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

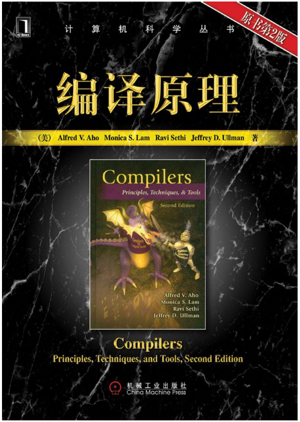
语义分析

语法制导的定义 (SDD)

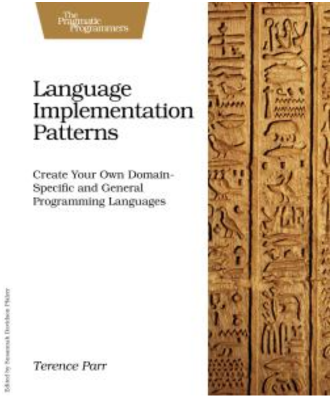
语法制导的翻译 (SDT)

符号表与类型检查

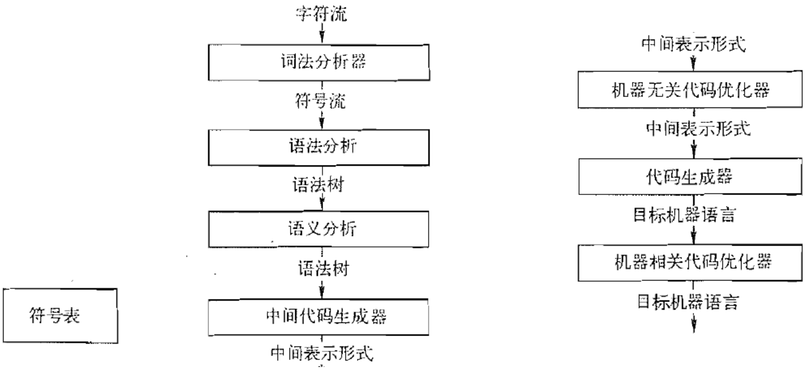
Another big difference is that we $\boxed{\text{discourage the use of actions directly within the grammar}}$ because ANTLR 4 automatically generates $\boxed{\text{listeners and visitors}}$ for you to use that trigger method calls when some phrases of interest are recognized during a tree walk after parsing. 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


字符流
词法分析器
符号流
语法分析
语法树
语义分析
语法树
中间代码生成器
中间表示形式
符号表
中间表示形式
机器无关代码优化器
中间表示形式
代码生成器
目标机器语言
机器相关代码优化器
目标机器语言

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

可使用 HashTable 或 Red-Black Tree

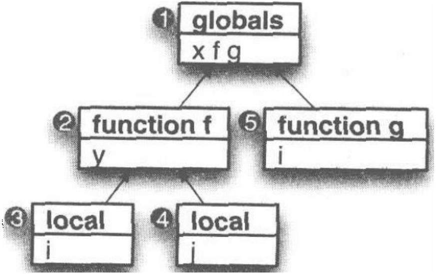
“领域特定语言” (DSL) 通常只有**单作用域** (全局作用域)

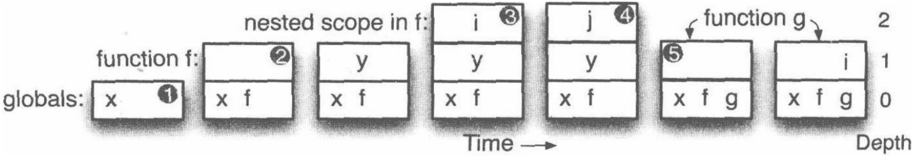
```
host=antlr.org
port=80
webmaster=parrt@antlr.org
```

“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
    }
```

```
❶
      int x;
❷     void f(){
        int y;
❸       { int i; }
❹       { int j; }
      }
❺     void g() {
        int i;
      }
```





```java
package symbols;                                    // 文件 Env.java
import java.util.*;
public class Env {
    private Hashtable table;
    protected Env prev;

    public Env(Env p) {
```

创建新的，旧的"压栈"

```java
        table = new Hashtable(); prev = p;
    }
```

放入当前符号表

```java
    public void put(String s, Symbol sym) {
        table.put(s, sym);
    }
```

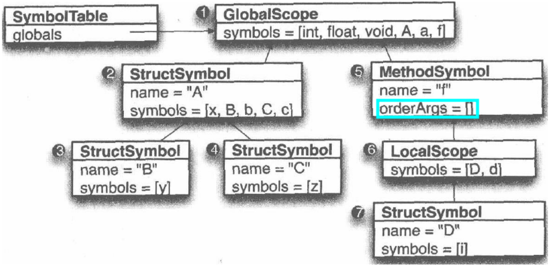
倒序搜索符号表链

```java
    public Symbol get(String s) {
        for( Env e = this; e != null; e = e.prev ) {
            Symbol found = (Symbol)(e.table.get(s));
            if( found != null ) return found;
        }
        return null;
    }
}
```

struct: 类型作用域

```
❶
❷ struct A {
     int x;
❸    struct B { int y; };
     B b;
❹    struct C {int z; };
     C c;
  };
  A a;

❺ void f()
❻ {
❼    struct D {
        int i;
     };
     D d;
     d.i = a.b.y;
  }
```



$d.i$ $a.b.y$

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; }
```

```
{ 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;

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);

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 $s \to t$ 是类型表达式;

program →	*block*	{ *top* = **null**; }
block →	'{'	{ *saved* = *top*; *top* = **new** *Env*(*top*); print("{ "); }
	decls *stmts* '}'	{ *top* = *saved*; print("} "); }
decls →	*decls* *decl*	
|	ϵ	
decl →	**type** **id** ;	{ *s* = **new** *Symbol*; *s.type* = **type**.*lexeme*; *top.put*(**id**.*lexeme*, *s*); }
stmts →	*stmts* *stmt*	
|	ϵ	
stmt →	*block*	
|	*factor* ;	{ print("; "); }
factor →	**id**	{ *s* = *top.get*(**id**.*lexeme*); print(**id**.*lexeme*); print(":"); print(*s.type*);}

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

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

program	$\rightarrow$		{ *top* = **null**; }
		block	
			局部变量 saved
block	$\rightarrow$	'{'	{ *saved* = *top*; *top* = **new** *Env*(*top*); print("{ "); }
		decls *stmts* '}'	{ *top* = *saved*; print("} "); }
decls	$\rightarrow$	*decls* *decl*	
	|	ϵ	
decl	$\rightarrow$	**type id** ;	{ *s* = **new** *Symbol*; *s.type* = **type**.*lexeme*; *top*.*put*(**id**.*lexeme*, *s*); }
stmts	$\rightarrow$	*stmts* *stmt*	
	|	ϵ	
stmt	$\rightarrow$	*block*	
	|	*factor* ;	{ print("; "); }

factor	$\rightarrow$	**id**	{ *s* = *top*.*get*(**id**.*lexeme*);
			print(**id**.*lexeme*);
			print(":");
			print(*s.type*); }

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)

$$
\begin{array}{lcl}
D & \rightarrow & T\ \mathbf{id}\ ;\ D \mid \epsilon \\
T & \rightarrow & B\ C \mid \mathbf{record}\ `\{'\ D\ `\}' \\
B & \rightarrow & \mathbf{int} \mid \mathbf{float} \\
C & \rightarrow & \epsilon \mid [\ \mathbf{num}\ ]\ C
\end{array}
$$

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 $C \to \epsilon$

(在语法制导定义中, t 与 w 是 C 的**继承属性**)

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

图 6-15 计算类型及其宽度

`float x` : 目前仅关心类型声明 (`float`)

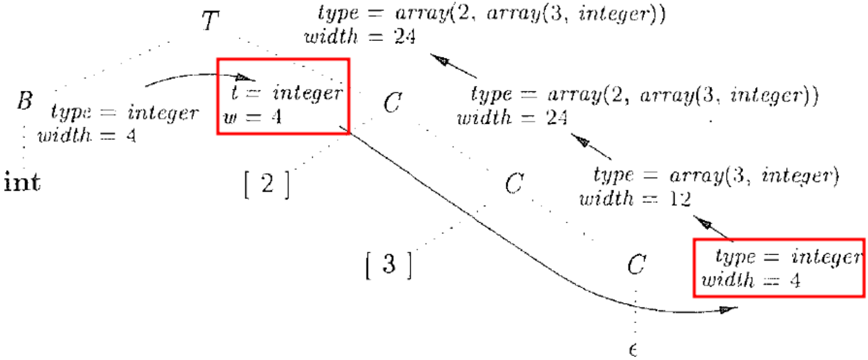


图 6-16　数组类型的语法制导翻译

`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

$$
\begin{array}{llll}
P & \rightarrow & & \{\ \boxed{offset} = 0;\ \} \\
 & & D & \\
D & \rightarrow & T\ \mathbf{id}\ ; & \{\ \boxed{top.}put(\mathbf{id}.lexeme,\ T.type,\ offset); \\
 & & & \quad offset = offset + T.width;\ \} \\
 & & D_1 & \\
D & \rightarrow & \epsilon &
\end{array}
$$

图 6-17　计算被声明变量的相对地址

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

$$
\begin{array}{llll}
T & \rightarrow & \textbf{record}\ '\{' & \{\ \boxed{Env.push}(top);\ \boxed{top} = \textbf{new}\ Env(); \\
 & & & \quad Stack.push(offset);\ offset = 0;\ \} \\
 & & D\ '\}' & \{\ \boxed{T.type = record(top);}\ \boxed{T.width = offset;} \\
 & & & \quad top = \boxed{Env.pop()};\ offset = Stack.pop();\ \}
\end{array}
$$

图 6-18　处理记录中的字段名

$$
\begin{array}{lll}
T \rightarrow \textbf{record} \text{ '\{'} & \{ \boxed{Env.push}(top); \boxed{top} = \textbf{new } Env(); \\
& \quad Stack.push(offset);\ offset = 0; \} \\
\quad D \text{ '\}'} & \{ \boxed{T.type = record(top);} \boxed{T.width = offset;} \\
& \quad top = \boxed{Env.pop()};\ offset = Stack.pop(); \}
\end{array}
$$

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

$$
\begin{array}{lll}
T \rightarrow \textbf{record} \; '\{' & \{ & \boxed{Env.push}(top);\ \boxed{top} = \textbf{new}\ Env(); \\
 & & Stack.push(offset);\ offset = 0;\ \} \\
\quad D \; '\}' & \{ & \boxed{T.type = record(top);}\ \boxed{T.width = offset;} \\
 & & top = \boxed{Env.pop()};\ offset = Stack.pop();\ \}
\end{array}
$$

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

$$
\begin{array}{llll}
T & \rightarrow & \textbf{record}\ '\{' & \{\ \boxed{Env.push}(top);\ \boxed{top} = \textbf{new}\ Env();\\
 & & & \quad Stack.push(offset);\ offset = 0;\ \}\\
 & & D\ '\}' & \{\ \boxed{T.type = record(top);}\ \boxed{T.width = offset;}\\
 & & & \quad top = \boxed{Env.pop()};\ offset = Stack.pop();\ \}
\end{array}
$$

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

$$
\begin{array}{lll}
T & \rightarrow \ \textbf{record}\ '\{' & \{\ \boxed{Env.push}(top);\ \boxed{top} = \textbf{new}\ Env(); \\
 & & \quad Stack.push(offset);\ offset = 0;\ \} \\
 & \quad D\ '\}' & \{\ \boxed{T.type = record(top);}\ \boxed{T.width = offset;} \\
 & & \quad top = \boxed{Env.pop()};\ offset = Stack.pop();\ \}
\end{array}
$$

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

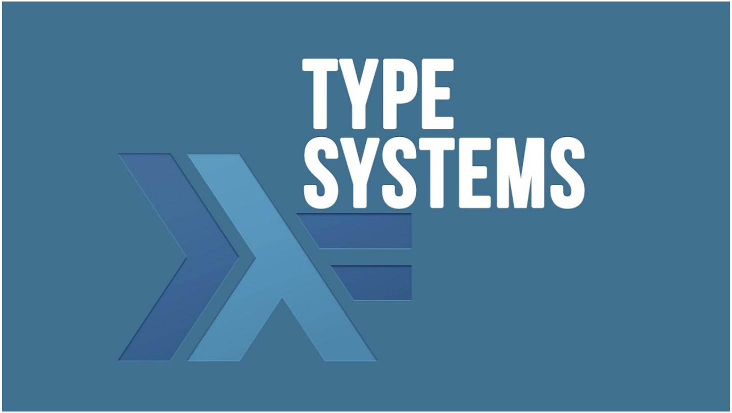
全局变量 Env 表示符号表栈

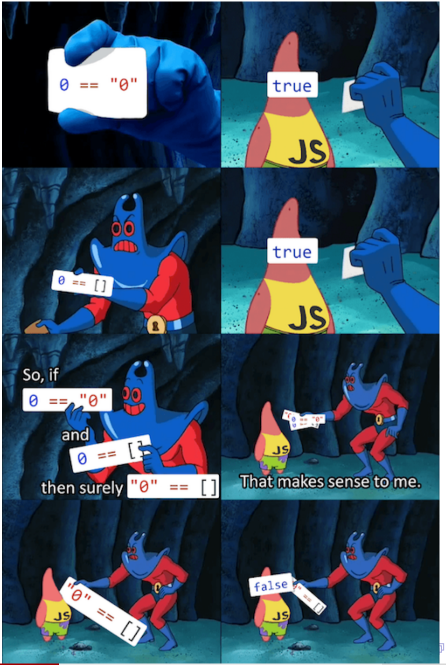
```
record { float x; float y; } p;
```

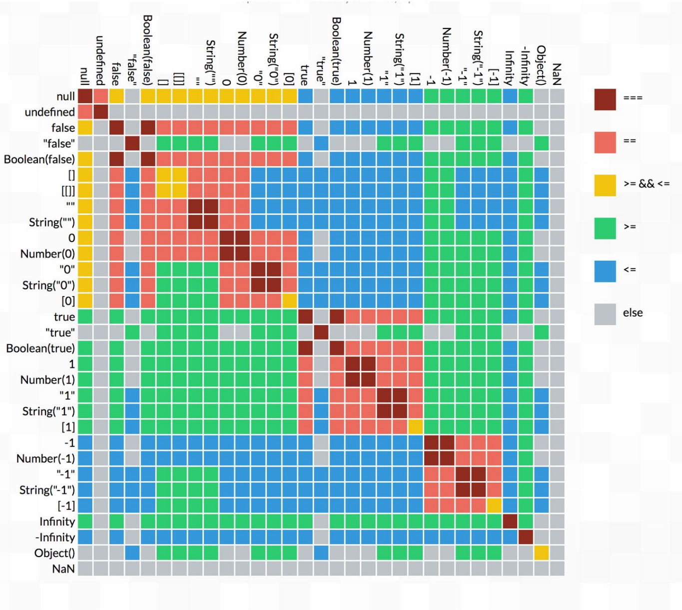
```
float x;
record { float x; float y; } p;
record { int tag; float x; float y; } q;
```

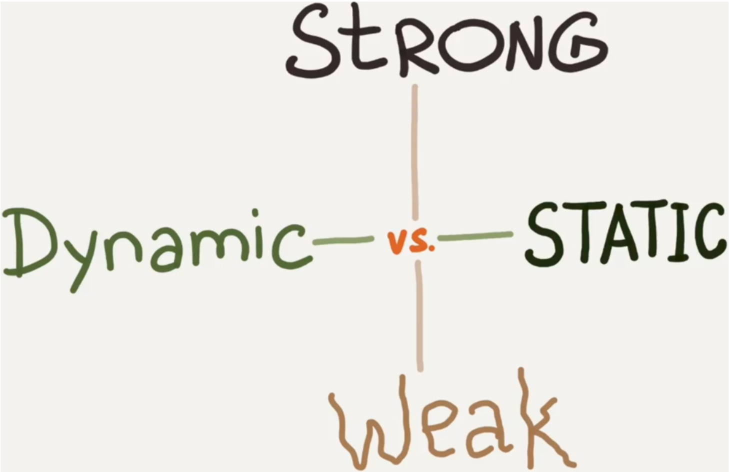
symtab @ antlr by parrt

symtab @ cs652 by parrt


TYPE
SYSTEMS


0 == "0"
true
0 == []
true
So, if
0 == "0"
and
0 == []
then surely "0" == []
That makes sense to me.
"0" == []
false
JS


null
undefined
false
"false"
Boolean(false)
[]
[[]]
""
String("")
0
Number(0)
"0"
String("0")
[0]
true
"true"
Boolean(true)
1
Number(1)
"1"
String("1")
[1]
-1
Number(-1)
"-1"
String("-1")
[-1]
Infinity
-Infinity
Object()
NaN
===
==
>= && <=
>=
<=
else



https://youtu.be/C5fr0LZLMAs

类型检查的常见形式

if two type expressions are $\boxed{\text{equivalent}}$
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?

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

结构等价中的“结构”又是什么意思？

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

```
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？

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

```
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 $\boxed{f(x)}$ 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 $\boxed{s_i \to t_i}$ $(1 \leqslant i \leqslant n)$，其中，$s_i \neq s_j (i \neq j)$

and x 的类型为 $\boxed{s_k}$ $(1 \leqslant k \leqslant n)$

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 $\boxed{f(x)}$ 是一个表达式,

then 对某些 α 和 β, f 的类型为 $\alpha \to \beta$ 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

$$t_1 = \boxed{(\texttt{float})}\ 2$$

$$t_2 = t_1 * 3.14$$

类型转换

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

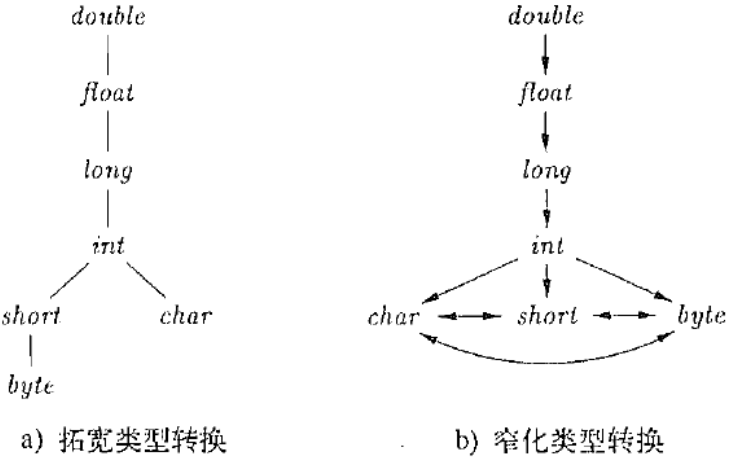
if ($E_1.type = integer$ **and** $E_2.type = integer$) $E.type = integer$;
else if ($E_1.type = float$ **and** $E_2.type = integer$) $\cdots$
$\cdots$

类型转换

$$t_1 = \boxed{\texttt{(float)}}\ 2$$
$$t_2 = t_1 * 3.14$$

if ($E_1.type = integer$ **and** $E_2.type = integer$) $E.type = integer$;
else if ($E_1.type = float$ **and** $E_2.type = integer$) $\cdots$
$\cdots$

不要写这样的代码!!!



Conversion@cppreference.com

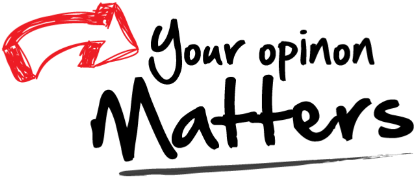
$$
\begin{array}{lll}
E \rightarrow E_1 + E_2 & \{ & E.type = \boxed{max}(E_1.type, E_2.type); \\
& & a_1 = \boxed{widen}(E_1.addr, E_1.type, E.type); \\
& & a_2 = \boxed{widen}(E_2.addr, E_2.type, E.type); \\
& & E.addr = \mathbf{new}\ Temp(); \\
& & gen(E.addr\ '='\ a_1\ '+'\ a_2); \}
\end{array}
$$

图 6-27 在表达式求值中引入类型转换

$$
\begin{array}{l}
Addr\ widen(Addr\ a,\ Type\ t,\ Type\ w) \\
\quad \textbf{if}\ (\boxed{t = w})\ \ \mathbf{return}\ a; \\
\quad \textbf{else if}\ (\ t = integer\ \textbf{and}\ w = float\ )\ \{ \\
\quad\quad temp = \mathbf{new}\ Temp(); \\
\quad\quad gen(temp\ '='\ '\boxed{\textbf{(float)}}'\ a); \\
\quad\quad \mathbf{return}\ temp; \\
\quad \} \\
\quad \textbf{else}\ \ \mathbf{error}; \\
\}
\end{array}
$$

https://youtu.be/SWTWkYbcWU0


Thank
You!



Office 926

hfwei@nju.edu.cn