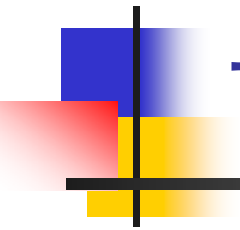


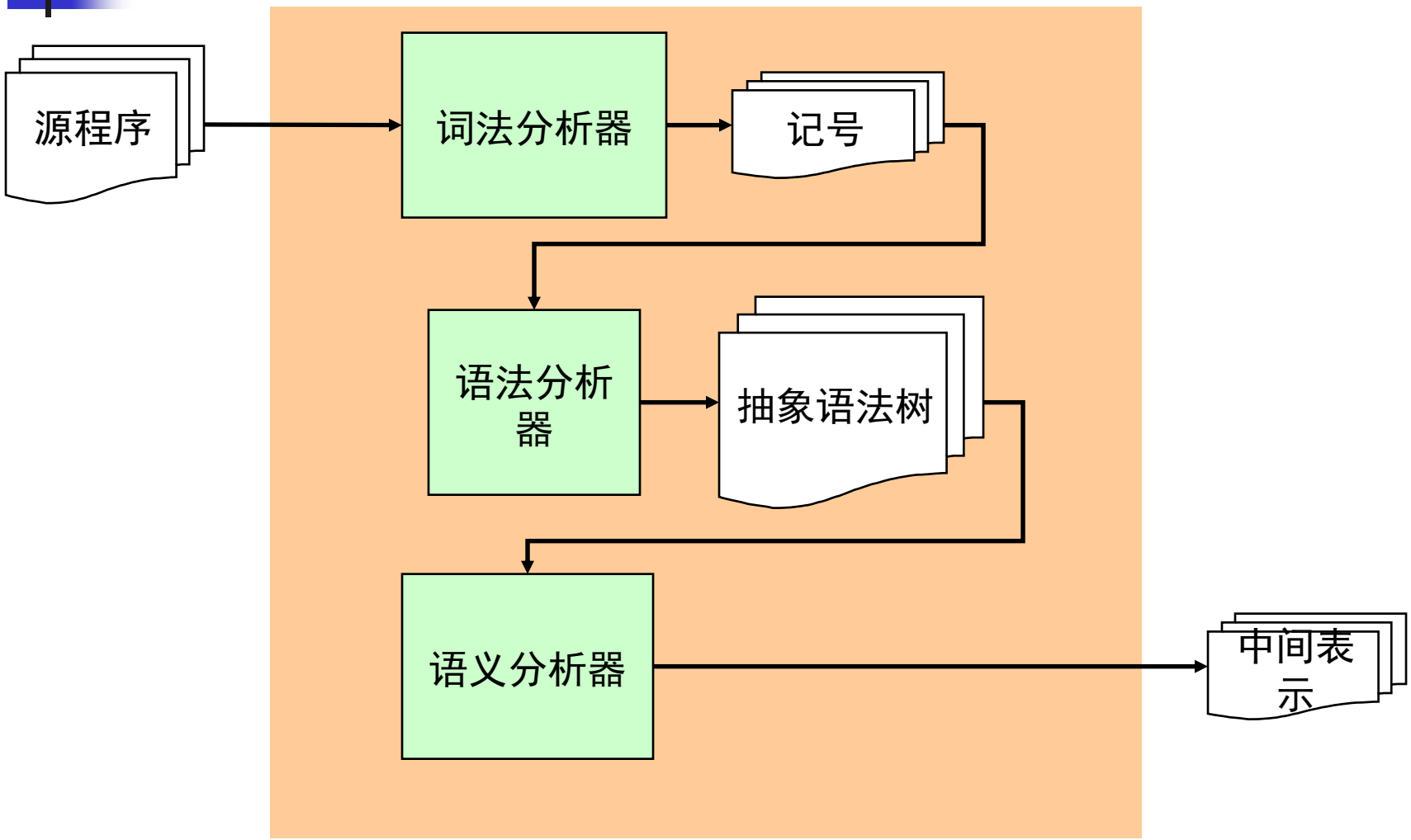
语义分析：语义检查

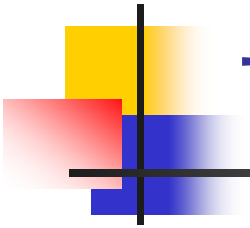
编译原理

华保健

bjhua@ustc.edu.cn

前端





语义分析的任务

- 语义分析也称为类型检查、上下文相关分析
- 负责检查程序（语法树）的上下文相关的属性：
  - 这是具体语言相关的，典型的情况包括：
    - 变量在使用前先进行声明
    - 每个表达式都有合适的类型
    - 函数调用和函数的定义一致
    - …

C--语言

```
E -> n
   | true
   | false
   | E + E
   | E && E
```

```
// 类型合法的程序：
3+4
false && true

// 类型不合法的程序：
3 + true
true + false

// 对这个语言，语义分析的任务是：对给定的一个表达式e，写一
// 个函数
type check(e);
// 返回表达式e的类型；若类型不合法，则报错。
```

类型检查算法

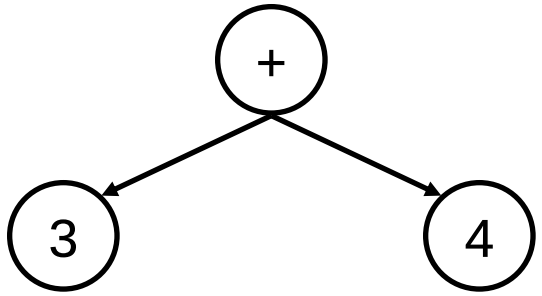
```
E -> n
   | true
   | false
   | E + E
   | E && E
```

```
enum type {INT, BOOL};

enum type check_exp (Exp_t e)
  switch(e->kind)
    case EXP_INT: return INT;
    case EXP_TRUE: return BOOL;
    case EXP_FALSE: return BOOL;
    case EXP_ADD: t1 = check_exp (e->left);
                  t2 = check_exp (e->right);
                  if (t1!=INT || t2!=INT)
                    error (“type mismatch”);
                  else return INT;
    case EXP_AND: … // 类似；留作练习
```

示例1

```
E -> n
   | true
   | false
   | E + E
   | E && E
```

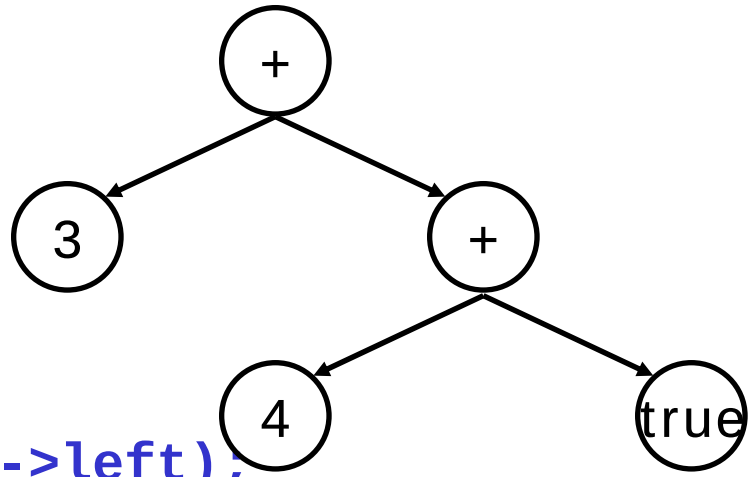
```
enum type {INT, BOOL};

enum type check_exp (Exp_t e)
  switch(e->kind)
    case EXP_INT: return INT;
    case EXP_TRUE: return BOOL;
    case EXP_FALSE: return BOOL;
    case EXP_ADD: t1 = check_exp (e->left);
                  t2 = check_exp (e->right);
                  if (t1!=INT || t2!=INT)
                    error (“type mismatch”);
                  else return INT;
    case EXP_AND: … // 类似；留作练习
```



示例2

```
E -> n
   | true
   | false
   | E + E
   | E && E
```

```
enum type {INT, BOOL};

enum type check_exp (Exp_t e)
  switch(e->kind)
    case EXP_INT: return INT;
    case EXP_TRUE: return BOOL;
    case EXP_FALSE: return BOOL;
    case EXP_ADD: t1 = check_exp (e->left);
                  t2 = check_exp (e->right);
                  if (t1!=INT || t2!=INT)
                    error (“type mismatch”);
                  else return INT;
    case EXP_AND: … // 类似；留作练习
```



变量声明的处理

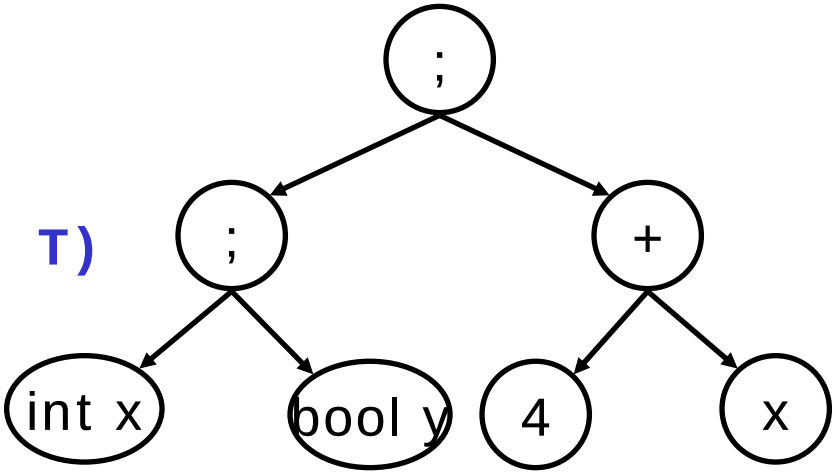
```
// 类型合法的程序：
int x;
x+4
// 类型合法的程序：
bool y;
false && y

// 类型不合法的程序：
x + 3
// 类型不合法的程序：
int x;
x + false
```

```
P -> D E
D -> T id; D
   |
T -> int
   | bool
E -> n
   | id
   | true
   | false
   | E + E
   | E && E
```

类型检查算法

```
P -> D E
D -> T id; D
   |
T -> int
   | bool
E -> n
   | id
   | true
   | false
   | E + E
   | E && E
```

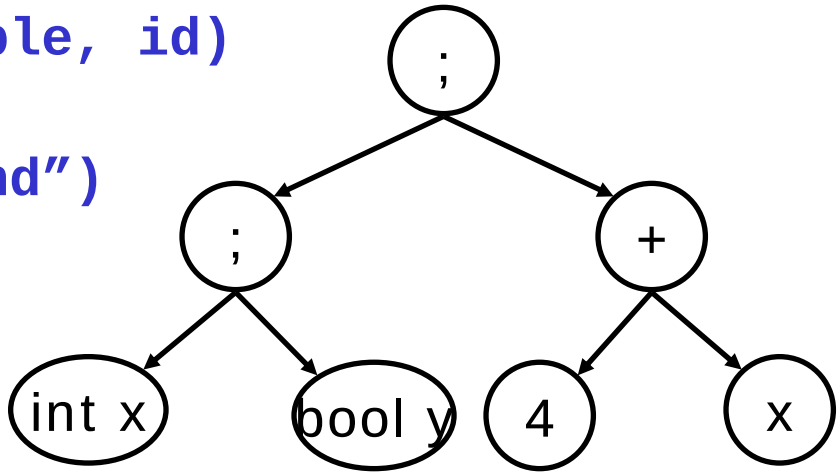
```
enum type {INT, BOOL};
Table_t table;

enum type check_prog (Dec_t d, Exp_t e)
  table = check_dec (d)
  return check_exp (e)

Table_t check_dec (Dec_t d)
  foreach (T id ∈ d)
    table_enter (table, id, T)
```



类型检查算法（续）

```
P -> D E
D -> T id; D
   |
T -> int
   | bool
E -> n
   | id
   | true
   | false
   | E + E
   | E && E
```

```
enum type {INT, BOOL};
Table_t table;

enum type check_exp (Exp_t e)
  switch (e->kind)
    case EXP_ID:
      t = Table_lookup (table, id)
      if (id not exist)
        error (“id not found”)
      else return t
```



语句的处理

```
void check_stm (Table_t table, Stm_t s)
  switch(s->kind)
    case STM_ASSIGN:
      t1 = Table_lookup (s->id)
      t2 = check_exp (table, s->exp)
      if (t1!=t2)
        error(“type mismatch”)
      else return INT;
    case STM_PRINTI:
      t = check_exp(s->exp)
      if (t!=INT)
        error (“type mismatch”)
      else return;
    case STM_PRINTB: … // 类似；留作练习
```

```
P -> D S
D -> T id; D
   |
T -> int
   | bool
S -> id = E
   | printi (E)
   | printb (E)
E -> n
   | id
   | true
   | false
   | E + E
   | E && E
```