四、中间代码生成
(1. LLVM IR 简介)

魏恒峰

hfwei@nju.edu.cn

2023 年 05 月 05 日

编译器大佬Chris Lattner全新编程语言「Mojo」：兼容Python核心功能，提速35000倍

机器之心 2023-05-04 12:58 发表于浙江

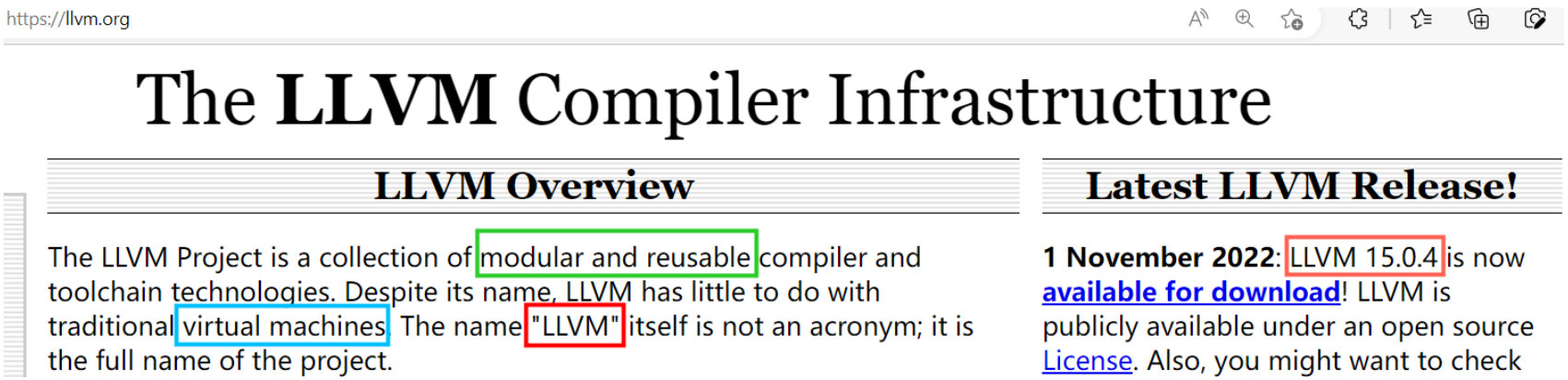
机器之心报道

编辑：蛋酱、陈萍

> 它可与 Python 无缝衔接，但克服了很多 Python 的缺点。Jeremy Howard 试用后表示：「Mojo 可能是几十年来最大的编程进步。」

对于全球各地开发者来说，Chris Lattner 这个名字绝对不陌生。

https://llvm.org/

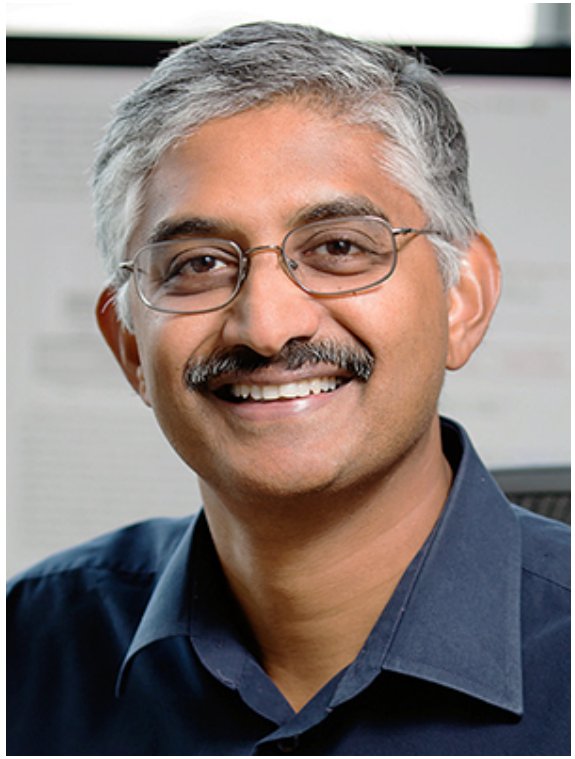


"Low Level Virtual Machine"

Year ⬍	Project
2021	CompCert
2020	Berkeley DB
2019	DNS
2018	Wireshark
2017	Project Jupyter
2016	Andrew File System
2015	GCC
2014	Mach
2013	Coq
2012	LLVM
2011	Eclipse

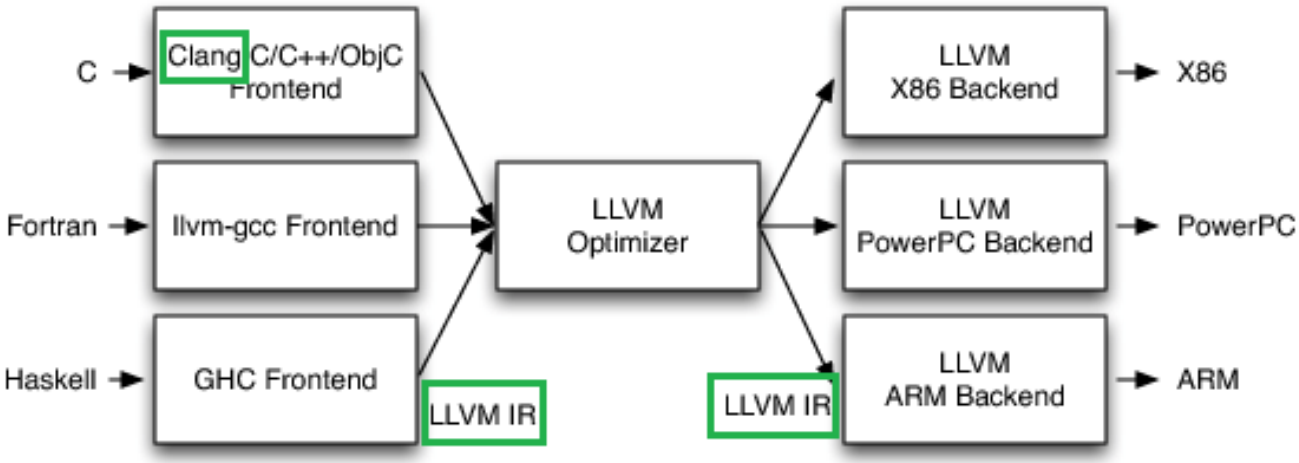
ACM Software System Award

Chris Lattner (1978); UIUC 2020



Vikram Adve (1966)

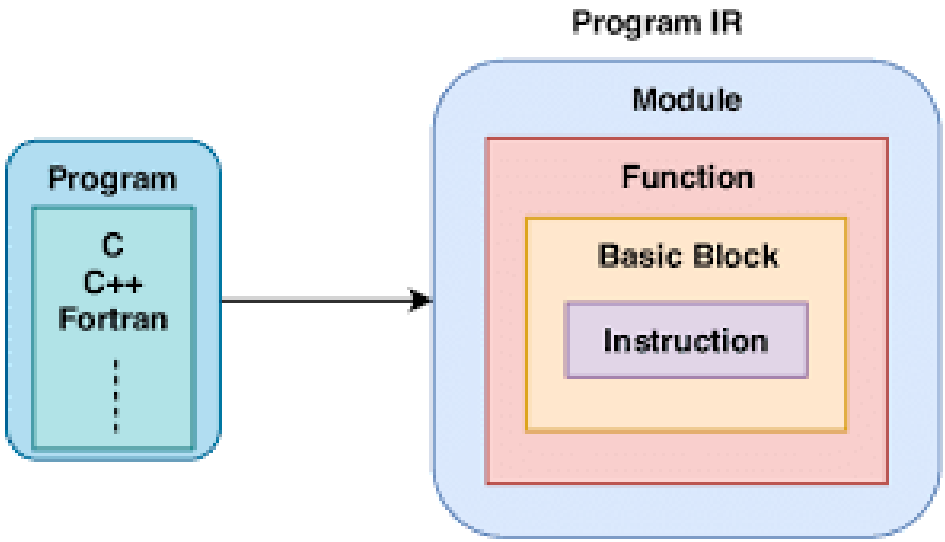
LLVM IR (Intermediate Representation)



“IR 设计的优秀与否决定着整个编译器的好坏”

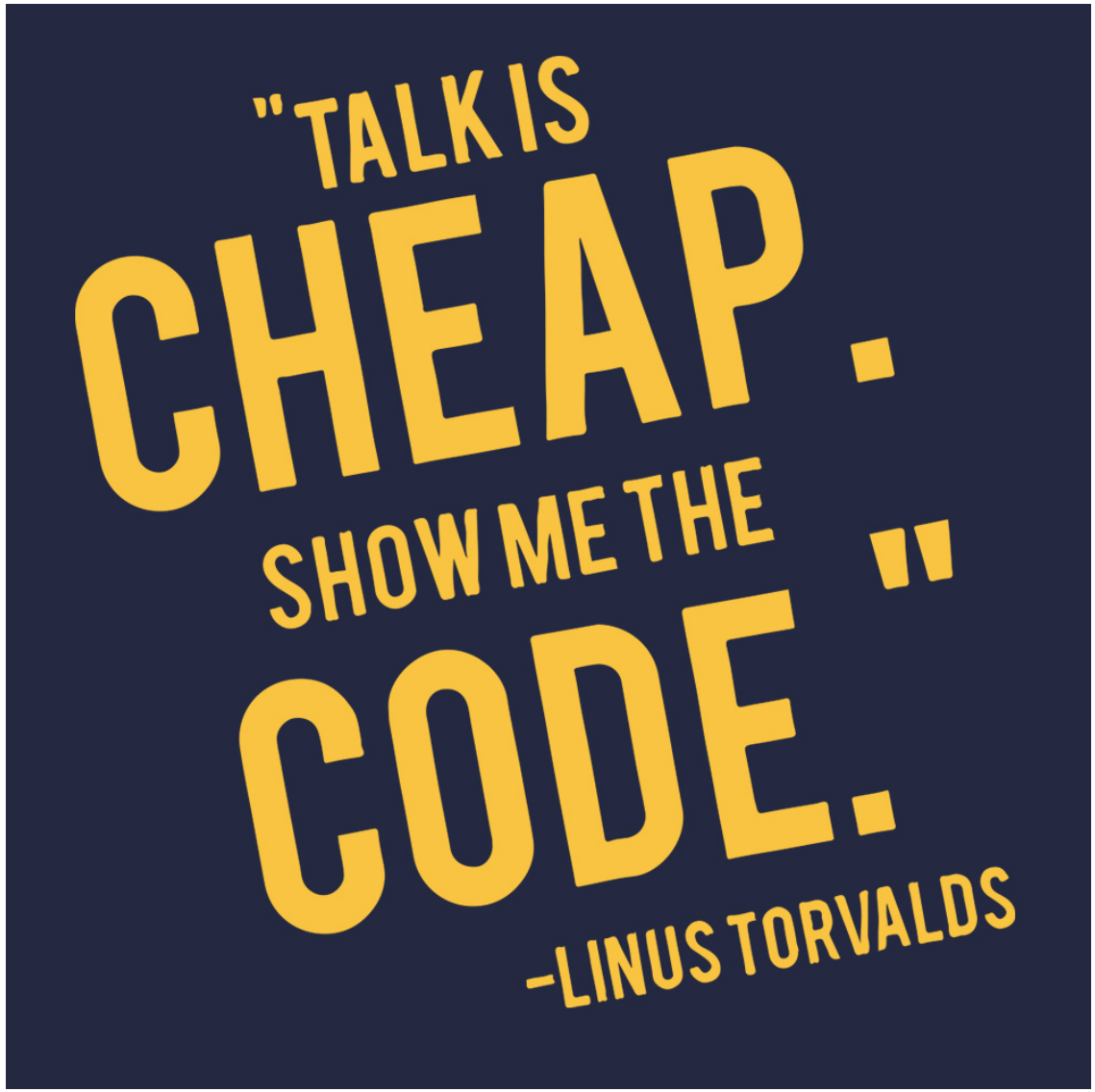
8 章技术内容, 其中 4 章介绍 Maple IR, 另外 4 章基于 Maple IR

LLVM Language Reference Manual



IR: Intermediate Representation

LLVM IR: 带类型的、介于高级程序设计语言与汇编语言之间
(LLVM Assembly Language)


"TALK IS
CHEAP.
SHOW ME THE
CODE."
-LINUS TORVALDS

```c
int factorial(int val);

int main(int argc, char **argv) {
  return factorial( val: 2) * 7 == 42;
}
```

factorial0.c

```
6  ; Function Attrs: noinline nounwind optnone uwtable
7  define dso_local i32 @main(i32 noundef %0, i8** noundef %1) #0 {
8    %3 = alloca i32, align 4
9    %4 = alloca i32, align 4
10   %5 = alloca i8**, align 8
11   store i32 0, i32* %3, align 4
12   store i32 %0, i32* %4, align 4
13   store i8** %1, i8*** %5, align 8
14   %6 = call i32 @factorial(i32 noundef 2)
15   %7 = mul nsw i32 %6, 7
16   %8 = icmp eq i32 %7, 42
17   %9 = zext i1 %8 to i32
18   ret i32 %9
19 }
```

```
clang -S -emit-llvm factorial0.c -o f0-opt0.ll
```

Three Address Code (TAC)

```
6  ; Function Attrs: noinline nounwind optnone uwtable
7  define dso_local i32 @main(i32 noundef %0, i8** noundef %1) #0 {
8    %3 = alloca i32, align 4
9    %4 = alloca i32, align 4
10   %5 = alloca i8**, align 8
11   store i32 0, i32* %3, align 4
12   store i32 %0, i32* %4, align 4
13   store i8** %1, i8*** %5, align 8
14   %6 = call i32 @factorial(i32 noundef 2)
15   %7 = mul nsw i32 %6, 7
16   %8 = icmp eq i32 %7, 42
17   %9 = zext i1 %8 to i32
18   ret i32 %9
19 }
```

```
clang -S -emit-llvm factorial0.c -o f0-opt0.ll
```

```
6  ; Function Attrs: noinline nounwind optnone uwtable
7  define dso_local i32 @main(i32 noundef %0, i8** noundef %1) #0 {
8    %3 = alloca i32, align 4
9    %4 = alloca i32, align 4
10   %5 = alloca i8**, align 8
11   store i32 0, i32* %3, align 4
12   store i32 %0, i32* %4, align 4
13   store i8** %1, i8*** %5, align 8
14   %6 = call i32 @factorial(i32 noundef 2)
15   %7 = mul nsw i32 %6, 7
16   %8 = icmp eq i32 %7, 42
17   %9 = zext i1 %8 to i32
18   ret i32 %9
19 }
```

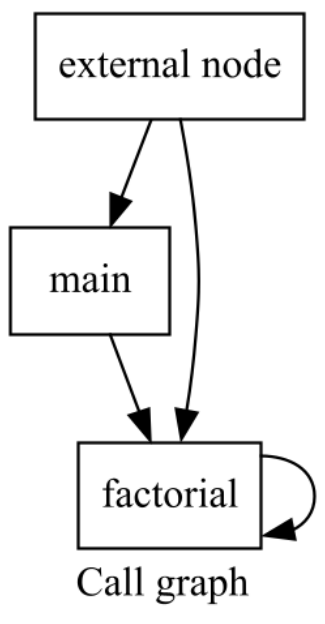
```
clang -S -emit-llvm factorial0.c -o f0-opt0.ll
```

mem2reg

```
6   ; Function Attrs: nounwind uwtable
7   define dso_local i32 @main(i32 %0, i8** nocapture readnone %1)
8     %3 = call i32 @factorial(i32 2) #2
9     %4 = mul nsw i32 %3, 7
10    %5 = icmp eq i32 %4, 42
11    %6 = zext i1 %5 to i32
12    ret i32 %6
13  }
```

```
clang -S -emit-llvm factorial0.c -o f0-opt1.ll -O1 -g0
```

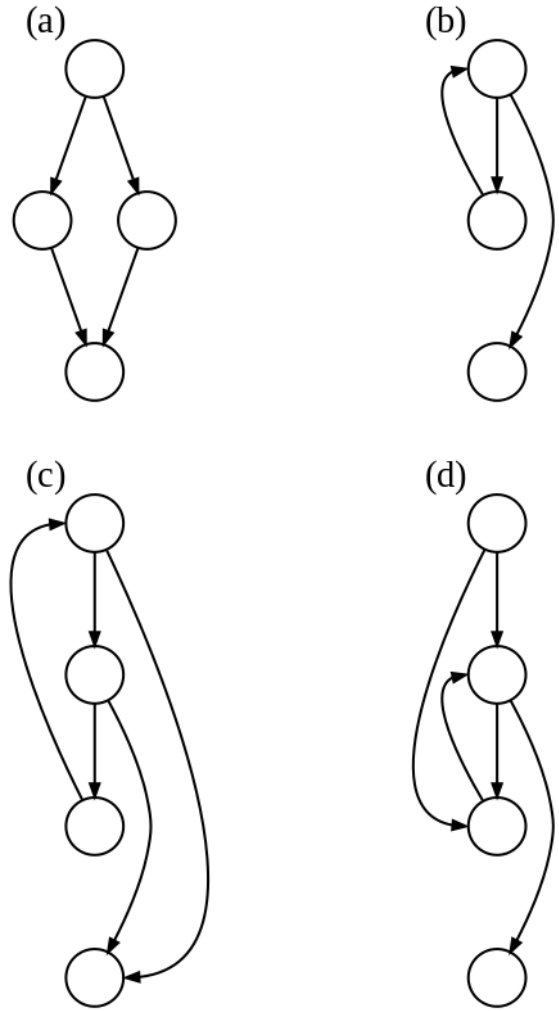
```c
int factorial(int val);

int main(int argc, char **argv) {
  return factorial( val: 2) * 7 == 42;
}

// precondition: val is non-negative
int factorial(int val) {
  if (val == 0) {
    return 1;
  }

  return val * factorial( val: val - 1);
}
```



Call graph

factorial1.c

Frances Elizabeth Allen
(1932 $\sim$ 2020; 2006 Turing Award)



(Intra-procedure) Control Flow Graph (CFG)

Control Flow Graph (CFG)

Definition (CFG)

Each node represents a *basic block*, i.e. a straight-line code sequence with no **branches/jumps** in except to the entry point and no **branches/jumps** out except at the exit point.

Control Flow Graph (CFG)

Definition (CFG)

Each node represents a *basic block*, i.e. a straight-line code sequence with no **branches/jumps** in except to the entry point and no **branches/jumps** out except at the exit point.

Jump targets start a block, and jumps end a block.

Control Flow Graph (CFG)

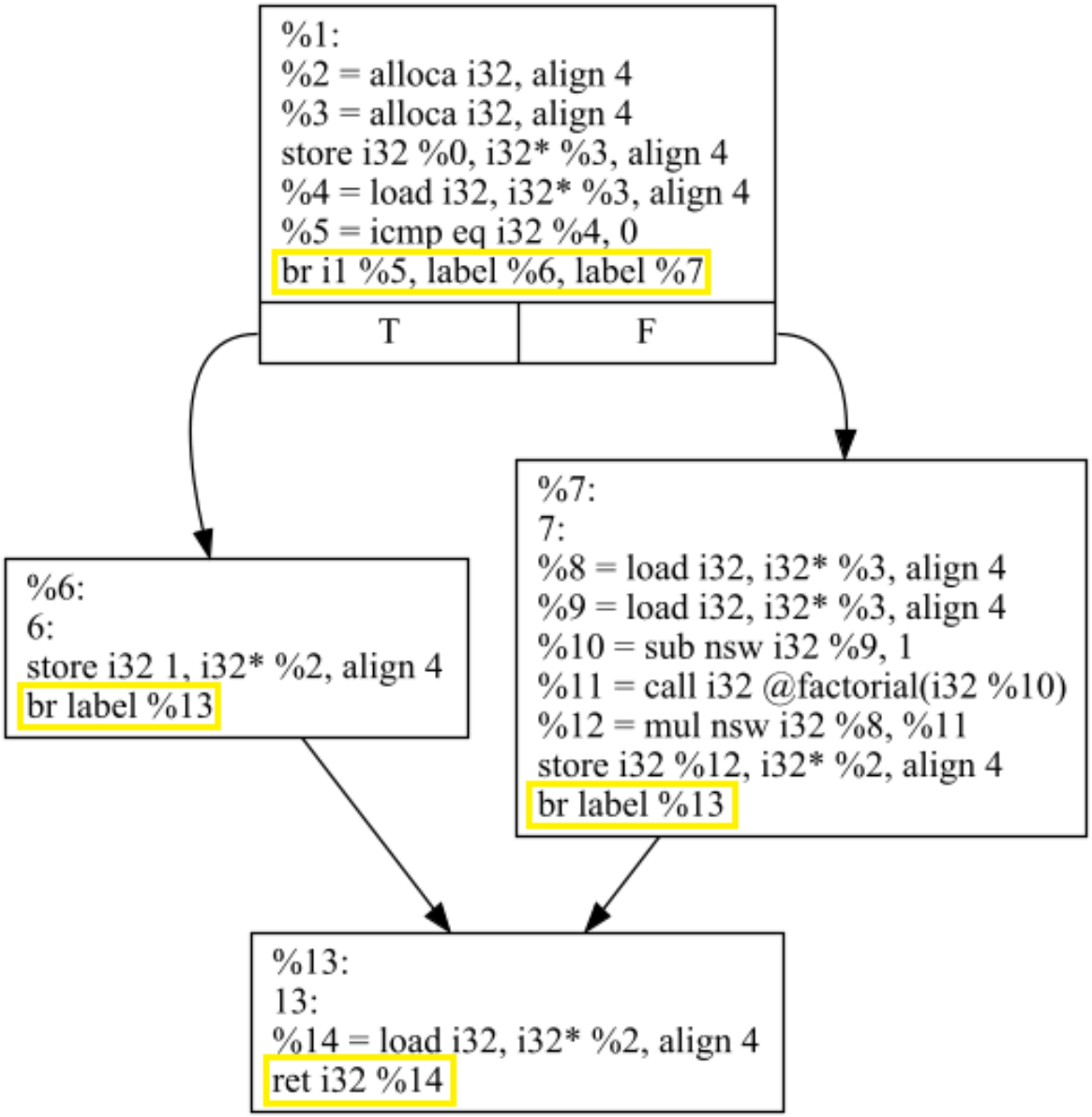
Definition (CFG)

Each node represents a *basic block*, i.e. a straight-line code sequence with no **branches/jumps** in except to the entry point and no **branches/jumps** out except at the exit point.

Jump targets start a block, and jumps end a block.

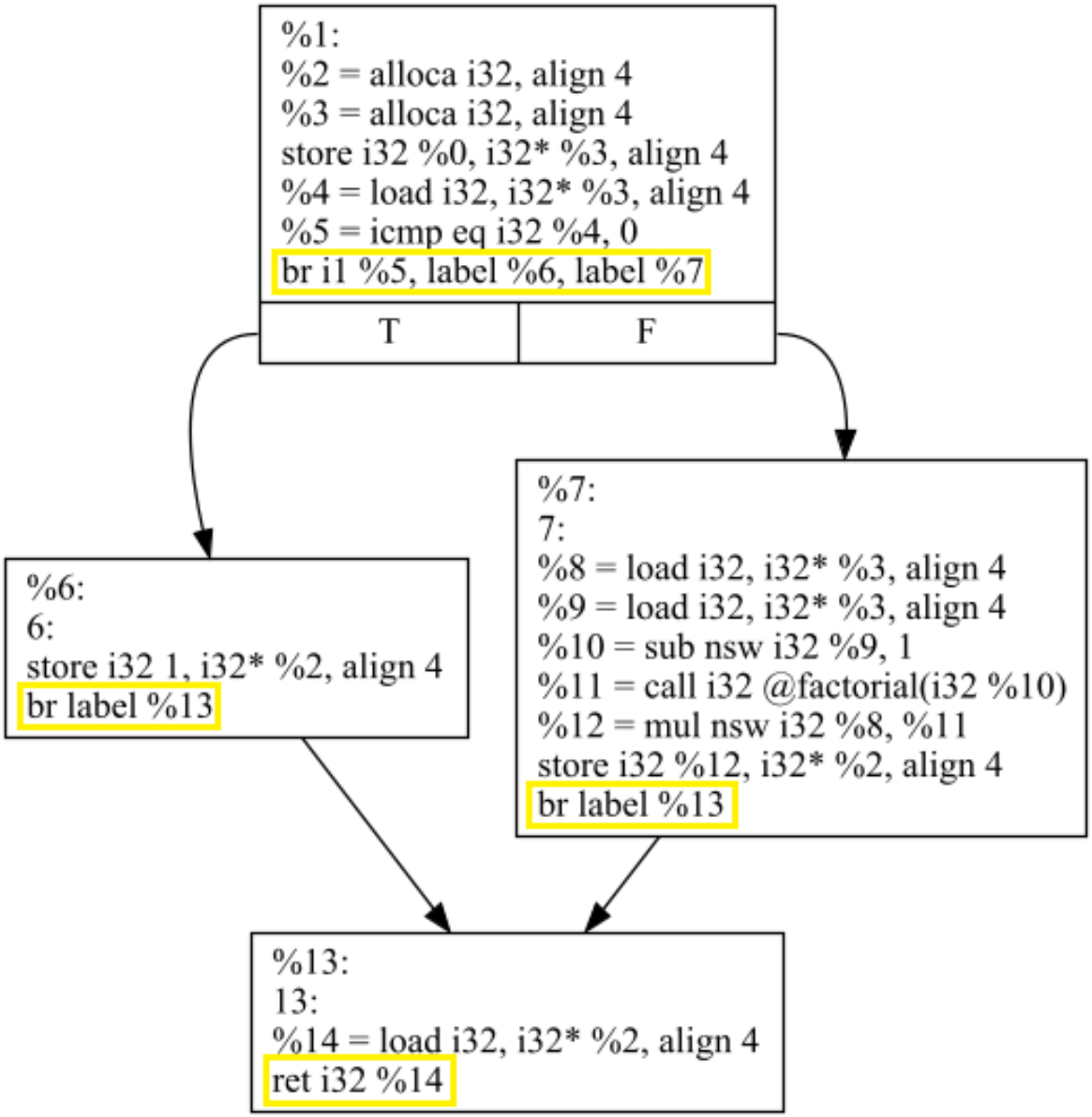
Directed edges are used to represent jumps in the control flow.

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

  return val * factorial( val: val - 1);
}
```



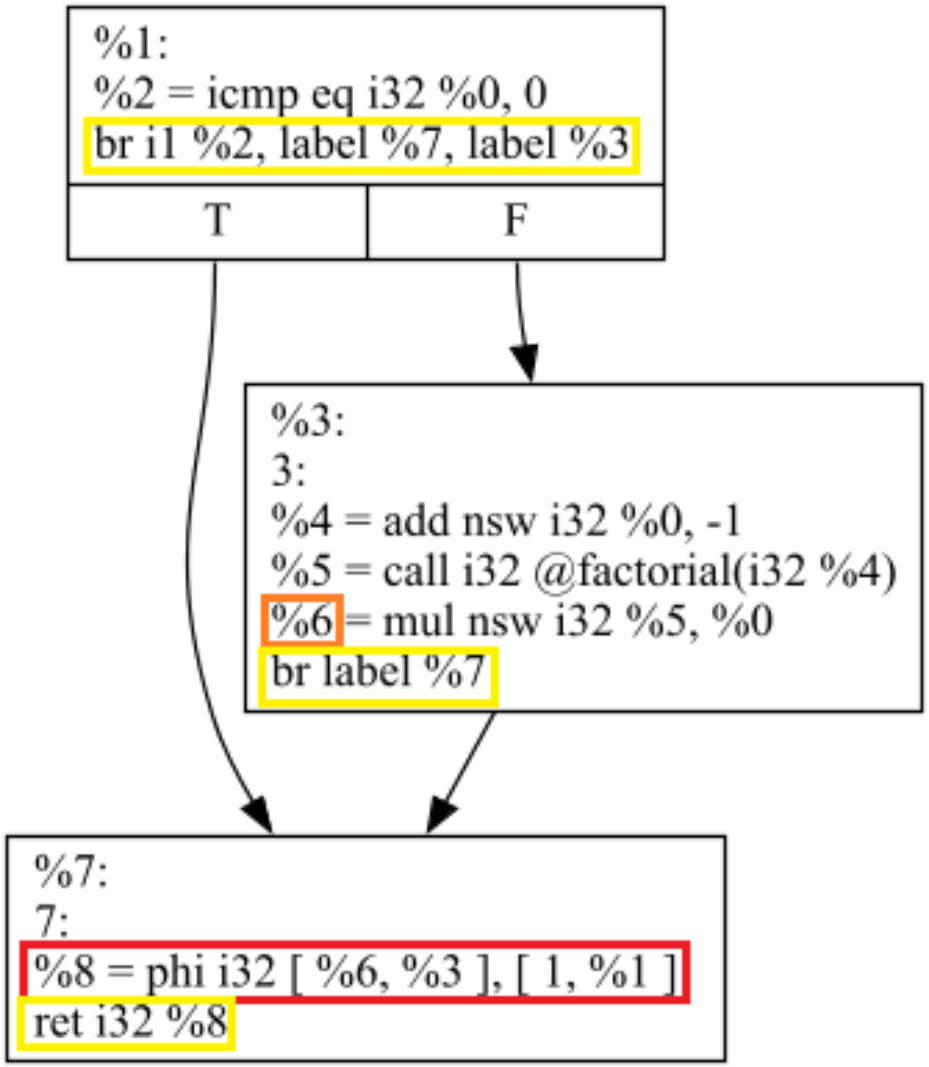
CFG for 'factorial' function

%2: store the return value (in different branches)



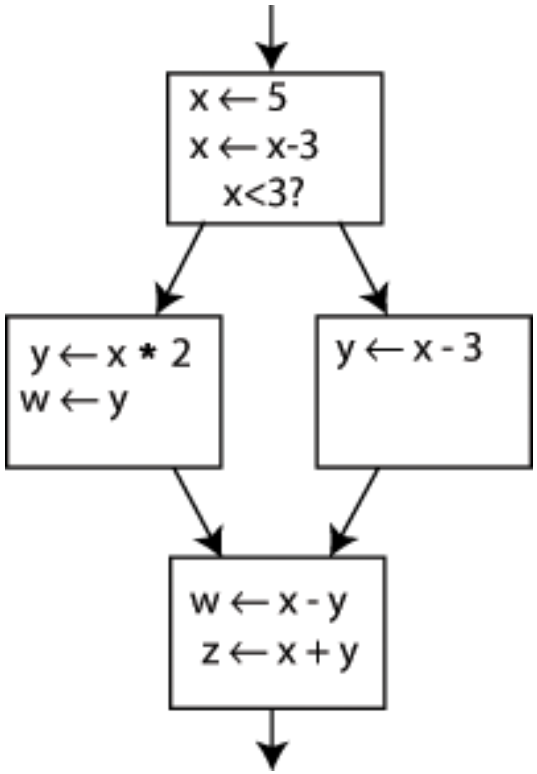
CFG for 'factorial' function

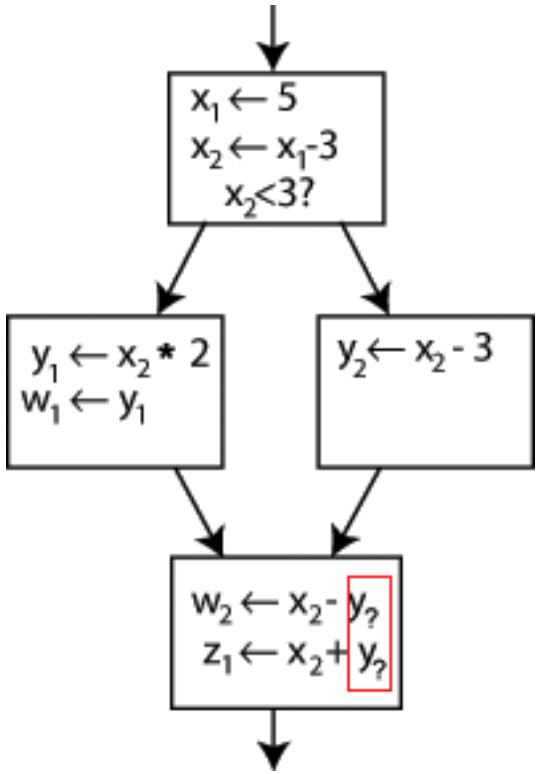
- Instruction Reference
  - Terminator Instructions
    - ‘`ret`’ Instruction
    - ‘`br`’ Instruction
    - ‘`switch`’ Instruction
    - ‘`indirectbr`’ Instruction
    - ‘`invoke`’ Instruction
    - ‘`callbr`’ Instruction
    - ‘`resume`’ Instruction
    - ‘`catchswitch`’ Instruction
    - ‘`catchret`’ Instruction
    - ‘`cleanupret`’ Instruction
    - ‘`unreachable`’ Instruction

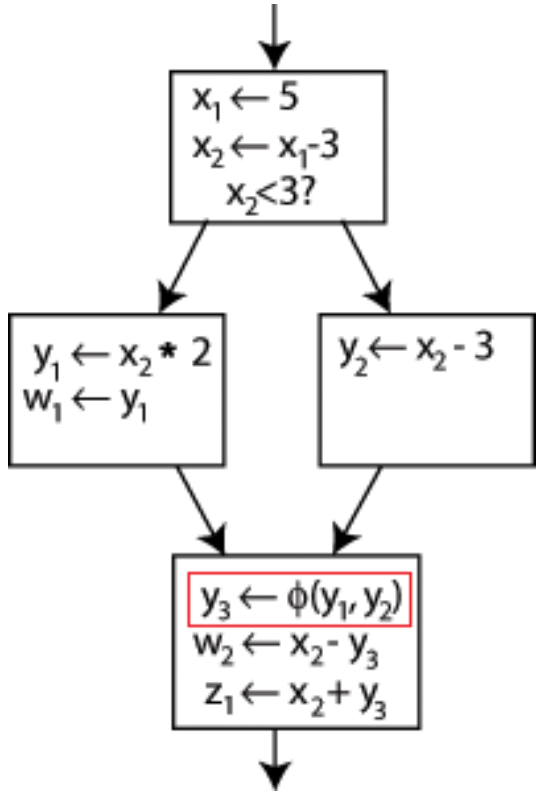


CFG for 'factorial' function

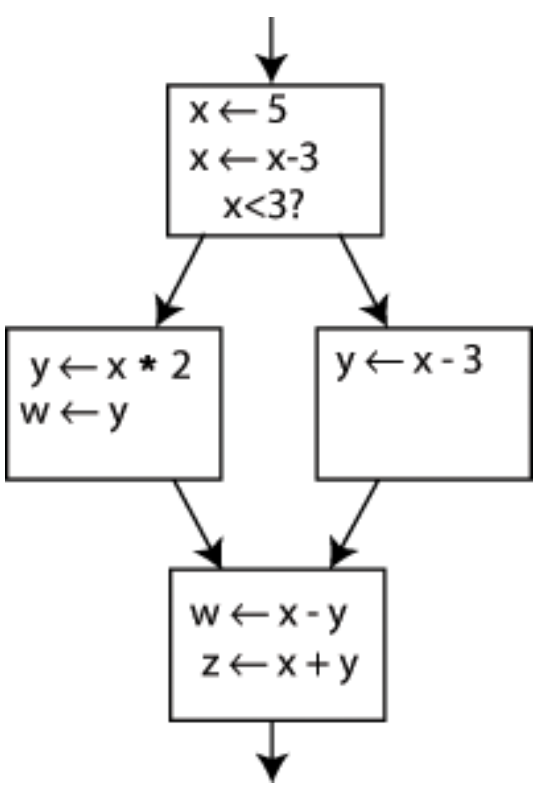
ϕ 根据控制流决定选择 y_1 还是 y_2

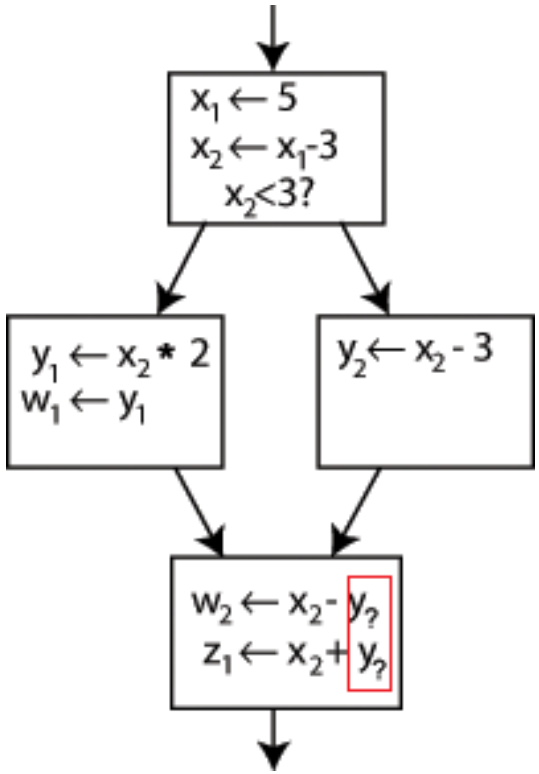


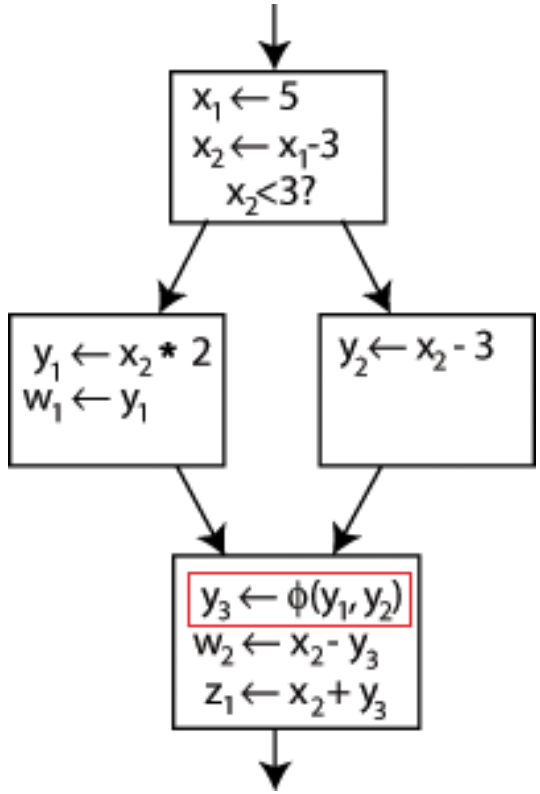




ϕ 根据控制流决定选择 y_1 还是 y_2

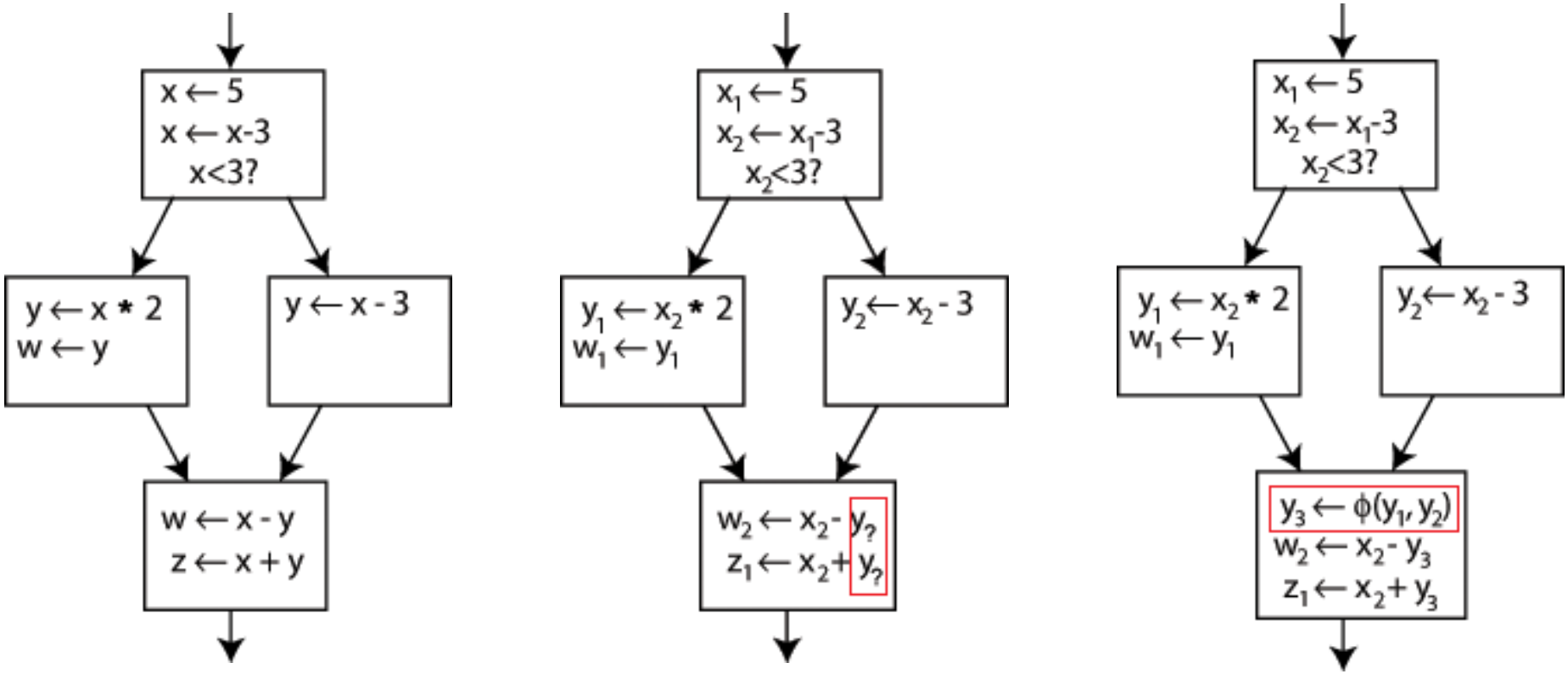






How to implement ϕ instruction?

ϕ 根据控制流决定选择 y_1 还是 y_2



How to implement ϕ instruction?

基本思想: 将 ϕ 指令转换成若干赋值指令, 上推至前驱基本块中

SSA 形式的构建与消去

Section 4.3

SSA 形式的构建与消去

Section 4.3

Section 9.3

SSA 形式的构建与消去

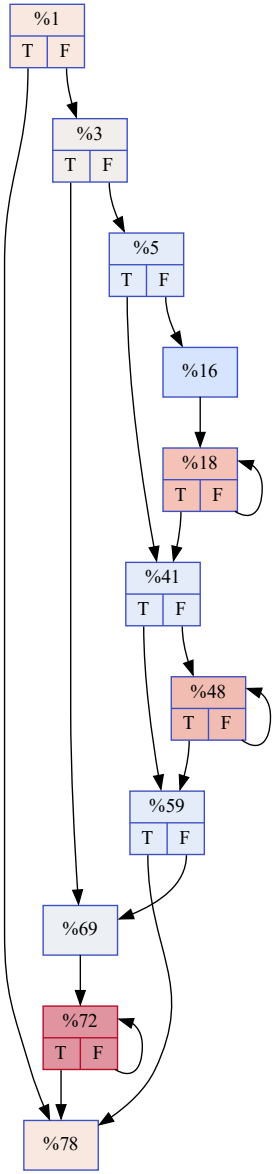
Section 4.3

Efficiently Computing Static Single Assignment Form and the Control Dependence Graph

RON CYTRON, JEANNE FERRANTE, BARRY K. ROSEN, and MARK N. WEGMAN
IBM Research Division
and
F. KENNETH ZADECK
Brown University

TOPLAS1991 @
compilers-papers-we-love

Section 9.3

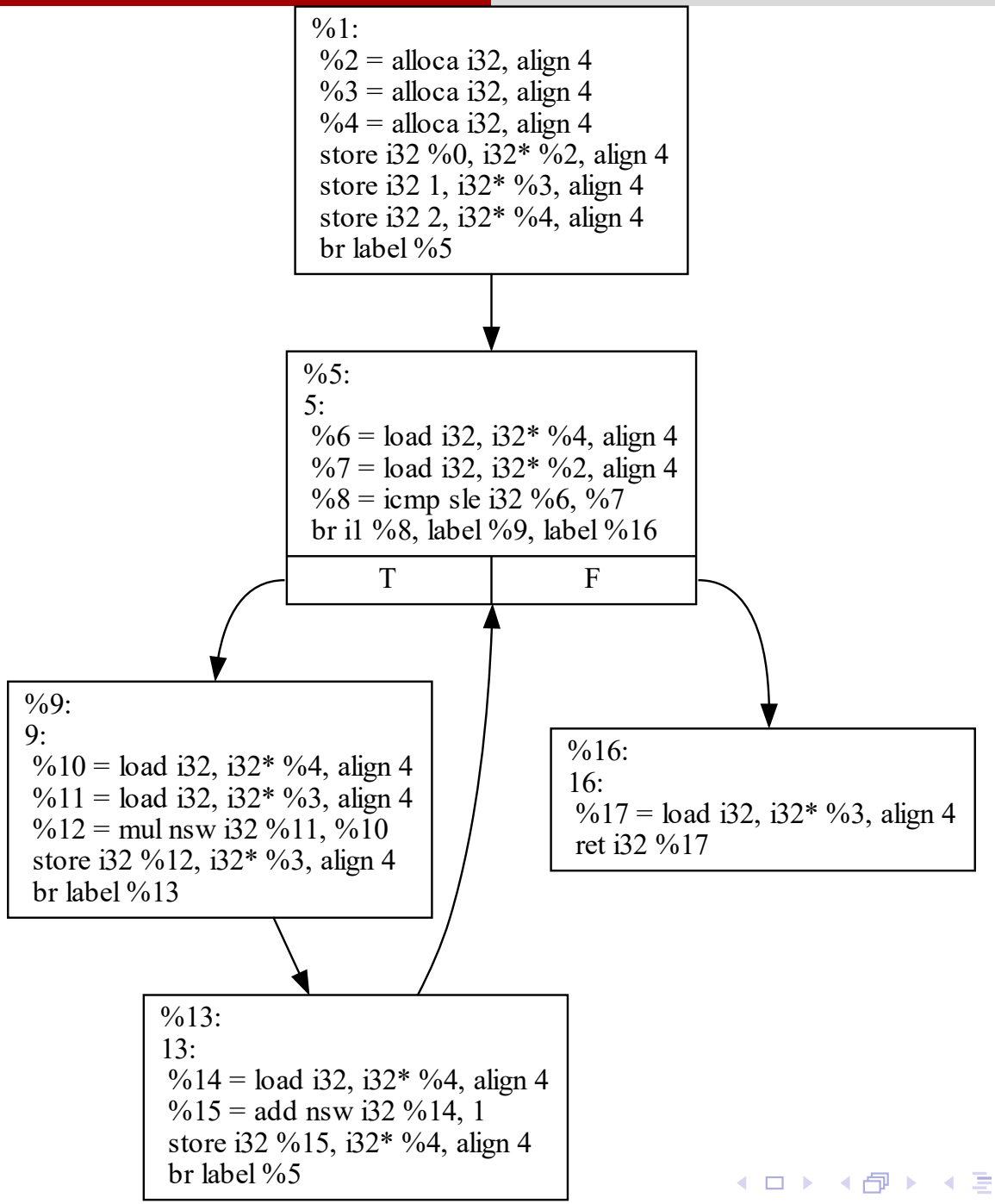


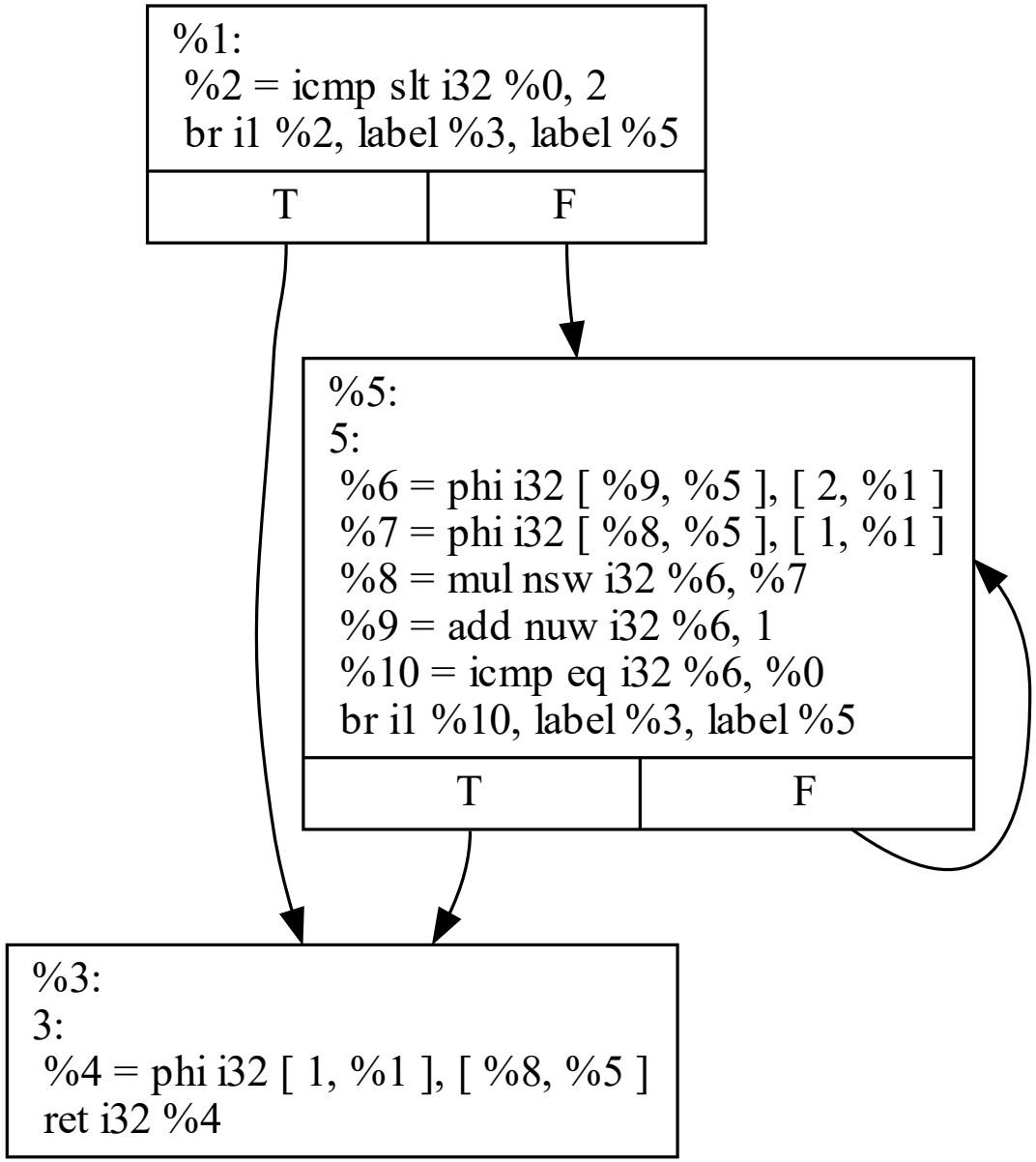
CFG for 'factorial' function

factorial1 (opt3)

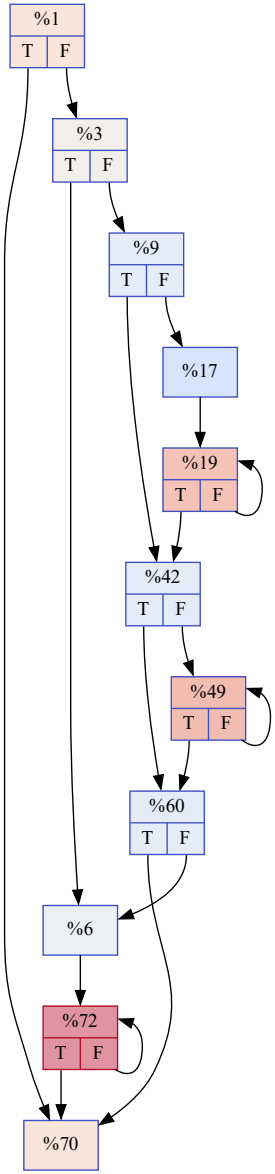
```c
int factorial(int val) {
  int temp = 1;

  for (int i = 2; i <= val; i++) {
    temp *= i;
  }

  return temp;
}
```

factorial2.c


%1:
%2 = alloca i32, align 4
%3 = alloca i32, align 4
%4 = alloca i32, align 4
store i32 %0, i32* %2, align 4
store i32 1, i32* %3, align 4
store i32 2, i32* %4, align 4
br label %5
%5:
5:
%6 = load i32, i32* %4, align 4
%7 = load i32, i32* %2, align 4
%8 = icmp sle i32 %6, %7
br i1 %8, label %9, label %16
T
F
%9:
9:
%10 = load i32, i32* %4, align 4
%11 = load i32, i32* %3, align 4
%12 = mul nsw i32 %11, %10
store i32 %12, i32* %3, align 4
br label %13
%16:
16:
%17 = load i32, i32* %3, align 4
ret i32 %17
%13:
13:
%14 = load i32, i32* %4, align 4
%15 = add nsw i32 %14, 1
store i32 %15, i32* %4, align 4
br label %5



CFG for 'factorial' function



CFG for 'factorial' function

factorial2 (opt3)


Q
A

- 为什么基本块的中间某条指令可以是 `call` 指令?

- 为什么基本块的中间某条指令可以是 `call` 指令?
- 一个函数里可以包含多个含有 `ret` 指令的基本块吗?

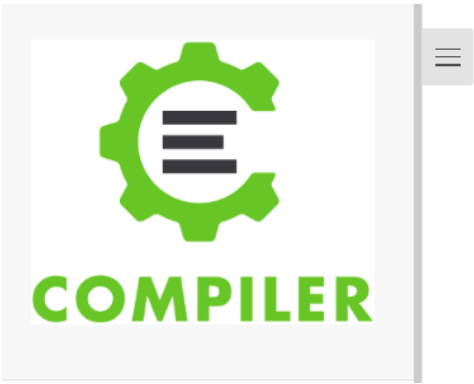
- 为什么基本块的中间某条指令可以是 `call` 指令?
- 一个函数里可以包含多个含有 `ret` 指令的基本块吗?
- 如果可以, 那不就可以解决 ϕ 指令相关的问题了吗?

- 为什么基本块的中间某条指令可以是 `call` 指令?
- 一个函数里可以包含多个含有 `ret` 指令的基本块吗?
- 如果可以, 那不就可以解决 ϕ 指令相关的问题了吗?
- 为什么是 “`%0，%1`” 之后是 “`%3`”?

如何用编程的方式生成 LLVM IR?

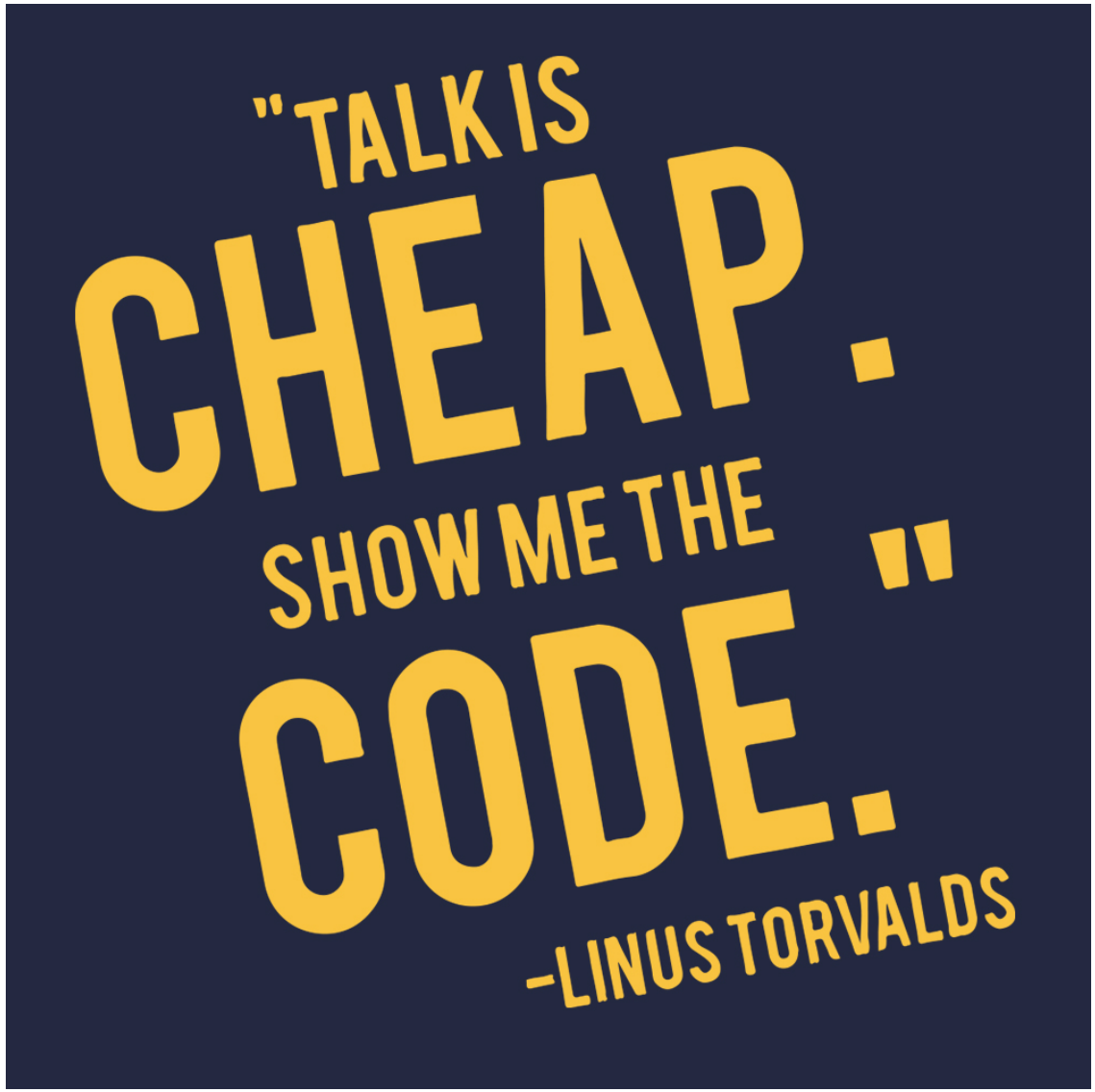
Bytedeco/javacpp @ github

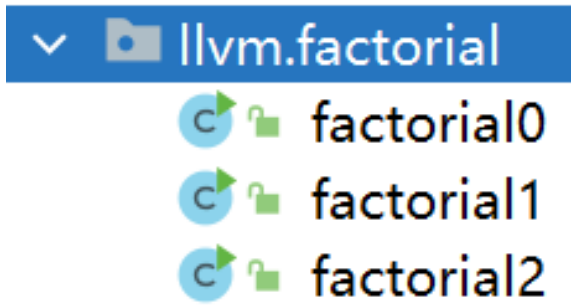
JavaCPP Presets Platform For LLVM



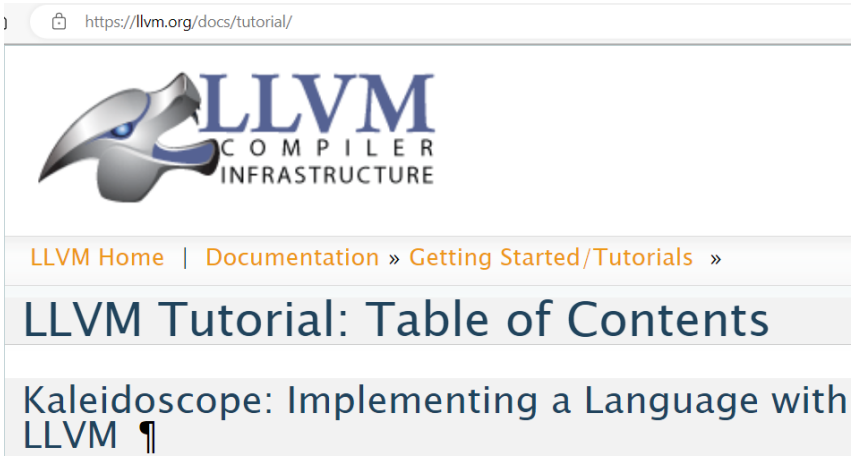
LLVM JAVA API使用手册

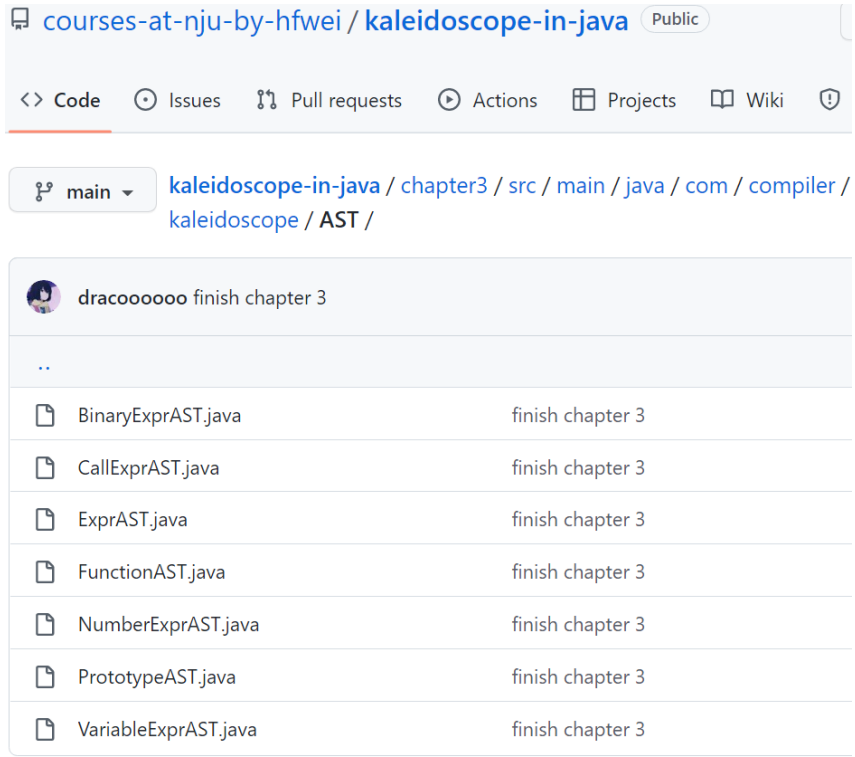
准备工作


"TALK IS
CHEAP.
SHOW ME THE
CODE."
-LINUS TORVALDS

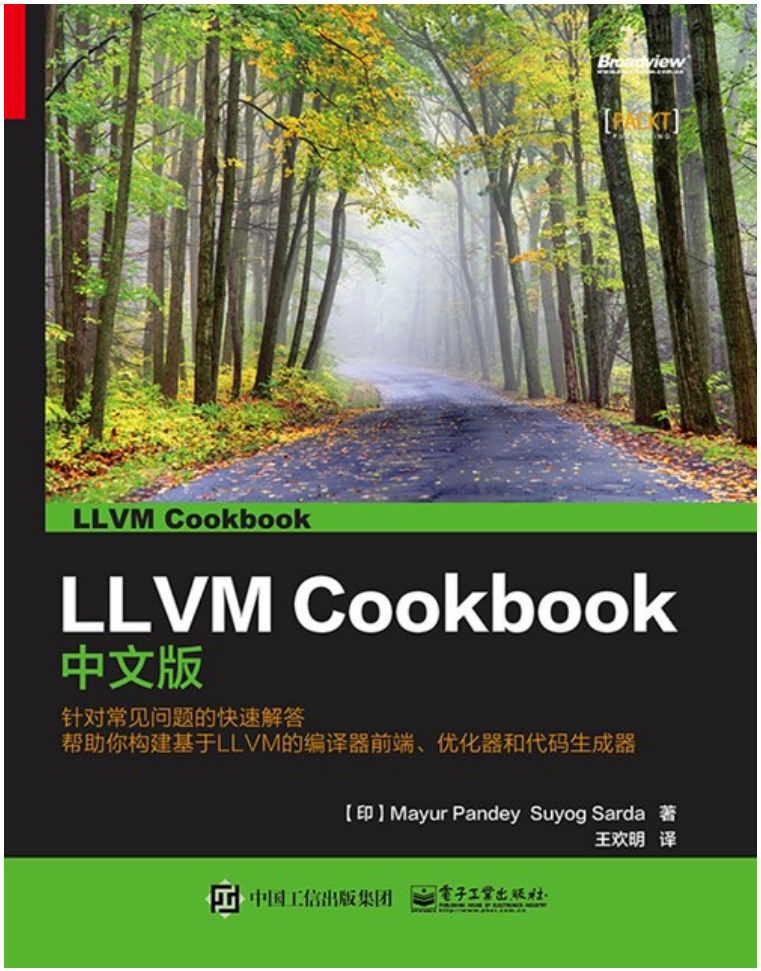

llvm.factorial
factorial0
factorial1
factorial2

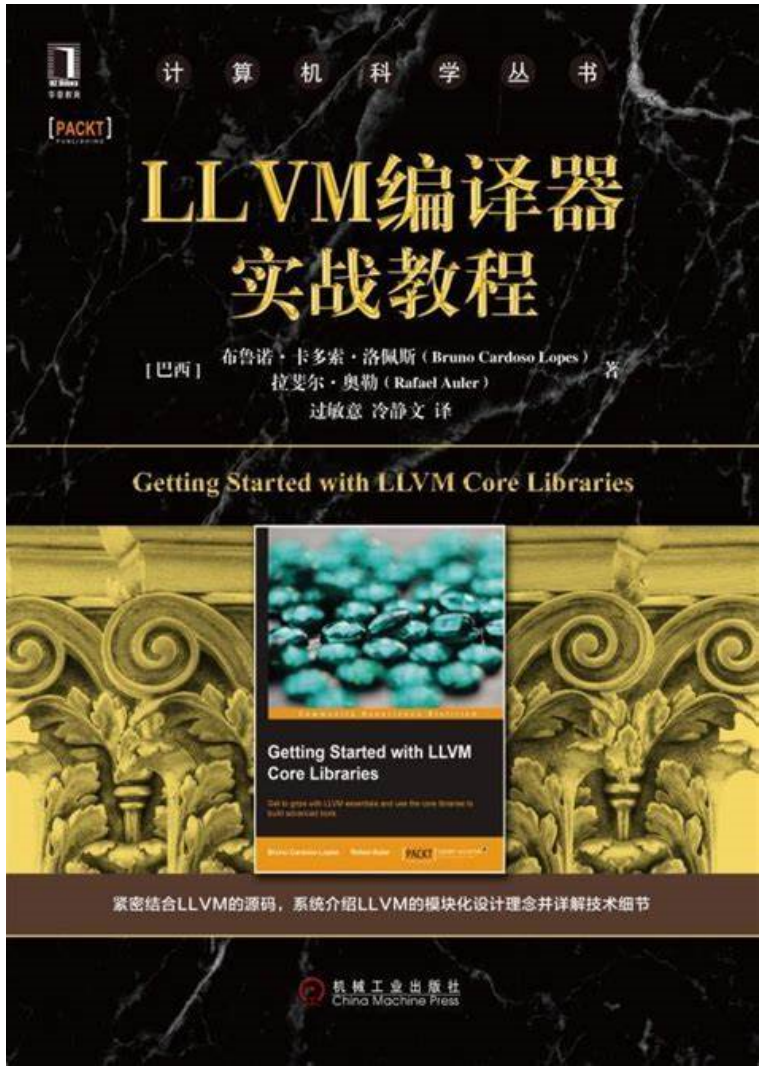
Kaleidoscope: Implementing a Language with LLVM



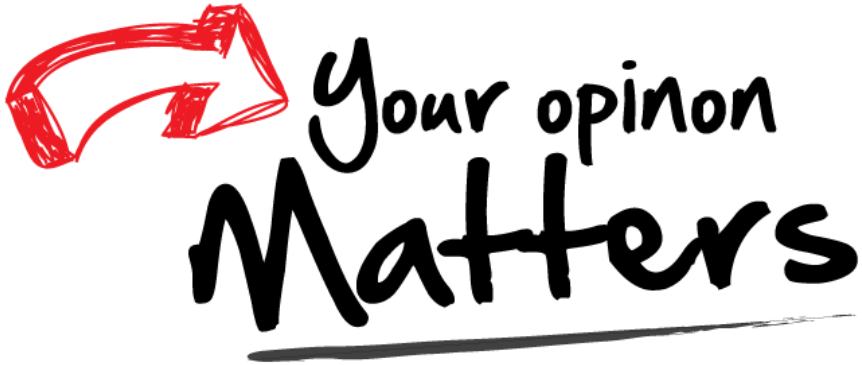


kaleidoscope-in-java@github


LLVM Cookbook
LLVM Cookbook
中文版
针对常见问题的快速解答
帮助你构建基于LLVM的编译器前端、优化器和代码生成器
[印] Mayur Pandey Suyog Sarda 著
王欢明 译
中国工信出版集团


计 算 机 科 学 丛 书
LLVM编译器
实战教程
[巴西] 布鲁诺·卡多索·洛佩斯（Bruno Cardoso Lopes）
拉斐尔·奥勒（Rafael Auler）著
过敏意 冷静文 译
Getting Started with LLVM Core Libraries
Getting Started with LLVM Core Libraries
紧密结合LLVM的源码，系统介绍LLVM的模块化设计理念并详解技术细节
机械工业出版社
China Machine Press


Thank
You!



Office 926

hfwei@nju.edu.cn