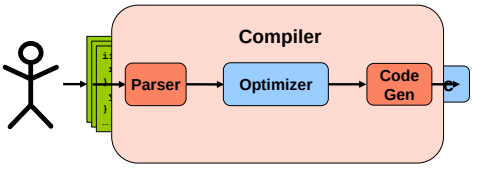
Advanced Compiler Design

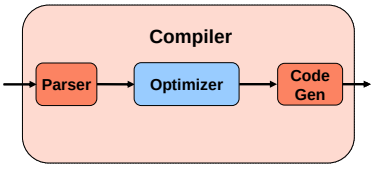
CSE 231

Instructor: Sorin Lerner

Let's look at a compiler



Let's look at a compiler

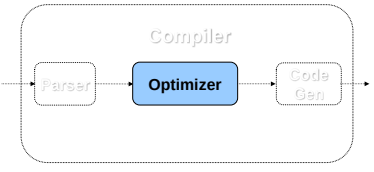


Advanced Optimizer Design

CSE 231

Instructor: Sorin Lerner

What does an optimizer do?



1. Compute information about a program
2. Use that information to perform program transformations
   (with the goal of improving some metric, e.g. performance)

What do these tools have in common?

- Bug finders
- Program verifiers
- Code refactoring tools
- Garbage collectors
- Runtime monitoring system
- And… optimizers

What do these tools have in common?

- Bug finders
- Program verifiers
- Code refactoring tools
- Garbage collectors
- Runtime monitoring system
- And… optimizers

They all analyze and transform programs
We will learn about the techniques underlying all these tools

Program Analyses, Transformations, and Applications

CSE 231

Instructor: Sorin Lerner

Course goals

- Understand basic techniques
  - cornerstone of a variety of program analysis tools
  - useful no matter what your future path
- Get a feel for compiler research/implementation
  - useful for research-oriented students
  - useful for implementation-oriented students

Course topics

- Representing programs
- Analyzing and transforming programs
- Applications of these techniques

Course topics (more details)

- Representations
  - Abstract Syntax Tree
  - Control Flow Graph
  - Dataflow Graph
  - Static Single Assignment
  - Control Dependence Graph
  - Program Dependence Graph
  - Call Graph

Course topics (more details)

- Analysis/Transformation Algorithms
  - Dataflow Analysis
  - Interprocedural analysis
  - Pointer analysis
  - Rule-based analyses and transformations
  - Constraint-based analysis

Course topics (more details)

- Applications
  - Scalar optimizations
  - Loop optimizations
  - Object oriented optimizations
  - Program verification
  - Bug finding

Course pre-requisites

- No compilers background necessary
- No familiarity with lattices
  - I will review what is necessary in class
- Familiarity with functional/OO programming
  - Optimization techniques for these kinds of languages
- Familiarity with C/C++
  - Project will be in C++
- Standard ugrad cs curriculum likely enough
  - Talk to me if you're concerned

Course work

- In-class midterm (30%)
  - Date posted on web site
- In-class final (30%)
  - Date published by official calendar
- Course project (35%)
- Class participation (5%)

Course project

- Goal of the project
  - Get some hands on experience with compilers
  - Two options, most will do option 1
- **Option 1: LLVM project**
  - Implement some analyses in LLVM, three milestones
  - Hand in your code and it's auto-graded
- **Option 2: Research (by instructor approval)**
  - Pick some interesting idea, and try it out
  - Proposals due at the beginning of the second week
  - Can leverage your existing research

LLVM Project

- M1: Simple instrumentation
- M2: Analysis framework
- M3: Implement Analyses in framework
- You will extend LLVM. This will require C++
  - If you don't know C++, you should be super confident that you can learn it. Otherwise, drop the class
- To be done alone

Research Project

- Requires instructor approval
  - You need to come up with your own idea…
  - … by the end of week 1
  - Most students doing this will be PhD students
  - It's ok to leverage or overlap with existing research
- To be done alone
- I envision at most 10 people doing this

Readings

- Paper readings throughout the quarter
- Seminal papers and state of the art
- Gives you historical perspective
- Shows you lineage from idea to practice

Administrative info

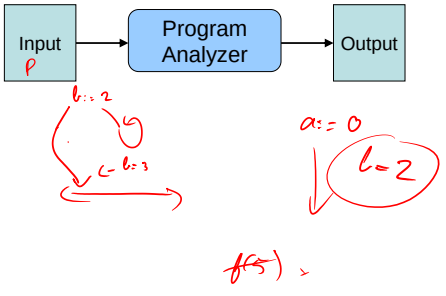
- Class web page is up
  - https://ucsd-pl.github.io/cse231/wi18/
  - (or Google “Sorin Lerner”, follow “Teaching Now”)
  - Will post lectures, readings, project info, etc.
- Piazza link on web page
  - Use for questions, answers
  - Especially LLVM/project Q&A

Academic Integrity

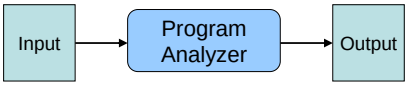
- Governed by Policy on Integrity of Scholarship (http://senate.ucsd.edu/Operating-Procedures/Senate-Manual/Appendices/2)
- Allegations are handled by Academic Integrity Office (https://students.ucsd.edu/academics/academic-integrity)
- Course penalty for cheating in 231 may result in failing the assignment or the entire class
- Cheaters may be subject to additional administrative sanctions

Questions?

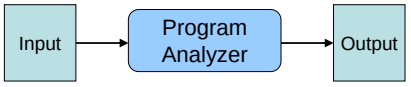
Program Analyzer Issues (discuss)



Program Analyzer Issues (discuss)

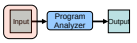


Program Analyzer Issues (discuss)



Instructor's discussion notes

Input issues



- Input is a program, but…
- What language is the program written in?
  - imperative vs. functional vs. object-oriented? maybe even declarative?
  - what pointer model does the language use?
  - reflection, exceptions, continuations?
  - type system trusted or not?
  - one often analyzes an intermediate language... how does one design such a language?

Instructor's discussion notes

Input issues

- How much of the program do we see?
  - all?
  - one file at a time?
  - one library at a time?
  - reflection…
- Any additional inputs?
  - any human help?
  - profile info?

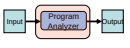
Instructor's discussion notes

Analysis issues

- Analysis/compilation model
  - Separate compilation/analysis
    - quick, but no opportunities for interprocedural analysis
  - Link-time
    - allows interprocedural and whole program analysis
    - but what about shared precompiled libraries?
    - and what about compile-time?
  - Run-time
    - best optimization/analysis potential (can even use run-time state as additional information)
    - can handle run-time extensions to the program
    - but severe pressure to limit compilation time
  - Selective run-time compilation
    - choose what part of compilation to delay until run-time
    - can balance compile-time/benefit tradeoffs

Instructor's discussion notes

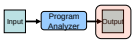
Analysis issues



- Does running-time matter?
  - for use in IDE?
  - or in overnight compile?

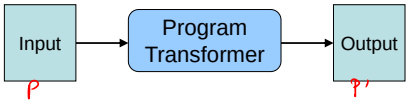
Instructor's discussion notes

Output issues



- Form of output varies widely, depending on analysis
  - alias information
  - constantness information
  - loop terminates/does not terminate
- Correctness of analysis results
  - depends on what the results are used for
  - are we attempting to design algorithms for solving undecidable problems?
  - notion of approximation
  - statistical output

Program Transformation Issues (discuss)



P P'

```
main()
  int x;
  print(x);
}
```

assert (x = 10)

Instructor's discussion notes

Input issues

- A program, and …
- Program analysis results
- Profile info?
- Environment: # of CPUs, # of cores/CPU, cache size, etc.
- Anything else?

Instructor's discussion notes

Transformation issues

- What is profitable?
- What order to perform transformations?
- What happens to the program representation?
- What happens to the computed information? For example alias information? Need to recompute?

Instructor's discussion notes

Output issues

- Output in same IL as input?
- Should the output program behave the same way as the input program?