PHENIKAA UNIVERSITY
Realizing your potential

CHƯƠNG TRÌNH DỊCH

Chương 7. Xử lý ngữ nghĩa

TS. Phạm Văn Cảnh
Khoa Công nghệ thông tin

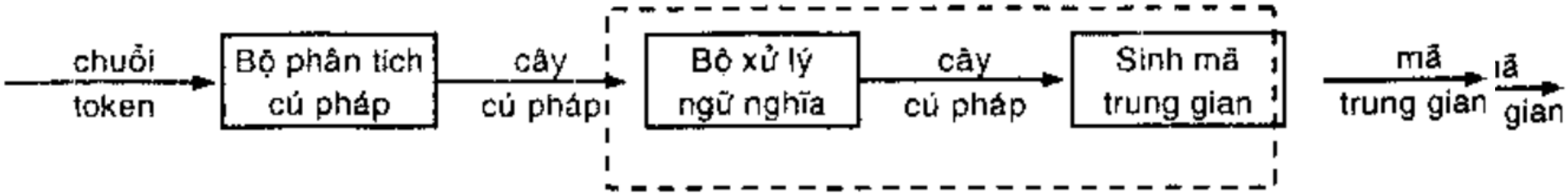
Email: canh.phamvan@phenikaa-uni.edu.vn

Nội dung

PHENIKAA UNIVERSITY
Realizing your potential

1. Tổng quan xử lý ngữ nghĩa
2. Truyền thuộc tính
3. Xử lý ngữ nghĩa với phân tích từ dưới lên
4. Kiểm tra kiểu dữ liệu
5. Chuyển đổi kiểu
6. Xử lý ngữ nghĩa cho phát biểu Goto tham khảo trước
7. Bài tập

1. Tổng quan xử lý ngữ nghĩa



- Trình biên dịch phải kiểm tra chương trình nguồn về cú pháp và ngữ nghĩa.
- Trong chương này giới thiệu các phương pháp kiểm tra ngữ nghĩa:
  - Kiểm tra tĩnh (static check): thực hiện trong thời gian biên dịch;
  - Kiểm tra động: thực hiện trong thời gian thực thi.
- Chương này tập trung vào kiểm tra ngữ nghĩa tĩnh

1. Tổng quan xử lý ngữ nghĩa

Xử lý nghĩa nghĩa tĩnh bao gồm:

- Truyền thuộc tính: là quá trình xác định trị thuộc tính (hay còn gọi là trị nhữ nghĩa) cho mỗi nút trên cây cú pháp.
- Kiểm tra kiểu: kiểm tra kiểu của các toán hạng có hợp với toán tử không.
  - Ví dụ: toán hạng số học không thể áp dụng cho kiểu logic
- Kiểm tra trình tự điều khiển: trình biên dịch kiểm tra trình tự thực thi của các phát biểu, sự rẽ nhánh của các phát biểu điều kiện, chuyển sự điều khiển này sang phát biểu khác.
- Kiểm tra tính duy nhất: mỗi đối tượng chỉ khai báo một lần.
- Kiểm tra mối liên hệ của tên: kiểm tra các tên trùng nhau đã hợp lệ.

1. Tổng quan xử lý ngữ nghĩa

Xử lý nghĩa nghĩa tĩnh bao gồm:

- Xử lý các tham khảo goto trước: Trình biên dịch phải có khả năng lưu nhớ được các vị trí (địa chỉ) của các phát biểu goto.

2. Truyền thuộc tính

- Trong quá trình truyền thuộc tính trên cây cú pháp, từ các nút con lên nút cha.
  - Nếu nút cha là nút tượng trưng cho phép toán thì bộ xử lý ngữ nghĩa sẽ cung cấp thủ tục yêu cầu hệ thống cấp phát vị trí nhớ để chứa kết quả phép toán nếu cần gọi và thủ tục tạo mã trung gian

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

- Đối với trình biên dịch, việc phân tích cú pháp và xử lý ngữ nghĩa thường được thực hiện xen kẽ.
- Mỗi khi bộ phân tích cú pháp nhận dạng ra một cấu trúc của chương trình nguồn (toán hạng, biểu thức, cú pháp) thì nó sẽ gọi một chương trình con xử lý ngữ nghĩa chuyên trách tiếp nhận cấu trúc ấy, kiểm tra tính đúng đắn của nó về mặt ngữ nghĩa và sinh mã trung gian.

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

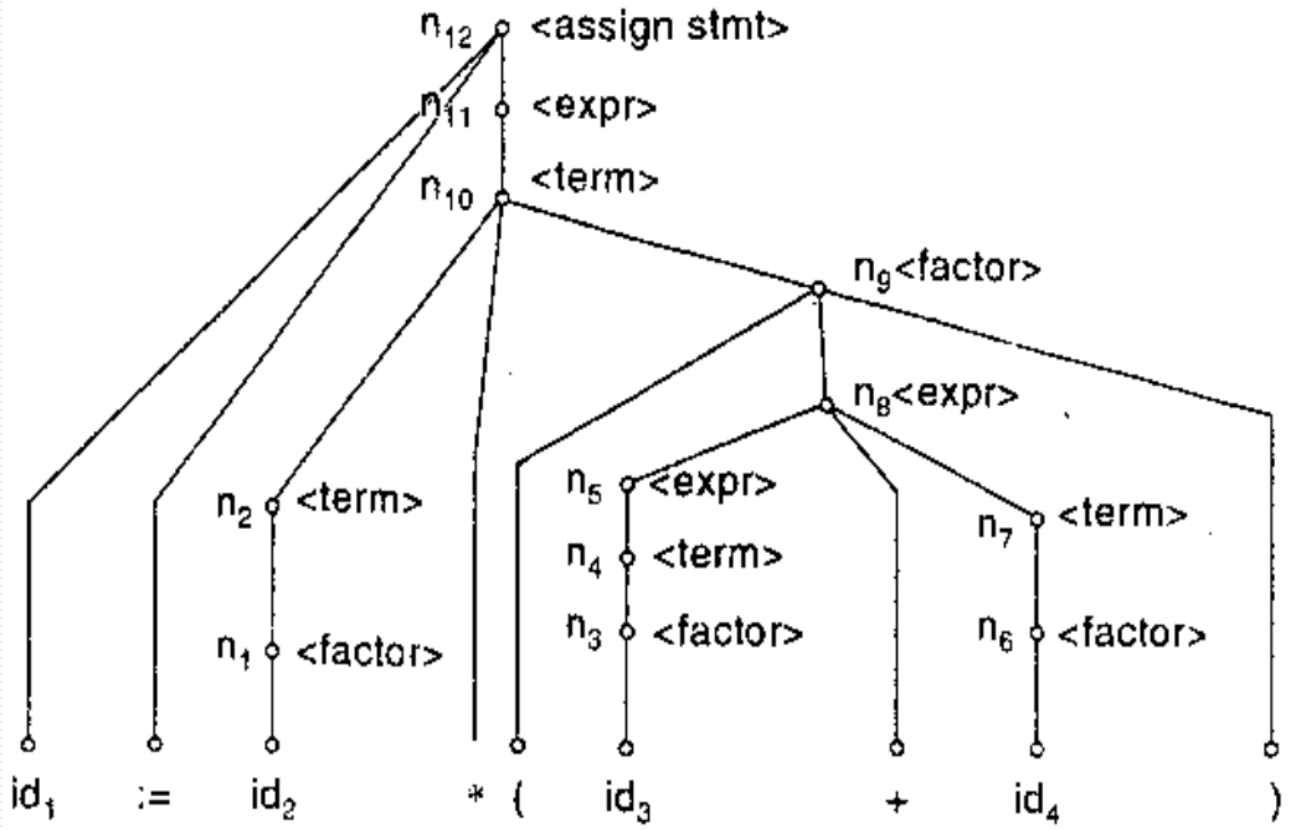
- Vấn đề truyền thuộc tính
- Cho văn phạm. Phân tích cú pháp cho biểu thức: A=X*(R+Q)

<assign stmt> → id := <expr>

<expr> → < expr> + <term> | <term>

<term> → < term> * <factor> | <factor>

<factor> → id | (< expr>)



3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

- Vấn đề truyền thuộc tính
- Cho văn phạm. Phân tích cú pháp cho biểu thức: A=X*(R+Q)

<assign stmt> → id := <expr>

<expr> → < expr> + <term> | <term>

<term> → < term> * <factor> | <factor>

<factor> → id | (< expr>)

Bảng danh biểu

Token	Trị từ vựng	Kiểu dữ liệu
id	A	thực
id	X	thực
id	R	thực
id	Q	thực

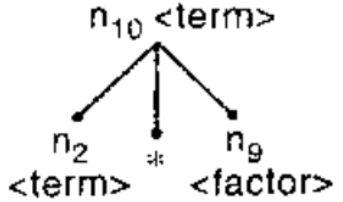
Mã bộ tứ được sinh ra: + (R, Q, T1)

* (X, T1, T2)

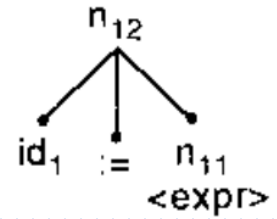
as (T2, A,)

+ (R, Q, T1) được sinh ra ở cây con:

* (X, T1, T2) được sinh ra tại cây con:



as (T2, A,) được sinh ra tại cây:

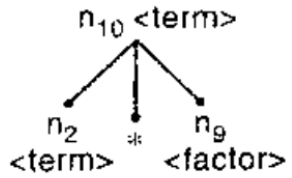


3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

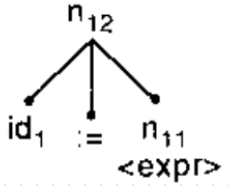
Mã bộ tứ được sinh ra: + (R, Q, T1)
* (X, T1, T2)
as (T2, A,)

+ (R, Q, T1) được sinh ra ở cây con:

* (X, T1, T2) được sinh ra tại cây con:



as (T2, A,) được sinh ra tại cây:



- Các cây con không tồn tại giá trị từ vựng: A, X, R, T1, T2→ không thể sinh mã trung gian.
- Muốn sinh được mã trung gian, chúng ta phải gắn ngữ nghĩa vào cây cú pháp
  - Ví dụ: n5 gắn biến thực R, n7 gắn biến thực Q.

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

PHENIKAA UNIVERSITY
Realizing your potential

- Trong quá trình phân tích cú pháp ứng với mỗi bước thu giảm sẽ sinh ra hành vi xử lý ngữ nghĩa:
  - Truyền thuộc tính: Ví dụ giảm nút n4 về nút n5 phải truyền thuộc tính từ n4 về n5 để biết rằng <term> hay <expr> đều chỉ là biến thực R.
  - Sinh ra biến tạm khi thu giảm.

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

PHENIKAA UNIVERSITY
Realizing your potential

- Phương pháp thực hiện sự truyền thuộc tính.
  - Để thực hiện xử lý ngữ nghĩa trong quá trình phân tích cú pháp ta sẽ dụng một stack đặc biệt mà mỗi phần tử của stack là một mẩu tin bao gồm:
    - A: Ký hiệu văn phạm (tương trưng cho danh biểu)
    - B: có giá trị 0 hoặc 1 (ký hiệu 1 là biến tạm)
    - C: Con trỏ chỉ đến bảng danh biểu (thực chất là vị trí của biểu thức trong bảng danh biểu)
  - Stack có các phần tử như trên được gọi là stack cú pháp, được xem như 3 stack song song.
  - A, B là stack trạng thái, C là stack ngữ nghĩa.

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

Bảng danh biểu

Token	Trị từ vựng	Kiểu dữ liệu
id	A	thực
id	X	thực
id	R	thực
id	Q	thực

- Ví dụ: cho văn phạm sau:

<asign stmt> → id := <expr>

<expr> → <expr> + <term> | <term>

<term> → <term> * <factor> | <factor>

<factor> → id | (<expr>)

- Thực hiện phân tích cú pháp và xử lý ngữ nghĩa cho câu: A:=X*(R+Q)
- Ký hiệu:

id là	I	# là ký hiệu đánh dấu đáy
<expr>	E	stack trạng thái (stack A)
<term>	T	và kết thúc chuỗi nhập
<factor>	F	
<assign stack>	AS	

- Quá trình phân tích từ dưới lên như sau:

Bước	Tình trạng stack	Chuỗi nhập còn lại	Chú thích
1	2	3	4
0	A B C #	$I_1 := I_2 * (I_3 + I_4)$#	Trạng thái bắt đầu
1	I_1 0 1 #	$:= I_2 * (I_3 + I_4)$#	đẩy biến A sang stack
2	:= I_1 0 1 #	$I_2 * (I_3 + I_4)$ #	đẩy dấu ":=" sang stack
3	I_2 0 2 := I_1 0 1 #	$* (I_3 + I_4)$ #	đẩy X sang stack
4	F 0 2 := I_1 0 1	$* (I_3 + I_4)$ #	Thu giảm I_2 về F
5	T 0 2 := I_1 0 1 #	$* (I_3 + I_4)$ #	Thu giảm F về T

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

Bước	Tình trạng stack	Chuỗi nhập còn lại	Chú thích
1	2	3	4
6	* T 0 2 := I_1 0 1 #	(I_3 + I_4) #	đẩy dấu '*' sang stack
7	(* T 0 2 := I_1 0 1 #	(I_3 + I_4) #	đẩy dấu '(' sang stack
8	I_3 0 3 (* T 0 2 := I_1 0 1 #	+ I_4) #	đẩy R sang stack
9	F 0 3	+ I_4) #	Thu giảm I_3 về F
10	T 0 3 (* T 0 2 := I_1 0 1 #	+ I_4) #	Thu giảm F về T
11	E 0 3 (* T 0 2 := I_1 0 1 #	+ I_4) #	Thu giảm T về E
12	+ E 0 3 (* T 0 2 := I_1 0 1 #	I_4)	đẩy dấu '+' sang stack

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

13	I_4	0	4	) #	đẩy Q sang stack
	+				
	E	0	3		
	(				
	*				
	T	0	2		
	:=				
	I_1	0	1		
	#				
14	F	0	4	) #	Thu giảm I_4 về F
	+				
	E	0	3		
	(				
	*				
	T	0	2		
	:=				
	I_1	0	1		
	#				
15	T	0	4	) #	Thu giảm F về T
	+				
	E	0	3		
	(				
	*				
	T	0	2		
	:=				
	I_1	0	1		
	#				
16	E	1	5	#	i) Thu giảm E+T về E ii) E có thuộc tính 1 (biến tạm) là vị trí của T1 trong bảng danh biểu. Trình biên dịch tạo biến tạm T1 để lưu chứa kết quả phép toán R+Q iii) Sinh mã bộ tứ + (3, 4, 5)
	(				
	*				
	T	0	2		
	:=				
	I_1	0	1		
	#				
17	)			#	Đẩy dấu ')' sang stack
	E	1	5		
	(				
	*				
	T	0	2		
	:=				
	I_1	0	1		
	#				
18	F	1	5	#	Thu giảm (E) về F
	*				
	T	0	2		
	:=				
	I_1	0	1		
	#				

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

Bước	Tình trạng stack	Chuỗi nhập còn lại	Chú thích
1	2	3	4
19	T 1 6 := I_1 0 1 #	#	i) Thu giảm T*F về T ii) Sinh biến tạm T2 có thuộc tính 1, 6 là vị trí của T2 trong bảng danh biểu iii) Sinh mã bộ tứ * (2, 5, 6)
20	E 1 6 := I_1 0 1 #	#	Thu giảm T về E
21	AS #	#	i) Thu giảm I_1 := E về AS ii) Sinh mã bộ tứ AS (1, 6,)

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

PHENIKAA UNIVERSITY
Realizing your potential

Nguyên tắc xử lý ngữ nghĩa:

- Khi có một dãy con x=$x_1 x_2 .. x_n$ của một dạng câu đang phân tích sắp được thu giảm về ký hiệu không kết thúc U, thì hành vi ngữ nghĩa tại nút V là hàm của:

1) Luật sinh U → x_1 x_2 ... x_n

2) Các xử lý ngữ nghĩa của các nút xi, tức là các nút con của V

Một hành vi ngữ nghĩa tại một nút V có thể là

i) Tra cứu bảng danh biểu

ii) Tạo ra biến tạm

iii) Sinh mã trung gian

it) Truyền thuộc tính từ x về V

- Động tác sinh mã trung gian là động tác ngữ nghĩa chính, còn lại là động tác ngữ nghĩa phụ.

3. Xử lý ngữ nghĩa với phân tích cú pháp từ dưới lên

PHENIKAA UNIVERSITY
Realizing your potential

- Ví dụ: Xét văn phạm:

<assign stmt> → id := <expr>

<expr> → <expr> + <term> | <term>

<term> → <term> * <factor> | <factor>

<factor> → id | (<expr>)

- Hành vi nghữ nghĩa liên kết với hành vi thu giảm luật như sau

1- <factor> → id

Gắn con trỏ của danh biểu id (vị trí của id trong bảng danh biểu) ô ngữ nghĩa của <factor> trên stack, bằng việc gọi hàm NAME.

F := NAME (id)

Trị ngữ nghĩa của F là con trỏ của id

2- <factor> → (<expr>)

truyền trị ngữ nghĩa của E sang cho F.

F = E

3- <expr> → <expr> + <term>

Hành vi ngữ nghĩa là:

i) Tạo ra một biến tạm X

ii) Sinh mã trung gian + (E1, T1, X)

iii) Truyền thuộc tính của X sang E (<expr> là vế trái luật sinh)

Các hành vi ngữ nghĩa đó được thực hiện như sau

X := Newtemp ();

tạo dạng mã bộ tứ: + '('E1','T1','X ')'

E := X

4- <assign stmt> → id := <expr>

sinh ra AS (NAME (id), E,)

E là trị ngữ nghĩa của <expr>

4. Kiểm tra kiểu dữ liệu
Hệ thống kiểu

- Muốn thiết kế bộ kiểm tra kiểu dữ liệu cho một ngôn ngữ cần phải dựa vào
  - Các thông tin về cấu trúc cú pháp của ngôn ngữ,
  - Đặc điểm về kiểu
  - Các quy tác cho phép gán kiểu
- Kiểu của cáu trúc ngôn ngữ được xác định bằng biểu thức kiểu. Biểu thức kiểu là kiểu cơ bản hoặc được hình thành khi áp dụng toán tử.

4. Kiểm tra kiểu dữ liệu
Biểu thức kiểu

PHENIKAA UNIVERSITY
Realizing your potential

- Biểu thức kiểu bao gồm:

1. Kiểu cơ sở là một biểu thức kiểu: boolean, char, integer, real.
   - Ngoài ra còn có các kiểu cơ sở đặc biệt như: kiểu type_error: chỉ ra một lỗi trong quá trình kiểm tra kiểu;
   - kiểu void, "không có giá trị", cho phép kiểm tra kiểu đối với lệnh.
2. Vì biểu thức kiểu có thể được đặt tên, tên kiểu là một biểu thức kiểu.

4. Kiểm tra kiểu dữ liệu
Biểu thức kiểu

3. Cấu trúc kiểu là một biểu thức kiểu, các cấu trúc bao gồm:

a. **Mảng (array):** Nếu T là một biểu thức kiểu thì array(I, T) là một biểu thức kiểu. Một mảng có tập chỉ số I và các phần tử có kiểu T.

b. **Tích (products):** Nếu T1, T2 là biểu thức kiểu thì tích Decas T1* T2 là biểu thức kiểu.

c. **Mẩu tin (records):** Là cấu trúc bao gồm một bộ các tên trường, kiểu trường.

d. **Con trỏ (pointers):** Nếu T là một biểu thức kiểu thì pointer(T) là một biểu thức kiểu T.

e. **Hàm (functions):** Một cách toán học, hàm ánh xạ các phần tử của tập xác định (domain) lên tập giá trị (range). Một hàm là một biểu thức kiểu D → R

4. Kiểm tra kiểu dữ liệu
Biểu thức kiểu

PHENIKAA UNIVERSITY
Realizing your potential

- Hệ thống kiểu:
  - Hệ thống kiểu là một bộ sưu tập các quy tắc để gán các biểu thức kiểu vào các phần của một chương trình.
- Kiểm tra kiểu tĩnh và kiểm tra kiểu động:
  - Kiểm tra được thực hiện bởi chương trình dịch được gọi là kiểm kiểu tĩnh.
  - Kiểm tra được thực hiện trong khi chạy chương trình đích gọi là kiểm tra kiểu động.
  - Trong thực tế có một số trường hợp chỉ kiểm tra kiểu động. Ví dụ: Khi khai báo mảng nhưng chưa sử dụng.

4. Kiểm tra kiểu dữ liệu
Biểu thức kiểu

- Phát hiện lỗi: Khi kiểm tra kiểu, bộ kiểm tra có khả năng phát hiện và khắc phục lỗi trong chương trình.
  - Khi tìm được lỗi bộ kt kiểu sẽ thông báo lỗi, phát hiện phạm vi lỗi xảy ra và có khả năng kiểm tra kiểu ở phần nhập còn lại.

4. Kiểm tra kiểu dữ liệu
Đặc tả bộ kiểm tra kiểu đơn giản

PHENIKAA UNIVERSITY
Realizing your potential

- Trong phần này chúng ta mô tả một bộ kiểm tra kiểu cho một ngôn ngữ đơn giản trong đó kiểu của mỗi một danh biểu phải được khai báo trước khi sử dụng.
- Bộ kiểm tra kiểu là một lược đồ dịch mà nó tổng hợp kiểu của mỗi biểu thức từ kiểu của các biểu thức con của nó.
- Xét một văn phạm cho ngôn ngữ đơn giản:
  - Văn phạm sau sinh ra một chương trình, biểu diễn bởi một ký hiệu chưa kết thúc P chứa một chuỗi các khai báo D và một biểu thức đơn giản E

$$P \rightarrow D \;;\; E$$

$$D \rightarrow D \;;\; D \mid \textbf{id} : T$$

$$T \rightarrow \textbf{char} \mid \textbf{integer} \mid \textbf{array[num] of } T_1 \mid \uparrow T_1$$

$$E \rightarrow \textbf{literal} \mid \textbf{num} \mid \textbf{id} \mid E_1 \textbf{ mod } E_2 \mid E_1 \textbf{ [} E_2 \textbf{]} \mid E_1 \uparrow$$

- Các kiểu cơ sở: char, integer và type-error
- Mảng bắt đầu từ 1. Chẳng hạn array[256] of char là biểu thức kiểu (1...256, char)
- Kiểu con trỏ ↑T là một biểu thức kiểu pointer(T).

4. Kiểm tra kiểu dữ liệu
Đặc tả bộ kiểm tra kiểu đơn giản

PHENIKAA UNIVERSITY
Realizing your potential

- Ta có lược đồ dịch để lưu trữ kiểu của một danh biểu

P → D ; E	
D → D ; D	
D → **id** : T	*{addtype(id.entry, T.type) }*
T → **char**	*{T.type := char }*
T → **integer**	*{T.type := integer }*
T → ↑T_1	*{T.type := pointer(T_1.type) }*
T → **array[num] of** T_1	*{T.type := array(1...num.val, T_1.type) }*

4. Kiểm tra kiểu dữ liệu
Đặc tả bộ kiểm tra kiểu đơn giản

PHENIKAA UNIVERSITY
Realizing your potential

- Kiểm tra cho biểu thức: Lược đồ dịch cho kiểm tra kiểu của biểu thức như sau:

E → **literal**	{*E.type := char* }
E → **num**	{*E.type := integer* }
E → **id**	{*E.type := lookup(id.entry)* }
E → E_1 **mod** E_2	{*E.type := if* E_1*.type = integer and* E_2*.type = integer then integer else type_error* }
E → E_1[E_2]	{*E.type := if* E_2*.type = integer and* E_1*.type = array(s,t) then t else type_error* }
E → E_1↑	{ *E.type := if* E_1*.type = pointer(t) then t else type_error* }

4. Kiểm tra kiểu dữ liệu
Đặc tả bộ kiểm tra kiểu đơn giản

PHENIKAA UNIVERSITY
Realizing your potential

- Kiểm tra cho biểu thức: Lược đồ dịch cho kiểm tra kiểu của biểu thức như sau:

E → **literal**	{*E.type := char* }
E → **num**	{*E.type := integer* }
E → **id**	{*E.type := lookup(id.entry)* }
E → E_1 **mod** E_2	{*E.type := if* E_1*.type = integer and* E_2*.type = integer then integer else type_error* }
E → $E_1[E_2]$	{*E.type := if* E_2*.type = integer and* E_1*.type = a then t else type_error* }
E → $E_1\uparrow$	{ *E.type := if* E_1*.type = pointer(t) then t else type_error* }

1. Quy tắc ngữ nghĩa phát biểu rằng hằng số có kiểu tc literal và num thì có kiểu là char và integer

E → **literal** {E. type := char}

E → **num** {E. type := integer}

2. Chúng ta dùng hàm lookup (e) để tìm kiểu dữ liệu đ trong bảng danh biểu tại vị trí e. Khi một danh biểu xuất hiệ biểu thức thì kiểu dữ liệu của danh biểu đã được khai báo, sẽ đ thấy và được gán vào kiểu thuộc tính. Ví dụ:

E → **id** {E. type := lookup (**id**. entry)}

4. Kiểm tra kiểu dữ liệu
Đặc tả bộ kiểm tra kiểu đơn giản

PHENIKAA UNIVERSITY
Realizing your potential

E → **literal**	{*E.type := char* }
E → **num**	{*E.type := integer* }
E → **id**	{*E.type := lookup(id.entry)* }
E → E_1 **mod** E_2	{*E.type := if* E_1*.type = integer and* E_2*.type = integer then integer else type_error* }
E → E_1[E_2]	{*E.type := if* E_2*.type = integer and* E_1*.type = array(s,t) then t else type_error* }
E → E_1↑	{ *E.type := if* E_1*.type = pointer(t) then t else type_error* }

3. Biểu thức được hình thành bằng việc áp dụng toán tử mod cho hai biểu thức có kiểu là integer, thì sẽ có kiểu là integer. Ngược lại sẽ có kiểu type – error:

E → E1 **mod** E2 {E. type := **if** (E1. type = integer) and
(E2. type = integer) **then** integer
else type - error}

4. Khi tham chiếu phần tử E1 [E2] của dãy E1, E2 là biểu thức chỉ số có kiểu là integer. Như vậy kết quả sẽ có kiểu là t, với t là kiểu array (s, t) của E1.

E → E1 [E2] {E. type := **if** (E2. type = integer) and
(E1. type = array (s, t)) **then** t **else** type - error}

5. Trong biểu thức, toán tử ↑ đứng sau toán hạng thuộc kiểu con trỏ chính là đối tượng dữ liệu được con trỏ chỉ đến. Như vậy E ↑ có kiểu dữ liệu t, là kiểu của đối tượng dữ liệu được con trỏ E chỉ đến.

E → E1 ↑ {E. type := **if** E1. type =pointer (t) **then** t
else type - error}

4. Kiểm tra kiểu dữ liệu
Đặc tả bộ kiểm tra kiểu đơn giản

PHENIKAA UNIVERSITY
Realizing your potential

Kiểm tra kiểu cho các phát biểu:

- Những phát biểu không có kiểu trong ngôn ngữ thì bộ kiểm tra kiểu dùng kiểu **void** gán cho chúng.
- Nếu xảy ra lỗi trong quá trình kiểm tra thì bộ kiểm tra kiểu gán **type-error**
- Xét việc kiểm tra kiểu cho phát biểu gán, điều kiện và vòng lặp. Các phát biểu cách nhau bởi dấu ";"

4. Kiểm tra kiểu dữ liệu
Đặc tả bộ kiểm tra kiểu đơn giản

PHENIKAA UNIVERSITY
Realizing your potential

- Ta có lược đồ dịch cho kiểu dữ liệu của các phát biểu như sau:

```
P → D; S

S → id := E {S. type := if id. type = E. type then void
             else type - error}

S → if E then S1 {S. type := if E. type = boolean then
                             S1. type else type - error}

S → while E do S1 {S. type := if E. type = boolean then
                              S1. type else type - error}

S → S1; S2 {S. type := if (S1. type = void) and
                       (S2. type = void) then void
                       else type - error}
```

4. Kiểm tra kiểu dữ liệu
Đặc tả bộ kiểm tra kiểu đơn giản

Kiểm tra kiểu của hàm: kiểm tra kiểu của hàm có đối số được mô tả bằng luật sinh: E → E (E).

- Lược đồ dịch cho kiểm tra kiểu cho một áp dụng hàm là:

$$E \rightarrow E_1\ (E_2) \quad \{E.type := \textit{if}\ E_2.type = s\ \textit{and}\ E_1.type = s \text{ -> } t\ \textit{then}\ t\ \textit{else}\ \textit{type\_error}\}$$

4. Kiểm tra kiểu dữ liệu
Sự tương đương của biểu thức kiểu

- Tồn tại những vấn đề không rõ ràng này sinh khi kiểm tra hai biểu thức tương đương với nhau →Phải có quy tắc rõ ràng để xác định hai biểu thức khi nào tương đương với nhau.
- Khi có tính tương đương về kiểu và sự biểu diễn ta cần xét cả hai vấn đề:

4. Kiểm tra kiểu dữ liệu
Sự tương đương của biểu thức kiểu

PHENIKAA UNIVERSITY
Realizing your potential

Vấn đề 1: Sự tương đương về cấu trúc của biểu thức kiểu

- Hai biểu thức kiểu được gọi là tương đương cấu trúc nếu cấu trúc của chúng đồng nhất, được tượng trưng bằn một nút giống nhau trên cây.
- Giải thuật kiểm tra tính tương đương cấu trúc

```
function sequiv (s, t): boolean;
    begin
        if s và t cùng một kiểu cơ bản then true
            else if s = array (s1, s2) and t = array (t1, t2) then
                return sequiv (s1, t1) and sequiv (s2, t2)
            else if s = s1 × s2 and t = t1 × t2 then
                return sequiv (s1, t1) and sequiv (s2, t2)
            else if s = pointer (s1) and t = pointer (t1) then
                return sequiv (s1, t1)
            else if s = s1 → s2 and t = t1 → t2 then
                return sequiv (s1, t1) and sequiv (s2, t2)
                else return false;
    end;
```

4. Kiểm tra kiểu dữ liệu
Sự tương đương của biểu thức kiểu

PHENIKAA UNIVERSITY
Realizing your potential

Vấn đề 1: Sự tương đương về cấu trúc của biểu thức kiểu

- Ví dụ: Cách mã hóa các biểu thức kiểu của trình biên dịch C do D.M Ritchie viết

```
                    char
                freturns(char )
        pointer (freturns(char))
        array (pointer (freturns (char))
```

- Biểu thức kiểu xét cho các kiểu con trỏ, hàm và dãy pointer (t) xác định kiểu con trỏ cho t. freturns (t) xác định hàm của một đối số, trả về đối tượng dữ liệu của t.
- Mỗi một biểu thức trên sẽ được mã hóa bởi các bít như sau:

Bộ kiến thiết kiểu	Mã hóa
pointer	01
array	10
freturns	11

Các kiểu cơ bản của ngôn ngữ C được John [1979] mã hóa bằng 4 bit

Kiểu cơ bản	Mã hóa
boolean	0000
char	0001
integer	0010
real	0011

4. Kiểm tra kiểu dữ liệu
Sự tương đương của biểu thức kiểu

PHENIKAA UNIVERSITY
Realizing your potential

Vấn đề 2: Tên cho một biểu thức kiểu

- Trong một số ngôn ngữ, kiểu dữ liệu được đặt tên.

```
type  link = ↑ cell;
var   next : link;
      last : link;
      p    : ↑cell;
      q, r : ↑cell;
```

- Danh biểu link được khai báo là tên của kiểu ↑cell. Vấn đề đặt ra là next, last, p, q, r có kiểu giống nhau hay không? Câu trả lời phụ thuộc vào sự cài đặt.
- Hai biểu thức kiểu là tương đương tên nếu tên của chúng giống nhau.
- Theo quan niệm tương đương tên thì last và next có cùng kiểu; p, q và r có cùng một kiểu nhưng next và p có kiểu khác nhau

5. Chuyển đổi kiểu

- Khi thực hiện một chương trình, đôi khi trình biên dịch phải thực hiện chuyển đổi kiểu:
  - Ví dụ: Xét biểu thức x + i trong đó x có kiểu real và i có kiểu integer.
  - Vì biểu diễn các số nguyên, số thực khác nhau trong máy tính do đó các chỉ thị máy khác nhau được dùng cho số thực và số nguyên.
  - Trình biên dịch có thể thực hiện việc chuyển đổi kiểu để hai toán hạng có cùng kiểu khi phép toán cộng xảy ra.
  - Bộ kiểm tra kiểu trong trình biên dịch có thể được dùng để thêm các phép toán biến đổi kiểu vào trong biểu diễn trung gian của chương trình nguồn. Chẳng hạn ký hiệu hậu tố của x + i có thể là: x i inttoreal real+
  - Trong đó: inttoreal đổi số nguyên i thành số thực, real+ thực hiện phép cộng các số thực.

5. Chuyển đổi kiểu

Vấn đề 1: Sự áp đặt toán tử (ép kiểu)

- Sự chuyển đổi từ kiểu này sang kiểu khác được gọi là ẩn nếu nó được làm một cách tự động bởi chương trình dịch. Chuyển đổi kiểu ẩn còn gọi là ép buộc chuyển đổi kiểu (coercions).
- Việc chuyển đổi là tường minh, nếu lập trình viên phải tự thực hiện.
  - Ví dụ: viết hàm làm tròn.
- Quy tắc kiểm tra cho việc ép kiểu:

Luật sinh	Luật ngữ nghĩa
E → **num**	E. type := integer
E → **num**. **num**	E. type := real
E → **id**	E. type := lookup (**id**. entry)
E → E1 **op** E2	E. type := **if** (E1. type = integer) and (E2. type = integer) **then** integer **else if** (E1. type = integer) **and** (E2. type = real) **then** real **else if** (E1. type = real) **and** (E2. type = integer) **then** real **else if** (E1. type = real) **and** (E2. type = real) **then** real **else** type - error

5. Chuyển đổi kiểu

Vấn đề 1: Sự áp đặt toán tử (ép kiểu)

- Chú ý: Việc ép kiểu ảnh hưởng thời gian thực thi của thuật toán.
- → Khắc phục: Việc chuyển đổi được thực hiện ngay trong thời gian dịch.

6. Xử lý ngữ nghĩa cho phát biểu Goto

- Khi một lệnh goto L xuất hiện trước một phát biểu có tên L thì chúng ta nói rằng đó là sự tham khảo trước.
- Trong trường hợp này, khi sinh ra một tứ ta không thể điền chỉ số mà mã JMP chuyển sự điều khiển đến, đó đó một từ JMP có dạng JMP(0,0,0) cho đến khi quá trình phân tích ngữ nghĩa và sinh mã trung gian gặp phát biểu có tên L
  → Xác định vị trí của L
- Nếu có nhiều phát biểu goto cùng tham khảo tới L thì việc xử lý phức tạp hơn.
- Phương pháp giải quyết: liên kết tất cả các bộ tứ ứng với goto L lại thành một danh sách liên kết ngược và cất chỉ số tương ứng với goto cuối cùng vào bảng danh biểu.

6. Xử lý ngữ nghĩa cho phát biểu Goto

- Khi một lệnh goto L xuất hiện trước một phát biểu có tên L thì chúng ta nói rằng đó là sự tham khảo trước.
- Trình biên dịch cập nhật lại địa chỉ và ghi 1

phát biểu	địa chỉ	mã bộ tứ
...		
goto L	(10)	JMP (0,0,0)
...		
goto L	(50)	JMP (10,0,0)
...		
goto L	(90)	JMP (50,0,0)
...		
L: x := x + 1;	(120)	

Bảng 6.8 *Bảng lưu giữ tên phát biểu và chỉ số đầu danh sách liên kết*

Tên p/b	Địa chỉ	Định nghĩa
L	90	0

Tên p/b	Địa chỉ	Định nghĩa
L	120	1

7. Bài tập

1. Viết các biểu thức kiểu cho các kiểu dữ liệu sau đây:

a) Một mảng của các con trỏ có kích thước từ 1 đến 100, trỏ đến đối tượng các số thực.

b) Mảng 2 chiều của các số nguyên, hàng có kích thước từ 0 đến 9, cột có chỉ số từ -10 đến 10.

c) Các hàm mà miền định nghĩa là các hàm với các đối số nguyên, trị là con trỏ trỏ đến các số nguyên và miền xác định của nó là các mẫu tin chứa số nguyên và ký tự.

2. Giả sử có một khai báo C như sau:

```
typedef struct {
int a, b ;
} CELL, * PCELL ;
CELL foo [ 100 ] ;
PCELL bar (x, y) int x ; CELL y { ..... }
```

Viết các biểu thức kiểu cho các kiểu dữ liệu foo và bar.

7. Bài tập

3. Cho văn phạm sau đây định nghĩa chuỗi của các chuỗi ký tự:

P → D ; E

D → D ; D | id : T

T → list of T | char | integer

E → (L) | literal | num | id

L → E , L | E

Hãy viết các quy tắc biên dịch để xác định các biểu thức kiểu (E) và list (L).

4. Cho văn phạm sau. Sinh mã trung gian cho biểu thức T=(P+Q)*(E+F)

<assign stmt> → id := <expr>

<expr> → < expr> + <term> | <term>

<term> → < term> * <factor> | <factor>

<factor> → id | (< expr>)

7. Bài tập

5. Cho văn phạm sau. Hãy sinh mã trung gian cho biểu thức: 4*5+3n

L → E**n**

E → E_1 + T

E → T

T → T_1 * F

T → F

F → (E)

F → **digit**