

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：嵇茜、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/>

https://llvm.org



The LLVM Compiler Infrastructure

LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

Latest LLVM Release!

1 November 2022: LLVM 15.0.4 is now available for download! LLVM is publicly available under an open source license. Also, you might want to check

“Low Level Virtual Machine”



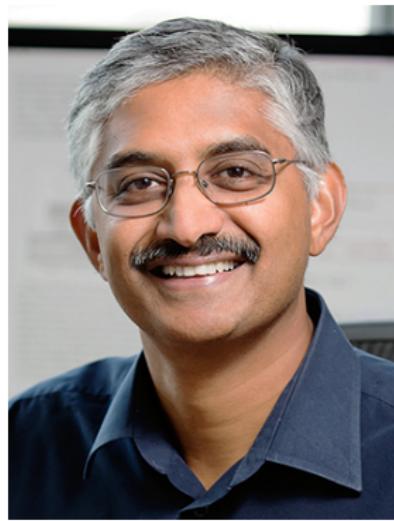


Year	Project
2021	CompCert
2020	Berkeley DB
2019	DNS
2018	Wireshark
2017	Project Jupyter
2016	Andrew File System
2015	GCC
2014	Mach
2013	Coq
2012	LLVM
2011	Eclipse

ACM Software System Award



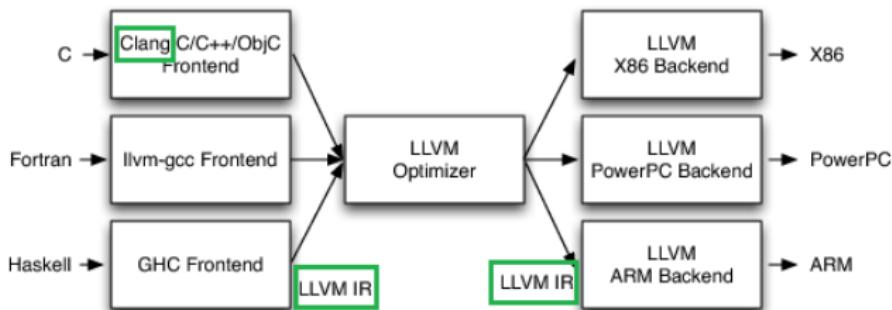
Chris Lattner (1978); UIUC 2020



Vikram Adve (1966)



LLVM IR (Intermediate Representation)

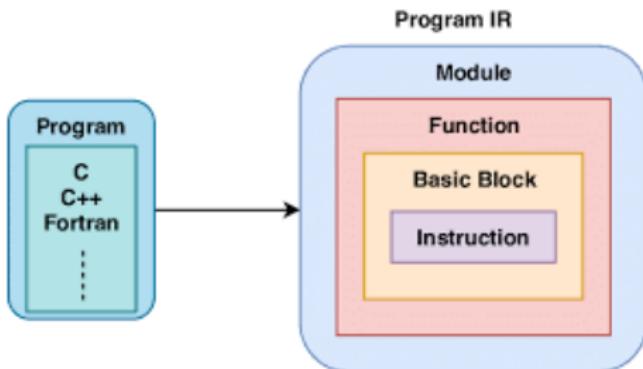


“IR 设计的优秀与否决定着整个编译器的好坏”



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

```
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
```

Three Address Code (TAC)

Static Single Assignment (SSA)

```
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
```

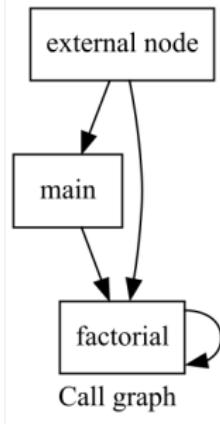
```
int factorial(int val);

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;
    }

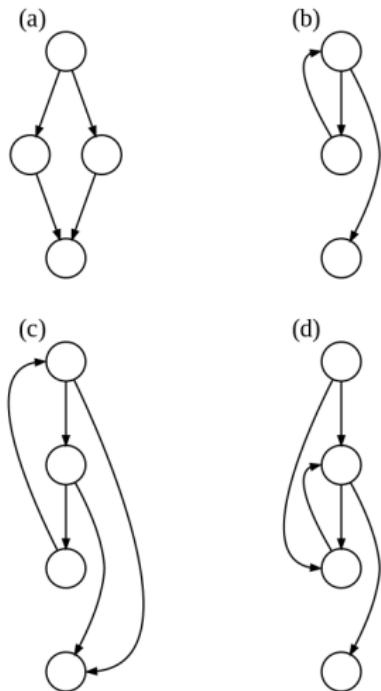
    return val * factorial(val: val - 1);
}
```

factorial1.c





Frances Elizabeth Allen
(1932 ~ 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)

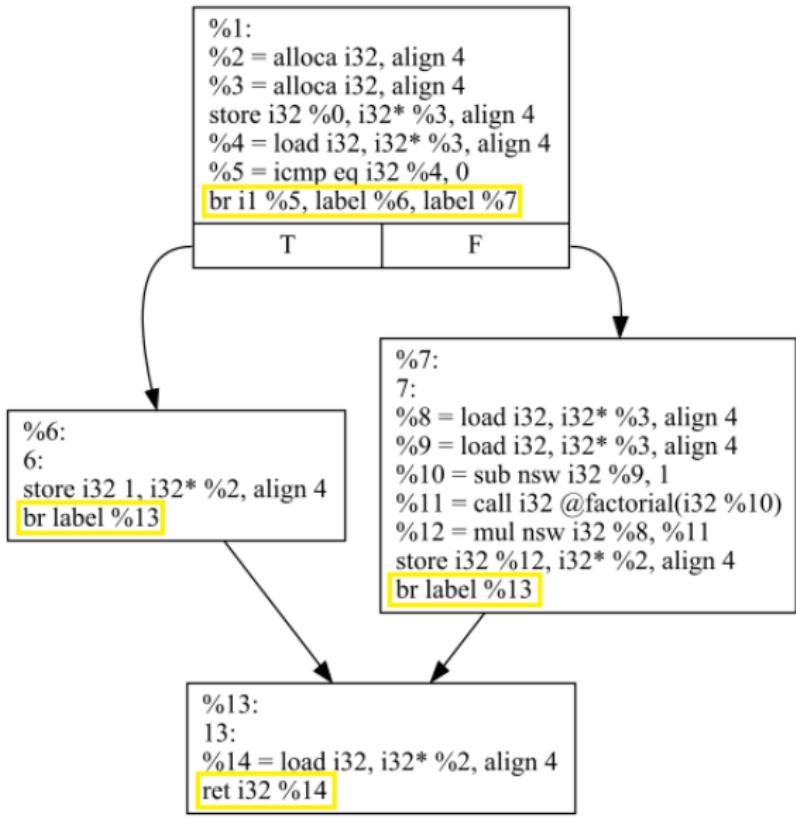
Definition (CFG)

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.

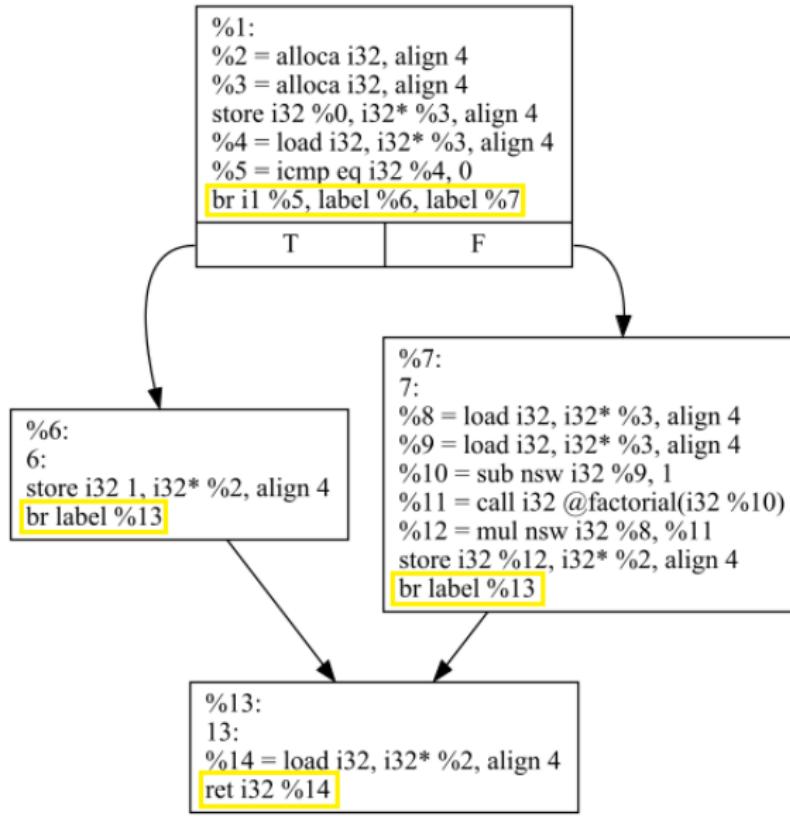
Directed **edges** are used to represent jumps in the control flow.

```
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)

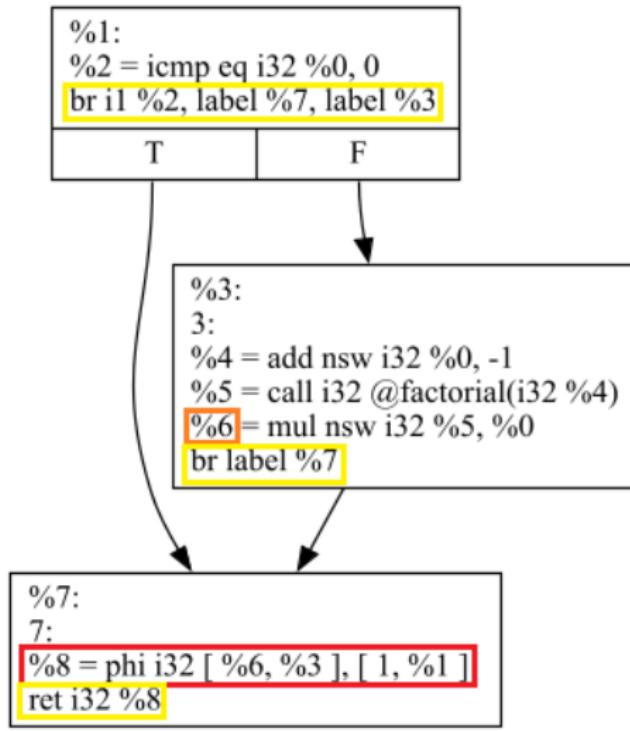


CFG for 'factorial' function

Instruction Reference

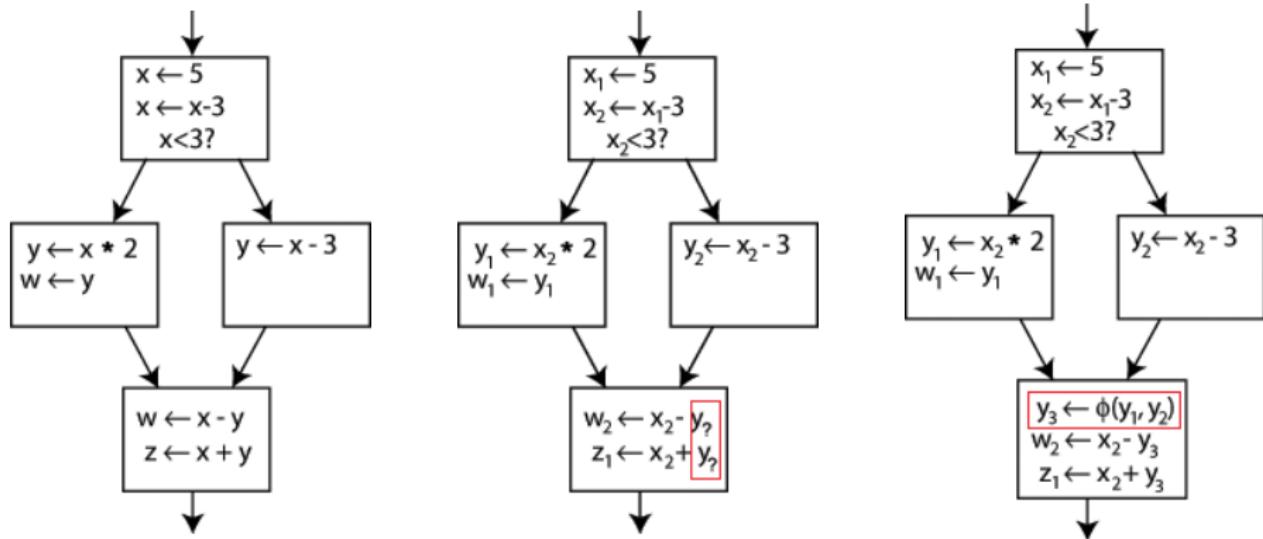
- Terminator Instructions

- ‘ret’ Instruction
- ‘br’ Instruction
- ‘switch’ Instruction
- ‘indirectbr’ Instruction
- ‘invoke’ Instruction
- ‘callbr’ Instruction
- ‘resume’ Instruction
- ‘catchswitch’ Instruction
- ‘catchret’ Instruction
- ‘cleanupret’ Instruction
- ‘unreachable’ Instruction

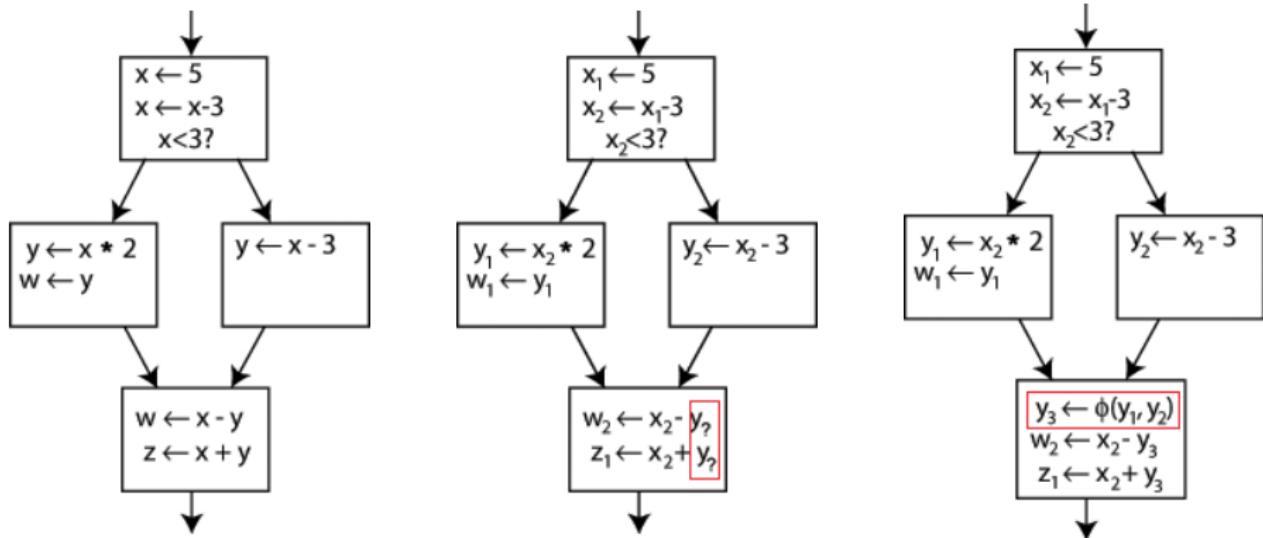


CFG for 'factorial' function

ϕ 根据控制流决定选择 y_1 还是 y_2

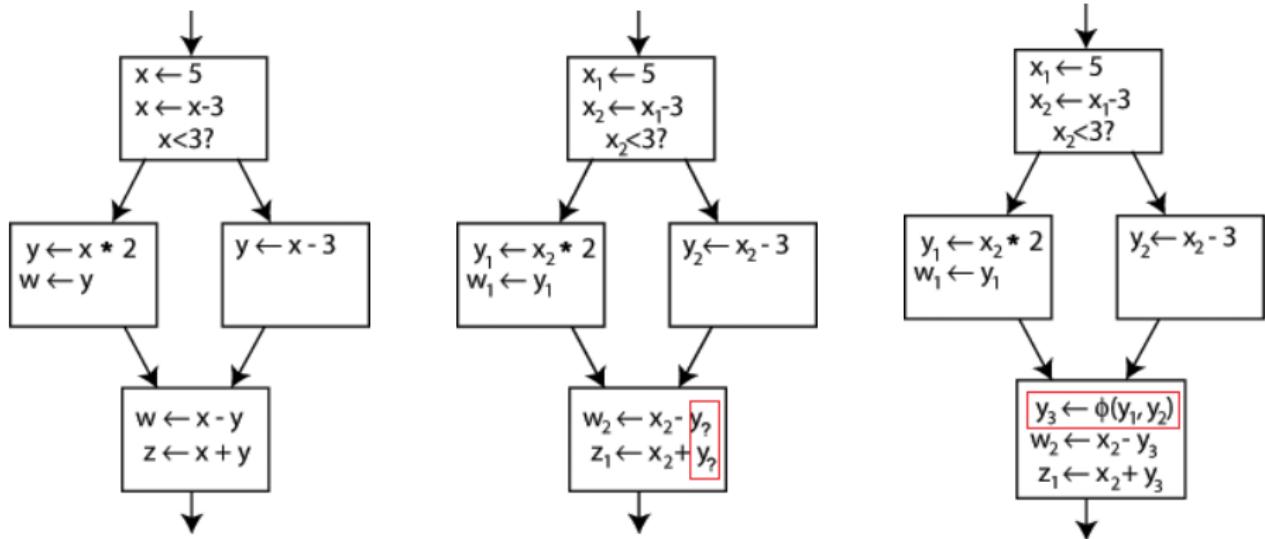


ϕ 根据控制流决定选择 y_1 还是 y_2



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?

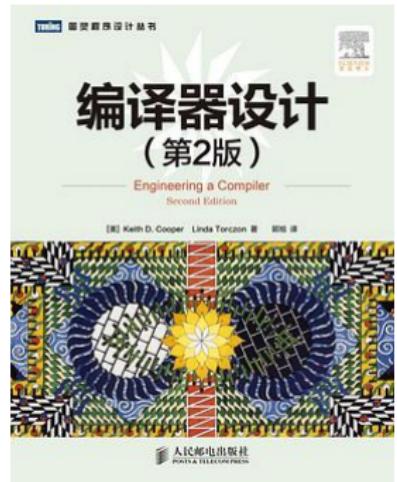
基本思想：将 ϕ 指令转换成若干赋值指令，上推至前驱基本块中

SSA 形式的构建与消去



Section 4.3

SSA 形式的构建与消去



Section 4.3

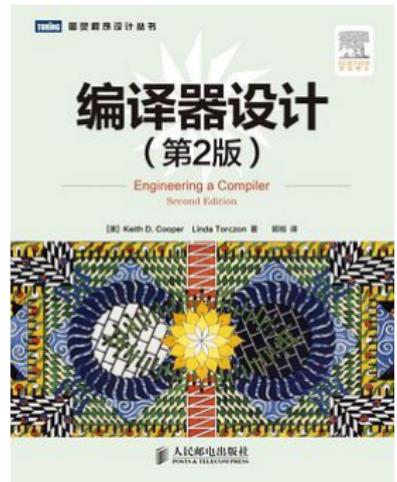
SSA 形式的构建与消去



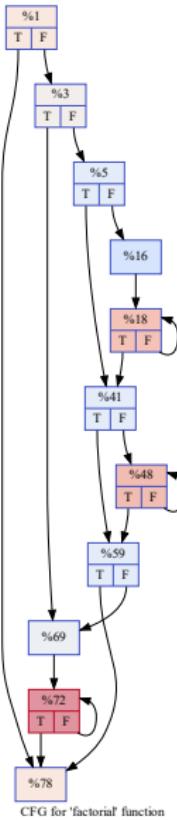
Efficiently Computing Static Single Assignment Form and the Control Dependence Graph

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



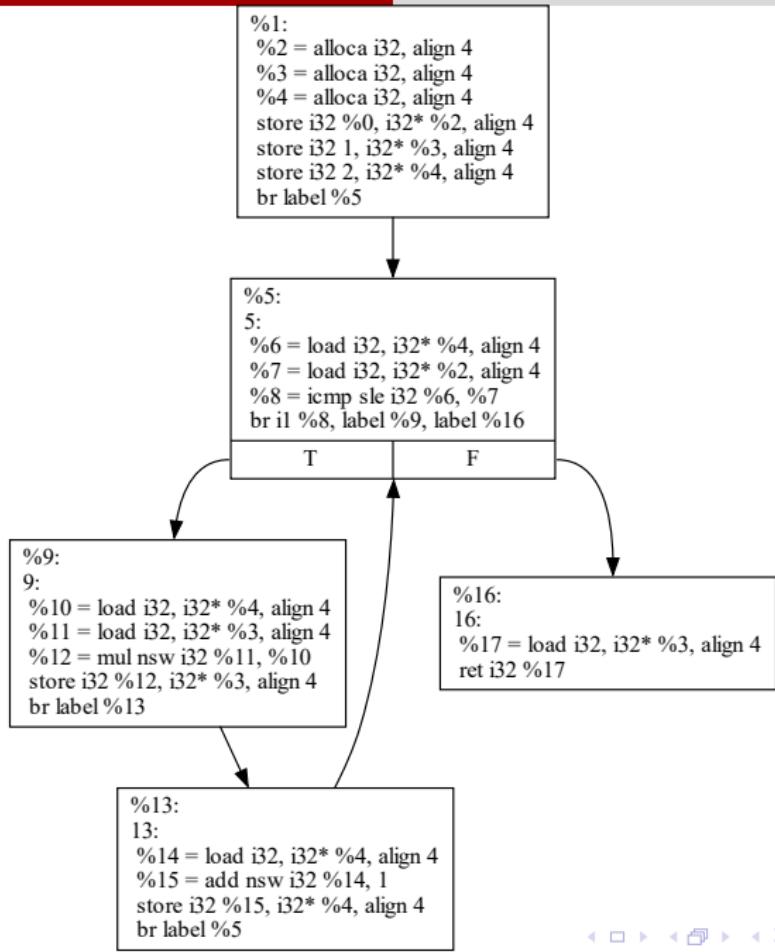
Section 4.3

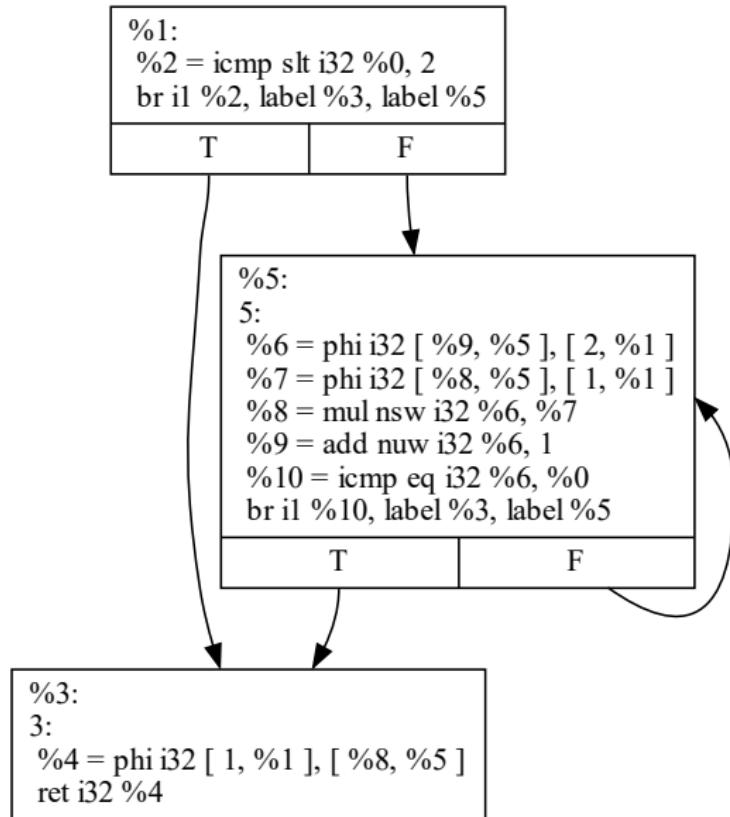


factorial1 (opt3)

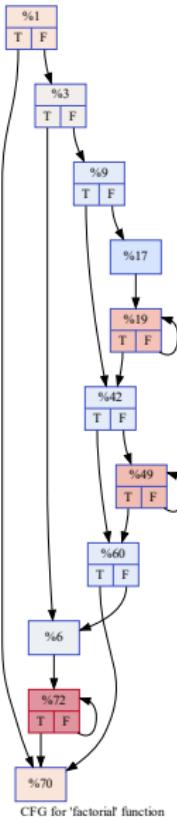
```
int factorial(int val) {  
    int temp = 1;  
  
    for (int i = 2; i <= val; i++) {  
        temp *= i;  
    }  
  
    return temp;  
}
```

factorial2.c





CFG for 'factorial' function



factorial2 (opt3)





- ▶ 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?
- ▶ 课程实验中具体如何生成 ϕ 指令? “课上不讲, 课后自学” 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?
- ▶ 课程实验中具体如何生成 ϕ 指令? “课上不讲, 课后自学” 吗?
- ▶ 为什么是 “%0, %1” 之后是 “%3”?

LLVM Language Reference Manual

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



[LLVM Home](#) | [Documentation](#) » [Reference](#) »

LLVM Language Reference Manual

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



Kaleidoscope: Implementing a Language with LLVM

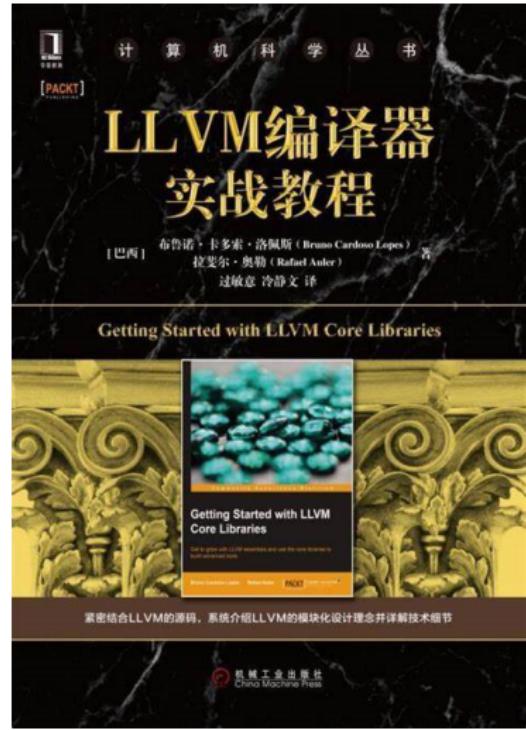
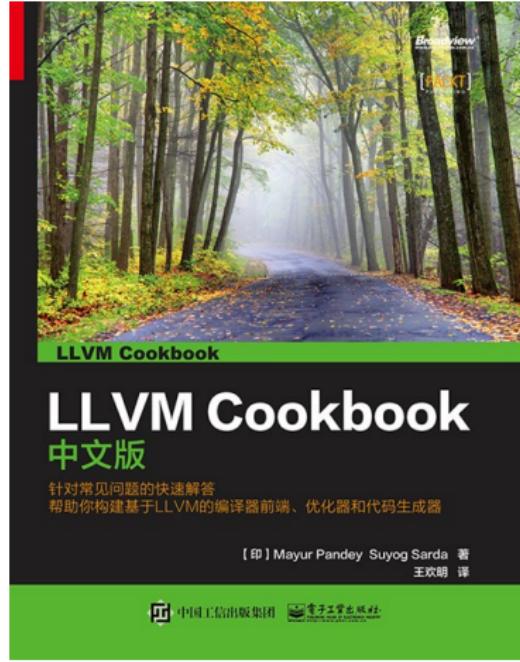
The screenshot shows the LLVM Documentation website at <https://llvm.org/docs/tutorial/>. The main navigation bar includes 'LLVM Home', 'Documentation', 'Getting Started/Tutorials', and 'Table of Contents'. The 'Table of Contents' section is currently selected, displaying the following items:

- Kaleidoscope: Implementing a Language with LLVM

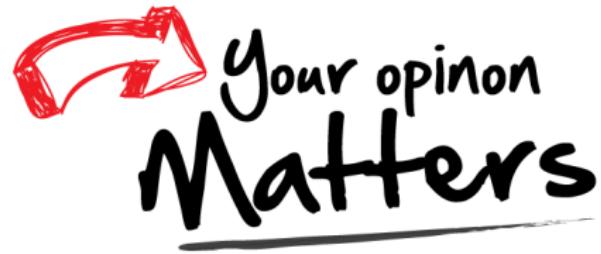
The screenshot shows a GitHub repository page for [courses-at-nju-by-hfwei / kaleidoscope-in-java](#). The repository is public and contains code for the Kaleidoscope project. A pull request titled "finish chapter 3" has been merged. The commit message is "finish chapter 3". The pull request details show the following files and their status:

File	Status
BinaryExprAST.java	finish chapter 3
CallExprAST.java	finish chapter 3
ExprAST.java	finish chapter 3
FunctionAST.java	finish chapter 3
NumberExprAST.java	finish chapter 3
PrototypeAST.java	finish chapter 3
VariableExprAST.java	finish chapter 3

kaleidoscope-in-java@github



Thank You!



Office 926

hfwei@nju.edu.cn