

# 语义分析

## (3. 符号表与类型检查)

魏恒峰

hfwei@nju.edu.cn

2021 年 12 月 14 日



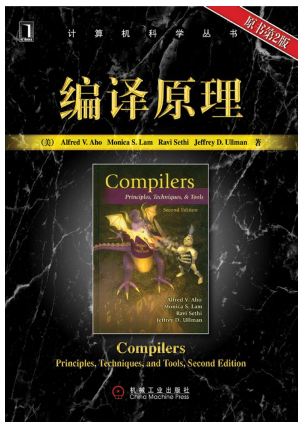
# 语义分析

语法制导的定义 (SDD)

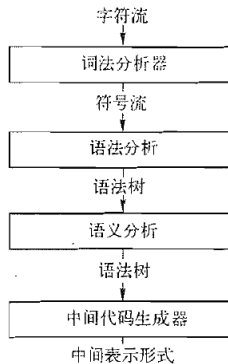
语法制导的翻译 (SDT)

符号表与类型检查

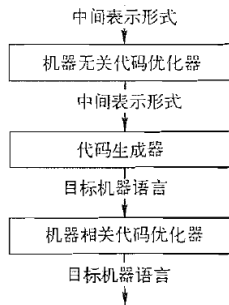
Another big difference is that we discourage the use of actions directly within the grammar because ANTLR 4 automatically generates [listeners and visitors](#) for you to use that trigger method calls when some phrases of interest are recognized during a tree walk after parsing. See also [Parse Tree Matching and XPath](#).



属性文法：语义动作嵌入到文法中      ANTLR4：将语义动作与文法分离



符号表



## Definition (符号表 (Symbol Table))

**符号表**是用于保存各种信息的**数据结构**。

**标识符**: 词素、类型、大小、存储位置等

```
public class ST<Key extends Comparable<Key>, Value>
```

ST()	<i>create an empty symbol table</i>
void put(Key key, Value val)	<i>associate val with key</i>
Value get(Key key)	<i>value associated with key</i>
void remove(Key key)	<i>remove key (and its associated value)</i>
boolean contains(Key key)	<i>is there a value associated with key?</i>
int size()	<i>number of key-value pairs</i>
Iterable<Key> keys()	<i>all keys in the symbol table</i>

可使用 HashTable 或 Red-Black Tree

“领域特定语言” (DSL) 通常只有**单作用域** (全局作用域)

```
host=antlr.org  
port=80  
webmaster=parrt@antlr.org
```

“通用程序设计语言” (GPL) 通常需要**嵌套作用域**

```
❶ //开始全局作用域
   int x;           //定义全局作用域中的变量x
❷ void f(){         //定义全局作用域中的函数f
    int y;          //定义f作用域中的变量y
❸   { int i; }      //在嵌套作用域中定出变量i
❹   { int j; }      //在嵌套作用域中定出变量j
    }
❺ void g() {        //定义全局作用域中的函数g
    int i;           //定义g作用域中的变量i
    }
```



①

```
int x;
```

②

```
void f(){
```

```
    int y;
```

③

```
    { int i; }
```

④

```
    { int j; }
```

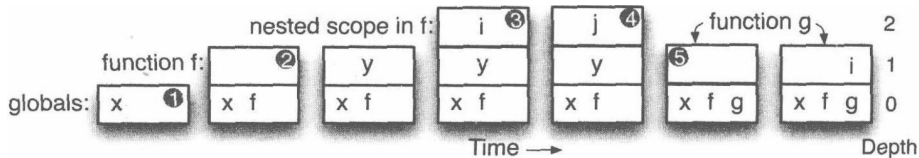
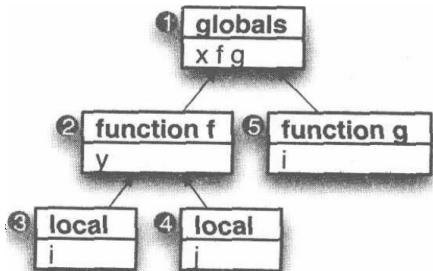
```
}
```

⑤

```
void g() {
```

```
    int i;
```

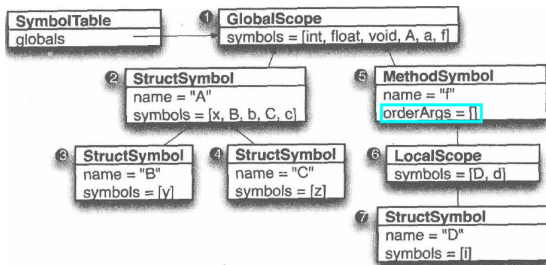
```
}
```



```
1) package symbols;                                // 文件 Env.java
2) import java.util.*;
3) public class Env {
4)     private Hashtable table;
5)     protected Env prev;
6)     public Env(Env p) { 创建新的, 旧的“压栈”
7)         table = new Hashtable(); prev = p;
8)     } 放入当前符号表
9)     public void put(String s, Symbol sym) {
10)         table.put(s, sym);
11)     } 倒序搜索符号表链
12)     public Symbol get(String s) {
13)         for( Env e = this; e != null; e = e.prev ) {
14)             Symbol found = (Symbol)(e.table.get(s));
15)             if( found != null ) return found;
16)         }
17)         return null;
18)     }
19) }
```

## struct: 类型作用域

```
1  
2 struct A {  
3     int x;  
4     struct B { int y; };  
5     B b;  
6     struct C {int z; };  
7     C c;  
8 };  
9 A a;  
10  
11 void f()  
12 {  
13     struct D {  
14         int i;  
15     };  
16     D d;  
17     d.i = a.b.y;  
18 }
```



*d.i*      *a.b.y*

声明: 添加符号与作用域 (“def”)

使用: 解析符号 (“ref”)

先声明后使用

```
{ int x; char y; { bool y; x; y; } x; y; }
```

**翻译任务:** 解析每个标识符 (引用处) 的**类型**

```
{ { x:int; y:bool; } x:int; y:char; }
```

## Definition (类型表达式 (Type Expressions))

- ▶ **基本类型**是类型表达式;
  - ▶ char, bool, int, float, double, void, ...
- ▶ **类名**是类型表达式;
- ▶ 如果  $t$  是类型表达式, 则 **array(num, t)** 是类型表达式;
- ▶ **record(<id : t, ...>)** 是类型表达式 (C 结构体);
- ▶ 如果  $s$  和  $t$  是类型表达式, 则  $s \times t$  是类型表达式;
- ▶ 如果  $s$  和  $t$  是类型表达式, 则  $s \rightarrow t$  是类型表达式;

<i>program</i>	$\rightarrow$	<span style="border: 1px solid red; padding: 2px;"><i>block</i></span>	$\{ \textit{top} = \textbf{null}; \}$
<i>block</i>	$\rightarrow$	'{'	$\{ \textit{saved} = \textit{top};$
			$\textit{top} = \textbf{new Env}(\textit{top});$
		<span style="border: 1px solid blue; padding: 2px;"><i>decls</i></span> <span style="border: 1px solid magenta; padding: 2px;"><i>stmts</i></span> '}'	$\{ \textit{top} = \textit{saved};$
			$\text{print}(\text{"{ "}); \}$
			$\text{print}(\text{"} \text{ "}); \}$
<i>decls</i>	$\rightarrow$	<i>decls decl</i>	
		$\epsilon$	
<i>decl</i>	$\rightarrow$	<b>type id ;</b>	$\{ \textit{s} = \textbf{new Symbol};$
			$\textit{s.type} = \textbf{type.lexeme};$
			$\textit{top.put}(\textbf{id.lexeme}, \textit{s}); \}$
<i>stmts</i>	$\rightarrow$	<i>stmts stmt</i>	
		$\epsilon$	
<i>stmt</i>	$\rightarrow$	<span style="border: 1px solid red; padding: 2px;"><i>block</i></span>	$\{ \text{print}(\text{"; "}); \}$
		<i>factor ;</i>	
<i>factor</i>	$\rightarrow$	<b>id</b>	$\{ \textit{s} = \textit{top.get}(\textbf{id.lexeme});$
			$\text{print}(\textbf{id.lexeme});$
			$\text{print}(\text{" : "});$
			$\text{print}(\textit{s.type}); \}$

```
{ char y; { bool y; y; } y; }
```

```
{ { y : bool; } y : char; }
```



<i>program</i> →	<i>block</i>	{ <i>top</i> = null; }
<i>block</i> →	{	局部变量 <i>saved</i> { <i>saved</i> = <i>top</i> ; <i>top</i> = new Env( <i>top</i> ); print("{ "); }
	<i>decls stmts</i> }	{ <i>top</i> = <i>saved</i> ; print("} "); }
<i>decls</i> →	<i>decls decl</i>   ε	
<i>decl</i> →	type id ;	{ <i>s</i> = new Symbol; <i>s.type</i> = type.lexeme; <i>top.put</i> (id.lexeme, <i>s</i> ); }
<i>stmts</i> →	<i>stmts stmt</i>   ε	
<i>stmt</i> →	<i>block</i>   <i>factor</i> ;	{ print("; "); }
<i>factor</i> →	id	{ <i>s</i> = <i>top.get</i> (id.lexeme); print(id.lexeme); print(": "); print( <i>s.type</i> ); }

## 类型声明

```
float x;  
record { float x; float y; } p;  
record { int tag; float x; float y; } q;
```

符号表中记录标识符的类型、宽度 (width)、偏移地址 (offset)

$$\begin{aligned}
 D &\rightarrow T \text{ id } ; D \mid \epsilon \\
 T &\rightarrow B C \mid \text{record } \{ D \} \\
 B &\rightarrow \text{int} \mid \text{float} \\
 C &\rightarrow \epsilon \mid [ \text{num} ] C
 \end{aligned}$$

需要为每个 **record** 生成单独的符号表

全局变量  $t$  与  $w$  将  $B$  的类型与宽度信息传递给产生式  $C \rightarrow \epsilon$   
(在语法制导定义中,  $t$  与  $w$  是  $C$  的继承属性)

$T \rightarrow B$	$\{ t = B.type; w = B.width; \}$
$C$	$\{ T.type = C.type; T.width = C.width \}$
$B \rightarrow \text{int}$	$\{ B.type = \text{integer}; B.width = 4; \}$
$B \rightarrow \text{float}$	$\{ B.type = \text{float}; B.width = 8; \}$
$C \rightarrow \epsilon$	$\{ C.type = t; C.width = w; \}$
$C \rightarrow [\text{num}] C_1$	$\{ C.type = \text{array}(\text{num.value}, C_1.type);$ $C.width = \text{num.value} \times C_1.width; \}$

图 6-15 计算类型及其宽度

`float x`: 目前仅关心类型声明 (`float`)

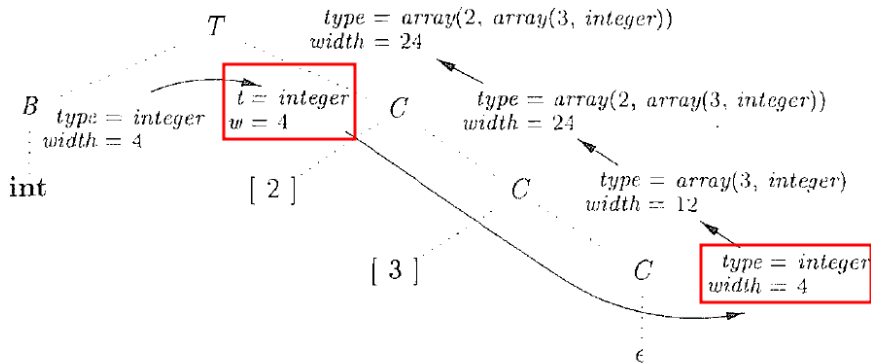


图 6-16 数组类型的语法制导翻译

`int [2] [3]`

全局变量 `offset` 表示变量的相对地址

全局变量 `top` 表示当前的符号表

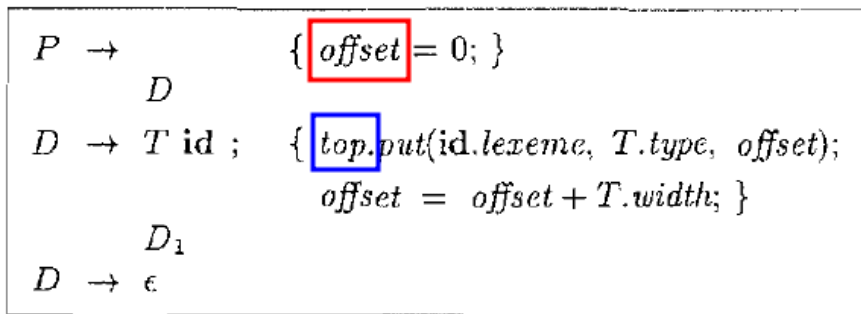


图 6-17 计算被声明变量的相对地址

```
float x; float y;
```

```
T → record '{' { Env.push(top); top = new Env();  
                  Stack.push(offset); offset = 0; }  
      D '}'      { T.type = record(top); T.width = offset;  
                  top = Env.pop(); offset = Stack.pop(); }
```

图 6-18 处理记录中的字段名

**record 类型表达式:** record(top)

字段名的偏移量是**相对**地址

**全局变量 top** 表示当前的符号表

**全局变量 Env** 表示符号表栈


```
record { float x; float y; } p;
```

```
float x;  
record { float x; float y; } p;  
record { int tag; float x; float y; } q;
```



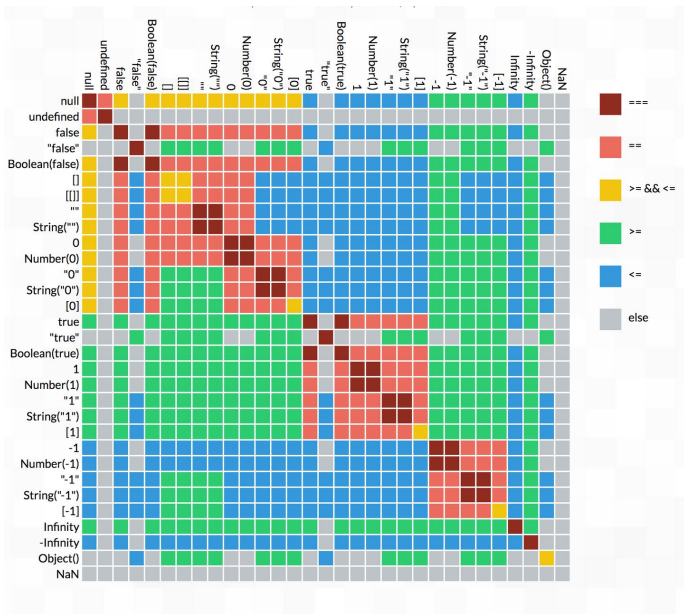
symtab @ antlr by parrt

symtab @ cs652 by parrt

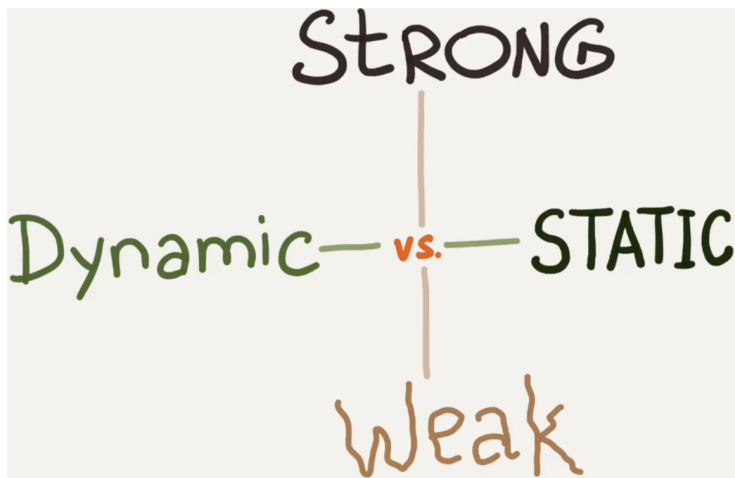


# TYPE SYSTEMS





## OH MY DEAR JAVASCRIPT



<https://youtu.be/C5fr0LZLMAs>

## 类型检查的常见形式

```
if two type expressions are equivalent  
then return a given type  
else return type_error
```

```
typedef struct {  
    int data[100];  
    int count;  
} Stack;
```

```
typedef struct {  
    int data[100];  
    int count;  
} Set;
```

```
Stack x, y;  
Set r, s;
```

## Definition (结构等价 (Structurally Equivalent))

此处不作严格定义

## Definition (名等价 (Name Equivalent))



结构等价中的“结构”又是什么意思？

$array(n, t)$        $array(m, t)$

```
type R3 = record
  a : integer;
  b : integer
end;
```

```
type R4 = record
  b : integer;
  a : integer
end;
```

不同的语言有不同的设计方案

**类型综合:** 根据子表达式的类型确定表达式的类型

**if**  $f$  的类型为  $s \rightarrow t$  且  $x$  的类型为  $s$   
**then** 表达式  $f(x)$  的类型为  $t$

$$E_1 + E_2$$

```
void err() { ... }  
void err(String s) { ... }
```

### 重载函数的类型综合规则

**if**  $f$  可能的类型为  $s_i \rightarrow t_i$  ( $1 \leq i \leq n$ ), 其中,  $s_i \neq s_j$  ( $i \neq j$ )  
**and**  $x$  的类型为  $s_k$  ( $1 \leq k \leq n$ )  
**then** 表达式  $f(x)$  的类型为  $t_k$

**类型推导:** 根据某语言结构的使用方式确定表达式的类型

**if**  $f(x)$  是一个表达式,  
**then** 对某些  $\alpha$  和  $\beta$ ,  $f$  的类型为  $\alpha \rightarrow \beta$  且  $x$  的类型为  $\alpha$

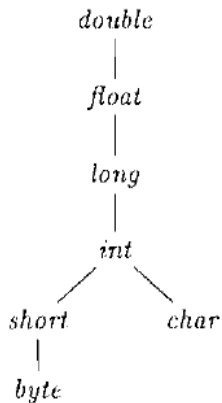
$null(x) : x$  是一个列表, 它的元素类型未知

## 类型转换

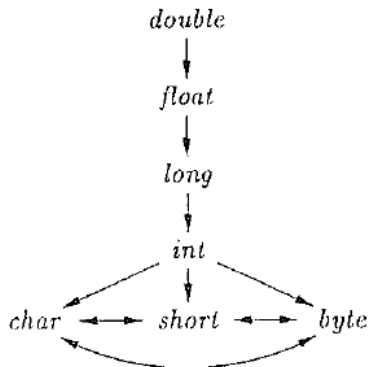
```
t1 = (float) 2  
t2 = t1 * 3.14
```

```
if ( E1.type = integer and E2.type = integer ) E.type = integer;  
else if ( E1.type = float and E2.type = integer ) ...  
...
```

不要写这样的代码!!!



a) 拓宽类型转换



b) 窄化类型转换

[Conversion@cppreference.com](http://Conversion.cppreference.com)

```

E → E1 + E2  { E.type = max(E1.type, E2.type);
                     a1 = widen(E1.addr, E1.type, E.type);
                     a2 = widen(E2.addr, E2.type, E.type);
                     E.addr = new Temp();
                     gen(E.addr '=' a1 '+' a2); }

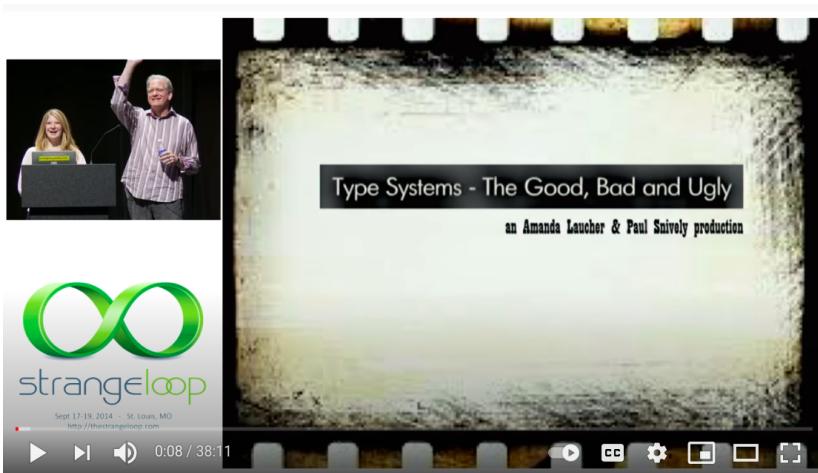
```

图 6-27 在表达式求值中引入类型转换

```

Addr widen(Addr a, Type t, Type w)
    if (t = w) return a;
    else if (t = integer and w = float) {
        temp = new Temp();
        gen(temp '=' (float) a);
        return temp;
    }
    else error;
}

```



<https://youtu.be/SWTWkYbcWU0>



Thank  
You!



Office 926

hfwei@nju.edu.cn