

COMP130014.02 编译

第六讲：类型推导

徐辉

xuh@fudan.edu.cn



大纲

- ❖ 一、类型推导问题
- ❖ 二、识别标识符作用域
- ❖ 三、类型约束和求解

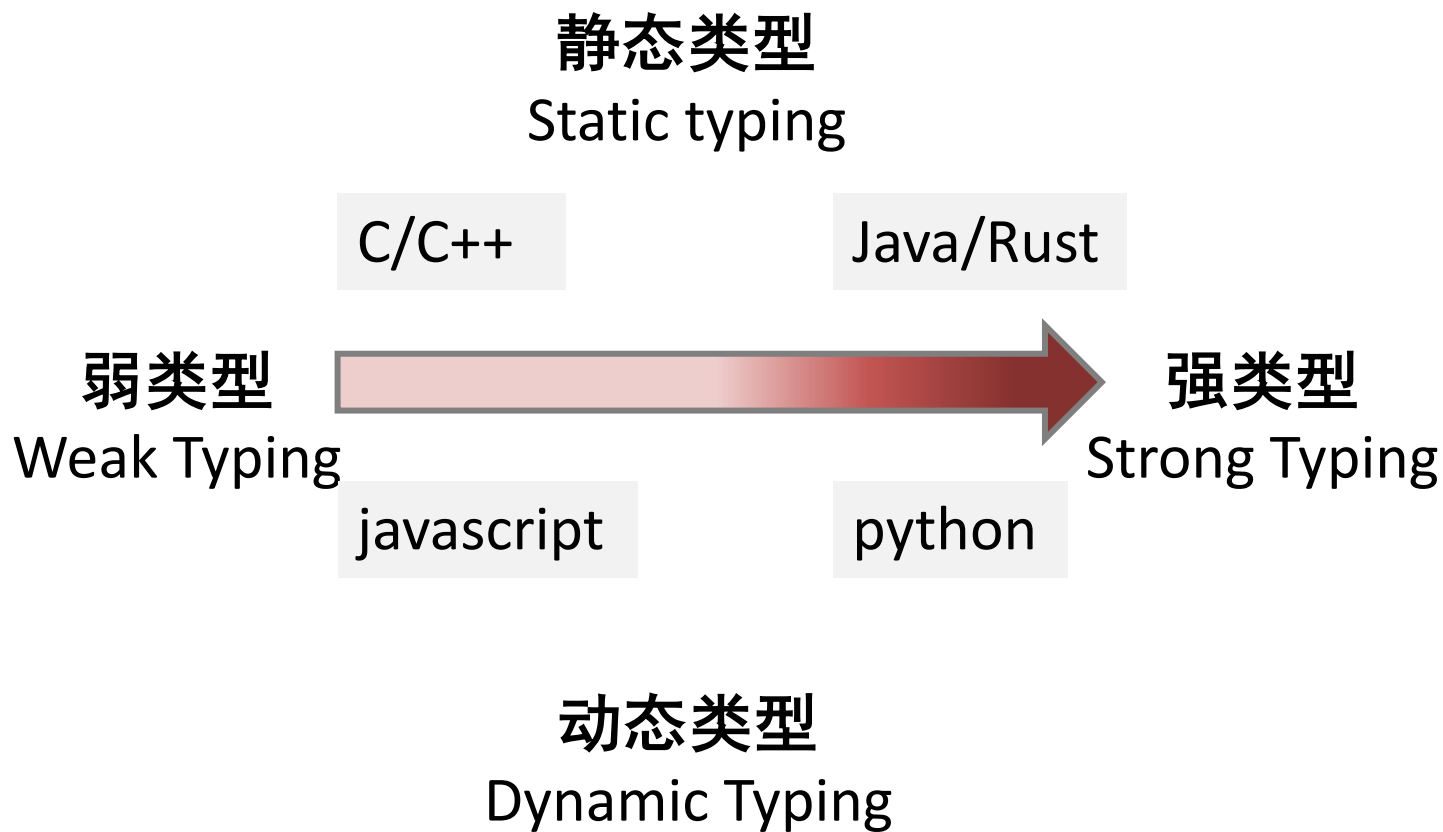
一、类型推导问题



类型系统基本概念

- 类型系统包括由类型和规则组成
- 类型：
 - 基础类型（Primitive Type）
 - 标量类型（Scalar Types）： bool/char/int/long/float/double
 - 复合类型（Compound Type）： 数组、元组
 - 自定义类型： 结构体/类
- 类型规则：
 - 类型推导和检查规则
 - 是否允许隐式类型转换？

类型系统分类



动态类型 vs 静态类型

- 静态类型系统：编译时检查类型的一致性
- 动态类型系统：运行时检查类型的一致性

//python代码

```
def foo(x):  
    if x == 1:  
        return "bingo!"  
    return x
```

//foo的类型是什么?

```
print(foo(10))  
print(foo(1))  
print(foo(10) + foo(1))
```

```
#: python factorial.py
```

```
10
```

```
bingo!
```

```
Traceback (most recent call last):
```

```
File "factorial.py", line 11, in <module>
```

```
    print(foo(10) + foo(1))
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

强类型 vs 弱类型（相对）

- 强类型系统：一般不允许隐式类型转换，如Python、Java
- 弱类型类型：自动隐式转换，灵活但易出错，如C、Javascript

//python代码

```
b = 1 + True;  
a = 1 + '2';  
c = '1' + True;
```

2
类型错误
类型错误

//C代码

```
int a = 1 + true;  
int b = '1' + true;  
int c = 1 + '2';  
int d = 1 + "2";
```

2
50
51
4202501

//Javascript代码

```
1 + true;  
1 + '2';  
'1' + true;
```

2
'12'
'1true'

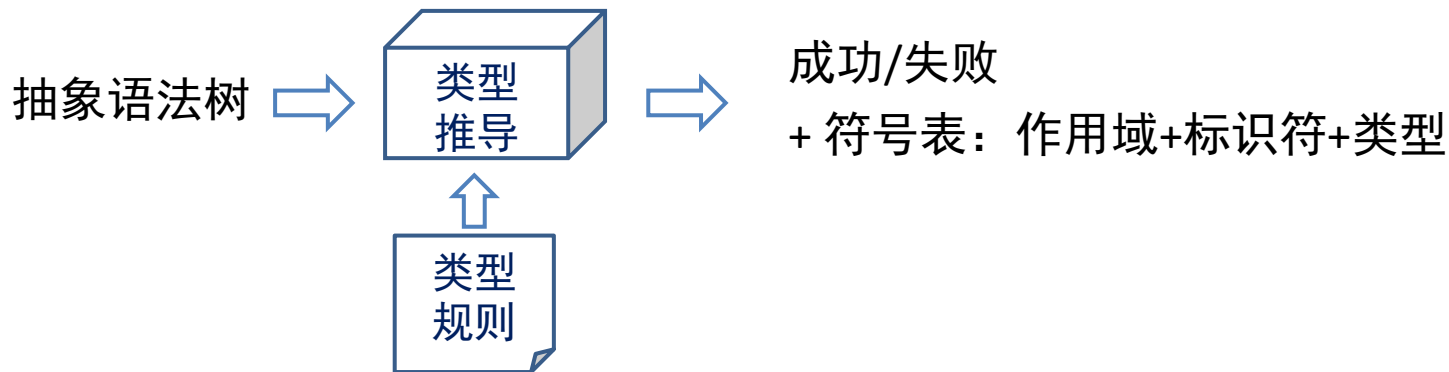
//Javascript代码

```
var a = 42;  
var b = "42";  
var c = [42];  
a === b;  
a == b;  
a == c;
```

false
true
true

类型推导问题

- 已知源代码（一般是抽象语法树形式）和类型规则
- 为代码中的所有标识符或对象找到满足类型规则的唯一解
 - 如满足运算符、函数签名等要求
- 类型检查可以认为是类型推导问题的特例



类型推导思路

- 基于AST对标识符进行作用域识别和类型分析
 - 声明新标识符：确定作用域
 - 使用标识符：推导或检查类型

```
let g:int = 10;
fn fib(x: int) -> int {
  if (x <= 1) {
    ret x;
  }
  let a = fib(x - 1);
  let b = fib(x - 2);
  let r = a + b;
  ret r;
}
fn main() {
  let r = fib(10) + g;
}
```

标识符	作用域 (粗粒度)	索引	类型
g	global	0xd9c2	int
fib	global	0xd470	(int) → int
main	global	0xd318	(void) → void
x	fib	0xd398	int
a	fib	0xd5b0	int
b	fib	0xd2c2	int
r	fib	0x1234	int
r	main	0xa3c2	int

二、识别标识符作用域

作用域分析（细粒度）

```
let g:int = 10;
fn foo(x: int) -> int { //scope fib
  if (x <= 1) {
    ret x;
  }
  let a = fib(x - 1); //{scope 1
    let b = fib(x - 2); //{ scope 2
      let r = a + b; //{scope 3
        ret r;
      //}
    //}
  //}
}

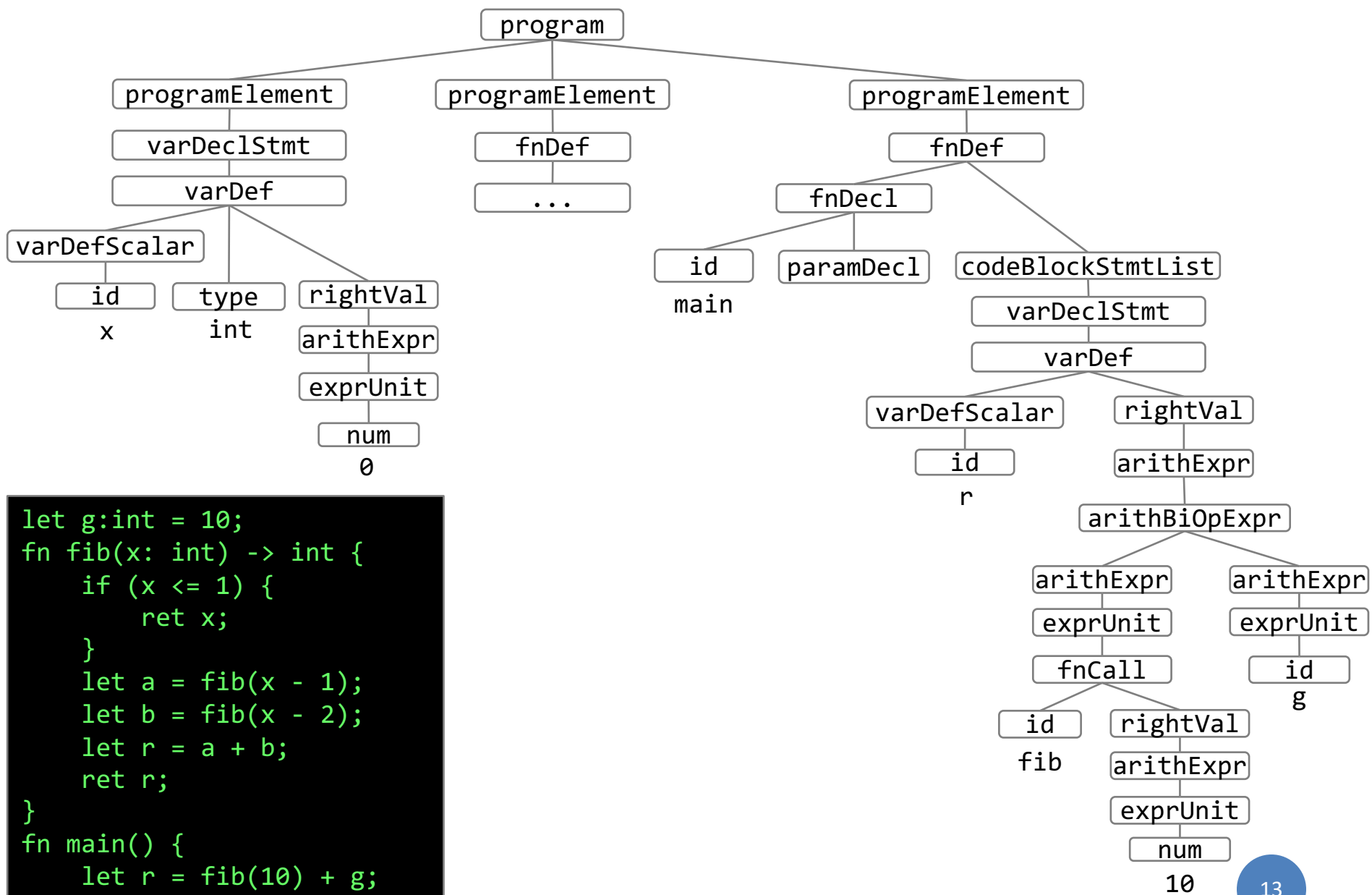
fn main() { //scope main
  let r = fib(10) + g;
}
```

Scope之间的偏序关系: fib > 1 > 2 > 3

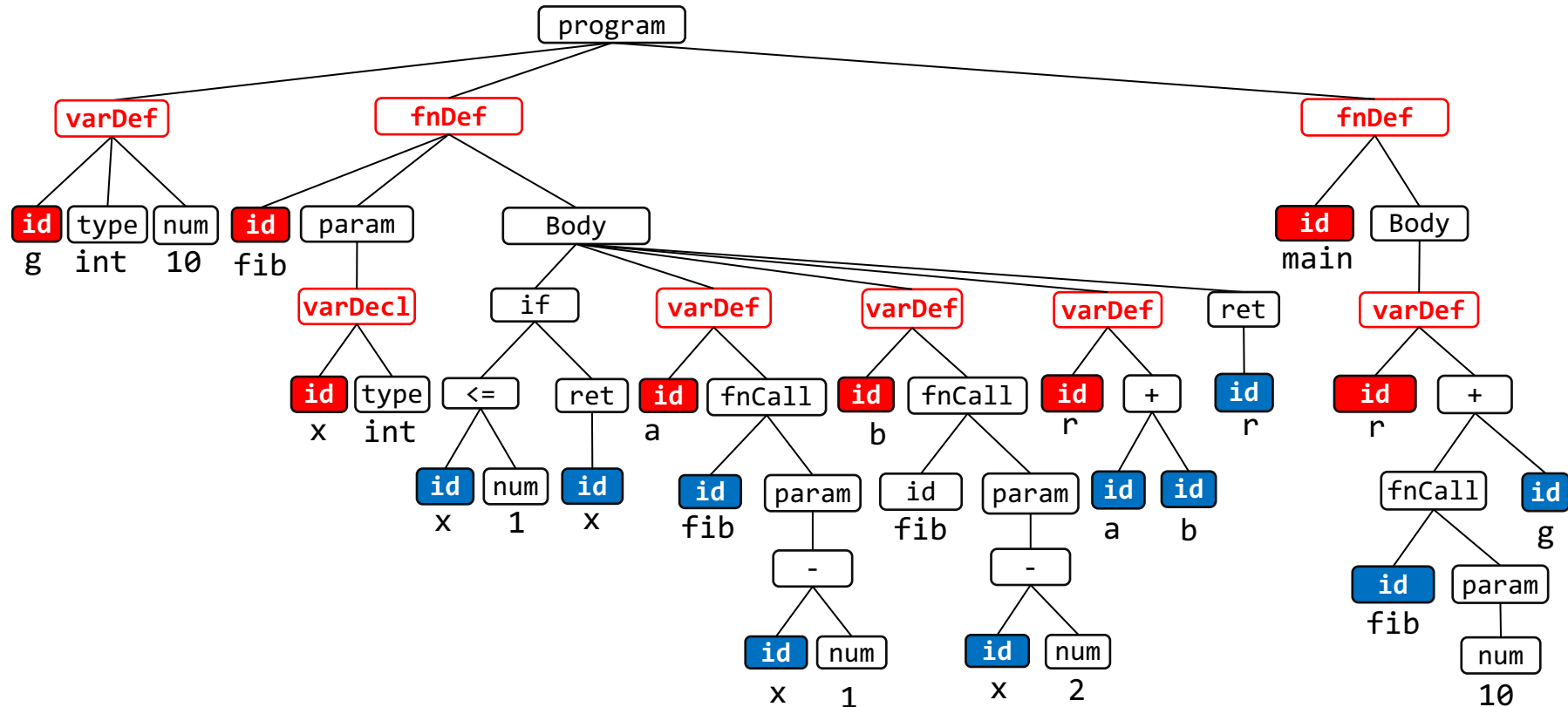
抽象语法树：Abstract Syntax Tree

- 具体语法：程序员实际写的代码
 - 解析源代码得到语法解析树，是对源代码的完整表示
- 抽象语法：编译器实际需要的内容
- 抽象语法树：消除解析过程中的一些步骤或节点
 - 单一展开形式塌陷，如 $E_1 \rightarrow E_2 \rightarrow E_3 \rightarrow \text{NUM} \rightarrow \langle \text{UNUM} \rangle$
 - 去掉括号等冗余信息
 - 运算符和保留字一般不作为叶子结点
- 可被编译器后续编辑，记录上下文相关的信息

示例：TeaPL编译器生成的语法树

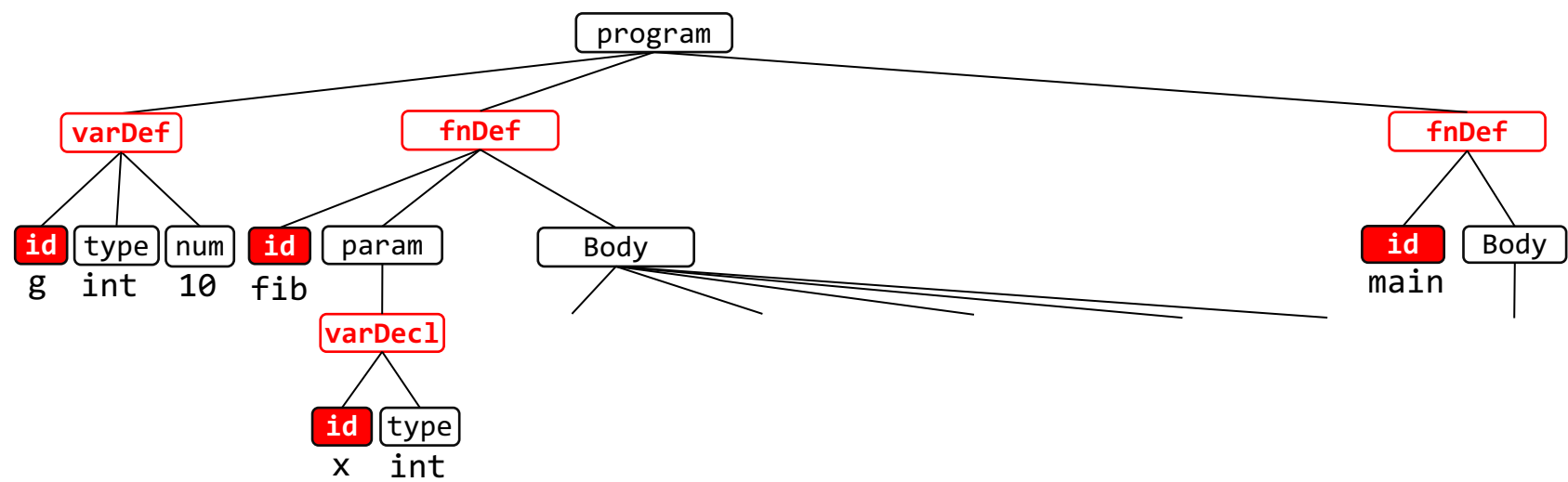


语法树化简=>问题抽象



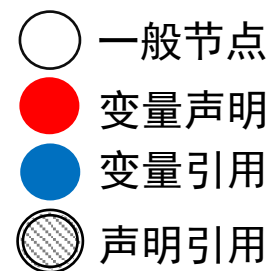
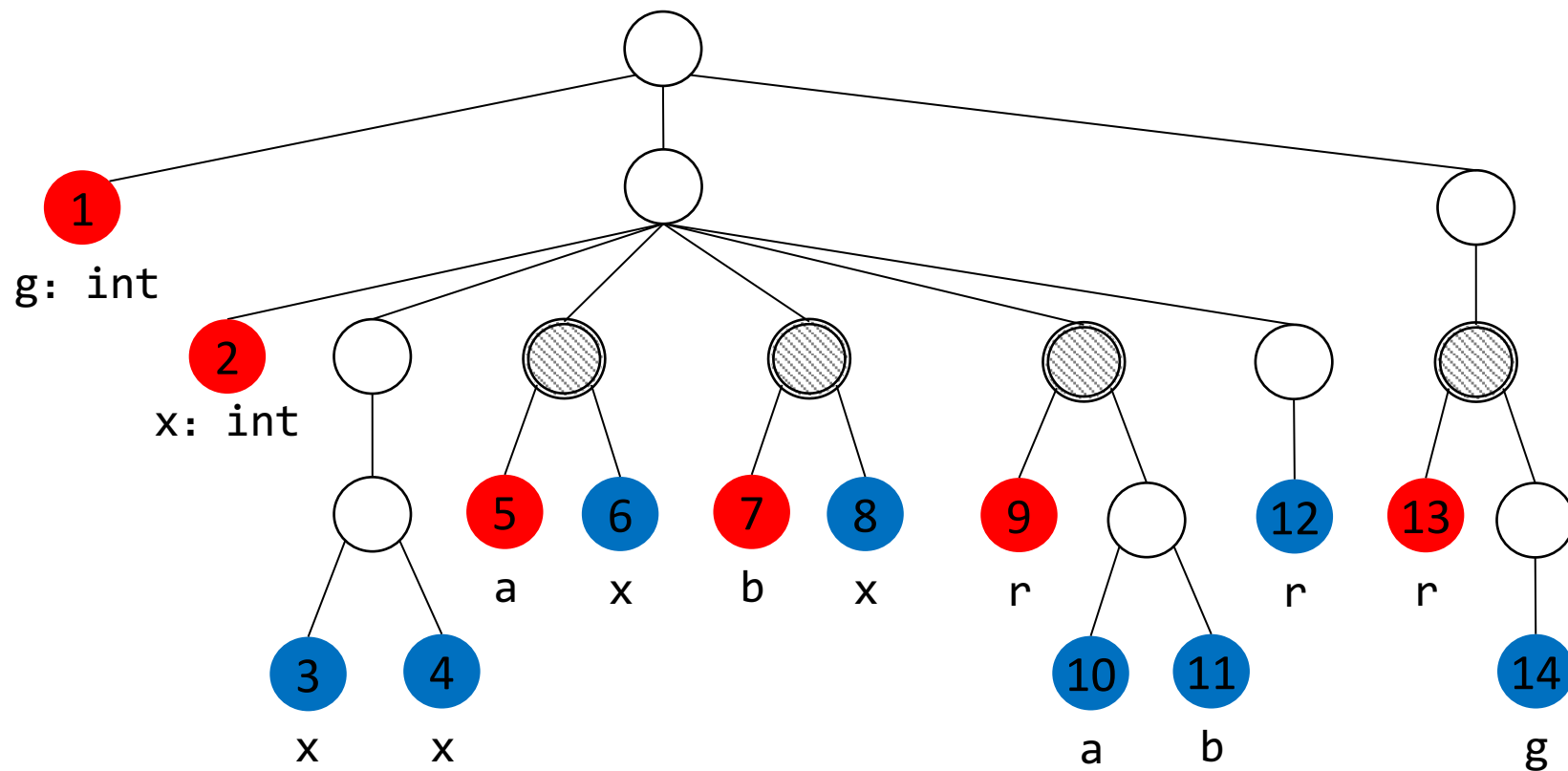
● 变量声明
● 变量引用

函数/全局变量符号表

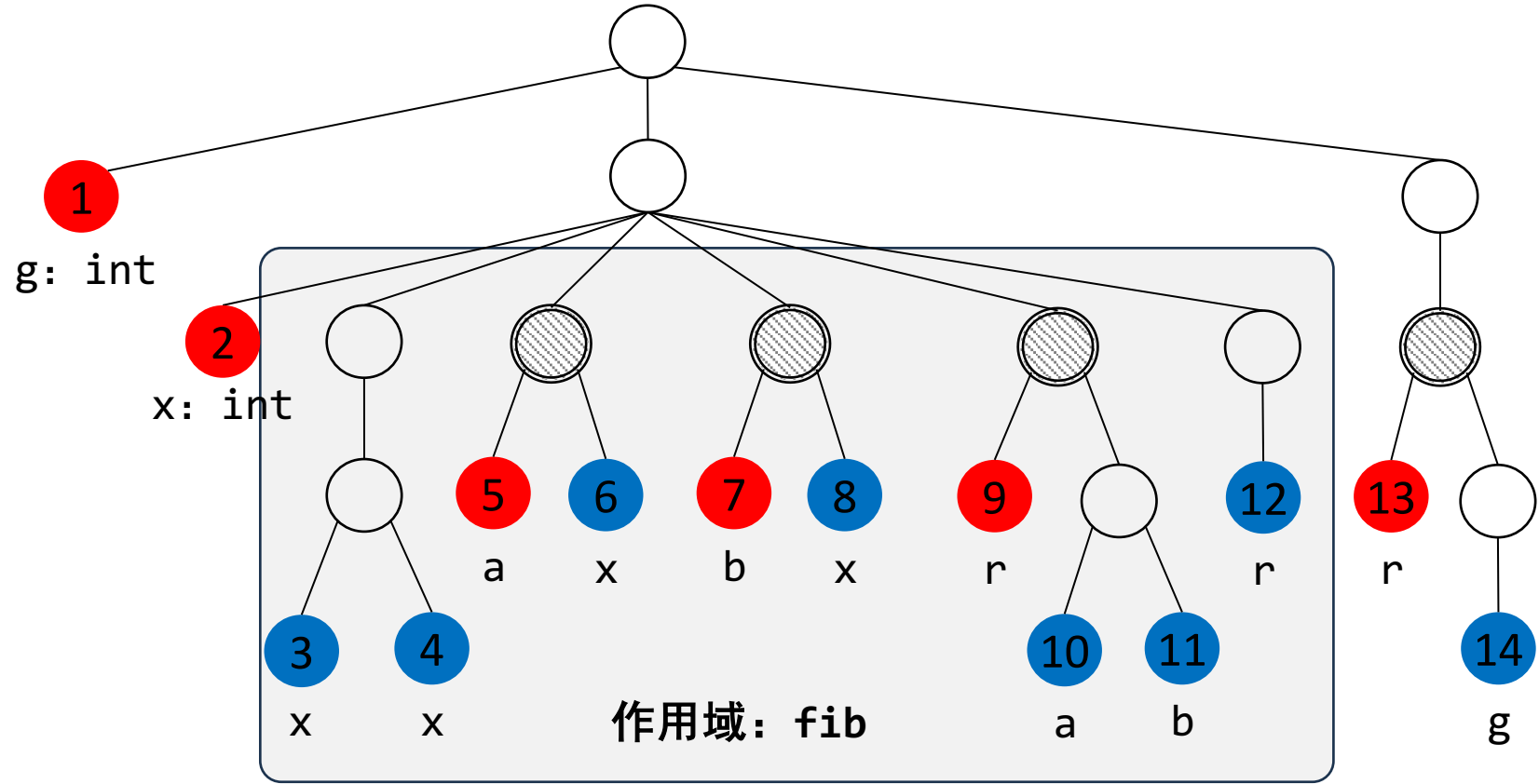


标识符	作用域	索引	类型
g	global	0xd9c2	int
fib	global	0xd470	(int) → int
main	global	0xd318	(void) → void

局部变量类型分析



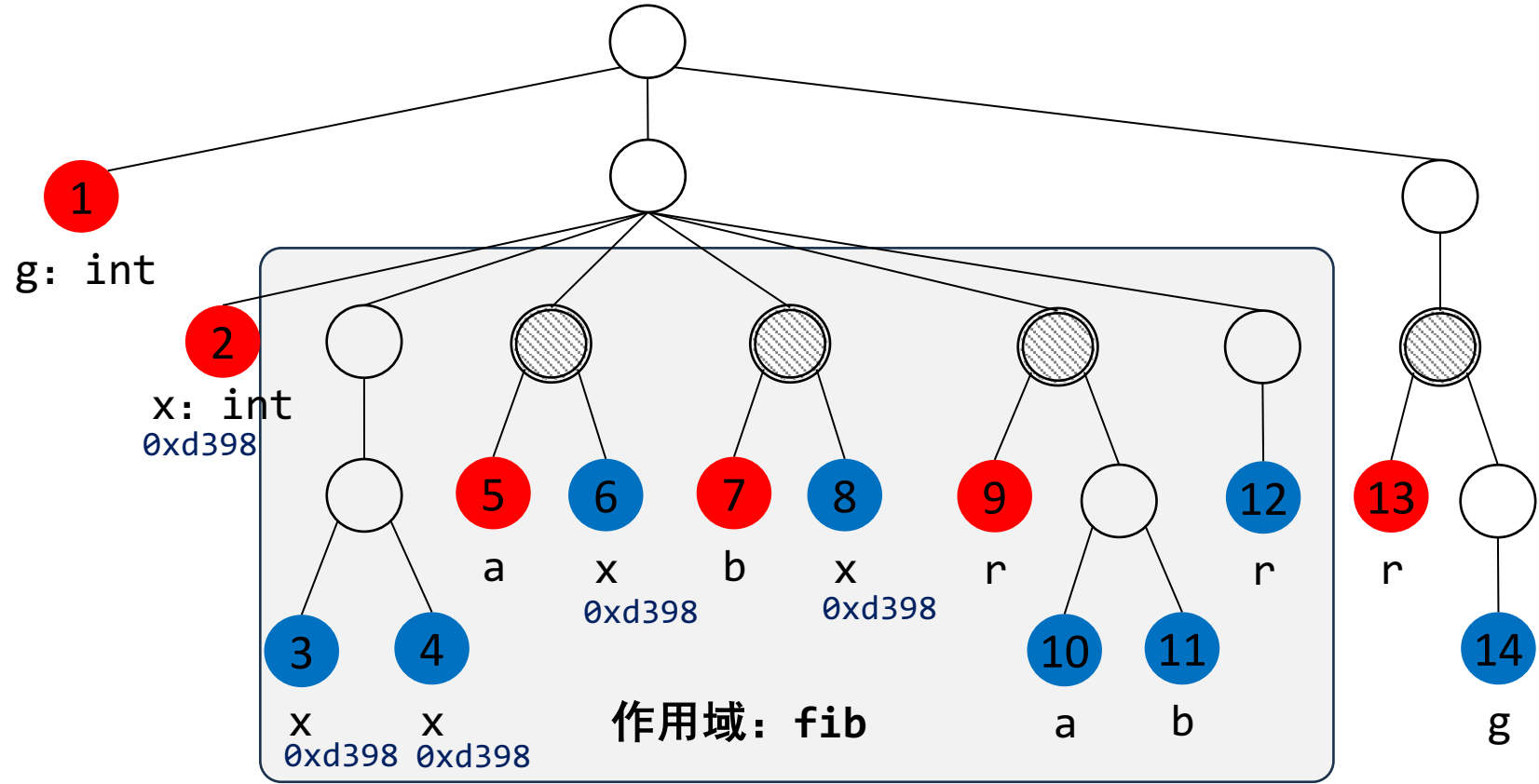
局部变量类型分析：x



标识符	作用域	索引	类型
x	fib	0xd398	int
a			
b			
r			

- 一般节点
- 变量声明
- 变量引用
- ⊘ 声明引用

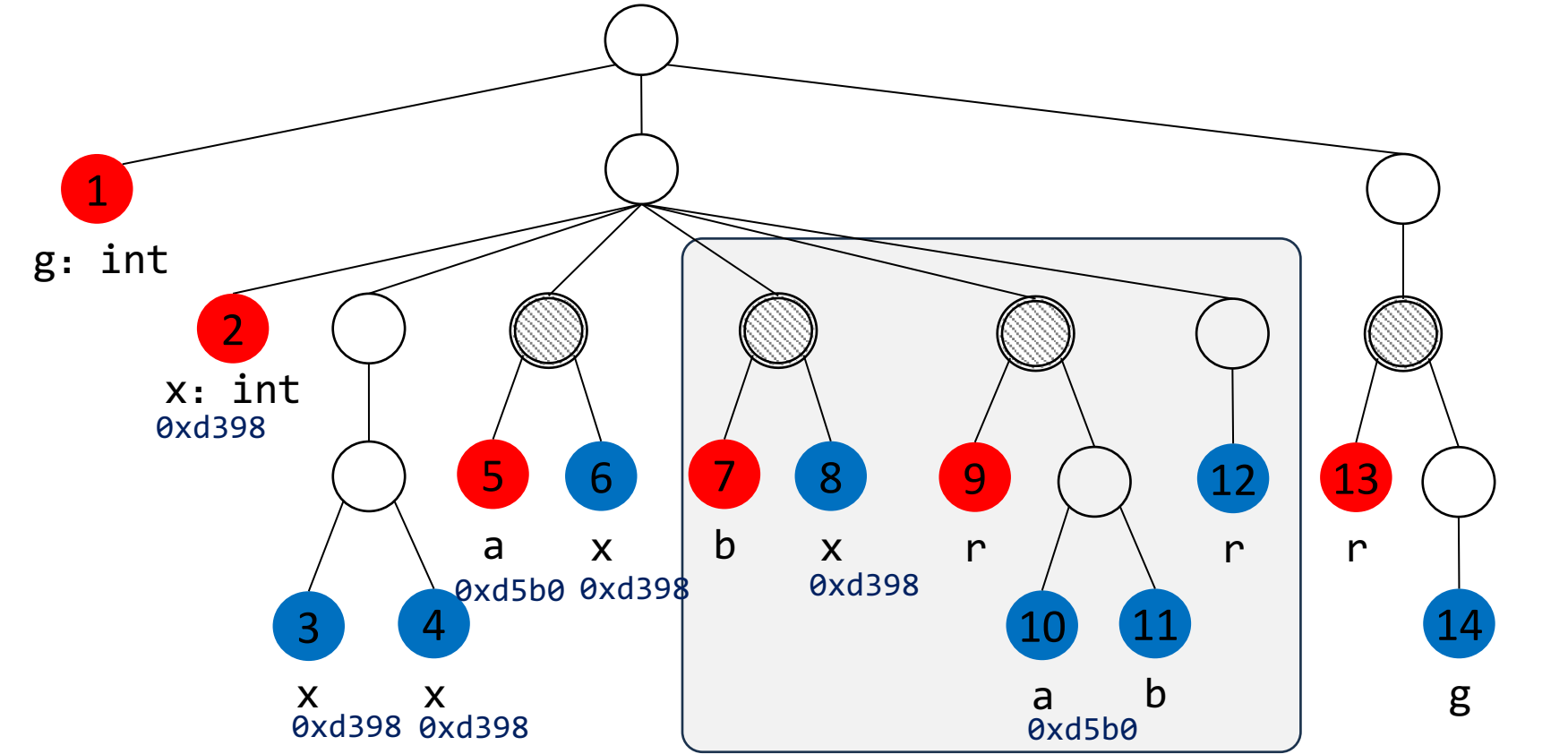
标识符索引化: x



标识符	作用域	索引	类型
x	fib	0xd398	int
a			
b			
r			

- 一般节点
- 变量声明
- 变量引用
- ⊘ 声明引用

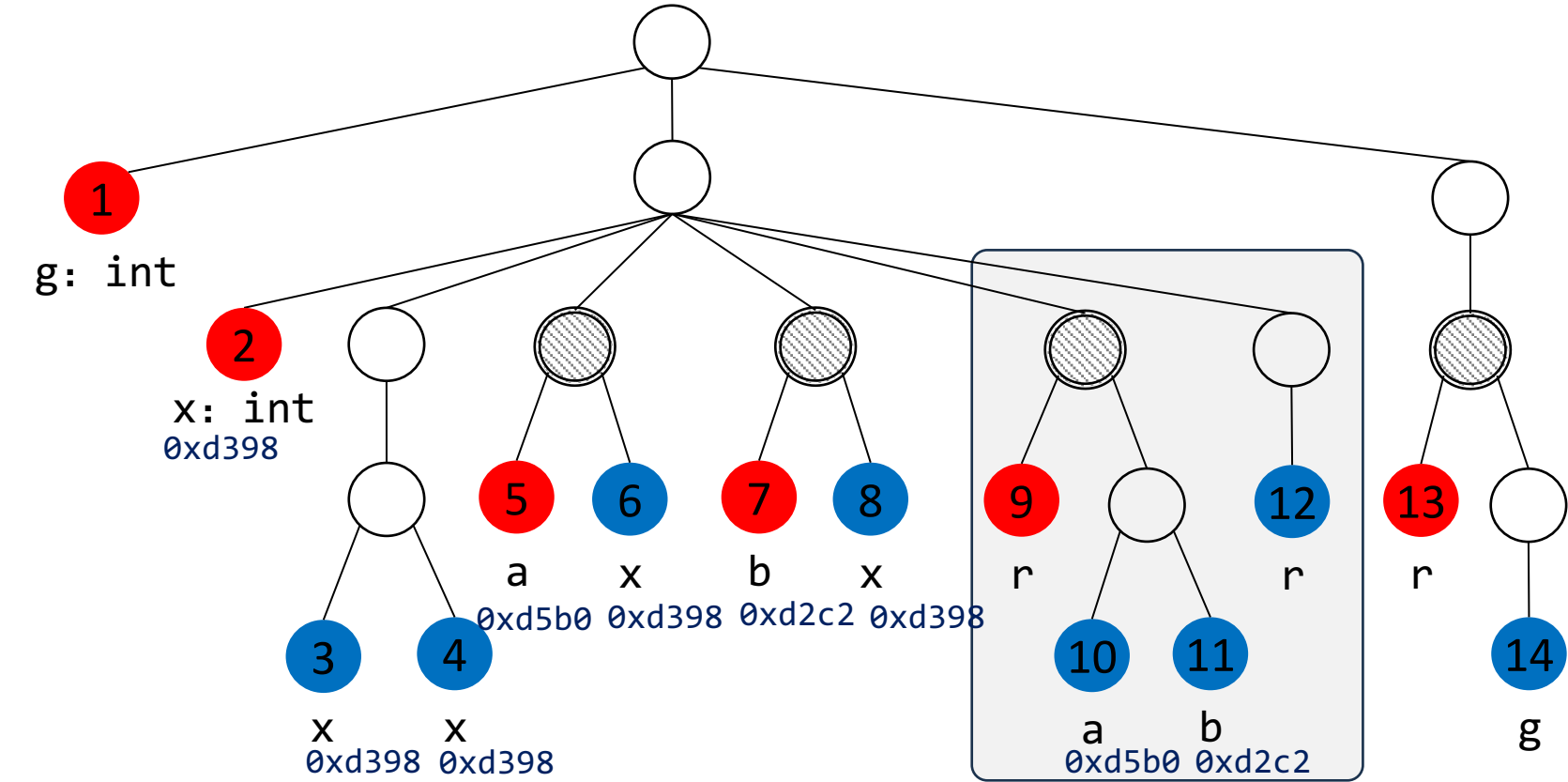
局部变量类型分析/标识符索引化：a



标识符	作用域	索引	类型
x	fib	0xd398	int
a	fib:scope1	0xd5b0	
b			
r			

- 一般节点
- 变量声明
- 变量引用
- ⊘ 声明引用

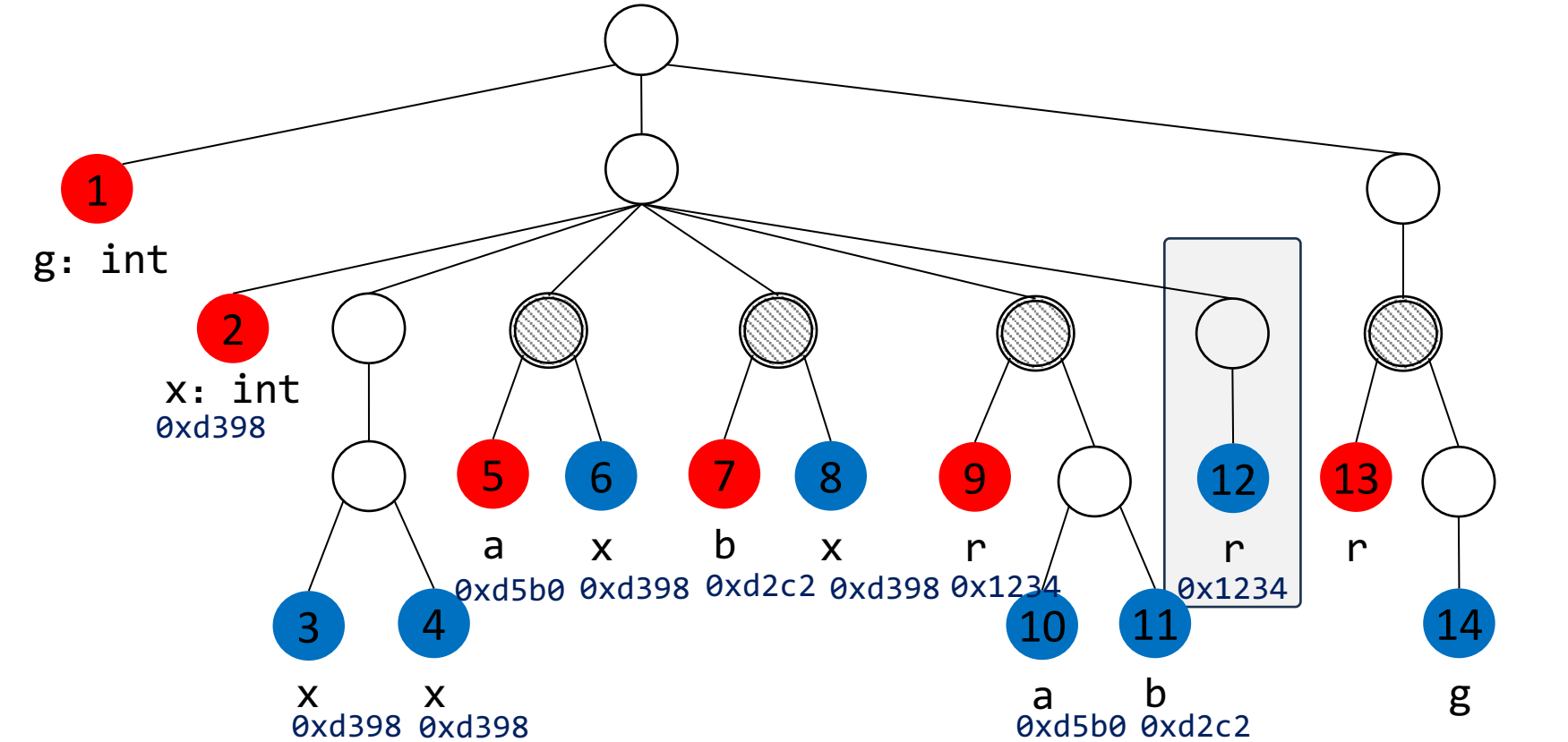
局部变量类型分析/标识符索引化: b



标识符	作用域	索引	类型
x	fib	0xd398	int
a	fib:scope1	0xd5b0	
b	fib:scope2	0xd2c2	
r			

- 一般节点
- 变量声明
- 变量引用
- ⊘ 声明引用

局部变量类型分析/标识符索引化: r



标识符	作用域	索引	类型
x	fib	0xd398	int
a	fib:scope1	0xd5b0	
b	fib:scope2	0xd2c2	
r	fib:scope3	0x1234	

- 一般节点
- 变量声明
- 变量引用
- ⊘ 声明引用

算法实现思路

- 动态维护两个哈希表记录符号作用域和类型
 - 全局符号表
 - 局部符号表
- 遍历抽象语法树对符号进行索引化
 - 符号声明时加入符号表
 - 跳出作用域将符号移除

作用域导致的类型错误举例

```
fn foo(n: int) -> int {  
  while (n>0) {  
    ...  
    n = n-1;  
  }  
  ret x;  
}
```

错误：变量x未声明

当前使用变量不在符号表中

```
fn foo(n: int) -> int {  
  while (n>0) {  
    let x:int;  
    ...  
    n = n-1;  
  }  
  ret x;  
}
```

声明变量x，但作用域太小

错误：变量x未声明

作用域导致的类型检查问题

```
fn foo(n: int) -> int {  
  let x:int;  
  if (n>0) {  
    let x:int;  
    ...  
    x = n-1;  
  }  
  ret x;  
}
```

声明变量x

错误：重复声明

当前声明变量名已经在局部变量表中


```
fn foo(n: int) -> int {  
  if (n>0) {  
    let x:int;  
    ...  
  }  
  else {  
    let x:int;  
    ...  
  }  
}
```

声明变量x


声明变量x

TeaPL中的全局变量使用要求

```
fn foo(n: int) -> int {  
    x = x + n;  
    ret x;  
}  
let x:int = 0;
```




```
let x:int = a + 5;  
let a:int = 5;
```




全局变量声明和在全局变量中的使用顺序有关

全局变量声明和在函数中的使用顺序无关

```
let a:int = 203;  
let b:int = 713;  
  
fn foo(a:int) -> int {  
    let b:int = 10;  
    ret a + b;  
}
```



```
let a:MyStruct;  
fn foo(a:int) -> int{  
    ret 2 * a;  
}
```




实验作业中的例外

局部变量不能和全局变量重名

TeaPL中的函数重名问题：是否允许重载/多态？

```
fn foo(x:int);  
fn foo() -> int;
```



TeaPL暂时不允许重载

支持方法：在全局符号表中增添项目即可

标识符	作用域	索引	类型
foo	global	0xadc2	(int) → void
foo	global	0xbc70	(void) → int

三、类型约束和求解

TeaPL的类型系统

- 类型：
 - 基础类型（Primitive Type）
 - 标量类型（Scalar Types）：int、bool
 - 复合类型（Compound Type）：数组
 - 函数类型
 - 自定义类型：使用struct定义
- 规则（静态类型系统）
 - 类型推导：为代码中的每个标识符和表达式确定类型
 - 类型检查：分析每个参数类型是否符合运算符或函数签名要求
 - 类型转换：允许隐式类型转换？（类型强弱）

类型错误举例

```
fn foo(n: int) -> int {  
    ...  
}  
fn main() {  
    foo(foo);  
}
```

参数类型错误

```
fn foo(n: int) -> int {  
    if (n <= 1) {  
        ret n;  
    }  
    let x:int = foo(n - 1);  
}
```

缺少返回语句

类型推导/检查思路

Damas–Hindley–Milner方法

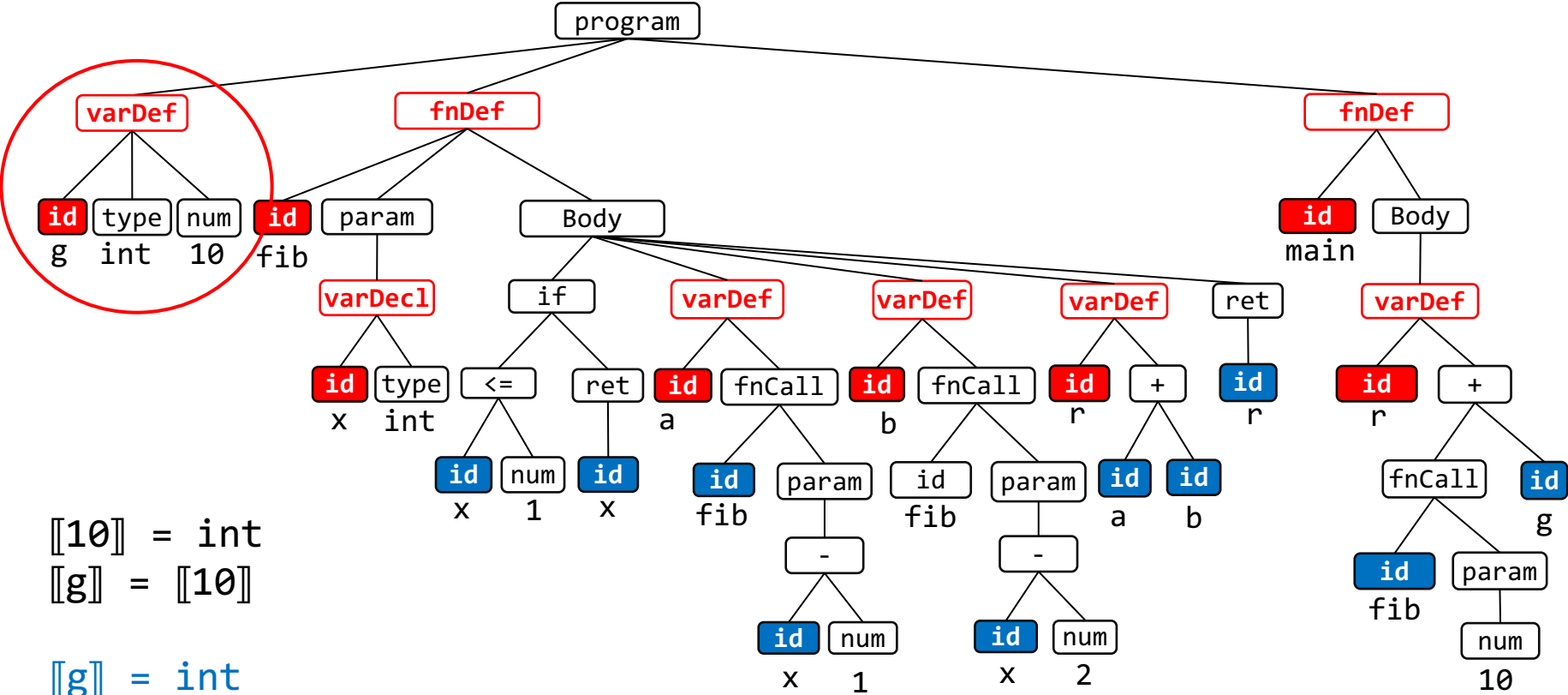
- 1) 根据符号表确定变量类型
- 2) 基于类型规则提取类型约束
 - 类型表示：用 $\llbracket X \rrbracket$ 表示变量 x 的类型
 - 约束提取：一般都为等价关系；子类型和范型除外
- 3) 约束求解

类型规则设计

- 为不同的语法制定相应的推断规则

代码示例	代码模式	约束
\emptyset	N	$\llbracket N \rrbracket = \text{int}$
$a = \emptyset;$	$X = N$	$\llbracket N \rrbracket = \text{int}, \llbracket X \rrbracket = \llbracket N \rrbracket$
$a = b;$	$X = Y$	$\llbracket X \rrbracket = \llbracket Y \rrbracket$
$a + b;$	$X \text{ bop } Y$	$\llbracket X \rrbracket = \llbracket Y \rrbracket = \llbracket X + Y \rrbracket$
$c = a + b;$	$T = X \text{ bop } Y$ $Z = T$	$\llbracket X \rrbracket = \llbracket Y \rrbracket = \llbracket X + Y \rrbracket = \llbracket T \rrbracket,$ $\llbracket Z \rrbracket = \llbracket T \rrbracket$

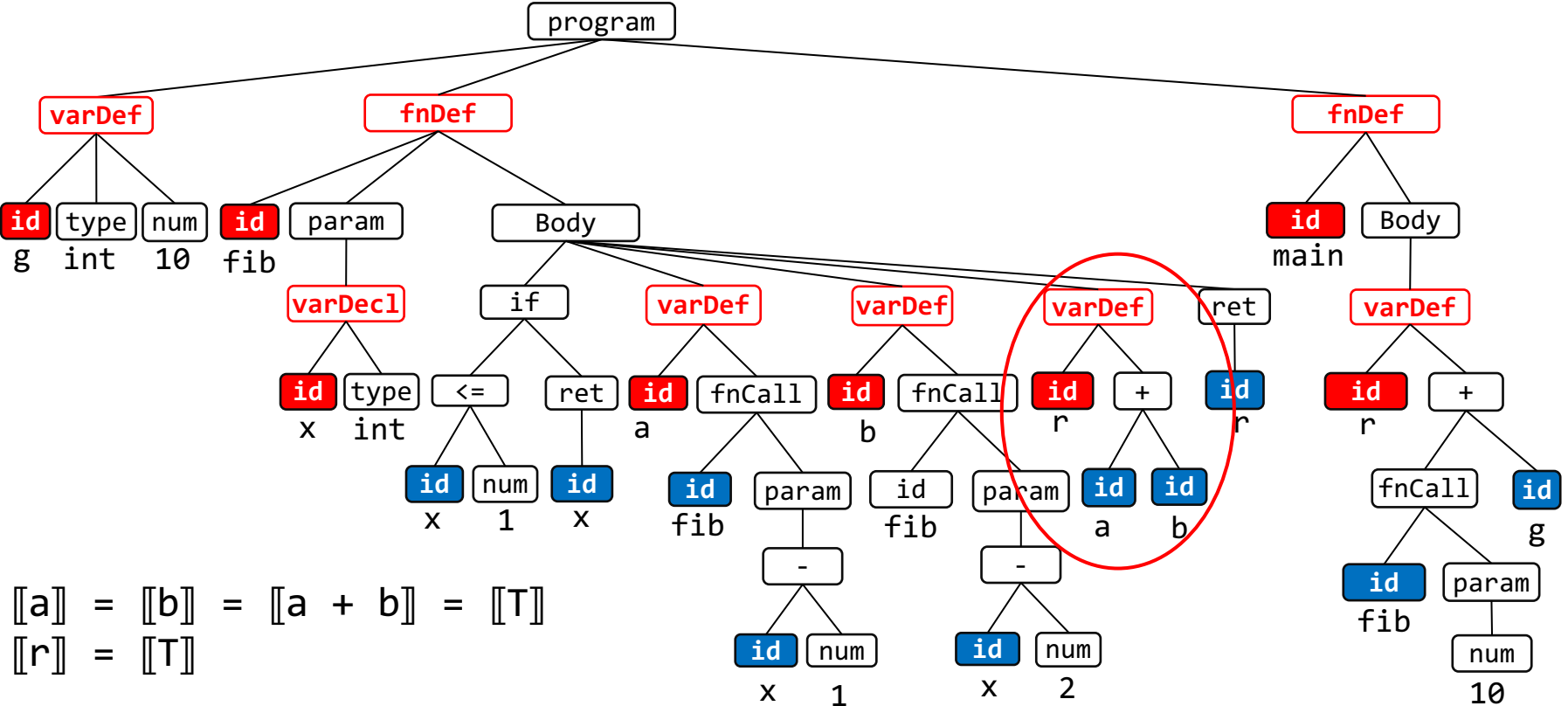
示例：约束提取



标识符	作用域	索引	类型
g	global	0xd9c2	int
fib	global	0xd470	(int) → int
main	global	0xd318	(void) → void

● 变量声明
● 变量引用

示例：约束提取



标识符	作用域	索引	类型
x	fib	0xd398	int
a	fib:scope1	0xd5b0	
b	fib:scope2	0xd2c2	
r	fib:scope3	0x1234	

● 变量声明
● 变量引用

更多类型规则

代码示例

代码模式

约束

```
if(a)
```

```
if(X)
```

 $\llbracket X \rrbracket = \text{bool}$

```
a > b
```

```
X rop Y
```

 $\llbracket X \rrbracket = \llbracket Y \rrbracket, \llbracket X \text{ rop } Y \rrbracket = \text{bool}$

```
if(a > b){...}
```

```
T = X rop Y  
if(T)
```

 $\llbracket X \rrbracket = \llbracket Y \rrbracket, \llbracket X \text{ rop } Y \rrbracket = \text{bool} = \llbracket T \rrbracket, \llbracket T \rrbracket = \text{bool}$

```
a && b
```

```
X lop Y
```

 $\llbracket X \rrbracket = \llbracket Y \rrbracket = \llbracket X \text{ lop } Y \rrbracket = \text{bool}$

```
a > b && c
```

```
T = X rop Y  
T lop Z
```

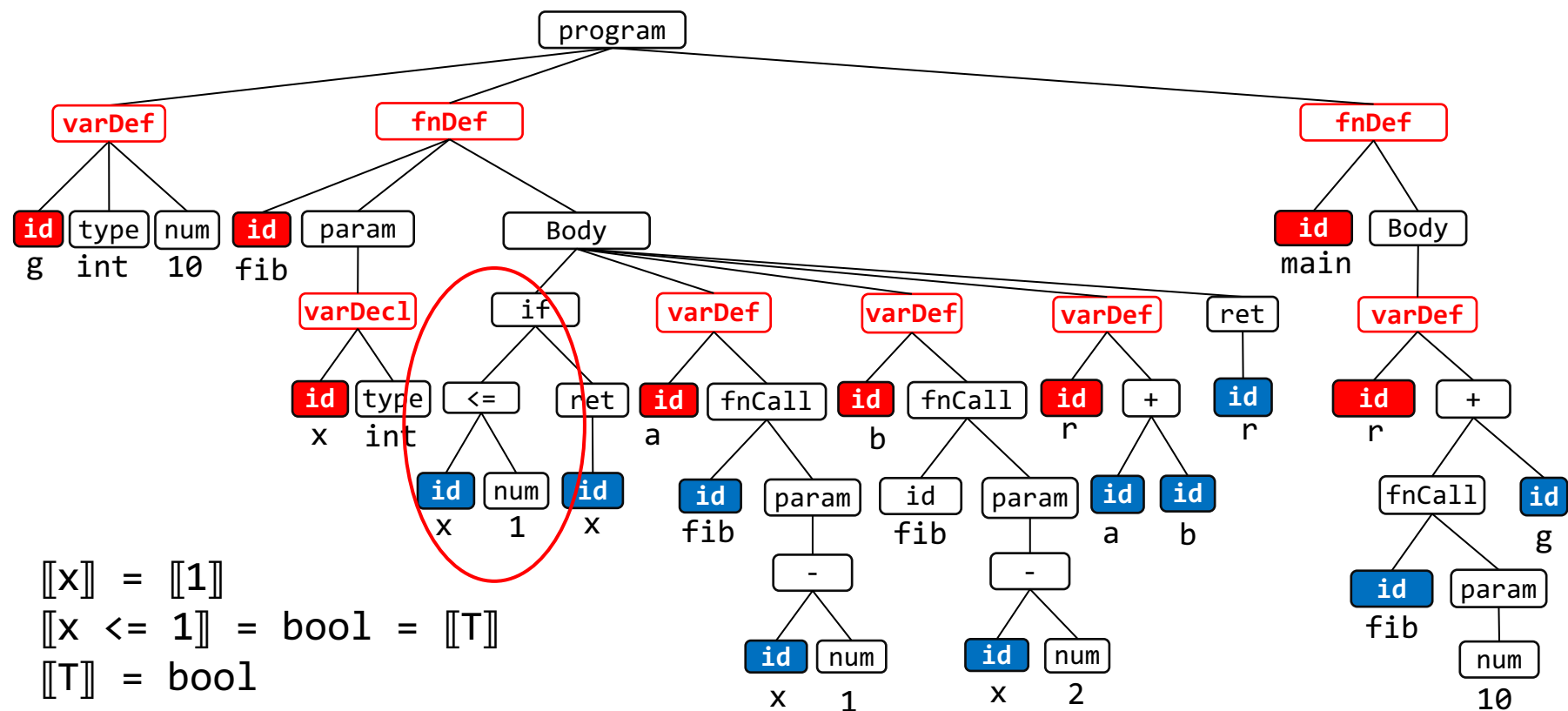
 $\llbracket X \rrbracket = \llbracket Y \rrbracket = \llbracket X \text{ lop } Y \rrbracket = \text{bool} = \llbracket T \rrbracket, \llbracket T \rrbracket = \llbracket Z \rrbracket = \llbracket T \text{ lop } Z \rrbracket = \text{bool}$

```
while(a > b){...}
```

```
T = X rop Y  
while(T)
```

...

示例：约束提取



$\llbracket x \rrbracket = \llbracket 1 \rrbracket$
 $\llbracket x \leq 1 \rrbracket = \text{bool} = \llbracket T \rrbracket$
 $\llbracket T \rrbracket = \text{bool}$
 $\llbracket x \rrbracket = \text{int}$

标识符	作用域	索引	类型
x	fib	0xd398	int
a	fib:scope1	0xd5b0	
b	fib:scope2	0xd2c2	
r	fib:scope3	0x1234	

● 变量声明
● 变量引用

更多类型规则

代码示例

```
foo(a, b);
```

代码模式

$F(X, Y)$

约束

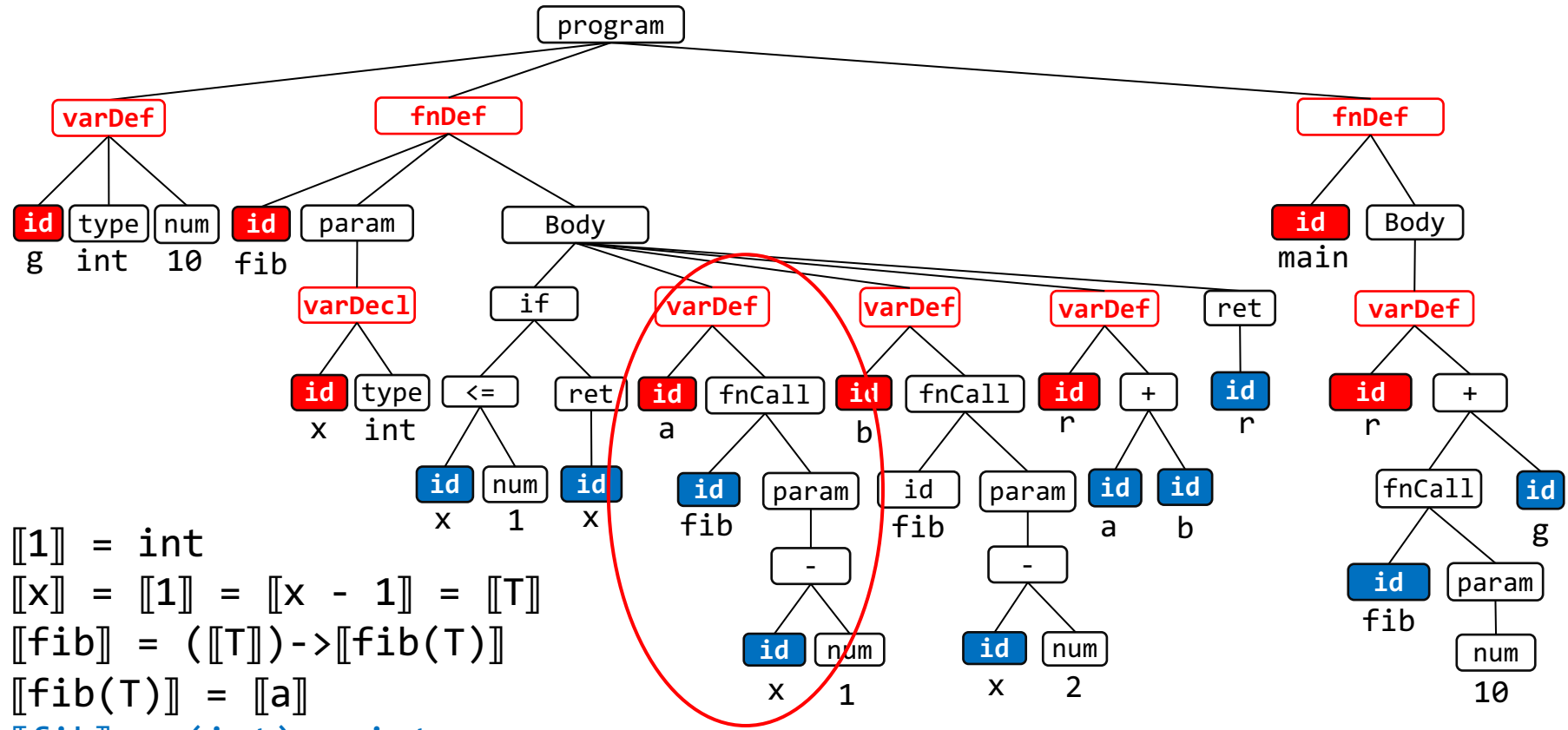
$\llbracket F \rrbracket = (\llbracket X \rrbracket, \llbracket Y \rrbracket) \rightarrow \llbracket F(X, Y) \rrbracket$

```
ret a;
```

$F(X) \rightarrow Y \{$
 \dots
 $\text{ret } Z;$
 $\}$

$\llbracket F \rrbracket = (\llbracket X \rrbracket) \rightarrow \llbracket Z \rrbracket, \llbracket Z \rrbracket = \llbracket Y \rrbracket$

示例：约束提取

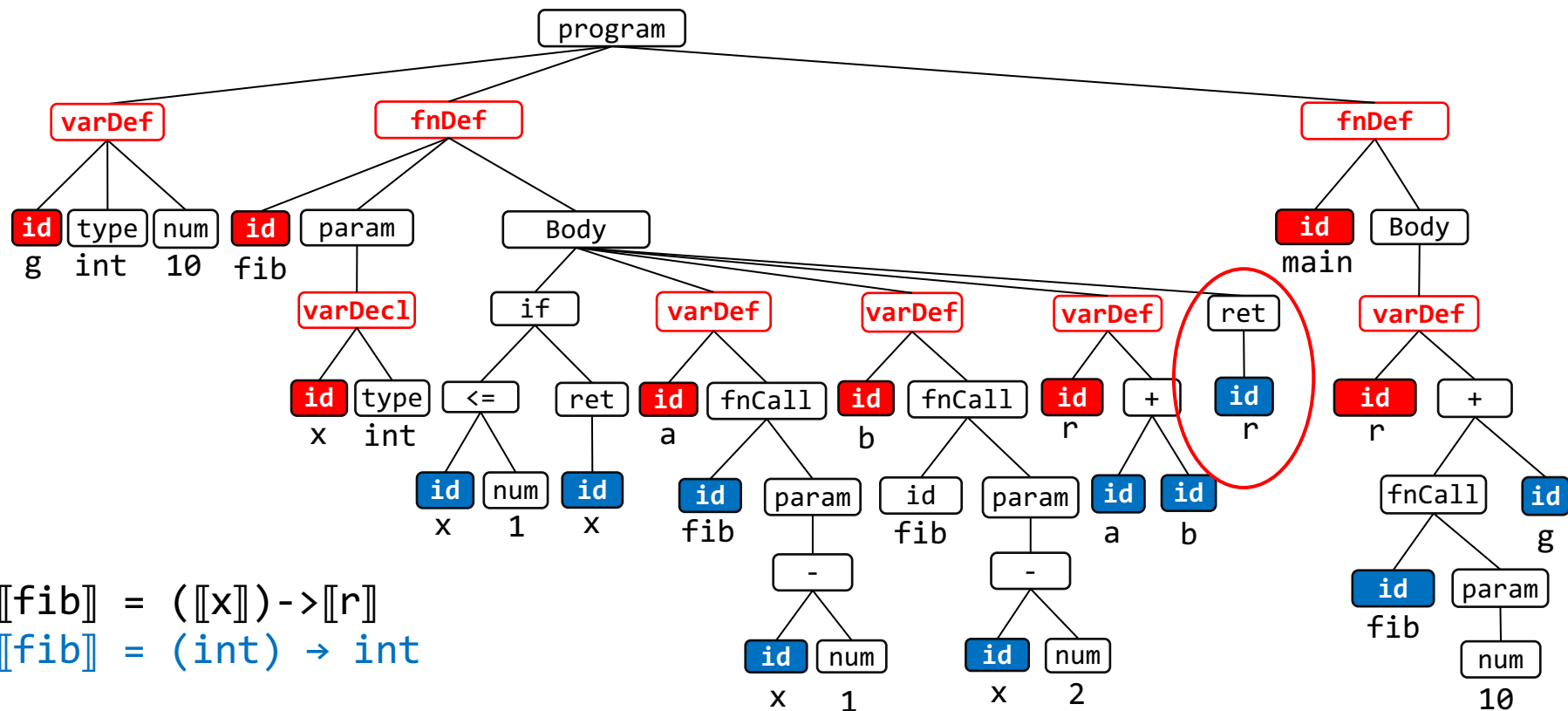


$\llbracket 1 \rrbracket = \text{int}$
 $\llbracket x \rrbracket = \llbracket 1 \rrbracket = \llbracket x - 1 \rrbracket = \llbracket T \rrbracket$
 $\llbracket \text{fib} \rrbracket = (\llbracket T \rrbracket) \rightarrow \llbracket \text{fib}(T) \rrbracket$
 $\llbracket \text{fib}(T) \rrbracket = \llbracket a \rrbracket$
 $\llbracket \text{fib} \rrbracket = (\text{int}) \rightarrow \text{int}$

标识符	作用域	索引	类型
g	global	0xd9c2	int
fib	global	0xd470	(int) → int
main	global	0xd318	(void) → void

● 变量声明
● 变量引用

示例：约束提取



```
[[fib]] = ([[x]])->[[r]]
[[fib]] = (int) → int
```

标识符	作用域	索引	类型
g	global	0xd9c2	int
fib	global	0xd470	(int) → int
main	global	0xd318	(void) → void

- 变量声明
- 变量引用

约束提取结果

$\llbracket 10 \rrbracket = \text{int}$

$\llbracket g \rrbracket = \llbracket 10 \rrbracket$

$\llbracket g \rrbracket = \text{int}$

$\llbracket a \rrbracket = \llbracket b \rrbracket = \llbracket a + b \rrbracket = \llbracket T1 \rrbracket$

$\llbracket r \rrbracket = \llbracket T1 \rrbracket$

$\llbracket x \rrbracket = \llbracket 1 \rrbracket$

$\llbracket x \leq 1 \rrbracket = \text{bool} = \llbracket T2 \rrbracket$

$\llbracket T2 \rrbracket = \text{bool}$

$\llbracket x \rrbracket = \text{int}$

$\llbracket \text{fib} \rrbracket = (\llbracket x \rrbracket) \rightarrow \llbracket r \rrbracket$

$\llbracket \text{fib} \rrbracket = (\text{int}) \rightarrow \text{int}$

$\llbracket 1 \rrbracket = \text{int}$

$\llbracket x \rrbracket = \llbracket 1 \rrbracket = \llbracket x - 1 \rrbracket = \llbracket T3 \rrbracket$

$\llbracket \text{fib} \rrbracket = (\llbracket T3 \rrbracket) \rightarrow \llbracket \text{fib}(T3) \rrbracket$

$\llbracket \text{fib}(T3) \rrbracket = \llbracket a \rrbracket$

$\llbracket \text{fib} \rrbracket = (\text{int}) \rightarrow \text{int}$

$\llbracket 1 \rrbracket = \text{int}$

$\llbracket x \rrbracket = \llbracket 1 \rrbracket = \llbracket x - 2 \rrbracket = \llbracket T4 \rrbracket$

$\llbracket \text{fib} \rrbracket = (\llbracket T4 \rrbracket) \rightarrow \llbracket \text{fib}(T4) \rrbracket$

$\llbracket \text{fib}(T4) \rrbracket = \llbracket b \rrbracket$

$\llbracket \text{fib} \rrbracket = (\text{int}) \rightarrow \text{int}$

基于并查集方法求解

- 维护不存在相交关系的集合，支持查找和联合两种操作
 - Find(x): 返回包含变量x的集合
 - Union(x, y): 联合包含x和y的两个集合

```
while(getPair()!=NULL){  
    [p,q] = readPair(p,q);  
    pset = find(p);  
    qset = find(q);  
    if(pset == qset)  
        continue;  
    else union(p,q);  
}
```


应用并查集方法求解

$\llbracket 10 \rrbracket = \text{int}$ $\llbracket g \rrbracket = \llbracket 10 \rrbracket$ $\llbracket g \rrbracket = \text{int}$	$S1: \{\llbracket 10 \rrbracket, \text{int}, \llbracket g \rrbracket\}$
$\llbracket a \rrbracket = \llbracket b \rrbracket = \llbracket a + b \rrbracket = \llbracket T1 \rrbracket$ $\llbracket r \rrbracket = \llbracket T1 \rrbracket$	$S2: \{\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket r \rrbracket, \llbracket T1 \rrbracket\}$
$\llbracket x \rrbracket = \llbracket 1 \rrbracket$ $\llbracket x \leq 1 \rrbracket = \text{bool} = \llbracket T2 \rrbracket$ $\llbracket T2 \rrbracket = \text{bool}$ $\llbracket x \rrbracket = \text{int}$	$S1: \{\llbracket x \rrbracket, \llbracket 1 \rrbracket, \text{int}, \llbracket 10 \rrbracket, \llbracket g \rrbracket\}$ $S3: \{\llbracket x \leq 1 \rrbracket, \llbracket T2 \rrbracket, \text{bool}\}$
$\llbracket 1 \rrbracket = \text{int}$ $\llbracket x \rrbracket = \llbracket 1 \rrbracket = \llbracket x - 1 \rrbracket = \llbracket T3 \rrbracket$ $\llbracket \text{fib} \rrbracket = (\llbracket T3 \rrbracket) \rightarrow \llbracket \text{fib}(T3) \rrbracket$ $\llbracket \text{fib}(T3) \rrbracket = \llbracket a \rrbracket$ $\llbracket \text{fib} \rrbracket = (\text{int}) \rightarrow \text{int}$	$S1: \{\llbracket x \rrbracket, \llbracket 1 \rrbracket, \text{int}, \llbracket x - 1 \rrbracket, \llbracket T3 \rrbracket, \llbracket 10 \rrbracket, \llbracket g \rrbracket\}$ $S4: \{\llbracket \text{fib} \rrbracket, (\llbracket T3 \rrbracket) \rightarrow \llbracket \text{fib}(T3) \rrbracket, (\text{int}) \rightarrow \text{int}\}$ $S2: \{\llbracket \text{fib}(T3) \rrbracket, \llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket r \rrbracket, \llbracket T1 \rrbracket\}$
$\llbracket \text{fib} \rrbracket = (\llbracket x \rrbracket) \rightarrow \llbracket r \rrbracket$ $\llbracket \text{fib} \rrbracket = (\text{int}) \rightarrow \text{int}$	$S4: \{\llbracket \text{fib} \rrbracket, (\llbracket T3 \rrbracket) \rightarrow \llbracket \text{fib}(T3) \rrbracket, (\text{int}) \rightarrow \text{int}, (\llbracket x \rrbracket) \rightarrow \llbracket r \rrbracket\}$
...	...

最终解

S1: {[x], [1], int, [x - 1], [T3], [10], [g]}

S2: {[fib(T3)], [a], [b], [r], [T1]}

S3: {[x <= 1], [T2], bool}

S4: {[fib], ([T3]) -> [fib(T3)], (int) → int, ([x]) -> [r]}



[g] = int

[a] = int

[b] = int

[r] = int

[x <= 1] = [T2] = bool

[fib] = (int) → int

[T3] = int

[fib(T3)] = int

更多类型规则：数组

代码示例

代码模式

约束

```
{0,1}
```

```
{M,N}
```

```
{M,N} = &int
```

```
a = {0,1};
```

```
X = {M,N}
```

```
{M,N} = &int =  $\llbracket T \rrbracket$ ,  $\llbracket X \rrbracket = \llbracket T \rrbracket$ 
```

```
b = a[i];
```

```
Y = X[Z]
```

```
 $\llbracket Z \rrbracket = \text{int}$ ,  $\llbracket Y \rrbracket = \llbracket *X \rrbracket$ ,  $\llbracket X \rrbracket = \&\llbracket *X \rrbracket$ 
```

```
a[i] = b;
```

```
X[Z] = Y
```

```
 $\llbracket Z \rrbracket = \text{int}$ ,  $\llbracket X \rrbracket = \&\llbracket Y \rrbracket$ 
```

更多类型规则：结构体

代码示例

```
struct Foo {  
    a:int,  
    b:int,  
}
```

```
foo.a = d;
```

代码模式

```
struct ST {  
    A:int,  
    B:int,  
}
```

```
X.A = Y
```

约束

$$\llbracket ST \rrbracket = \llbracket \text{int}, \text{int} \rrbracket$$
$$\llbracket X.A \rrbracket = \llbracket Y \rrbracket, \llbracket X.A, _ \rrbracket = \llbracket X \rrbracket$$

类型推断可能存在的问题


- 解不唯一的问题：优选选择的类型
- 无解的情况：是否允许隐式类型转换？
- 如何判断类型是否等价：名字相同 vs 结构相同

```
struct Pos { x:int, y:int, }  
struct Loc { x:int, y:int, }
```

递归问题


//TeaPL 代码

```
fn factorial(n:int) -> int {  
  if (n == 0) {  
    ret 1;  
  } else {  
    let r = n * factorial(n-1);  
    ret r;  
  }  
}
```



//TeaPL 代码


```
struct List {  
  v:int,  
  next List,  
}
```



$\llbracket \text{List} \rrbracket = \phi = (\text{int}, \phi)$

//C 代码

```
struct List{  
  int data;  
  struct List* next;  
};
```



$\llbracket \text{List} \rrbracket = \phi = (\text{int}, \&\phi)$

练习：类型检查

- 应用类型检查方法分析实验中的测试用例
- 链接： https://github.com/hxuhack/compiler_project/blob/24s-assignment2/src/tests

练习

- 假如TeaPL函数声明可缺省类型，设计方法分析下列代码中的类型信息

```
fn factorial(n) {  
    if (n == 0) {  
        ret 1;  
    } else {  
        let r = n * factorial(n-1);  
        ret r;  
    }  
}
```

```
fn factorial(f, n) {  
    if (n == 0) {  
        ret 1;  
    } else {  
        let r = n * f(f, n-1);  
        ret r;  
    }  
}
```