中国科学技术大学
University of Science and Technology of China

USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》
类型及其应用
(静态类型检查)

计算机科学与技术学院
李 诚
6/11/2019

通知

- 重申一下：抄袭零容忍，包括copy的实施者和material的提供者
- 今天课后有习题课和Lab3-1发布
- 复习词法分析和语法分析的时候，欢迎大家git上发issue提问。除非带有隐私的信息，请大家尽量不要发邮件，issue辐射面更广泛，更能提高效率。
- 周日上午8:30-11:30，我本人在东校区高性能计算中心503答疑，如有需要，欢迎！

通知

- 期中考试卷子已经批改完毕，本周内反馈个人成绩和统计结果。
- Lab3还未组队的同学一定要尽快按照助教的要求组队。

类型化的语言

- 变量的类型
  - 限定了变量在程序执行期间的取值范围和存储空间消耗
- 类型化的语言(typed language)
  - 变量都被给定类型的语言
  - 表达式、语句等程序构造的类型都可以静态确定，运行时不需要额外的操作
- 未类型化的语言(untyped language)
  - 不限制变量值范围的语言,如JavaScript、Perl

类型的两个典型应用

- 静态的语义分析
  - 类型检查

利用**逻辑规则**分析运算分量的类型与运算符预期是否匹配?

- 中间代码生成
  - 声明语句的翻译
  - 数组寻址的翻译
  - 类型转换

- 通过声明语句**收集**变量或函数的类型
- **计算**所占存储空间
- **分配**相对地址
- 类型**转换**适配指令选择

类型的应用

- 静态的语义分析
  - 类型检查

本节课重点关注这一部分

- 中间代码生成
  - 声明语句的翻译
  - 数组寻址的翻译
  - 类型转换

下一节课重点关注这一部分

主要内容

- 类型表达式
  - 类型的结构
- 类型等价
  - 结构等价和名字等价
- 类型检查
  - 语法制导翻译方案实现
  - 函数和算符的重载
- 其他知识点

层次一：形式化描述类型结构

层次二：判定两个类型相同的依据

层次三：定义一组逻辑规则检查语句或者表达式中是否存在类型错误

类型表达式 (Type expression)

- 类型可以是语法的一部分，因此也是结构的

考虑以下文法，D代表声明语句，S代表一般语句

$P \rightarrow D\ ;\ S$

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

$T \rightarrow \mathrm{boolean} \mid \mathrm{integer} \mid \mathrm{array}\ [\mathrm{num}\ ]\ \mathrm{of}\ T \mid \uparrow T \mid T\ \text{‘}\rightarrow\text{’}\ T$

类型表达式 (Type expression)

❑类型可以是语法的一部分，因此也是结构的

考虑以下文法，D代表声明语句，S代表一般语句

$P \rightarrow D\ ;\ S$

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

$T \rightarrow \mathrm{boolean} \mid \mathrm{integer} \mid \mathrm{array}\ [\mathrm{num}\ ]\ \mathrm{of}\ T \mid \uparrow T \mid T\ \text{‘}\rightarrow\text{’}\ T$

数组　指针　函数

基本类型　　复杂且可组合的类型

类型表达式 (Type expression)

- 基本类型是类型表达式
  - *integer*
  - *real*
  - *char*
  - *boolean*
  - *type_error //出错类型*
  - *void //无类型*

在类型检查中传递错误

语句的类型

类型表达式 (Type expression)

中国科学技术大学
University of Science and Technology of China

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
    - 如果T是类型表达式，N是一个整数，则*array(N, T)*是类型表达式

类型表达式 (Type expression)

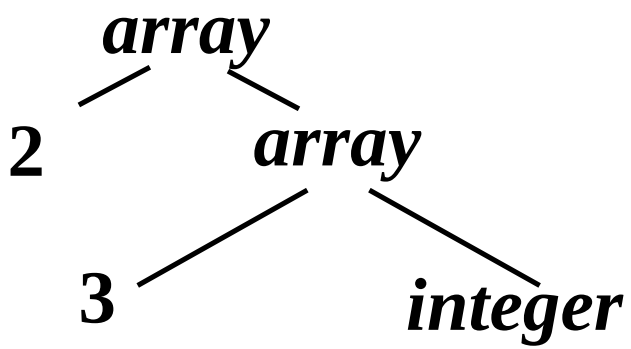
- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
    - 如果T是类型表达式，N是一个整数，则*array(N, T)*是类型表达式

类型	类型表达式
int[3]	*array* (3, *integer*)
int[2][3]	*array* (2, *array* (3, *integer*))

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
    - 如果T是类型表达式，N是一个整数，则*array(N, T)*是类型表达式

类型	类型表达式
int[3]	*array* (3, *integer*)
int[2][3]	*array* (2, *array* (3, *integer*))

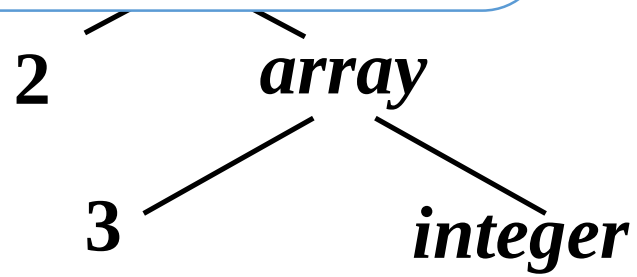


类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
    - 如果T是类型表达式，N是一个

类型	类型表达式
int[3]	*array* (3, *integer*)
int[2][3]	*array* (2, *array* (3, *integer*))

也可以写为
array ({0,…,2}, integer)
其中{0,…,2}代表索引集合
如首元素索引从1开始，则为*array ({1,…,3}, integer)*



类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
  - 指针类型构造算子*pointer*
    - 如果T是类型表达式，则*pointer(T)*是类型表达式

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
  - 指针类型构造算子*pointer*
  - 笛卡尔乘积类型构造算子×
    - 如果T_1和T_2是类型表达式，则$T_1 \times T_2$也是类型表达式
    - 主要用于描述列表和元组，如：表示函数的参数

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
  - 指针类型构造算子*pointer*
  - 笛卡尔乘积类型构造算子×
  - 函数类型构造算子→
    - 若$T_1, T_2, \ldots, T_n$和R是类型表达式，则$T_1 \times T_2 \times \ldots \times T_n \rightarrow R$也是

函数参数　函数返回值

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
  - 指针类型构造算子*pointer*
  - 笛卡尔乘积类型构造算子×
  - 函数类型构造算子→
  - 记录类型构造算子*record*
    - 若有标识符$N_1, N_2, \ldots, N_n$以及对应的类型表达式$T_1, T_2, \ldots, T_n$，则 *record*$((N_1 \times T_1) \times (N_2 \times T_2) \times \ldots \times (N_n \times T_n))$也是类型表达式

记录中的字段

字段对应的类型表达式

类型表达式-例1

- 考虑C语言中数组double a[10][20]，写出a、a[0]、a[0][0]的类型表达式

类型表达式-例1

- 考虑C语言中数组double a[10][20]，写出a、a[0]、a[0][0]的类型表达式

a[0][0]: double

类型表达式-例1

- 考虑C语言中数组double a[10][20]，写出a、a[0]、a[0][0]的类型表达式

a[0][0]: double

a[0]: array(20,double);
pointer(double)

类型表达式-例1

- 考虑C语言中数组double a[10][20]，写出a、a[0]、a[0][0]的类型表达式

a[0][0]: double

a[0]: array(20,double);
pointer(double)

a: array(10,array(20,double));
pointer(array(20,double))

类型表达式-例2

- 为row、table和p分别写出类型表达式：

```
typedef struct{
    int address;
    char lexeme[15];
} row;
row table[101];
row *p;
```

类型表达式-例2

❑为row、table和p分别写出类型表达式：

```
typedef struct{
    int address;
    char lexeme[15];
} row;
row table[101];
row *p;
```

row的类型表达式：

record((address×integer) × (lexeme × (array(15, char))))

table的类型表达式：

array(101, row) //此处row是类型名，因此也是类型表达式

p的类型表达式：

pointer(row)

类型表达式-例3

- 考虑下面的函数f，写出其类型表达式。

```
int *f(char a, char b);
```

f的类型表达式：

(char×char) → pointer(integer)

类型表达式-例3

- 考虑下面的函数f，写出其类型表达式。

```
int *f(char a, char b);
```

f的类型表达式：

(char×char) → pointer(integer)

问题：可否自动化地实现类型表达式的生成?

构造类型表达式的SDD

- 为以下文法制定构造类型表达式的语义规则

产 生 式	语 义 规 则
$T \rightarrow B\ C$	
$B \rightarrow \mathbf{int}$	
$B \rightarrow \mathbf{float}$	
$C \rightarrow \mathbf{[num]}\ C_1$	
$C \rightarrow \varepsilon$	

构造类型表达式的SDD

- 为以下文法制定构造类型表达式的语义规则

产 生 式	语 义 规 则
$T \rightarrow B\ C$	
$B \rightarrow$ int	
$B \rightarrow$ float	
$C \rightarrow$ [num] C_1	
$C \rightarrow \varepsilon$	

- 为每个文法符号设置综合属性t和继承属性b
  - t：该符号对应的类型表达式
  - b：将类型信息从左到右传递

构造类型表达式的SDD

❑为以下文法制定构造类型表达式的语义规则

产 生 式	语 义 规 则
$T \rightarrow B\ C$	$T.t = C.t;\ C.b = B.t$
$B \rightarrow$ int	$B.t =$ *integer*
$B \rightarrow$ float	$B.t =$ *float*
$C \rightarrow$ [num] C_1	$C.t =$ *array*$(num.val, C_1.t);\ C_1.b = C.b$
$C \rightarrow \varepsilon$	$C.t = C.b$

❑为每个文法符号设置**综合属性**t和**继承属性**b

- ❖t：该符号对应的类型表达式
- ❖b：将类型信息从左到右传递

构造类型表达式的SDT

- 将SDD改造为SDT

$T \rightarrow B$ {$C.b = B.t$} C {$T.t = C.t$; }

$B \rightarrow$ int {$B.t = integer$}

$B \rightarrow$ float {$B.t = float$}

$C \rightarrow$ [num] {$C_1.b = C.b$} C_1 {$C.t = array(num.val, C_1.t)$; }

$C \rightarrow \varepsilon$ {$C.t = C.b$}

- 但是继承属性的计算与LR分析方法不适配
- 因此，如果要使用LR，就需要改造文法

构造类型表达式的SDT

- 通过改造文法，与LR适配
  - 引入标记M，C归约时可在栈顶以下位置找到B.t
  - 引入标记N，把继承属性C.b当做综合属性记录

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1\ \{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

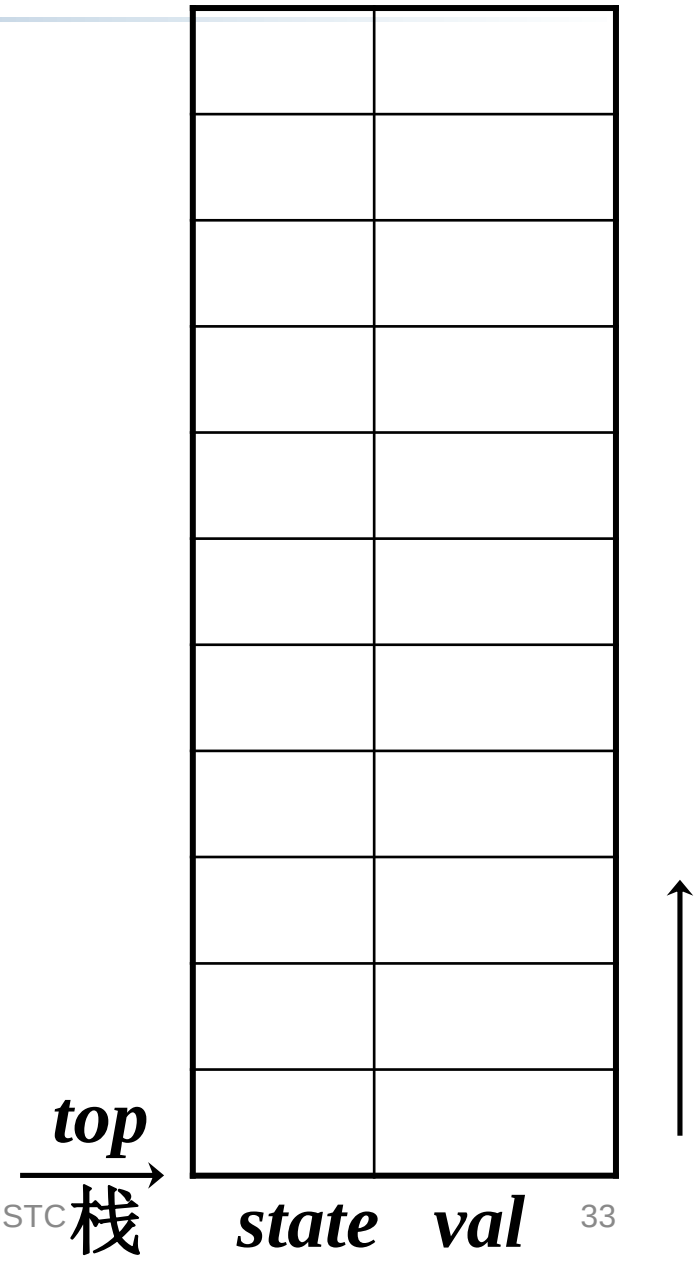
$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

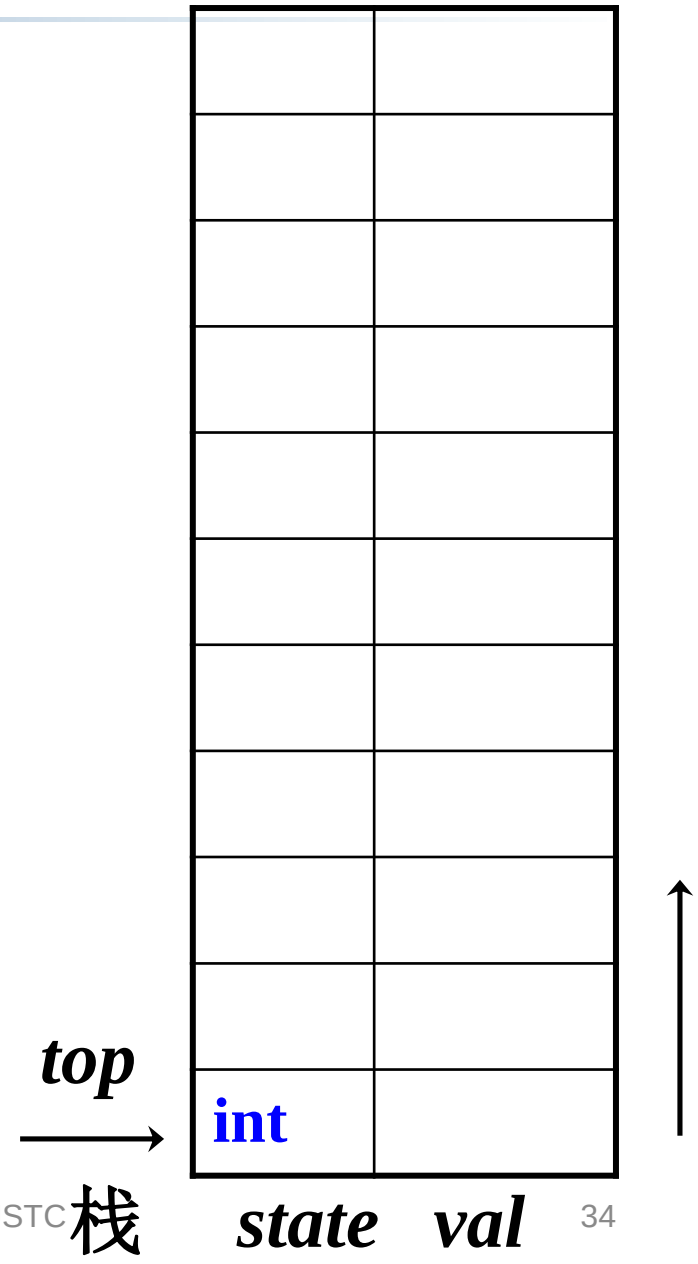
$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

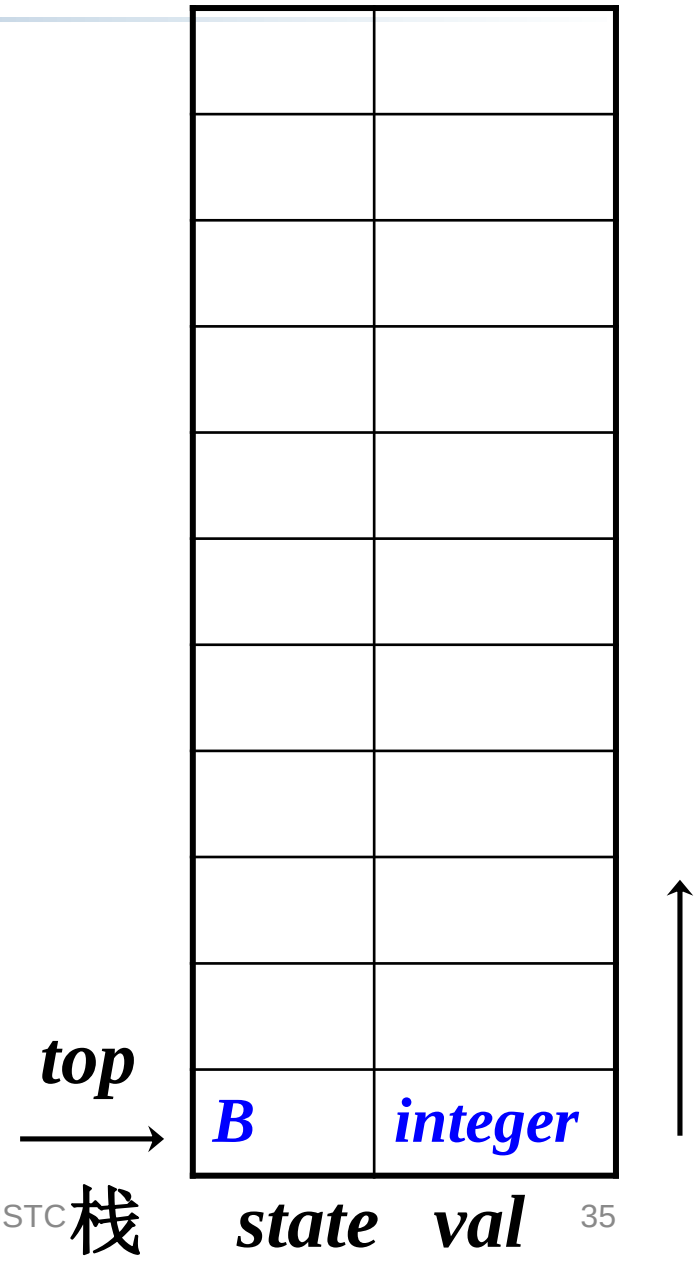
$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

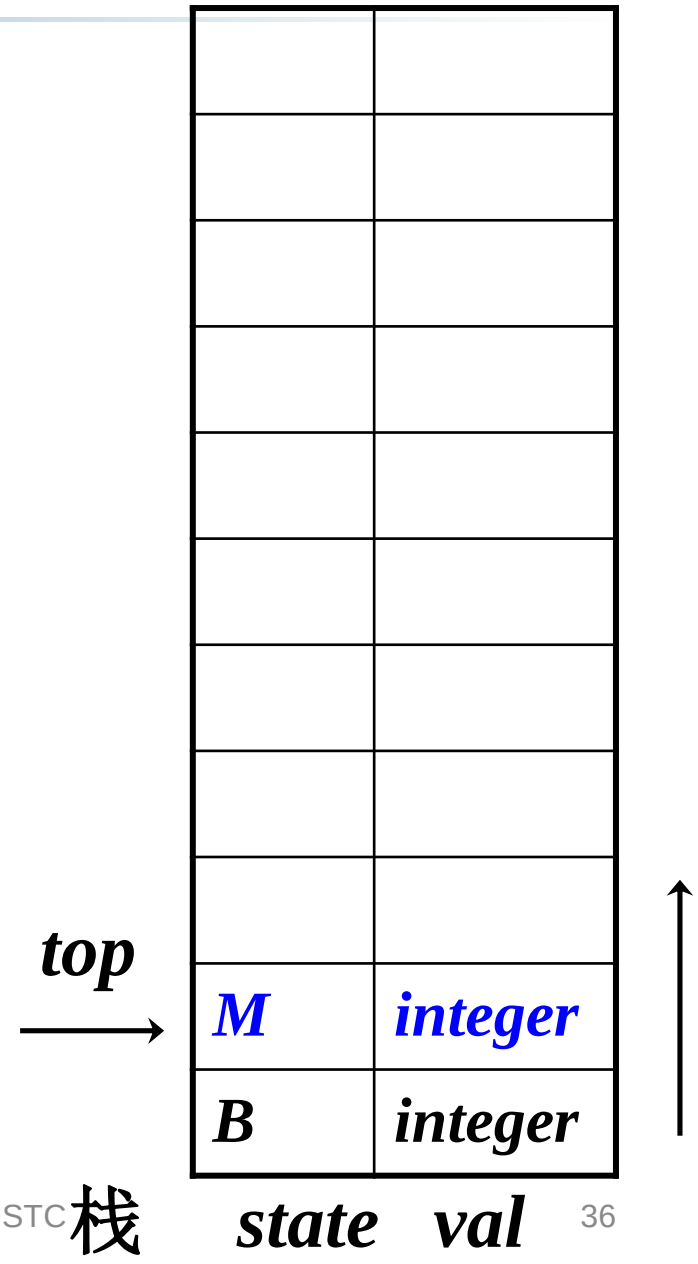
$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

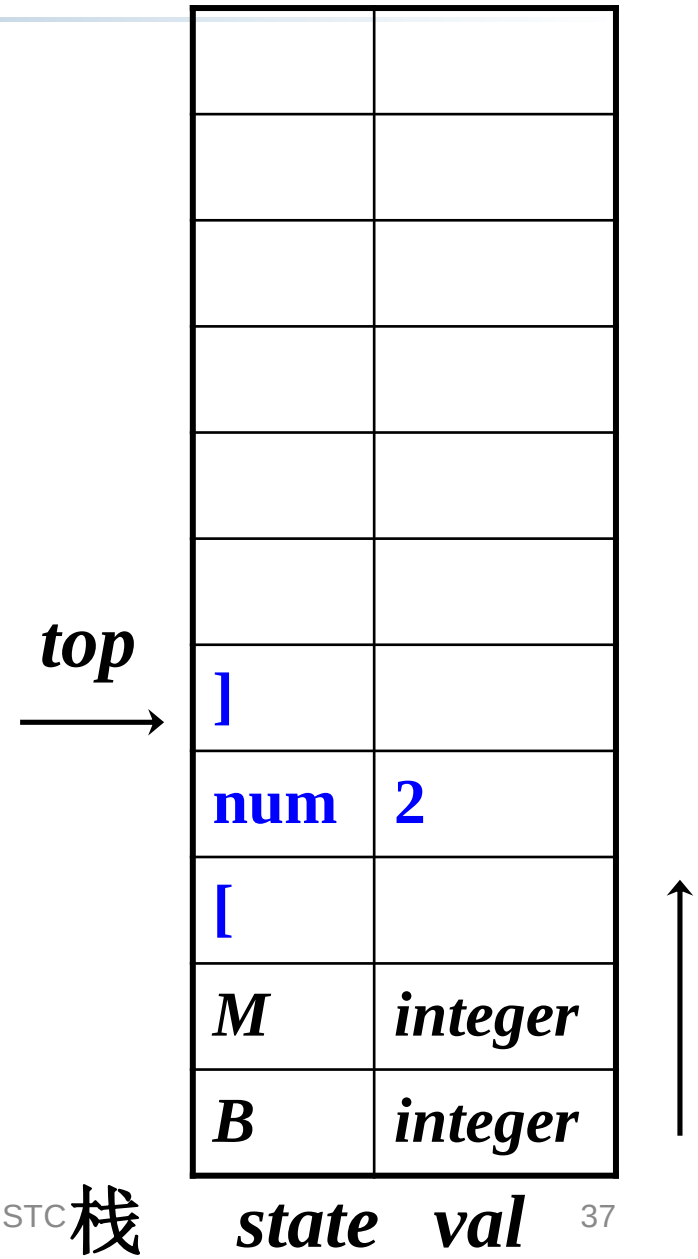
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

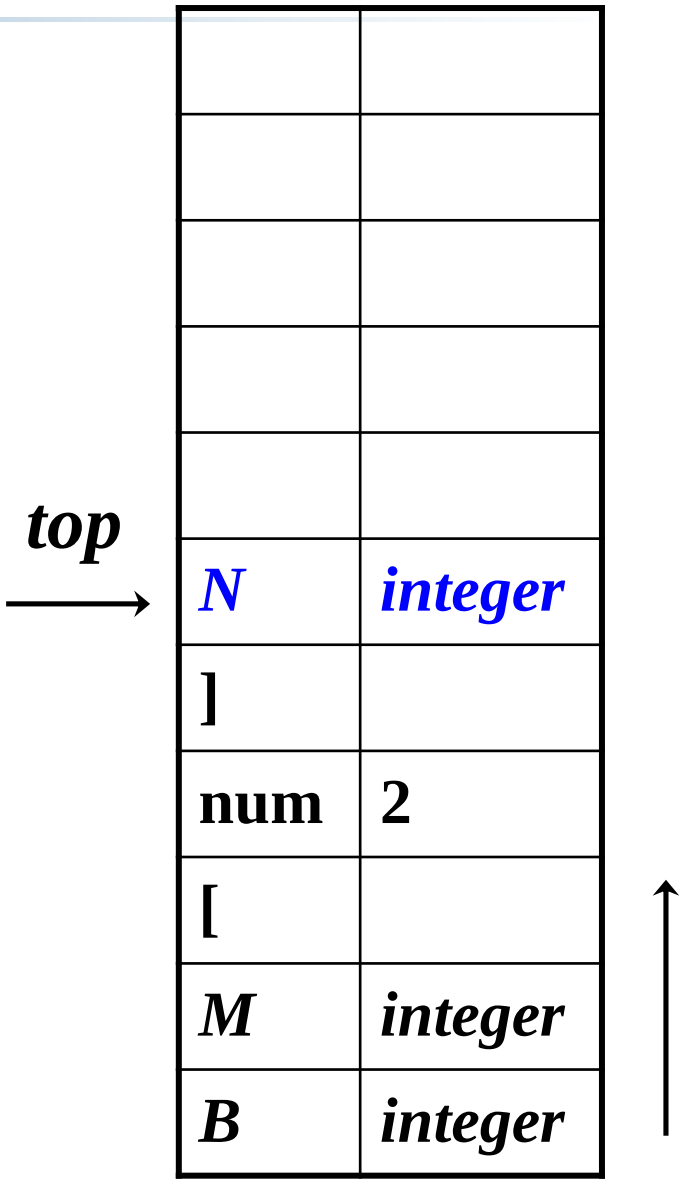
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$

top ⟶

]	
num	3
[	
N	*integer*
]	
num	2
[	
M	*integer*
B	*integer*

栈 *state* *val*

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

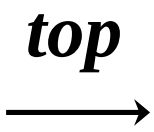
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



N	*integer*
]	
num	3
[	
N	*integer*
]	
num	2
[	
M	*integer*
B	*integer*

栈 *state* *val*

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

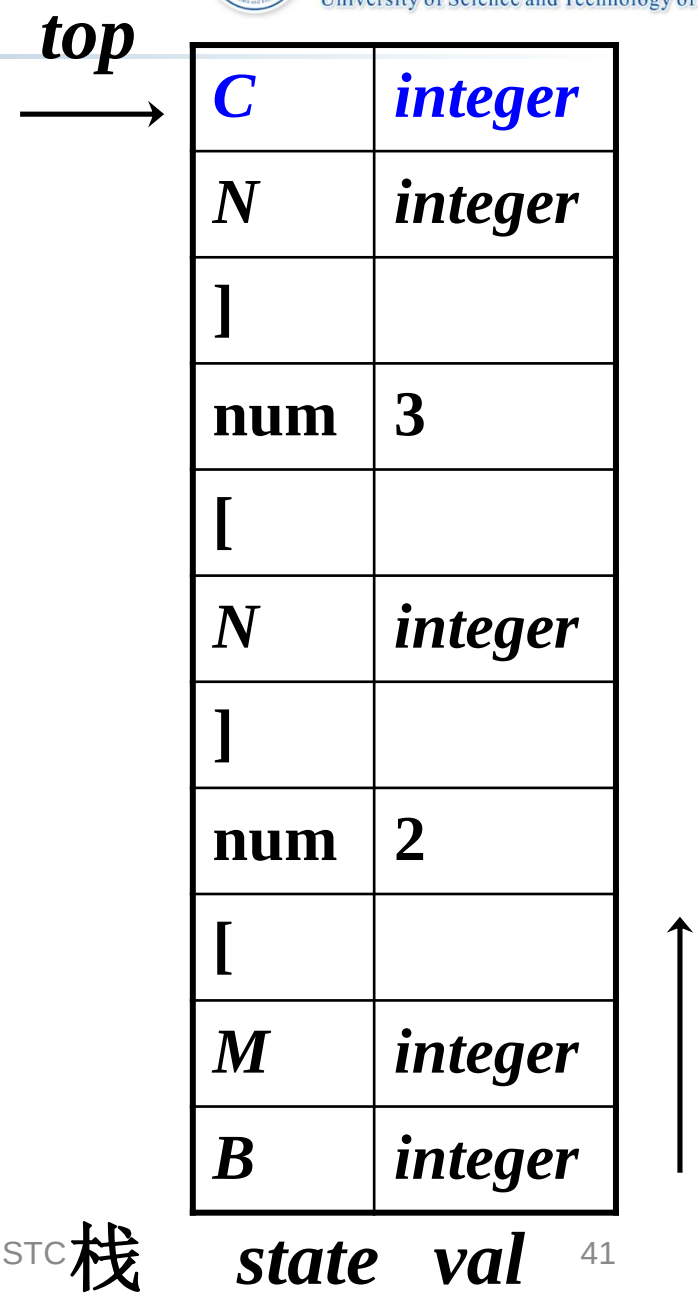
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C$ {$T.t = C.t$; }

$M \rightarrow \varepsilon$ {$M.t = B.t$}

$B \rightarrow$ int {$B.t = integer$}

$B \rightarrow$ float {$B.t = float$}

$C \rightarrow$ [num] $N\ C_1$

{$C.t = array(num.val, C_1.t)$; }

$N \rightarrow \varepsilon$ {$N.t = C.b$}

$C \rightarrow \varepsilon$ {$C.t = C.b$}

完成第一次归约
$C \rightarrow$ [num] $N\ C_1$

top →

栈 state	val
C	array(3, integer)
N	integer
]	
num	2
[	
M	integer
B	integer

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C$ {$T.t = C.t$; }

$M \rightarrow \varepsilon$ {$M.t = B.t$}

$B \rightarrow$ int {$B.t = integer$}

$B \rightarrow$ float {$B.t = float$}

$C \rightarrow$ [num] $N\ C_1$

{$C.t = array(num.val, C_1.t)$; }

$N \rightarrow \varepsilon$ {$N.t = C.b$}

$C \rightarrow \varepsilon$ {$C.t = C.b$}

top →

栈	state	val
C	array(2, array(3,integer))	
M	integer	
B	integer	

完成第二次归约
$C \rightarrow$ [num] $N\ C_1$

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C$ {$T.t = C.t$; }

$M \rightarrow \varepsilon$ {$M.t = B.t$}

$B \rightarrow$ int {$B.t = integer$}

$B \rightarrow$ float {$B.t = float$}

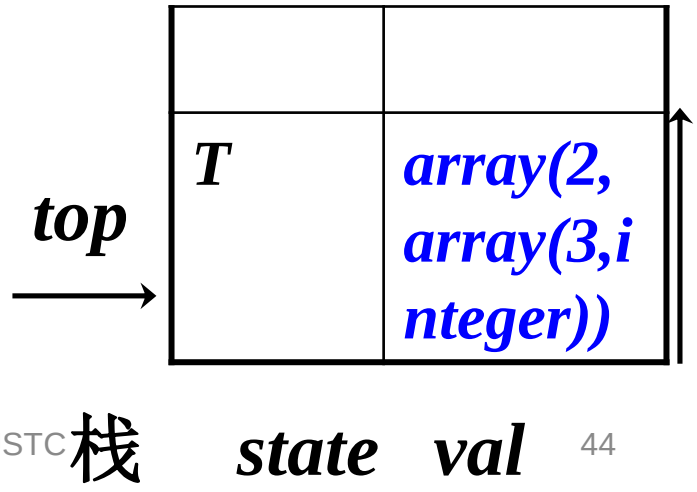
$C \rightarrow$ [num] $N\ C_1$

{$C.t = array(num.val, C_1.t)$; }

$N \rightarrow \varepsilon$ {$N.t = C.b$}

$C \rightarrow \varepsilon$ {$C.t = C.b$}

完成第三次归约
$T \rightarrow B\ M\ C$



主要内容

- 类型表达式
  - 类型的结构
- 类型等价
  - 结构等价和名字等价
- 类型检查
  - 语法制导翻译方案实现
  - 函数和算符的重载
- 其他知识点

层次一：形式化描述类型结构

层次二：判定两个类型相同的依据

层次三：定义一组逻辑规则检查语句或者表达式中是否存在类型错误

结构等价(Structurally equivalance)

- 两个类型表达式完全相同（当无类型名时）

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

结构等价(Structurally equivalance)

- 两个类型表达式完全相同（当无类型名时）
  - 类型表达式树一样

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

结构等价(Structurally equivalance)

- 两个类型表达式完全相同（当无类型名时）
  - 类型表达式树一样
  - 相同的类型构造符作用于相同的子表达式

```
type link = ↑cell;

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

结构等价(Structurally equivalance)

- 两个类型表达式完全相同（当无类型名时）
- 有类型名时，用它们所定义的类型表达式代换它们，所得表达式完全相同(类型定义无环时)

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

这里隐藏了递归检查，因此暂时不考虑有环的情况

next, last, p, q和r结构等价

结构等价(Structurally equivalance)

function sequiv(s, t) : boolean

{if s 和 t 是相同的基本类型then
  return true
else if s == *array*(s_1, s_2) and t == *array*(t_1, t_2) then
  return sequiv(s_1, t_1) and sequiv(s_2, t_2)
else if s == $s_1 \times s_2$ and t == $t_1 \times t_2$ then
  return sequiv(s_1, t_1) and sequiv(s_2, t_2)
else if s == *pointer* (s_1) and t == *pointer*(t_1) then
  return sequiv(s_1, t_1)
else if s == $s_1 \rightarrow s_2$ and t == $t_1 \rightarrow t_2$ then
  return squiv(s_1, t_1) and sequiv(s_2, t_2)
else return false
}

名等价(name equivalance)

- 把每个类型名看成是一个可区别的类型
- 两个类型表达式名字等价当且仅当
  - 它们是相同的基本类型
  - 不进行名字代换就能结构等价

名等价(name equivalance)

- 把每个类型名看成是一个可区别的类型
- 两个类型表达式名字等价当且仅当
  - 它们是相同的基本类型
  - 不进行名字代换就能结构等价

```
type link = ↑cell;          类型表达式

var     next    : link;     link
        last    : link;     link
        p       : ↑cell;    pointer(cell)
        q, r    : ↑cell;    pointer(cell)
```

next和last名字等价

p，q和r名字等价

名等价(name equivalance)

❑Pascal的许多实现用隐含的类型名和每个声明的标识符联系起来

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

实际代码

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

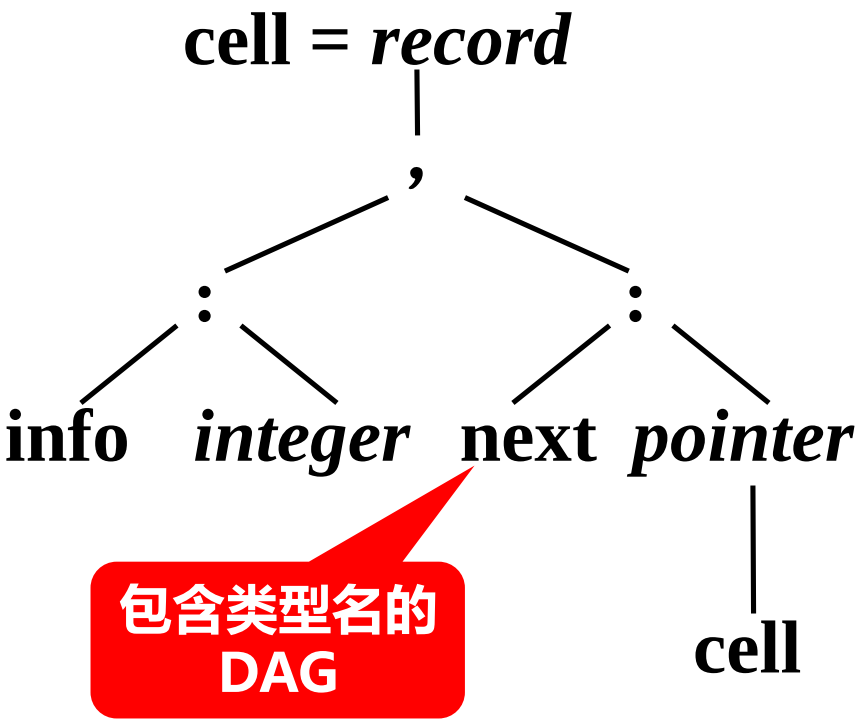
Pascal内部的理解

这时，p与q和r不是名字等价

递归定义的类型

- Where： Linked Lists, Trees, etc
- How: records containing pointers to similar records

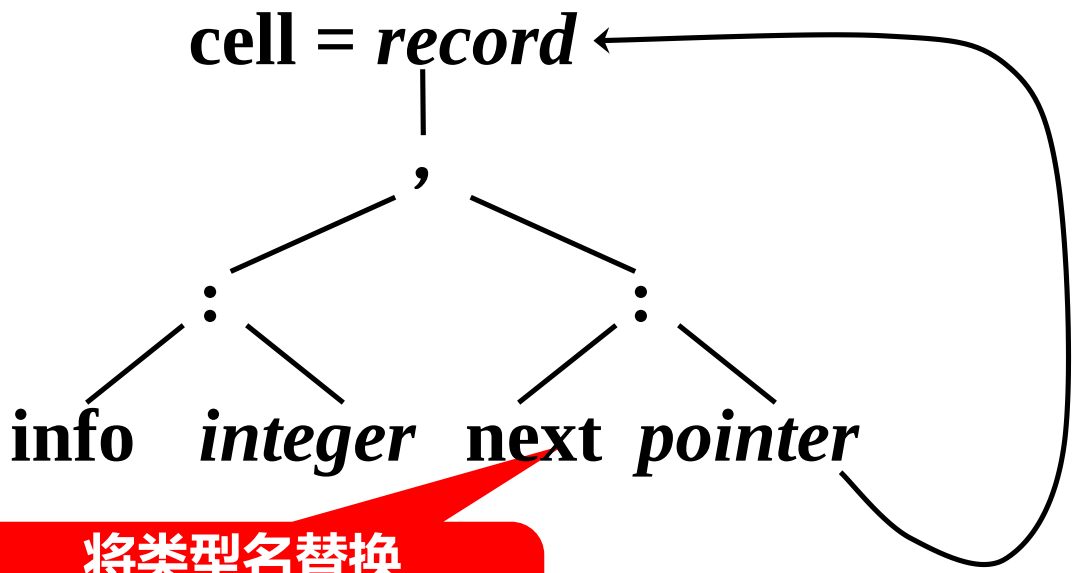
```
type link = ↑ cell ;
cell = record
          info : integer ;
          next : link
end;
```



递归定义的类型

- Where： Linked Lists, Trees, etc
- How: records containing pointers to similar records

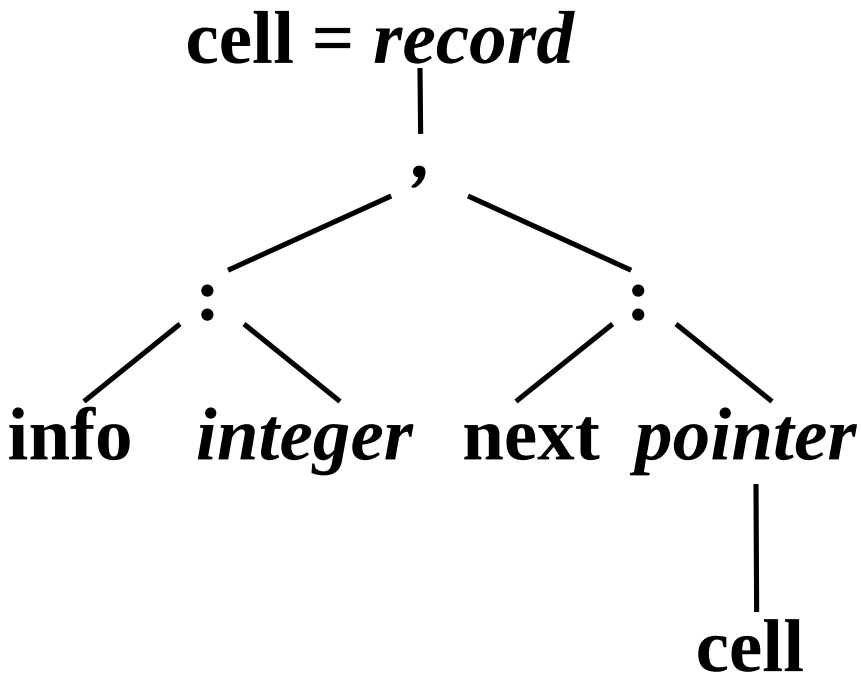
```
type link = ↑ cell ;
cell = record
          info : integer ;
          next : link
end;
```



将类型名替换
引入环，结构等价判定
有可能不终止

C语言中的递归定义的类型

C语言对除记录（结构体）以外的所有类型使用结构等价，而记录类型用的是名字等价，以避免类型图中的环



例题

在X86/Linux机器上，编译器报告最后一行有错误：

incompatible types in return

```
typedef int A1[10];                    | A2 *fun1( ) {
typedef int A2[10];                    |     return(&a);
A1 a;                                  | }
typedef struct {int i;}S1;             | S2  fun2( ) {
typedef struct {int i;}S2;             |     return(s);
S1 s;                                  | }
```

在C语言中，数组和结构体都是构造类型，为什么上面第2个函数有类型错误，而第1个函数却没有?

主要内容

- 类型表达式
  - 类型的结构
- 类型等价
  - 结构等价和名字等价
- 类型检查
  - 语法制导翻译方案实现
  - 函数和算符的重载
- 其他知识点

层次一：形式化描述类型结构

层次二：判定两个类型相同的依据

层次三：定义一组逻辑规则检查语句或者表达式中是否存在类型错误

一个简单的语言

$P \rightarrow D\ ;\ S$

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

$T \rightarrow \text{boolean} \mid \text{integer} \mid \text{array [num] of } T \mid$

$\uparrow T \mid T\ \text{‘}\rightarrow\text{’}\ T$

$S \rightarrow \text{id} := E \mid \text{if } E \text{ then } S \mid \text{while } E \text{ do } S \mid S\ ;\ S$

$E \rightarrow \text{truth} \mid \text{num} \mid \text{id} \mid E \text{ mod } E \mid E\ [\ E\ ] \mid$

$E\uparrow \mid E\ (E\ )$

例

```
i : integer;
j : integer;
j := i mod 2000
```

类型检查——声明语句

$D \rightarrow D; D$

$D \rightarrow \text{id} : T \ \{addtype\ (\text{id}.entry, T.type)\}$

addtype：把类型信息填入符号表

类型检查——声明语句

$D \rightarrow D; D$

$D \rightarrow \text{id} : T \ \{addtype\ (\text{id}.entry, T.type)\}$

$T \rightarrow \text{boolean} \quad \{T.type = boolean\}$

$T \rightarrow \text{integer} \quad \{T.type = integer\}$

$T \rightarrow \uparrow T_1 \quad \{T.type = pointer(T_1.type)\}$

类型检查——声明语句

$D \rightarrow D; D$

$D \rightarrow \text{id} : T \ \{addtype\ (\text{id}.entry, T.type)\}$

$T \rightarrow \text{boolean} \quad \{T.type = boolean\}$

$T \rightarrow \text{integer} \quad \{T.type = integer\}$

$T \rightarrow \uparrow T_1 \quad \{T.type = pointer(T_1.type)\}$

$T \rightarrow \text{array [num] of } T_1$

$\{T.type = array(\text{num}.val, T_1.type)\}$

类型检查——声明语句

$D \rightarrow D;\ D$

$D \rightarrow \text{id} : T$ {*addtype* (id.*entry*, *T.type*)}

$T \rightarrow \text{boolean}$ {*T.type* = *boolean*}

$T \rightarrow \text{integer}$ {*T.type* = *integer*}

$T \rightarrow \uparrow T_1$ {*T.type* = *pointer*(T_1.*type*)}

$T \rightarrow \text{array [num] of } T_1$

{*T.type* = *array*(num.*val*, T_1.*type*)}

$T \rightarrow T_1$ '$\rightarrow$' T_2 {*T.type* = T_1.*type* $\rightarrow$ T_2.*type* }

类型检查——表达式

$E \rightarrow$ truth $\quad \{E.type = boolean\}$

$E \rightarrow$ num $\quad \{E.type = integer\}$

$E \rightarrow$ id $\quad \{E.type = lookup(\text{id}.entry)\}$

类型检查——表达式

$E \rightarrow$ truth $\quad \{E.type = boolean\}$

$E \rightarrow$ num $\quad \{E.type = integer\}$

$E \rightarrow$ id $\quad \{E.type = lookup(\text{id}.entry)\}$

$E \rightarrow E_1 \text{ mod } E_2$

$\{E.type = \text{if } E_1.type == integer \text{ and}$

$E_2.\ type == integer \text{ then } integer$

$\text{else } type\_error\ \}$

类型检查——表达式

$$E \rightarrow E_1\ [E_2\ ]\ \{E.type = \text{if } E_2.\ type == integer \text{ and}$$

$$E_1.\ type == array(s,\ t) \text{ then } t$$

$$\text{else } type\_error\ \}$$

类型检查——表达式

$$E \rightarrow E_1\,[E_2\,]\,\{E.type = \text{if } E_2.type == integer \text{ and } E_1.type == array(s, t) \text{ then } t \text{ else } type\_error\,\}$$

$$E \rightarrow E_1\uparrow\,\{E.type = \text{if } E_1.type == pointer(t) \text{ then } t \text{ else } type\_error\,\}$$

类型检查——表达式

$E \rightarrow E_1\,[E_2\,]$ {*E.type* = if E_2. *type* == *integer* 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* }

$E \rightarrow E_1\,(E_2\,)$ {*E*. *type* = if E_2 . *type* == *s* and
E_1. *type* == $s \rightarrow t$ then *t*
else *type_error* }

类型检查——语句

$$S \rightarrow \text{id} := E \{ \text{if } (id.type == E.type \;\&\&\; E.type \in \{boolean, integer\}) \; S.type = void;$$

$$\text{else } S.type = type\_error;\}$$

类型检查——语句

$S \rightarrow$ id := E { if (*id.type* == *E.type* && *E.type* $\in$ {*boolean*, *integer*}) *S.type* = *void*;

else *S.type* = *type_error*;}

$S \rightarrow$ if E then S_1 {*S. type* = if *E. type* == *boolean*

then S_1. *type*

else *type_error* }

类型检查——语句

$S \rightarrow$ **while** E **do** S_1

{$S.type$ = **if** $E.type$ == $boolean$ **then** $S_1.type$

else $type\_error$ }

类型检查——语句

$S \rightarrow$ **while** E **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* }

类型检查——程序

$$P \rightarrow D; S$$

$$\{P.\ type = \text{if } S.type == void \text{ then } void$$

$$\text{else } type\_error \}$$

主要内容

- 类型表达式
  - 类型的结构
- 类型等价
  - 结构等价和名字等价
- 类型检查
  - 语法制导翻译方案实现
  - 函数和算符的重载
- 其他知识点

层次一：形式化描述类型结构

层次二：判定两个类型相同的依据

层次三：定义一组逻辑规则检查语句或者表达式中是否存在类型错误

函数和算符的重载

❑重载符号

- ❖有多个含义，但在每个引用点的含义都是唯一的

❑例如：

- ❖加法算符+可用于不同类型，"+"是多个函数的名字，而不是一个多态函数的名字

❑重载的消除

- ❖在重载符号的引用点，其含义能确定到唯一

子表达式的可能类型集合

❑例 Ada语言的声明:

function “*” (i, j : integer) return complex;

function “*” (x, y : complex) return complex;

使得算符*重载，可能的类型包括:

integer × integer → integer --这是预定义的类型

integer × integer → complex

complex × complex → complex

2 * (3 * 5)

(3 * 5) * z z是复型

子表达式的可能类型集合

- 以函数应用为例，考虑类型检查
  - 在每个表达式都有唯一的类型时，函数应用的类型检查是：

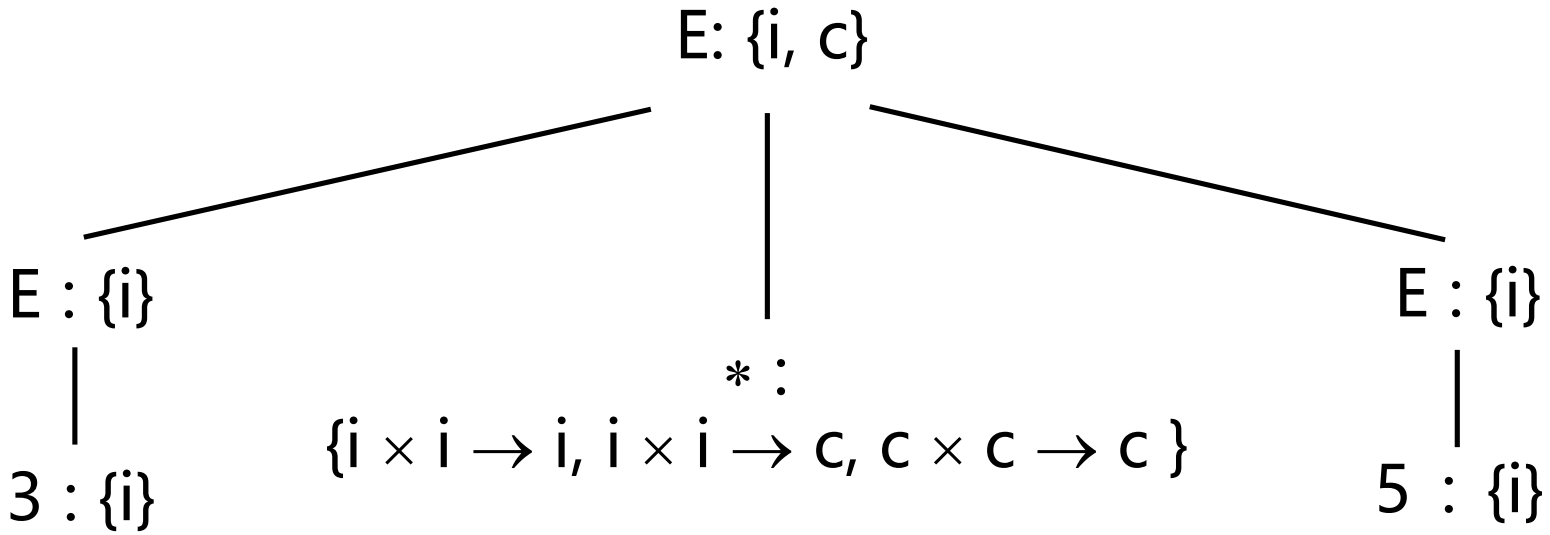
$E \rightarrow E_1(E_2)$ { E.type = if E_2.type == s and E_1.type == $s \rightarrow t$ then t else type_ error }

  - 确定表达式可能类型的集合（类型可能不唯一）

产 生 式	语 义 规 则
$E' \rightarrow E$	E′.types = E. types
$E \rightarrow id$	E. types = lookup(id. entry)
$E \rightarrow E_1(E_2)$	E. types = {t | E_2. types中存在一个s，使得s → t属于E_1.types }

子表达式的可能类型集合

- 例：表达式3 * 5可能的类型集合



缩小可能类型的集合

$E' \rightarrow E$ {E'.types = E. types

E.unique = if E'.types = {t} then t else error}

$E \rightarrow id$ {E.types = lookup(id.entry)}

$E \rightarrow E_1(E_2)$ {E.types = {s' | ∃ s ∈ E_2.types and

$s \rightarrow s' \in E_1$.types}

t = E.unique

S = {s | s ∈ E_2.types and S→t ∈ E_1.types}

E_2.unique = if S ={s} then S else error

E_1.unique = if S = {s} then S→t else error

主要内容

- 类型表达式
  - 类型的结构
- 类型等价
  - 结构等价和名字等价
- 类型检查
  - 语法制导翻译方案实现
  - 函数和算符的重载
- 其他知识点

层次一：形式化描述类型结构

层次二：判定两个类型相同的依据

层次三：定义一组逻辑规则检查语句或者表达式中是否存在类型错误

❑控制流检查

- ❖控制流语句必须使控制转移到合法的地方。
- ❖例如，在C语言中break语句使控制跳离包括该语句的最小while、for或者switch语句；否则就报错。

```
main() {
printf("\n%ld\n",gcd(4,12));
continue;
}
```

编译时的报错如下：

```
example.c: In function 'main':
example.c:3: continue statement not within a loop
```

其他静态语义分析

- 上下文相关检查
  - 标识符没有声明
  - 标识符重复声明
- 唯一性检查
  - Switch语句的分支常量表达式不能有重复
  - 枚举类型的元素不能重复

例题

❑编译时的唯一性检查的例子

```
main() {
  int i;
  switch(i){
  case 10: printf("%d\n", 10); break;
  case 20: printf("%d\n", 20); break;
  case 10: printf("%d\n", 10); break;
  }
}
```

❖编译时的报错如下：

```
switch.c: In function 'main':
switch.c:6: duplicate case value
switch.c:4: this is the first entry for that value
```

《编译原理与技术》
类型及其应用
(静态类型检查)

The end!