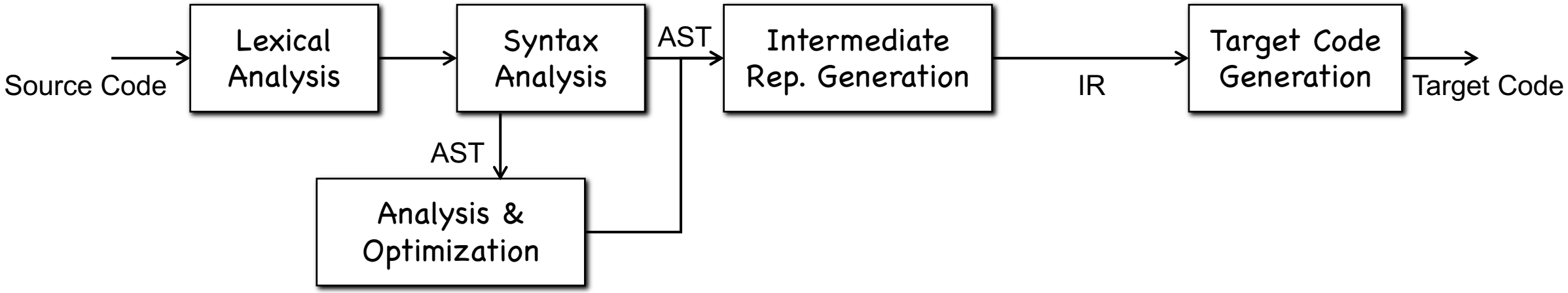
# Chapter 6
# Intermediate Representation (IR)

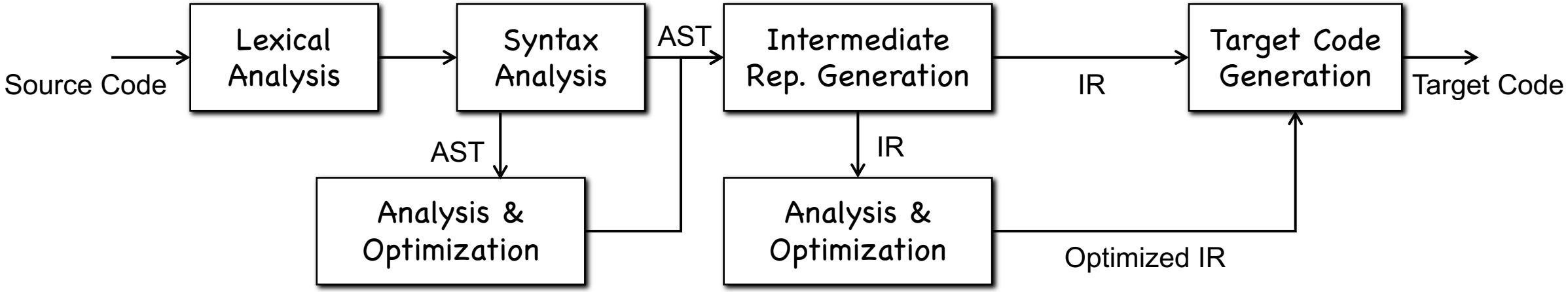
Qingkai Shi
qingkaishi@nju.edu.cn

# Intermediate Representation

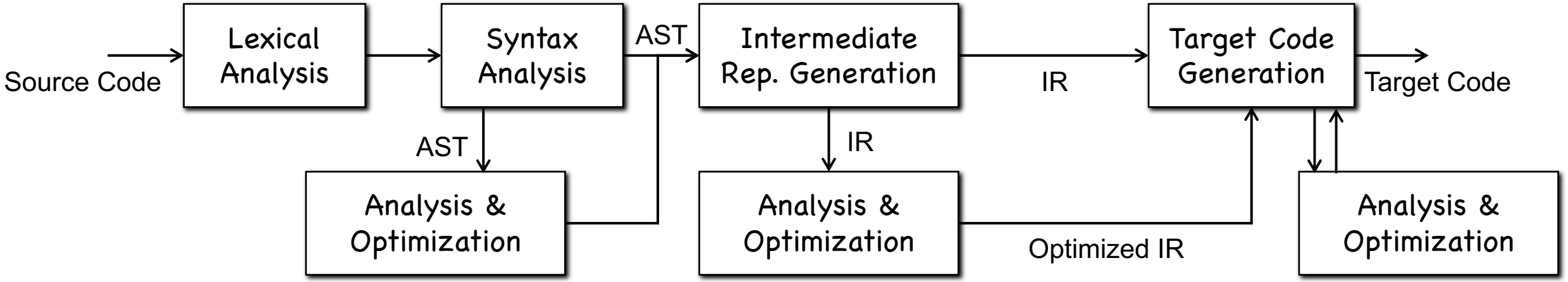
# Intermediate Representation



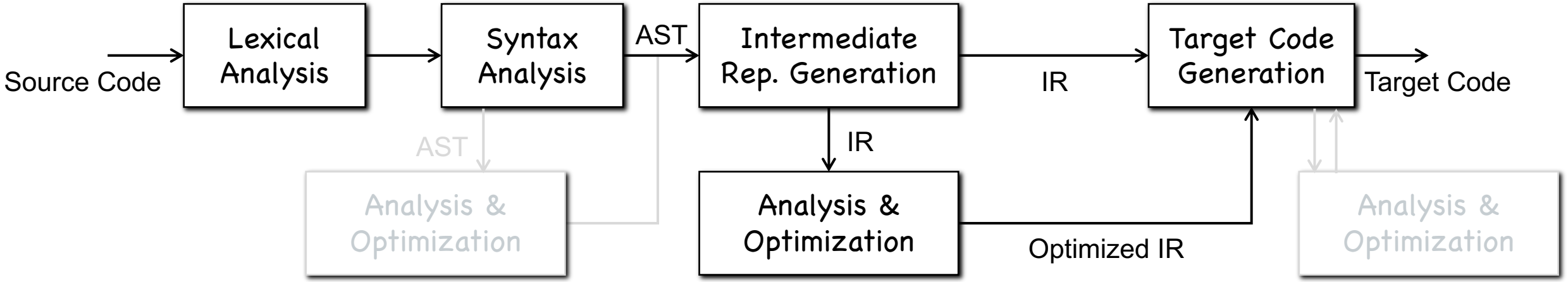
# Intermediate Representation



# Intermediate Representation

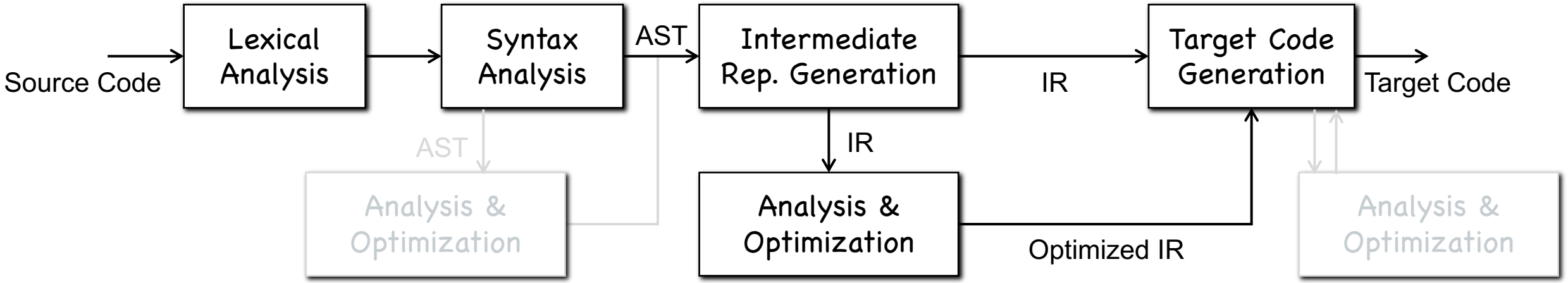


# Intermediate Representation



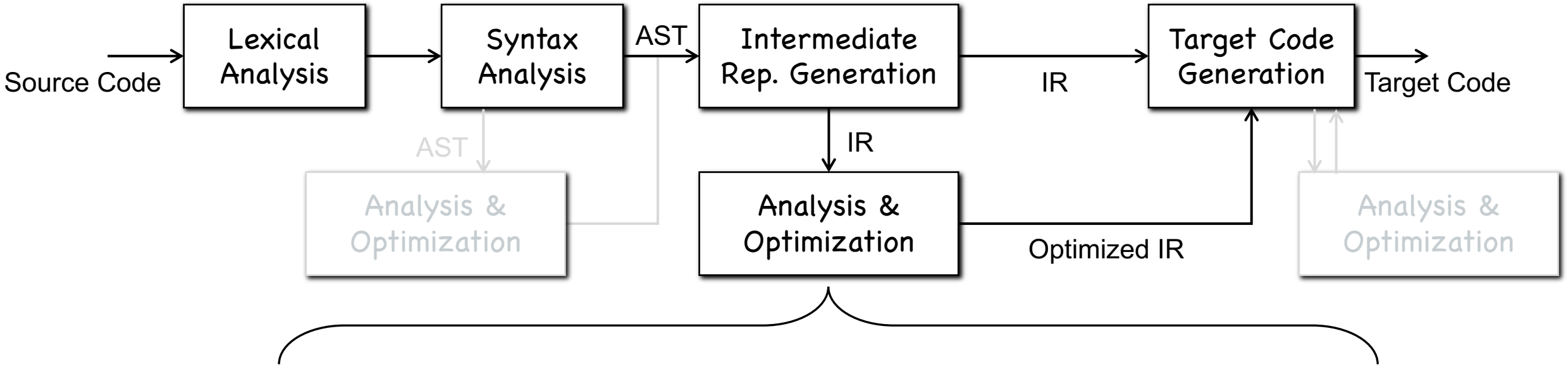

- Using *syntax-directed translation* to generate IR (**Ch 6**)

# Intermediate Representation




- Using *syntax-directed translation* to generate IR (**Ch 6**)
- Intermediate rep. is friendly to analysis and optimization (**Ch 9**)
  - Better structure, including control flow graph, etc.
  - Rich source code information, including types, etc.

# Intermediate Representation



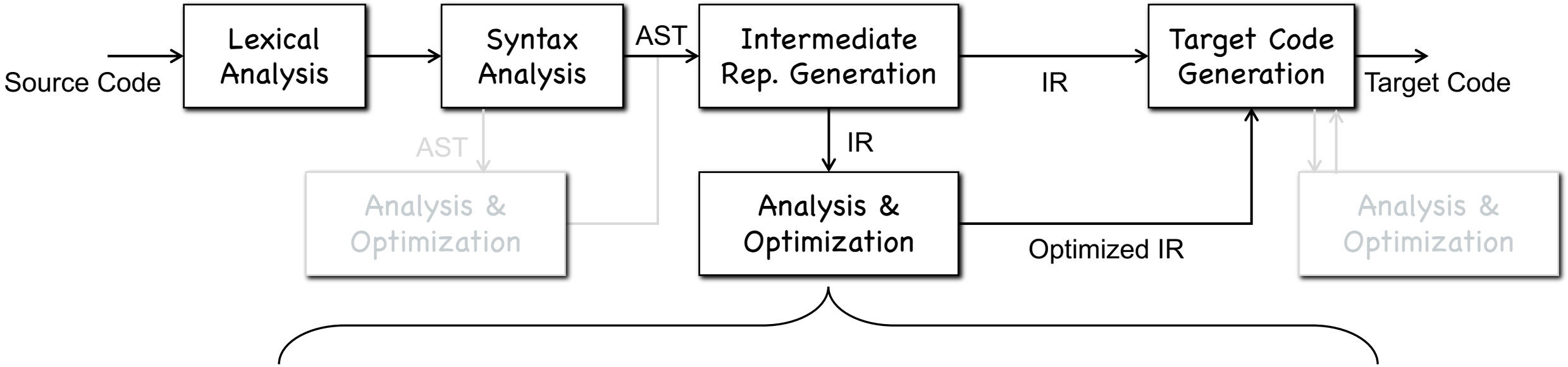

**Machine Independent Analysis**

1. Type inference
2. Bug detection
3. Code instrumentation
   a) logging behaviors
   b) defending attacks

**Machine Independent Optimization**

1. Dead code elimination
2. Constant propagation
3. Live variable analysis
4. Vectorization
5. …

# Intermediate Representation



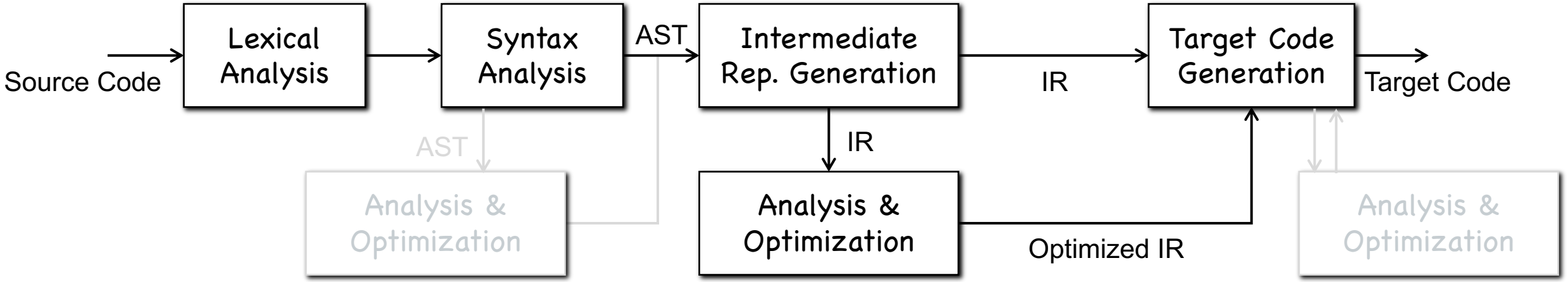

**Machine Independent Analysis**

1. Type inference
2. Bug detection
3. Code instrumentation
   a) logging behaviors
   b) defending attacks

**Machine Independent Optimization**

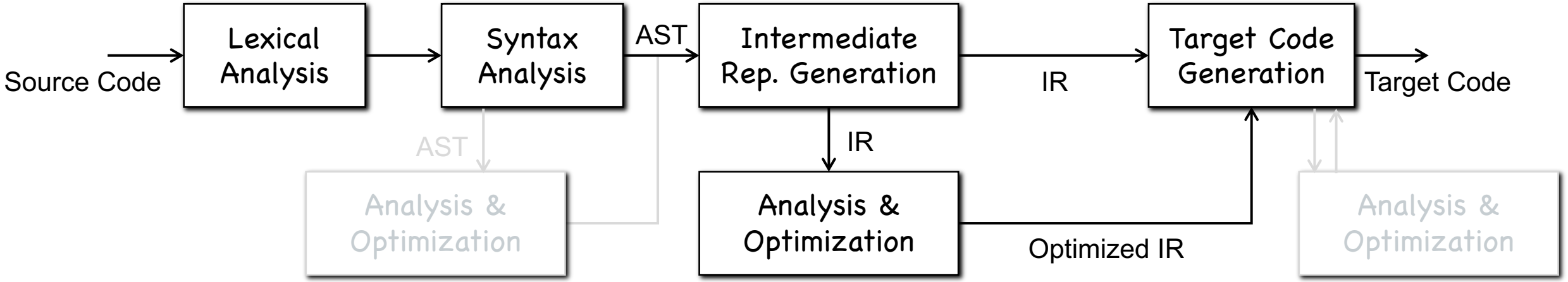
1. Dead code elimination
2. Constant propagation
3. Live variable analysis
4. Vectorization
5. …

# Intermediate Representation




```
char get(char *buffer, int size, int index) {
    return buffer[index];
}
```

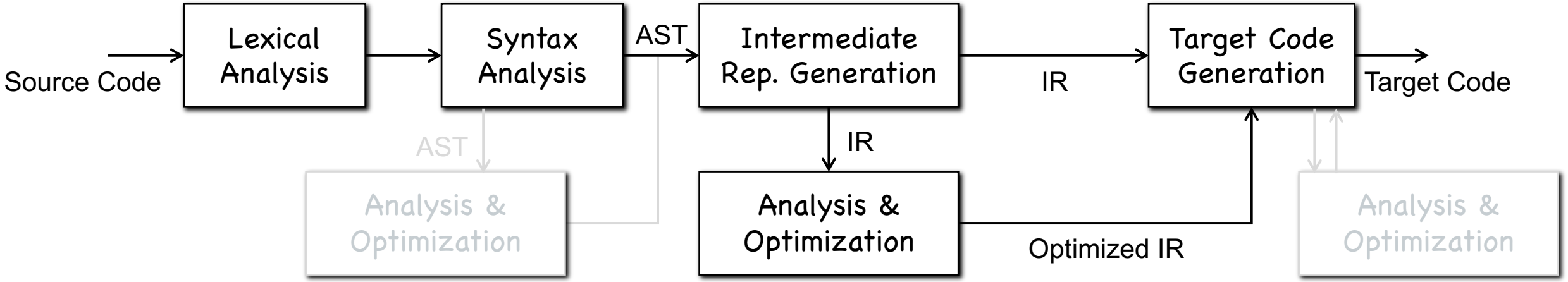
# Intermediate Representation




```
char get(char *buffer, int size, int index) {
    return buffer[index];
}
```

```
char get(char *buffer, int size, int index) {
    // if (index >= size) ……
    return buffer[index];
}
```

# Intermediate Representation




```
char get(char *buffer, int size, int index) {
    return buffer[index];
}
```

```
char get(char *buffer, int size, int index) {
    // if (index >= size) ……
    return buffer[index];
}
```

**CVE-2022-26125 Detail**

> 1000 Bugs, > 100 CVEs in mature systems, e.g., Apache, MySQL, etc.

## Description

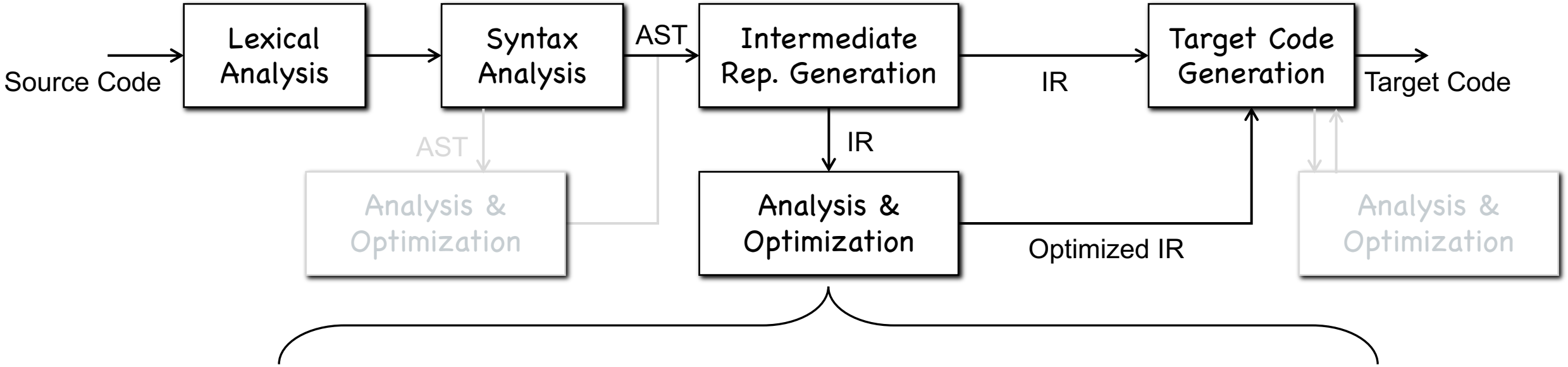
Buffer overflow vulnerabilities exist in FRRouting through 8.1.0 due to wrong checks on the input packet length in isisd/isis_tlvs.c.

**Severity** CVSS Version 3.x | CVSS Version 2.0

**CVSS 3.x Severity and Metrics:**

**NIST:** NVD **Base Score:** 7.8 HIGH **Vector:** CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

# Intermediate Representation



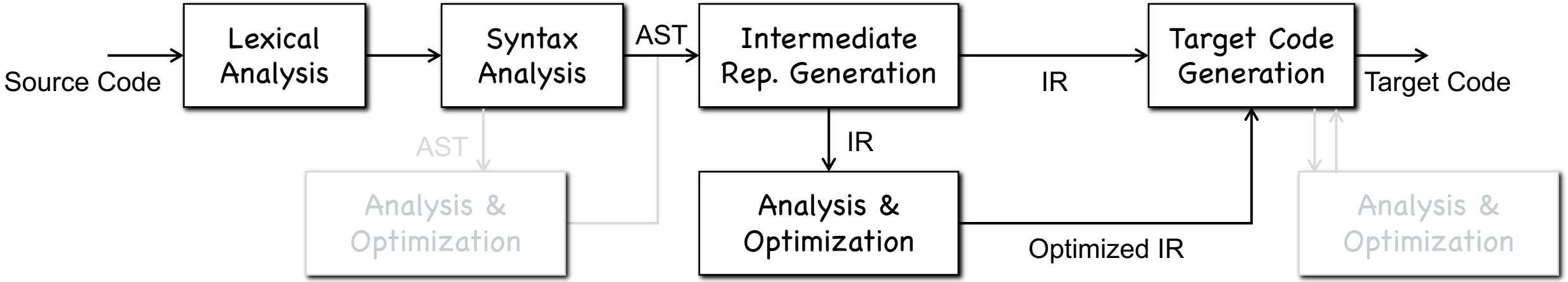

**Machine Independent Analysis**

1. Type inference
2. Bug detection
3. Code instrumentation
   a) logging behaviors
   b) defending attacks

**Machine Independent Optimization**

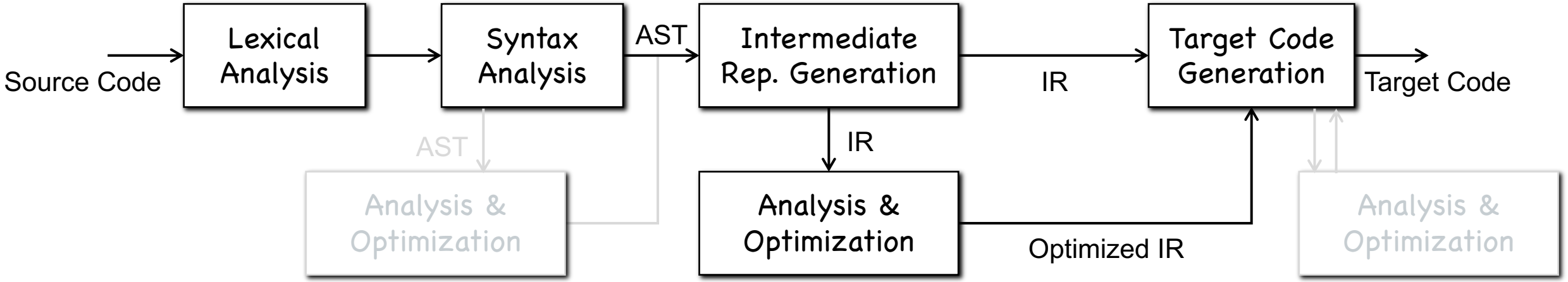
1. Dead code elimination
2. Constant propagation
3. Live variable analysis
4. Vectorization
5. …

# Intermediate Representation




```
class List { … }
class ArrayList: public List { … }
class LinkedList: public List { … }

void foo(List *list) { list->bar(); }
```

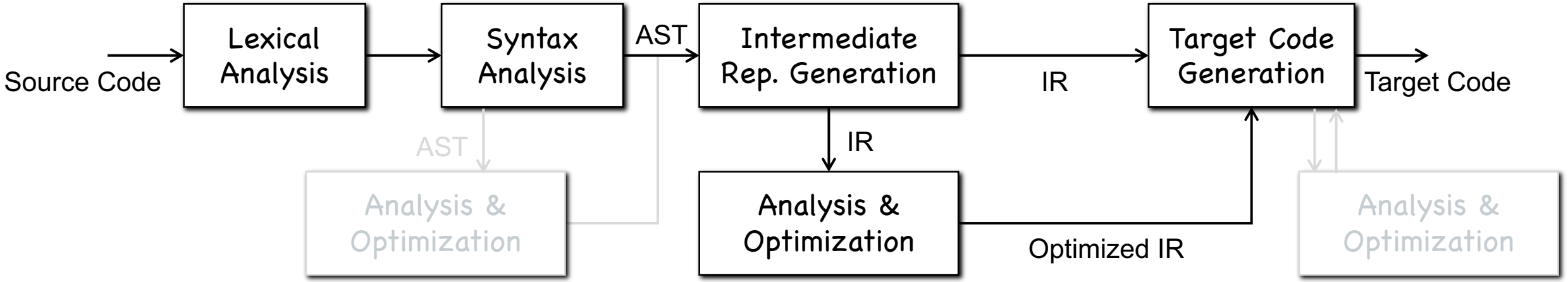
# Intermediate Representation




```
class List { … }
class ArrayList: public List { … }
class LinkedList: public List { … }

void foo(List *list) { list->bar(); }
```

```
class List { … }
class ArrayList: public List { … }
class LinkedList: public List { … }

void foo(ArrayList *list) { (*list).bar(); }
```

# Intermediate Representation




```
class List { … }
class ArrayList: public List { … }
class LinkedList: public List { … }

void foo(List *list) { list->bar(); }
```

```
class List { … }
class ArrayList: public List { … }
class LinkedList: public List { … }

void foo(ArrayList *list) { (*list).bar(); }
```

Code Size Reduction!
Time Cost Reduction!

# Intermediate Representation

- **Part I: Definition of Intermediate Representation**
  - Three-Address Code
  - Static Single-Assignment Form
  - LLVM IR
- **Part II: Generation of Intermediate Representation**
  - Generating IR for Expressions
  - Generating IR for Control Flows

# PART I: Definition of IR

# Three-Address Code

- Provide a standard representation, not as flexible as the source

# Three-Address Code

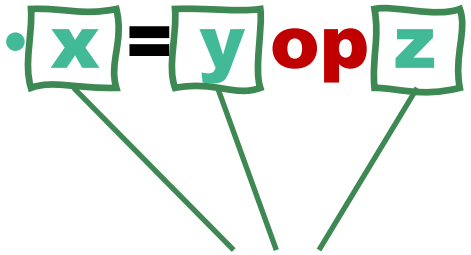
- Provide a standard representation, not as flexible as the source

- At most one operator on the right side of each instruction
- At most three **addresses/variables** in an instruction
  - **x = y op z**

# Three-Address Code

- Provide a standard representation, not as flexible as the source

- At most one operator on the right side of each instruction
- At most three **addresses/variables** in an instruction




Variables or constant;

Variables are from the source code or created by the compiler

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Computation and Assignment** | | |
| Arithmetic Operation | x = y ⊕ z;  x = ⊖ z; | x = y + z;  x = -z; |
| Logic Operation | x = y ⊕ z;  x = ⊖ z; | x = y && z;  x = !z; |
| Relational Operation | x = y ⊕ z; | x = y > z; |
| Copy/Assignment | x = y; | - |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Computation and Assignment** | | |
| Arithmetic Operation | x = y ⊕ z;  x = ⊖ z; | x = y + z;  x = -z; |
| Logic Operation | x = y ⊕ z;  x = ⊖ z; | x = y && z;  x = !z; |
| Relational Operation | x = y ⊕ z; | x = y > z; |
| Copy/Assignment | x = y; | - |
| **Control Flow** | | |
| Unconditional Jump | goto L; | - |
| Conditional Jump | if x goto L else goto L’; | - |
| | if x⊕y  goto L else goto L’; | if x > y  goto L else goto L’; |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Computation and Assignment** | | |
| Arithmetic Operation | x = y ⊕ z;  x = ⊖ z; | x = y + z;  x = -z; |
| Logic Operation | x = y ⊕ z;  x = ⊖ z; | x = y && z;  x = !z; |
| Relational Operation | x = y ⊕ z; | x = y > z; |
| Copy/Assignment | x = y; | - |
| **Control Flow** | | |
| Unconditional Jump | goto L; | - |
| Conditional Jump | if x goto L else goto L’; | => if x goto L; |
| | if x⊕y goto L else goto L’; | if x > y goto L else goto L’; |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Computation and Assignment** | | |
| Arithmetic Operation | x = y ⊕ z;  x = ⊖ z; | x = y + z;  x = -z; |
| Logic Operation | x = y ⊕ z;  x = ⊖ z; | x = y && z;  x = !z; |
| Relational Operation | x = y ⊕ z; | x = y > z; |
| Copy/Assignment | x = y; | - |
| **Control Flow** | | |
| Unconditional Jump | goto L; | - |
| Conditional Jump | if x goto L else goto L'; | => if x goto L; |
| | if x⊕y goto L else goto L'; | if x > y goto L else goto L'; |
| | | => if x > y goto L; |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Computation and Assignment** | | |
| Arithmetic Operation | x = y ⊕ z;  x = ⊖ z; | x = y + z;  x = -z; |
| Logic Operation | x = y ⊕ z;  x = ⊖ z; | x = y && z;  x = !z; |
| Relational Operation | x = y ⊕ z; | x = y > z; |
| Copy/Assignment | x = y; | - |
| **Control Flow** | | |
| Unconditional Jump | goto L; | - |
| Conditional Jump | if x goto L else goto L'; | => if x goto L; |
| | if x⊕y goto L else goto L'; | if x > y goto L else goto L'; |
| | | => if x > y goto L;  ≡ z = x > y; if z goto L; |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Memory Operations** | | |
| Indexed Copy | x = y[z];  y[z] = x; | |
| Address-Taken | x = &y; | - |
| Load Operation | x = *y; | - |
| Store Operation | *x = y; | - |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| | **Memory Operations** | |
| Indexed Copy | x = y[z]; y[z] = x; | x = y[z]; ≣ t = y + z; x = *t; |
| Address-Taken | x = &y; | - |
| Load Operation | x = *y; | - |
| Store Operation | *x = y; | - |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Memory Operations** | | |
| Indexed Copy | x = y[z]; y[z] = x; | x = y[z]; ≡ t = y + z; x = *t; / y[z] = x; ≡ t = y + z; *t = x; |
| Address-Taken | x = &y; | - |
| Load Operation | x = *y; | - |
| Store Operation | *x = y; | - |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Memory Operations** | | |
| Indexed Copy | x = y[z]; y[z] = x; x = y[z]; ≡ t = y + z; x = *t; | y[z] = x; ≡ t = y + z; *t = x; |
| Address-Taken | x = &y; | - |
| Load Operation | x = *y; | - |
| Store Operation | *x = y; | - |
| **Function Call and Return** | | |
| Function Call | param $p_1$;<br>param $p_2$;<br>...<br>param $p_n$;<br>x = call func n; call func n; | param 1;<br>param 2;<br>call foo 2; |
| Function Return | return; return x; | - |

# Three-Address Code

| Instruction Type | Instruction Forms | Examples |
|---|---|---|
| **Memory Operations** | | |
| Indexed Copy | x = y[z]; y[z] = x; x = y[z]; ≡ t = y + z; x = *t; | y[z] = x; ≡ t = y + z; *t = x; |
| Address-Taken | x = &y; | - |
| Load Operation | x = *y; | - |
| Store Operation | *x = y; | - |
| **Function Call and Return** | | |
| Function Call | param $p_1$;<br>param $p_2$;<br>...<br>param $p_n$;<br>x = call func n; call func n; | param 1;<br>param 2;<br>call foo 2;<br>=> foo(1, 2) |
| Function Return | return; return x; | - |

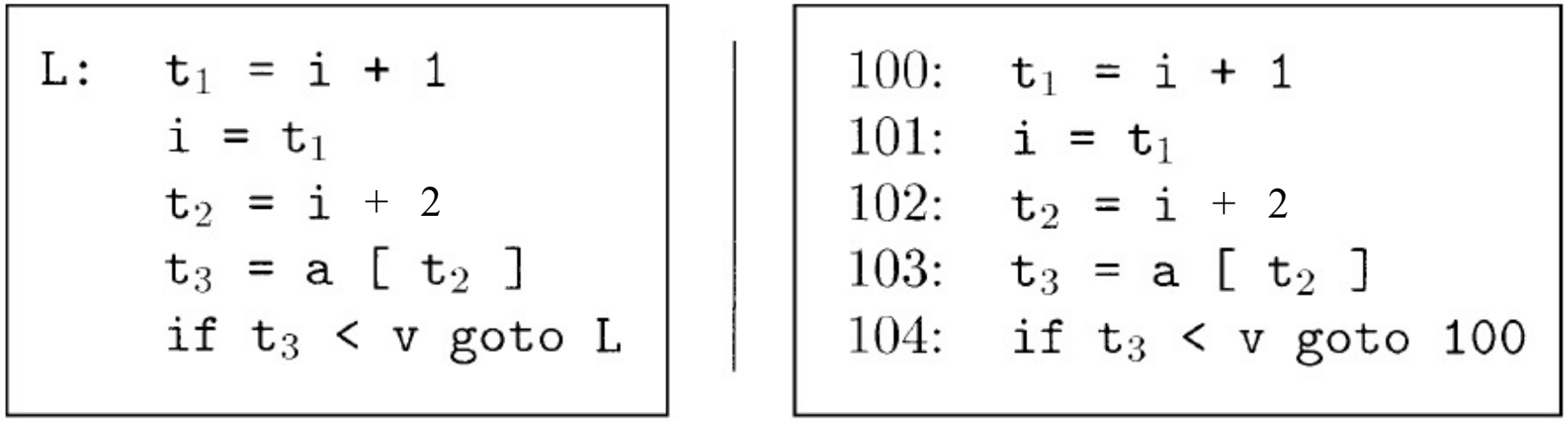
# Three-Address Code

- do i = i + 1; while (a[i + 2] < v);

Try!

# Three-Address Code

- do i = i + 1; while (a[i + 2] < v);




Symbolic Labels

Numeric Labels

- Implementation methods: *quadruples*, *triples*, etc.

# Quadruples

- a = b * (-c) + b * (-c)

```
t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
 a = t5
```

Three-Address Code

| | *op* | $arg_1$ | $arg_2$ | *result* |
|---|---|---|---|---|
| 0 | minus | c | | $t_1$ |
| 1 | * | b | $t_1$ | $t_2$ |
| 2 | minus | c | | $t_3$ |
| 3 | * | b | $t_3$ | $t_4$ |
| 4 | + | $t_2$ | $t_4$ | $t_5$ |
| 5 | = | $t_5$ | | a |
| | ... | | | |

Quadruples

# Quadruples

- a = b * (-c) + b * (-c)

| | | $op$ | $arg_1$ | $arg_2$ | $result$ |
|---|---|---|---|---|---|
| $t_1$ = minus c | 0 | minus | c | | $t_1$ |
| $t_2$ = b * $t_1$ | 1 | * | b | $t_1$ | $t_2$ |
| $t_3$ = minus c | 2 | minus | c | | $t_3$ |
| $t_4$ = b * $t_3$ | 3 | * | b | $t_3$ | $t_4$ |
| $t_5$ = $t_2$ + $t_4$ | 4 | + | $t_2$ | $t_4$ | $t_5$ |
| a = $t_5$ | 5 | = | $t_5$ | | a |
| | | ... | | | |

Three-Address Code　　Quadruples

# Quadruples

- a = b * (-c) + b * (-c)

```
t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
 a = t5
```

Three-Address Code

| | *op* | $arg_1$ | $arg_2$ | *result* |
|---|---|---|---|---|
| 0 | minus | c | | $t_1$ |
| 1 | * | b | $t_1$ | $t_2$ |
| 2 | minus | c | | $t_3$ |
| 3 | * | b | $t_3$ | $t_4$ |
| 4 | + | $t_2$ | $t_4$ | $t_5$ |
| 5 | = | $t_5$ | | a |
| | ... | | | |

Quadruples

# Quadruples

- a = b * (-c) + b * (-c)

```
t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
a = t5
```

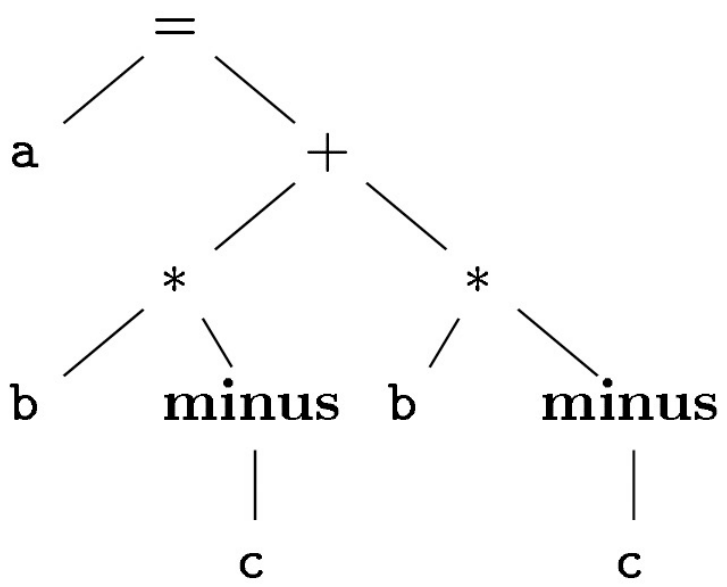
Three-Address Code

| | $op$ | $arg_1$ | $arg_2$ | $result$ |
|---|---|---|---|---|
| 0 | minus | c | | $t_1$ |
| 1 | * | b | $t_1$ | $t_2$ |
| 2 | minus | c | | $t_3$ |
| 3 | * | b | $t_3$ | $t_4$ |
| 4 | + | $t_2$ | $t_4$ | $t_5$ |
| 5 | = | $t_5$ | | a |
| | ... | | | |

Quadruples

# Triples

- a = b * (-c) + b * (-c)



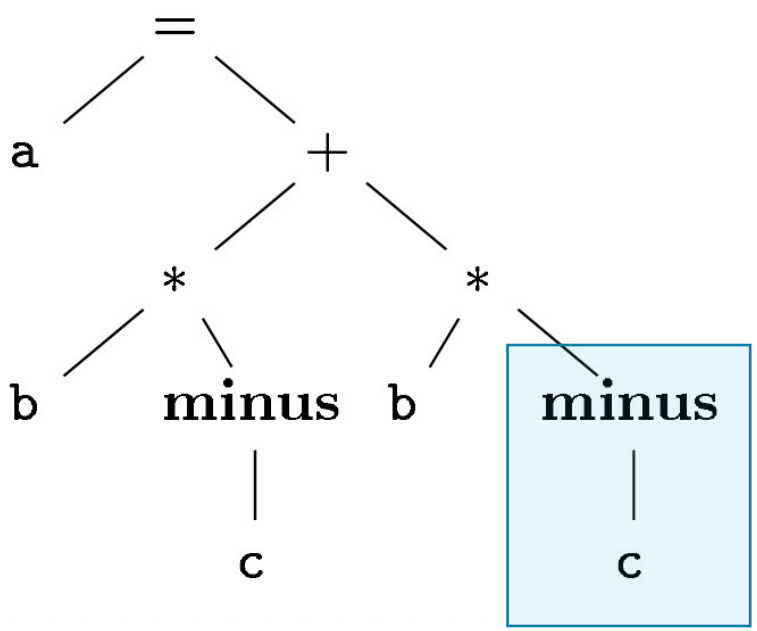

Syntax Tree

| | $op$ | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | ... | | |

Triples

# Triples

- a = b * (-c) + b * (-c)



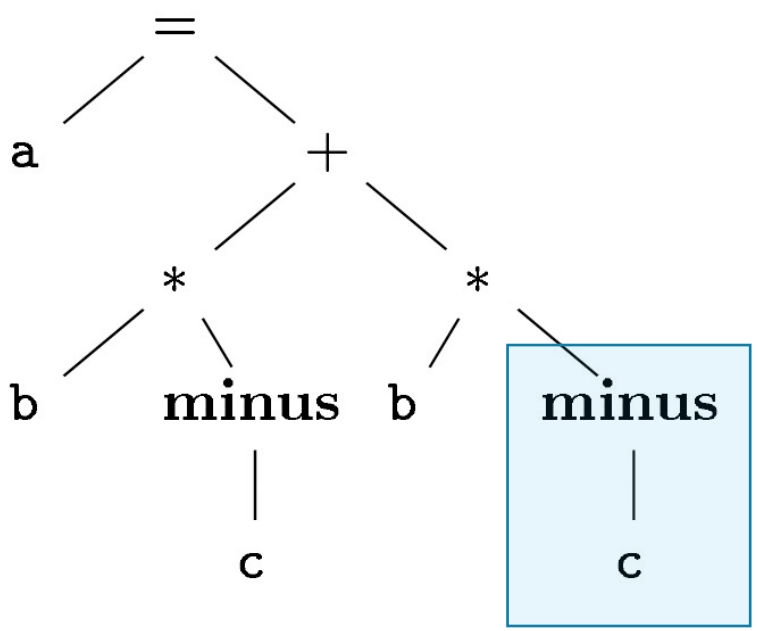

Syntax Tree

| | *op* | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | ... | | |

Triples

# Triples

- a = b * (-c) + b * (-c)



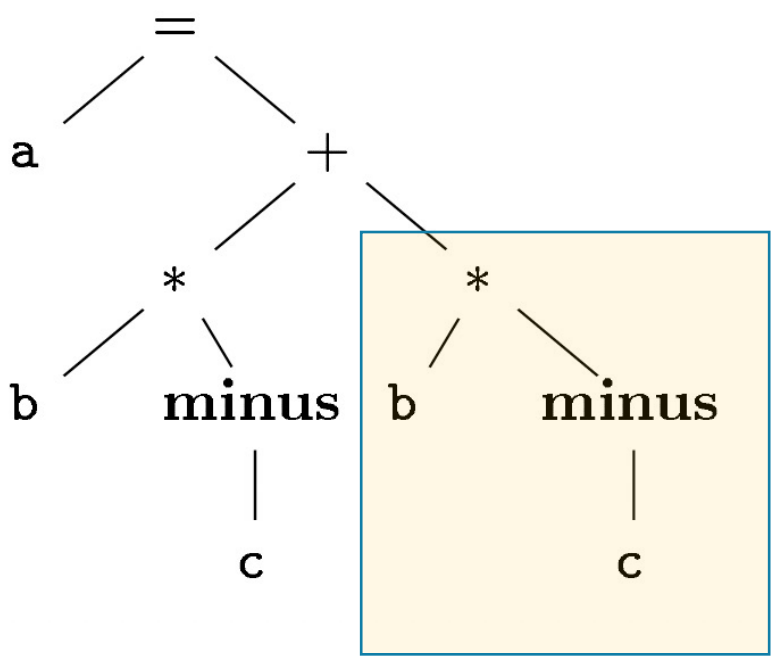

Syntax Tree

| | *op* | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | ... | | |

Triples

# Triples

- a = b * (-c) + b * (-c)



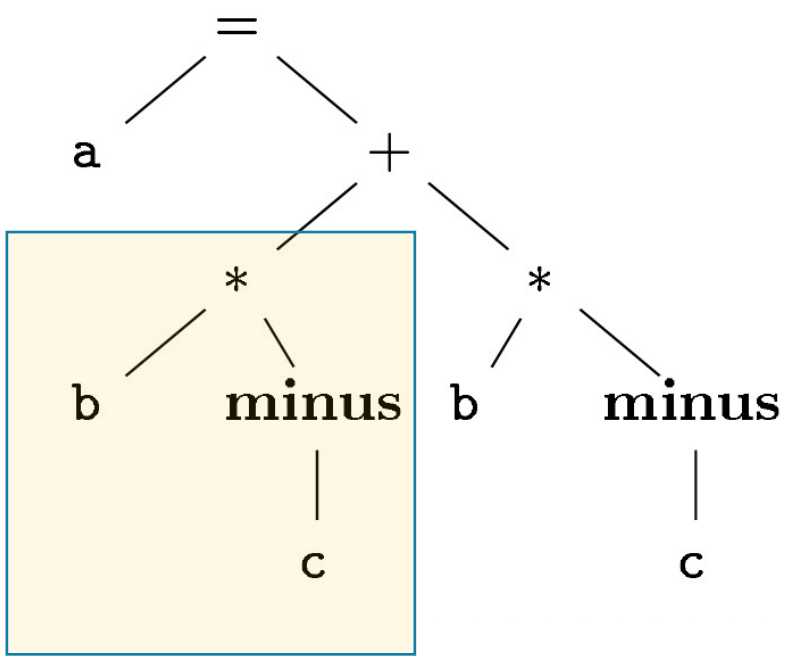

Syntax Tree

| | *op* | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | ... | | |

Triples

# Triples

- a = b * (-c) + b * (-c)



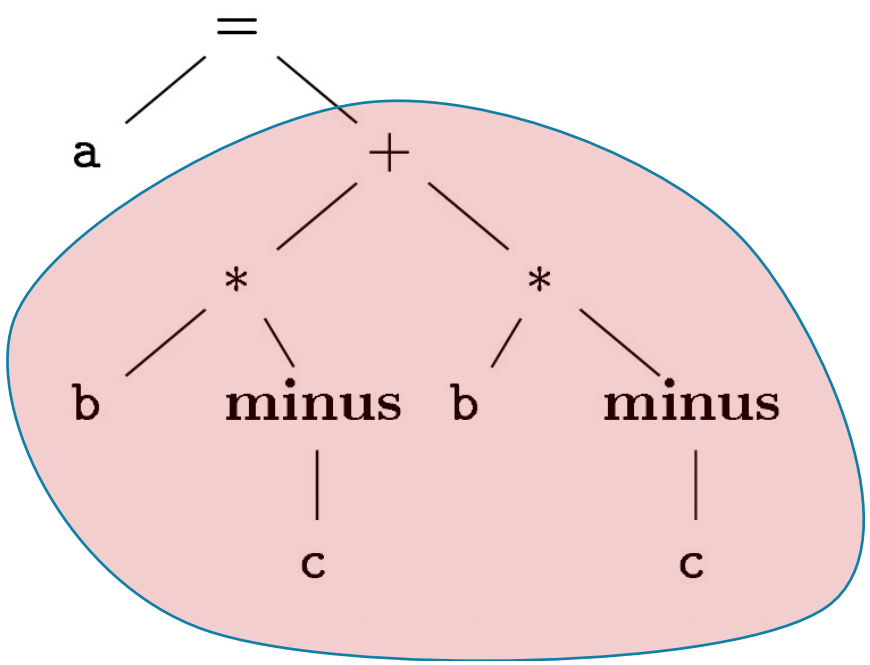

Syntax Tree

| | $op$ | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | ... | | |

Triples

# Triples

- a = b * (-c) + b * (-c)



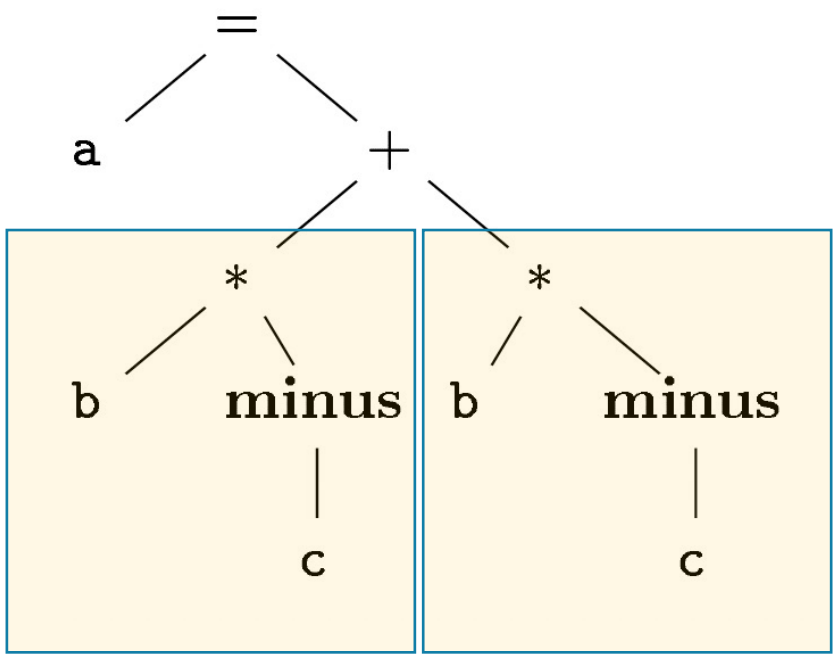

Syntax Tree

| | *op* | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | ... | | |

Triples

# Triples

- a = b * (-c) + b * (-c)



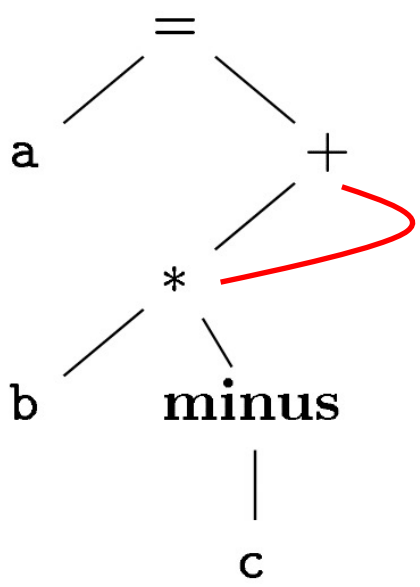

Syntax Tree

| | $op$ | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | ... | | |

Triples

# Triples with Syntax DAG

- a = b * (-c) + b * (-c)



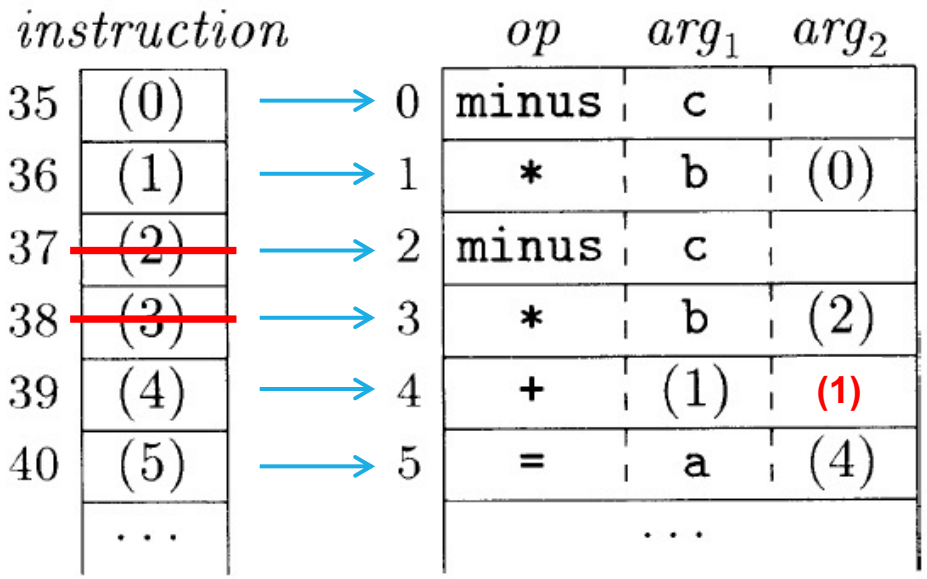

Syntax DAG

| | *op* | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (1) |
| 5 | = | a | (4) |
| | ... | | |

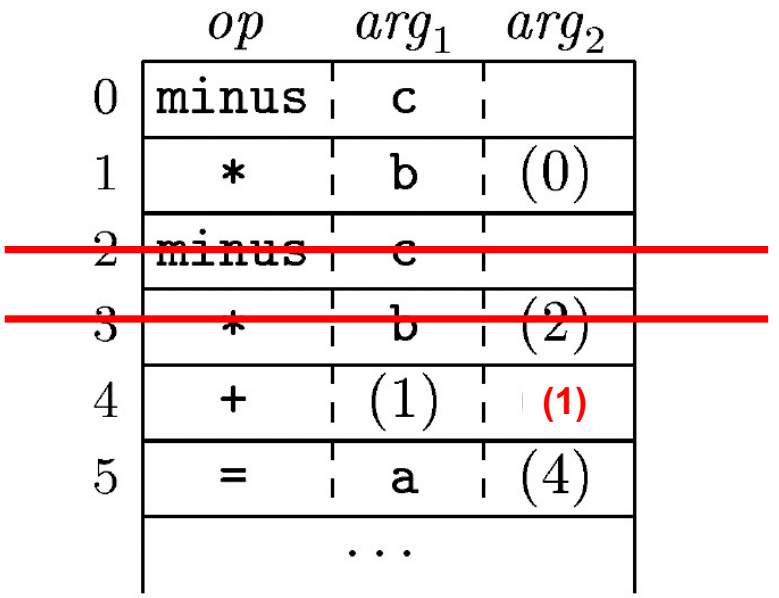
Triples

# Indirect Triples with Syntax DAG

- a = b * (-c) + b * (-c)



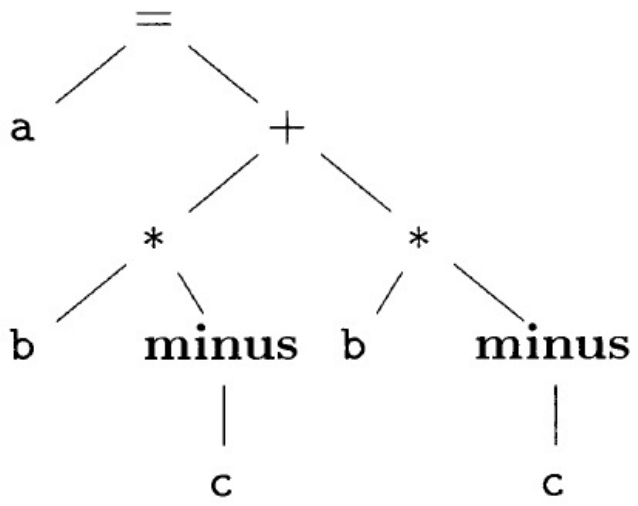

Indirect Triples




Triples

# Generation of Syntax Tree

- a = b * (-c) + b * (-c)

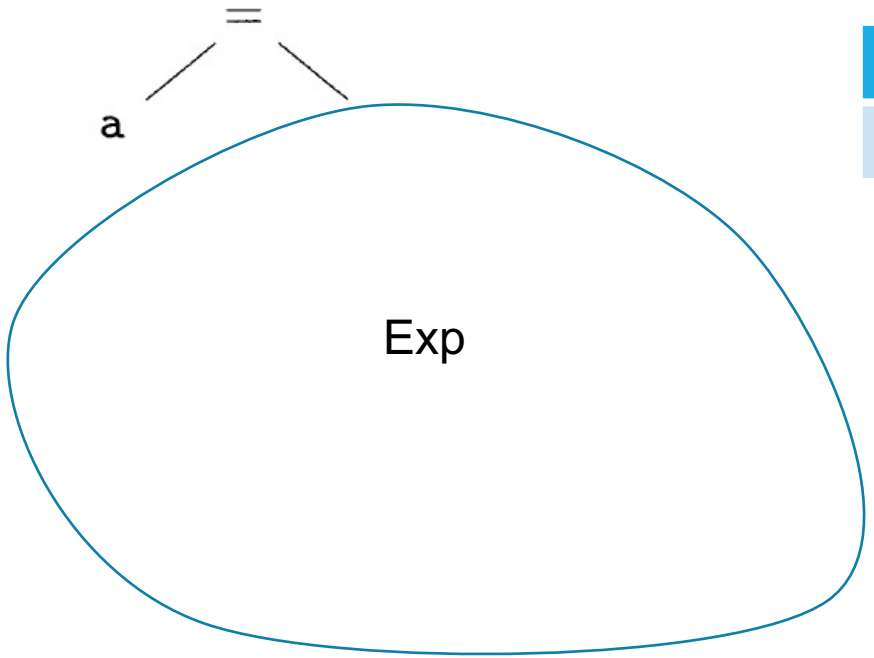



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax Tree

- a = b * (-c) + b * (-c)

variable　　Exp

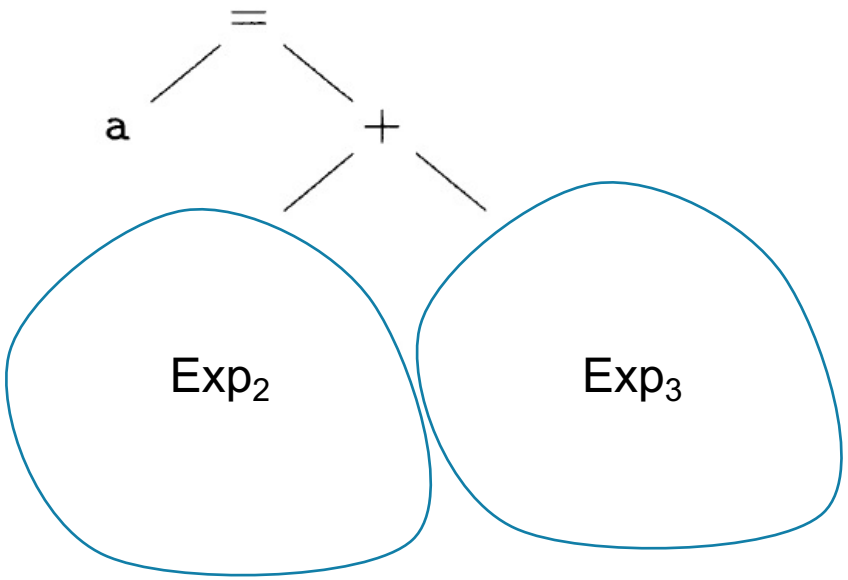



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |

# Generation of Syntax Tree

- a = b * (-c) + b * (-c)

  $Exp_2$ $Exp_3$

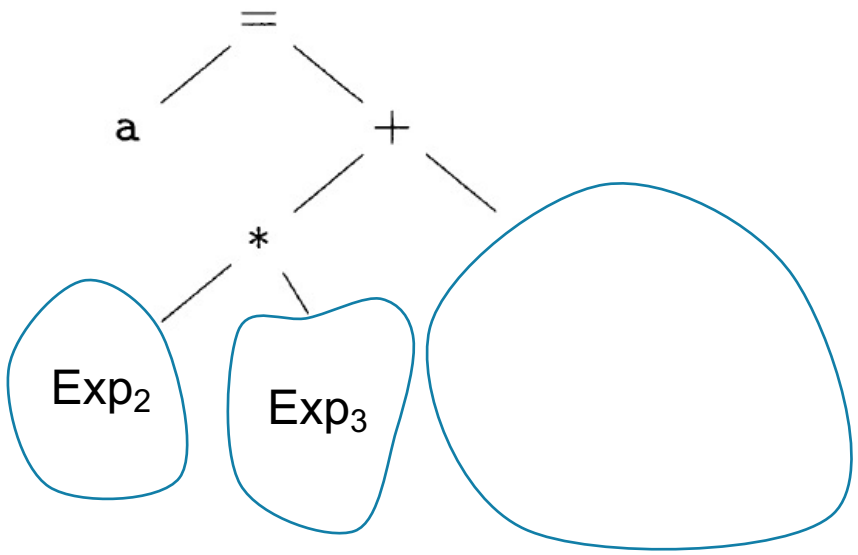



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |

# Generation of Syntax Tree

- a = b * (-c) + b * (-c)

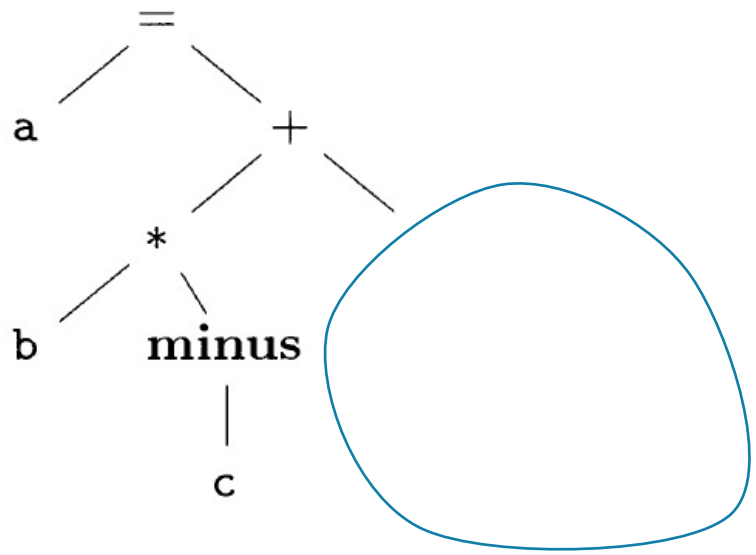
$Exp_2$ $Exp_3$




| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |

# Generation of Syntax Tree

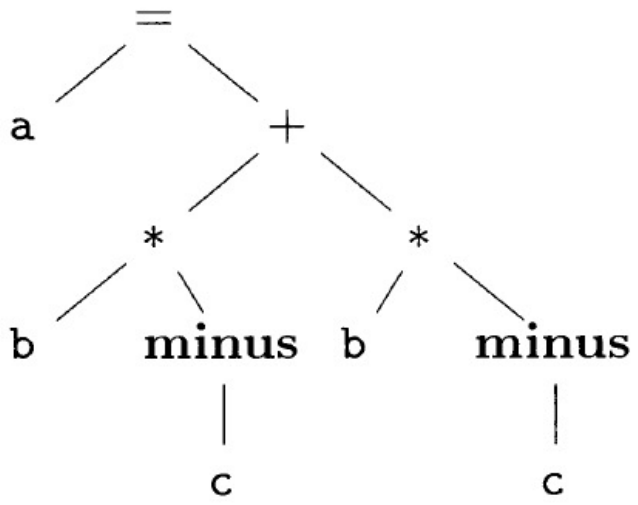
- a = b * (-c) + b * (-c)




| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax Tree

- a = b * (-c) + b * (-c)

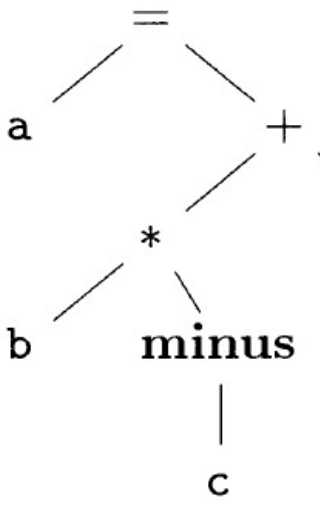



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

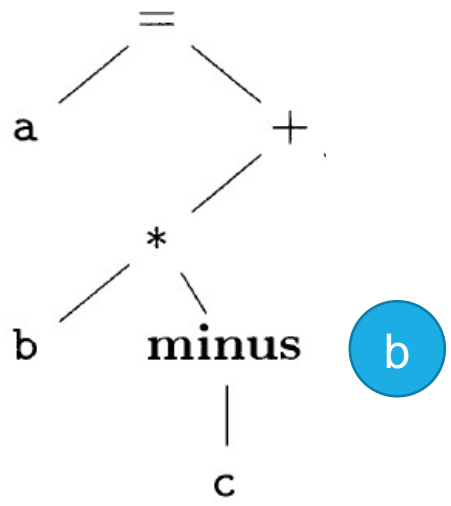



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

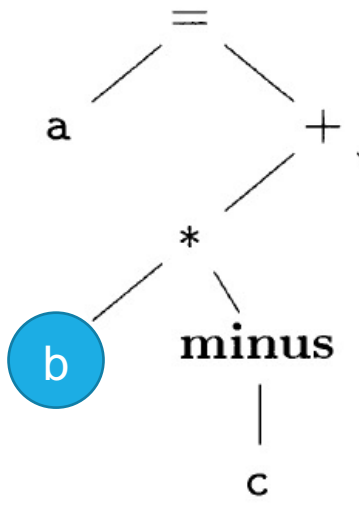



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

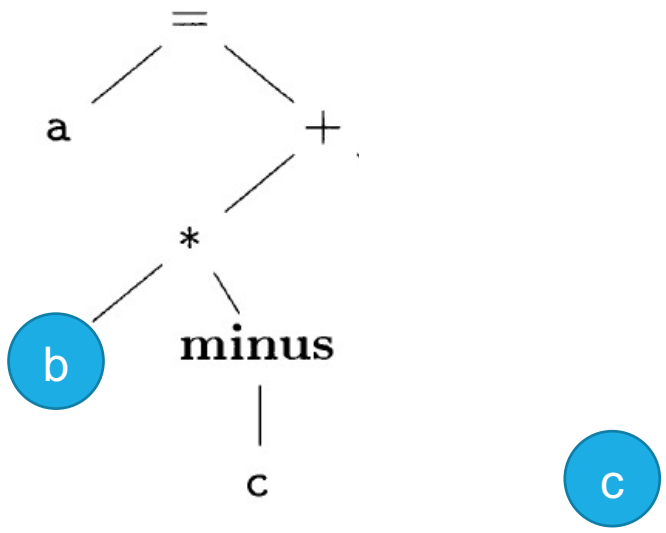



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

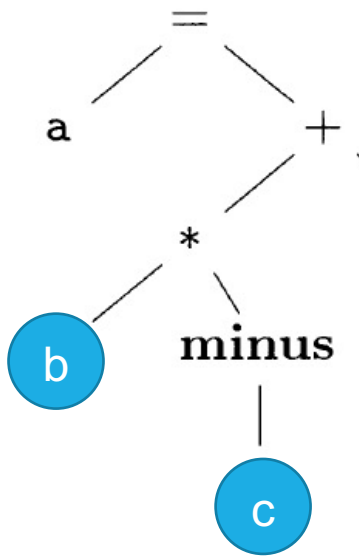



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

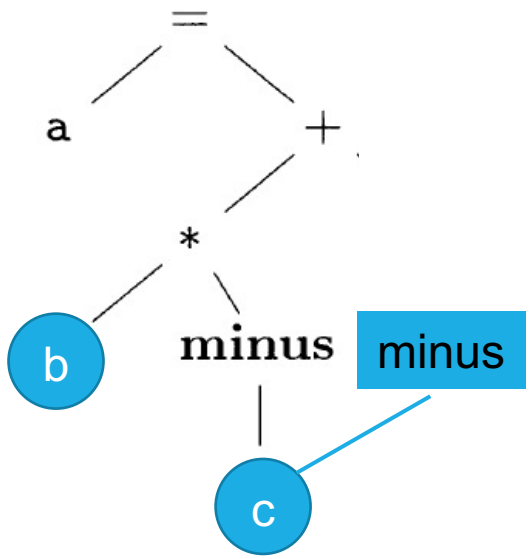



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

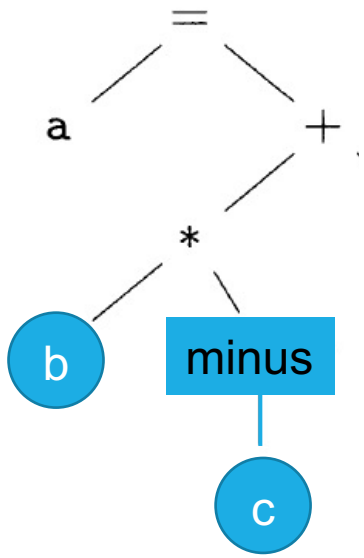



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

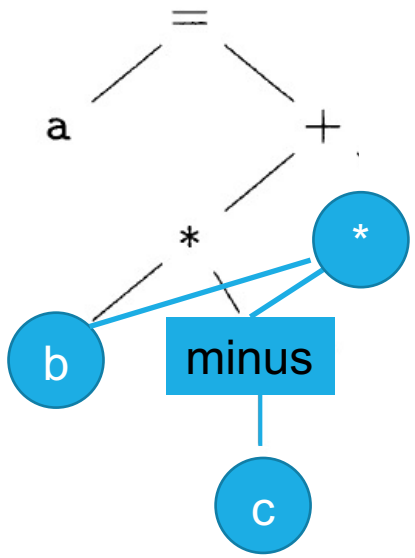



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

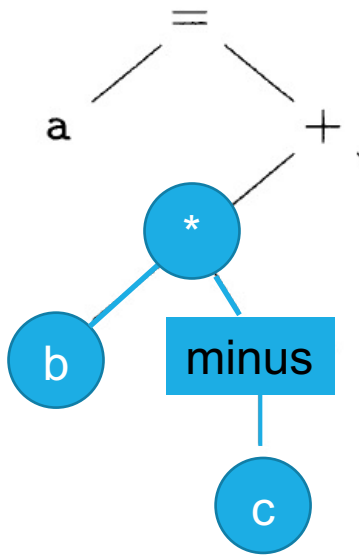



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

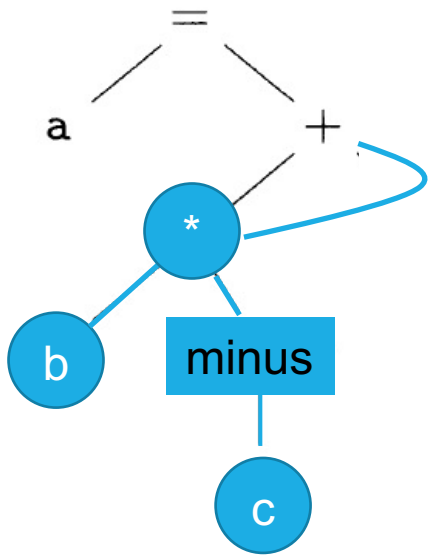



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

$Exp_2$ * $Exp_3$

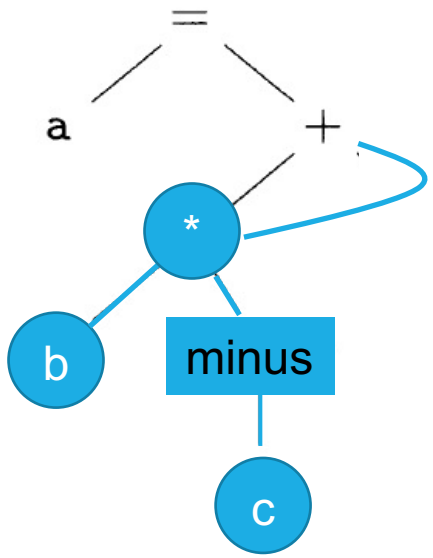



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

# Generation of Syntax DAG

- a = b * (-c) + b * (-c)

  $Exp_2$ * $Exp_3$

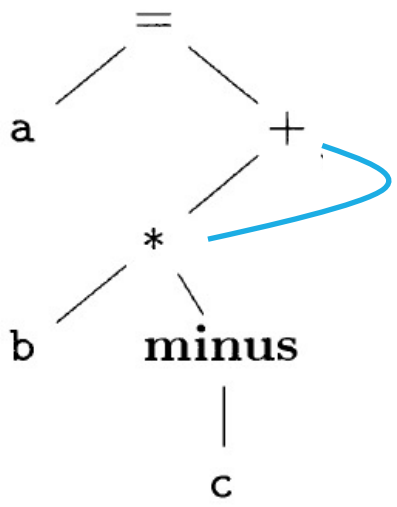



| | Production | Rules |
|---|---|---|
| 1 | Assignment → **variable** = Exp | node(=).addChild(node(**variable**), node(Exp)) |
| 2 | $Exp_1$ → $Exp_2$ + $Exp_3$ | node(+).addChild(node($Exp_2$), node($Exp_3$)) |
| 3 | $Exp_1$ → $Exp_2$ * $Exp_3$ | node(*).addChild(node($Exp_2$), node($Exp_3$)) |
| 4 | $Exp_1$ → (-$Exp_2$) | node(minus).addChild(node($Exp_2$)) |
| 5 | Exp → **variable** | node(**variable**) |

+ Hash consing
  A technique to share values that are structurally equal

# Triples with Syntax DAG

- a = b * (-c) + b * (-c)




Syntax DAG

| | $op$ | $arg_1$ | $arg_2$ |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (1) |
| 5 | = | a | (4) |
| | ... | | |

Triples

# Static Single-Assignment

- What features does SSA have?

# Static Single-Assignment

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using φ to merge definitions from multi paths
- => Direct def-use chains

# Static Single-Assignment

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using φ to merge definitions from multi paths
- => Direct def-use chains

# Static Single-Assignment

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using φ to merge definitions from multi paths
- => Direct def-use chains

```
p = a + b
q = p - c
p = q * d
p = e - p
q = p + q
```

(a) Three-address code.

# Static Single-Assignment

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using φ to merge definitions from multi paths
- => Direct def-use chains

```
p = a + b
q = p - c
p = q * d
p = e - p
q = p + q
```

(a) Three-address code.

```
p1 = a + b
q1 = p1 - c
p2 = q1 * d
p3 = e - p2
q2 = p3 + q1
```

(b) Static single-assignment form.

# Static Single-Assignment

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using φ to merge definitions from multi paths
- => Direct def-use chains


(a) Three-address code. (b) Static single-assignment form.

# Static Single-Assignment

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using φ to merge definitions from multi paths
- => Direct def-use chains


(a) Three-address code. (b) Static single-assignment form.

# Static Single-Assignment

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using φ to merge definitions from multi paths
- => Direct def-use chains

# Static Single-Assignment

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using φ to merge definitions from multi paths
- => Direct def-use chains

```
if ( flag ) x = -1; else x = 1;
y = x * a;
```

```
if ( flag ) x1 = -1; else x2 = 1;
x3 = φ (x1, x2);
y  = x3 *  a
```

# Static Single-Assignment

- X = 1; **while** (X < N) { **if** (P) X = X + 1; }

Try!

# Static Single-Assignment

- X = 1; **while** (X < N) { **if** (P) X = X + 1; }

```
     X = 1
H:   if X ≥ N goto E
     if ¬P goto H
     X = X + 1
     goto H
E:   ….
```

```
     X1 = 1
H:   X2 = φ (X1, X4)
     if X2 ≥ N goto E
     if ¬P goto L
     X3 = X2 + 1
L:   X4 = φ (X2, X3)
     goto H
E:   ….
```

# Gated SSA

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using $\gamma$ to merge definitions from multi-paths
- => Direct def-use chains + conditional def-use chains

# Gated SSA

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Using $\gamma$ to merge definitions from multi-paths
- => Direct def-use chains + conditional def-use chains

```
if ( flag ) x = -1; else x = 1;
y = x * a;
```

```
if ( flag ) x1 = -1; else x2 = 1;
x3 = γ (flag, x1, x2);
y = x3 * a
```

# Gated SSA

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Use recursive $\gamma$ to merge definitions from multi-paths
- => Direct def-use chains + conditional def-use chains

# Gated SSA

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Use recursive $\gamma$ to merge definitions from multi-paths
- => Direct def-use chains + conditional def-use chains

**if** ($flag_1$) { $x_1 = 1;$ }

**else if** ($flag_2$) { $x_2 = 2;$ }

**else** { $x_3 = 3;$ }

$x_4 = \gamma\ (flag_1, x_1, \gamma\ (flag_2, x_2, x_3));$

# Gated SSA

- **Feature 1**: Every variable has only one definition
- **Feature 2**: Use recursive $\gamma$ to merge definitions from multi-paths
- **Feature 3**: Use $\mu$ and $\eta$ for loop variables

# Gated SSA

- **Feature 3**: Using **μ** and **η** for loop variables
- X = 1; **while** (X < N) { **if** (P) X = X + 1; }

```
      X = 1
H:    if X ≥ N goto E
      if ¬P goto H
      X = X + 1
      goto H
E:    ….
```

SSA

```
      X1 = 1
H:    X2 = φ (X1, X4)
      if X2 ≥ N goto E
      if ¬P goto L
      X3 = X2 + 1
L:    X4 = φ (X2, X3)
      goto H
E:    ….
```

GSA

```
      X1 = 1
H:    X2 = μ (X1, X4)
      if X2 ≥ N goto E
      if ¬P goto L
      X3 = X2 + 1
L:    X4 = γ (P, X3, X2)
      goto H
E:    X5 = η (X2 ≥ N, X2)
```

# Research on SSA

- More powerful extension of SSA
- Program analysis via SSA
- Compiler optimization via SSA
- Theory of SSA

# Extensions of SSA

- SSA (1989)
- Gated SSA (1993)
- Hashed SSA (1996)
- SSI: Static Single Information (1999)
- IPSSA (2003)

# Extensions of SSA

- SSA (1989)
- Gated SSA (1993)
- Hashed SSA (1996)
- SSI: Static Single Information (1999)
- IPSSA (2003)
- **SSU: Static Single Use (2019)**
  - **ACM SIGSOFT Distinguished Paper Award**

ACM SIGSOFT
Distinguished Paper Award
ICSE 2019

41st IEEE/ACM International Conference on Software Engineering

May 25 – 31, 2019 Montréal, Canada

*Presented to*

**Gang Fan**, Hong Kong University of Science and Technology
**Rongxin Wu**, Hong Kong University of Science and Technology
**Qingkai Shi**, Hong Kong University of Science and Technology
**Xiao Xiao**, Sourcebrella Inc.
**Jinguo Zhou**, Sourcebrella Inc.
**Charles Zhang**, Hong Kong University of Science and Technology

*For*

**SMOKE: Scalable Path-Sensitive Memory Leak Detection for Millions of Lines of Code**

Joanne M. Atlee, General Chair
Tevfik Bultan, Program Co-Chair
Jon Whittle, Program Co-Chair

# SSA Complexity

- Intuitively, code in SSA is much larger than the non-SSA version

# SSA Complexity

- Intuitively, code in SSA is much larger than the non-SSA version
- X = 1; **while** (X < N) { **if** (P) X = X + 1; }

```
      X = 1
H:    if X ≥ N goto E
      if ¬P goto H
      X = X + 1
      goto H
E:    ….
```

```
      X1 = 1
H:    X2 = φ (X1, X4)
      if X2 ≥ N goto E
      if ¬P goto L
      X3 = X2 + 1
L:    X4 = φ (X2, X3)
      goto H
E:    ….
```

# SSA Complexity

- Space
  - Common three-address code vs. SSA code
  - **Almost Linear!**
- Time
  - From common three-address code to SSA code
  - **Almost Linear!**
- **How? What's the Algorithm?**

# SSA Complexity

- Space
  - Common three-address code vs. SSA code
  - **Almost Linear!**
- Time
  - From common three-address code to SSA code
  - **Almost Linear!**
- **How? What's the Algorithm?**

# Compilers using SSA

SSA form is a relatively recent development in the compiler community. As such, many older compilers only use SSA form for some part of the compilation or optimization process, but most do not rely on it. Examples of compilers that rely heavily on SSA form include:

- The ETH Oberon–2 compiler was one of the first public projects to incorporate "GSA", a variant of SSA.
- The LLVM Compiler Infrastructure uses SSA form for all scalar register values (everything except memory) in its primary code representation. SSA form is only eliminated once register allocation occurs, late in the compile process (often at link time).
- The Open64 compiler uses SSA form in its global scalar optimizer, though the code is brought into SSA form before and taken out of SSA form afterwards. Open64 uses extensions to SSA form to represent memory in SSA form as well as scalar values.
- Since the version 4 (released in April 2005) GCC, the GNU Compiler Collection, makes extensive use of SSA. The frontends generate "GENERIC" code that is then converted into "GIMPLE" code by the "gimplifier". High–level optimizations are then applied on the SSA form of "GIMPLE". The resulting optimized intermediate code is then translated into RTL, on which low–level optimizations are applied. The architecture–specific backends finally turn RTL into assembly language.
- IBM's open source adaptive Java virtual machine, Jikes RVM, uses extended Array SSA, an extension of SSA that allows analysis of scalars, arrays, and object fields in a unified framework. Extended Array SSA analysis is only enabled at the maximum optimization level, which is applied to the most frequently executed portions of code.
- In 2002, researchers modified IBM's JikesRVM (named Jalapeño at the time) to run both standard Java bytecode and a typesafe SSA (SafeTSA) bytecode class files, and demonstrated significant performance benefits to using the SSA bytecode.
- Oracle's HotSpot Java Virtual Machine uses an SSA–based intermediate language in its JIT compiler.[18]
- Microsoft Visual C++ compiler backend available in Microsoft Visual Studio 2015 Update 3 uses SSA[19]
- Mono uses SSA in its JIT compiler called Mini.
- jackcc is an open–source compiler for the academic instruction set Jackal 3.0. It uses a simple 3–operand code with SSA for its intermediate representation. As an interesting variant, it replaces Φ functions with a so–called SAME instruction, which instructs the register allocator to place the two live ranges into the same physical register.
- Although not a compiler, the Boomerang decompiler uses SSA form in its internal representation. SSA is used to simplify expression propagation, identifying parameters and returns, preservation analysis, and more.

- **GCC**
- **CLANG**
- **RustC**
- **…**


# LLVM and Clang https://llvm.org/

- From the beginning of this century, Jan 23rd → version 18.x
- LLVM Compiler Infrastructure → Clang/Clang++/….

# LLVM and Clang https://llvm.org/

- From the beginning of this century, Jan 23rd → version 18.x
- LLVM Compiler Infrastructure → Clang/Clang++/….

- Core: LLVM IR (Bitcode) and Utilities
- Static Single Assignment (SSA) style IR

# LLVM and Clang https://llvm.org/

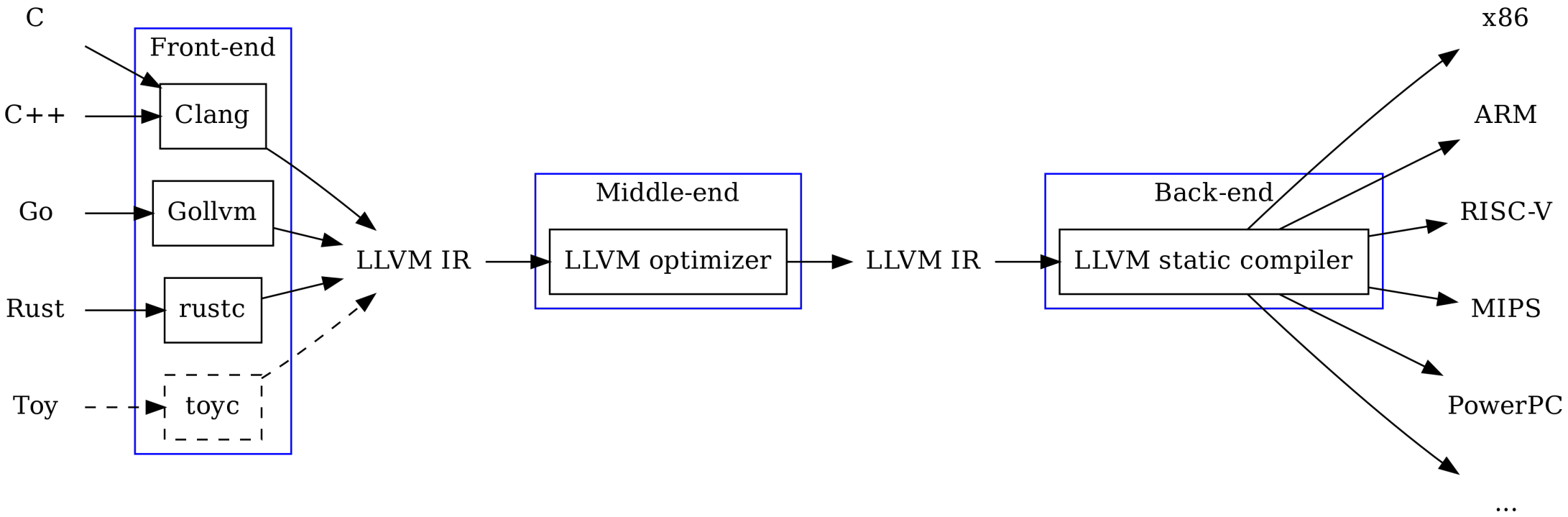
- From the beginning of this century, Jan 23rd → version 18.x
- LLVM Compiler Infrastructure → Clang/Clang++/….

- Core: LLVM IR (Bitcode) and Utilities
- Static Single Assignment (SSA) style IR


- Analysis, Optimization, Testing, Debugging, ……

# LLVM and Clang https://llvm.org/

- LLVM defines a set of APIs to manipulate the LLVM IR



# LLVM IR / Bitcode

https://llvm.org/docs/LangRef.html

- clang –c –emit-llvm hello.c [–o hello.bc]/[-S –o hello.ll]

# LLVM IR / Bitcode

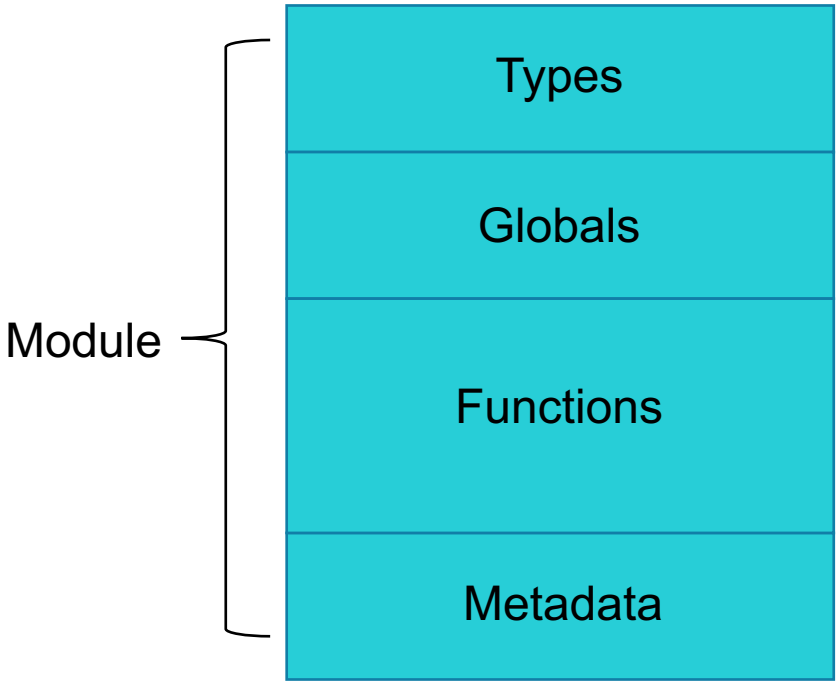
https://llvm.org/docs/LangRef.html

- clang –c –emit-llvm hello.c [–o hello.bc]/[-S –o hello.ll]
- llvm-dis hello.bc –o hello.ll　　llvm-as hello.ll –o hello.bc

# LLVM IR / Bitcode

https://llvm.org/docs/LangRef.html

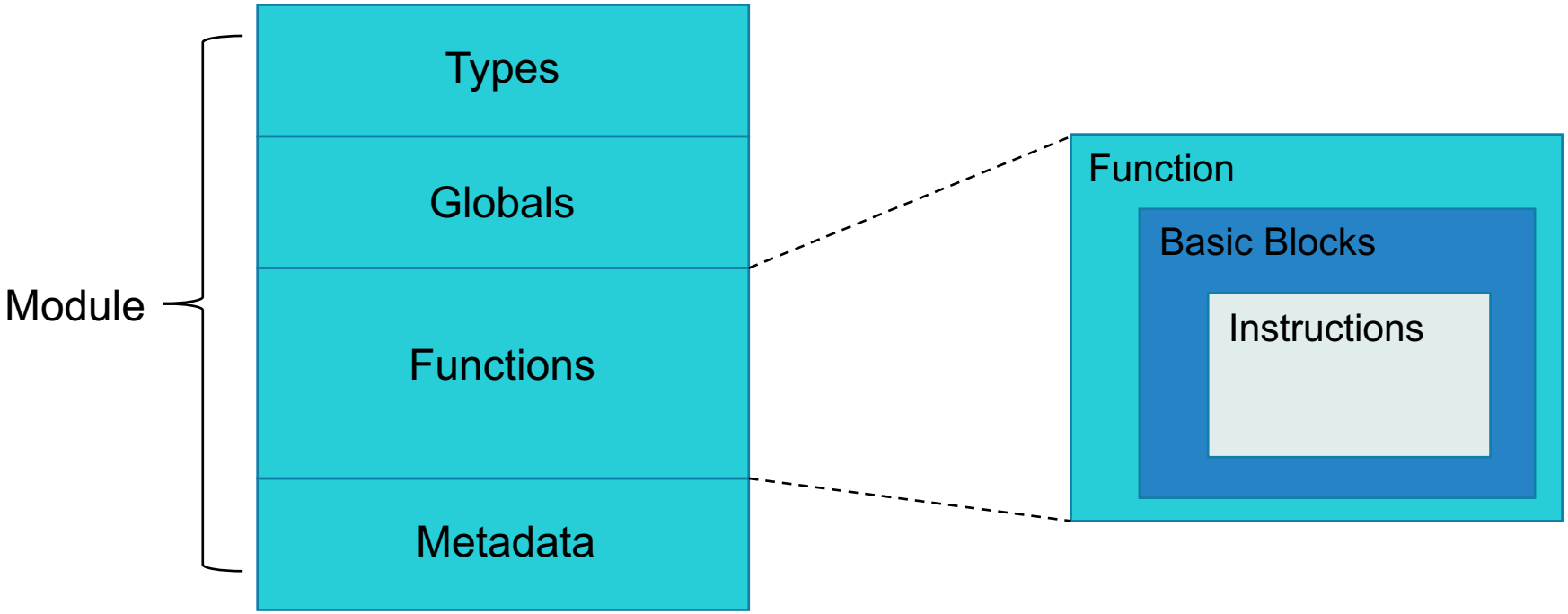
- clang –c –emit-llvm hello.c [–o hello.bc]/[-S –o hello.ll]
- llvm-dis hello.bc –o hello.ll　　llvm-as hello.ll –o hello.bc



# LLVM IR / Bitcode

https://llvm.org/docs/LangRef.html

- clang –c –emit-llvm hello.c [–o hello.bc]/[-S –o hello.ll]
- llvm-dis hello.bc –o hello.ll　　llvm-as hello.ll –o hello.bc



# LLVM Function

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
    return val * factorial(val - 1);
}
```

# LLVM Function

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
    return val * factorial(val - 1);
}
```

```
define i32 @factorial(i32 %0)  {
1:
  %2 = icmp eq i32 %0, 0
  br i1 %2, label %7, label %3

3:
  %4 = add i32 %0, -1
  %5 = call i32 @factorial(i32 %4)
  %6 = mul i32 %5, %0
  br label %7

7:
  %8 = phi i32 [ %6, %3 ], [ 1, %1 ]
  ret i32 %8
}
```

# LLVM Function

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
    return val * factorial(val - 1);
}
```

```
define i32 @factorial(i32 %0) {
1:
  %2 = icmp eq i32 %0, 0
  br i1 %2, label %7, label %3

3:
  %4 = add i32 %0, -1
  %5 = call i32 @factorial(i32 %4)
  %6 = mul i32 %5, %0
  br label %7

7:
  %8 = phi i32 [ %6, %3 ], [ 1, %1 ]
  ret i32 %8
}
```

# LLVM Function

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
    return val * factorial(val - 1);
}
```

```
define i32 @factorial(i32 %0)  {
1:
  %2 = icmp eq i32 %0, 0
  br i1 %2, label %7, label %3

3:
  %4 = add i32 %0, -1
  %5 = call i32 @factorial(i32 %4)
  %6 = mul i32 %5, %0
  br label %7

7:
  %8 = phi i32 [ %6, %3 ], [ 1, %1 ]
  ret i32 %8
}
```

# LLVM Function

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
    return val * factorial(val - 1);
}
```

```
define i32 @factorial(i32 %0)  {
1:
  %2 = icmp eq i32 %0, 0
  br i1 %2, label %7, label %3

3:
  %4 = add i32 %0, -1
  %5 = call i32 @factorial(i32 %4)
  %6 = mul i32 %5, %0
  br label %7

7:
  %8 = phi i32 [ %6, %3 ], [ 1, %1 ]
  ret i32 %8
}
```

# LLVM Function

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
    return val * factorial(val - 1);
}
```

```
define i32 @factorial(i32 %0)  {
1:
  %2 = icmp eq i32 %0, 0
  br i1 %2, label %7, label %3

3:
  %4 = add i32 %0, -1
  %5 = call i32 @factorial(i32 %4)
  %6 = mul i32 %5, %0
  br label %7

7:
  %8 = phi i32 [ %6, %3 ], [ 1, %1 ]
  ret i32 %8
}
```

# LLVM Metadata

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
    return val * factorial(val - 1);
}
```

```
define i32 @factorial(i32 %0) !dbg !7 {
1:
  %2 = icmp eq i32 %0, 0, !dbg !14
  br i1 %2, label %7, label %3, !dbg !16

3:
  %4 = add i32 %0, -1, !dbg !17
  %5 = call i32 @factorial(i32 %4), !dbg !18
  %6 = mul i32 %5, %0, !dbg !19
  br label %7, !dbg !20

7:
  %8 = phi i32 [ %6, %3 ], [ 1, %1 ], !dbg !13
  ret i32 %8, !dbg !21
}
```

```
clang –g –c –emit-llvm hello.c -S –o hello.ll
```

# LLVM Metadata

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
```

```
define i32 @factorial(i32 %0) !dbg !7 {
1:
  %2 = icmp eq i32 %0, 0, !dbg !14
```

```
!0 = distinct !DICompileUnit(language: DW_LANG_C99, file: !1, producer: "Ubuntu clang v
d: FullDebug, enums: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "hello.c", directory: "/home/qingkaishi")
!2 = !{}
!3 = !{i32 7, !"Dwarf Version", i32 4}
!4 = !{i32 2, !"Debug Info Version", i32 3}
!5 = !{i32 1, !"wchar_size", i32 4}
!6 = !{!"Ubuntu clang version 12.0.1-19ubuntu3"}
!7 = distinct !DISubprogram(name: "factorial", scope: !1, file: !1, line: 3, type: !8,
ISPFlagDefinition | DISPFlagOptimized, unit: !0, retainedNodes: !11)
!8 = !DISubroutineType(types: !9)
!9 = !{!10, !10}
!10 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_signed)
!11 = !{!12}
!12 = !DILocalVariable(name: "val", arg: 1, scope: !7, file: !1, line: 3, type: !10)
!13 = !DILocation(line: 0, scope: !7)
!14 = !DILocation(line: 4, column: 13, scope: !15)
!15 = distinct !DILexicalBlock(scope: !7, file: !1, line: 4, column: 9)
!16 = !DILocation(line: 4, column: 9, scope: !7)
```

dbg !16

4), !dbg !18

9

1 ], !dbg !13

S –o hello.ll

# LLVM Metadata

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
```

```
define i32 @factorial(i32 %0) !dbg !7 {
1:
  %2 = icmp eq i32 %0, 0, !dbg !14
```

```
!0 = distinct !DICompileUnit(language: DW_LANG_C99, file: !1, producer: "Ubuntu clang v
d: FullDebug, enums: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "hello.c", directory: "/home/qingkaishi")
!2 = !{}
!3 = !{i32 7, !"Dwarf Version", i32 4}
!4 = !{i32 2, !"Debug Info Version", i32 3}
!5 = !{i32 1, !"wchar_size", i32 4}
!6 = !{!"Ubuntu clang version 12.0.1-19ubuntu3"}
!7 = distinct !DISubprogram(name: "factorial", scope: !1, file: !1, line: 3, type: !8,
ISPFlagDefinition | DISPFlagOptimized, unit: !0, retainedNodes: !11)
!8 = !DISubroutineType(types: !9)
!9 = !{!10, !10}
!10 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_signed)
!11 = !{!12}
!12 = !DILocalVariable(name: "val", arg: 1, scope: !7, file: !1, line: 3, type: !10)
!13 = !DILocation(line: 0, scope: !7)
!14 = !DILocation(line: 4, column: 13, scope: !15)
!15 = distinct !DILexicalBlock(scope: !7, file: !1, line: 4, column: 9)
!16 = !DILocation(line: 4, column: 9, scope: !7)
```

```
dbg !16
4), !dbg !18
9
1 ], !dbg !13
```

```
S –o hello.ll
```

# LLVM Metadata

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
```

```
define i32 @factorial(i32 %0) !dbg !7 {
1:
  %2 = icmp eq i32 %0, 0, !dbg !14
```

```
!0 = distinct !DICompileUnit(language: DW_LANG_C99, file: !1, producer: "Ubuntu clang v
d: FullDebug, enums: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "hello.c", directory: "/home/qingkaishi")
!2 = !{}
!3 = !{i32 7, !"Dwarf Version", i32 4}
!4 = !{i32 2, !"Debug Info Version", i32 3}
!5 = !{i32 1, !"wchar_size", i32 4}
!6 = !{!"Ubuntu clang version 12.0.1-19ubuntu3"}
!7 = distinct !DISubprogram(name: "factorial", scope: !1, file: !1, line: 3, type: !8,
ISPFlagDefinition | DISPFlagOptimized, unit: !0, retainedNodes: !11)
!8 = !DISubroutineType(types: !9)
!9 = !{!10, !10}
!10 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_signed)
!11 = !{!12}
!12 = !DILocalVariable(name: "val", arg: 1, scope: !7, file: !1, line: 3, type: !10)
!13 = !DILocation(line: 0, scope: !7)
!14 = !DILocation(line: 4, column: 13, scope: !15)
!15 = distinct !DILexicalBlock(scope: !7, file: !1, line: 4, column: 9)
!16 = !DILocation(line: 4, column: 9, scope: !7)
```

# LLVM Metadata

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
```

```
define i32 @factorial(i32 %0) !dbg !7 {
1:
  %2 = icmp eq i32 %0, 0, !dbg !14
```

```
!0 = distinct !DICompileUnit(language: DW_LANG_C99, file: !1, producer: "Ubuntu clang v
d: FullDebug, enums: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "hello.c", directory: "/home/qingkaishi")
!2 = !{}
!3 = !{i32 7, !"Dwarf Version", i32 4}
!4 = !{i32 2, !"Debug Info Version", i32 3}
!5 = !{i32 1, !"wchar_size", i32 4}
!6 = !{!"Ubuntu clang version 12.0.1-19ubuntu3"}
!7 = distinct !DISubprogram(name: "factorial", scope: !1, file: !1, line: 3, type: !8,
ISPFlagDefinition | DISPFlagOptimized, unit: !0, retainedNodes: !11)
!8 = !DISubroutineType(types: !9)
!9 = !{!10, !10}
!10 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_signed)
!11 = !{!12}
!12 = !DILocalVariable(name: "val", arg: 1, scope: !7, file: !1, line: 3, type: !10)
!13 = !DILocation(line: 0, scope: !7)
!14 = !DILocation(line: 4, column: 13, scope: !15)
!15 = distinct !DILexicalBlock(scope: !7, file: !1, line: 4, column: 9)
!16 = !DILocation(line: 4, column: 9, scope: !7)
```

# LLVM Metadata

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
```

```
define i32 @factorial(i32 %0) !dbg !7 {
1:
  %2 = icmp eq i32 %0, 0, !dbg !14
```

```
!0 = distinct !DICompileUnit(language: DW_LANG_C99, file: !1, producer: "Ubuntu clang v
d: FullDebug, enums: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "hello.c", directory: "/home/qingkaishi")
!2 = !{}
!3 = !{i32 7, !"Dwarf Version", i32 4}
!4 = !{i32 2, !"Debug Info Version", i32 3}
!5 = !{i32 1, !"wchar_size", i32 4}
!6 = !{!"Ubuntu clang version 12.0.1-19ubuntu3"}
!7 = distinct !DISubprogram(name: "factorial", scope: !1, file: !1, line: 3, type: !8,
ISPFlagDefinition | DISPFlagOptimized, unit: !0, retainedNodes: !11)
!8 = !DISubroutineType(types: !9)
!9 = !{!10, !10}
!10 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_signed)
!11 = !{!12}
!12 = !DILocalVariable(name: "val", arg: 1, scope: !7, file: !1, line: 3, type: !10)
!13 = !DILocation(line: 0, scope: !7)
!14 = !DILocation(line: 4, column: 13, scope: !15)
!15 = distinct !DILexicalBlock(scope: !7, file: !1, line: 4, column: 9)
!16 = !DILocation(line: 4, column: 9, scope: !7)
```

# LLVM Metadata

https://llvm.org/docs/LangRef.html

```
int factorial(int val) {
    if (val == 0)
        return 1;
```

```
define i32 @factorial(i32 %0) !dbg !7 {
1:
  %2 = icmp eq i32 %0, 0, !dbg !14
```

```
!0 = distinct !DICompileUnit(language: DW_LANG_C99, file: !1, producer: "Ubuntu clang v
d: FullDebug, enums: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "hello.c", directory: "/home/qingkaishi")
!2 = !{}
!3 = !{i32 7, !"Dwarf Version", i32 4}
!4 = !{i32 2, !"Debug Info Version", i32 3}
!5 = !{i32 1, !"wchar_size", i32 4}
!6 = !{!"Ubuntu clang version 12.0.1-19ubuntu3"}
!7 = distinct !DISubprogram(name: "factorial", scope: !1, file: !1, line: 3, type: !8,
ISPFlagDefinition | DISPFlagOptimized, unit: !0, retainedNodes: !11)
!8 = !DISubroutineType(types: !9)
!9 = !{!10, !10}
!10 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_signed)
!11 = !{!12}
!12 = !DILocalVariable(name: "val", arg: 1, scope: !7, file: !1, line: 3, type: !10)
!13 = !DILocation(line: 0, scope: !7)
!14 = !DILocation(line: 4, column: 13, scope: !15)
!15 = distinct !DILexicalBlock(scope: !7, file: !1, line: 4, column: 9)
!16 = !DILocation(line: 4, column: 9, scope: !7)
```

# LLVM Types

https://llvm.org/docs/LangRef.html

- Void Type
  - void
- Primitive Types
  - i32, i8, i1, …
  - float, double, …

# LLVM Types

https://llvm.org/docs/LangRef.html

- Void Type
  - void
- Primitive Types
  - i32, i8, i1, …
  - float, double, …
- Function Type
  - return-type (parameter types)
  - i32 (i32)

# LLVM Types

https://llvm.org/docs/LangRef.html

- Void Type
  - void
- Primitive Types
  - i32, i8, i1, …
  - float, double, …
- Function Type
  - return-type (parameter types)
  - i32 (i32)
- Pointer Type
  - i32*, [40 x i32]*
  - ptr
- Array Type
  - [40 x i32]
  - [30 x i32*]

# LLVM Types

https://llvm.org/docs/LangRef.html

- Void Type
  - void
- Primitive Types
  - i32, i8, i1, …
  - float, double, …
- Function Type
  - return-type (parameter types)
  - i32 (i32)
- Pointer Type
  - i32*, [40 x i32]*
  - ptr
- Array Type
  - [40 x i32]
  - [30 x i32*]
- Structure Type
  - { i32, [3 x i32], i8 }
  - …

# LLVM GEP

https://llvm.org/docs/LangRef.html

```
struct RT {
    char A;
    int B[10][20];
    char C;
};

struct ST {
    int X;
    double Y;
    struct RT Z;
};

int foo(struct ST *s) {
    return s[1].Z.B[5][13];
}
```

# LLVM GEP

https://llvm.org/docs/LangRef.html

```
struct RT {
    char A;
    int B[10][20];
    char C;
};

struct ST {
    int X;
    double Y;
    struct RT Z;
};

int foo(struct ST *s) {
    return s[1].Z.B[5][13];
}
```

```
%struct.RT = type { i8, [10 x [20 x i32]], i8 }
%struct.ST = type { i32, double, %struct.RT }
```

# LLVM GEP

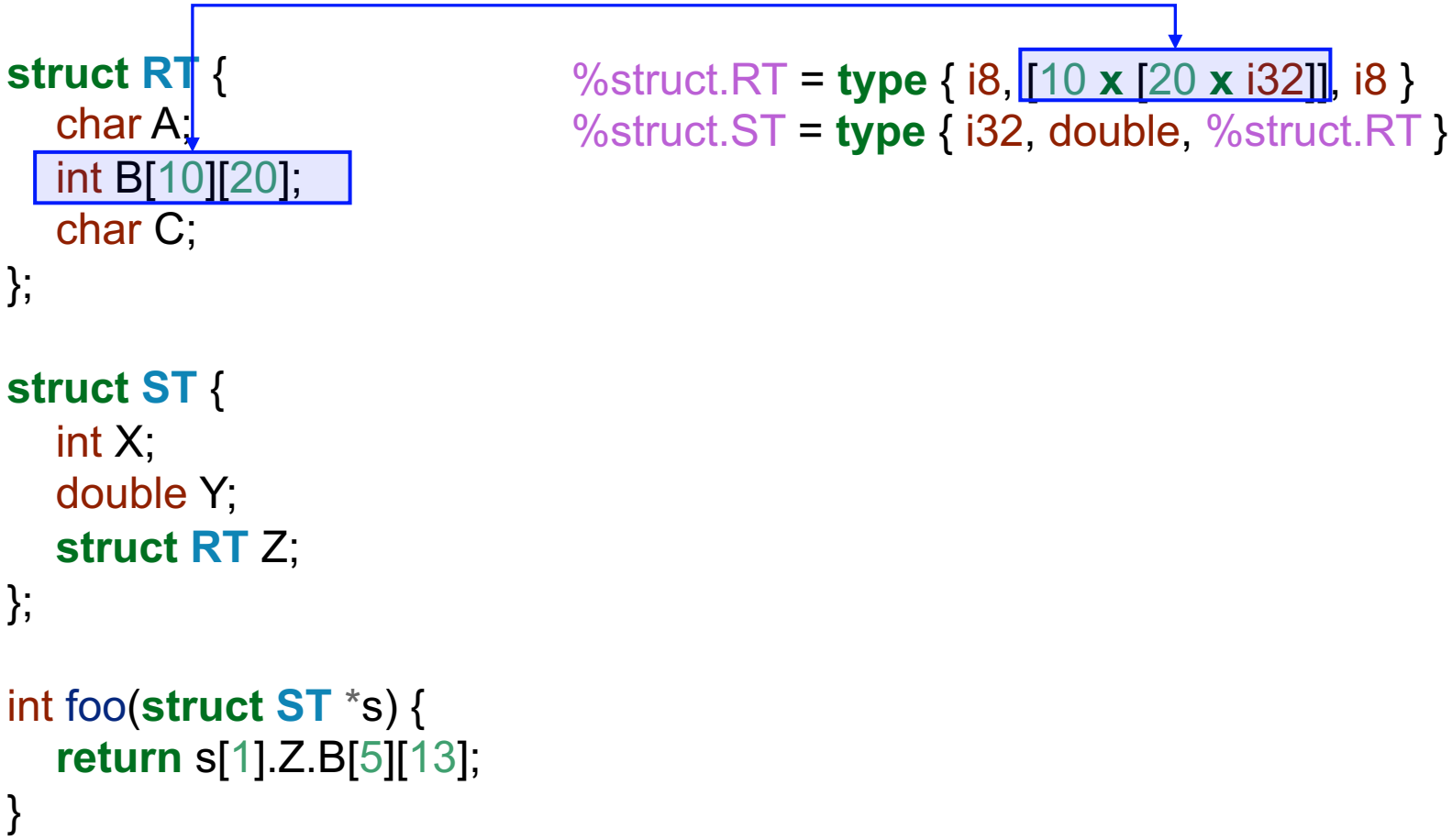
https://llvm.org/docs/LangRef.html

```
struct RT {
    char A;
    int B[10][20];
    char C;
};

struct ST {
    int X;
    double Y;
    struct RT Z;
};

int foo(struct ST *s) {
    return s[1].Z.B[5][13];
}
```

```
%struct.RT = type { i8, [10 x [20 x i32]], i8 }
%struct.ST = type { i32, double, %struct.RT }
```

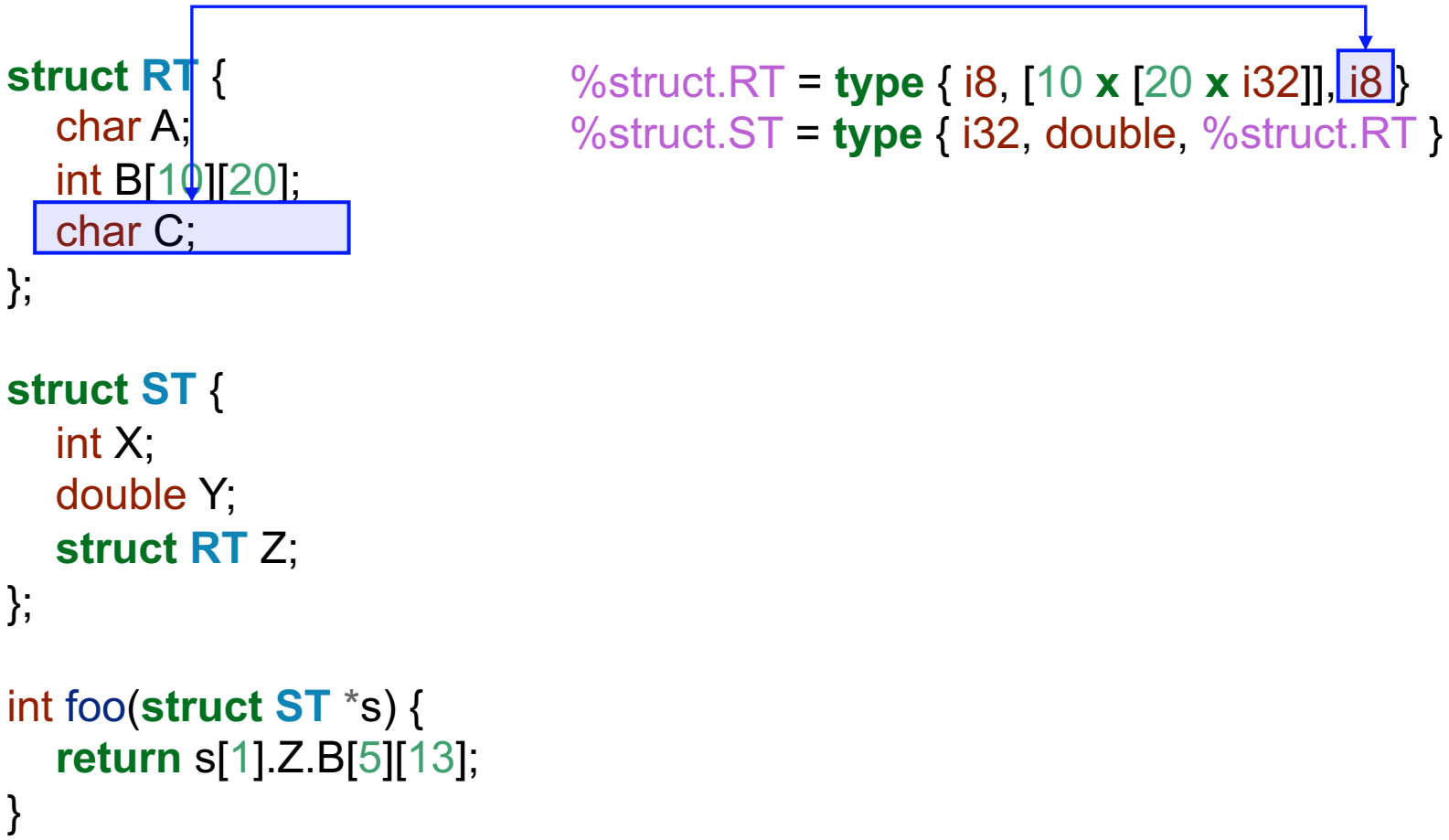
# LLVM GEP

https://llvm.org/docs/LangRef.html



# LLVM GEP

https://llvm.org/docs/LangRef.html



# LLVM GEP

https://llvm.org/docs/LangRef.html

```
struct RT {
    char A;
    int B[10][20];
    char C;
};

struct ST {
    int X;
    double Y;
    struct RT Z;
};

int foo(struct ST *s) {
    return s[1].Z.B[5][13];
}
```

```
%struct.RT = type { i8, [10 x [20 x i32]], i8 }
%struct.ST = type { i32, double, %struct.RT }
```

# LLVM GEP

https://llvm.org/docs/LangRef.html

```
struct RT {
    char A;
    int B[10][20];
    char C;
};

struct ST {
    int X;
    double Y;
    struct RT Z;
};

int foo(struct ST *s) {
    return s[1].Z.B[5][13];
}
```

```
%struct.RT = type { i8, [10 x [20 x i32]], i8 }
%struct.ST = type { i32, double, %struct.RT }

define i32 @foo(ptr %s) {



}
```

# LLVM GEP

https://llvm.org/docs/LangRef.html

```
struct RT {
    char A;
    int B[10][20];
    char C;
};

struct ST {
    int X;
    double Y;
    struct RT Z;
};

int foo(struct ST *s) {
    return s[1].Z.B[5][13];
}
```

```
%struct.RT = type { i8, [10 x [20 x i32]], i8 }
%struct.ST = type { i32, double, %struct.RT }

define i32 @foo(ptr %s) {
    %0 = getelementptr %struct.ST, ptr %s, i64 1, i32 2, i32 1, i64 5, i64 13
    %1 = load i32, ptr %0
    ret i32 %1
}
```

# LLVM Intrinsics

- Special functions, working as extension of instructions
- These functions have well known names and semantics

# LLVM Intrinsics

- Special functions, working as extension of instructions
- These functions have well known names and semantics

Standard C/C++ Library Intrinsics

- 'llvm.abs.*' Intrinsic
- 'llvm.smax.*' Intrinsic
- 'llvm.smin.*' Intrinsic
- 'llvm.umax.*' Intrinsic
- 'llvm.umin.*' Intrinsic

# LLVM Intrinsics

- Special functions, working as extension of instructions
- These functions have well known names and semantics

Standard C/C++ Library Intr

- ‘llvm.abs.*’ Intrinsic
- ‘llvm.smax.*’ Intrinsic
- ‘llvm.smin.*’ Intrinsic
- ‘llvm.umax.*’ Intrinsic
- ‘llvm.umin.*’ Intrinsic

Bit Manipulation Intrinsics

- ‘llvm.bitreverse.*’ Intrinsics
- ‘llvm.bswap.*’ Intrinsics
- ‘llvm.ctpop.*’ Intrinsic
- ‘llvm.ctlz.*’ Intrinsic
- ‘llvm.cttz.*’ Intrinsic
- ‘llvm.fshl.*’ Intrinsic
- ‘llvm.fshr.*’ Intrinsic

# LLVM Intrinsics

- Special functions, working as extension of instructions
- These functions have well known names and semantics

```
void foo(void* src, void *dst, long size) {
    memcpy(dst, src, size);
}
```

```
; Function Attrs: nofree nounwind uwtable willreturn
define dso_local void @foo(i8* nocapture readonly %0, i8* nocapture %1, i32 %2) local_unnamed_addr #0 {
  %4 = sext i32 %2 to i64
  call void @llvm.memcpy.p0i8.p0i8.i64(i8* align 1 %1, i8* align 1 %0, i64 %4, i1 false)
  ret void
}
```

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

```
extern int x;
int y = 1;

void foo() {
    printf("x = %d; y = %d\n", x, y);
}
```

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

```
extern int x;
int y = 1;

void foo() {
    printf("x = %d; y = %d\n", x, y);
}
```

How many global variables in the LLVM IR?

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

```
extern int x;
int y = 1;

void foo() {
    printf("x = %d; y = %d\n", x, y);
}
```

How many global variables in the LLVM IR?

-- Explicit global variables

-- Constant string/structs/…

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

```
@x = external global i32
@y = global i32 1
@.str = private unnamed_addr constant [16 x i8] c"x = %d; y = %d\0A\00"
```

```
define void @foo() {
  %1 = load i32, i32* @x
  %2 = load i32, i32* @y
  %3 = call i32 (i8*, ...) @printf(@.str, i32 %1, i32 %2)
  ret void
}
```

```
extern int x;
int y = 1;

void foo() {
     printf("x = %d; y = %d\n", x, y);
}
```

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

```
@x = external global i32
@y = global i32 1
@.str = private unnamed_addr constant [16 x i8] c"x = %d; y = %d\0A\00"

define void @foo() {
  %1 = load i32, i32* @x
  %2 = load i32, i32* @y
  %3 = call i32 (i8*, ...) @printf(@.str, i32 %1, i32 %2)
  ret void
}
```

```
extern int x;
int y = 1;

void foo() {
     printf("x = %d; y = %d\n", x, y);
}
```

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

```
@x = external global i32
@y = global i32 1
@.str = private unnamed_addr constant [16 x i8] c"x = %d; y = %d\0A\00"

define void @foo() {
  %1 = load i32, i32* @x
  %2 = load i32, i32* @y
  %3 = call i32 (i8*, ...) @printf(@.str, i32 %1, i32 %2)
  ret void
}
```

```
extern int x;
int y = 1;

void foo() {
     printf("x = %d; y = %d\n", x, y);
}
```

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

```
@x = external global i32
@y = global i32 1
@.str = private     unnamed_addr     constant     [16 x i8]      c"x = %d; y = %d\0A\00"

define void @foo() {
  %1 = load i32, i32* @x
  %2 = load i32, i32* @y
  %3 = call i32 (i8*, ...) @printf(@.str, i32 %1, i32 %2)
  ret void
}
```

```
extern int x;
int y = 1;

void foo() {
     printf("x = %d; y = %d\n", x, y);
}
```

# LLVM Global Variables

https://llvm.org/docs/LangRef.html

- All global variables (@...) in LLVM IR are pointers!

```
@x = external global i32
@y = global i32 1
@.str = private unnamed_addr constant [16 x i8] c"x = %d; y = %d\0A\00"

define void @foo() {
  %1 = load i32, i32* @x
  %2 = load i32, i32* @y
  %3 = call i32 (i8*, ...) @printf(@.str, i32 %1, i32 %2)
  ret void
}
```

```
extern int x;
int y = 1;

void foo() {
     printf("x = %d; y = %d\n", x, y);
}
```

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

- ModulePass and FunctionPass

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

- ModulePass and FunctionPass

```
// PassA.h
class PassA : public ModulePass {
public:
    static char ID;


}
```

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

- ModulePass and FunctionPass

```
// PassA.h
class PassA : public ModulePass {
public:
    static char ID;
    PassA() : ModulePass(ID) {}

}
```

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

- ModulePass and FunctionPass

```
// PassA.h
class PassA : public ModulePass {
public:
    static char ID;
    PassA() : ModulePass(ID) {}
    bool runOnModule(Module& M) override;
}
```

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

- // PassA.cpp
- #include “PassA.h”

- **char** PassA::ID = 0;
- **static** RegisterPass<PassA> X(“pass-a”, ”…");

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

- // PassA.cpp
- #include “PassA.h”

- **char** PassA::ID = 0;
- **static** RegisterPass<PassA> X(“pass-a”, ”…");

- **bool** PassA::runOnModule(Module &M) {
-
-
- }

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

```
// PassA.cpp
#include “PassA.h”

char PassA::ID = 0;
static RegisterPass<PassA> X(“pass-a”, ”…");

bool PassA::runOnModule(Module &M) {
    for (auto &F : M) outs() << F.getName() << “\n”;

}
```

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

```
// PassA.cpp
#include “PassA.h”

char PassA::ID = 0;
static RegisterPass<PassA> X(“pass-a”, ”…");

bool PassA::runOnModule(Module &M) {
    for (auto &F : M) outs() << F.getName() << “\n”;
    return false;
}
```

# Write a Pass

https://llvm.org/docs/WritingAnLLVMPass.html

```
opt –load pass-a.so --pass-a hello.bc
```

```
// PassA.cpp
#include “PassA.h”

char PassA::ID = 0;
static RegisterPass<PassA> X(“pass-a”, ”…");

bool PassA::runOnModule(Module &M) {
    for (auto &F : M) outs() << F.getName() << “\n”;
    return false;
}
```

# Internal Passes

https://llvm.org/docs/Passes.html


LLVM Home | Documentation » User Guides » LLVM's Analysis and Transform Passes

## LLVM's Analysis and Transform Passes

# Internal Passes

https://llvm.org/docs/Passes.html


LLVM Home | Documentation » User Guides » LLVM's Analysis and Transform Passes

## LLVM's Analysis and Transform Passes

- Introduction
- Analysis Passes
  - **aa-eval**: Exhaustive Alias Analysis Precision Evaluator
  - **basic-aa**: Basic Alias Analysis (stateless AA impl)
  - **basiccg**: Basic CallGraph Construction
  - **da**: Dependence Analysis
  - **domfrontier**: Dominance Frontier Construction
  - **domtree**: Dominator Tree Construction
  - **dot-callgraph**: Print Call Graph to "dot" file
  - **dot-cfg**: Print CFG of function to "dot" file
  - **dot-cfg-only**: Print CFG of function to "dot" file (with no function bodies)
  - **dot-dom**: Print dominance tree of function to "dot" file
  - **dot-dom-only**: Print dominance tree of function to "dot" file (with no function bodies)
  - **dot-post-dom**: Print postdominance tree of function to "dot" file
  - **dot-post-dom-only**: Print postdominance tree of function to "dot" file (with no function bodies)

# Internal Passes

https://llvm.org/docs/Passes.html


LLVM Home | Documentation » User Guides » LLVM's Analysis and Transform Passes

## LLVM's Analysis and Transform Passes

- Introduction
- Analysis Passes
  - **aa-eval**: Exhaustive Alias Analysis Precision Evaluator
  - **basic-aa**: Basic Alias Analysis (stateless AA impl)
  - **basiccg**: Basic CallGraph Construction
  - **da**: Dependence Analysis
  - **domfrontier**: Dominance Frontier Construction
  - **domtree**: Dominator Tree Construction
  - **dot-callgraph**: Print Call Graph to "dot" file
  - **dot-cfg**: Print CFG of function to "dot" file
  - **dot-cfg-only**: Print CFG of function to "dot" file (with no function bodies)
  - **dot-dom**: Print dominance tree of function to "dot" file
  - **dot-dom-only**: Print dominance tree of function to "dot" file (with no function bodies)
  - **dot-post-dom**: Print postdominance tree of function to "dot" file
  - **dot-post-dom-only**: Print postdominance tree of function to "dot" file (with no function bodies)

```
opt    --dot-cfg    hello.bc
opt    --dot-callgraph    hello.bc
…
```

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

- An analysis/transform (LLVM Pass) may depend on the results of other analyses (LLVM Passes)

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

- An analysis/transform (LLVM Pass) may depend on the results of other analyses (LLVM Passes)

```
char get(char *buffer, int size, int index) {
    return buffer[index];
}
```

```
char get(char *buffer, int size, int index) {
    assert (index < size); // to avoid buffer overrun
    return buffer[index];
}
```

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

- An analysis/transform (LLVM Pass) may depend on the results of other analyses (LLVM Passes)

```
char get(char *buffer, int size, int index) {
    return buffer[index];
}
```

```
char get(char *buffer, int size, int index) {
    assert (index < size); // to avoid buffer overrun
    return buffer[index];
}
```

**Analysis Pass**: Find possible buffer overruns

**Transform Pass**: Insert assertion to avoid overrun

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

```
// PassA.h
class PassA : public ModulePass {
public:
    static char ID;
    PassA() : ModulePass(ID) {}
    bool runOnModule(Module& M) override;

    void getAnalysisUsage(AnalysisUsage &AU) override;
}
```

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

```
// PassA.h
class PassA : public ModulePass {
public:
    static char ID;
    PassA() : ModulePass(ID) {}
    bool runOnModule(Module& M) override;

    void getAnalysisUsage(AnalysisUsage &AU) override;
}
```

LLVM's pass manager pre-computes an execution order according to the dependency declared in this function!

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

```
// PassA.cpp
……
void PassA::getAnalysisUsage(AnalysisUsage& AU) {
    AU.setPreservesAll();
    AU.addRequired<PassB>();
}
```

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

```
// PassA.cpp
……
void PassA::getAnalysisUsage(AnalysisUsage& AU) {
    AU.setPreservesAll();
    AU.addRequired<PassB>();
}
```

```
class PassB : public ModulePass {
private:
   int NumFuncs;

public:
   int getNumFunctions() {  return NumFuncs;  }

   bool runOnModule(Module &M) {
         NumFuncs = M.size();  return false;
   }
}
```

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

```
// PassA.cpp
……
void PassA::getAnalysisUsage(AnalysisUsage& AU) {
    AU.setPreservesAll();
    AU.addRequired<PassB>();
}

bool PassA::runOnModule(Module &M) {
    PassB *PB = &getAnalysis<PassB>();
    outs() << PB->getNumFunctions() << "\n";
    return false;
}
```

```
class PassB : public ModulePass {
private:
   int NumFuncs;

public:
   int getNumFunctions() {  return NumFuncs;  }

   bool runOnModule(Module &M) {
         NumFuncs = M.size();  return false;
   }
}
```

# Pass Dependency

https://llvm.org/docs/WritingAnLLVMPass.html

```
// PassA.cpp
……
void PassA::getAnalysisUsage(AnalysisUsage& AU) {
    AU.setPreservesAll();  // or AU.addPreserved<PassB>(); for specific passes
    AU.addRequired<PassB>();
}

bool PassA::runOnModule(Module &M) {
    PassB *PB = &getAnalysis<PassB>();
    outs() << PB->getNumFunctions() << "\n";
    return false;
}
```

# LLVM API

- C/C++ API
- https://llvm.org/docs/ProgrammersManual.html
- From C/C++ API to Java API
- https://github.com/bytedeco/javacpp

- Read the document and send TA emails for questions

# PART II-1: Generation of IR

--- Generating IR for Variable Declarations

# Variable Declaration

```
• int main(){
•     int a; int b; int c;
•     scanf("%d%d", &a, &b);
•     c = a + b;
•     printf("%d", c);
•     return 0;
• }
```

Declaration → TypeExpression Variable;

**Generating IR for variable declaration:**

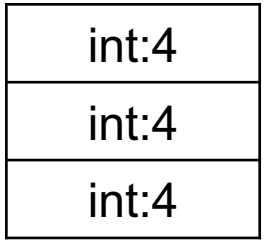
1. **Type of a variable!**
2. **Memory layout!**

# Type Expression

- Primary Types
  - int, char, float, …
- Composite Types
  - type expression [number]
    - int[3]
    - int[3][4]
  - record {list of declarations}
    - record {float x; int[3] y;}
    - record {float x; record {int a; int b;} y}

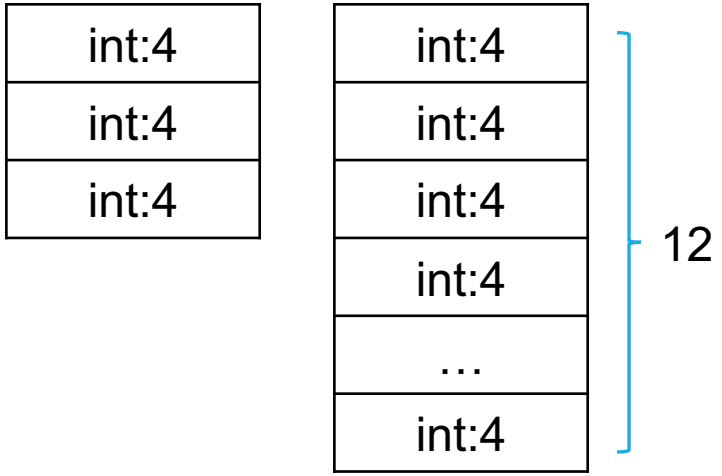
# Type Expression

- Primary Types
  - int, char, float, …
- Composite Types
  - type expression [number]
    - int[3]
    - int[3][4]
  - record {list of declarations}
    - record {float x; int[3] y;}
    - record {float x; record {int a; int b;} y}



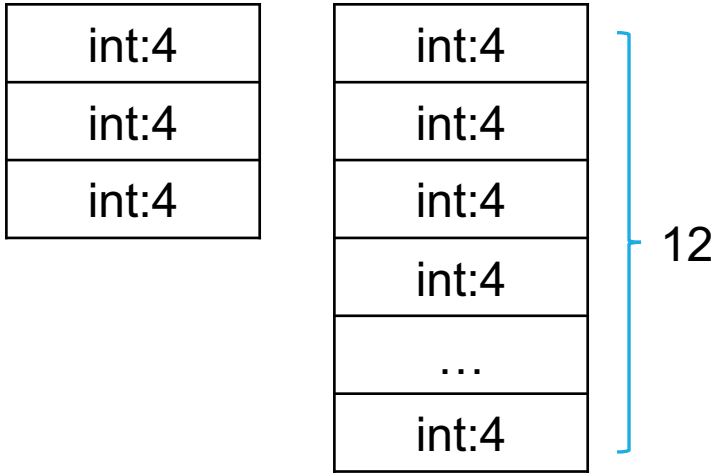
# Type Expression

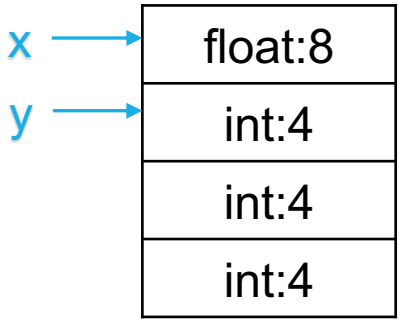
- Primary Types
  - int, char, float, …
- Composite Types
  - type expression [number]
    - int[3]
    - int[3][4]
  - record {list of declarations}
    - record {float x; int[3] y;}
    - record {float x; record {int a; int b;} y}



# Type Expression

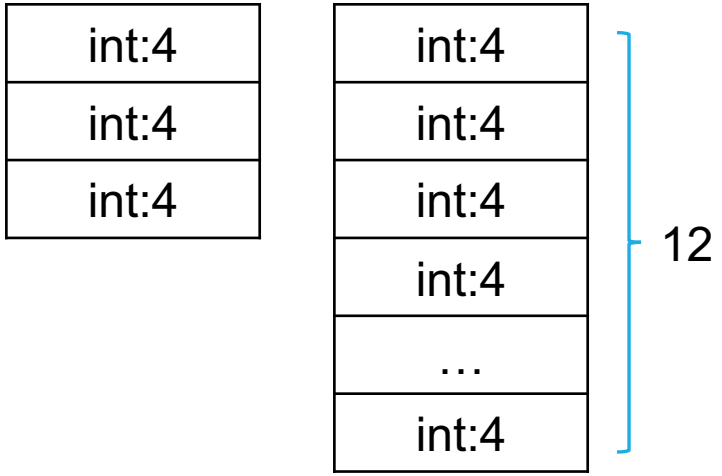
- Primary Types
  - int, char, float, …
- Composite Types
  - type expression [number]
    - int[3]
    - int[3][4]
  - record {list of declarations}
    - record {float x; int[3] y;}
    - record {float x; record {int a; int b;} y}

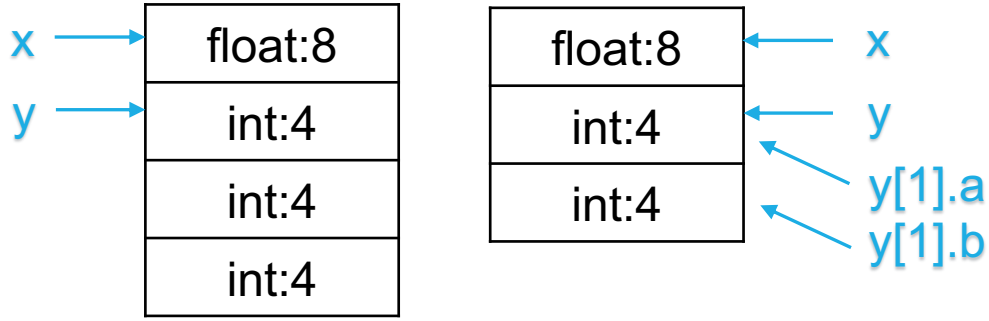


# Type Expression

- Primary Types
  - int, char, float, …
- Composite Types
  - type expression [number]
    - int[3]
    - int[3][4]
  - record {list of declarations}
    - record {float x; int[3] y;}
    - record {float x; record {int a; int b;} y}



# Variable Declaration

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

# Variable Declaration

Declaration List

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

# Variable Declaration

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

Type Expressions

# Variable Declaration

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

Type Expressions

# SDT for Type Expressions

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

# SDT for Type Expressions

- Compute the **type** as well as the **type width**

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

# SDT for Type Expressions

- Compute the **type** as well as the **type width**

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

# SDT for Type Expressions

- Compute the **type** as well as the **type width**

$$
\begin{array}{lcll}
T & \rightarrow & B & \{\ t = B.type;\ w = B.width;\ \} \\
 & & C & \{\ T.type = C.type;\ T.width = C.width;\ \} \\
B & \rightarrow & \textbf{int} & \{\ B.type = integer;\ B.width = 4;\ \} \\
B & \rightarrow & \textbf{float} & \{\ B.type = float;\ B.width = 8;\ \} \\
C & \rightarrow & \epsilon & \{\ C.type = t;\ C.width = w;\ \} \\
C & \rightarrow & \texttt{[}\ \textbf{num}\ \texttt{]}\ C_1 & \{\ C.type = array(\textbf{num}.value,\ C_1.type); \\
 & & & \quad C.width = \textbf{num}.value \times C_1.width;\ \}
\end{array}
$$

# SDT for Type Expressions

- Compute the **type** as well as the **type width**

| | | |
|---|---|---|
| $T \rightarrow B$ | | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B \rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B \rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C \rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C \rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |

# SDT for Type Expressions

- Compute the **type** as well as the **type width**

| | | |
|---|---|---|
| $T \rightarrow B$ | | $\{\ t = B.type;\ w = B.width;\ \}$ |
| $C$ | | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B \rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B \rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C \rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C \rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |

# SDT for Type Expressions

- Compute the **type** as well as the **type width**


| | | |
|---|---|---|
| $T \rightarrow B$ | | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B \rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B \rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C \rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C \rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |

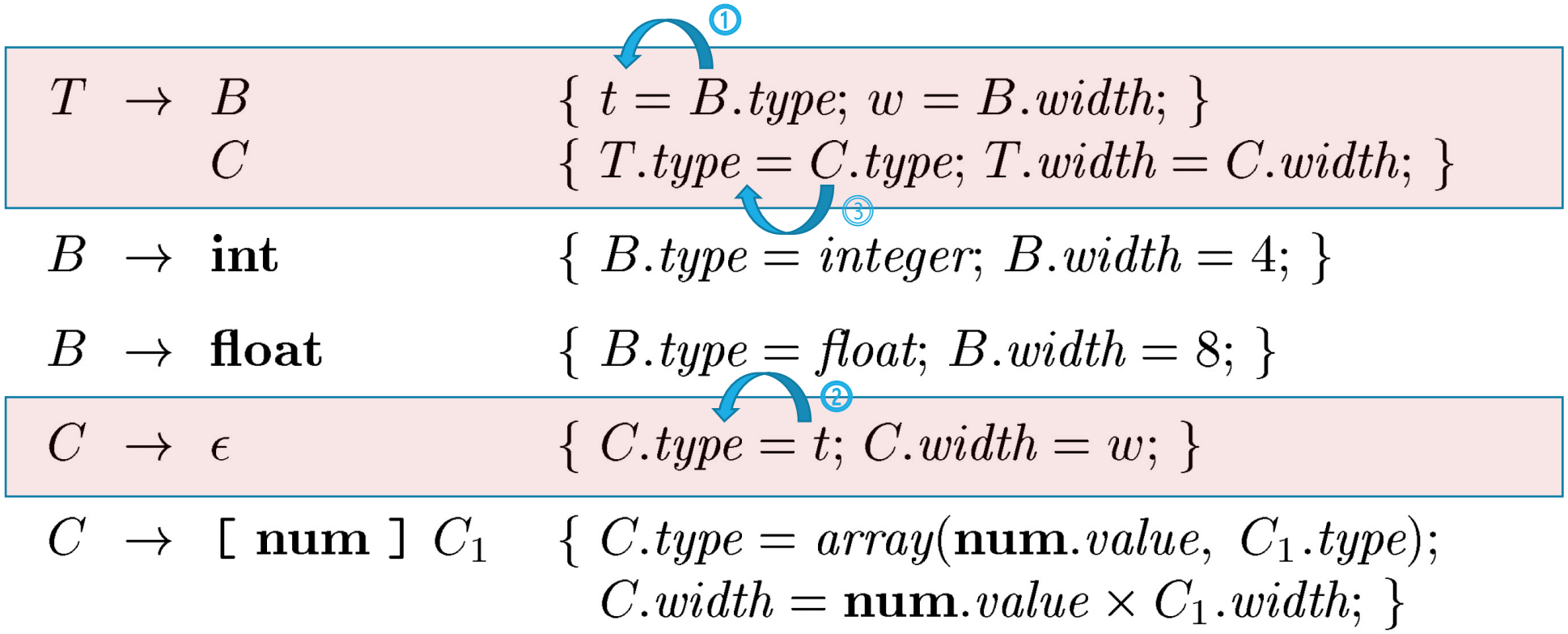
# SDT for Type Expressions

- Compute the **type** as well as the **type width**


| | | |
|---|---|---|
| $T$ | $\rightarrow B$ | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B$ | $\rightarrow$ **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B$ | $\rightarrow$ **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C$ | $\rightarrow \epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C$ | $\rightarrow$ [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |

# SDT for Type Expressions

- Compute the **type** as well as the **type width**




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

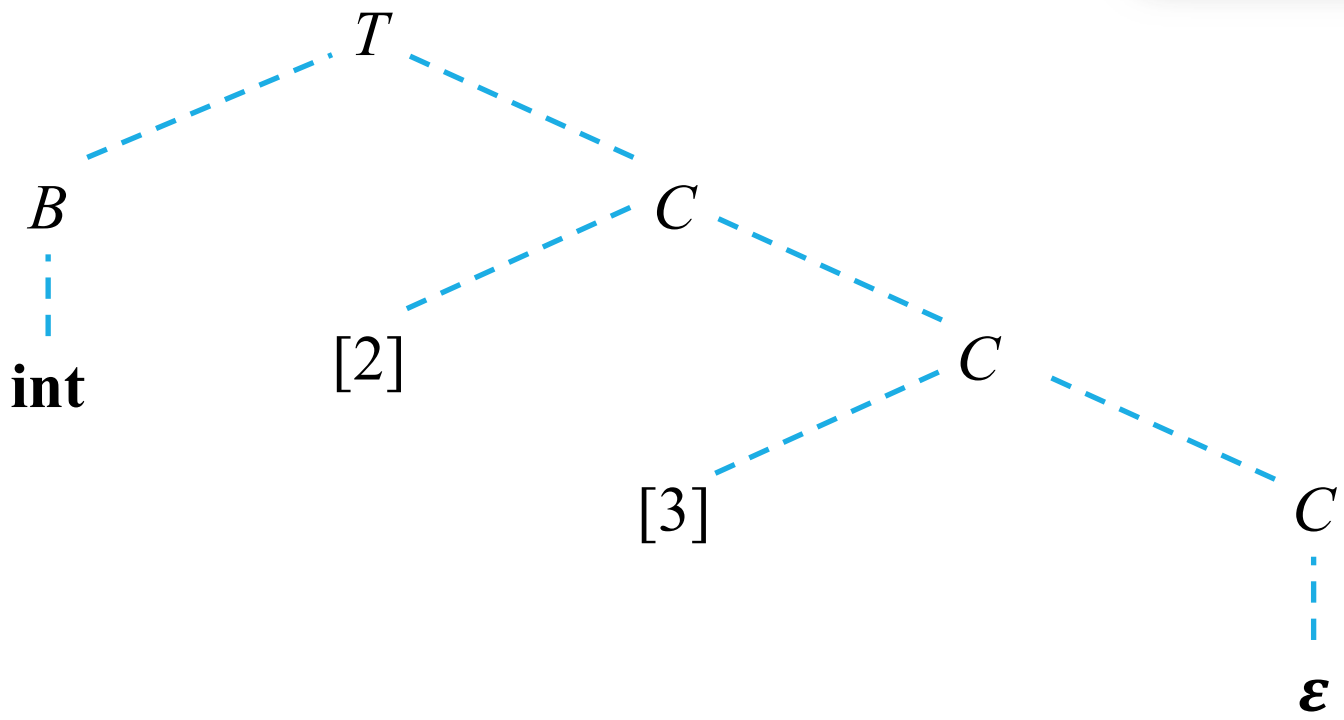
# SDT for Type Expressions

- Compute the **type** as well as the **type width**

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

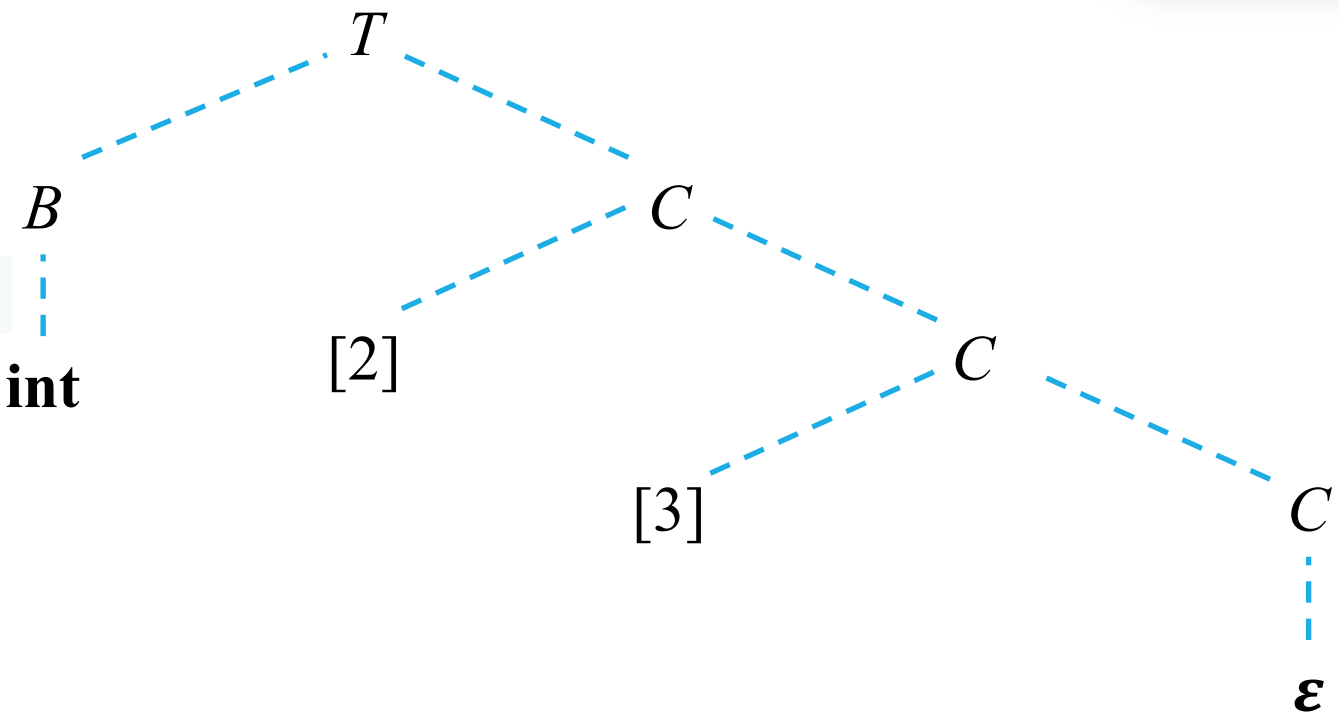
# Example: int[2][3]

| | | | |
|---|---|---|---|
| $T$ | $\rightarrow$ | $B$ | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B$ | $\rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B$ | $\rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C$ | $\rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C$ | $\rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |



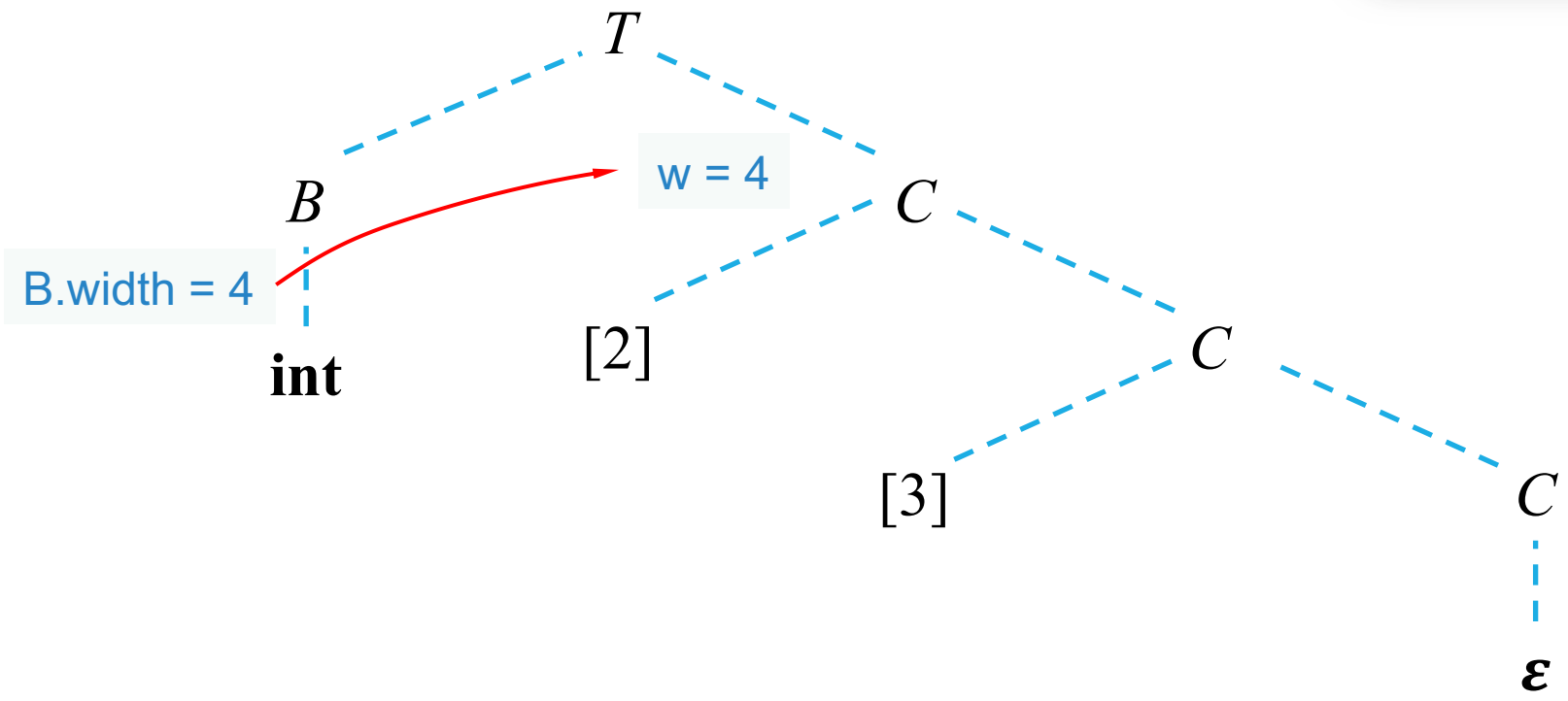
# Example: int[2][3]

| | | | |
|---|---|---|---|
| $T$ | $\rightarrow$ | $B$ | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B$ | $\rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B$ | $\rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C$ | $\rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C$ | $\rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |



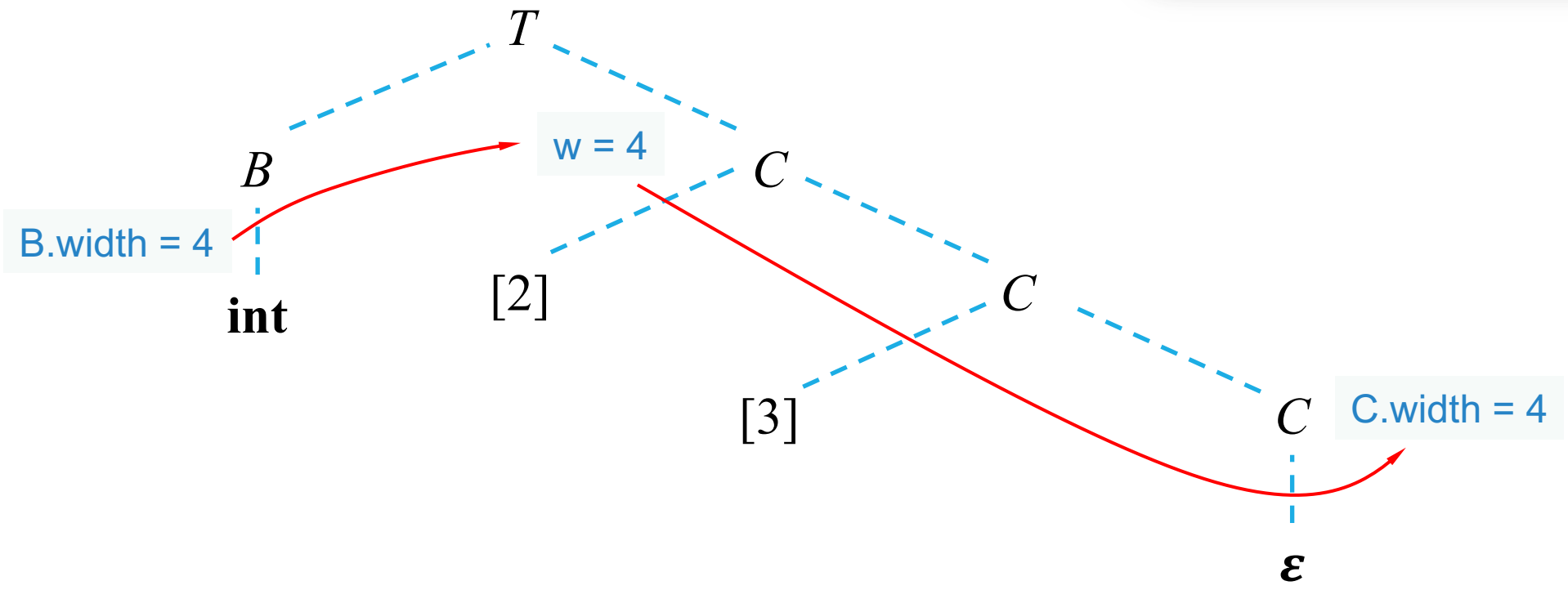
# Example: int[2][3]

| | | | |
|---|---|---|---|
| $T$ | $\rightarrow$ | $B$ | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B$ | $\rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B$ | $\rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C$ | $\rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C$ | $\rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |



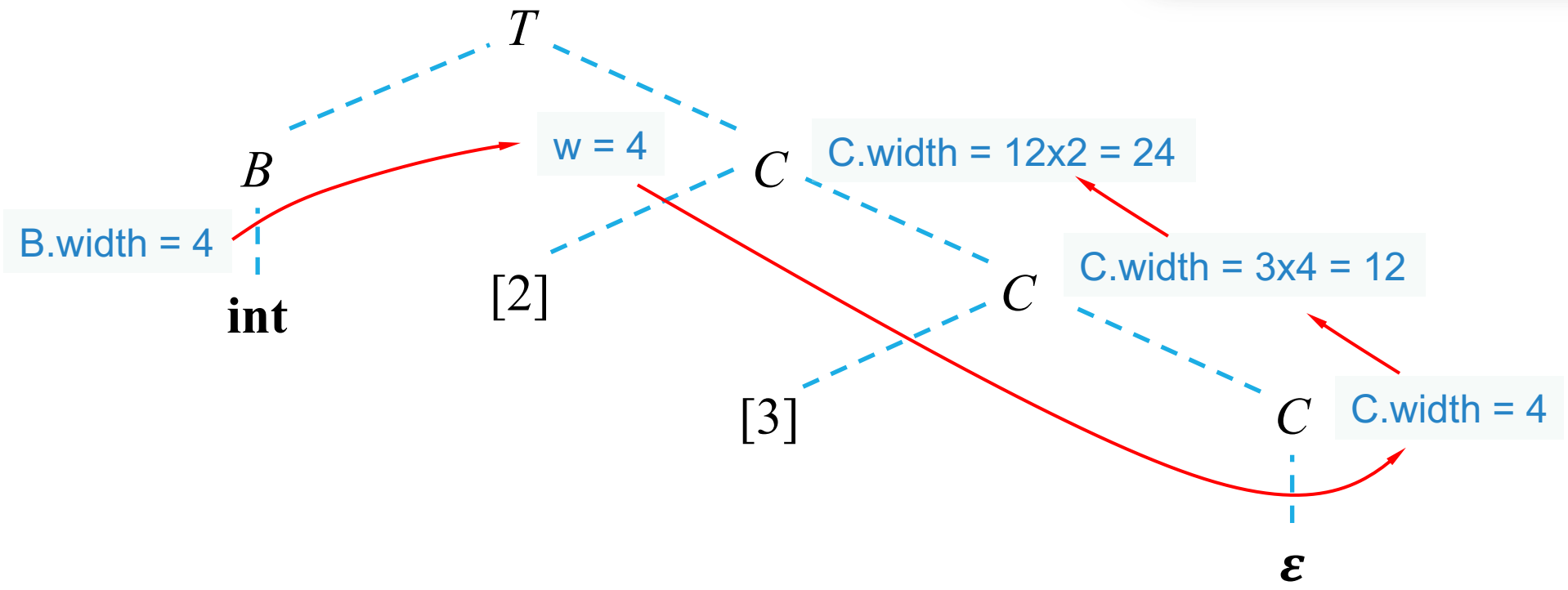
# Example: int[2][3]

| | | | |
|---|---|---|---|
| $T$ | $\rightarrow$ | $B$ | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B$ | $\rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B$ | $\rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C$ | $\rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C$ | $\rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |



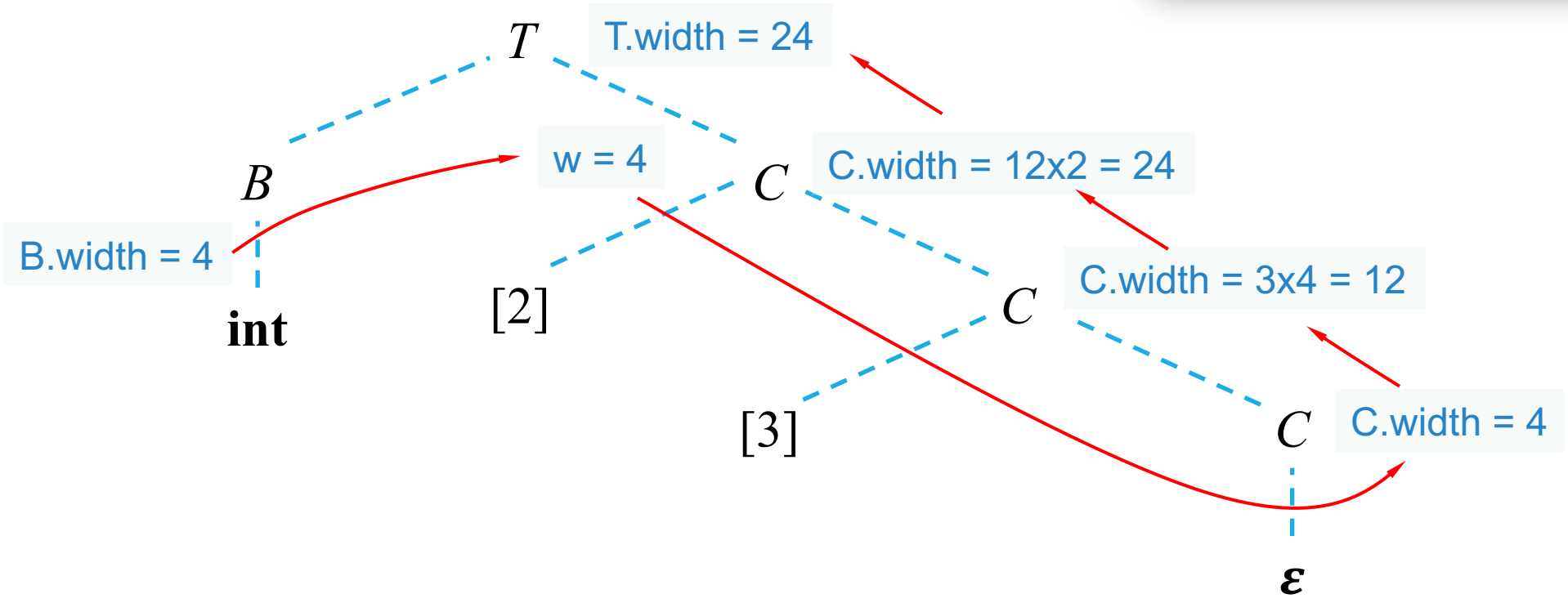
# Example: int[2][3]

| | | | |
|---|---|---|---|
| $T$ | $\rightarrow$ | $B$ | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B$ | $\rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B$ | $\rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C$ | $\rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C$ | $\rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |



# Example: int[2][3]

| | | | |
|---|---|---|---|
| $T$ | $\rightarrow$ | $B$ | $\{\ t = B.type;\ w = B.width;\ \}$ |
| | | $C$ | $\{\ T.type = C.type;\ T.width = C.width;\ \}$ |
| $B$ | $\rightarrow$ | **int** | $\{\ B.type = integer;\ B.width = 4;\ \}$ |
| $B$ | $\rightarrow$ | **float** | $\{\ B.type = float;\ B.width = 8;\ \}$ |
| $C$ | $\rightarrow$ | $\epsilon$ | $\{\ C.type = t;\ C.width = w;\ \}$ |
| $C$ | $\rightarrow$ | [ **num** ] $C_1$ | $\{\ C.type = array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$ |



# SDT for Variable Declarations

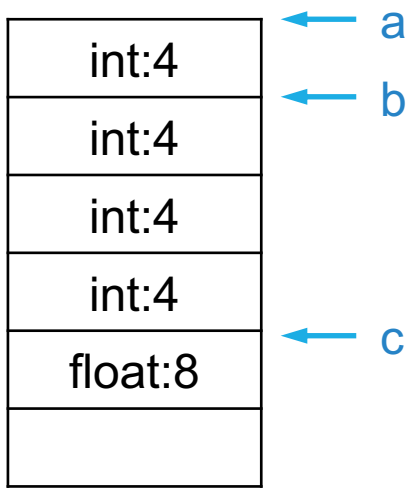
$$
\begin{array}{lcl}
D & \rightarrow & T\ \mathbf{id}\ ;\ D \mid \epsilon \\
T & \rightarrow & B\ C \mid \mathbf{record}\ '\{'\ D\ '\}' \\
B & \rightarrow & \mathbf{int} \mid \mathbf{float} \\
C & \rightarrow & \epsilon \mid [\ \mathbf{num}\ ]\ C
\end{array}
$$

# SDT for Variable Declarations

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

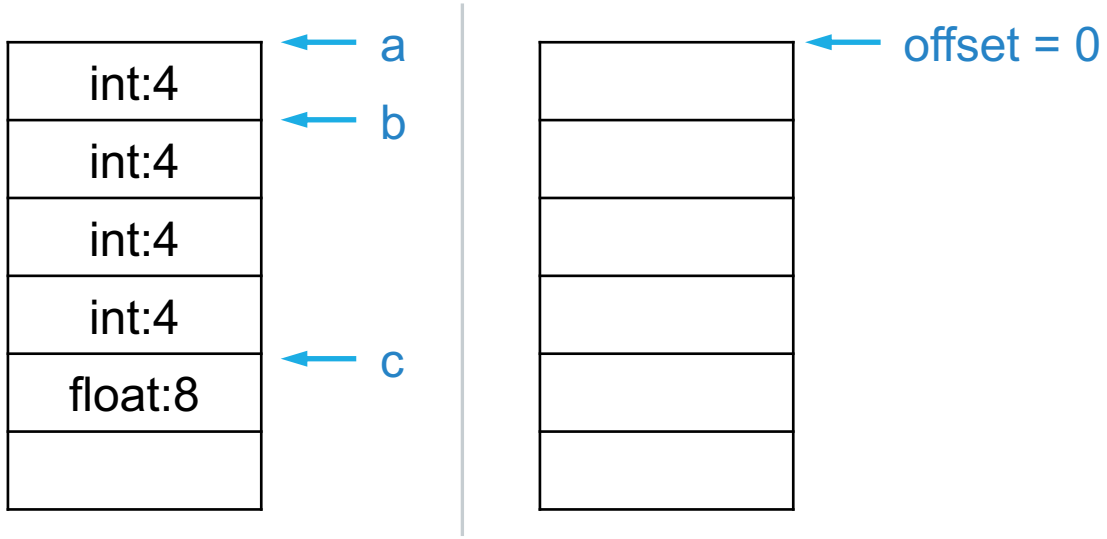
# SDT for Variable Declarations

- **int** a; **int**[3] b; **float** c;…



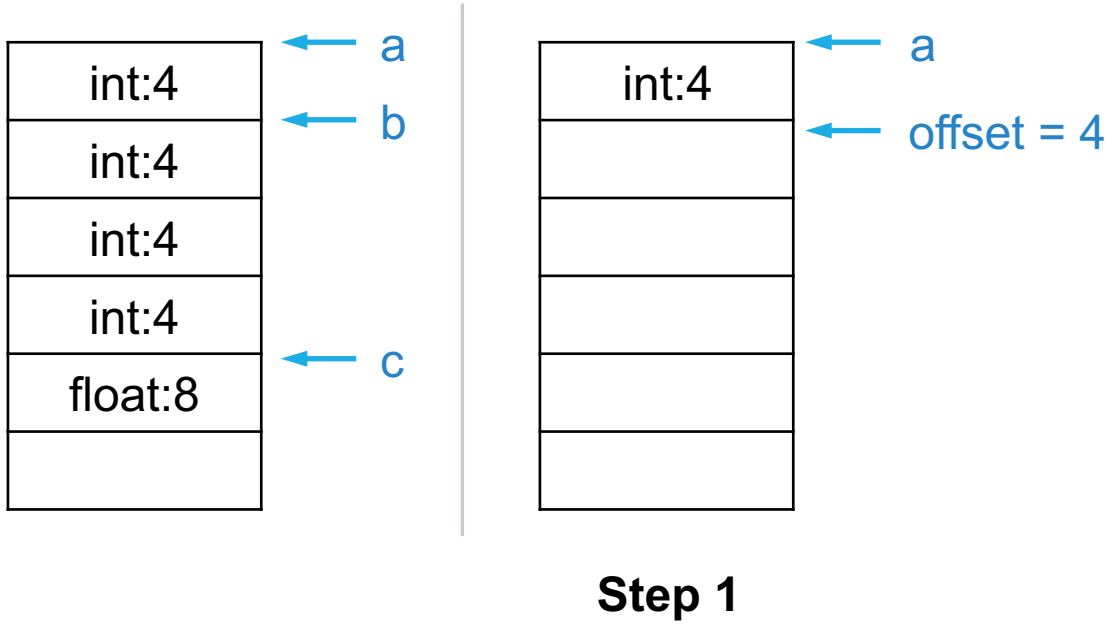
# SDT for Variable Declarations

- **int** a; **int**[3] b; **float** c;…



# SDT for Variable Declarations

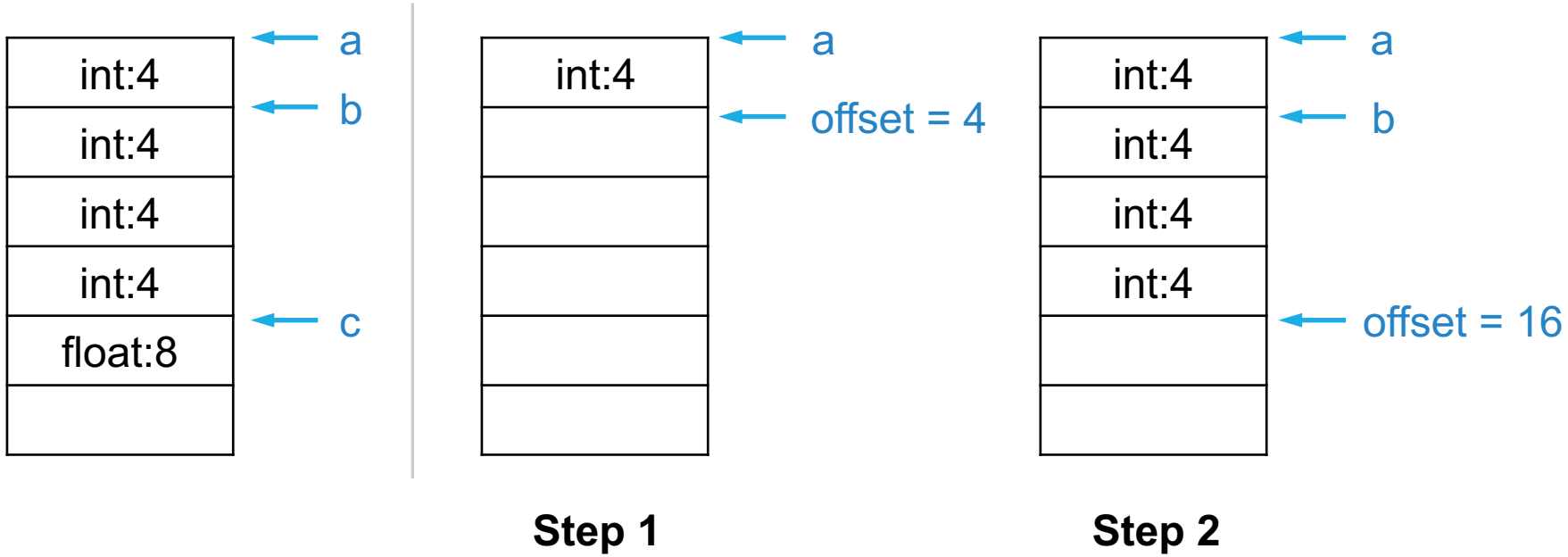
- **int** a; **int**[3] b; **float** c;…




**Step 1**

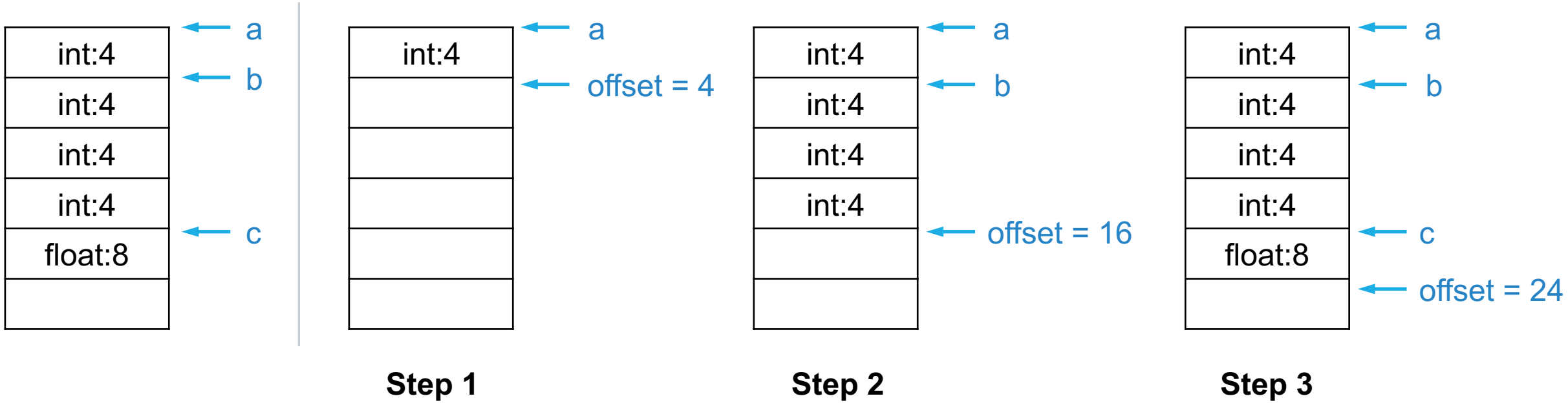
# SDT for Variable Declarations

- **int** a; **int**[3] b; **float** c;…



# SDT for Variable Declarations

- **int** a; **int**[3] b; **float** c;…



# SDT for Variable Declarations

$$\{\ offset = 0;\ \}$$

$$D \rightarrow T\ \mathbf{id}\ ; \quad \{\ top.put(\mathbf{id}.lexeme,\ T.type,\ offset);\ offset = offset + T.width;\ \}$$

$$D \rightarrow D_1$$

$$D \rightarrow \epsilon$$

# SDT for Variable Declarations

$$\{\ offset = 0;\ \}$$

$$D \rightarrow T\ \mathbf{id}\ ; \quad \{\ top.put(\mathbf{id}.lexeme,\ T.type,\ offset);\ offset = offset + T.width;\ \}$$

$$D \rightarrow D_1$$

$$D \rightarrow \epsilon$$

# SDT for Variable Declarations

$$\{\ offset = 0;\ \}$$

$$D \rightarrow T\ \mathbf{id}\ ; \quad \{\ top.put(\mathbf{id}.lexeme,\ T.type,\ offset);\ offset = offset + T.width;\ \}$$

$$D \rightarrow D_1$$

$$D \rightarrow \epsilon$$

# SDT for Record Type

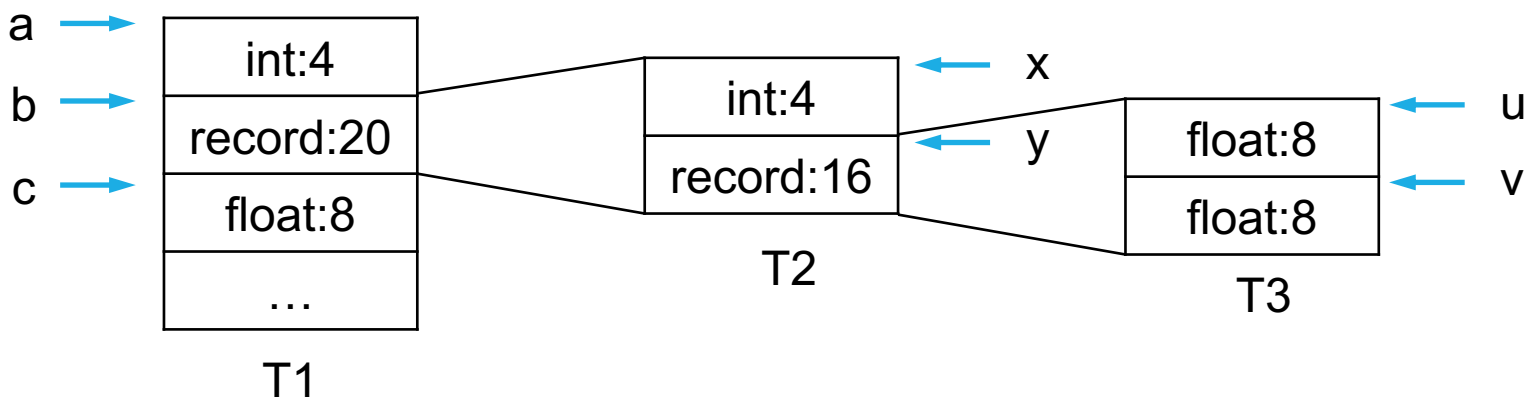
$$
\begin{array}{lcl}
D & \rightarrow & T\ \mathbf{id}\ ;\ D \mid \epsilon \\
T & \rightarrow & B\ C \mid \mathbf{record}\ '\{'\ D\ '\}' \\
B & \rightarrow & \mathbf{int} \mid \mathbf{float} \\
C & \rightarrow & \epsilon \mid [\ \mathbf{num}\ ]\ C
\end{array}
$$

# SDT for Record Type

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

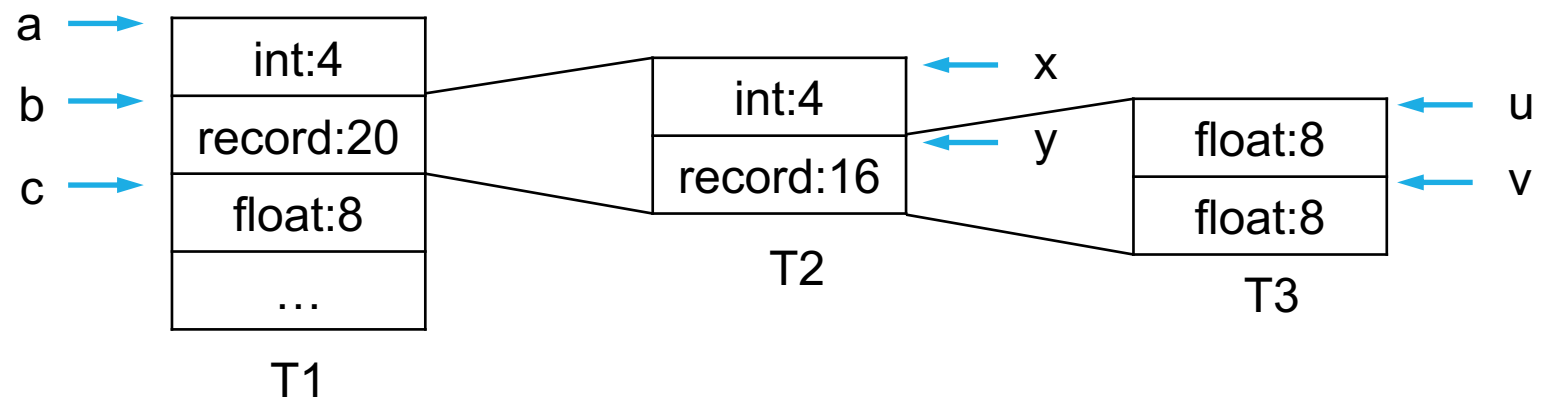
# SDT for Record Type

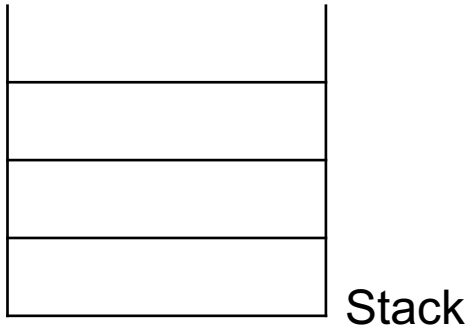
- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;



# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;



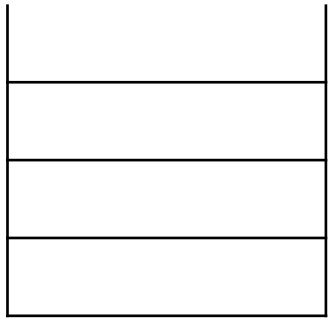




Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

parser

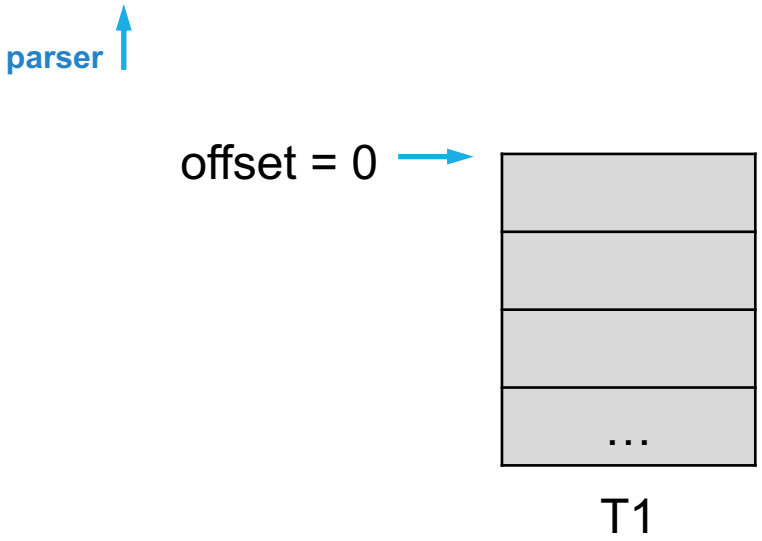


Stack

Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;




Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

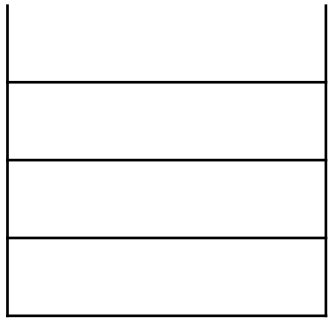
# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

parser

offset = 0 →


T1



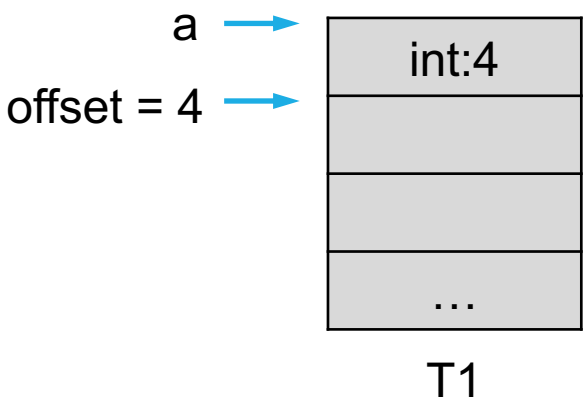
Stack

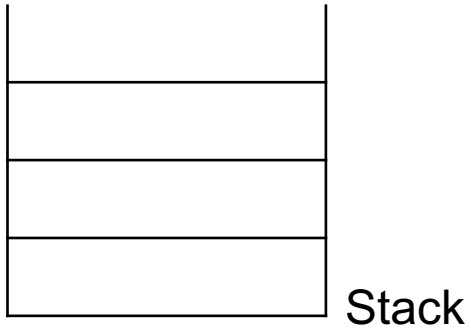
Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

parser



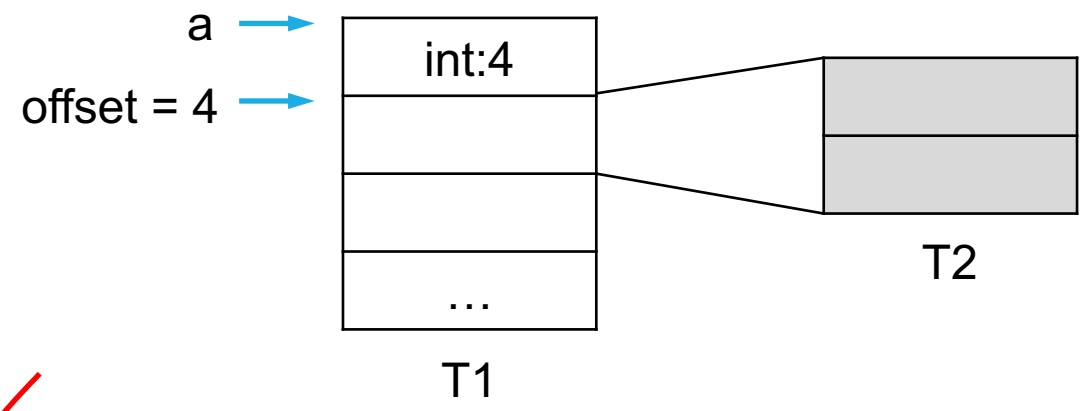



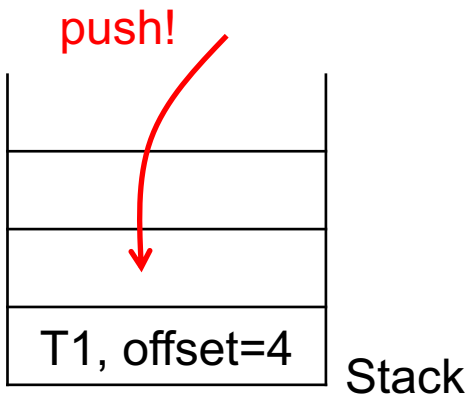

Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

parser

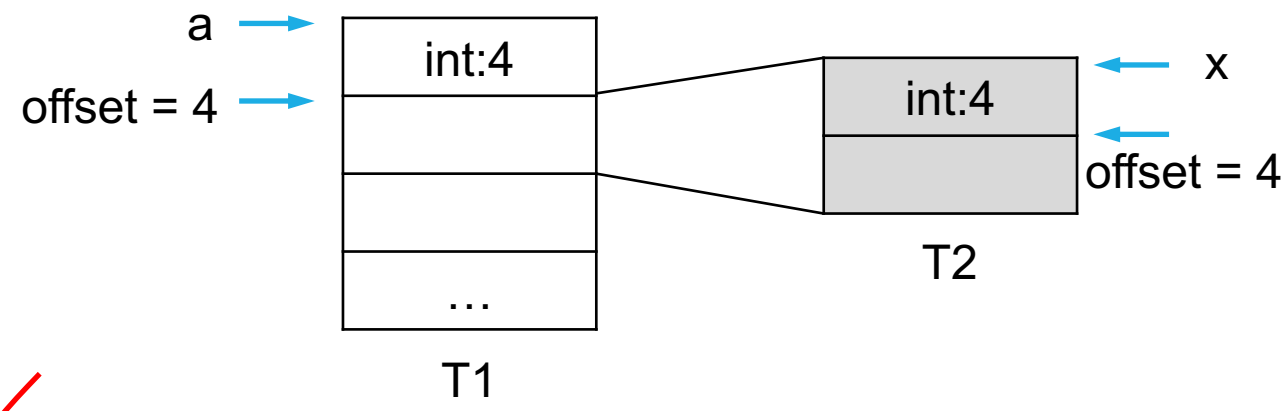


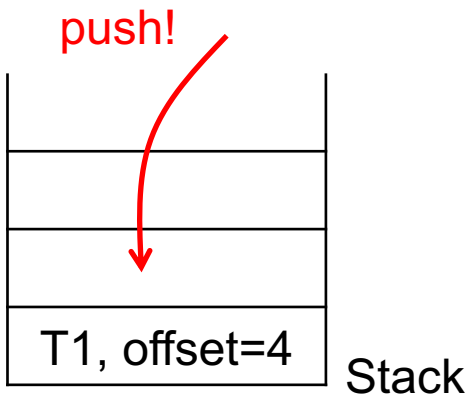




Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

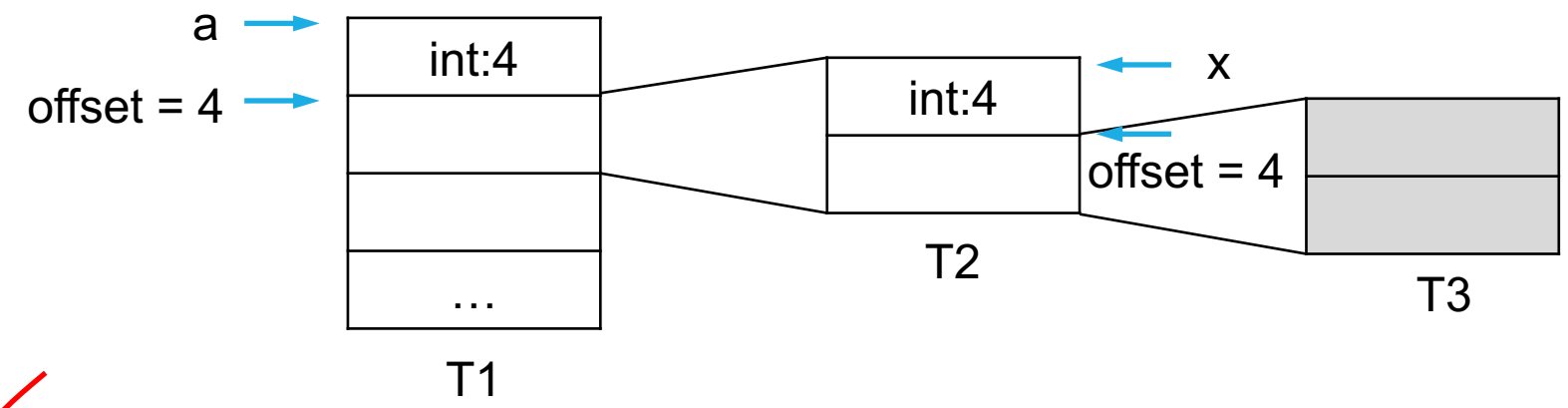


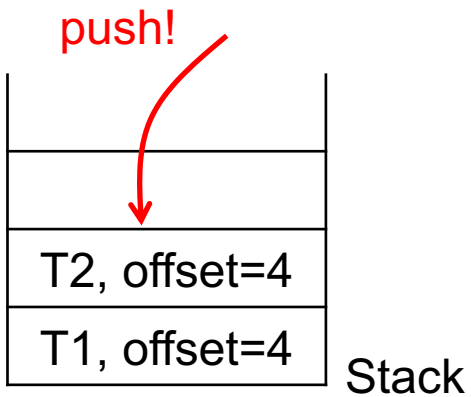




Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

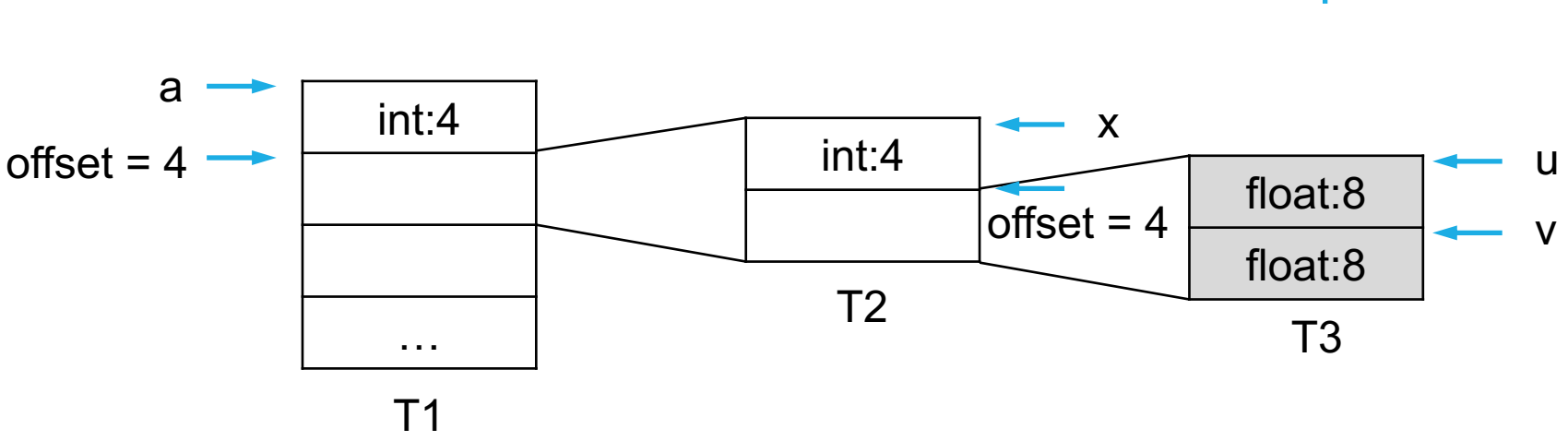


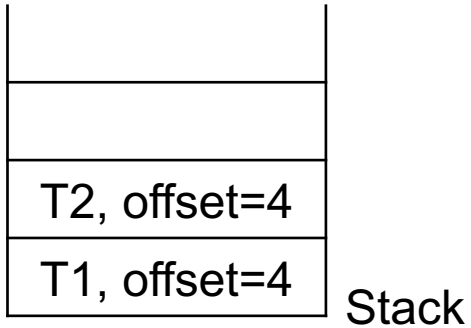




Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

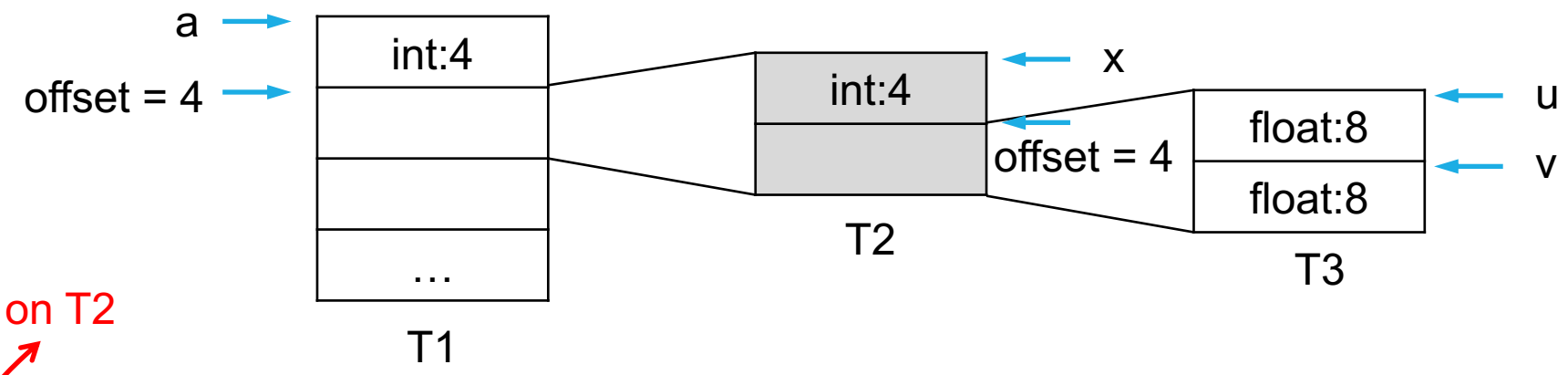


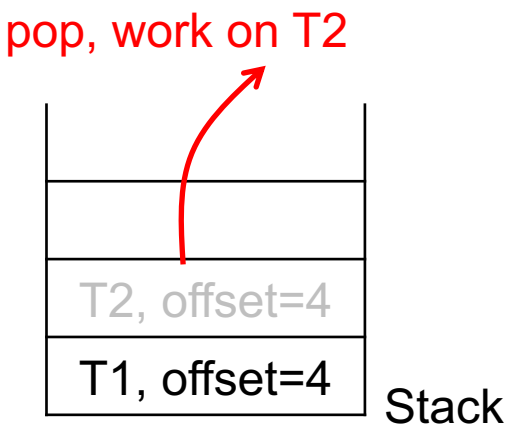




Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

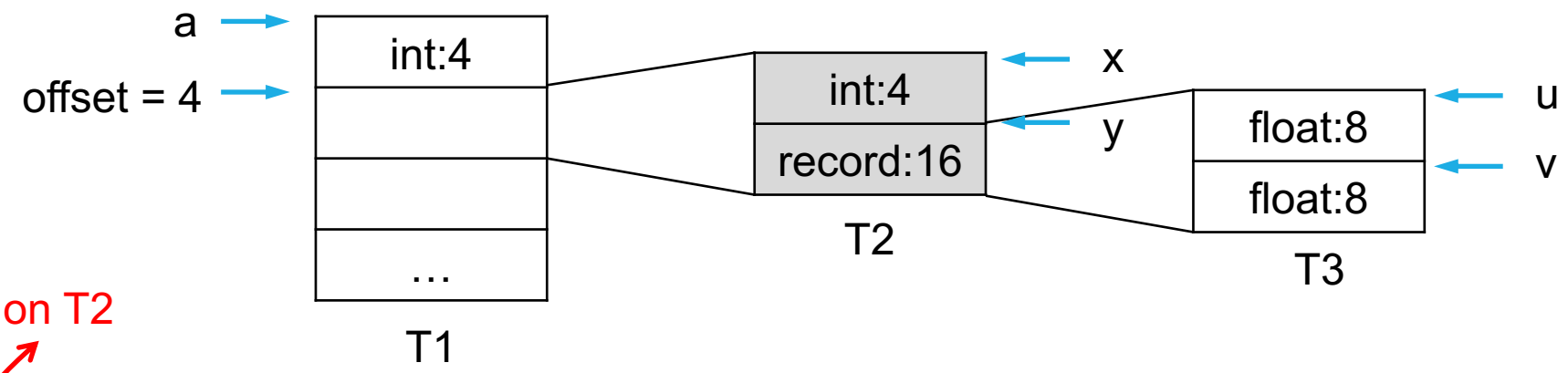


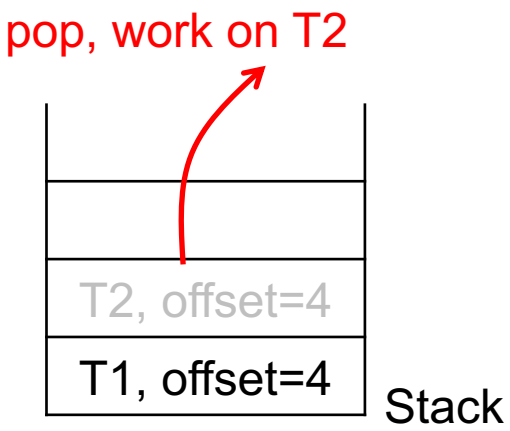




Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;



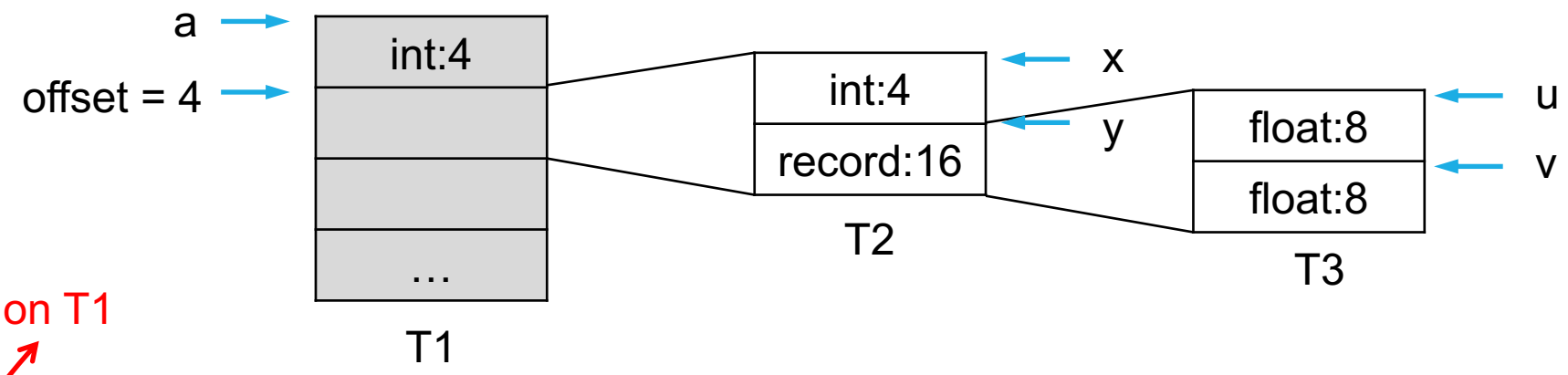



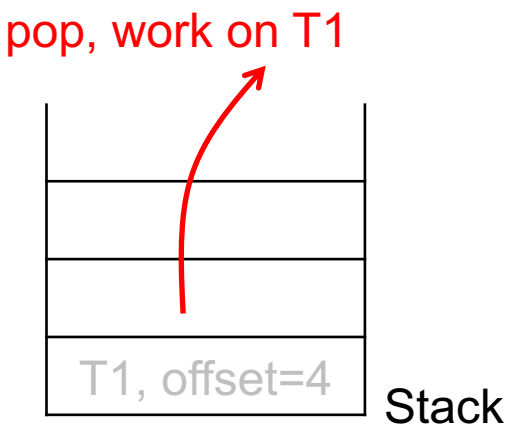

Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

parser



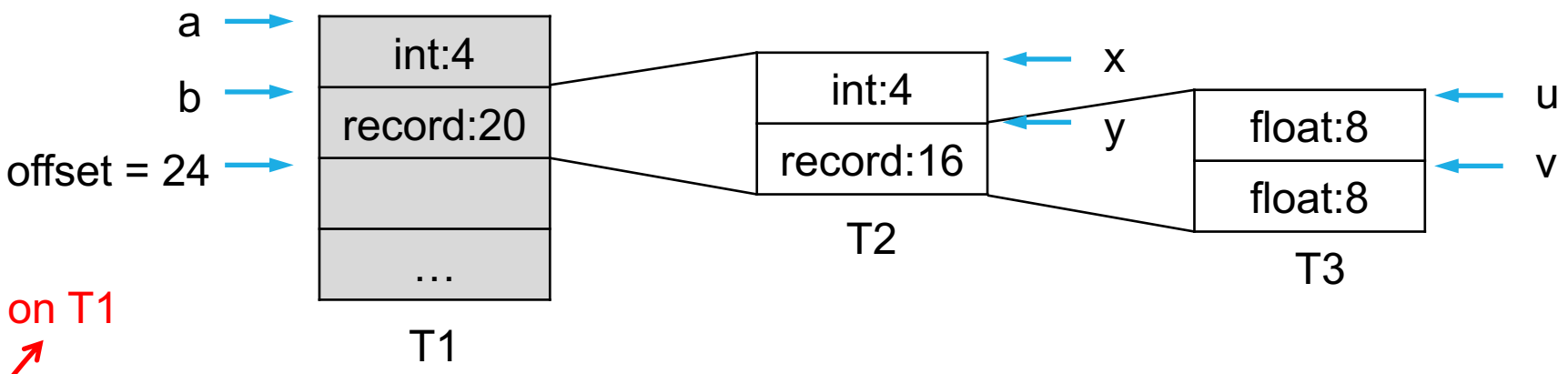



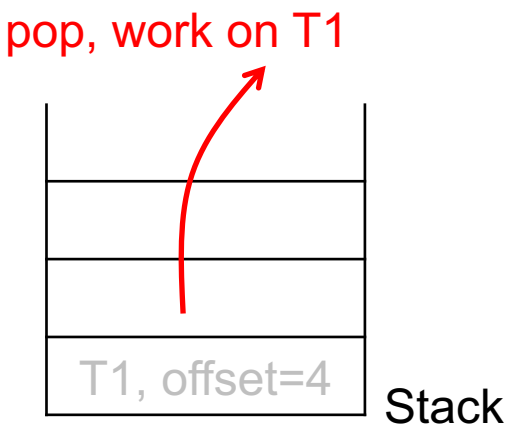

Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

parser



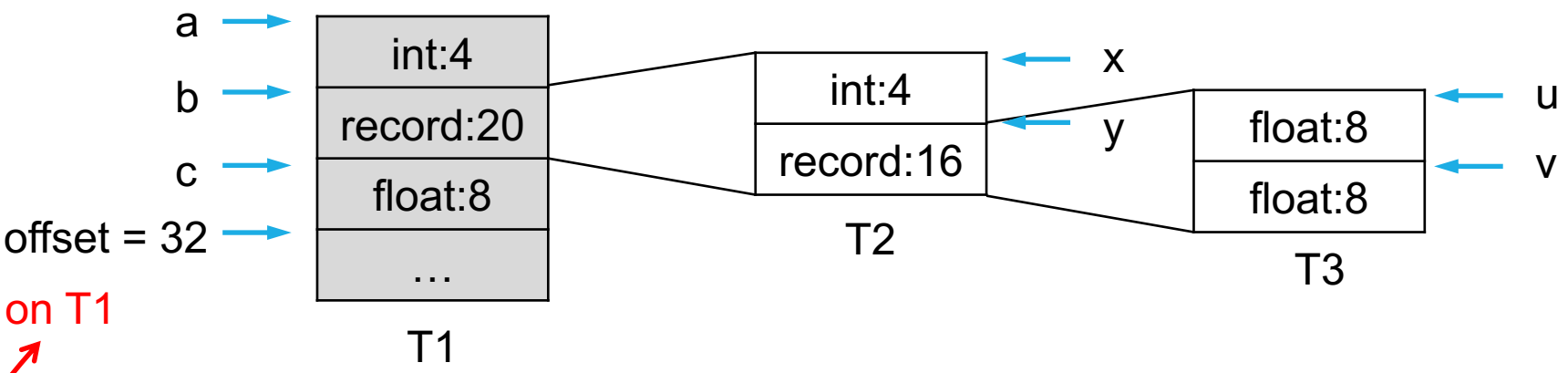



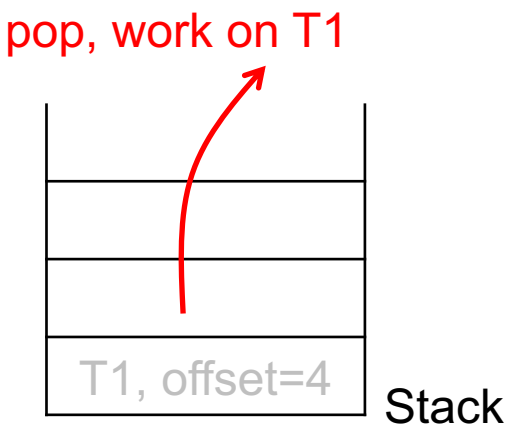

Build a stack of symbol tables;
Once the parser visits '{', a new symbol table is created and pushed to the stack;
Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

- **int** a; **record { int** x; **record { float** u; **float** v;**}** y; **}** b; **float** c;

parser




Build a stack of symbol tables;

Once the parser visits '{', a new symbol table is created and pushed to the stack;

Once the parser visits '}', the top symbol table is popped.

# SDT for Record Type

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

# SDT for Record Type

$$T \rightarrow \textbf{record}\ '\{'\ \{\ Env.push(top);\ top = \textbf{new}\ Env();\ Stack.push(offset);\ offset = 0;\ \}$$

$$D\ '\}'\ \{\ T.type = record(top);\ T.width = offset;\ top = Env.pop();\ offset = Stack.pop();\ \}$$

# SDT for Record Type

$$T \rightarrow \textbf{record}\ '\{'\ \{\ Env.push(top);\ top = \textbf{new}\ Env();\ Stack.push(offset);\ offset = 0;\ \}$$

$$D\ '\}'\ \{\ T.type = record(top);\ T.width = offset;\ top = Env.pop();\ offset = Stack.pop();\ \}$$

# SDT for Record Type

$$T \rightarrow \textbf{record}\ '\{'\quad \{\ Env.push(top);\ top = \textbf{new}\ Env();\ Stack.push(offset);\ offset = 0;\ \}$$

$$D\ '\}' \qquad \{\ T.type = record(top);\ T.width = offset;\ top = Env.pop();\ offset = Stack.pop();\ \}$$

# SDT for Record Type

$$T \rightarrow \textbf{record}\ '\{'\quad \{\ Env.push(top);\ top = \textbf{new}\ Env();\ Stack.push(offset);\ offset = 0;\ \}$$

$$D\ '\}' \qquad \{\ T.type = record(top);\ T.width = offset;\ top = Env.pop();\ offset = Stack.pop();\ \}$$

# PART II-2: Generation of IR

--- Generating IR for Expressions/Statements

# Expressions

```
int main(){
    int a, b, c;
    scanf("%d%d", &a, &b);
    c = a + b;
    printf("%d", c);
    return 0;
}
```

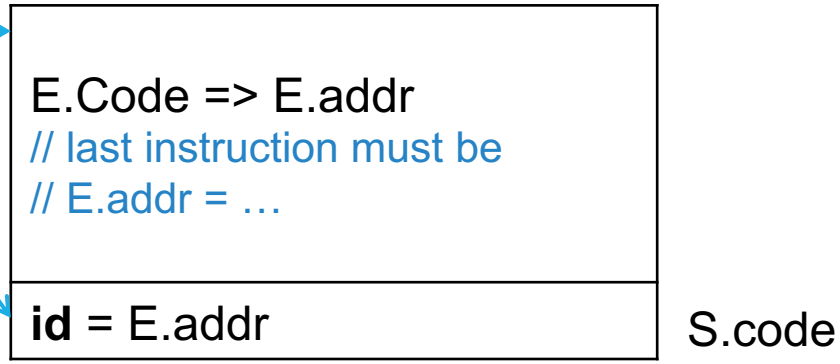
Common Expressions

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ ( E_1 )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ ( E_1 )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |



# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \textbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\textbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \textbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \textbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\textbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \textbf{id}$ | $E.addr = top.get(\textbf{id}.lexeme)$<br>$E.code = ''$ |

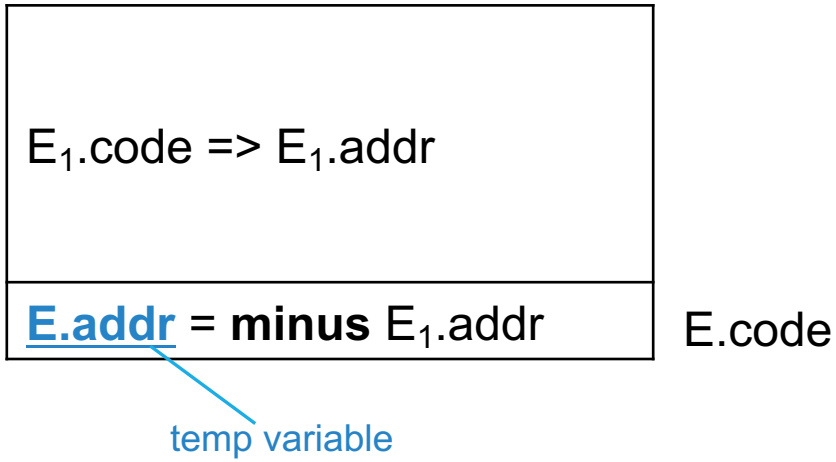
| |
|---|
| $E_1$.code => $E_1$.addr |
| $E_2$.code => $E_2$.addr |
| **E.addr** = $E_1$.addr + $E_2$.addr |

E.code

temp variable

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |



# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |

**Example**: `x = y + (-c)`

$temp_1$ = **minus** c

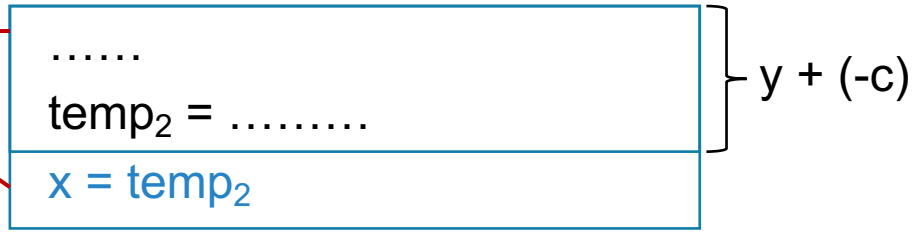
$temp_2$ = y + $temp_1$

x = $temp_2$

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \textbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\textbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \textbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \textbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\textbf{minus}'\ E_1.addr)$ |
| $\mid\ ( E_1 )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \textbf{id}$ | $E.addr = top.get(\textbf{id}.lexeme)$<br>$E.code = ''$ |

**Example**: `x = y + (-c)`



# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |

**Example**: x = y + (-c)

| |
|---|
| $temp_1$ = …… |
| $temp_2$ = y + $temp_1$ |

(-c)

x = $temp_2$

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |

**Example**: x = y + ( - c )

$temp_1$ = ……

$temp_2$ = y + $temp_1$

(-c)

x = $temp_2$

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |

**Example**: x = y + ( - c )

$temp_1$ = ……

$temp_2$ = y + $temp_1$

-c

x = $temp_2$

# Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \mathbf{id} = E\ ;$ | $S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \mathbf{new}\ Temp()$<br>$E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$ |
| $\mid\ (\ E_1\ )$ | $E.addr = E_1.addr$<br>$E.code = E_1.code$ |
| $\mid\ \mathbf{id}$ | $E.addr = top.get(\mathbf{id}.lexeme)$<br>$E.code = ''$ |

**Example**: x = y + ( - c )

temp$_1$ = **minus** c } -c

temp$_2$ = y + temp$_1$

x = temp$_2$

# Type Analysis

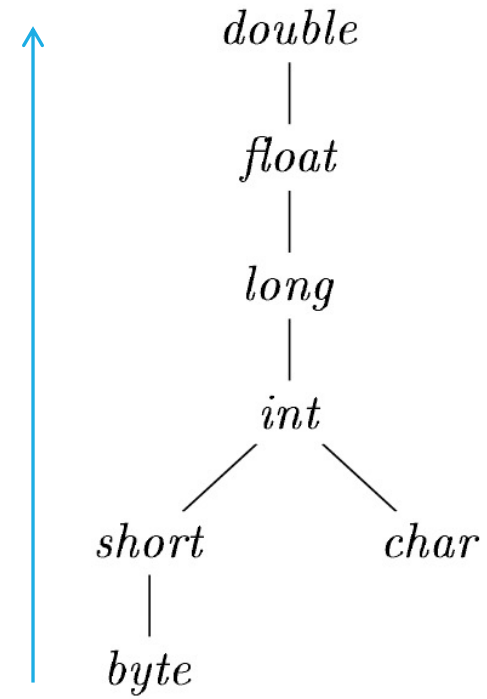
```
int main(){
    int a; char b; char c;
    scanf("%d%d", &a, &b);
    c = a + b;
    printf("%d", c);
    return 0;
}
```

# Type Analysis

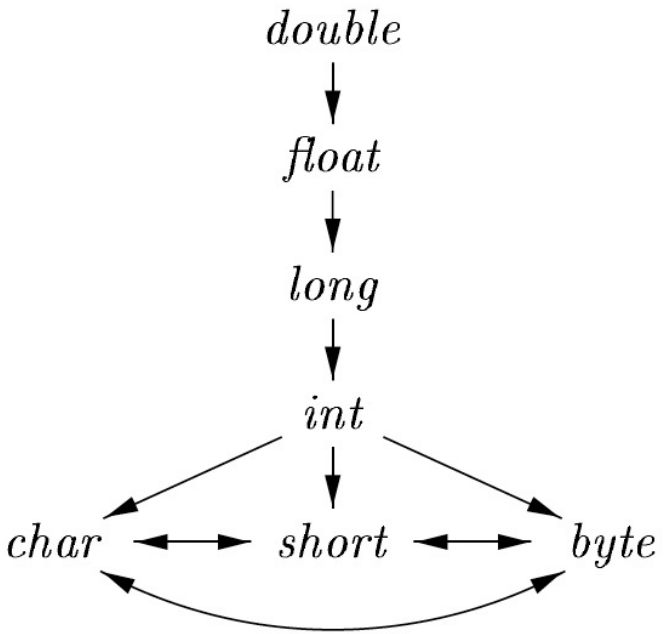
```
• int main(){
•     int a; char b; char c;
•     scanf("%d%d", &a, &b);
•     c = a + b;
•     printf("%d", c);
•     return 0;
• }
```

1. What is the type of a + b?
2. Is it safe to assign the result to c?
3. Why type analysis in compilers?

# Type Widening/Narrowing (Java)




(a) Widening conversions




(b) Narrowing conversions

Narrowing loses precision!

# Type Widening

Compute Target Type

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

# Type Widening

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

Cast to Target Type

# Type Widening

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

# Type Widening

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

| $E_1$.code => $E_1$.addr |
| --- |
| $E_2$.code => $E_2$.addr |
| E.addr = $E_1$.addr + $E_2$.addr |

E.code

# Type Widening

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

| $E_1$.code => $E_1$.addr |
|---|
| $E_2$.code => $E_2$.addr |
| E.addr = $E_1$.addr + $E_2$.addr |

E.code

➡

| $E_1$.code => $E_1$.addr<br>**$a_1$ = (target type) $E_1$.addr** |
|---|
| $E_2$.code => $E_2$.addr<br>**$a_2$ = (target type) $E_2$.addr** |
| E.addr = **$a_1$** + **$a_2$** |

E.code

# Type Checking

# Type Checking

- Checking types for assignment
  - S → **id** = E
  - { **id**.type vs. E.type }

- Similar cases for function overloading
  - E → **id**(ParamList)
  - { compare **id**.type vs. ParamList.types }

# PART II-3: Generation of IR

--- Generating IR for Control Flows

# Outline

- if-else statement; while statement
- break and continue
- Boolean expressions and short-circuit

# Control Flow

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

# Control Flow

| Production | Semantic Rules |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow \textbf{assign}$ | $S.code = \textbf{assign}.code$ |
| $S \rightarrow \textbf{if} ( B ) S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow \textbf{if} ( B ) S_1 \textbf{ else } S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow \textbf{while} ( B ) S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

# Control Flow

| Production | Semantic Rules |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow \textbf{assign}$ | $S.code = \textbf{assign}.code$ |
| $S \rightarrow \textbf{if} ( B ) S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow \textbf{if} ( B ) S_1 \textbf{ else } S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow \textbf{while} ( B ) S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

# Control Flow

Branching

| Production | Semantic Rules |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

# Control Flow

Branching

Sequencing

| Production | Semantic Rules |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow \textbf{assign}$ | $S.code = \textbf{assign}.code$ |
| $S \rightarrow \textbf{if} ( B ) S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow \textbf{if} ( B ) S_1 \textbf{ else } S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow \textbf{while} ( B ) S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| | $S_1$.code |
|---|---|
| $S_1$.next: | $S_2$.code |
| S.next/$S_2$.next: | … |

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| | $S_1$.code |
|---|---|
| $S_1$.next: | $S_2$.code |
| S.next/$S_2$.next: | … |

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

maintain a
linked list of stmts

| | $S_1$.code |
|---|---|
| $S_1$.next: | $S_2$.code |
| S.next/$S_2$.next: | … |

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

**Example:**

u = v * 2
w = u / 3
x = w + 1
y = x + 2

maintain a
linked list of stmts

| | $S_1$.code |
|---|---|
| $S_1$.next: | $S_2$.code |
| S.next/$S_2$.next: | … |

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| | |
|---|---|
| | if B goto B.true<br>goto B.false |
| B.true: | $S_1$.code |
| B.false: | |

B.code

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| | |
|---|---|
| | if B goto B.true<br>goto B.false |
| B.true: | $S_1$.code |
| B.false: | |

B.code

maintain a linked list

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| | |
|---|---|
| | if B goto B.true<br>goto B.false |
| B.true: | $S_1$.code |
| B.false: | |

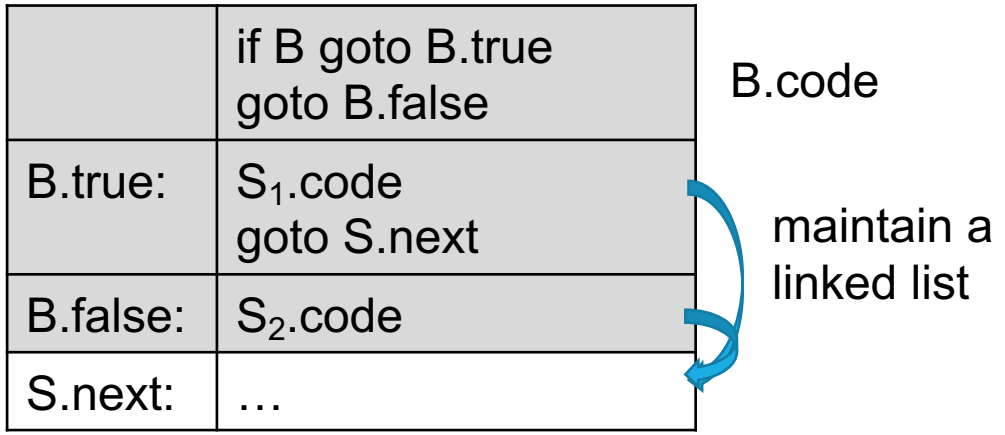
B.code

maintain a linked list

**Example:**

```
if (v)
    w = u / 3
    x = w + 1
y = x + 2
```

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |



| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| | |
|---|---|
| **Begin:** | if B goto B.true<br>goto B.false |
| B.true: | $S_1$.code |
| | **goto Begin** |
| B.false: | … |

maintain a linked list

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** ( $B$ ) $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| | |
|---|---|
| **Begin:** | if B goto B.true<br>goto B.false |
| B.true: | $S_1$.code |
| | **goto Begin** |
| B.false: | … |

maintain a linked list

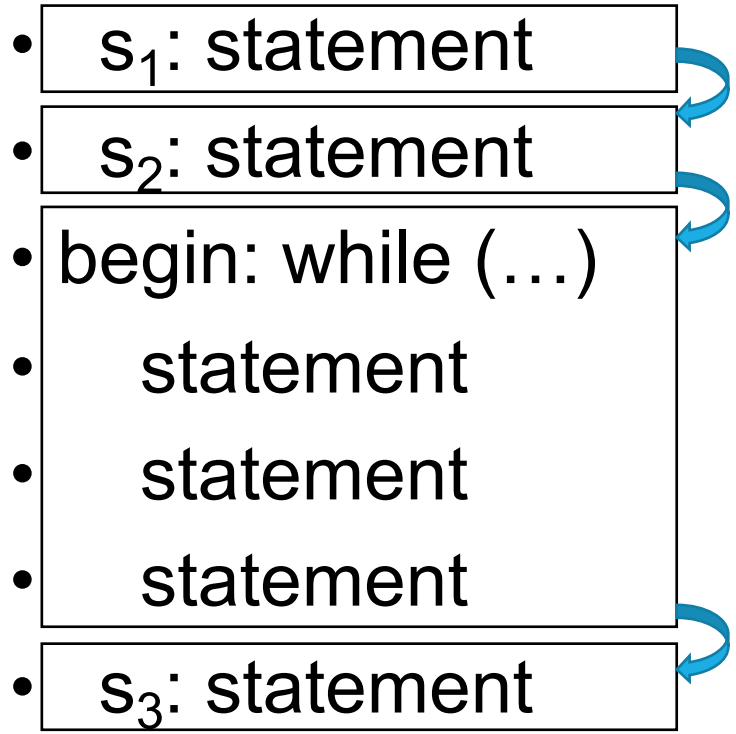
**Example:**

```
while (v)
    w = u / 3
    x = w + 1
y = x + 2
```

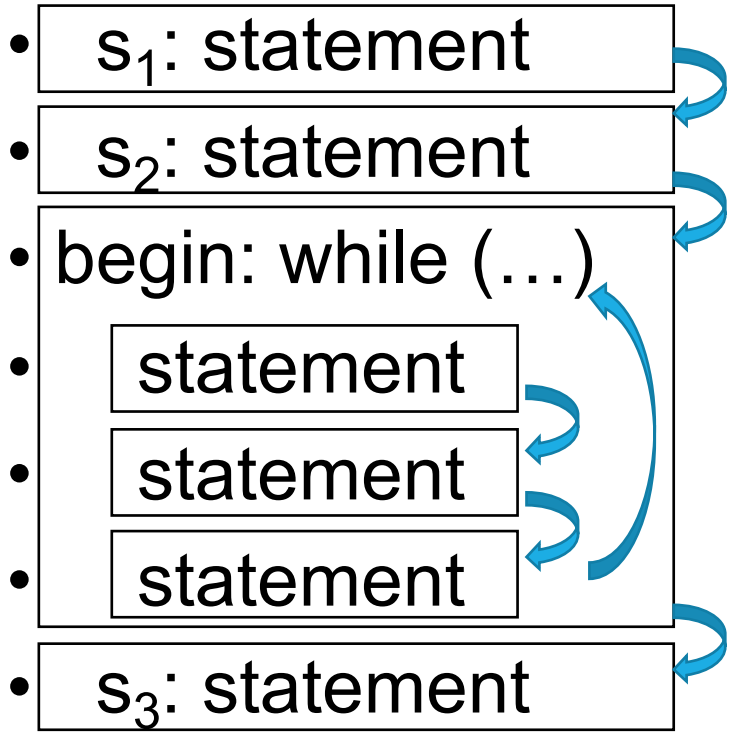
# LinkedList of Statements

- $s_1$: statement
- $s_2$: statement
- begin: while (…)
- statement
- statement
- statement
- $s_3$: statement
- …

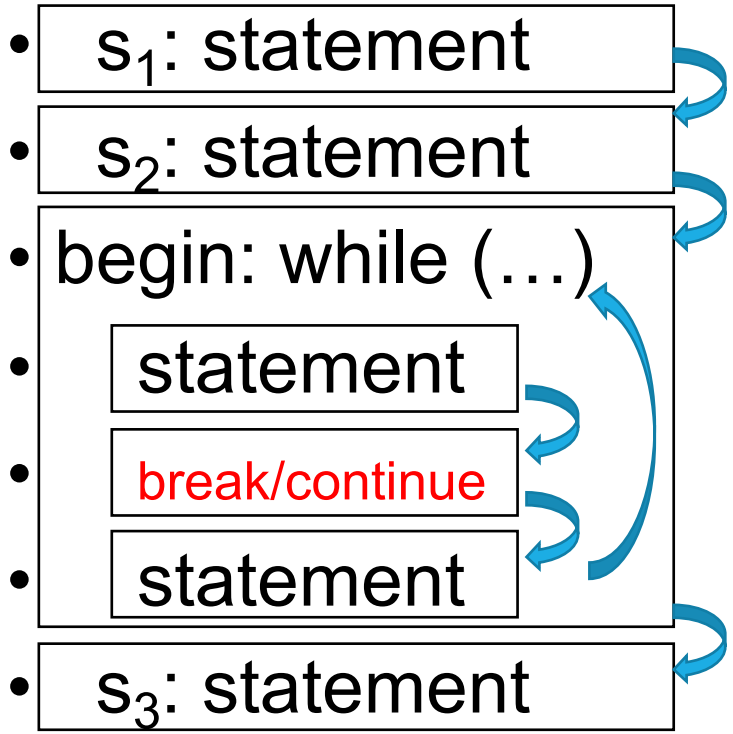
# LinkedList of Statements



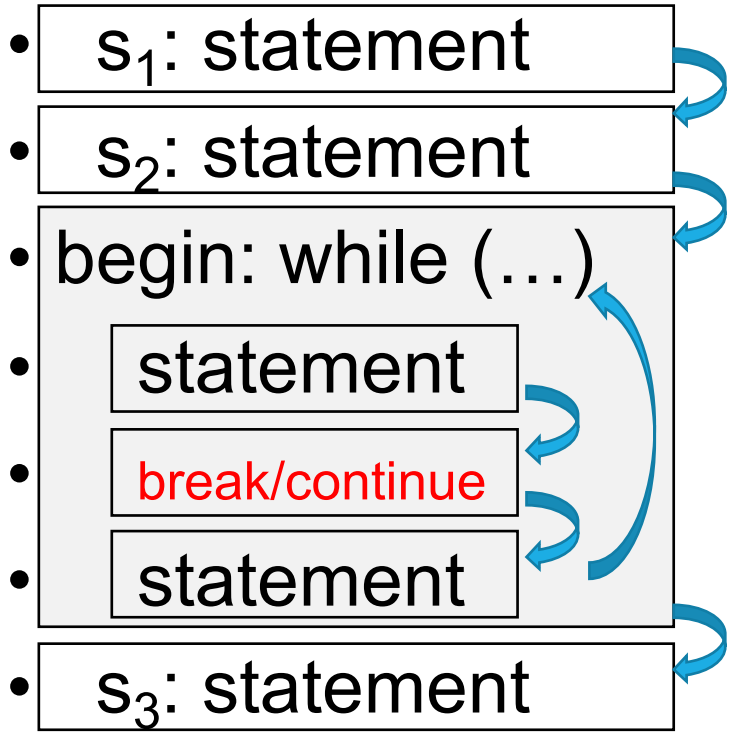
# LinkedList of Statements



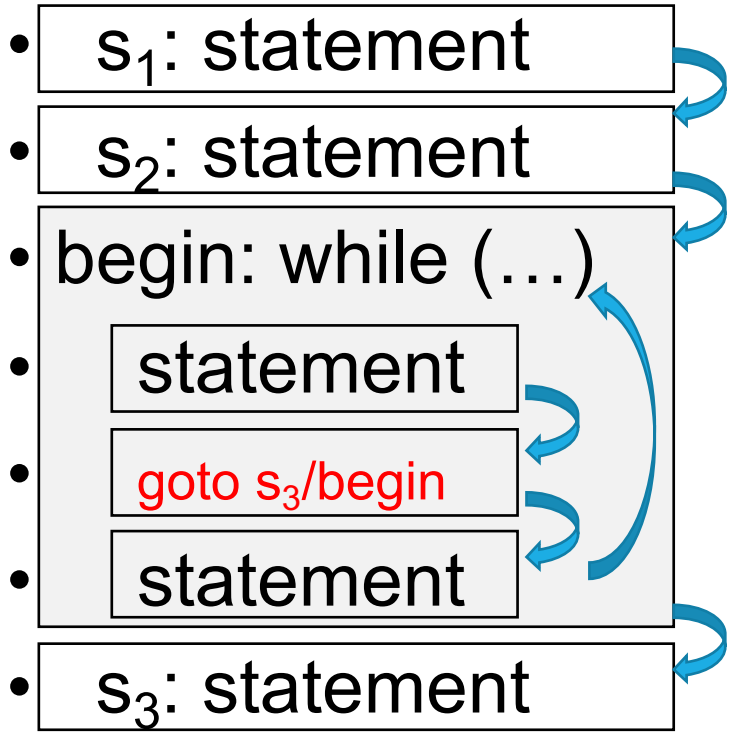
# Break and Continue



# Break and Continue



# Break and Continue



# Boolean Expressions

# Boolean Expressions

| | |
|---|---|
| $S \rightarrow \textbf{if}\ (\ B\ )\ S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow \textbf{if}\ (\ B\ )\ S_1\ \textbf{else}\ S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow \textbf{while}\ (\ B\ )\ S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

| | |
|---|---|
| | if B goto B.true<br>goto B.false |
| B.true: | $S_1$.code |
| B.false: | |

**B.code**

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \ \texttt{||} \ B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \ \|\| \ label(B_1.false) \ \|\| \ B_2.code$ |
| $B \rightarrow B_1 \ \&\& \ B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \ \|\| \ label(B_1.true) \ \|\| \ B_2.code$ |
| $B \rightarrow \texttt{!} \ B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1 \ \textbf{rel} \ E_2$ | $B.code = E_1.code \ \|\| \ E_2.code$<br>$\|\| \ gen('\texttt{if}' \ E_1.addr \ \textbf{rel}.op \ E_2.addr \ '\texttt{goto}' \ B.true)$<br>$\|\| \ gen('\texttt{goto}' \ B.false)$ |
| $B \rightarrow \textbf{true}$ | $B.code = gen('\texttt{goto}' \ B.true)$ |
| $B \rightarrow \textbf{false}$ | $B.code = gen('\texttt{goto}' \ B.false)$ |

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \ \|\| \ B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \ \|\| \ label(B_1.false) \ \|\| \ B_2.code$ |
| $B \rightarrow B_1 \ \&\& \ B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \ \|\| \ label(B_1.true) \ \|\| \ B_2.code$ |
| $B \rightarrow \ ! \ B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1 \ \mathbf{rel} \ E_2$ | $B.code = E_1.code \ \|\| \ E_2.code$<br>$\|\| \ gen('\texttt{if}' \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ '\texttt{goto}' \ B.true)$<br>$\|\| \ gen('\texttt{goto}' \ B.false)$ |
| $B \rightarrow \mathbf{true}$ | $B.code = gen('\texttt{goto}' \ B.true)$ |
| $B \rightarrow \mathbf{false}$ | $B.code = gen('\texttt{goto}' \ B.false)$ |

| |
|---|
| $E_1$.code => $E_1$.addr |
| $E_2$.code => $E_2$.addr |
| If $E_1$.addr ***rel*** $E_2$.addr goto B.true |
| goto B.false |

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \mid\mid B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$ |
| $B \rightarrow B_1 \;\&\&\; B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$ |
| $B \rightarrow ! B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1$ **rel** $E_2$ | $B.code = E_1.code \mid\mid E_2.code$<br>$\mid\mid gen(\texttt{'if'}\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ \texttt{'goto'}\ B.true)$<br>$\mid\mid gen(\texttt{'goto'}\ B.false)$ |
| $B \rightarrow$ **true** | $B.code = gen(\texttt{'goto'}\ B.true)$ |
| $B \rightarrow$ **false** | $B.code = gen(\texttt{'goto'}\ B.false)$ |

| |
|---|
| $E_1$.code => $E_1$.addr |
| $E_2$.code => $E_2$.addr |
| If $E_1$.addr ***rel*** $E_2$.addr goto B.true |
| goto B.false |

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \mid\mid B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$ |
| $B \rightarrow B_1 \;\&\&\; B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$ |
| $B \rightarrow ! B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1$ **rel** $E_2$ | $B.code = E_1.code \mid\mid E_2.code$<br>$\mid\mid gen(\texttt{'if'}\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ \texttt{'goto'}\ B.true)$<br>$\mid\mid gen(\texttt{'goto'}\ B.false)$ |
| $B \rightarrow$ **true** | $B.code = gen(\texttt{'goto'}\ B.true)$ |
| $B \rightarrow$ **false** | $B.code = gen(\texttt{'goto'}\ B.false)$ |

| |
|---|
| $E_1$.code => $E_1$.addr |
| $E_2$.code => $E_2$.addr |
| If $E_1$.addr ***rel*** $E_2$.addr goto B.true |
| goto B.false |

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \mid\mid B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$ |
| $B \rightarrow B_1 \,\&\&\, B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$ |
| $B \rightarrow ! B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1$ **rel** $E_2$ | $B.code = E_1.code \mid\mid E_2.code$<br>$\mid\mid gen('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$<br>$\mid\mid gen('\texttt{goto}'\ B.false)$ |
| $B \rightarrow$ **true** | $B.code = gen('\texttt{goto}'\ B.true)$ |
| $B \rightarrow$ **false** | $B.code = gen('\texttt{goto}'\ B.false)$ |

$S \rightarrow$ **if** $(B)\ S_1$

| | B.code |
|---|---|
| B.true: | $S_1$.code |
| B.false: | |

**Example:**

if (v + 1 > z + 2)
w = u / 3

| |
|---|
| $E_1$.code => $E_1$.addr |
| $E_2$.code => $E_2$.addr |
| If $E_1$.addr ***rel*** $E_2$.addr goto B.true |
| goto B.false |

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \ \|\| \ B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \ \|\| \ label(B_1.false) \ \|\| \ B_2.code$ |
| $B \rightarrow B_1 \ \&\& \ B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \ \|\| \ label(B_1.true) \ \|\| \ B_2.code$ |
| $B \rightarrow \ ! \ B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1 \ \mathbf{rel} \ E_2$ | $B.code = E_1.code \ \|\| \ E_2.code$<br>$\|\| \ gen('\texttt{if}' \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ '\texttt{goto}' \ B.true)$<br>$\|\| \ gen('\texttt{goto}' \ B.false)$ |
| $B \rightarrow \mathbf{true}$ | $B.code = gen('\texttt{goto}' \ B.true)$ |
| $B \rightarrow \mathbf{false}$ | $B.code = gen('\texttt{goto}' \ B.false)$ |

| | |
|---|---|
| | $B_1$.code |
| $B_1$.true: | $B_2$.code |

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \mid\mid B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$ |
| $B \rightarrow B_1 \;\&\&\; B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$ |
| $B \rightarrow ! B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1 \;\mathbf{rel}\; E_2$ | $B.code = E_1.code \mid\mid E_2.code$<br>$\mid\mid gen('\texttt{if}'\ E_1.addr\ \mathbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$<br>$\mid\mid gen('\texttt{goto}'\ B.false)$ |
| $B \rightarrow \mathbf{true}$ | $B.code = gen('\texttt{goto}'\ B.true)$ |
| $B \rightarrow \mathbf{false}$ | $B.code = gen('\texttt{goto}'\ B.false)$ |

| | |
|---|---|
| | $B_1$.code |
| $B_1$.true: | $B_2$.code |

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \mid\mid B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$ |
| $B \rightarrow B_1 \;\&\&\; B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$ |
| $B \rightarrow ! B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1 \;\mathbf{rel}\; E_2$ | $B.code = E_1.code \mid\mid E_2.code$<br>$\mid\mid gen('\texttt{if}'\ E_1.addr\ \mathbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$<br>$\mid\mid gen('\texttt{goto}'\ B.false)$ |
| $B \rightarrow \mathbf{true}$ | $B.code = gen('\texttt{goto}'\ B.true)$ |
| $B \rightarrow \mathbf{false}$ | $B.code = gen('\texttt{goto}'\ B.false)$ |

$S \rightarrow \mathbf{if}\ (B)\ S_1$

| | B.code |
|---|---|
| B.true: | $S_1$.code |
| B.false: | |

| | $B_1$.code |
|---|---|
| $B_1$.true: | $B_2$.code |

**Example:**

if (a > b && c > d)
w = u / 3

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \mid\mid B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$ |
| $B \rightarrow B_1 \;\&\&\; B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$ |
| $B \rightarrow ! B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1$ **rel** $E_2$ | $B.code = E_1.code \mid\mid E_2.code$<br>$\mid\mid gen('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$<br>$\mid\mid gen('\texttt{goto}'\ B.false)$ |
| $B \rightarrow$ **true** | $B.code = gen('\texttt{goto}'\ B.true)$ |
| $B \rightarrow$ **false** | $B.code = gen('\texttt{goto}'\ B.false)$ |

$S \rightarrow \textbf{if}\ (B)\ S_1$

| | B.code |
|---|---|
| B.true: | $S_1$.code |
| B.false: | |

| | $B_1$.code |
|---|---|
| $B_1$.true: | $B_2$.code |

**Example:**

if (a > b && c > d)
w = u / 3

# Boolean Expressions

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $B \rightarrow B_1 \mid\mid B_2$ | $B_1.true = B.true$<br>$B_1.false = newlabel()$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$ |
| $B \rightarrow B_1 \;\&\&\; B_2$ | $B_1.true = newlabel()$<br>$B_1.false = B.false$<br>$B_2.true = B.true$<br>$B_2.false = B.false$<br>$B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$ |
| $B \rightarrow ! B_1$ | $B_1.true = B.false$<br>$B_1.false = B.true$<br>$B.code = B_1.code$ |
| $B \rightarrow E_1$ **rel** $E_2$ | $B.code = E_1.code \mid\mid E_2.code$<br>$\mid\mid gen('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$<br>$\mid\mid gen('\texttt{goto}'\ B.false)$ |
| $B \rightarrow$ **true** | $B.code = gen('\texttt{goto}'\ B.true)$ |
| $B \rightarrow$ **false** | $B.code = gen('\texttt{goto}'\ B.false)$ |

$S \rightarrow \textbf{if}\ (B)\ S_1$

| | B.code |
|---|---|
| B.true: | $S_1$.code |
| B.false: | |

| | $B_1$.code |
|---|---|
| $B_1$.true: | $B_2$.code |

**Example:**

if (a > b && c > d)
w = u / 3

**Short-circuit!**

# Others

- Optimizations:
  - Avoid unnecessary gotos
  - Backpatching
- Translation for:
  - Switch statement
  - Function / Function call

- ***Read the dragon book, Chapter 6.***

# THANKS!