Bài 10
Phân tích ngữ nghĩa

ONE LOVE. ONE FUTURE.

Nội dung

- **Những vấn đề ngữ nghĩa**
- **Bảng ký hiệu**
  - Luật về phạm vi ảnh hưởng của biến
  - Các sơ đồ dịch để xây dựng bảng ký hiệu
- **Kiểm tra kiểu (Type checking)**
  - Hệ thống kiểu trong ngôn ngữ lập trình
  - Đặc tả một bộ kiểm tra kiểu
  - Chuyển đổi kiểu

Phân tích ngữ nghĩa

- Tìm ra các lỗi sau giai đoạn phân tích cú pháp
  - Kiểm tra sự tương ứng về kiểu
  - Kiểm tra sự tương ứng giữa việc sử dụng hàm, biến với khai báo của chúng
  - Xác định phạm vi ảnh hưởng của các biến trong chương trình
- Phân tích ngữ nghĩa thường sử dụng cây cú pháp

Bảng ký hiệu và phạm vi

- Phạm vi là gì
- Quản lý phạm vi tĩnh và động
- Những vấn đề liên quan đến phạm vi
- Bảng ký hiệu
- Xây dựng bảng ký hiệu

Quản lý phạm vi

- Đây là vấn đề liên quan đến sự phù hợp giữa khai báo và sử dụng của mỗi định danh.
- Phạm vi ảnh hưởng của mỗi định danh là phần chương trình có thể truy cập tới định danh đó.
- Một định danh có thể tham chiếu các đối tượng khác nhau trong các phạm vi khác nhau của chương trình. Phạm vi của hai định danh giống nhau không được giao nhau

Phạm vi tĩnh và động

- Phần lớn các ngôn ngữ quản lý phạm vi theo kiểu tĩnh. Thông tin phạm vi chỉ phụ thuộc văn bản chương trình. Luật phạm vi gần nhát được áp dụng
- Một số ít ngôn ngữ cho phép quản lý phạm vi động, quản lý phạm vi khi thực hiện chương trình (Lisp, Snobol, Perl khi dùng một số từ khóa đặc biệt)
- Quản lý phạm vi động nghĩa là khi một định danh được tham chiếu, chương trình dịch sẽ tham chiếu vào stack chứa các bản hoạt động để tìm ra thông tin về ký hiệu đó

Ví dụ

QL phạm vi tĩnh

```
1  const int b = 5;
2  int foo()
3  {
4     int a = b + 5;
5     return a;
6  }
7
8  int bar()
9  {
10    int b = 2;
11    return foo();
12 }
13
14 int main()
15 {
16    foo(); // returns 10
17    bar(); // returns 10
18    return 0;
19 }
```

QL phạm vi động

```
1  const int b = 5;
2  int foo()
3  {
4     int a = b + 5;
5     return a;
6  }
7
8  int bar()
9  {
10    int b = 2;
11    return foo();
12 }
13
14 int main()
15 {
16    foo(); // returns 10
17    bar(); // returns 7
18    return 0;
19 }
```

Những vấn đề về quản lý phạm vi

- Khi xét một khai báo chứa một định danh, liệu đã tồn tại định danh đó trong phạm vi hiện hành chưa?
- Khi sử dụng một định danh, liệu nó đã được khai báo chưa? Nếu nó đã được khai báo (theo luật phạm vi gần nhất) liệu khai báo có tương thích với sử dụng không?

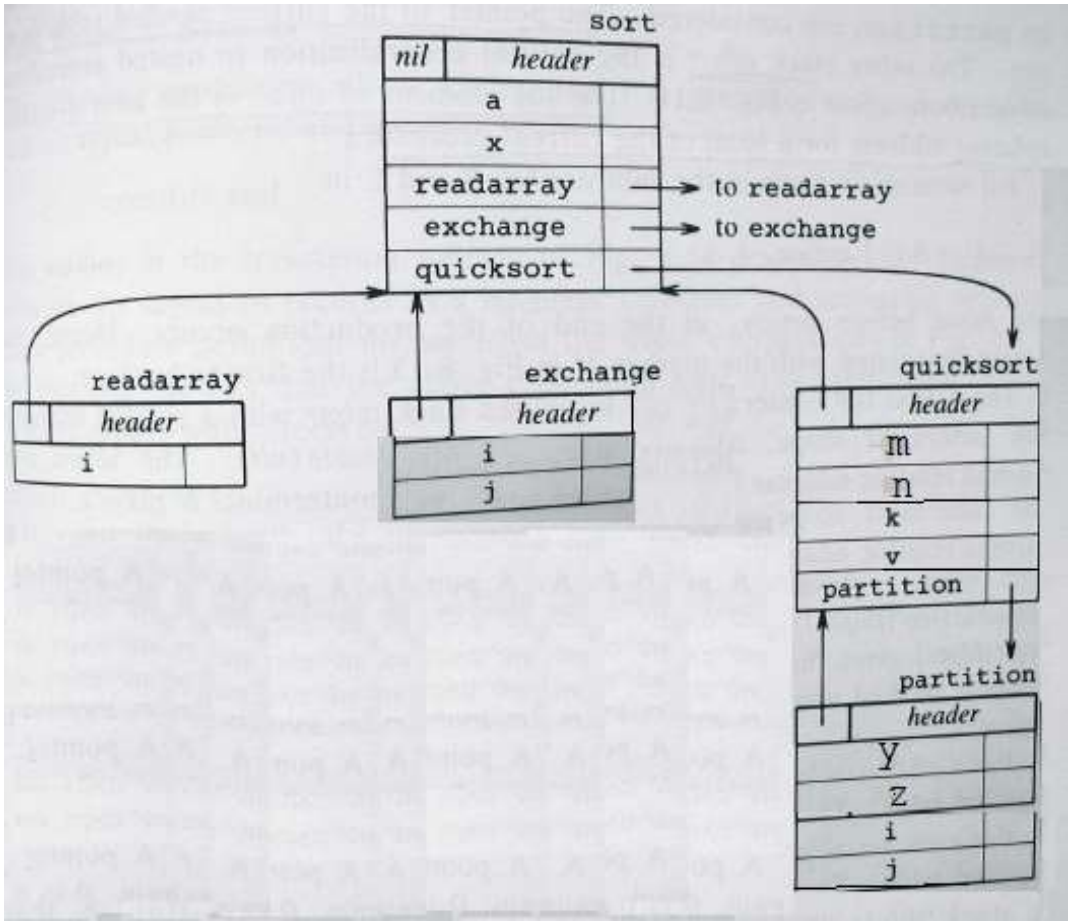
Bảng ký hiệu (Symbol table)

- Một cấu trúc dữ liệu cho phép theo dõi quan hệ hiện hành của các định danh (để kiểm tra ngữ nghĩa và sinh mã một cách hiệu quả)
- Phần quan trọng trong phân tích ngữ nghĩa là theo dõi các hằng/biến/kiểu/hàm/thủ tục xem có phù hợp với khai báo của chúng không?
- Khi thêm một định danh vào bảng ký hiệu, cần ghi lại thông tin trong khai báo của định danh đó.

Ngôn ngữ có cấu trúc khối

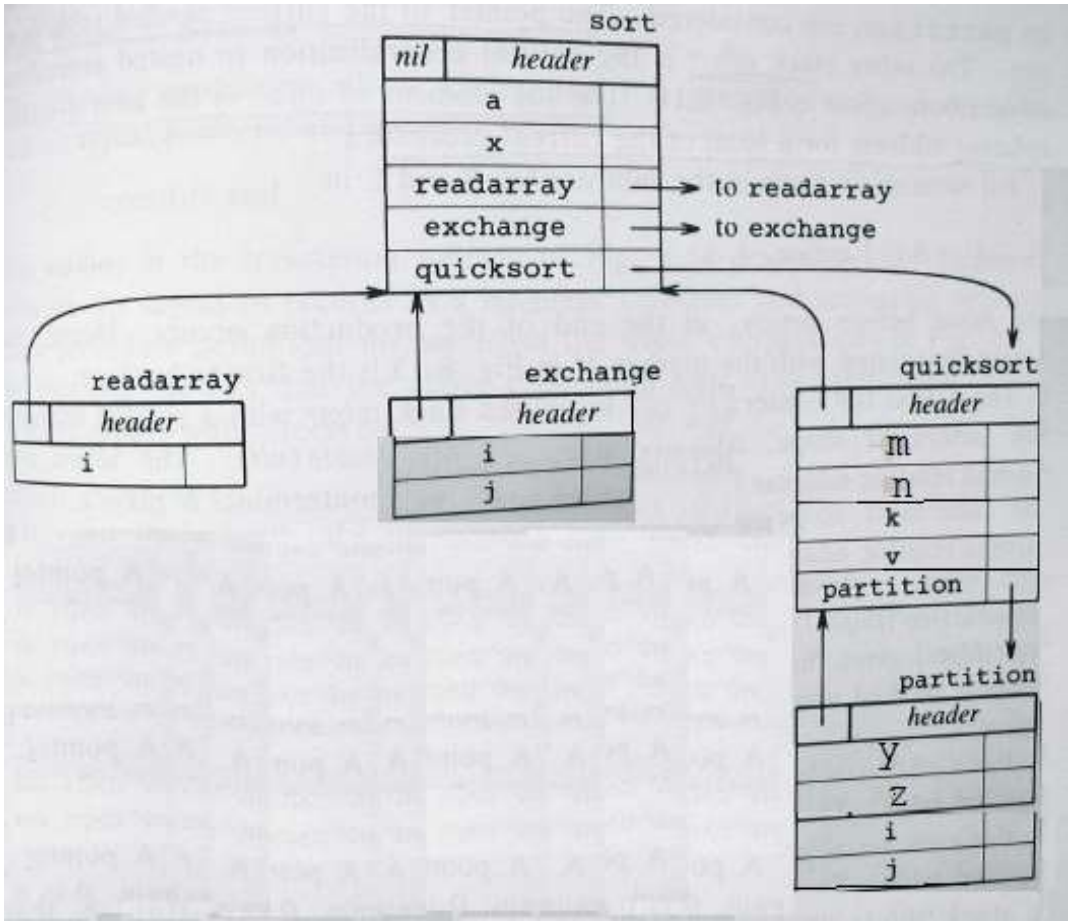
- Khối trong ngôn ngữ lập trình là tập các cấu trúc ngôn ngữ có chứa khai báo
- Một ngôn ngữ là có cấu trúc khối nếu
  - Các khối được lồng bên trong những khối khác
  - Phạm vi của khai báo trong mỗi khối là chính khối đó và các khối chứa trong nó
- Luật lồng nhau gần nhất
  - Cho nhiều khai báo của cùng một tên. Khai báo có hiệu lực là khai báo nằm trong khối gần nhất

Giải pháp nhiều bảng ký hiệu




```
program sort;
  var a : array [0..10] of integer;
      x : integer;
  procedure readarray;
   var i : integer;
   begin … end;
  procedure exchange(var i:integer; var j :
integer);
   begin x := a(.i.); a(.i.) := a(.j.);
         a(.j.) := x
   end;
  procedure quicksort(m:integer;n:integer);
   var k:integer; v : integer;
   function partition(y:integer;z:integer):
   integer
         var i:integer; j : integer;
         begin …
           call exchange(i, j; …
         end
   begin
         if (n > m) then
          begin
            i := partition(m, n);
            call quicksort(m, i - 1);
            call quicksort(i + 1, n);
          end
   end;
  BEGIN
      …
      call quicksort(1, 9);
  END.
```

Giải pháp nhiều bảng ký hiệu (tiếp)



Bảng ký hiệu cho sort:

- Chứa các định danh xuất hiện trong sort
  a,x là các biến được khai báo trong chương trình; thông
  tin: loại đối tượng, kiểu, địa chỉ lưu trữ
  readarray, *exchange* và *quicksort* là các hàm và thủ tục; thông tin: loại đối tượng, kiểu trả về, *danh sách tham số hình thức* (con trỏ đầu danh sách)

Bảng ký hiệu cho readarray

i: biến địa phương

Bảng ký hiệu cho exchange

i,j là các tham biến

Bảng ký hiệu cho quicksort

m, n là các tham trị
k,v là các biến địa phương
con trỏ tới danh sách tham số hình thức của *partition*

Bảng ký hiệu cho partition

y,z là các tham trị
i,j là các biến địa phương

Bảng ký hiệu cho phạm vi toàn cục

Những định danh chuẩn chỉ tên của các hàm, thủ tục mẫu

Xây dựng bảng ký hiệu

- Những thao tác cần thiết
- Vào phạm vi (enter scope): tạo ra một phạm vi mới trong các phạm vi lồng nhau
- Xử lý khai báo: Thêm một định danh vào bảng ký hiệu của phạm vi hiện hành
- Xử lý việc sử dụng định danh: Kiểm tra xem định danh có xuất hiện trong bảng ký hiệu của
  - Phạm vi hiện hành
  - Các phạm vi từ phạm vi hiện hành ra ngoài theo luật phạm vi gần nhất
- Ra khỏi phạm vi (exit scope): ra khỏi phạm vi hiện hành

Các luật về phạm vi lồng nhau

- Toán tử insert vào bảng ký hiệu không được ghi đè những khai báo trước
- Toán tử lookup vào bảng ký hiệu luôn luôn tham chiếu luật phạm vi gần nhất
- Toán tử delete chỉ được xóa khai báo gần nhất

Cấu trúc dữ liệu cho bảng ký hiệu

- Danh sách liên kết không sắp thứ tự: Thích hợp cho việc phân tích các chương trình sử dụng số lượng biến nhỏ
- Danh sách liên kết sắp thứ tự. Thao tác bổ sung tốn kém nhưng thao tác tìm kiếm lại nhanh chóng hơn
- Cây nhị phân tìm kiếm
- Bảng băm: thường được dùng nhất

Xây dựng trong giai đoạn phân tích cú pháp

- Chỉ có thể bắt đầu nhập thông tin vào bảng ký hiệu từ khi phân tích từ vựng nếu ngôn ngữ lập trình không cho khai báo tên trùng nhau.
- Nếu cho phép các phạm vi, bộ phân tích từ vựng chỉ trả ra tên của định danh cùng với token
  - Định danh được thêm vào bảng ký hiệu khi vai trò cú pháp của định danh được phát hiện

Khái niệm kiểm tra kiểu

- Kiểm tra xem chương trình có tuân theo các luật về kiểu của ngôn ngữ không
- Trình biên dịch quản lý thông tin về kiểu
- Việc kiểm tra kiểu được thực hiện bởi bộ kiểm tra kiểu (type checker), một bộ phận của trình biên dịch

Ví dụ về kiểm tra kiểu

- Toán tử % của C chỉ thực hiện khi các toán hạng là số nguyên (Ngôn ngữ KPL không có toán tử này, nhưng với phép chia ngầm định là phép chia nguyên, có thể dễ dàng tính ra số dư của phép chia)
- Chỉ có mảng mới có chỉ số và kiểu của chỉ số phải nguyên
- Một hàm phải có một số lượng tham số nhất định và các tham số phải đúng kiểu

Kiểm tra kiểu

- Có hai phương pháp tĩnh và động
- Phương pháp áp dụng trong thời gian dịch là tĩnh
- Trong các ngôn ngữ như C hay Pascal, kiểm tra kiểu là tĩnh và được dùng để kiểm tra tính đúng đắn của chương trình trước khi nó đươc thực hiện
- Kiểm tra kiểu tĩnh cũng được sử dụng khi xác định dung lượng bộ nhớ cần thiết cho các biến
- Bộ kiểm tra kiểu được xây dựng dựa trên
  - Các biểu thức kiểu của ngôn ngữ
  - Bộ luật để định kiểu cho các cấu trúc

Biểu thức kiểu (Type Expression)

Biểu diễn kiểu của một cấu trúc ngôn ngữ. Cách cấu trúc tương tự biểu thức số học

Một biểu thức kiểu là một kiểu dữ liệu chuẩn hoặc được xây dựng từ các kiểu dữ liệu khác bởi cấu trúc kiểu (*Type Constructor)*

1. Kiểu dữ liệu cơ sở (int, real, boolean, char) là biểu thức kiểu
2. Biểu thức kiểu có thể liên hệ với một tên. Tên kiểu là biểu thức
3. Cấu trúc kiểu được ứng dụng vào các biểu thức kiểu tạo ra biểu thức kiểu

Cấu trúc kiểu

(a)Mảng (*Array*).Nếu T là biểu thực kiểu thì *array*(I,T) là biểu thức kiểu biểu diễn một mảng với các phần tử kiểu T và chỉ số trong miền I

Ví dụ : array [10] of integer có kiểu array(1..10,int);

(b) Tích Descarter Nếu T_1 và T_2 là các biểu thức kiểu thì tích Descarter T1× T2 là biểu thức kiểu

(c) Bản ghi (*Record)* Tượng tự như tích Descarter nhưng chứa các tên khác nhau cho các kiểu khắc nhau,

Ví dụ

```
struct
{
double r;
int i;
}
```

Có kiểu ((r x double) x (i x int))

Cấu trúc kiểu (tiếp)

(d) Con trỏ: Nêu T là biểu thức kiểu thì pointer(T) là biểu thức kiểu

(e) Hàm Nếu D là miền xác định và R là miền giá trị của hàm thì kiểu của nó được biểu diễn là

biểu_ thức : D : R.

Ví dụ hàm của C

int f(char a, b)

Có kiểu: *char* × *char* : *int*.

Hệ thống kiểu (Type System)

- Tập các luật để xây dựng các biểu thức kiểu trong những phần khác nhau của chương trình
- Được định nghĩa thông qua định nghĩa tựa cú pháp
- Bộ kiểm tra kiểu thực hiện một hệ thống kiểu
- Ngôn ngữ định kiểu mạnh: Chương trình dịch kiểm soát được hết các lỗi về kiểu

Đặc tả một bộ kiểm tra kiểu

- Ngôn ngữ đơn giản với mỗi tên được liên kết với một kiểu
- Văn phạm thuộc tính đơn giản cho biểu thức, với P: program, D: declaration, T: type, E: Expression đơn giản

  $P \rightarrow D;E$

  $D \rightarrow D;D \mid id : T$

  $T \rightarrow char \mid int \mid array[num] \text{ of } T \mid \uparrow T$

  $E \rightarrow literal \mid num \mid id \mid E \bmod E \mid E[E] \mid E\uparrow$

- Thuộc tính cần quan tâm ở đây là .type, có ở cả ký hiệukết thúc và không kết thúc

Thuộc tính

- Thuộc tính là khái niệm trừu tượng biểu diễn một đại lượng bất kỳ , chẳng hạn một số, một xâu, một vị trí trong bộ nhớ . . . .

- Thuộc tính được gọi là tổng hợp nếu giá trị của nó tại một nút trong cây được xác định từ giá trị của các nút con của nó.

- Thuộc tính kế thừa là thuộc tính tại một nút mà giá trị của nó được định nghĩa dựa vào giá trị nút cha và/hoặc các nút anh em của nó.

Định nghĩa tựa cú pháp (syntax directed definition)

Định nghĩa tựa cú pháp là dạng tổng quát của văn phạm phi ngữ cảnh để đặc tả cú pháp của ngôn ngữ vào.

- Mỗi ký hiệu của văn phạm liên kết với một tập thuộc tính ,
- Mỗi sản xuất $A \rightarrow \alpha$ liên hệ với một tập các quy tắc ngữ nghĩa để tính giá trị thuộc tính liên kết với những ký hiệu xuất hiện trong sản xuất. Tập các quy tắc ngữ nghĩa có dạng

$$b = f(c_1, c_2, \ldots\ldots, c_n)$$

f là một hàm và b thoả một trong hai yêu cầu sau:

- b là một thuộc tính tổng hợp của A và $c_1, \ldots, c_n$ là các thuộc tính liên kết với các ký hiệu trong vế phải sản xuất $A \rightarrow \alpha$
- b là một thuộc tính thừa kế một trong những ký hiệu xuất hiện trong α, và $c_1, \ldots, c_n$ là thuộc tính của các ký hiệu trong vế phải sản xuất $A \rightarrow \alpha$

Ví dụ một định nghĩa tựa cú pháp với thuộc tính tổng hợp

Sản xuất	Quy tắc ngữ nghĩa
L → E **return**	Print (E.val)
E → E1+T	E.val = E1.val + T.val
E → T	E.val = T.val
T → T1 * F	T.val = T1.val * F.val
T → F	T.val = F.val
F → (E)	F.val = E.val
F → **num**	F.val = num.Lexval

- Các ký hiệu E, T, F liên hệ với thuộc tính **tổng hợp** val
- Từ tố num có thuộc tính tổng hợp lexval (Được bộ phân tích từ vựng đưa ra)
- Định nghĩa tựa cú pháp này nhằm tính ra giá trị của biểu thức xuất phát từ giá trị lưu trữ trong bảng ký hiệu của các nhân tử là số

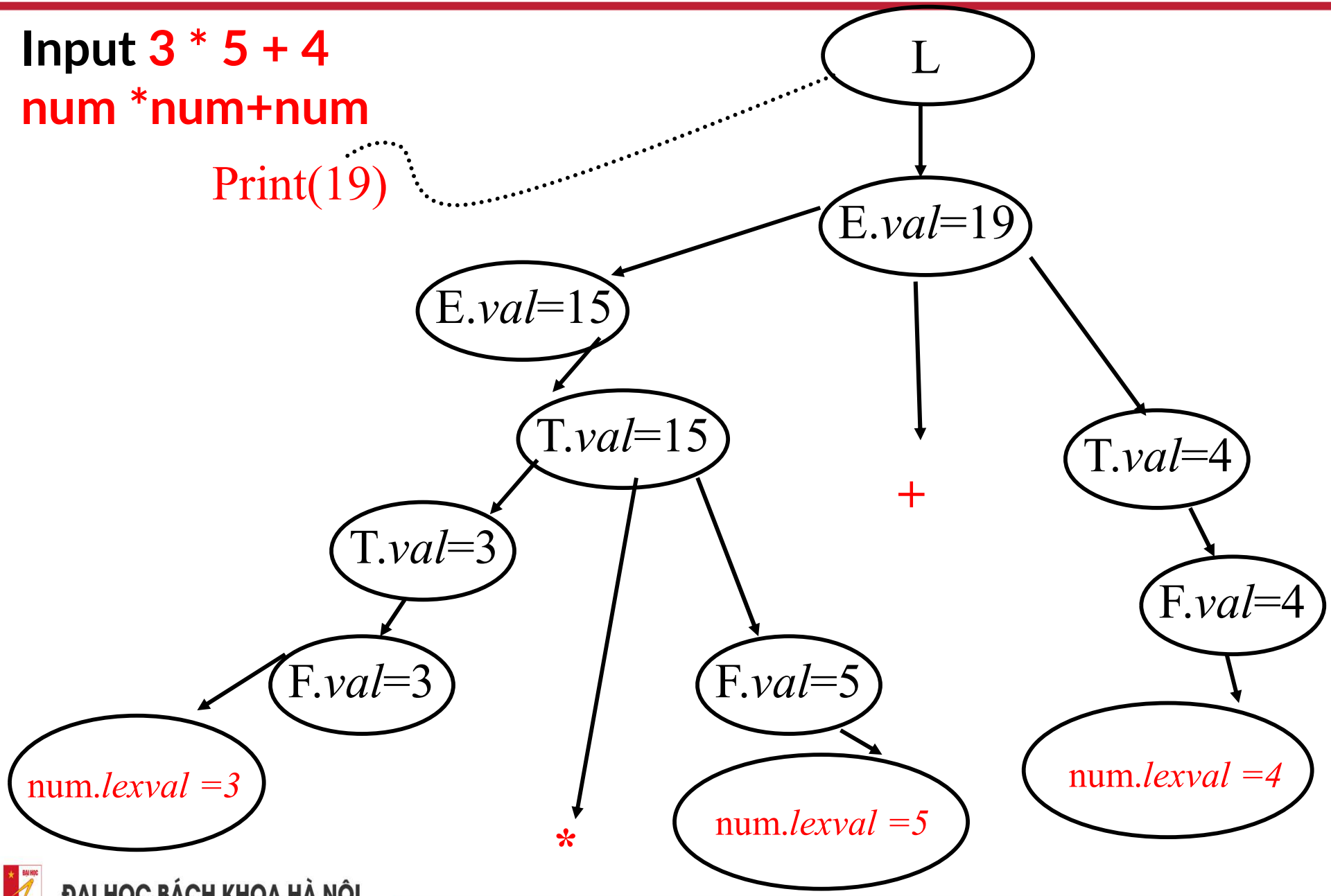
Cây phân tích cú pháp có chú giải

- Cây cú pháp chỉ ra giá trị các thuộc tính tại mỗi nút được gọi là ***cây cú pháp có chú giải***.

Ví dụ: Cây PTCPCCG có các thuộc tính val và lexval

Input 3 * 5 + 4

num *num+num



Ví dụ DTC với thuộc tính kế thừa

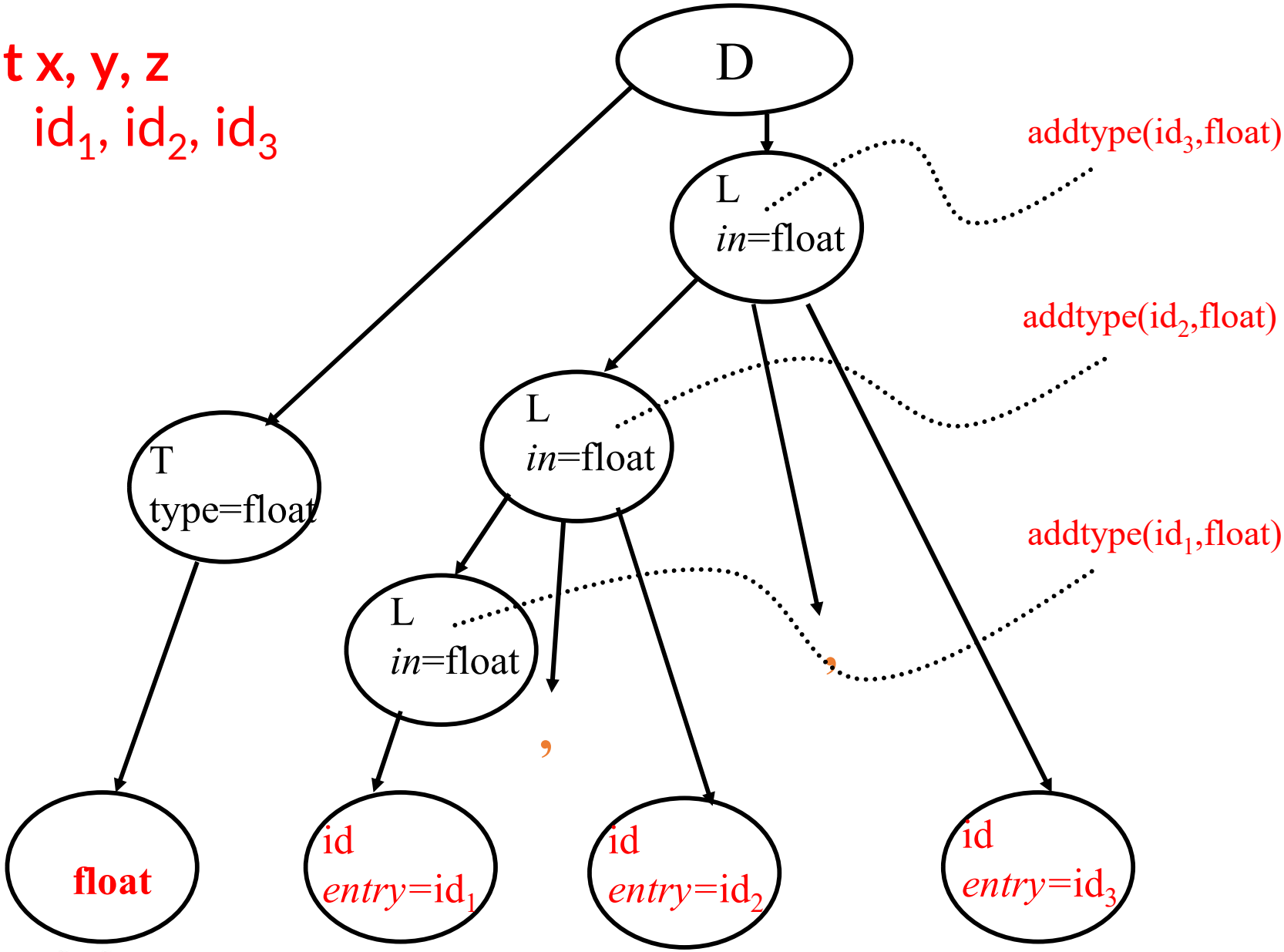
Sản xuất	Quy tắc ngữ nghĩa
$D \rightarrow T\ L$	L.in = T.type
$T \rightarrow int$	T.type = integer
$T \rightarrow float$	T.type = float
$L \rightarrow L_1\ ,\ id$	L1.in = L.in addtype(id.entry, L.in)
$L \rightarrow id$	addtype(id.entry, L.in)

- L.in là thuộc tính kế thừa, chỉ kiểu của danh sách biến L
- Kiểu của danh sách biến L được tính từ kiểu của T
- Kiểu trong thuộc tính L.in được truyền cho id cuối cùng và danh sách L1 gồm các id còn lại Thông tin về kiểu được lưu vào bảng ký hiệu.
- Thông tin kiểu được lưu cho tất cả các biến trong danh sách

Ví dụ: Cây PTCPCCG cho ví dụ 2

Input float x, y, z

id_1, id_2, id_3



Mô hình: Thêm thuộc tính kiểu cho định danh

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
$D \rightarrow \mathsf{id} : T$	$addtype(\mathsf{id}.entry, T.type)$
$T \rightarrow \mathsf{char}$	$T.type := char$
$T \rightarrow \mathsf{int}$	$T.type := int$
$T \rightarrow \uparrow T_1$	$T.type := pointer(T_1.type)$
$T \rightarrow \mathsf{array}[\mathsf{num}]\ \mathsf{of}\ T_1$	$T.type := array(1..\mathsf{num}.val, T_1.type)$

Ứng dụng: Thêm thông tin về kiểu của định danh trong KPL

Sản xuất	Xử lý
<VarDecl> ::= TK_IDENT SB_COLON <Type> SB_SEMICOLON	Trong hàm compileVardecl có lời gọi đến hàm: Type* compileType(void)
<Type> ::= KW_INTEGER	type = makeIntType();
<Type> ::= KW_CHAR	type = makeCharType();
<Type> ::= TK_IDENT	obj = checkDeclaredType(currentToken->string); type = duplicateType(obj->typeAttrs->actualType);

<Type> ::= KW_ARRAY SB_LSEL TK_NUMBER SB_RSEL KW_OF <Type>	arraySize = currentToken->value;
	elementType = compileType();
	type = makeArrayType(arraySize, elementType);

Mô hình: đánh giá thuộc tính kiểu cho biểu thức với văn phạm chỉ có 1 biến E

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
$E \rightarrow$ literal	$E.type := char$
$E \rightarrow$ num	$E.type := int$
$E \rightarrow$ id	$E.type := lookup(\text{id}.entry)$
$E \rightarrow E_1$ mod E_2	$E.type :=$ if $E_1.type = int$ and $E_2.type = int$ then int else $type\_error$
$E \rightarrow E_1[E_2]$	$E.type :=$ if $E_2.type = int$ and $E_1.type = array(s,t)$ then t else $type\_error$
$E \rightarrow E_1\uparrow$	$E.type :=$ if $E_1.type = pointer(t)$ then t else $type\_error$

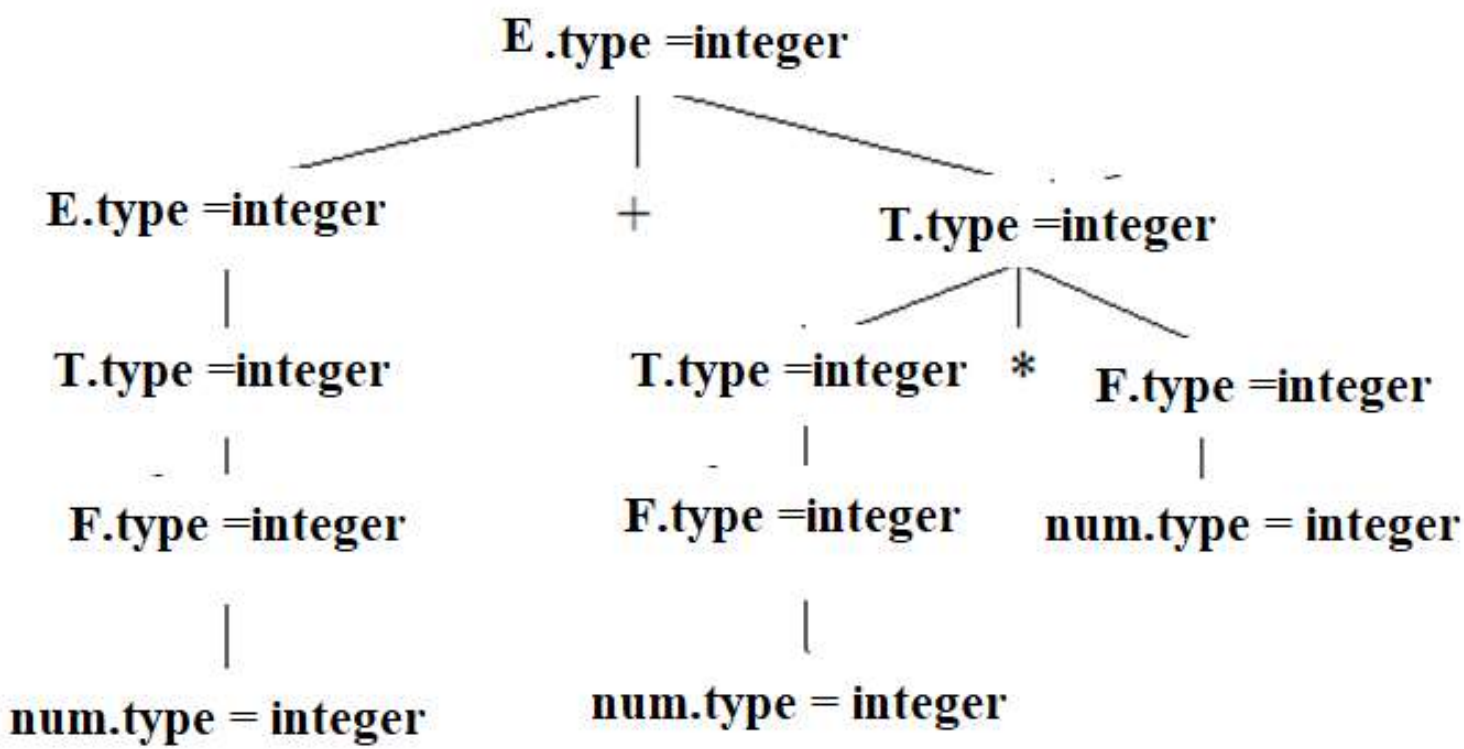
Mô hình: Xác định thuộc tính kiểu cho biểu thức với văn phạm 3 biến E, T, F

Sản xuất	Quy tắc ngữ nghĩa
E → E1+T	If E1.type= integer && T.type =integer then E.type = integer
E → T	E.Type = T.type
T → T1 * F	If T1.type= integer && F.type =ineger then T.type = integer
T → F	T.type= F.type
F → (E)	F.type= E.type
F → **num**	F.type = integer

- *Các ký hiệu E, T, F liên hệ với thuộc tính tổng hợp type*
- *Từ tố num có thuộc tính tổng hợp type là integer*

Ví dụ: xác định giá trị thuộctính tổng hợp type.

Input: 5+3*4



Ứng dụng: Xác định kiểu của factor trong KPL

Sản xuất	Xử lý

<Factor> ::= TK_CHAR	case TK_CHAR:
	  eat(TK_CHAR);
	  type = charType;
	  break;

<Factor> ::= TK_NUMBER	case TK_NUMBER:
	  eat(TK_NUMBER);
	  type = intType;
	  break;

<Factor> ::= SB_LPAR <Expression> SB_RPAR	case SB_LPAR:
	  eat(SB_LPAR);
	  type = compileExpression();
	  eat(SB_RPAR);
	  break;

Ứng dụng: Xác định kiểu của factor trong KPL

Sản xuất	Xử lý

<Factor> ::= TK_IDENT <Indexes>	case TK_IDENT:
	eat(TK_IDENT);
	switch (obj->kind) {
	case OBJ_CONSTANT:
	switch (obj->constAttrs->value->type) {
	case TP_INT:
	type = intType;
	break;
	case TP_CHAR:
	type = charType;
	break;
	default:
	break;
	}

Ứng dụng: Xác định kiểu của factor trong KPL

Sản xuất	Xử lý

<Factor> ::= TK_IDENT <Indexes>	case OBJ_VARIABLE:
	if (obj->varAttrs->type->typeClass == TP_ARRAY)
	type = compileIndexes(obj->varAttrs->type);
	else
	type = obj->varAttrs->type;
	break;
	case OBJ_PARAMETER:
	type = obj->paramAttrs->type;
	break;
	case OBJ_FUNCTION:
	compileArguments(obj->funcAttrs->paramList);
	type = obj->funcAttrs->returnType;
	break;

Ứng dụng: Xác định kiểu của expression trong KPL

- Kiểu của biểu thức được lấy là kiểu của factor đầu tiên.
- Nếu có các dấu phép toán, kiểu của factor đầu bắt buộc là nguyên,
- Các factor sau dấu phép toán cũng bắt buộc kiểm tra kiểu có là nguyên hay không
- Hàm trả về kiểu nguyên nếu biểu thức chứa ít nhất 1 toán tử.

Mô hình: kiểm tra kiểu của lệnh

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
$S \rightarrow \text{id} := E$	$S.type :=$ if id$.type = E.type$ then $void$ else $type\_error$
$S \rightarrow \text{if } E \text{ then } S_1$	$S.type :=$ if $E.type = boolean$ then $S_1.type$ else $type\_error$
$S \rightarrow \text{while } E \text{ do } S_1$	$S.type :=$ if $E.type = boolean$ then $S_1.type$ else $type\_error$
$S \rightarrow S_1; S_2$	$S.type :=$ if $S_1.type = void$ and $S_2.type = void$ then $void$ else $type\_error$

- Lệnh gán: Vế trái và vế phải phải cùng kiểu
- Lệnh if và while: Kiểu của điều kiện phải là Boolean
- Lệnh for : tương tự lệnh gán
- Một chương trình là đúng về kiểu nếu mọi lệnh đều đúng về kiểu

Ứng dụng: Kiểm tra kiểu cho lệnh gán của KPL

- Lệnh gán: Kiểu của vế trái và vế phải phải giống nhau
- Đánh giá: kiểu của vế trái

```
varType = compileLValue();
```

- Đánh giá: kiểu của vế phải

```
expType = compileExpression();
```

- Kiểm tra tương ứng kiểu

```
checkTypeEquality(varType, expType);
```

Ứng dụng: Kiểm tra kiểu cho lệnh if và while của KPL

1/Việc đầu tiên là kiểm tra tính đúng đắn về kiểu của điều kiện.

- Điều kiện của KPL thuộc loại đơn giản nhất, chỉ là biểu thức quan hệ
- Biểu thức quan hệ đúng về kiểu, thể hiện ở vế trái và vế phải phép so sánh phải cùng kiểu cơ sở.

2/ Các lệnh ở nhánh THEN và ELSE phải đúng về kiểu

3/Lệnh ở thân của vòng WHILE phải đúng về kiểu

Mô hình: kiểm tra kiểu của hàm

- Dùng khi phân tích khai báo hàm và sử dụng hàm
- D là danh sách tham số hình thức
- Elist là danh sách tham số thực sự

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
$D \rightarrow \mathsf{id} : T$	*addtype*(id.*entry*,*T.type*); $D.type := T.type$
$D \rightarrow D_1 ; D_2$	$D.type := D_1.type \times D_2.type$
$Fun \rightarrow \mathsf{fun}\ \ \mathsf{id}(D) : T ; B$	*addtype*(id.*entry*,*D.type*:*T.type*)
$B \rightarrow \{S\}$	
$S \rightarrow \mathsf{id}(EList)$	$E.type :=$ if *lookup*(id.*entry*)=$t_1 : t_2$ and *EList.type*= t then t_2 else *type_error*
$EList \rightarrow E$	$EList.type := E.type$
$EList \rightarrow EList, E$	$EList.type := EList_1.type \times E.type$

Ứng dụng :Thông tin kiểu của TS hình thức trong bảng ký hiệu của KPL

Sản xuất	Xử lý

<Param>::= TK_IDENT SB_COLON <BasicType>
<Param>::= KW_VAR TK_IDENT SB_COLON <BasicType>
	param = createParameterObject (currentToken->string, PARAM_VALUE, symtab->currentScope->owner);
	eat(SB_COLON);
	type = compileBasicType();
	param->paramAttrs->type = type;
	…..
	Xử lý cho tham biến tương tự

Ứng dụng: Nhập thông tin kiểu trả về của hàm vào bảng ký hiệu

Sản xuất	Xử lý

<FunDecl>::= KW_FUNCTION
TK_IDENT <Params> SB_COLON
<BasicType> SB_SEMICOLON
<Block> SB_SEMICOLON
	compileParams();
	…..
	returnType =
	compileBasicType();
	funcObj->funcAttrs-
	>returnType = returnType;

Ứng dụng: Tương ứng về số lượng giữa TS hình thức và TS thực sự

Sản xuất	Xử lý

<Arguments> ::= SB_LPAR <Expression> <Arguments2> SB_RPAR
<Arguments> ::= ε
 	
<Arguments2>::= SB_COMMA <Argument> <Arguments2>
<Arguments2>::= ε
(CT mẫu dùng chu trình thay vì gọi đệ quy)
	if (node == NULL)
	error(…);
	compileArgument(node->object);
	node = node->next;
	while (lookAhead->tokenType == SB_COMMA) {
	eat(SB_COMMA);
	if (node == NULL)
	error(…);
	compileArgument(node->object);
	node = node->next;
	}
	if (node != NULL)
	error(…);

Ứng dụng: Tương ứng kiểu giữa TS hình thức và TS thực sự (tham trị)

Sản xuất	Xử lý

<Argument>::=
<Expression>|<Lvalue>
	if (param->paramAttrs->kind
	== PARAM_VALUE) {
	type = compileExpression();
	checkTypeEquality(type,
	param->paramAttrs->type);}
	else {type = compileLValue();
	checkTypeEquality(type,
	param->paramAttrs->type);
	}

Hàm kiểm tra biểu thức kiểu tương đương

```
function sequiv(s, t): boolean;
begin
    if s và t là cùng kiểu dữ liệu chuẩn then
          return 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 x s2 and t  = t1 x 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;
```

Chuyển đổi kiểu

- Kiểu của x+i với

  x kiểu real

  i kiểu int

  Khi dịch sang lệnh máy, phép cộng với kiểu real và kiểu int có mã lệnh khác nhau
- Tùy ngôn ngữ và bộ luật chuyển đổi sẽ quy đổi các toán hạng về một trong hai kiểu

Mô hình: chuyển đổi kiểu trong biểu thức

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
$E \rightarrow$ num	$E.type := int$
$E \rightarrow$ num.num	$E.type := real$
$E \rightarrow$ id	$E.type := lookup(\text{id}.entry)$
$E \rightarrow E_1$ op E_2	$E.type :=$ if $E_1.type = int$ and $E_2.type = int$ then int else if $E_1.type = int$ and $E_2.type = real$ then $real$ else if $E_1.type = real$ and $E_2.type = int$ then $real$ else if $E_1.type = real$ and $E_2.type = real$ then $real$ else $type\_error$