

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



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

Definition (符号表 (Symbol Table))

符号表是用于保存各种信息的**数据结构**。

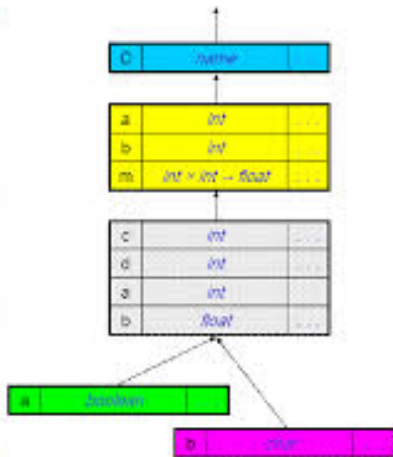
标识符: 词素、类型、大小、存储位置等

```
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>

通常使用 HashTable

为每个**作用域**建立单独的符号表



可以使用符号表**栈**实现**树形的**嵌套关系

```

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) }

```

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

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

翻译任务：忽略**声明**部分，为每个标识符标明**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;

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;

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;
- ▶ 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;

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;
- ▶ 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;
- ▶ 类型表达式可以包含取值为类型表达式的变量。

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

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

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

<i>program</i> →	block	{ top = null; }
<i>block</i> →	'{'	{ saved = <i>top</i> ;
		<i>top</i> = new Env(<i>top</i>);
		print("{ "); }
	decls stmts '}'	{ <i>top</i> = saved
		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 ;
		top.put (<i>id.lexeme</i> , <i>s</i>); }
<i>stmts</i> →	<i>stmts stmt</i>	
		ε
<i>stmt</i> →	block	
		<i>factor</i> ;
<i>factor</i> →	id	{ <i>s</i> = top.get (<i>id.lexeme</i>);
		print(<i>id.lexeme</i>);
		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 的类型与宽度信息递给产生式 $C \rightarrow \epsilon$

$T \rightarrow B$	$\{ t = B.type; w = B.width; \}$
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

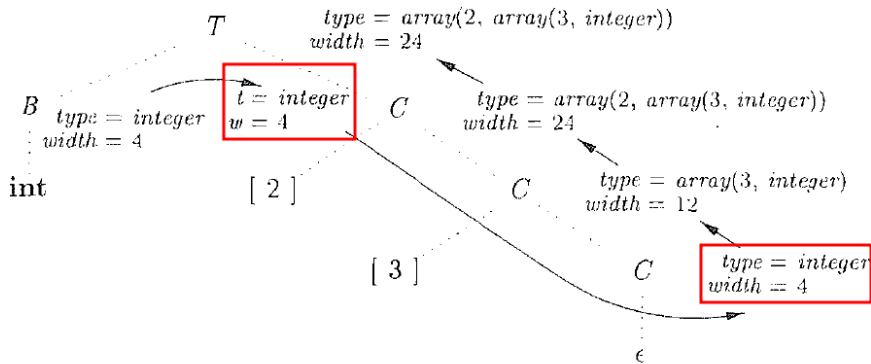


图 6-16 数组类型的语法制导翻译

`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

$$\begin{aligned} P &\rightarrow D \quad \{ \text{offset} = 0; \} \\ D &\rightarrow T \text{ id } ; \quad \{ \text{top.put}(\text{id.lexeme}, T.\text{type}, \text{offset}); \\ &\quad \text{offset} = \text{offset} + T.\text{width}; \} \\ D &\rightarrow \epsilon \end{aligned}$$

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

```

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)

```

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)

全局变量 top 表示当前的符号表

```

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)

全局变量 top 表示当前的符号表

全局变量 Env 表示符号表栈

```

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

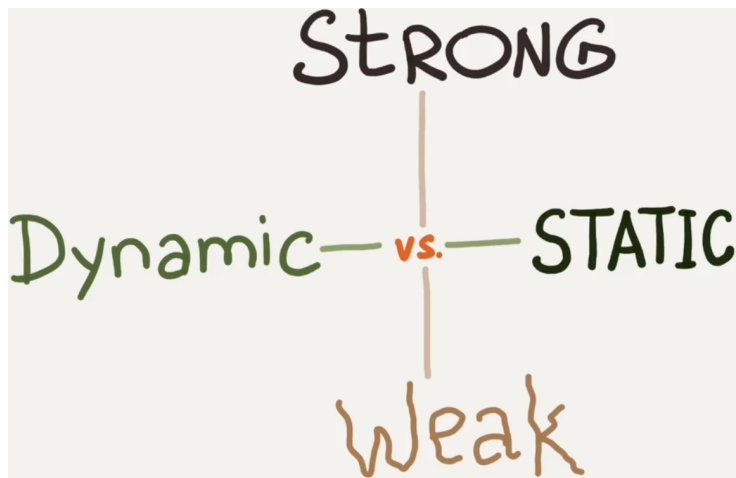
record 类型表达式: record(top)

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;
```



<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))

两种类型**结构等价**当且仅当以下任一条件为真:

- ▶ 它们是相同的基本类型;
- ▶ 它们是将相同的类型构造算子应用于结构等价类型而构造得到;
- ▶ 一个类型是另一个类型表达式的名字。

Definition (结构等价 (Structurally Equivalent))

两种类型**结构等价**当且仅当以下任一条件为真:

- ▶ 它们是相同的基本类型;
- ▶ 它们是将相同的类型构造算子应用于结构等价的类型而构造得到;
- ▶ **一个类型是另一个类型表达式的名字。**

Definition (名等价 (Name Equivalent))

两种类型**名等价**当且仅当以下任一条件为真:

- ▶ 它们是相同的基本类型;
- ▶ 它们是将相同的类型构造算子应用于结构等价的类型而构造得到。

结构等价中的“结构”又是什么意思？

$array(n, t)$ $array(m, t)$

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;
```


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 α 和 β , f 的类型为 $\alpha \rightarrow \beta$ 且 x 的类型为 α

$null(x) : x$ 是一个列表, 它的元素类型未知

类型转换

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

类型转换

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

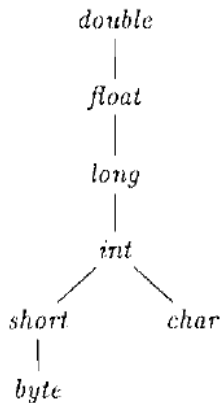
```
if ( E1.type = integer and E2.type = integer ) E.type = integer;  
else if ( E1.type = float and E2.type = integer ) ...  
...
```

类型转换

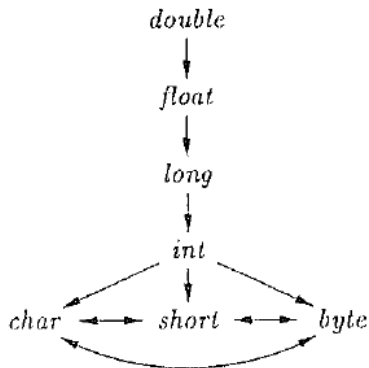
```
t1 = (float) 2  
t2 = t1 * 3.14
```

```
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


```

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;
}

```



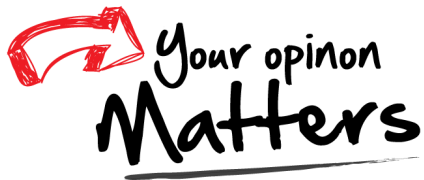
Sept 17-19, 2014 - St. Louis, MO
<http://thestrangeloop.com>

0:08 / 38:11



<https://youtu.be/SWTWkYbcWU0>

Thank
You!



Office 926

hfwei@nju.edu.cn