Check-In

Review - SSA

Put the following code into SSA form

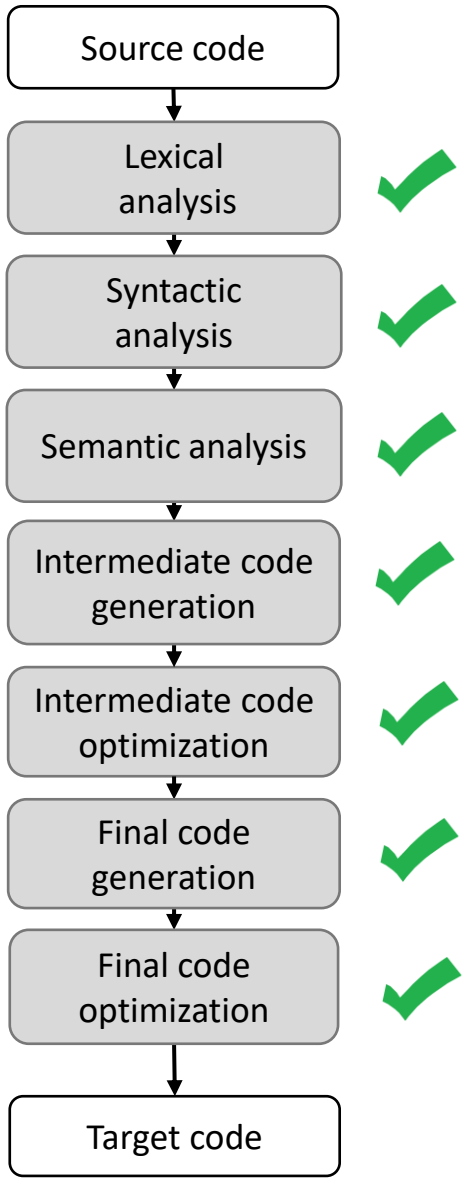
```
int foo(int arg){
  int a;
  a = 2;
  while (arg < 2){
    a = a * 3;
    arg = arg + 1;
  }
  return a;
}
```

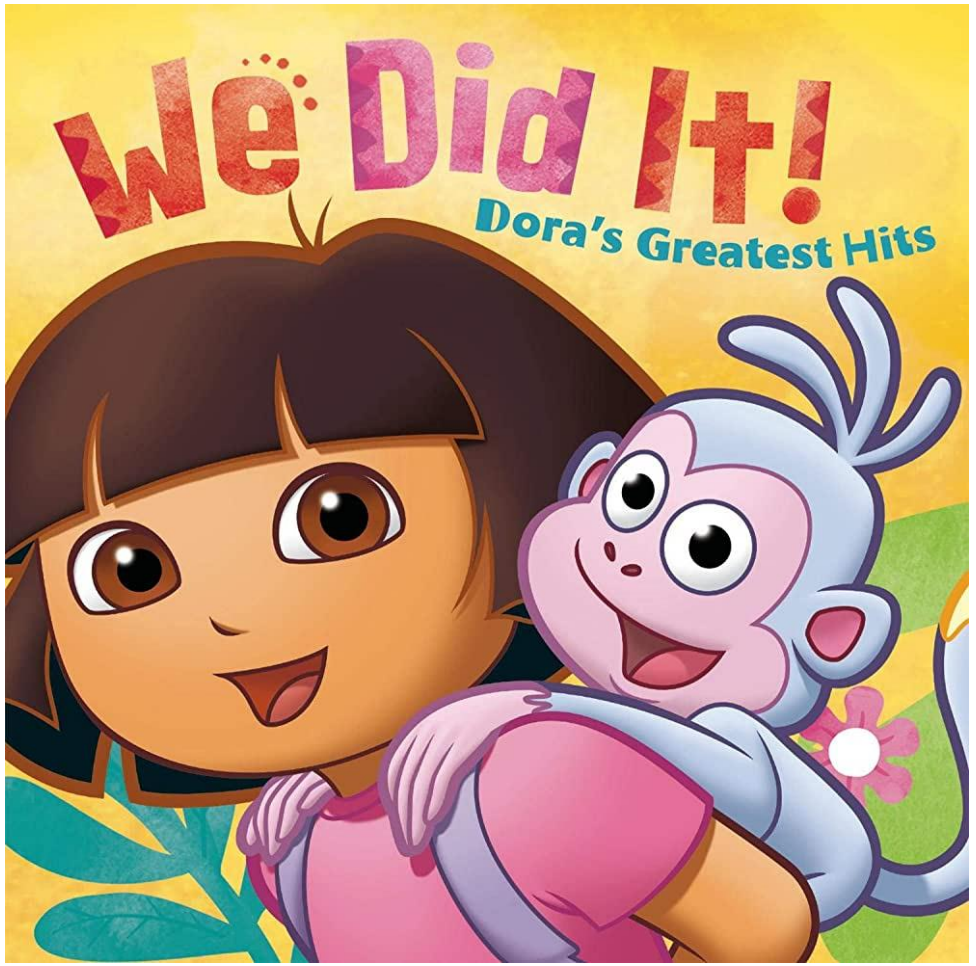
Announcements

Administrivia

Compiler Construction

Progress Pics






EECS 665
COMPILER
CONSTRUCTED

What Next?

Scheduling

We “Finished Early” by design

- Meets my teaching goal of testing material only after it’s been taught
- Hopefully “front-loads” your schedule

Still a lot to learn about compilers!

- Some topics benefit from knowledge of the workflow
- Some topics are just fun/useful and “compilery”

University of Kansas | Drew Davidson



LLVM

Previously…

SSA

Static Single Assignment

- Motivation
- Concept
- Importance
- Implementation

<u>You should know</u>

- How to put a program into SSA form
- The advantages of SSA
- Dominators



Optimization


COMPILER
Postcompilation Pit-stop
Codegen
COMPILER
LAND
The World of Sweets
Architecture
Sema Anal
Syntactic Definition
Syntax-Dir Translation
Advanced Topics
ICE CREAM SLOPES
Duke of Swirl
Regular Languages
Lexical Analysis

Lecture Outline

LLVM

Introducing LLVM
- History
- Features

Getting started with LLVM
- Installation
- Command-line tools

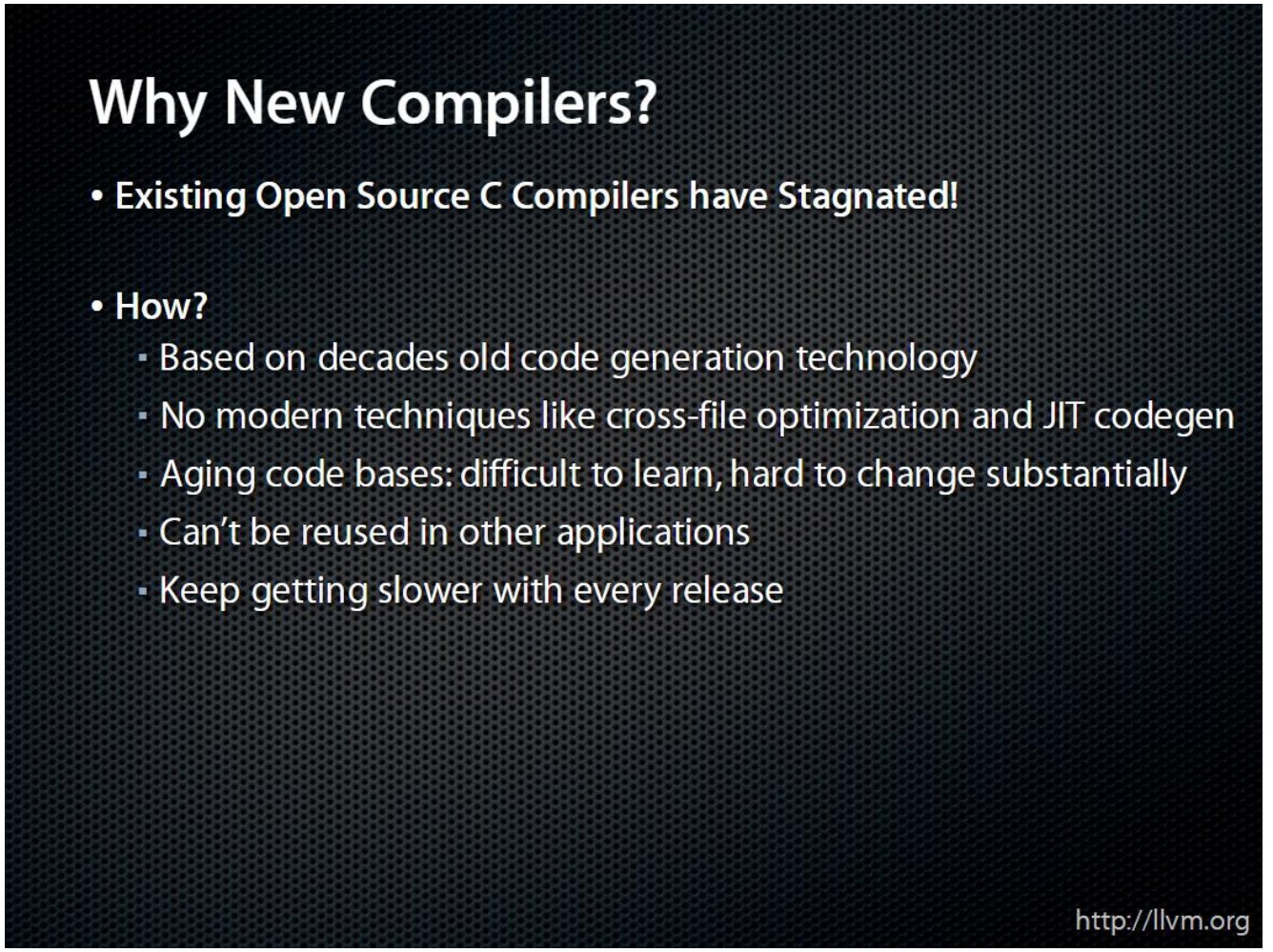
Getting More Out of LLVM
- Language development
- A Taste of bitcode
- Beyond codegen



Advanced Topics

Impetus

Introducing LLVM - History



- https://llvm.org/pubs/2008-10-04-ACAT-LLVM-Intro.pdf

LLVM Origin Story

Introducing LLVM - History

2000

- Began at UIUC under Vikram Adve
- Main developer was M.Sc student Chris Lattner

2005

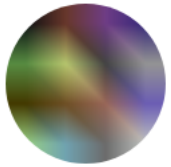
- Lattner graduates with PhD, hired by Apple
- Apple begins heavy contributions to LLVM

2013

- LLVM binaries considered competitive to GCC

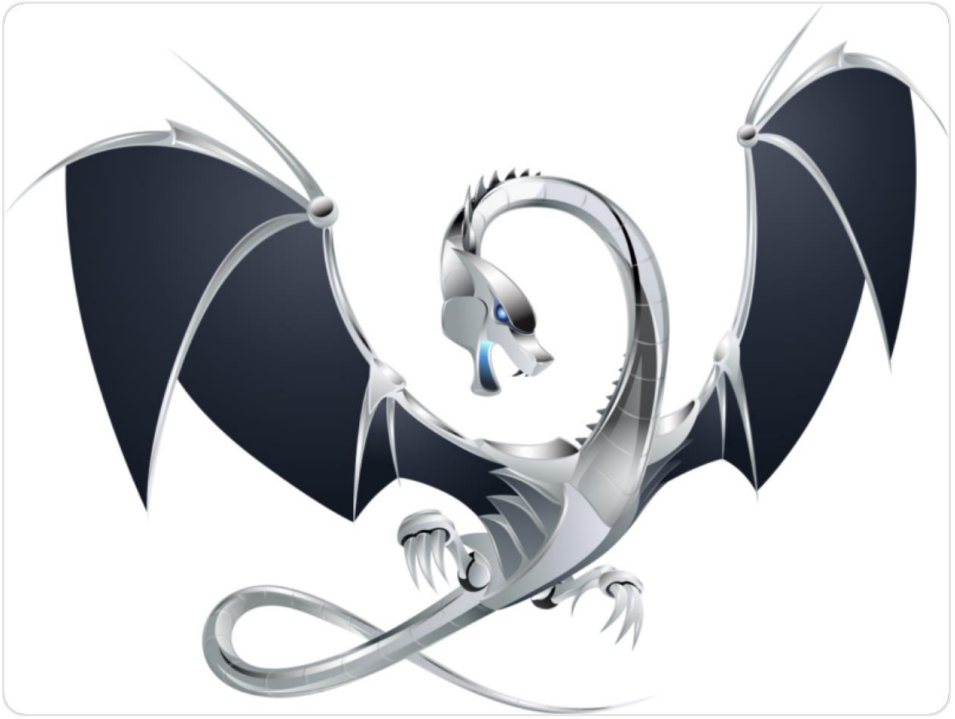
LLVM's Goofy Logo

Introducing LLVM - History



JP @vectorpoem · 5 Mar 2018

Apple: We are the preeminent design company. The best design is humble; it delights; it fulfills imperatives the user is not even consciously aware of.
Also Apple: Our compiler's logo is a massive chrome dragon.



Compilerland Drama!

Introducing LLVM - History

“For GCC to be replaced by another technically superior compiler that defended freedom equally well would cause me some personal regret, but I would rejoice for the community's advance. The existence of LLVM is a terrible setback for our community precisely because it is not copylefted and can be used as the basis for nonfree compilers -- so that all contribution to LLVM directly helps proprietary software as much as it helps us.”

- Richard Stallman, January 2014

LLVM: Software Engineering Marvel

Introducing LLVM - Features

Feature-rich command-line toolset

Ease of…

- use
- adaptation
- extensibility

modular architecture with object-oriented design discipline



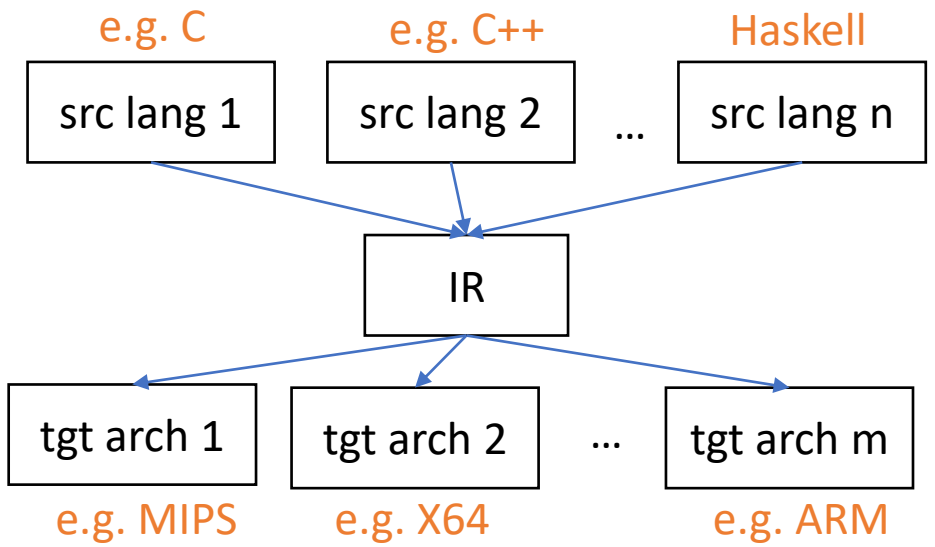
Plugin architecture to ease custom language development, optimization, and program analysis

The Multicompiler Realized

Introducing LLVM - Features

Really nailed the intermediate representation

- Frontends translate a distinct language into LLVM IR
- Optimizers check and improve the IR
- Backends translate IR to various runtimes



Lecture Outline

LLVM

Introducing LLVM

- History
- Features

Getting started with LLVM

- Installation
- Command-line tools

Understanding LLVM

- A taste of bitcode
- Beyond codegen



Advanced Topics

Learning about LLVM

Getting Started With LLVM

Why Learn Another Compiler Toolset?

- You're unlikely* to *write* a compiler professionally
- You're likely to *use* a compiler professionally

Understanding your tools is a key differentiator

- It's not too hard to pick up the basics

**I actually have had students go on to write compilers professionally*

LLVM Setup

Getting Started With LLVM

Should already be available in package managers

- apt install llvm-dev clang++

You may want to peg your install at a specific version

- apt install llvm-dev-9 clang++-9

LLVM Frontends

Understanding LLVM – Command-line Tools

Aim for GCC command-line capability

Handling Different Languages

- clang - The C frontend
- clang++ - The C++ frontend
- Llgo – The go frontend
- GHC – The Haskell compiler has an LLVM option!

```
clang++ -O2 input.c –o object.o –c ; clang++ object.o –o prog
```

The GCC Toolchain Replacements

Understanding LLVM – Command-Line Tools

clang / clang++

- C/C++ frontend (replace gcc/g++)
- Usually gcc/g++ is a clang/clang++ alias on OSX!

lld

- Linker (replace ld)

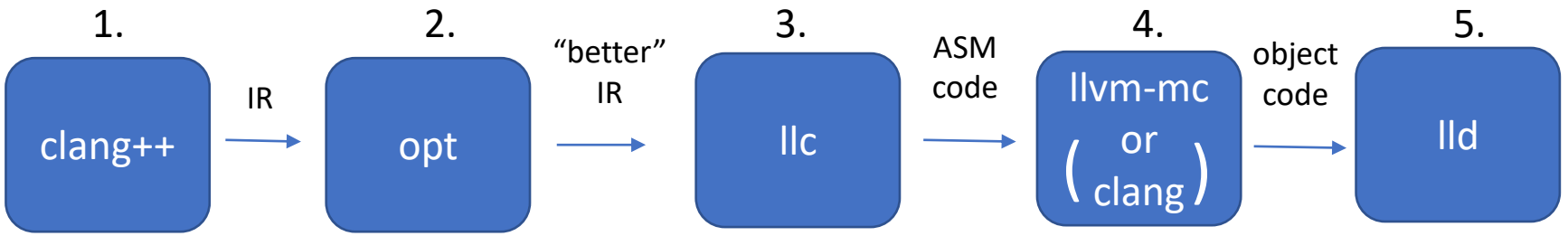
Integrated assembler

- Use clang as an assembler or llvm-mc (replace as)



Invoking the Tools

Understanding LLVM – Command-line use



fact.c

```
int fact(int arg){
  if (arg <= 0){
    return 1;
  } else {
    return arg * fact(arg - 1);
  }
}
```

1. clang++ fact.c –S –emit-llvm
2. opt fact.ll
3. llc fact.ll
4. clang fact.s
5. lld fact.o –o fact.prog

Lecture Outline

LLVM

Introducing LLVM

- History
- Features

Getting started with LLVM

- Installation
- Command-line tools

Getting More Out of LLVM

- Language development
- A Taste of bitcode
- Beyond codegen



Advanced Topics

Harnessing LLVM

Getting More Out of LLVM – Language Development

https://msm.runhello.com/p/1003

"Traditionally compiler implementation"

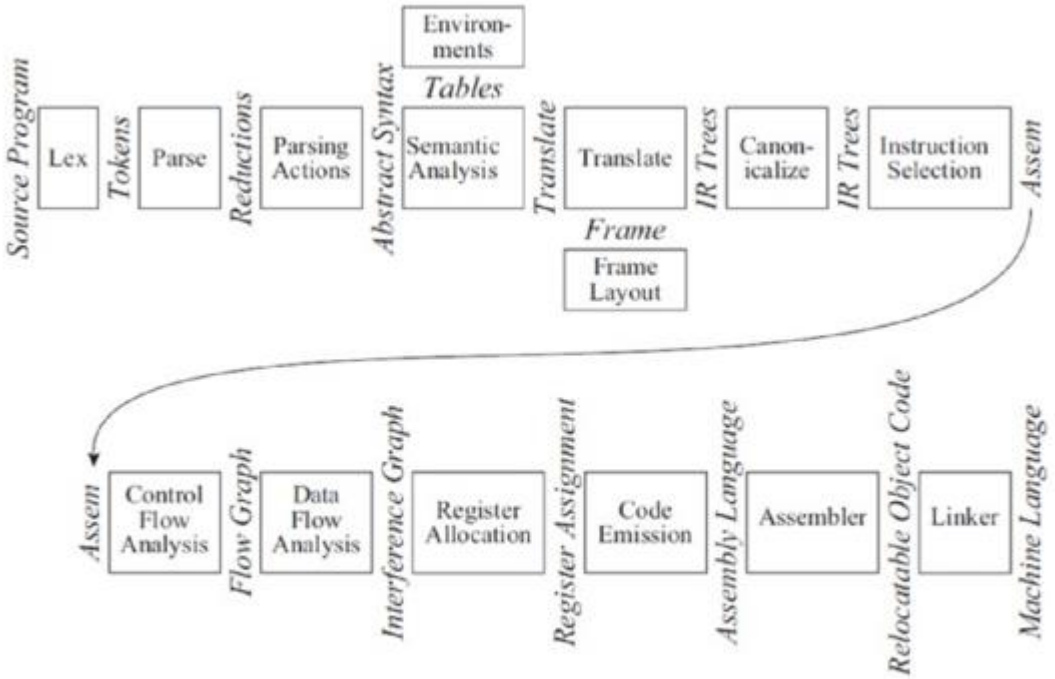


FIGURE 1.1. Phases of a compiler, and interfaces between them

Harnessing LLVM

Getting More Out of LLVM – Language Development

https://msm.runhello.com/p/1003

"Traditionally compiler implementation"

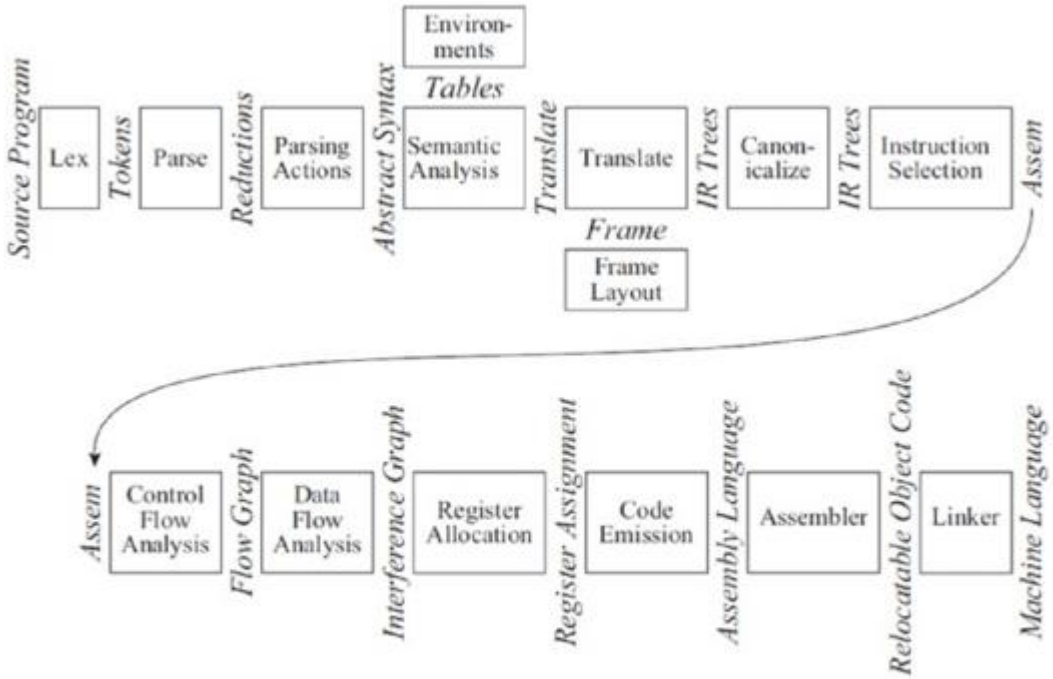


FIGURE 1.1. Phases of a compiler, and interfaces between them

"Modern compiler implementation"

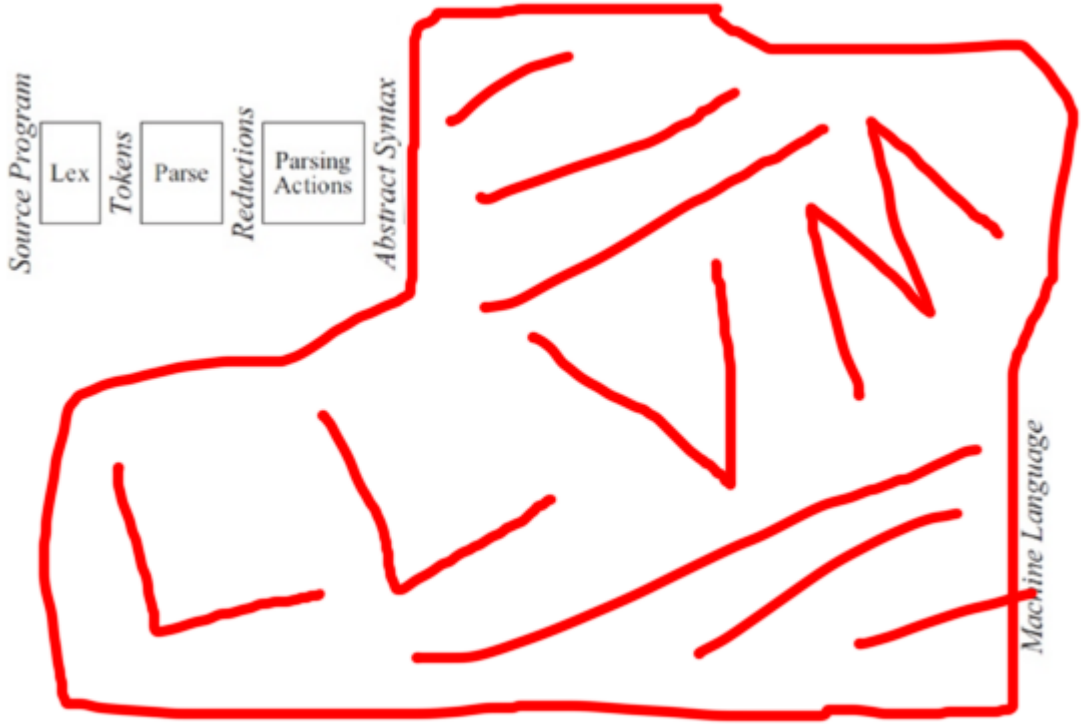


FIGURE 1.1. Phases of a compiler, and interfaces between them

Harnessing LLVM

Getting More Out of LLVM – Language Development

Two ways to use LLVM middle- and back-ends

- Output an IR file, use LLVM command line tools to read it
- Construct the LLVM “quads” directly within the LLVM API

Both approaches require understanding the IR

“Modern compiler implementation”

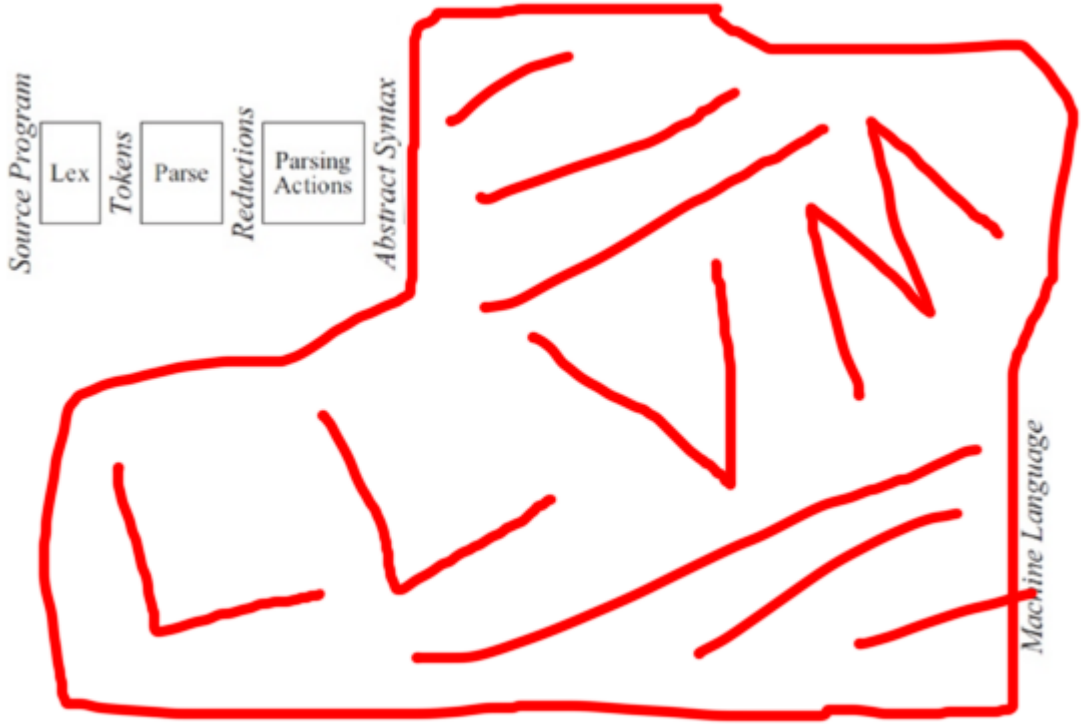


FIGURE 1.1. Phases of a compiler, and interfaces between them

Working with LLVM IR

Getting More Out of LLVM – A Taste of Bitcode

The true power of LLVM is its intermediate representation

- Expressive enough for translation/analysis
- Simple enough for human readability

LLVM IR files have two formats

- .ll (text - human readable format)
- .bc (binary - machine readable format)

Some Interesting things about LLVM IR

Getting More Out of LLVM – A Taste of Bitcode

Instructions

- Explicitly typed
- Annotations such as nsw (no signed wrap)

Registers

- Infinite number in the abstraction
- Explicitly allocated by `alloca` instruction
- Forced SSA

Memory

- Infinite linear address space



The Poison Value

Getting More Out of LLVM – A Taste of Bitcode

An explicit value exists to indicate the result of undefined behavior

- The string poison can be used in place of any constant
- Poison value produced when strict semantics are violated (wrapping on an nsw instruction, for instance)



LLVM IR

Getting More out of LLVM – A Taste of Bitcode

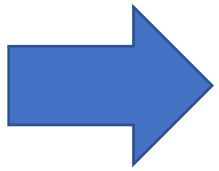
Suppress optnone by adding `-disable-o0-optnone`

"optimizer should ignore this function"

Best way to understand LLVM IR is

- Read the documentation: https://llvm.org/docs/LangRef.html
- then look at real bitcode programs: `clang++ prog.c -S -emit-llvm -o prog.ll`

```
int main(){
        int a;
        int b;
        a  = 1;
        b = 2 * a;
        return a + b;
}
```



```
; Function Attrs: noinline norecurse nounwind optnone uwtable
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 1, i32* %2, align 4
  %4 = load i32, i32* %2, align 4
  %5 = mul nsw i32 2, %4
  store i32 %5, i32* %3, align 4
  %6 = load i32, i32* %2, align 4
  %7 = load i32, i32* %3, align 4
  %8 = add nsw i32 %6, %7
  ret i32 %8
}
```

Other Cool Tools

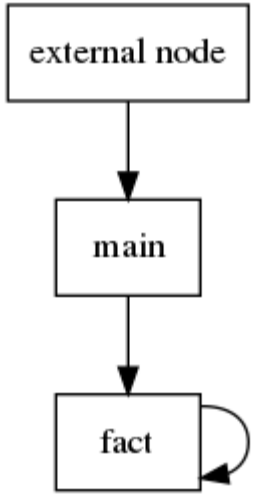
Getting More Out of LLVM – Beyond Codegen

- `lli` – directly execute LLVM bitcode programs
- `llvm-link` – link multiple IR files
- `llvm-as` – translate human-readable IR to machine-readable IR
- `llvm-dis` – translate machine-readable IR to human readable IR
- `lldb` – llvm debugger (akin to gdb)
- `opt` – llvm optimizer / IR code analyzer

Some Cool opt Stuff

Getting More Out of LLVM – Beyond Codegen

opt fact.ll –dot-callgraph

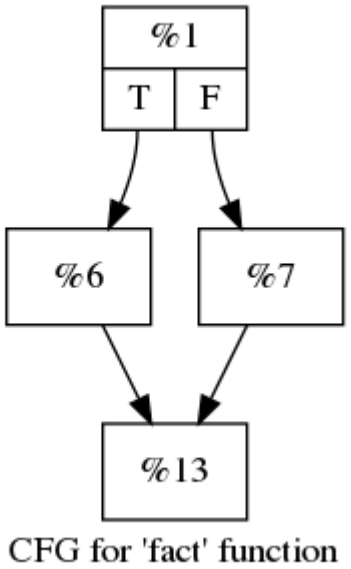


Call graph

Some Cool opt Stuff

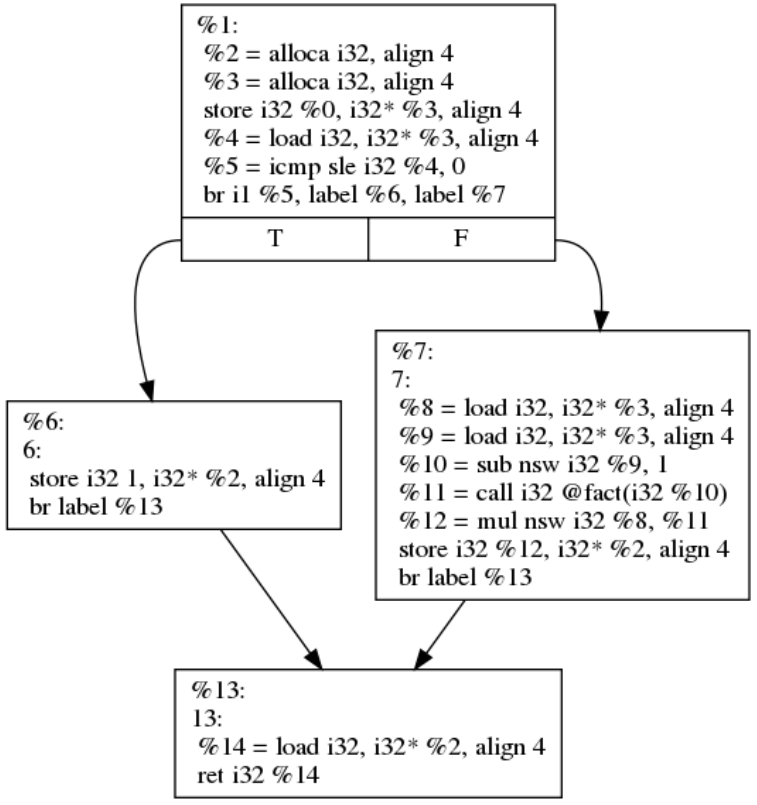
Getting More Out of LLVM – Beyond Codegen

opt fact.ll –dot-cfg-only



CFG for 'fact' function

opt fact.ll –dot-cfg

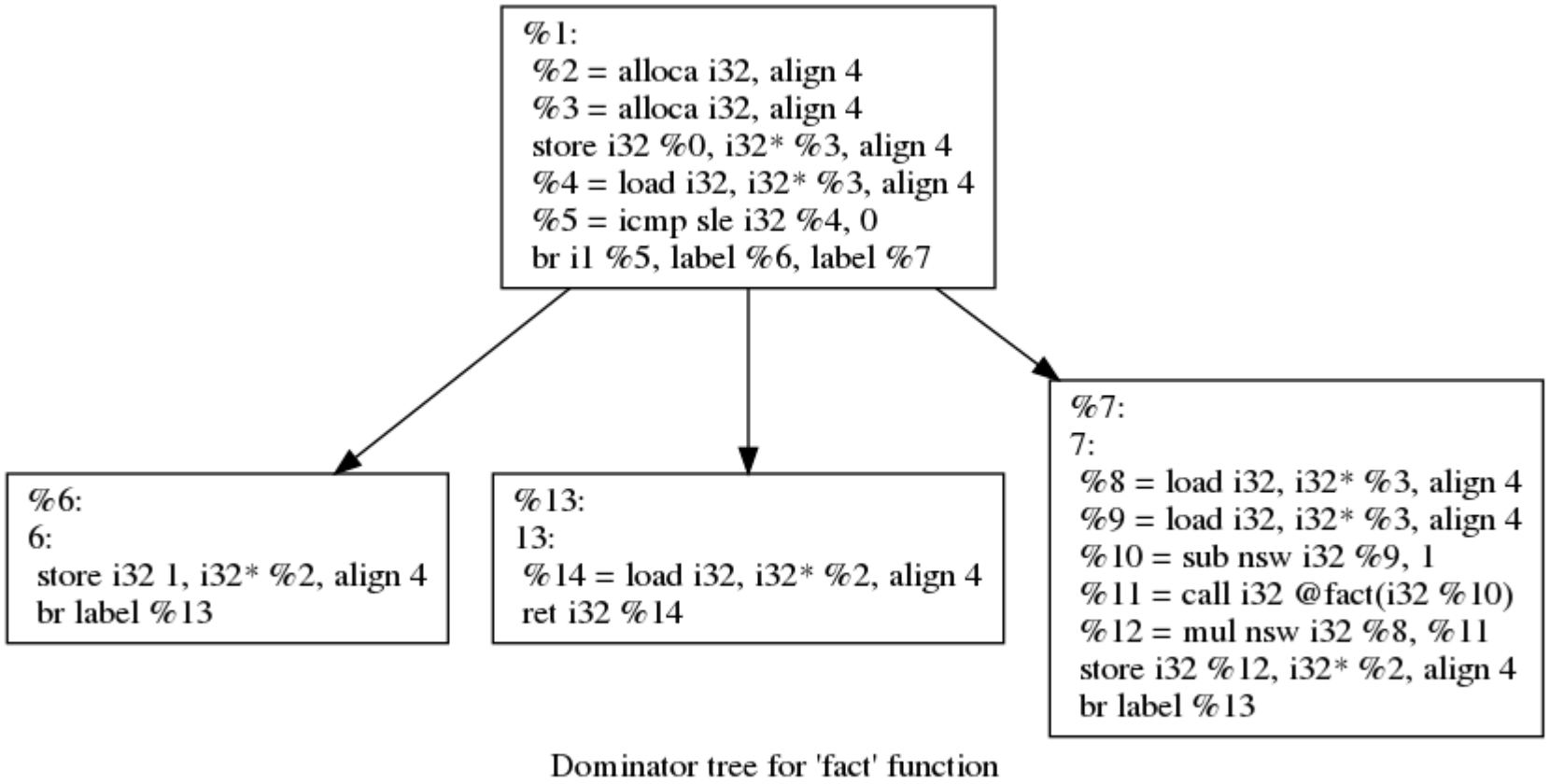


CFG for 'fact' function

Some Cool opt Stuff

Getting More Out of LLVM – Beyond Codegen

opt fact.ll –dot-dom



Dominator tree for 'fact' function

Code Checks

Getting More Out of LLVM – Beyond Codegen

Another cool LLVM trick

scan-build make

Summary

LLVM – Lecture Underview

LLVM

- A powerful IR
- A great suite of tools
  - May become your daily compiler
  - May just become a useful code analysis/enhancement

Caveats

- Documentation is a bit sparse

Next Time

LLVM – Lecture Underview

Topic: Beyond Compilers

- As LLVM shows us, compiler tools are useful beyond compiling!
- We'll talk about how some core concepts and techniques from this class are/could be used elsewhere

Administrivia: Next lecture is the last required lecture!

Questions