CHƯƠNG 5
BIÊN DỊCH TRỰC TIẾP CÚ PHÁP

Có hai khái niệm về các luật ngữ nghĩa có liên quan đến luật sinh: định nghĩa trực tiếp cú pháp và lược đồ dịch.

- *Định nghĩa trực tiếp cú pháp.*
- *Lược đồ dịch.*

Chuỗi nhập → cây phân tích → đồ thị phụ thuộc →
→ đánh giá thứ tự các luật ngữ nghĩa.

Hình 5.0. Khái niệm về dịch trực tiếp cú pháp

Khái niệm tổng quan của biên dịch trực tiếp cú pháp

5.1. Định nghĩa trực tiếp cú pháp

Là văn phạm phi ngữ cảnh mà trong đó mỗi ký hiệu văn phạm có tập thuộc tính. Tập thuộc tính này có hai loại: thuộc tính tổng hợp và thuộc tính kế thừa.

Cây cú pháp có giá trị thuộc tính ở mỗi nút được gọi là cây phân tích chú thích.

Dạng của định nghĩa trực tiếp cú pháp

Mỗi luật sinh có dạng $A \rightarrow \alpha$ đều có một tập luật ngữ nghĩa có dạng $b := f(c_1, c_2, ..., c_k)$ với f là hàm số và:

1. b là thuộc tính tổng hợp của A và $c_1, c_2, ..., c_k$ là các thuộc tính của ký hiệu văn phạm của luật sinh, hoặc
2. b là thuộc tính kế thừa của một trong các ký hiệu văn phạm bên vế phải của luật sinh và $c_1, c_2, ..., c_k$ là các thuộc tính của các ký hiệu văn phạm của luật sinh.

Thí dụ 5.1. Định nghĩa trực tiếp cú pháp ở bảng 5.1.

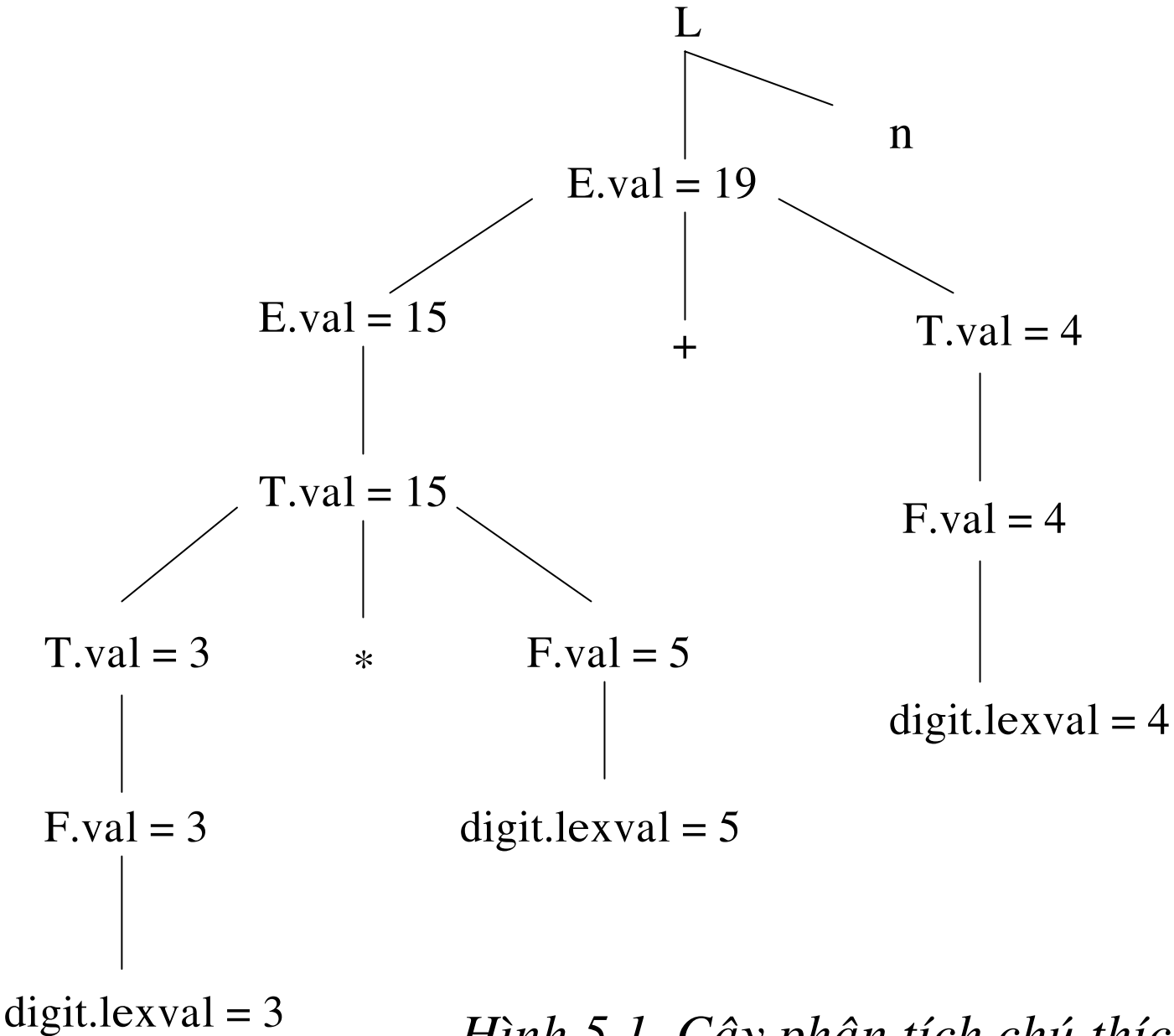
Bảng 5.1. *Định nghĩa trực tiếp cú pháp cho bảng tính đơn giản*

Luật sinh	Luật ngữ nghĩa
$L \rightarrow En$	Print (E.val)
$E \rightarrow E_1 + T$	E.val: = E1.val + T.val
$E \rightarrow TE.val: = T.val$	E.val: = T.val
$T \rightarrow T_1 * F$	T.val: = T.val x F.val
$T \rightarrow FT.val: = F.val$	T.val: = F.val
$F \rightarrow (E)$	F.val: = E.val
$F \rightarrow digit$	F.val: = digit . lexval

Thuộc tính tổng hợp

Định nghĩa trực tiếp cú pháp dùng các thuộc tính tổng hợp gọi là định nghĩa thuộc tính S. Thuộc tính S của một nút có thể được từ các thuộc tính ở mỗi nút từ dưới lên.

Thí dụ 5.2. Định nghĩa thuộc tính S ở thí dụ 5.1



*Hình 5.1. Cây phân tích chú thích 3 * 5 + 4n*

Thuộc tính kế thừa

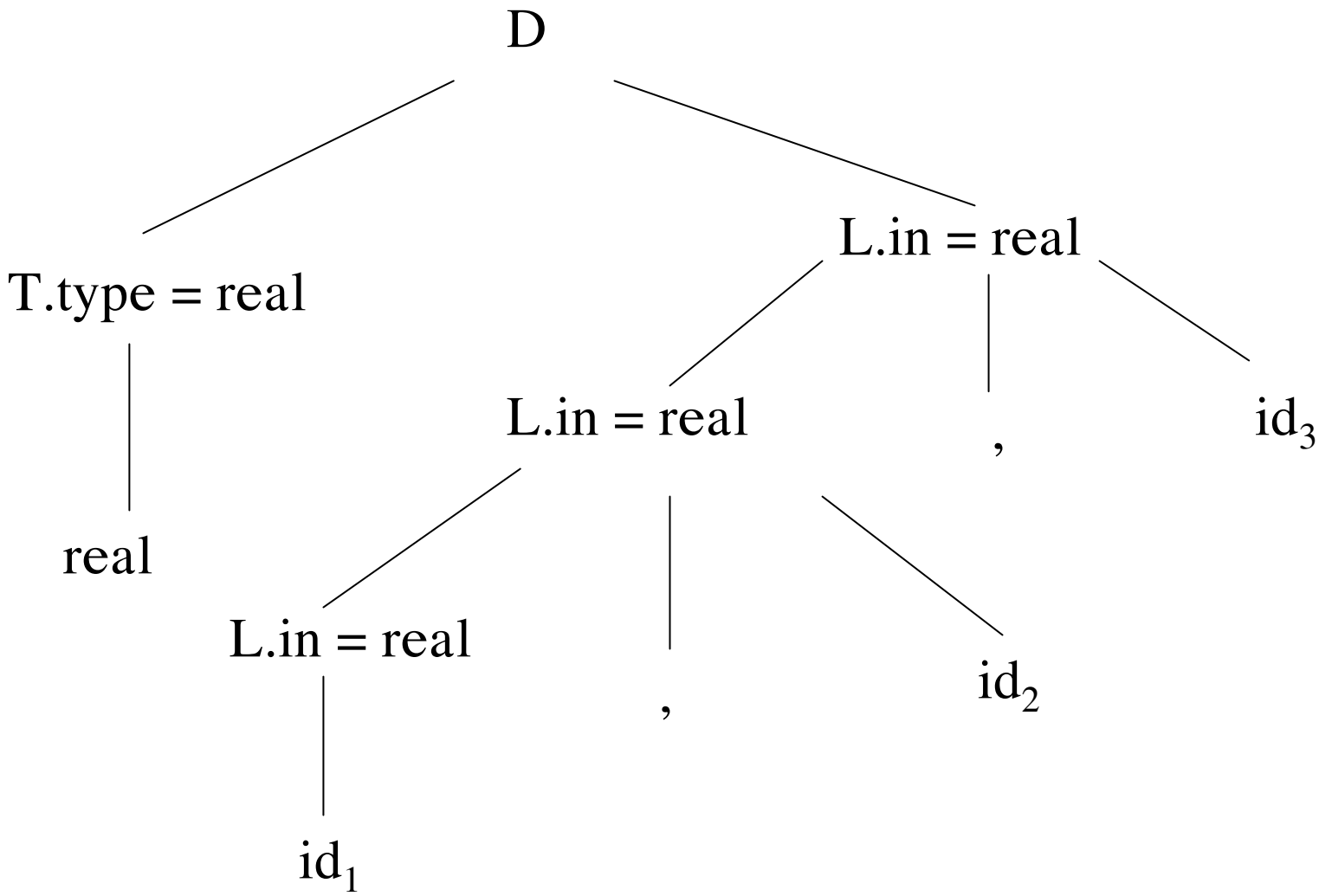
Thuộc tính kế thừa là thuộc tính mà giá trị của nó ở một nút trên cây phân tích được xác định bởi thuộc tính cha mẹ và/hoặc anh chị của nút đó.

Thí dụ 5.3. Sự khai báo được tạo bởi ký hiệu không kết thúc D trong định nghĩa trực tiếp cú pháp ở (bảng 5.2).

Bảng 5.2. *Định nghĩa trực tiếp cú pháp với thuộc tính kế thừa L.in.*

Luật sinh	Luật ngữ nghĩa
$D \rightarrow TL$	L.in: = T.type
$T \rightarrow$ int	T.type: = integer
$T \rightarrow$ real	T.type: = real
$L \rightarrow L_1$, id	L_1.in: = L.in
$L \rightarrow$ id	Addtype (id.entry, L.in)

Hình 5.2 là cây phân tích chú thích cho câu real id1, id2, id3.



Hình 5.2. Cây phân tích với thuộc tính kế thừa in ở mỗi nút có nhãn L.

Đồ thị phụ thuộc

Các sự phụ thuộc trung gian: thuộc tính kế thừa và tổng hợp trên các nút của cây phân tích có thể được miêu tả bằng đồ thị có hướng được gọi là đồ thị phụ thuộc (*dependency graph*).

Cây phụ thuộc của một cây phân tích cho trước, được xây dựng như sau:

for với mỗi nút n trên cây phân tích **do**

for với mỗi thuộc tính a của ký hiệu văn phạm tại nút n **do**

- xây dựng nút trên đồ thị phụ thuộc cho a;

for với mỗi nút n trên cây phân tích **do**

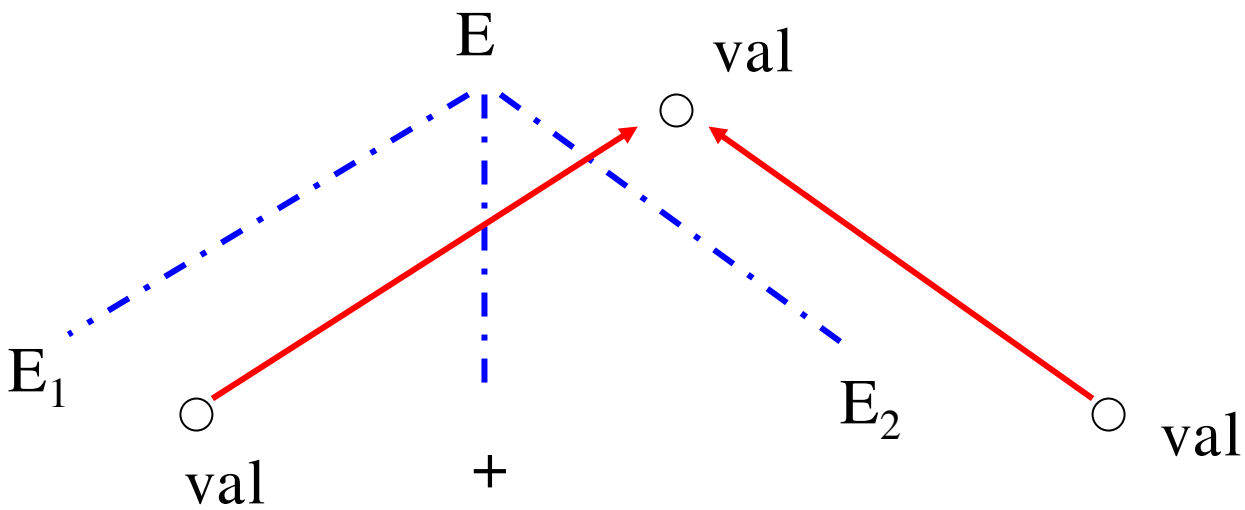
for với mỗi luật ngữ nghĩa b:= f (c_1, c_2, ..., c_k) tương ứng với luật sinh được dùng tại nút n

for i := 1 **to** k **do**

xây dựng cạnh đi từ nút của c_1 đến nút b.

Thí dụ 5.4. Khi ta dùng luật sinh E → E1 + E2 trên cây phân tích, chúng ta thêm các cạnh sau vào (H.5.3) chúng ta sẽ được đồ thị phụ thuộc.

Luật sinh	**Luật ngữ nghĩa**
E → E1 + E2	E.val := E1.val + E2.val

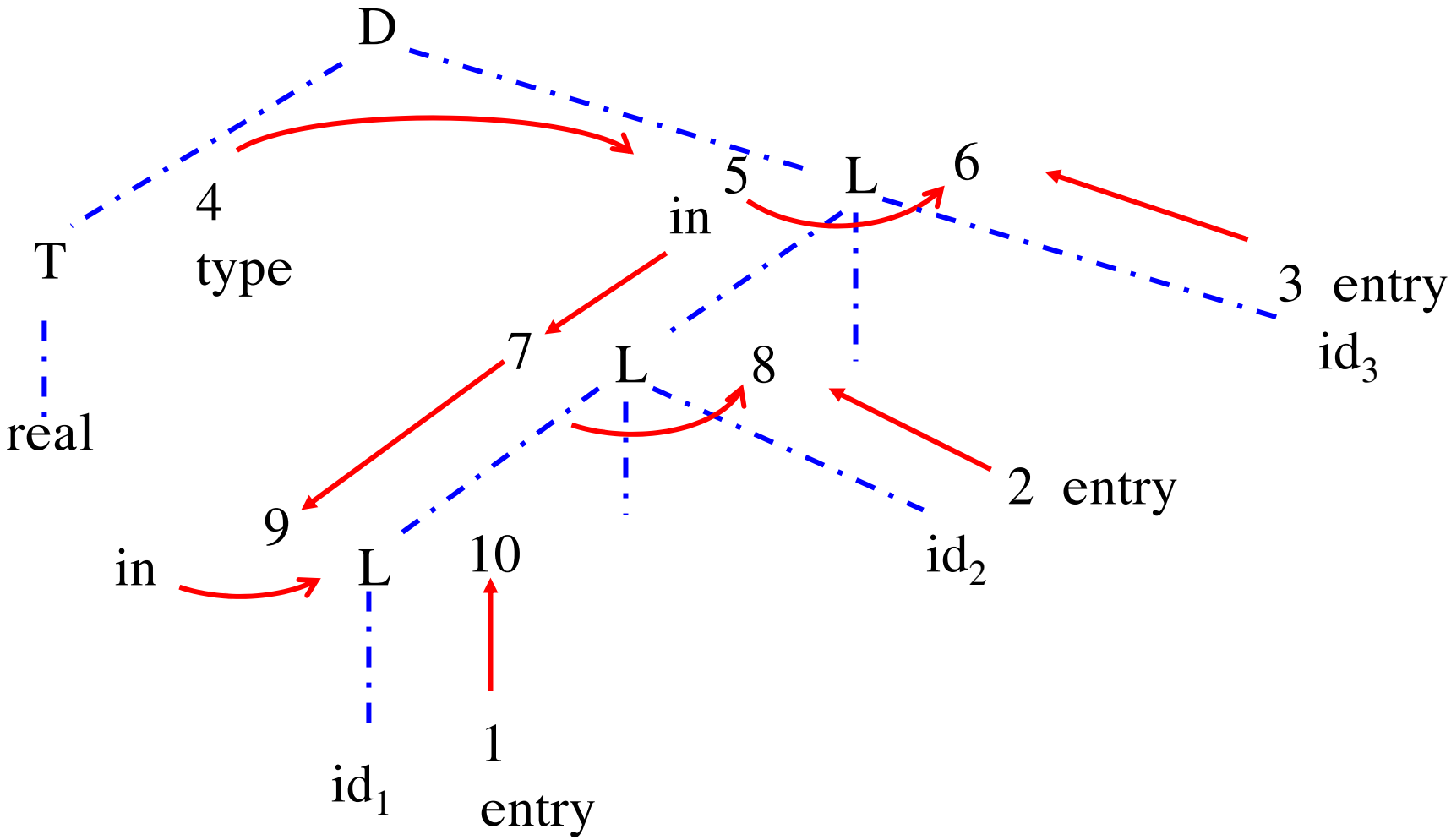


Hình 5.3. Đồ thị phụ thuộc của cây phân tích cho E → E1+ E2

Thí dụ 5.5. (H.5.4) là đồ thị phụ thuộc cho cây phân tích của (H.5.2).

Đánh giá thứ tự

Trong sắp xếp logic topo, các thuộc tính phụ thuộc c_1, c_2, ..., c_k trong luật ngữ nghĩa b:= f (c_1, c_2, ..., c_k) được đánh giá trước f.



Hình 5.4. Đồ thị phụ thuộc cho cây phân tích ở (H.5.2).

Thí dụ 5.6. Mỗi một cạnh của đồ thị phụ thuộc ở (H.5.4.) đi từ số thấp đến số cao của các nút. Từ thứ tự logic topo chúng ta sẽ có chương trình. Chúng ta viết an cho thuộc tính liên quan đến nút được đánh số n. trên đồ thị phụ thuộc.

```
a4 := real
a5 := a4
addtype (id3.entry, a5);
a7 := a5
addtype (id2.entry, a7);
a9 := a7
addtype (id1.entry, a9);
```

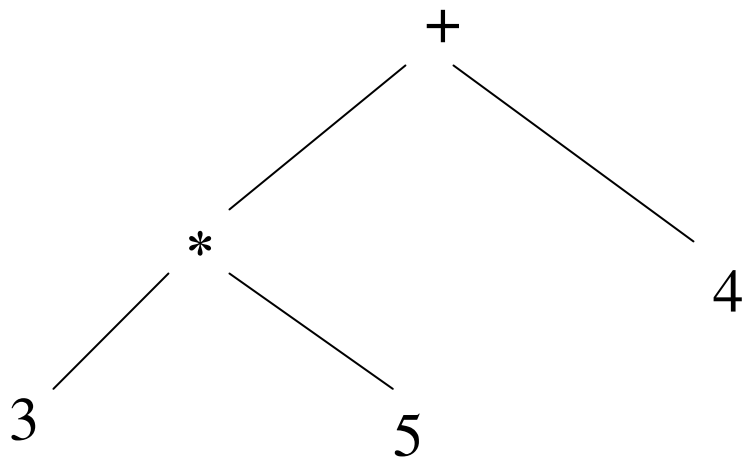
Một số phương pháp được đề nghị cho việc đánh giá các luật ngữ nghĩa

1. Phương pháp cây phân tích (parse-tree method)
2. Phương pháp cơ sở luật (rule-based method)
3. Phương pháp rõ ràng

5.2. Cấu trúc của cây phân tích

Cây cú pháp: là dạng thu gọn của cây phân tích, được dùng để biểu diễn cho cấu trúc ngôn ngữ.

Cây phân tích ở (H.5.1) sẽ được vẽ lại thành cây cú pháp.



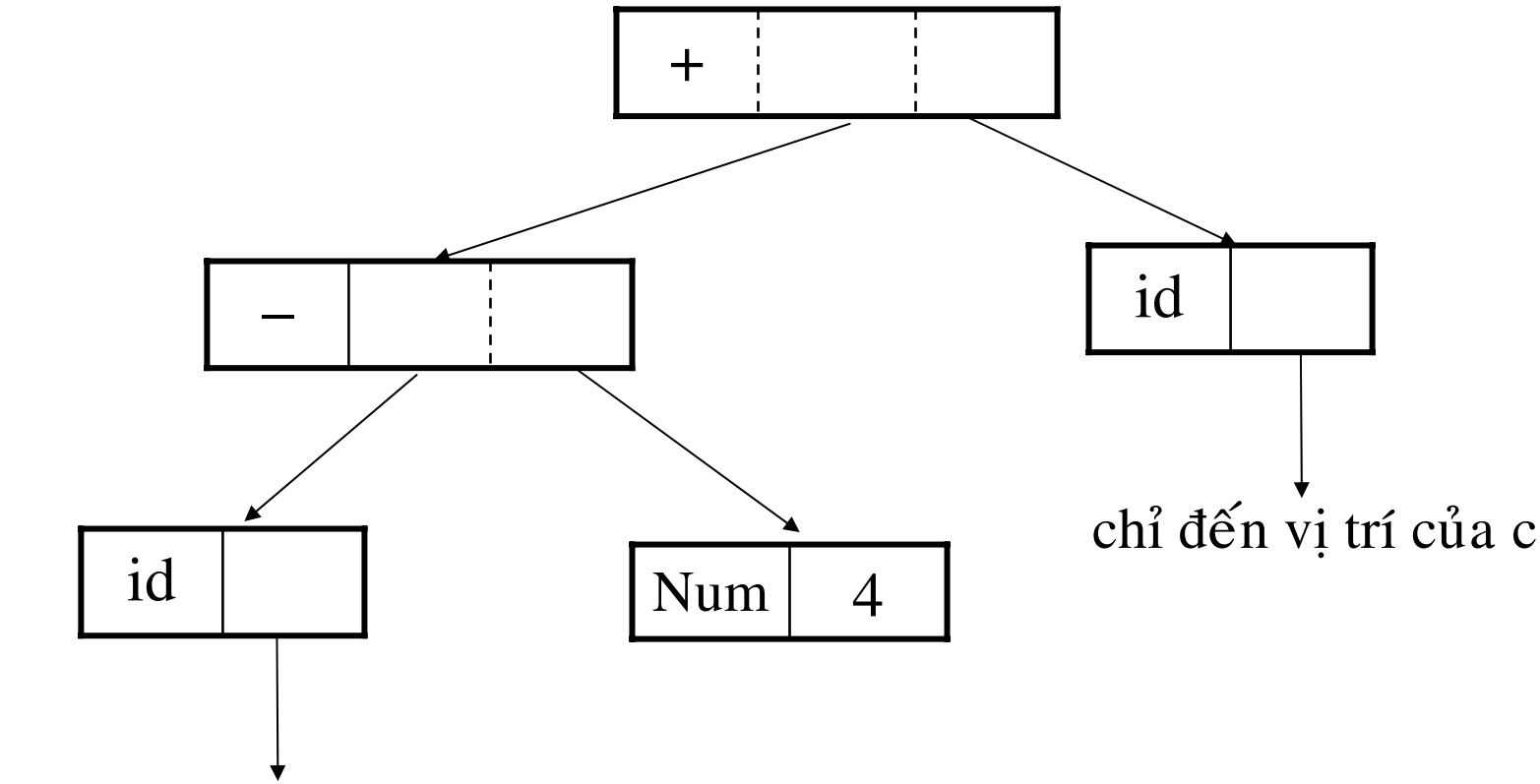
Xây dựng cây cú pháp cho biểu thức

Chúng ta sẽ dùng các hàm để tạo các nút cho cây cú pháp của biểu thức với phép toán hai ngôi. Mỗi hàm trả về con trỏ chỉ đến nút mới được tạo ra.

1. *mknode*(op, left, right).
2. *mkleaf*(id, entry).
3. *mkleaf*(num, val).

Thí dụ 5.7. Một chuỗi các hàm được gọi để tạo cây cú pháp cho biểu thức a – 4 + c ở (H.5.5).

(1) p1 := *mkleaf*(id, entry a);
(2) p2 := *mkleaf*(num, 4);
(3) p3 := *mknode*(‘-‘, p1, p2)
(4) p4 := *mkleaf*(id, entry c);
(5) p5 := *mknode*(‘+’, p3, p4)’



Hình 5.5. Cây cú pháp cho biểu thức a – 4 + c

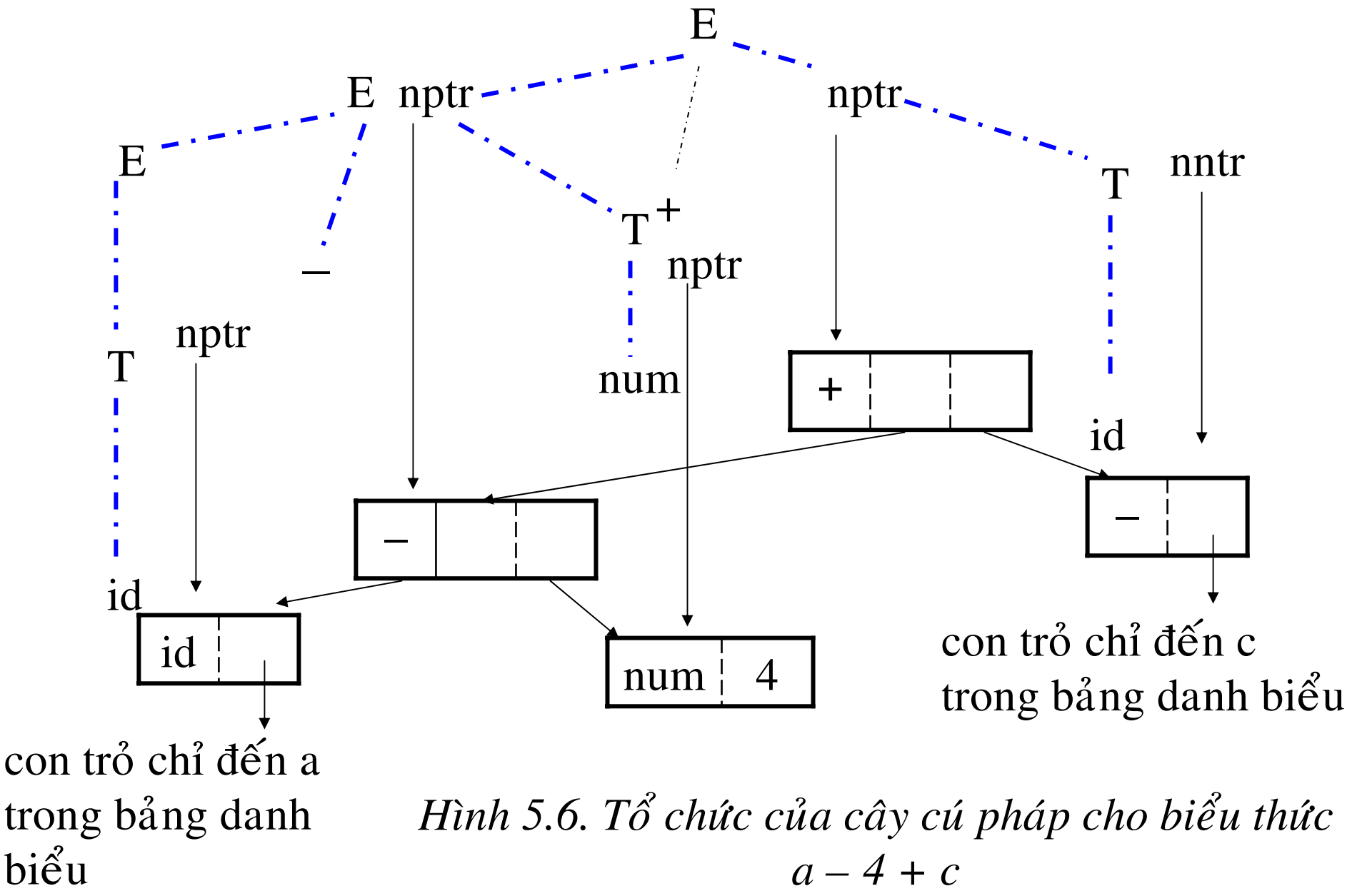
Định nghĩa trực tiếp cú pháp và cấu trúc cây cú pháp

Thí dụ ở bảng 5.3 là định nghĩa thuộc tính S dùng để xây dựng cây cú pháp cho biểu thức số học cộng (+) và trừ (-).

Bảng 5.3. *Định nghĩa trực tiếp cú pháp cho cấu trúc cây cú pháp của biểu thức*

Luật sinh	**Các 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)

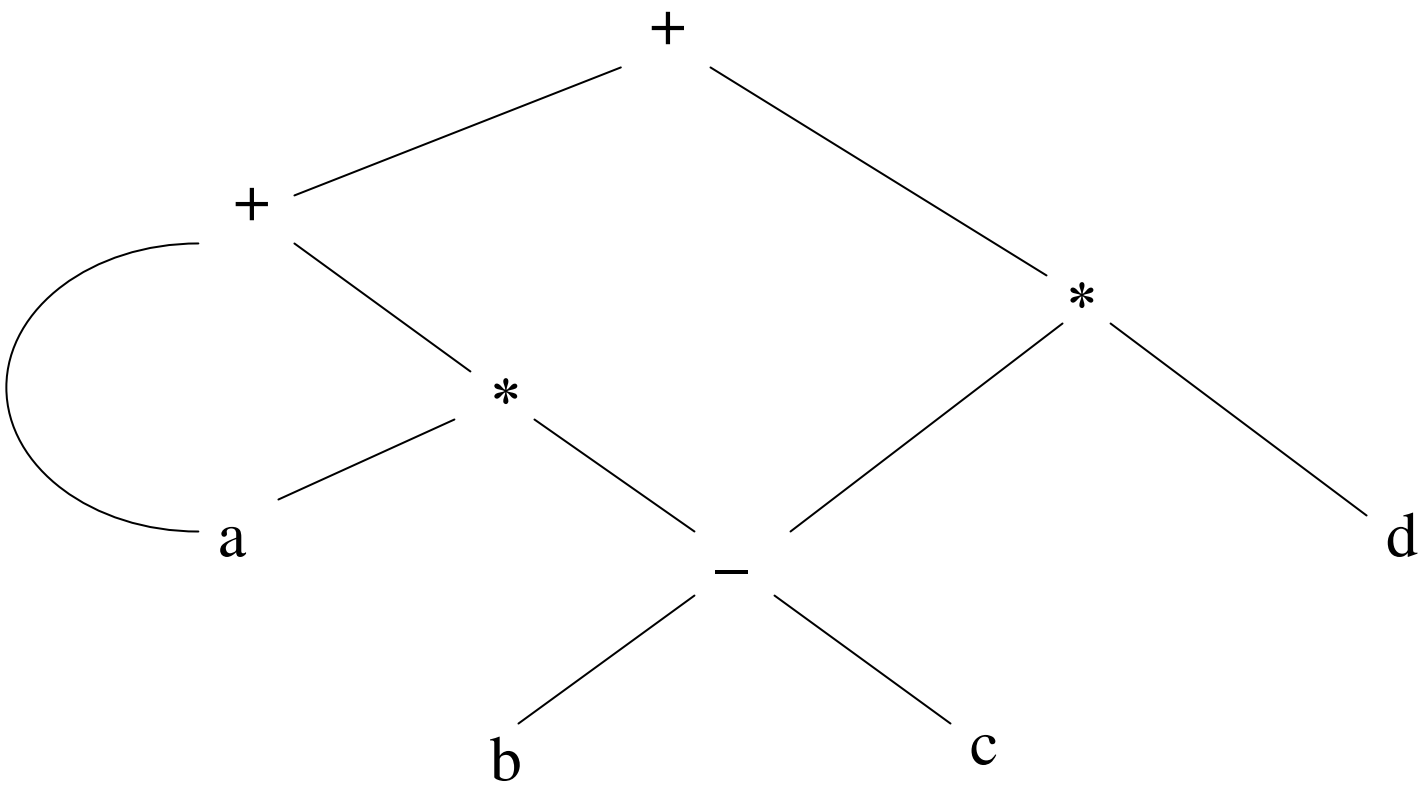
Thí dụ 5.8. Cây phân tích chú thích dùng để miêu tả cây cú pháp cho biểu thức a - 4 + c được trình bày ở (H.5.6).



Hình 5.6. Tổ chức của cây cú pháp cho biểu thức a – 4 + c

Đồ thị có hướng không lặp vòng miêu tả biểu thức

Đồ thị có hướng không lặp vòng (directed acyclic graph) gọi tắt là dag.



*Hình 5.7. Dag cho biểu thức a + a * (b – c) + (b – c) * d.*

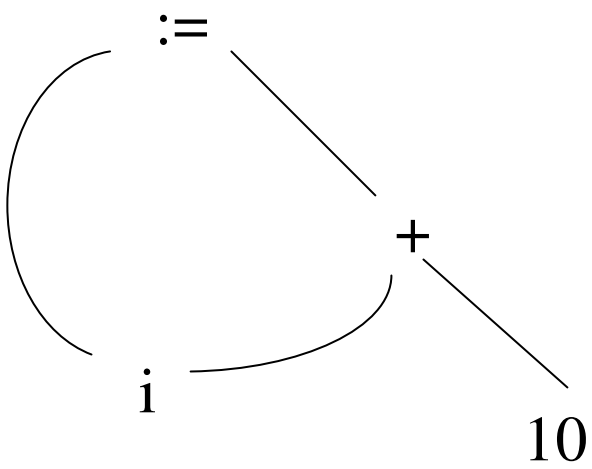
Bảng 5.4. *Các lệnh để tạo DAG ở (H.5.7)*

Hàng	Lệnh	Hàng	Lệnh
1	p_1 := *mkleaf*(**id**, a)	8	p_8 := *mkleaf*(**id**, b)
2	p_2 := *mkleaf*(**id**, a)	9	p_9 := *mkleaf*(**id**, c)
3	p_3 := *mkleaf*(**id**, b)	10	p_{10} := *mknode*(' −', p_8, p_5)
4	p_4 := *mkleaf*(id, c)	11	p_{11} := *mkleaf*(id, d)
5	p_5 := *mknode*('−', p_3, p_4)	12	p_{12} := *mknode*('*', p_{10}, p_{11})
6	p_6 := *mknode*('*', p_2, p_5)	13	p_{13} := *mknode*('+', p_7, p_{12})
7	p_7 := *mknode*('+', p_1, p_6)		

Mẫu tin tượng trưng cho nút được lưu chứa trong dãy như ở (H.5.8).

Phép gán Dag

i := i + 10



Biểu diễn cấu trúc dữ liệu

1	id	Chỉ đến danh biểu i	
2	num	10	
3	+	1	2
4	:=	1	3
5	.....		

Giải thuật 5.1. **Phương pháp số trị cho việc tạo nút của dag.**

Giả sử mỗi nút là một phần tử của dãy ở (H.5.8).

Nhập: nhãn *op*, nút *1* và nút *r*.

Xuất: nút với ký hiệu < op, 1, r >

Phương pháp

5.3. Đánh giá từ dưới lên cho định nghĩa thuộc tính S

Thuộc tính tổng hợp trên stack của bộ phân tích.

Bộ biên dịch cho định nghĩa thuộc tính S có thể được thực hiện dựa theo bộ sinh bộ phân tích cú pháp LR.

Bảng 5.5. *Stack của bộ phân tích có vùng lưu chứa các thuộc tính tổng hợp*

	state	val
	. . .	
	X	X.x
	Y	Y.y
top →	Z	Z.z
	.....	.....

Bảng 5.6. *Hiện thực bảng tính bằng bộ phân tích cú pháp LR*

Luật sinh	Đoạn mã
$L \rightarrow En$	Print (val [top])
$E \rightarrow E_1 + T$	val [ntop]: = val [top - 2] + val [top]
$E \rightarrow T$	
$T \rightarrow T_1 * F$	val [ntop]: = val [top - 2] x val [top]
$T \rightarrow F$	
$F \rightarrow (E)$	val [ntop]: = val [top - 1]
$F \rightarrow digit$	

Bảng 5.7. *Quá trình biên dịch cho chuỗi nhập 3 * 5 + 4n.*

Chuỗi nhập	Trạng thái	Trị val	Luật áp dụng
3 * 5 + 4n	–	–	
* 5 + 4n	3	3	
* 5 + 4n	F	3	F → digit
* 5 + 4n	T	3	T → F
5 + 4n	T *	3 –	
+ 4n	T * 5	3 – 5	
+ 4n	T * F	3 – 5	F → digit
+ 4n	T	15	T → T * F
+ 4n	E	15	E → T
4n	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 → E + T
	En	19	
	L	19	L → En

5.4. Định nghĩa thuộc tính L

Mô phỏng 5.1. *Thứ tự đánh giá depth – first cho các thuộc tính trên cây phân tích*

```
procedure dfvisit (n: node);
begin
        for với mỗi nút m là con của nút n, từ trái sang phải do
begin
        đánh giá thuộc tính kế thừa của m
        dfvisit (m)
        end
đánh giá thuộc tính tổng hợp của n
end;
```

Chúng ta trình bày lớp của định nghĩa trực tiếp cú pháp, được gọi là định nghĩa thuộc tính L như sau: *thuộc tính L luôn được đánh giá theo thứ tự depth – first*. Định nghĩa thuộc tính L bao gồm tất cả các định nghĩa trực tiếp cú pháp, được dựa trên cơ sở văn phạm LL (1).

Định nghĩa thuộc tính L

Định nghĩa trực tiếp cú pháp, được gọi là định nghĩa thuộc tính L, nếu mỗi thuộc tính kế thừa của xj với $1 < j \leq n$ mà x_j nằm ở vế phải luật sinh $A \rightarrow x_1x_2...x_n$, chỉ phụ thuộc vào:

1. Các thuộc tính của các ký hiệu $x_1, x_2, ..., x_{j-1}$ ở phía trái của xj trong luật sinh.
2. Thuộc tính kế thừa của ký hiệu A.

Bảng 5.8. *Định nghĩa trực tiếp cú pháp không phải thuộc tính L.*

Luật sinh	Luật ngữ nghĩa
$A \rightarrow LM$	L.i : = l (A.i)
	M.i := m (L.s)
	A.s : = f (M.s)
$A \rightarrow QR$	R.i : = r (A.i)
	Q.i : = q (R.s)
	A.s : = f (Q.s)

Lược đồ dịch

Mô phỏng 5.2. *Lược đồ dịch đơn giản cho biểu thức số học*

$E \rightarrow TR$

$R \rightarrow addop\ T\ \{print\ (addop.\ Lexeme)\}\ R \mid \in$

$T \rightarrow num\ \{print\ (num.\ val)\}$

Trường hợp đơn giản nhất nếu hành vi chỉ cần thuộc tính tổng hợp. Như vậy chúng ta sẽ xây dựng lược đồ dịch bằng cách tạo ra hành vi là phép gán cho mỗi luật ngữ nghĩa và gắn hành vi này vào tận cùng của vế phải luật sinh.

Thí dụ: ta có luật sinh và luật ngữ nghĩa sau:

Luật sinh	Luật ngữ nghĩa
T → T1 * F	T.val:= T1.val x F.val

ta đưa luật ngữ nghĩa 'nhúng' vào luật sinh và được:

T → T1 * F {T.val:= T1.val x F.val}

Nếu các hành vi cần cả thuộc tính tổng hợp và kế thừa thì chúng ta phải lưu ý:

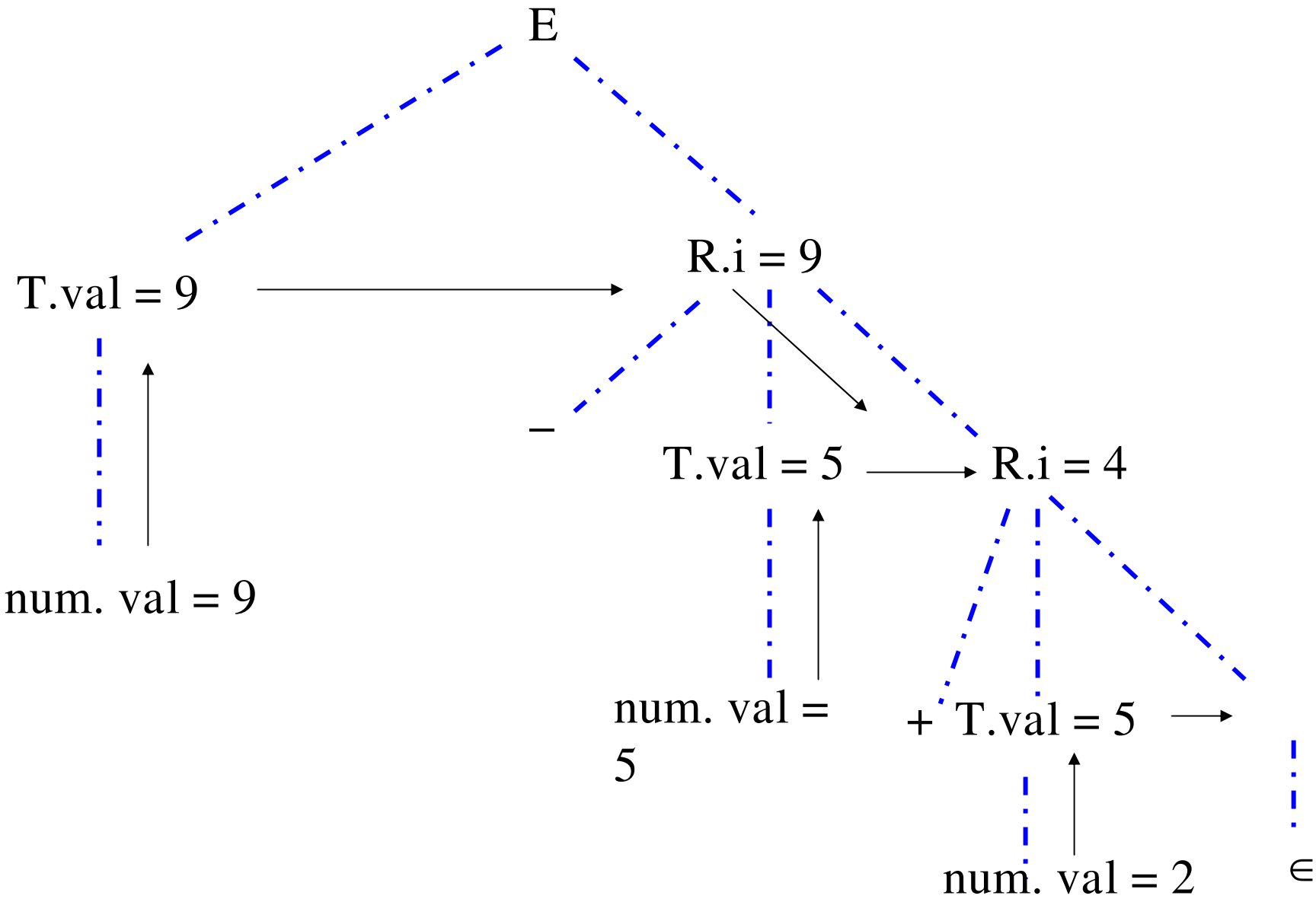
1. Thuộc tính kế thừa của một kýhiệu nằm ở vế phải luật sinh phải được tính trước trong hành vi đứng trước kýhiệu đó.
2. Hành vi không được tham khảo đến thuộc tính tổng hợp của ký hiệu nằm ở bên phải hành vi đó.
3. Thuộc tính tổng hợp của ký hiệu không kết thúc ở vế trái luật sinh chỉ có thể được tính sau tất cả các thuộc tính mà nó tham khảo tới.

5.5. Biên dịch từ trên xuống

Loại bỏ đệ quy trái cho lược đồ dịch

Mô phỏng 5.3. *Lược đồ dịch với văn phạm có đệ quy trái.*

$E \rightarrow E1 + T$	{E.val := E1.val + T. val | }
$E \rightarrow E1 - T$	{E.val := E1.val - T. val | }
$E \rightarrow T$	{E.val := T. val | }
$T \rightarrow E$	{T.val := E. val | }
$T \rightarrow$ **num**	{T.val := **num**. val | }



Hình 5.10. Đánh giá biểu thức 9 – 5 + 2.

Mô phỏng 5.4. *Lược đồ dịch chuyển đổi với văn phạm đệ quy trái.*

$E \rightarrow T$		{R.i := T.val | }
	R	{E.val := R.s}
$R \rightarrow +$		
	T	{R1.i := R.i + T.val | }
	R1	{R.s := R1.s}
$R \rightarrow -$		
	T	{R1.i := R.i - T.val | }
	R1	{R.s := R1.s}
$R \rightarrow \in$		{R.s := R.i}
$T \rightarrow ($		
	E	{T.val := E.val | }
)		
$T \rightarrow$ **num**		{T.val := **num**.val | }

Giả sử chúng ta có lược đồ dịch sau (với thuộc tính tổng hợp):

$A \rightarrow A1Y$ $\quad$ {A.a := g (A1.a, Y.y}

$A \rightarrow X$ $\quad$ {A.a := f (X.x)} $\quad$ (5.1)

Sau khi loại bỏ đệ quy trái chúng ta có văn phạm tương đương;

$A \rightarrow X$ $\quad$ {R.i := f (X.x)}

R $\quad$ {A.a := R.s}

$R \rightarrow Y$ $\quad$ {R1.i := g (R.i, Y.y)} $\quad$ (5.3)

R1 $\quad$ {R.i := R1.s}

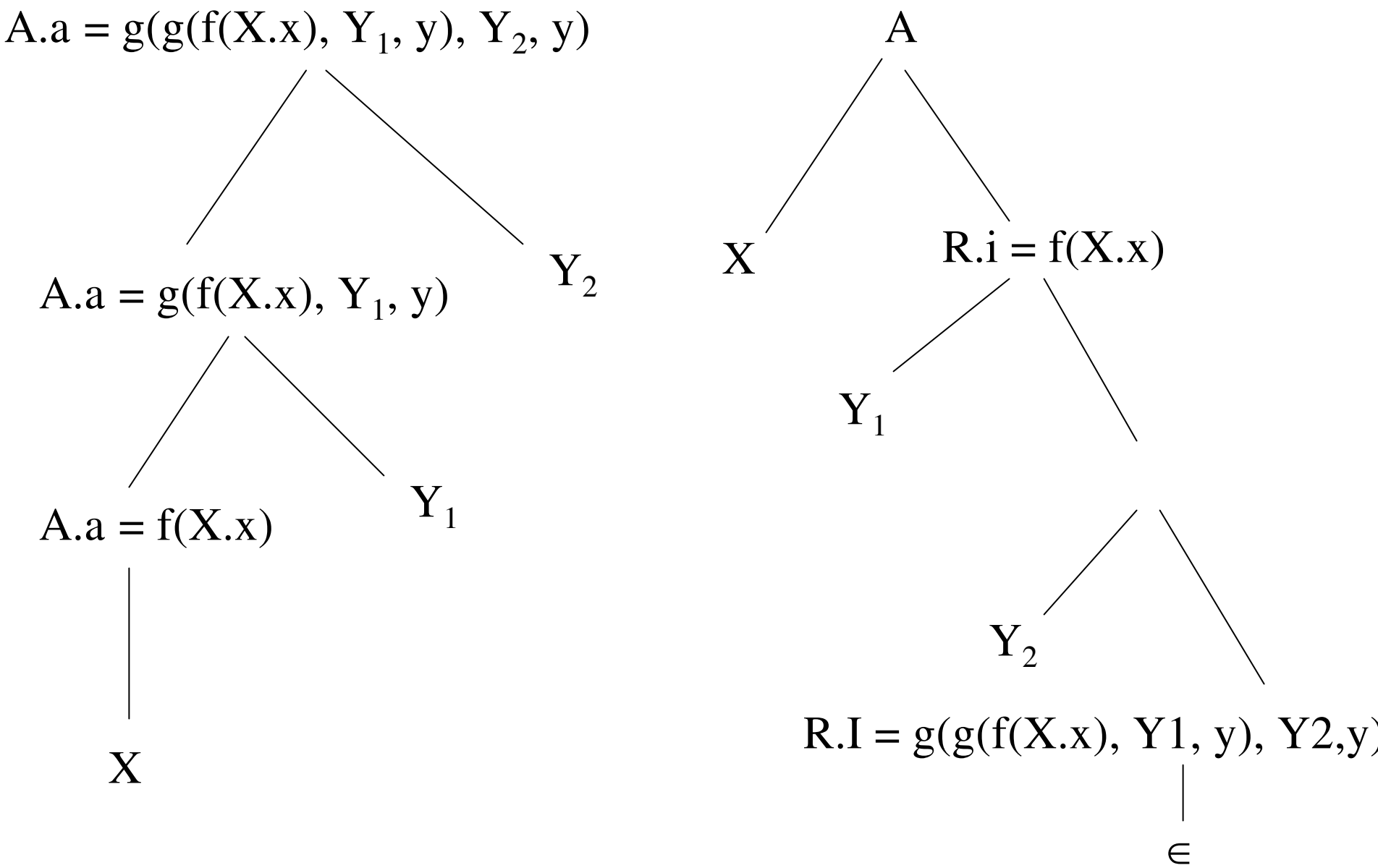
$R \rightarrow \in$ $\quad$ {R.s := R.i}

Thí dụ 5.13. Định nghĩa trực tiếp cú pháp ở bảng 5.3. dùng để xây dựng cây cú pháp được chuyển thành lược đồ dịch.

$E \rightarrow E1 + T$ $\quad$ {E.nptr := mknode (‘+’, E1.nptr, T.nptr)}

$E \rightarrow E1 - T$ $\quad$ {E.nptr := mknode (‘-’, E1.nptr, T.nptr)} $\quad$ (5.9)

$E \rightarrow T$ $\quad$ {E.nptr := T.nptr}



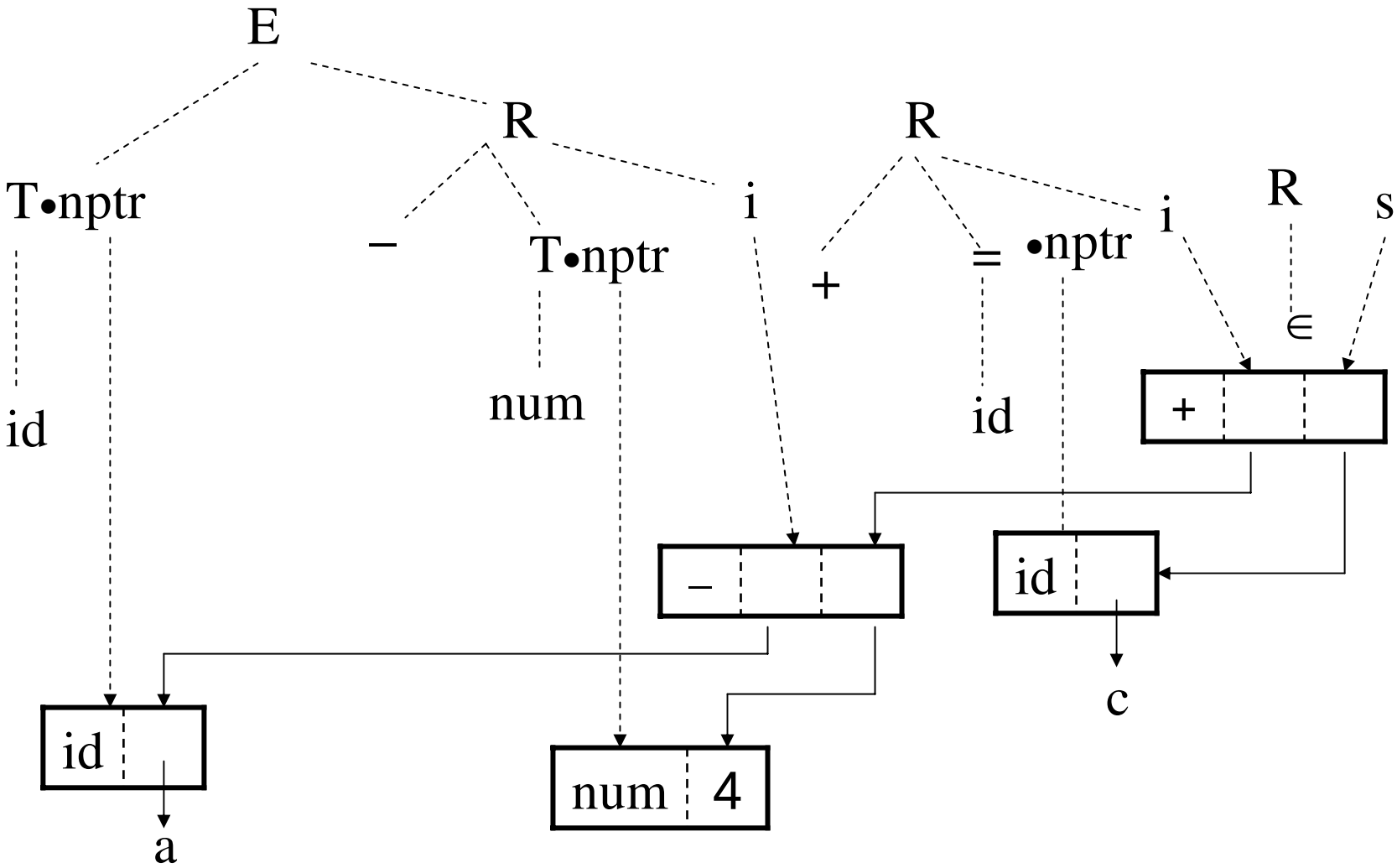
Hình 5.11. Hai cách tính giá trị thuộc tính.

Mô phỏng 5.5. *Lược đồ dịch chuyển đổi cho cấu trúc cây cú pháp.*

$E \rightarrow T$	{Rj := T.nptr}
R	{E.nptr := R.s}
$R \rightarrow +$	
T	{R_1j := mknode (‘+’, Rj.T.nptr)}
R_1	{R.s := R_1.s}
$R \rightarrow -$	
T	{R_1j := mknode (‘-’, Rj.T.nptr)}
R_1	{R.s := R_1.s}
$R \rightarrow \in$	{R.s := Rj}
$T \rightarrow ($	
E	
)	{T.nptr := E.nptr}
$T \rightarrow$ id	{T.nptr := mkleaf (id.id.entry)}
$T \rightarrow$ num	{T.nptr := mkleaf (num.num.val)}

Hình 5.12. biểu diễn toàn bộ các hành vi trong mô phỏng 5.5. cho cấu trúc cây cú pháp của câu a – 4 + c.

Thiết kế bộ dịch đoán nhận trước



Hình 5.12. Dùng các thuộc tính kế thừa để xây dựng cây cú pháp.

Giải thuật 5.2: xây dựng trình biên dịch trực tiếp cú pháp đoán nhận trước.

Nhập: cho lược đồ dịch trực tiếp cú pháp với văn phạm cơ sở phù hợp cho phân tích đoán nhận trước.

Xuất: mã cho trình biên dịch trực tiếp cú pháp.

Phương pháp:

1. Với mỗi ký hiệu không kết thúc A, xây dựng hàm, thông số là thuộc tính kế thừa của A, trả về giá trị của các thuộc tính tổng hợp của A.
2. Mã cho ký hiệu không kết thúc A sẽ quyết định luật sinh nào sẽ được dùng trên cơ sở ký hiệu nhập đang được đọc.
3. Mã cho mỗi luật sinh sẽ được tạo ra:

i) Với mỗi token X với thuộc tính tổng hợp x, cất giá trị x vào biến X.x. Tạo ra lệnh gọi chương trình con để so trùng token X với ký hiệu nhập được đọc.

ii) Với B, tạo ra phát biểu gán C1 = B (b1, b2, ..., bk), b1, b2, ..., bk là các biến chứa các thuộc tính kế thừa của B và C là biến chứa thuộc tính tổng hợp của B.

iii) Với mỗi hành vi, hãy chép mã vào cho bộ phận tích, thay mỗi tham chiếu đến các thuộc tính bằng biến chứa các thuộc tính đó.

Thí dụ 5.14. Văn phạm ở mô phỏng 5.5 là LL (1), nó phù hợp cho việc phân tích từ trên xuống.

function E: ↑ nút cây cú pháp
function R: (i: ↑ nút cây cú pháp): ↑ nút cây cú pháp;
function T: ↑ nút cây cú pháp;

Kết hợp hai luật sinh R ở mô phỏng 5.5.

$R \rightarrow$ **addop**
T {R1.i :=mknode (**addop**.lexeme, R.i, T.nptr)}
R1 {R.s := R1.s}
$R \rightarrow \in$ {R.s := R.i}

Mô phỏng 5.6. *Thủ tục phân tích cú pháp cho các luật sinh*

R: $R \rightarrow$ ***addop*** $TR \mid \in$

```
Procedur R:
        begin
            if lookahead = addop then begin
                match (addop): T; R;
            end
            else begin /*không làm gì cả*/
        end
```

Mô phỏng 5.7. *Cây cú pháp đệ quy đi xuống*

```
function R (i: ↑ nút cây cú pháp): ↑ nút cây cú pháp;
        var nptr. ll, sl, s: ↑ nút cây cú pháp;
            addoplexeme: char;
        begin if lookahead = addop then begin
            /* luật sinh R → addop TR*/
            addoplexeme := lexval;
            match (addop);
```

```
        il := mknode (addoplexeme, i, nptr);
              sl := R (il);
              s := sl;
            end
            else s := i; /* luật sinh R → ∈ */
            return s
        end;
```

5.6. Đánh giá thuộc tính kế thừa từ dưới lên

Loại bỏ hành vi được nhúng trong lược đồ dịch

Ví dụ: chúng ta có lược đồ dịch

$E \rightarrow TR$

$R \rightarrow + T \{print (‘+’)\} R \mid - T \{print (‘-‘)\} R \mid \in$

$T \rightarrow \textbf{num} \{print (\textbf{num}.val)\}$

Tạo ra lược đồ dịch với việc dùng các ký hiệu đánh dấu không kết thúc mới N, M.

$E \rightarrow TR$

$R \rightarrow + T\ MR \mid - TNR \mid \in$

$T \rightarrow num$ {print (**num**.val)}

$M \rightarrow \in$ {print ('+') }

$N \rightarrow \in$ {print ('−') }

Thuộc tính kế thừa trên stack của bộ phân tích

Thí dụ 5.15. Quá trình đánh giá thuộc tính kế thừa bằng bộ phân tích từ dưới lên cho câu nhập **real** p, q, r ở (H.5.13).

$D \rightarrow T$ {L.in := T.type}

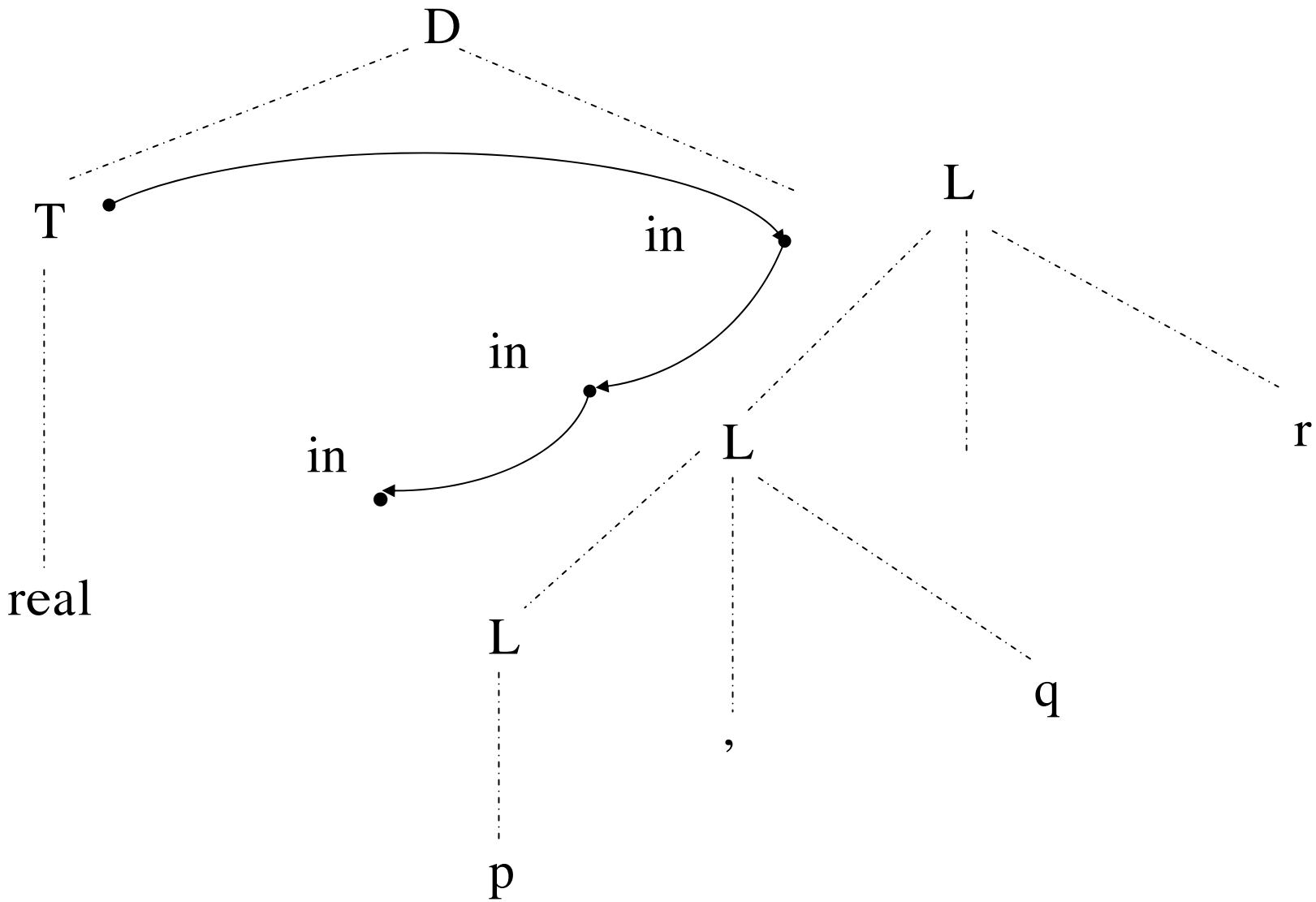
$T \rightarrow$ **int** {T.type := integer}

$T \rightarrow$ **real** {T.type := real}

$L \rightarrow$ {L1.in := L.in}

L1, **id** {add type (**id**.entry, L.in)}

$L \rightarrow$ **id** {add type (**id**.entry, L.in)}



Hình 5.13. Tại mỗi nút L có L.in := T.type.

Bảng 5.9. *Bất cứ lúc nào vế phải của L được thu giảm thì T luôn ở trên vế phải đó.*

Nhập	Trạng thái	Luật được áp dụng
Real p, q, r	-	
p, q, r	**real**	
p, q, r	T	T → **real**
, q, r	Tp	
, q, r	TL	L → **id**
, q, r	TL,	
, r	TL, q	
, r	TL	L → L, **id**
r	TL,	
	TL, r	
	TL	L → L, **id**
	D	D → TL

Bảng 5.10. *Giá trị của T.type được dùng ở vị trí L.in.*

Luật sinh	Đoạn mã
D → TL	
T → **int**	val [ntop] := integer
T → **real**	val [ntop] := real
L → L, **id**	addtype (val[top], val[top – 3])
L → **id**	addtype (val[top], val[top – 1])

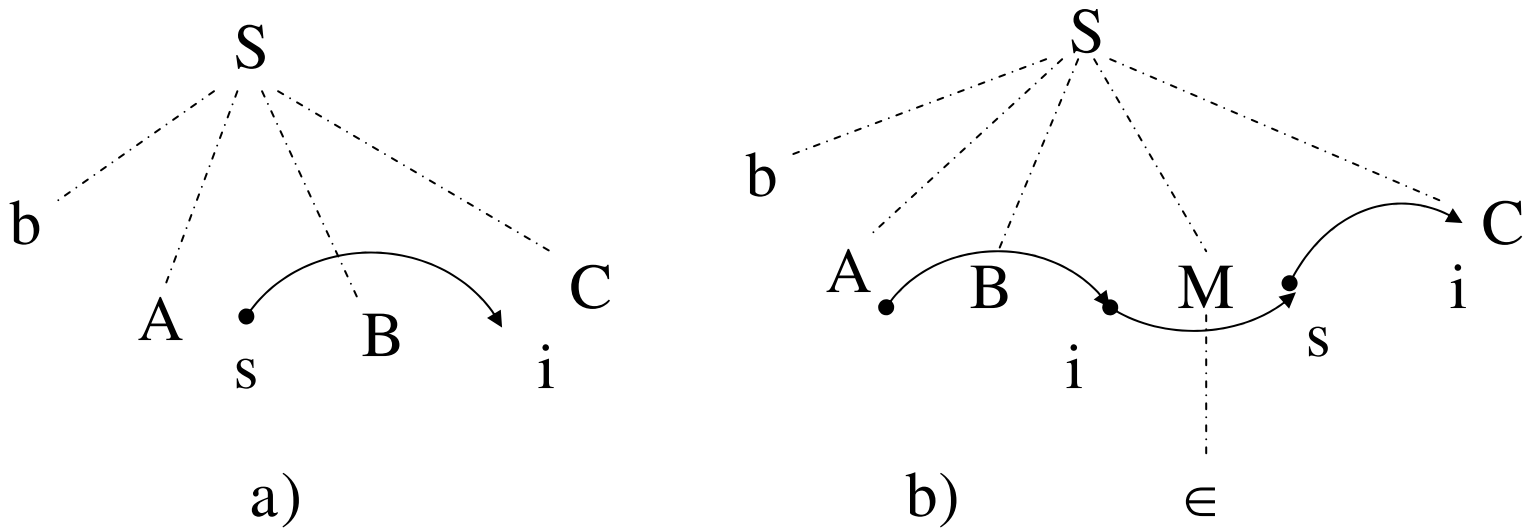
Đánh giá các thuộc tính kế thừa

Thí dụ 5.16. Đây là ví dụ về trường hợp không thể đoán nhận trước vị trí của thuộc tính trong lược đồ dịch.

Luật sinh	Luật ngữ nghĩa
S → aAC	C.i:= A.s
S → aABC	C.i := A.s
C → c	C.i := g (C.i)

(5.4)

Luật sinh	Luật ngữ nghĩa
S → aAC	C.i := A.s
S → bABMC	M.i := A.s ; C.i := M.s
C → c	C.i := g(C.i)
M → ∈	M.S := M.i



Hình 5.14. Sao chép thuộc tính thông qua ký hiệu M.
a) Luật sinh chưa biến đổi; b) Luật sinh đã được biến đổi.

Ký hiệu không kết thúc N cũng có thể được dùng để mô phỏng cho luật ngữ nghĩa mà nó không phải là luật sao chép. Ví dụ ta có luật sinh và luật ngữ nghĩa:

Luật sinh	Luật ngữ nghĩa
S → aAC	C.i := f(A.s)

(5.5)

Luật sinh	Luật ngữ nghĩa
S → aANC	N.i := A.s ; C.i := N.s
N → ∈	N.s := f(N.i)

(5.6)

Giải thuật 5.3. Phân tích từ dưới lên và sự biên dịch với các thuộc tính kế thừa.

Nhập: định nghĩa thuộc tính L với văn phạm cơ sở LL (1).

Xuất: bộ phân tích cú pháp tính các giá trị của tất cả các thuộc tính trên stack của bộ phân tích.

Phương pháp: giả sử mỗi ký hiệu không kết thúc A có một thuộc tính kế thừa A.i và mỗi ký hiệu văn phạm X có một thuộc tính tổng hợp X.x.

Với mỗi luật sinh A → X1 ... Xn, sẽ có n ký hiệu không kết thúc đánh dấu M1 ...Mn, sẽ thay luật trên thành luật sinh A → M1X1 ... MnXn. Để nhận thấy các thuộc tính có thể được tính trong quá trình phân tích từ dưới lên, hãy xét hai trường hợp.

Trường hợp thứ nhất nếu ta thu giảm về ký hiệu Mj ta phải biết luật sinh A → Mj X1 ... MnXn mà Mj có trong đó. Chúng ta phải biết các vị trí của các thuộc tính mà thuộc tính kế thừa Xj.i cần để tính giá trị cho nó. A.i ở val[top – 2j + 2], X1.i ở val[top – 2j + 3], X1.s ở tại val[top – 2i + 4], X2.i ở val[top – 2j + 5]...

Trường hợp thứ hai sẽ xuất hiện khi ta thu giảm về một ký hiệu không kết thúc của văn phạm giả sử bằng luật sinh A → M1X1 ... MnXn, và giả sử ta chỉ tính A.s, còn A.i đã được sinh và nằm trên stack ở vị trí trên vị trí của A. Các thuộc tính cần thiết để tính A.s đã sẵn sàng trên stack, đã được biết, đó chính là các vị trí của các Xj trong quá trình thu giảm.

Thay thế thuộc tính kế thừa bằng thuộc tính tổng hợp

Chúng ta có thể tránh dùng thuộc tính kế thừa bằng việc thay đổi văn phạm cơ sở. Trong ngôn ngữ của Pascal cho phép khai báo một chuỗi các biến và sau đó là kiểu dữ liệu của chúng. Thí dụ: m, n: integer.

$D \rightarrow L : T$

$T \rightarrow$ **integer** | **char**

$L \rightarrow L,$ id | id

$D \rightarrow$ id L

$L \rightarrow$,id Ll:T

$T \rightarrow$ **integer** | **char**