CHƯƠNG V

DỊCH TRỰC TIẾP CÚ PHÁP

Nội dung chính:
Khi viết một chương trình bằng một ngôn ngữ lập trình nào đó, ngoài việc quan tâm đến cấu trúc của chương trình (cú pháp – văn phạm), ta còn phải chú ý đến ý nghĩa của chương trình. Như vậy, khi thiết kế một trình biên dịch, ta không những chú ý đến văn phạm mà còn chú ý đến cả ngữ nghĩa. Chương 5 trình bày các cách biểu diễn ngữ nghĩa của một chương trình. Mỗi ký hiệu văn phạm kết hợp với một tập các *thuộc tính* – các thông tin. Mỗi luật sinh kết hợp với một tập các *luật ngữ nghĩa* – các quy tắc xác định trị của các thuộc tính. Việc đánh giá các luật ngữ nghĩa được sử dụng để thực hiện một công việc nào đó như tạo ra mã trung gian, lưu thông tin vào bảng ký hiệu, xuất các thông báo lỗi, v.v. Ta sẽ thấy rõ việc đánh giá này ở các chương sau: 6, 8, 9. Hai cách để kết hợp các luật sinh với các luật ngữ nghĩa được trình bày trong chương là: *Định nghĩa trực tiếp cú pháp* và *Lược đồ dịch*. Ở mức quan niệm, bằng cách sử dụng định nghĩa trực tiếp cú pháp hoặc lược đồ dịch, ta phân tích dòng thẻ từ, xây dựng cây phân tích cú pháp và duyệt cây khi cần để đánh giá các luật ngữ nghĩa tại các nút của cây.

Mục tiêu cần đạt:
Sau khi học xong chương này, sinh viên phải nắm được:

- Các cách kết hợp các luật sinh với các luật ngữ nghĩa: Định nghĩa trực tiếp cú pháp và Lược đồ dịch.
- Biết cách thiết kế chương trình – bộ dịch dự đoán - thực hiện một công việc nào đó từ một lược đồ dịch hay từ một định nghĩa trực tiếp cú pháp xác định.

Tài liệu tham khảo:

[1] **Compilers : Principles, Technique and Tools** - Alfred V.Aho, Jeffrey D.Ullman - Addison - Wesley Publishing Company, 1986.

[2] **Modern Compiler Implementation in C -** Andrew W. Appel - Cambridge University Press, 1997.

I. ĐỊNH NGHĨA TRỰC TIẾP CÚ PHÁP

Định nghĩa trực tiếp cú pháp là sự tổng quát hóa một văn phạm phi ngữ cảnh, trong đó mỗi ký hiệu văn phạm kết hợp với một tập các thuộc tính.

Cây phân tích cú pháp có trình bày giá trị các thuộc tính tại mỗi nút gọi là cây chú thích .

1. Khái niệm về định nghĩa trực tiếp cú pháp

Trong một định nghĩa trực tiếp cú pháp, mỗi luật sinh $\mathbf{A \rightarrow \alpha}$ kết hợp một tập luật ngữ nghĩa có dạng **b := f (c1, c2,..., ck)** trong đó **f** là một hàm và :

1- b là một thuộc tính tổng hợp của A và c1, c2,..., ck là các thuộc tính của các ký hiệu văn phạm của luật sinh. Hoặc

2- b là một thuộc tính kế thừa của một trong các ký hiệu văn phạm trong vế phải của luật sinh và c1, c2,..., ck là các thuộc tính của các ký hiệu văn phạm của luật sinh.

Ta nói b phụ thuộc c1, c2,..., ck.

1. Thuộc tính tổng hợp

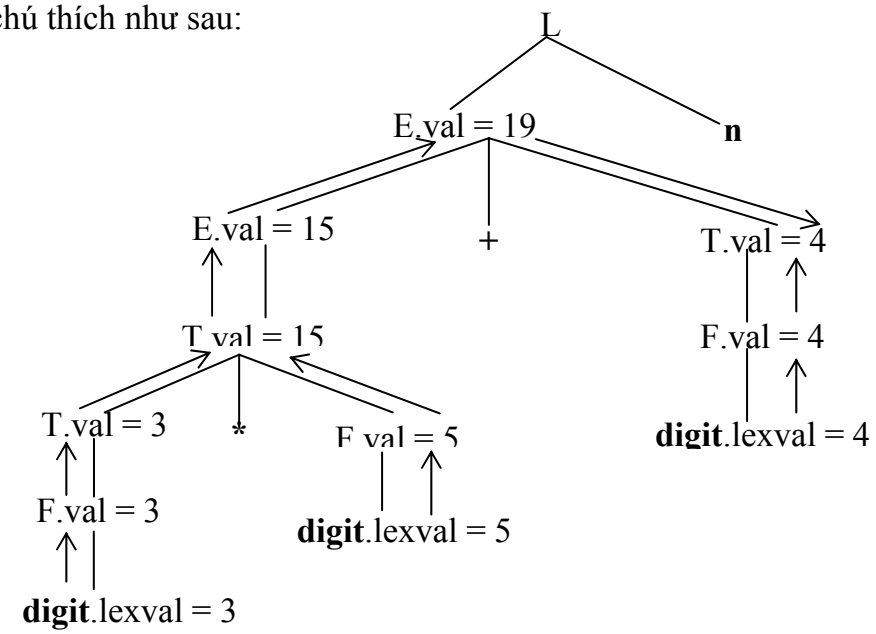
- Là thuộc tính mà giá trị của nó tại mỗi nút trên cây phân tích cú pháp được tính từ giá trị thuộc tính tại các nút con của nó.
- Định nghĩa trực tiếp cú pháp chỉ sử dụng các thuộc tính tổng hợp gọi là định nghĩa S _ thuộc tính.
- Cây phân tích cú pháp của định nghĩa S_ thuộc tính có thể được chú thích từ dưới lên trên.

Ví dụ 5.1: Xét định nghĩa trực tiếp cú pháp

Luật sinh	Luật ngữ nghĩa
L → E**n**	*print(E.val)*
E → E_1 + T	*E.val := E_1.val + T.val*
E → T	*E.val := T.val*
T → T_1 * F	*T.val := T_1.val * F.val*
T → F	*T.val := F.val*
F → **(**E**)**	*F.val := E.val*
F → **digit**	*F.val := digit.lexval*

Hình 5.1 - *Định nghĩa trực tiếp cú pháp cho một máy tính tay đơn giản*

Định nghĩa này kết hợp một thuộc tính tổng hợp có giá trị nguyên *val* với từng ký hiệu chưa kết thúc E, T và F. Token *digit* có một thuộc tính tổng hợp *lexval* với giả sử rằng giá trị của thuộc tính này được cung cấp bởi bộ phân tích từ vựng. Đây là một định nghĩa S_thuộc tính. Với biểu thức 3 * 5 + 4n (n là ký hiệu newline) có cây chú thích như sau:



Hình 5.2- *Cây chú thích cho biểu thức 3* 5+4n*

2. Thuộc tính kế thừa

- Là một thuộc tính mà giá trị của nó được xác định từ giá trị các thuộc tính của các nút cha hoặc anh em của nó.
- Nói chung ta có thể viết một định nghĩa trực tiếp cú pháp thành một định nghĩa S_ thuộc tính. Nhưng trong một số trường hợp, việc sử dụng thuộc tính kế thừa lại thuận tiện vì tính tự nhiên của nó.

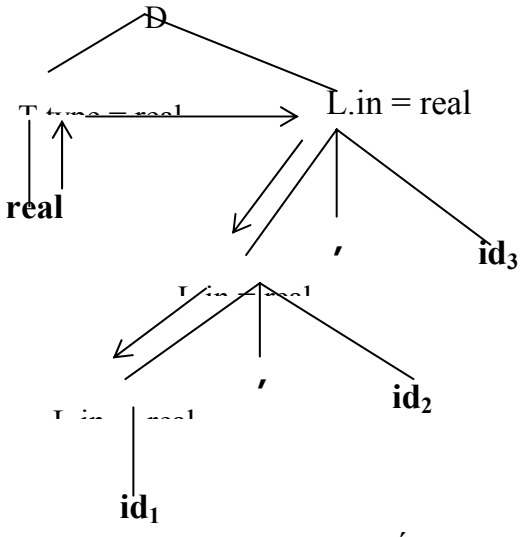
Ví dụ 5.2: Xét định nghĩa trực tiếp cú pháp sau cho sự khai báo kiểu cho biến:

Luật sinh	Luật ngữ nghĩa
D → TL	*L.in := T.type*
T → **int**	*T.type := integer*
T → **real**	*T.type := real*
L → L_1**, id**	*L_1.in := L.in; addtype (id.entry, L.in)*
L → **id**	*addtype (id.entry, L.in)*

Hình 5.3 - *Định nghĩa trực tiếp cú pháp với thuộc tính kế thừa L.in*

type là thuộc tính tổng hợp kết hợp với ký hiệu chưa kết thúc T, giá trị của nó được xác định bởi từ khóa trong khai báo. Bằng cách sử dụng thuộc tính kế thừa *in* kết hợp với ký hiệu chưa kết thúc L chúng ta xác định được kiểu cho các danh biểu và dùng thủ tục *addtype* đưa kiểu này vào trong bảng ký hiệu tương ứng với danh biểu.

Ví dụ 5.3: Xét phép khai báo: **real id1, id2, id3**. Ta có cây chú thích:



Hình 5.4- *Cây phân tích cú pháp với thuộc tính kế thừa in tại mỗi nút được gán nhãnL*

3. Đồ thị phụ thuộc

- Đồ thị phụ thuộc là một đồ thị có hướng mô tả sự phụ thuộc giữa các thuộc tính tại mỗt nút của cây phân tích cú pháp.
- Cho một cây phân tích cú pháp thì đồ thị phụ thuộc tương ứng được xây dựng theo giải thuật sau:

FOR mỗi một nút n trong cây phân tích cú pháp DO

FOR với mỗi một thuộc tính a của ký hiệu văn phạm tại n DO

Xây dựng một nút trong đồ thị phụ thuộc cho a

FOR với mỗi một nút n trên cây phân tích cú pháp DO

FOR với mỗi một luật ngữ nghĩa dạng b = f(c1, c2,..., ck) kết hợp với luật sinh được dùng tại nút n DO

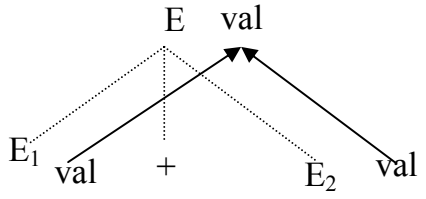
FOR i:=1 TO k DO

Xây dựng một cạnh từ nút cho ci đến nút cho b

Ví dụ 5.4: Với định nghĩa S_ thuộc tính

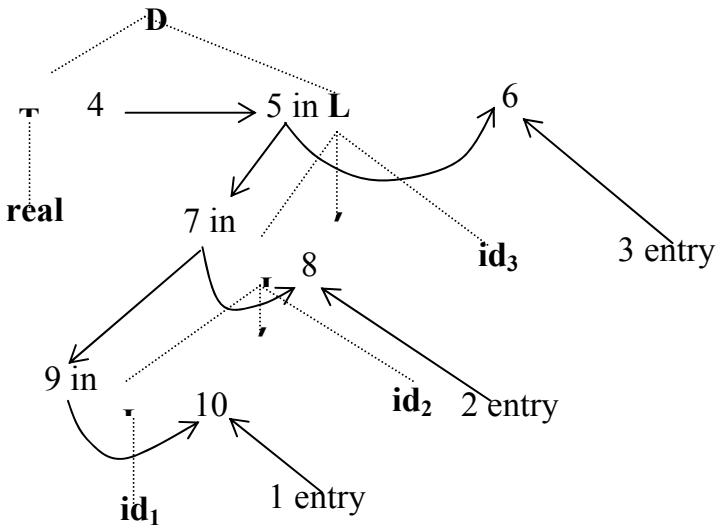
$E \rightarrow E_1 + E_2$ $\quad\quad$ *E.val := E_1.val + E_2.val*

Ta có đồ thị phụ thuộc:



Hình 5.5- *E.val được tổng hợp từ E1.val và E2.val*

Ví dụ 5.5: Dựa vào định nghĩa trực tiếp cú pháp trong ví dụ 5.2, ta có đồ thị phụ thuộc của khai báo **real id1, id2, id3**

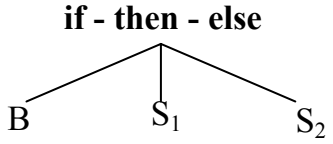


Hình 5.6- *Đồ thị phụ thuộc cho cây phân tích cú pháp trong hình 5.4*

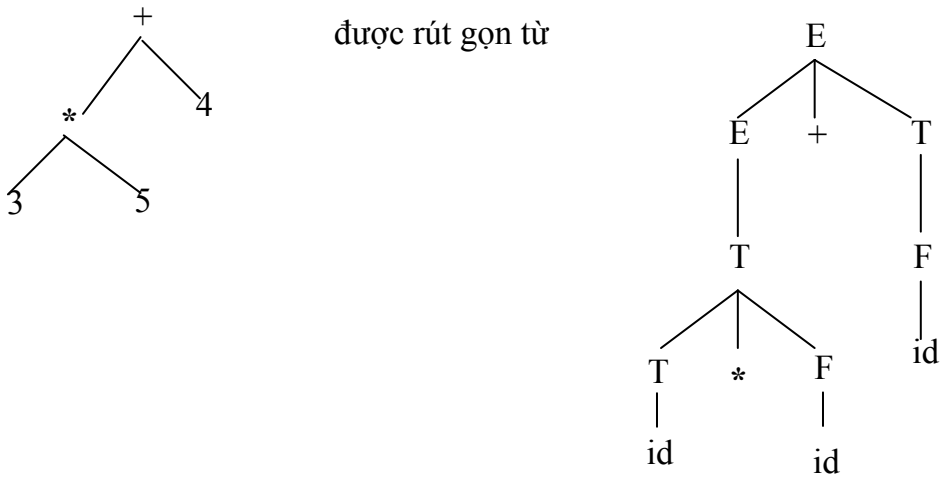
Chú ý: Với luật ngữ nghĩa dạng b = f(c1, c2, ..., ck) có chứa lời gọi thủ tục thì chúng ta tạo ra một thuộc tính tổng hợp giả. Trong ví dụ của chúng ta là nút 6, 8, 10.

II. XÂY DỰNG CÂY CÚ PHÁP

- Cây cú pháp (syntax - tree) là dạng rút gọn của cây phân tích cú pháp dùng để biểu diễn cấu trúc ngôn ngữ.
- Trong cây cú pháp các toán tử và từ khóa không phải là nút lá mà là các nút trong. Ví dụ với luật sinh S (if B then S1 else S2 được biểu diễn bởi cây cú pháp:



- Một kiểu rút gọn khác của cây cú pháp là chuỗi các luật sinh đơn được rút gọn lại. Chẳng hạn ta có:



1. Xây dựng cây cú pháp cho biểu thức

Tương tự như việc dịch một biểu thức thành dạng hậu tố.

Xây dựng cây con cho biểu thức con bằng cách tạo ra một nút cho toán hạng và toán tử.

Con của nút toán tử là gốc của cây con biểu diễn cho biểu thức con toán hạng của toán tử đó.

Mỗi một nút có thể cài đặt bằng một mẩu tin có nhiều trường.

Trong nút toán tử, có một trường chỉ toán tử như là nhãn của nút, các trường còn lại chứa con trỏ, trỏ tới các nút toán hạng.

Để xây dựng cây cú pháp cho biểu thức chúng ta sử dụng các hàm sau đây:

1. **mknode(op, left, right)**: Tạo một nút toán tử có nhãn là op và hai trường chứa con trỏ, trỏ tới con trái left và con phải right.

2. **mkleaf(id, entry)**: Tạo một nút lá với nhãn là id và một trường chứa con trỏ entry, trỏ tới ô trong bảng ký hiệu.

3. **mkleaf(num,val)**: Tạo một nút lá với nhãn là num và trường val, giá trị của số.

Ví dụ 5.6: Để xây dựng cây cú pháp cho biểu thức: **a - 4 + c** ta dùng một dãy các lời gọi các hàm nói trên.

(1): p1 := mkleaf(id, entrya)

(2): p2 := mkleaf(num,4)

(3): p3 := mknode('-', p1, p2)

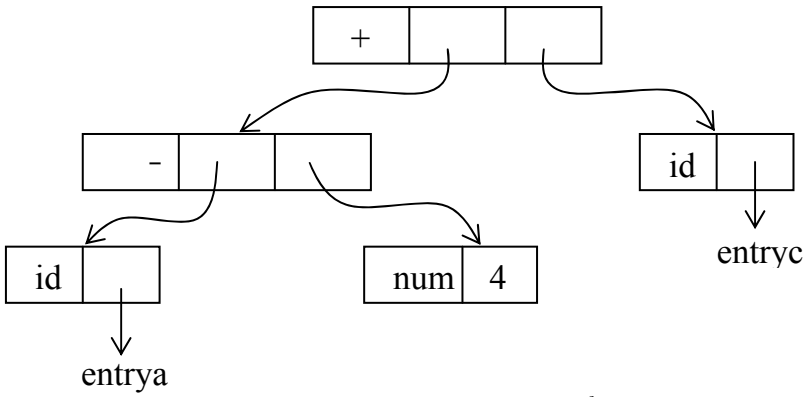
(4): p4 := mkleaf(id, entryc)

(5): p5 := mknode('+', p3, p4)

Cây được xây dựng từ dưới lên

entrya là con trỏ, trỏ tới ô của a trong bảng ký hiệu

entryc là con trỏ, trỏ tới ô của c trong bảng ký hiệu



Hình 5.7- *Cây cú pháp cho biểu thức a - 4 + c*

2. Xây dựng cây cú pháp từ định nghĩa trực tiếp cú pháp

Căn cứ vào các luật sinh văn phạm và luật ngữ nghĩa kết hợp mà ta phân bổ việc gọi các hàm mknode và mkleaf để tạo ra cây cú pháp.

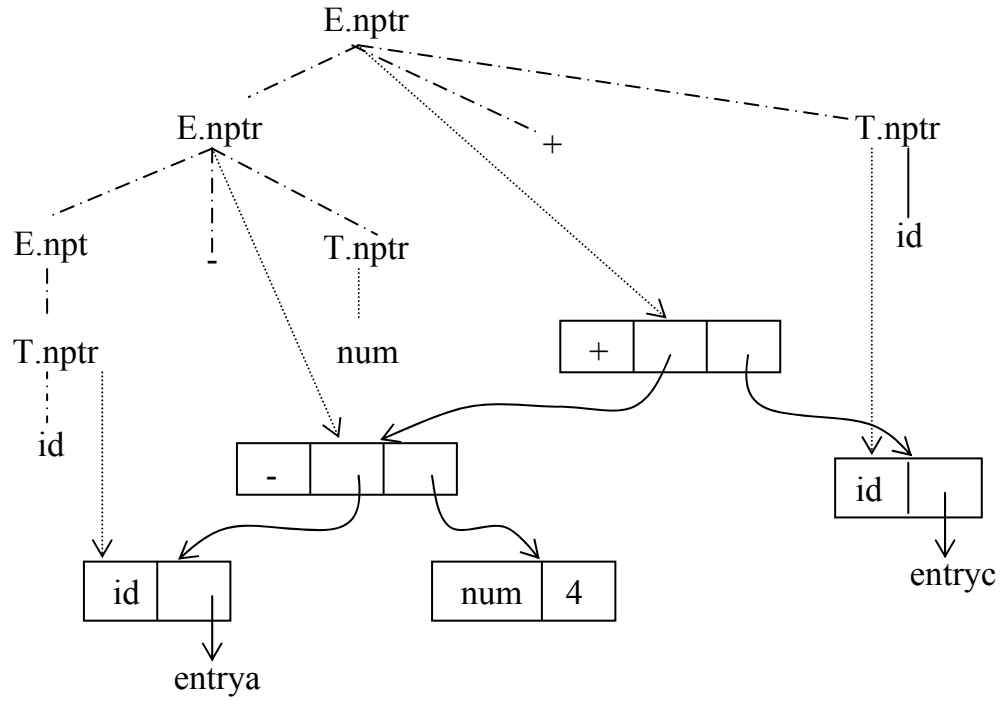
Ví dụ 5.7: Định nghĩa trực tiếp cú pháp giúp việc xây dựng cây cú pháp cho biểu thức là:

Luật sinh	Luật ngữ nghĩa
$E \rightarrow E_1 + T$	*E.nptr := mknode('+', E_1.nptr, T.nptr)*
$E \rightarrow E_1 - T$	*E.nptr := mknode('-', E_1.nptr, T.nptr)*
$E \rightarrow T$	*E.nptr := T.nptr*
$T \rightarrow$ **(**E**)**	*T.nptr := E.nptr*
$T \rightarrow$ **id**	*T. nptr := mkleaf(id, id.entry)*
$T \rightarrow$ **num**	*T.nptr := mkleaf(num, num.val)*

Hình 5.8 - *Định nghĩa trực tiếp cú pháp để tạo cây cú pháp cho biểu thức*

Các nút trên cây phân tích cú pháp có nhãn là các ký hiệu chưa kết thúc E và T sử dụng thuộc tính tổng hợp nptr để lưu con trỏ trỏ tới một nút trên cây cú pháp.

Với biểu thức a - 4 + c ta có cây phân tích cú pháp (biểu diễn bởi đường chấm) tronh hHình 5.9.



Hình 5.9 - *Xây dựng cây cú pháp cho a - 4 + c*

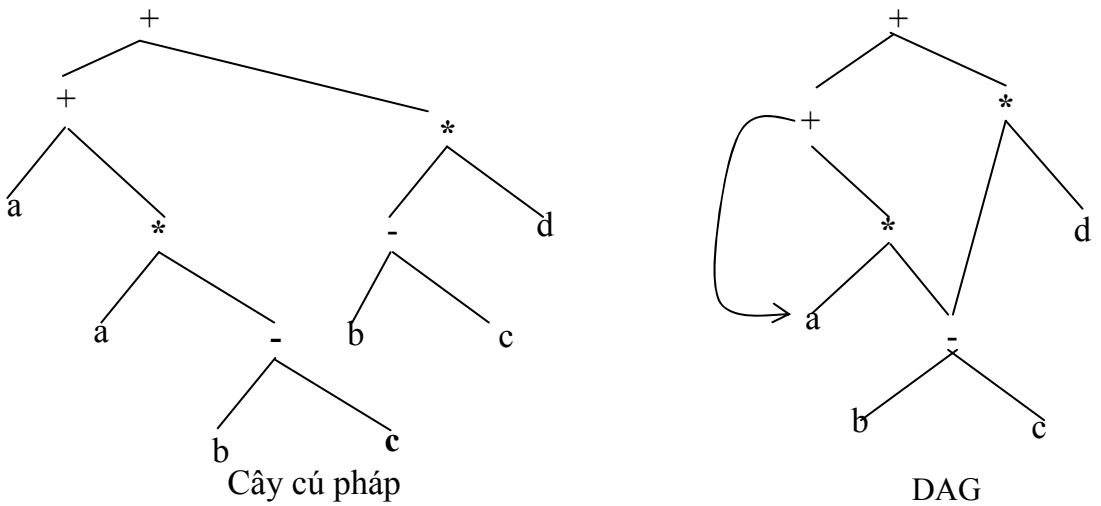
Luật ngữ nghĩa cho phép tạo ra cây cú pháp.

Cây cú pháp có ý nghĩa về mặt cài đặt còn cây phân tích cú pháp chỉ có ý nghĩa về mặt logic.

3. Đồ thị có hướng không tuần hoàn cho biểu thức (Directed Acyclic Graph - DAG)

DAG cũng giống như cây cú pháp, tuy nhiên trong cây cú pháp các biểu thức con giống nhau được biểu diễn lặp lại còn trong DAG thì không. Trong DAG, một nút con có thể có nhiều "cha".

Ví dụ 5.8: Cho biểu thức **a + a * (b - c) + (b - c) * d**. Ta có cây cú pháp và DAG:



Hình 5.10 - *Cây cú pháp và DAG của một biểu thức*

Để xây dựng một DAG, trước khi tạo một nút phải kiểm tra xem nút đó đã tồn tại chưa, nếu đã tồn tại thì hàm tạo nút (mknode, mkleaf) trả về con trỏ của nút đã tồn tại, nếu chưa thì tạo nút mới.

Cài đặt DAG:

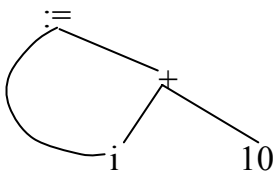
Người ta thường sử dụng một mảng mẩu tin , mỗi mẩu tin là một nút. Ta có thể tham khảo tới nút bằng chỉ số của mảng.

Ví dụ 5.9:

Lệnh gán

i := i + 10

DAG



Biểu diễn

1	id	entry i	
2	num	10	
3	+	1	2
4	:=	1	3

Hình 5.11- *Minh họa cài đặt DAG*

Nút 1: có nhãn là id, con trỏ trỏ tới entry i.

Nút 2: có nhãn là num, giá trị là 10.

Nút 3: có nhãn là +, con trái là nút 1, con phải là nút 2.

Nút 4: có nhãn là :=, con trái là nút 1, con phải là nút 3.

Giải thuật: Phương pháp số giá trị (value – number) để xây dựng một nút trong DAG.

Giả sử rằng các nút được lưu trong một mảng và mỗi nút được tham khảo bởi số giá trị của nó. Mỗi một nút toán tử là một bộ ba <op, l, r >

Input: Nhãn op, nút l và nút r.

Output: Một nút với <op, l, r>

Phương pháp: Tìm trong mảng một nút m có nhãn là op con trái là l, con phải là r. Nếu tìm thấy thì trả về m, ngược lại tạo ra một nút mới n, có nhãn là op, con trái là l, con phải là r và trả về n.

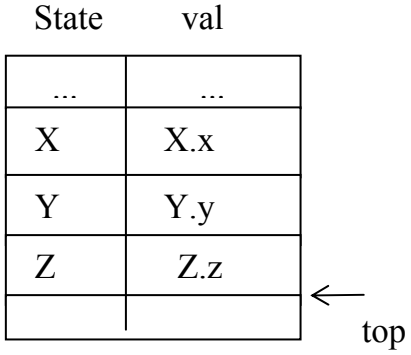
III. ĐÁNH GIÁ DƯỚI LÊN ĐỐI VỚI ĐỊNH NGHĨA S_THUỘC TÍNH

1. Sử dụng Stack

Như đã biết, định nghĩa S_ thuộc tính chỉ chứa các thuộc tính tổng hợp do đó phương pháp phân tích dưới lên là phù hợp với định nghĩa trực tiếp cú pháp này. Phương pháp phân tích dưới lên sử dụng một STACK để lưu trữ thông tin về cây con đã được phân tích. Chúng ta có thể mở rộng STACK này để lưu trữ giá trị thuộc tính tổng hợp. STACK được cài đặt bởi một cặp mảng state và val.

Giả sử luật ngữ nghĩa A.a := f (X.x, Y.y, Z.z) kết hợp với luật sinh $A \rightarrow XYZ$. Trước khi XYZ được rút gọn thành A thì val[top] = Z.z, val[top - 1] = Y.y, val[top - 2]

= X.x. Sau khi rút gọn, top bị giảm 2 đơn vị, A nằm trong state[top] và thuộc tính tổng hợp nằm trong val[top].



Mỗi ô trong stack là một con trỏ trỏ tới bảng phân tích LR(1). Nếu phần tử thứ I của stack là ký hiệu A thì val[i] là giá trị thuộc tính kết hợp với A.

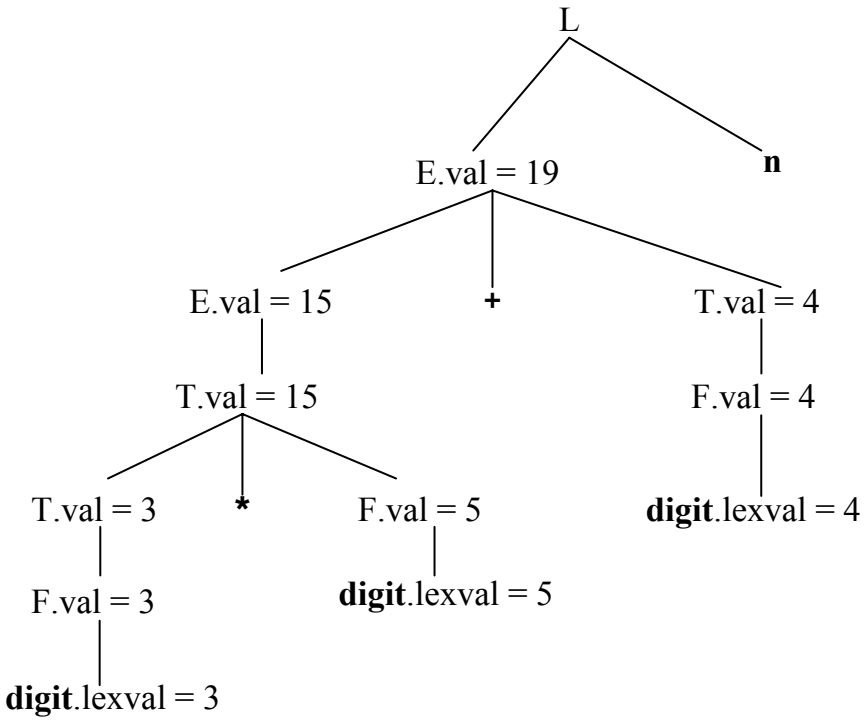
Hình 5.12 - *Stack phân tích cú pháp vào một trường lưu giữ thuộc tính tổng hợp*

2. Ví dụ

Ví dụ 5.10: Xét định nghĩa trực tiếp cú pháp:

Luật sinh	Luật ngữ nghĩa
L → E**n**	*print(E.val)*
E → E_1 + T	*E.val := E_1.val + T.val*
E → T	*E.val := T.val*
T → T_1 * F	*T.val := T_1.val * F.val*
T → F	*T.val := F.val*
F → **(**E**)**	*F.val := E.val*
F → **digit**	*F.val := digit.lexval*

Với biểu thức 3 * 5 + 4 n ta có cây chú thích:



Hình 5.13 – *Cây chú thích cho biểu thức 3 * 5 + 4 n*

Cây chú thích này có thể được đánh giá bằng một bộ phân tích cú pháp LR từ dưới lên trên. Chú ý rằng bộ phân tích đã nhận biết giá trị thuộc tính digit.lexval. Khi digit được đưa vào stack thì token digit được đưa vào state[top] và giá trị thuộc tính của nó được đưa vào val[top]. Chúng ta có thể sử dụng kỹ thuật trong mục VI của chương IV để xây dựng bộ phân tích LR. Để đánh giá các thuộc tính chúng ta thay đổi bộ phân tích cú pháp để thực hiện đoạn mã sau:

Luật sinh	Luật ngữ nghĩa
L → E**n**	*print(val[top])*
E → E_1 + T	*val[ntop] := val[top - 2] + val[top]*
E → T	
T → T_1 * F	*val[ntop] := val[top - 2] * val[top]*
T → F	
F → **(**E**)**	*val[ntop] := val[top - 1]*
F → **digit**	

Hình 5.14- *Cài đặt một máy tính tay sử dụng bộ phân tích cú pháp LR*

Khi một luật sinh với r ký hiệu bên vế phải được rút gọn thì ntop = top - r + 1. Sau khi một đoạn mã thực hiện thì đặt top = ntop

Bảng sau trình bày quá trình thực hiện của bộ phân tích cú pháp

Input	State	Val	Luật sinh được dùng
3 * 5 + 4 n	_	_	
* 5 + 4 n	3	3	
* 5 + 4 n	F	3	*F → digit*
*5 + 4 n	T	3	*T → F*
5 + 4 n	T*	3 -	
+ 4 n	T * 5	3 - 5	
+ 4 n	T * F	3 - 5	*F → digit*
+ 4 n	T	15	*T → T * F*
+ 4 n	E	15	*E → T*
4 n	E +	15 -	
n	E + 4	15 - 4	
n	E + F	15 - 4	*F → digit*
n	E + T	15 - 4	*T → F*

n	E	19	$E \rightarrow E + T$
	E n	19 -	
	L	19	$L \rightarrow En$

Hình 5.15- *Các phép chuyển được tạo ra bởi bộ thông dịch trên chuỗi nhập 3* 5+4n*

IV. ĐỊNH NGHĨA L_THUỘC TÍNH

1. Định nghĩa L_thuộc tính.

Mỗi định nghĩa trực tiếp cú pháp là một định nghĩa L thuộc tính nếu mỗi một thuộc tính kế thừa của Xi (1 <= i <= n) trong vế phải của luật sinh A → X1X2...Xn phụ thuộc chỉ vào:

1. Các thuộc tính của X1, X2,..., Xi - 1

2. Các thuộc tính kế thừa của A.

Ví dụ 5.11: Cho định nghĩa trực tiếp cú pháp:

Luật sinh	Luật ngữ nghĩa
A → L M	*L.i := l(A.i)*
	M.i := m(L.s) *A.s := f(M.s)*
	R.i := r(A.i)
A → Q R	*Q.i := q(R.s)* *A.s := f(Q.r)*

Hình 5.16 - *Định nghĩa trực tiếp cú pháp không phải là định nghĩa L_thuộc tính*

Đây không phải là một định nghĩa L_thuộc tính vì thuộc tính kế thừa Q.i phụ thuộc vào thuộc tính R.s của ký hiệu bên phải nó trong luật sinh.

2. Lược đồ dịch

Lược đồ dịch là một văn phạm phi ngữ cảnh trong đó các thuộc tính được kết hợp với các ký hiệu văn phạm và các hành vi ngữ nghĩa nằm trong cặp dấu { } được xen vào bên phải của luật sinh.

Ví dụ 5.12: Lược đồ dịch một biểu thức trung tố với phép cộng và trừ thành dạng hậu tố:

E → T R

R → **addop** T { *print (addop.lexeme)* } R_1 | ε

T → **num** { *print (num.val)* }

Với biểu thức 9 - 5 + 2 ta có cây phân tích cú pháp (hình 5.16)

Để xây dựng một lược đồ dịch, chúng ta xét hai trường hợp sau đây:

Trường hợp 1: Chỉ chứa thuộc tính tổng hợp:

Với mỗi một luật ngữ nghĩa, ta tạo một hành vi ngữ nghĩa và đặt hành vi này vào cuối vế phải luật sinh.

Ví dụ 5.13:

Luật sinh	Luật ngữ nghĩa
$T \rightarrow T_1 * F$	$T.val := T_1.val * F.val$

Ta có lược đồ dịch:

$T \rightarrow T_1 * F$ { $T.val := T_1.val * F.val$}

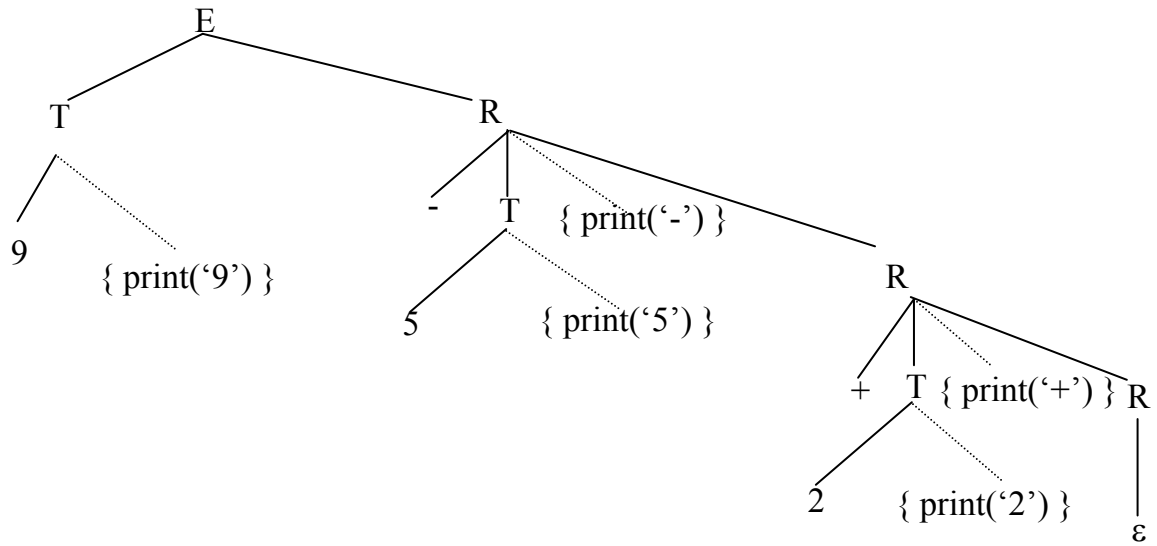
Trường hợp 2: Có cả thuộc tính tổng hợp và kế thừa phải thỏa mãn 3 yêu cầu sau đây:

1. Thuộc tính kế thừa của một ký hiệu trong vế phải của luật sinh phải được xác định trong hành vi nằm trước ký hiệu đó.

2. Một hành vi không được tham khảo tới thuộc tính tổng hợp của một ký hiệu nằm bên phải hành vi đó.

3. Thuộc tính tổng hợp của ký hiệu chưa kết thúc trong vế trái chỉ có thể được xác định sau khi tất cả các thuộc tính mà nó tham khảo đã được xác định. Hành vi xác định các thuộc tính này luôn đặt cuối vế phải của luật sinh.

Với một định nghĩa trực tiếp cú pháp L_thuộc tính ta có thể xây dựng lược đồ dịch thỏa mãn 3 yêu cầu nói trên.



Hình 5.17 - *Cây phân tích cú pháp của các hoạt động biểu diễn 9-5+2*

Ví dụ 5.14: Bộ xử lý các công thức toán học – EQN - có thể xây dựng các biểu thức toán học từ các toán tử sub (subscripts) và sup (superscripts). Chẳng hạn:

input	**output**
BOX **sub** box	BOX_{box}
a **sub** {i **sup** 2 }	a_{i^2}

Để xác định chiều rộng và chiều cao của các hộp ta có định nghĩa L_thuộc tính như sau:

Luật sinh	Luật ngữ nghĩa
S → B	*B.ps := 10*
	S.ht := B.ht
B → B_1B_2	*B_1.ps := B.ps*
	B_2.ps := B.ps
	B.ht := max(B_1.ht, B_2.ht)
B → B_1 **sub** B_2	*B_1.ps := B.ps*
	B_2.ps := shrink(B.ps)
	B.ht := disp(B_1.ht, B_2.ht)
B → **text**	*B.ht := text.h * B.ps*

Hình 5.18 - *Định nghĩa trực tiếp cú pháp xác định kích thước và chiều cao của các hộp*

Trong đó:

- Ký hiệu chưa kết thúc B biểu diễn một công thức.
- Luật sinh B → BB biểu diễn sự kề nhau của hai hộp.
- Luật sinh B → B sub B biểu diễn sự sắp đặt, trong đó hộp chỉ số thứ 2 có kích thước nhỏ hơn, nằm thấp hơn hộp thứ nhất.
- Thuộc tính kế thừa ps (point size - kích thước điểm) phản ánh độ lớn của công thức.
- Luật sinh B → text ứng với luật ngữ nghiã B.ht:= text.h * B.ps lấy chiều cao thực của text (h) nhân với kích thước điểm của B để có được chiều cao của hộp.
- Luật sinh B → B1B2 được áp dụng thì B1, B2 kế thừa kích thước điểm của B bằng luật copy. Độ cao của B bằng giá trị lớn nhất trong độ cao của B1, B_2.
- Khi luật sinh B → B1 sub B2 được áp dụng thì hàm shrink sẽ giảm kích thước điểm của B2 còn 30% và hàm disp đẩy hộp B2 xuống.

Đây là một định nghĩa L_thuộc tính vì chỉ có duy nhất một thuộc tính kế thừa ps và thuộc tính này chỉ phụ thuộc vào vế trái của luật sinh.

Dựa vào 3 yêu cầu nói trên, ta xen các hành vi ngữ nghĩa tương ứng với luật ngữ nghĩa vào vế phải của mỗi luật sinh để được lược đồ dịch.

S → {*B.ps := 10* }

B {*S.ht := B.ht* }

B → { *B_1.ps := B.ps* }

B_1 {*B_2.ps := B.ps* }

B_2 {*B.ht := max(B_1.ht, B_2.ht)* }

B → B_1 sub B_2 {$B_1.ps := B.ps$ } {$B_2.ps := shrink(B.ps)$ } {$B.ht := disp(B_1.ht, B_2.ht)$ }

B → text {$B.ht := text.h * B.ps$ }

Hình 5.19 - *Lược đồ dịch được tạo ra từ hình 5.18*

Chú ý: Để dễ đọc mỗi ký hiệu văn phạm trong luật sinh được viết trên một dòng và hành vi được viết vào bên phải.

Chẳng hạn:

S → {$B.ps := 10$ } B {$S.ht := B.ht$ }

Được viết thành S → {B.ps := 10 }
B {$S.ht := B.ht$ }

V. DỊCH TRÊN XUỐNG

1. Loại bỏ đệ qui trái

Vấn đề loại bỏ đệ qui trái của một văn phạm đã được trình bày trong mục III của chương IV. Ở đây chúng ta giải quyết vấn đề chuyển một lược đồ dịch của văn phạm đệ quy trái thành một lược đồ dịch mới không còn đệ quy.

Giả sử, ta có lược đồ dịch dạng

A → A_1 Y {$A.a := g(A_1.a, Y.y)$ }

A → X {$A.a := f(X.x)$ }

Đây là một văn phạm đệ quy trái, áp dụng giải thuật khử đệ qui trái ta được văn phạm không đệ quy trái

A → X R

R → Y R | ε

Bổ sung hành vi ngữ nghĩa cho văn phạm ta được lược đồ dịch:

A → X { $R.i := f(X.x)$ }
R {$A.a := R.s$ }

R → Y {$R_1.i := g(R.i, Y.y)$ }
R_1 {$R.s := R_1.s$ }

R → ε {$R.s := R.i$ }

Ví dụ 5.15: Xét lược đồ dịch của văn phạm đệ quy trái cho biểu thức.

E → E_1 + T {$E.val := E_1.val + T.val$ }

E → E_1 - T {$E.val := E_1.val - T.val$ }

E → T {$E.val := T.val$ }

T → **(**E**)** {*T.val := E.val* }

T → **num** {*T.val := num.val* }

Hình 5.20 - *Lược đồ dịch của một văn phạm đệ quy trái*

Vận dụng ý kiến trên ta khử đệ quy trái để được lược đồ dịch không đệ quy trái

E → T {*R.i := T.val* }

R {*E.val := R.s* }

R → +

T {R_1*.i := R.i + T.val* }

R_1 {*R.s :=* R_1*.s* }

R → -

T {R_1*.i := R.i - T.val* }

R_1 {*R.s :=* R_1*.s* }

R → ε {*R.s := R.i* }

T → **(**

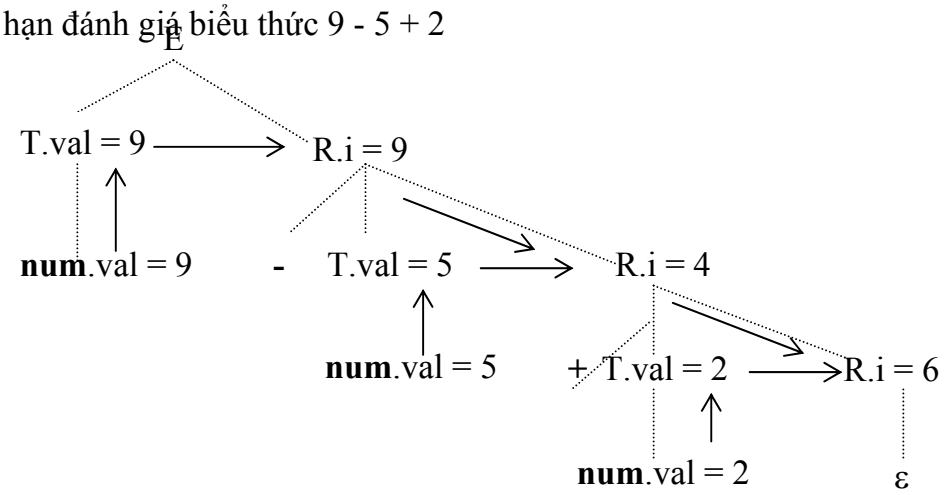
E

) {*T.val := E.val* }

T → **num** { *T.val := num.val* }

Hình 5.21 - *Lược đồ dịch đã được chuyển đổi có văn phạm đệ quy phải*

Chẳng hạn đánh giá biểu thức 9 - 5 + 2



Hình 5.22 - *Xác định giá trị của biểu thức 9-5+2*

Ví du 5.16: Xét lược đồ dịch xây dựng cây cú pháp cho biểu thức

E → E_1 + T {*E.nptr := mknode('+',* E_1*.nptr, T.nptr)* }

E → E_1 - T {*E.nptr := mknode('-',* E_1*.nptr, T.nptr)* }

E → T {*E.nptr := T.nptr* }

T → **(**E**)** {*T.nptr := E.nptr* }

T → **id** {*T.nptr := mkleaf(id, id.entry)* }

T → **num** {*T.nptr := mkleaf(num, num.val)* }

Áp dụng quy tắc khử đệ quy trái trên với E ≈ A, +T, -T ≈ Y và T ≈ X ta có lược đồ dịch

E → T {*R.i := T.nptr* }
R {*E.nptr := R.s* }

R → +
T {R_1*.i := mknode('+', R.nptr, T.nptr)* }
R_1 {*R.s :=* R_1*.s* }

R → -
T {R_1*.i := mknode('-', R.nptr, T.nptr)* }
R_1 {*R.s :=* R_1*.s* }

R → ε {*R.s := R.i* }

T → **(**
E
) {*T.nptr := E.nptr* }

T → **id** {*T.nptr := mkleaf(id, id.entry)* }

T → **num** { *T.nptr := mkleaf(num, num.val)* }

Hình 5.23 - *Lược đồ dịch được chuyển đổi để xây dựng cây cú pháp*

2. Thiết kế bộ dịch dự đoán

Giải thuật: Xây dựng bộ dịch trực tiếp cú pháp dự đoán (Predictive - Syntax - Directed Translation)

Input: Một lược đồ dịch cú pháp trực tiếp với văn phạm có thể phân tích dự đoán.

Output: Mã cho bộ dịch trực tiếp cú pháp.

Phương pháp:

1. Với mỗi ký hiệu chưa kết thúc A, xây dựng một hàm có các tham số hình thức tương ứng với các thuộc tính kế thừa của A và trả về giá trị của thuộc tính tổng hợp của A.

2. Mã cho ký hiệu chưa kết thúc A quyết định luật sinh nào được dùng cho ký hiệu nhập hiện hành.

3. Mã kết hợp với mỗi luật sinh như sau: chúng ta xem xét token, ký hiệu chưa kết thúc và hành vi bên phải của luật sinh từ trái sang phải

i) Đối với token X với thuộc tính tổng hợp x, lưu giá trị của x vào trong biến được khai báo cho X.x. Sau đó phát sinh lời gọi để hợp thức (match) token X và dịch chuyển đầu đọc.

ii) Đối với ký hiệu chưa kết thúc B, phát sinh lệnh gán C := B(b1, b2, ..., bk) với lời gọi hàm trong vế phải của lệnh gán, trong đó b1, b2,..., bk là các biến cho các thuộc tính kế thừa của B và C là biến cho thuộc tính tổng hợp của B.

iii) Đối với một hành vi, chép mã vào trong bộ phân tích cú pháp, thay thế mỗi tham chiếu tới một thuộc tính bởi biến cho thuộc tính đó.

Ví dụ 5.17: Xét lược đồ dịch cho việc xây dựng cây cú pháp cho biểu thức. Ta thấy đó là một văn phạm LL(1) nên phù hợp cho phân tích trên xuống. Với 3 ký hiệu chưa kết thúc E, R, T ta xây dựng 3 hàm tương ứng:

function E: ↑ syntax - tree - node; /* E không có thuộc tính kế thừa */

function R (i : ↑ syntax - tree - node) : ↑ syntax - tree - node

function T : ↑ syntax - tree - node;

Dùng token addop biểu diễn cho + và - ta có thể kết hợp hai luật sinh thành một luật sinh mới.

R → addop
T {$R_1.i := mknode(addop.lexeme, R.i, T.nptr)$ }
R_1 {$R.s := R_1.s$ }

R → ε {$R.s := R.i$ }

Ta có hàm R như sau:

```
function R(i : ↑syntax_ tree_node) : ↑syntax_tree_node;
var   nptr, i1, s1, s : ↑syntax_tree_node;
      addoplexeme : char;
begin
  if  lookahead = addop  then
    begin  /*  luật sinh R → addop TR */
          addoplexeme := lexval;
          match(addop);
          nptr := T;
          i1 := mknode(addoplexeme, i, nptr);
          s1 := R(i1);
          s := s1;
    end
  else   s := i;   /* Luật sinh R → ε */
  return  s;
end;
```

Hình 5.24 - *Xây dựng cây cú pháp đệ quy giảm*

BÀI TẬP CHƯƠNG V

5.1. Xây dựng một cây phân tích cú pháp chú thích cho biểu thức số học sau:

(4 * 7+ 1) * 2

5.2. Xây dựng một cây phân tích cú pháp và cây cú pháp cho biểu thức ((a) + (b)) theo:

a) Định nghĩa trực tiếp cú pháp cho biểu thức số học.

b) Lược đồ dịch cho biểu thức số học.

5.3. Xây dựng một DAG cho các biểu thức sau đây:

a) **a + a + (a+ a + a + (a+ a+ a+ a))**

b) **x *(3 *x + x * x)**

5.4. Văn phạm sau đây sinh ra các biểu thức có được khi áp dụng một toán tử số học + cho các hằng số nguyên và số thực. Khi 2 số nguyên được công lại, kiểu kết quả là kiểu nguyên, ngược lại nó là kiểu số thực.

$E \rightarrow E + T \mid T$

$T \rightarrow num \bullet num \mid num$

a) Đưa ra một định nghĩa trực tiếp cú pháp xác định kiểu của mỗi biểu thức con.

b) Mở rộng định nghĩa trực tiếp cú pháp trên để dịch các biểu thức thành ký pháp hậu tố và xác định các kiểu. Dùng toán tử một ngôn inttoreal để chuyển một giá trị nguyên thành giá trị thực tương đương mà nhờ đó cả hai toán hạng của + ở dạng hậu tố có cùng kiểu.

5.5. Giả sử các khai báo sinh bởi văn phạm sau:

$D \rightarrow$ **id** L

L → **,** **id** L | **:** T

T → **integer** | **real**

a) Xây dựng một lược đồ dịch để nhập kiểu của mỗi định danh vào bảng danh biểu.

b) Xây dựng chương trình dịch dự đoán từ lược đồ dịch trên.

5.6. Cho văn phạm sinh ra các dòng text như sau:

$S \rightarrow L$

$L \rightarrow L\ B \mid B$

$B \rightarrow B$ **sub** $F \mid F$

$F \rightarrow$ **{** L **}** $\mid$ **text**

a) Xây dựng một định nghĩa trực tiếp cú pháp cho văn phạm.

b) Chuyển định nghĩa trực tiếp cú pháp trên thành lược đồ dịch.

c) Loại bỏ đệ quy trái trong lược đồ dịch vừa xây dựng.