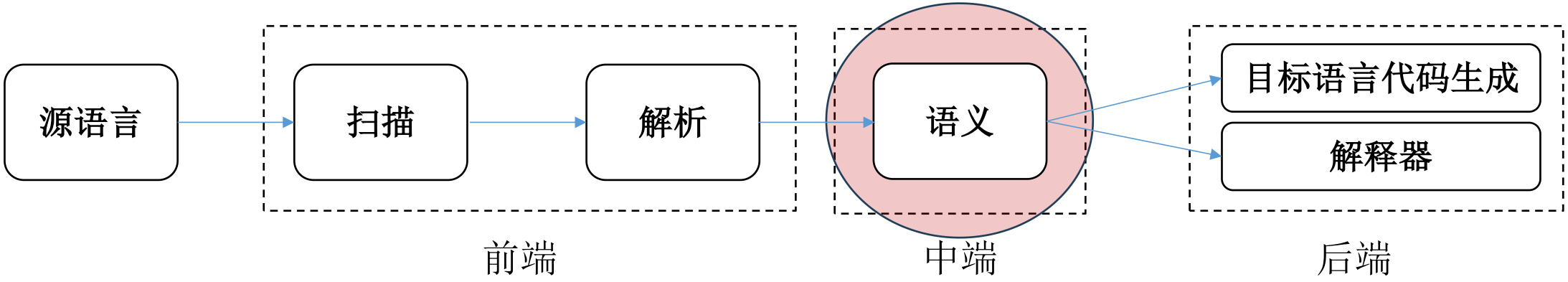
中间语言IR

杨广亮

2024年10月

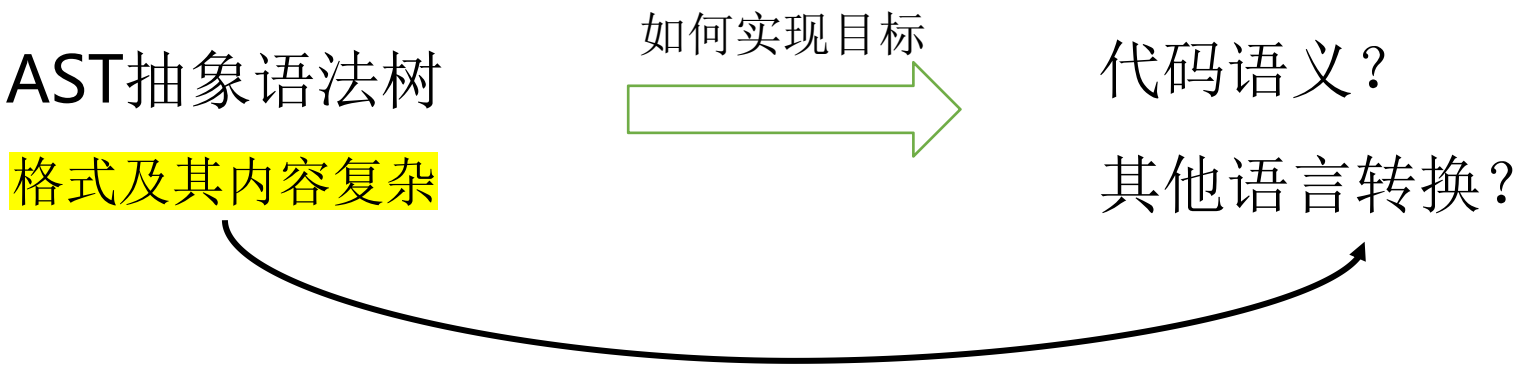
CODE SECURITY
RESEARCH



中端

- 抽象语法树
- 基本类型检查
- 中间语言
- SSA静态单一赋值
- 控制流
- 数据流
- 优化

中间语言



中间语言: Intermediate Representation/ IR

中间语言

- 什么是中间语言?
  - 由指令和地址做成，变量名、常量、编译器生成的临时变量或存储单元
  - 对代码的中间表示
  - 起到一个承上启下的作用：上-源代码AST，下-
- 中间语言规范代码内容
  - 三地址
    - 只有三个操作数 left = right1 <op> right2

3-address Instruction Format

中间语言

- 知名中间表示
  - llvm
  - Web assembly
  - Dalvik bytecode
  - MLIR
- 自研语言GLang

中间语言LLVM

LLVM

- 诞生于UIUC 2000-2003年
- 事实标准中间语言
  - 支持大量语言ActionScript, Ada, C# for .NET, Common Lisp, PicoLisp, Crystal, CUDA, D, Delphi, Dylan, Forth, Fortran, FreeBASIC, Free Pascal, Halide, Haskell, Java bytecode, Julia, Kotlin, Lua, Objective-C, OpenCL, PostgreSQL's SQL and PLpgSQL, Ruby, Rust, Scala, Swift, Xojo, and Zig.

Vikram Adve

Chris Lattner

LLVM编写者
Swift之父
Google ML Compiler
Mojo for AI

CODE SECURITY RESEARCH

面向底层逻辑的中间语言典范LLVM

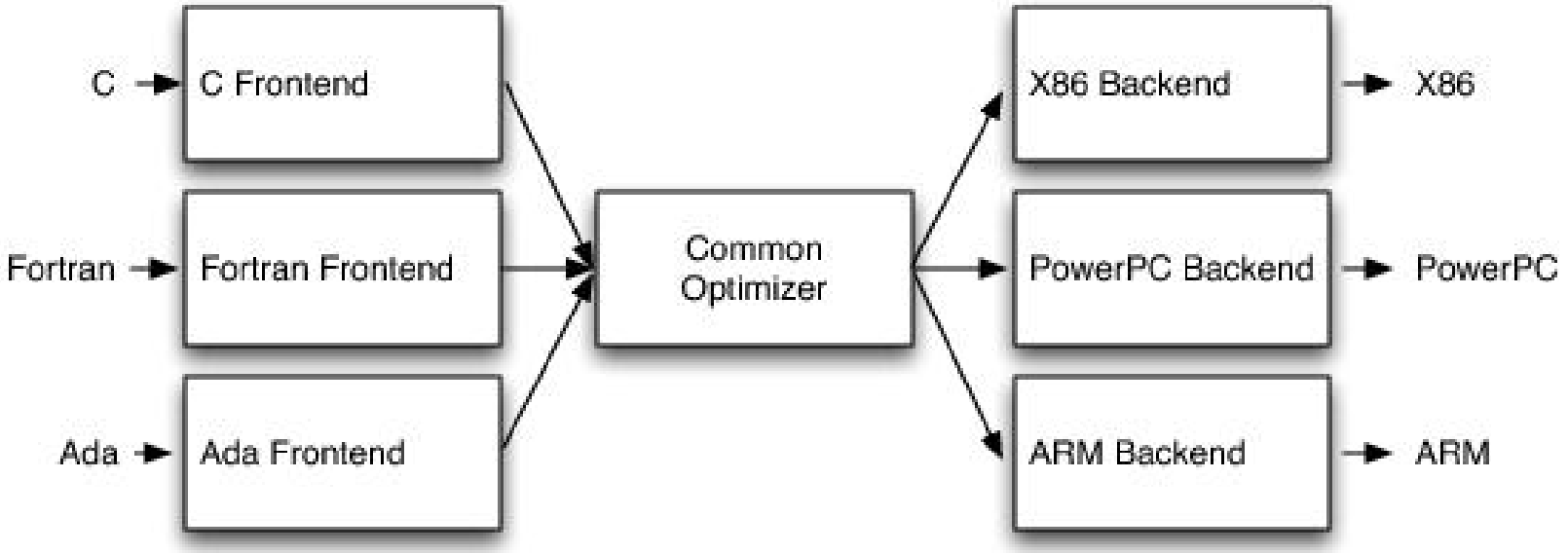
LLVM

LLVM is a set of compiler and toolchain technologies that can be used to develop a frontend for any programming language and a backend for any instruction set architecture.

- wikipedia

中间语言

- LLVM架构



llvm.org

例子


```
unsigned add1(unsigned a, unsigned b) {
  return a+b;
}

// Perhaps not the most efficient way to add two numbers.
unsigned add2(unsigned a, unsigned b) {
  if (a == 0) return b;
  return add2(a-1, b+1);
}
```

CODE SECURITY
RESEARCH

例子

```
define i32 @add1(i32 %a, i32 %b) {
entry:
  %tmp1 = add i32 %a, %b
  ret i32 %tmp1
}

define i32 @add2(i32 %a, i32 %b) {
entry:
  %tmp1 = icmp eq i32 %a, 0
  br i1 %tmp1, label %done, label %recurse

recurse:
  %tmp2 = sub i32 %a, 1
  %tmp3 = add i32 %b, 1
  %tmp4 = call i32 @add2(i32 %tmp2, i32 %tmp3)
  ret i32 %tmp4

done:
  ret i32 %b
}
```

符号系统：
- % 临时符号
  如%1，%2
- @全局符号
  如@add1

IR定义：标识符、基础类型、和数据存取

- 全局变量/函数名称：@name
- 局部变量/临时变量：%x、%0（不可重复定义，纯数字编号需连续）
- 类型：void、i32、i32*、i8、i8*、i1
- 栈空间分配：alloca
- 数据存取：load/store

```
@g = global i32 10

define i32 @fib(i32 %0) {
    %x = alloca i32
    store i32 %0, %x
    %g0 = load i32, i32* @g
    ret i32 %g0
}

define i32 @main() {
    %r0 = call i32 @fib(i32 1)
    ret i32 %r0;
}
```

声明全局变量g，类型*i32，初始值10

声明函数fib，参数名为%0，类型i32

声明局部变量%x，类型为i32*

声明临时变量%g0，类型为i32

IR定义：函数

- 声明：define
- 调用：call
- 返回：ret

```
@g = global i32 10

define i32 @fib(i32 %0) {
  %x = alloca i32
  store i32 %0, %x
  %g0 = load i32, i32* @g
  ret i32 %g0
}

define i32 @main() {
  %r0 = call i32 @fib(i32 1)
  ret i32 %r0;
}
```

声明函数fib，参数类型i32

声明函数main

调用函数fib

IR定义：复合类型数据定义和存取

```
define i32 @main() {
  %1 = alloca [2 x i32]
  %2 = getelementptr [2 x i32], [2 x i32]* %1, i32 0, i32 0
  store i32 99, i32* %2
  %3 = load i32, i32* %2
  ret i32 %3
}
```

创建一维数组，并返回数组指针

索引为0的元素

获取元素地址

```
%mystruct = type { i32, i32 }
define i32 @main() {
  %1 = alloca %mystruct
  %2 = getelementptr %mystruct, %mystruct* %1, i32 0, i32 0
  store i32 1, i32* %2
  ret i32 0
}
```

定义mystruct数据类型

创建mystruct对象，并返回指针

IR定义：算数运算

```
%2 = alloca i32
%3 = add i32 %0, 1
%4 = sub i32 %3, 2
%5 = mul i32 %3, 3
%6 = sdiv i32 %4, 4
store i32 %6, i32* %2
%7 = load i32, i32* %2
ret i32 %6
```

浮点数运算用
fadd/fsub/fmul/fdiv

IR定义：比较运算和类型转换

```
%2 = alloca i32
%3 = alloca i8
store i32 %0, i32* %2
%4 = load i32, i32* %2
%5 = icmp sgt i32 %4, 0
%6 = icmp sge i32 %4, 0
%7 = icmp slt i32 %4, 0
%8 = icmp sle i32 %4, 0
%9 = icmp eq i32 %4, 0
%10 = icmp ne i32 %4, 0
%11 = zext i1 %10 to i8
store i8 %11, i8* %3
%12 = load i8, i8* %3
%13 = trunc i8 %12 to i1
```

s: signed
g: greater
l: less
e: equal
n: not

类型转换：zero extend

类型转换：truncate

IR定义：跳转语句

- 基于br实现if-else和while等控制流功能

```
  %2 = alloca i32
  store i32 0, i32* %2
  %3 = load i32, i32* %2
  %4 = icmp sgt i32 %3, 0
  br i1 %4, label %bb1, label %bb2
bb1:
  store i32 1, i32* %2
  br label %bb3
bb2:
  store i32 0, i32* %2
  br label %bb3
bb3:
  %r0 = phi i32 [ 0, %bb1 ], [ %3, %bb2 ]
  ret i32 %r0
}
```

条件跳转

直接跳转

如前序代码块为%bb1，则%8=0，
如前序代码块为%bb2，则
%8=%3

IR定义：逻辑运算

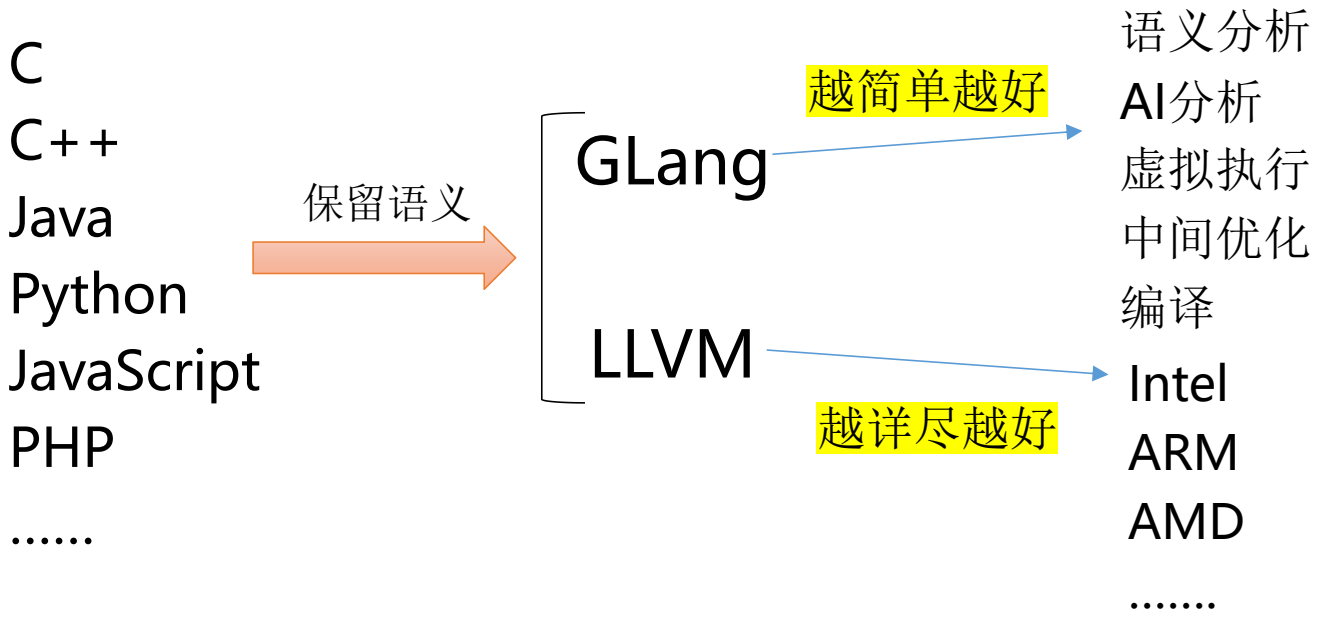
- 无需定义专门的逻辑“与”和“非”指令

```
%7 = xor i1 %6, true
```

→ Not运算：!%6

面向语义逻辑的中间语言General Language / GLang

- GLang IR
  - 理念：保留最少的符号信息



例子：python

```
def condition():
    a = "hello"
    b = a + " world"
    if len(a) > 0:
        x = b
    elif 3 != 4:
        x = 5
    else:
        x = 7
    return x
```


```
[{‘method_decl’: {'name': 'condition',
                  'body’: [
                        {'variable_decl': {'name': 'a'}},
                         {'assign_stmt': {'target': 'a', 'operand': '"hello"'}},
                         {'assign_stmt': {'target': '%v0', 'operator': '+', 'operand': 'a', 'operand2': '" world"'}},
                         {'variable_decl': {'name': 'b'}},
                         {'assign_stmt': {'target': 'b', 'operand': '%v0'}},
                         {'call_stmt': {'target': '%v1', 'name': 'len', 'args': ['a']}},
                         {'assign_stmt': {'target': '%v2', 'operator': '>', 'operand': '%v1', 'operand2': '0'}},
                         {'if_stmt': {'condition': '%v2',
                                 'then_body’: [
                                         {'variable_decl': {'name': 'x'}},
                                         {'assign_stmt': {'target': 'x', 'operand': 'b'}}],
                                 'else_body’: [
                                         {'assign_stmt': {'target': '%v3', 'operator': '!=', 'operand': '3', 'operand2': '4'}},
                                         {'if_stmt': {'condition': '%v3',
                                                 'then_body': [{'variable_decl': {'name': 'x'}},
                                                         {'assign_stmt': {'target': 'x', 'operand': '5'}}],
                                                 'else_body': [{'variable_decl': {'name': 'x'}},
                                                         {'assign_stmt': {'target': 'x', 'operand': '7'}}]}}]}]}},
                         {'return_stmt': {'target': 'x'}}]}}]
```


CODE SECURITY RESEARCH

例子

	operation	parent_stmt_id	stmt_id	name	attrs	unit_id	parameters	body	receiver_object	field	source	target	data_type	args	operand
0	variable_decl	0	10	aa	['global']	4									
1	variable_decl	0	11	dd	['global']	4									
2	method_decl	0	12	hh		4	13.0	15.0							
3	block_start	12	13			4									
4	parameter_decl	13	14	name		4									
5	block_end	12	13			4									
6	block_start	12	15			4									
7	field_write	15	16			4			%this	name	name				
8	block_end	12	15			4									
9	method_decl	0	17	bb		4	18.0	20.0							
10	block_start	17	18			4									
11	parameter_decl	18	19	n		4									
12	block_end	17	18			4									
13	block_start	17	20			4									
14	field_write	20	21			4			%this	name	n				
15	block_end	17	20			4									
16	variable_decl	0	29	cc	['const']	4									
17	method_decl	0	31	%unit_init		4		32.0							
18	block_start	31	32			4									
19	field_read	32	22			4			hh	prototype		%vv1			
20	field_write	32	23			4			%vv1	sayhello	bb				
21	field_read	32	24			4			hh	prototype		aa			
22	field_read	32	26			4			hh	sayhello		dd			
23	new_object	32	28			4						%vv4	hh		
24	assign_stmt	32	30			4						cc			%vv4
25	block_end	31	32			4									

CODE SECURITY
RESEARCH

GLang

控制流语句

return_stmt	target				
if_stmt	condition	then_body	else_body		
dowhile_stmt	condition	body			
while_stmt	condition	body	else_body		
for_stmt	init_body	condition	condition_prebody	update_body	body
forin_stmt	attr	data_type	name	target	body
switch_stmt	condition	body			
case_stmt	condition	body			
default_stmt	body				
break_stmt	target				
continue_stmt	target				
goto_stmt	target				
yield_stmt	target				
sync_stmt					
throw_stmt	target				
try_stmt	body	catch_body	else_body	final_body	
catch_stmt	body				
final_stmt	body				
label_stmt	name				
assert_stmt	condition				
del_stmt	target				
raise_stmt	target				
pass_stmt					
break_stmt	target				
global_stmt	target				
nonlocal_stmt	target				
type_alias_stmt	target	source			
with_stmt	attr	with_init	body		
block_start	stmt_id	parent_stmt_id			
block_end	stmt_id	parent_stmt_id			
new_array	target	attr	data_type		

符号

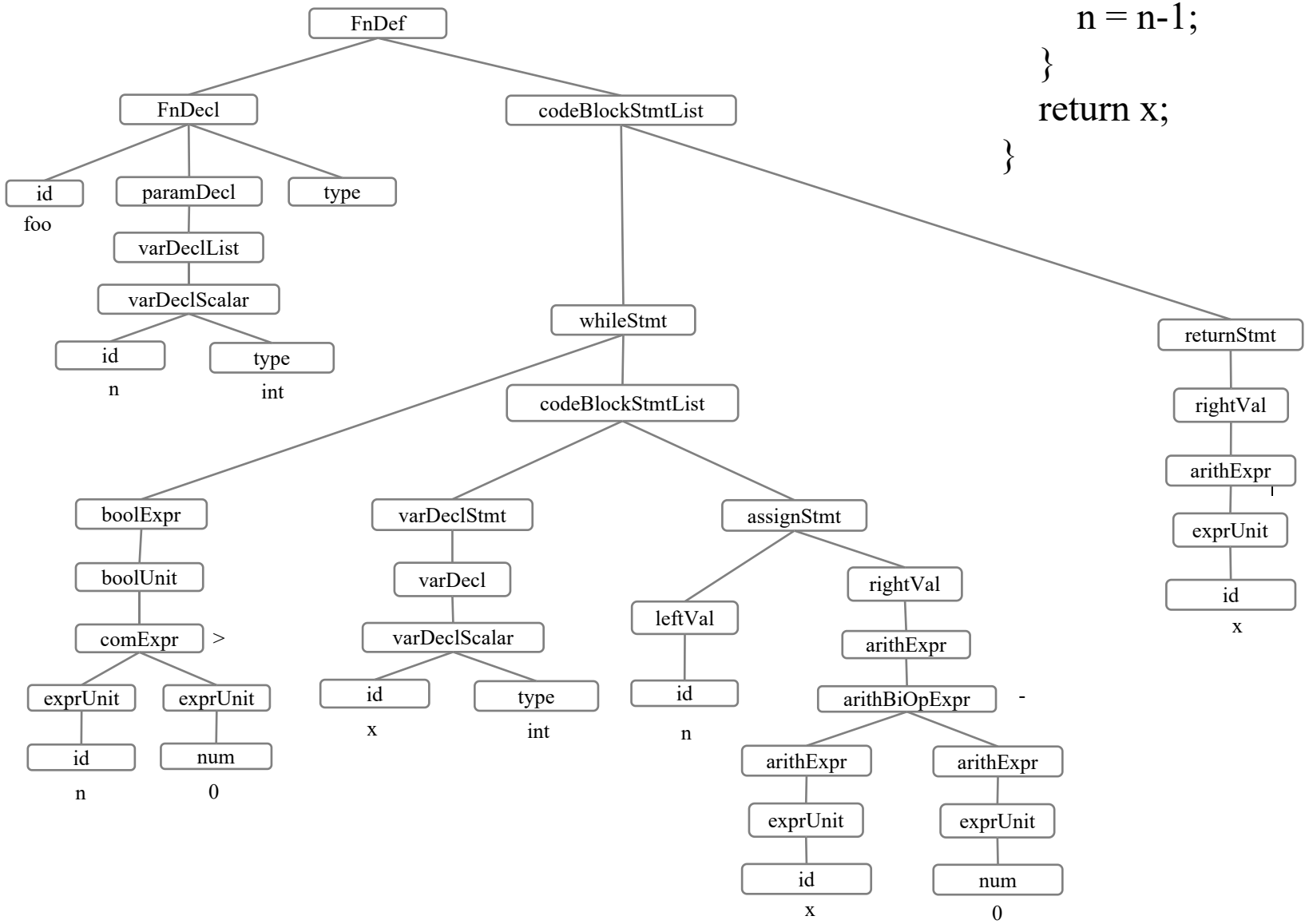
- 临时变量%+vv数字
- attrs:
  - 一条语句中的修饰符，例如private\public

如何把AST翻译为Glang?

CODE SECURITY
RESEARCH

AST转IR：自顶向下翻译

```
int foo(int n){
    while (n>0) {
            int x;
        n = n-1;
    }
    return x;
}
```



AST转IR：自顶向下翻译

- 详见例子common_parser和java_parser
  - Common_parser负责分发
  - Java_parser负责处理每一条具体语句

难点

- Target, error = f()?
- a, b = l?
- Lambda x: x+1?
- O.f.f.f = 10
- Def f(): return a, b, c, d