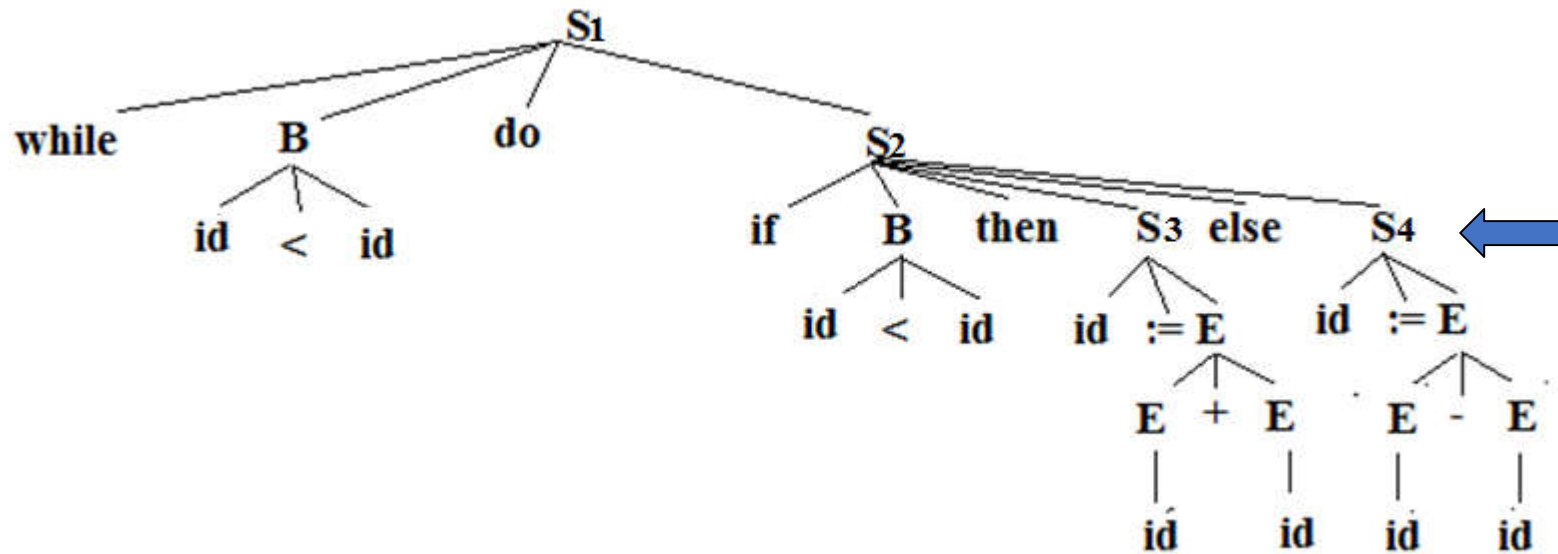
$E.Code = E_1.code \parallel E_2.code \parallel gen(E.place :=$

$E_1.place \text{ “-” } E_2.place)$

Thuộc tính

$E.place = newtemp() \quad t1$

$E.code = \text{“}t1 := y - z\text{”}$



SX Quy tắc ngữ nghĩa

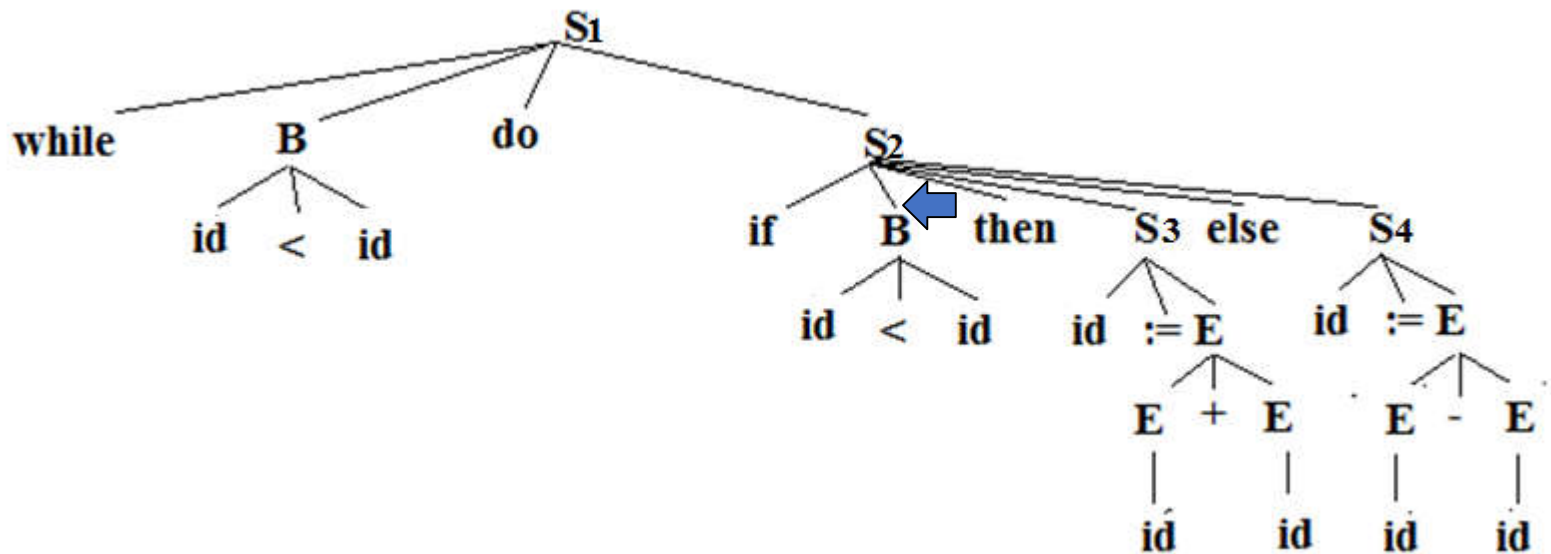
$S \rightarrow \text{id} := E \quad \{ S.code = E.code || gen(id.place \text{ ':=' } E.place) \}$

Thuộc tính

$id.place = x$

$S.code = \text{"t1:=y-z"}$

$x := t1$



SX:

$B \rightarrow id_1 < id_2$

QTNN:

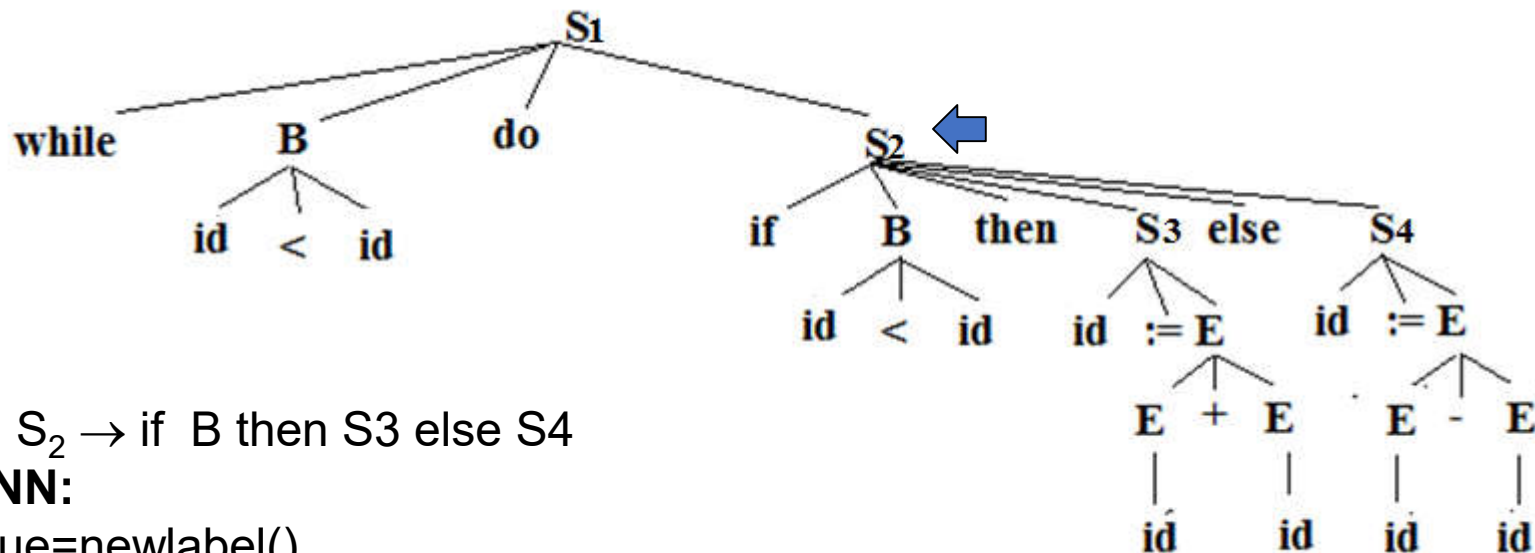
$B.code = \text{gen}('if' \ id_1.place < id_2.place) \ 'goto' \ B.true \ || \ \text{gen}('goto' \ B.false)$

Thuộc tính

B.code

if c < d goto L₁

goto L₂



SX: $S_2 \rightarrow \text{if } B \text{ then } S_3 \text{ else } S_4$

QTNN:

B.true=newlabel()

B.false=newlabel()

$S_3.\text{next} = S_4.\text{next} = S_2.\text{next}$

$S_2.\text{code} = B.\text{code} \parallel \text{label}(B.\text{true}) \parallel$

$S_3.\text{code} \parallel \text{gen}(\text{'goto' } S_2.\text{next}) \parallel$

$\text{label}(B.\text{false}) \parallel S_4.\text{code}$

Thuộc tính (B của lệnh if)

B.true = newlabel() $\rightarrow L_1$

B.false= newlable() $\rightarrow L_2$

$S_2.\text{next} = S_3.\text{next} = S_4.\text{next}$

$S_2.\text{code}$

if c < d goto L_1

goto L_2

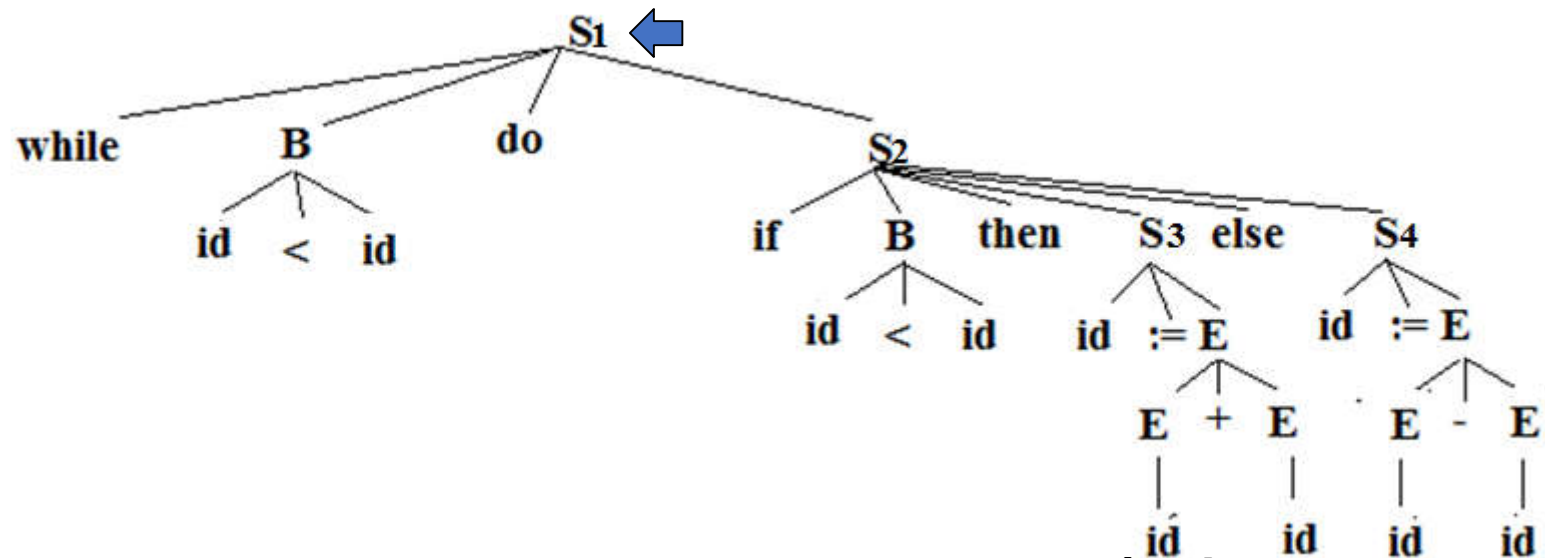
$L_1: t_2 = y + z$

$x = t_2$

goto $S_2.\text{next}$

$L_2: t_1 = y - z$

$x = t_1$



SX:

$S_1 \rightarrow \text{while } B \text{ do } S_2$

QTNN:

$S_1.\text{begin} = \text{newlabel}()$

$B.\text{True} = \text{newlabel}()$

$B.\text{False} = S_1.\text{next}$

$S_1.\text{code} =$

$S_1.\text{begin} \parallel B.\text{code}$

$\parallel B.\text{true} \parallel S_2.\text{code}$

$\parallel \text{gen}('goto' S_1.\text{begin})$

Thuộc tính

$S_1.\text{begin} = \text{newlabel}() \rightarrow L_3$

$B.\text{true} = \text{newlabel}() \rightarrow L_4$ B của lệnh while

$B.\text{False} = S_1.\text{next}$

$S_2.\text{next} = \text{begin} = L_3 \Rightarrow S_2.\text{next} = S_3.\text{next} = L_3$

$S.\text{code}$

$L_3: \text{if } a < b \text{ goto } L_4$

$\text{goto } L_0$

$L_4: \text{if } c < d \text{ goto } L_1$

$\text{goto } L_2$

$L_1: t1 = y + z$

$x = t1$

$\text{goto } L_3$

$L_2: t2 = y - z$

$x = t1$

$\text{goto } L_3$

Nhãn L_0 sẽ xuất hiện trong chương trình khi sử dụng các quy tắc ngữ nghĩa của luật $S \rightarrow S_1 S_2$