Today

- Course overview
- Program analysis introduction

Organization

- **Lectures**
  - **Tuesdays: I lecture**
  - **Thursdays: You present about specific tools**
    - **I'll start this week**
    - **Nathan will go next week**

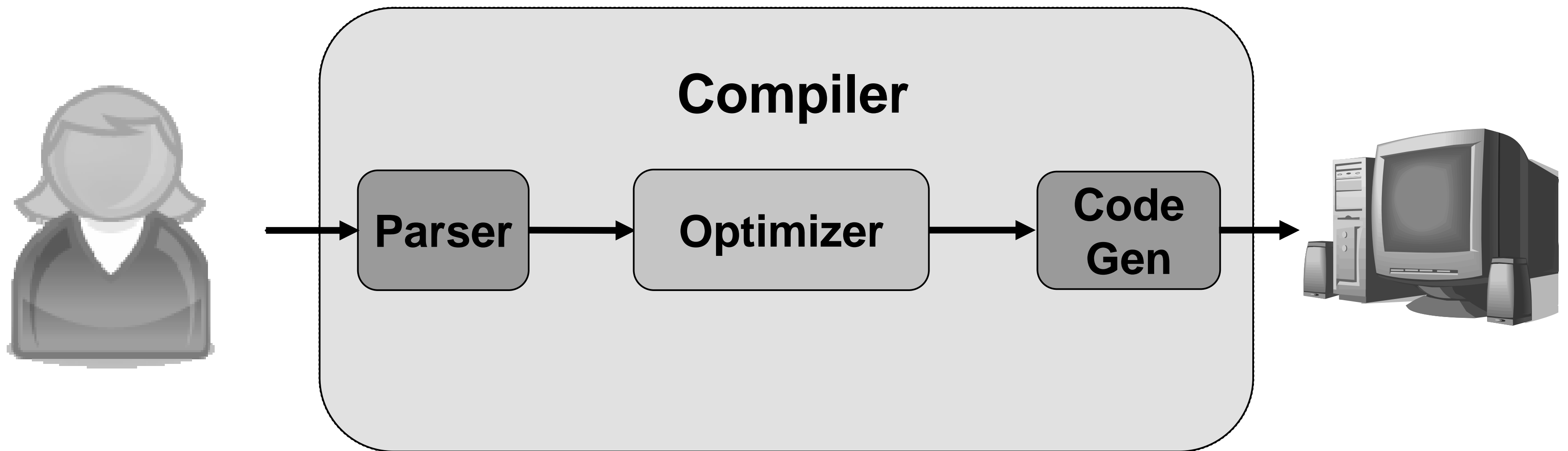
Projects

- Each student (or group or two) will complete a project
- Extend an existing analysis tool or compiler
- There are lots of open source tools that make great starting points for projects
  - CIL, cXprop, gcc, Saturn, LLVM, Locksmith, Sparse
- Example projects:
  - Add a new optimization
  - Add a new analysis feature
  - Make an existing analysis more precise
  - Etc.

More Projects

- 1-2 page proposal
  - Due mid-semester
- Implementation + in-class presentation
  - Due a few weeks before end of semester

What's in a compiler?



What's in an optimizer?

1. **Compute properties of a program using static analysis**
2. **Apply semantics-preserving transformations based on these properties**
   - **Goal: Improve some resource metric: memory usage, execution speed, etc.**

- **The field of program analysis started with optimizing compilers**
  - **But program analysis is more broadly used today**

“Program analysis offers static compile-time techniques for predicting safe and computable approximations to the set of values or behaviors arising dynamically at run-time when executing a program on a computer.”

From *Principles of Program Analysis*, Nielson & Nielson & Hankin

Program Analysis

- Used in:
  - Bug finders
  - Program verifiers
  - Code refactoring tools
  - Garbage collectors
  - Runtime monitoring system
  - And… compilers
- This class is about the fundamental program analysis techniques used in all of these applications

Course Goals

- **Understand basic techniques for doing program analyses and transformations**
  - **These techniques are the cornerstone of a variety of program analysis tools**
  - **They may come in handy, no matter what research you end up doing**
- **Get a feeling for what research is like in the area by reading research papers, and getting your feet wet in a small research project**

Motivation

- Improve program correctness
  - Prevent bugs
  - Find bugs
  - Predict resource usage
  - Prove properties

- Improve program performance
  - Avoid redundant computations
  - Substitute fast computations for slow ones

- **Example program property:**
  - **“Z holds the value 17 at line 10”**
  - **Why is this useful?**
- **Challenges: We need to compute this property**
  - **Safely**
  - **Precisely**
  - **Efficiently**

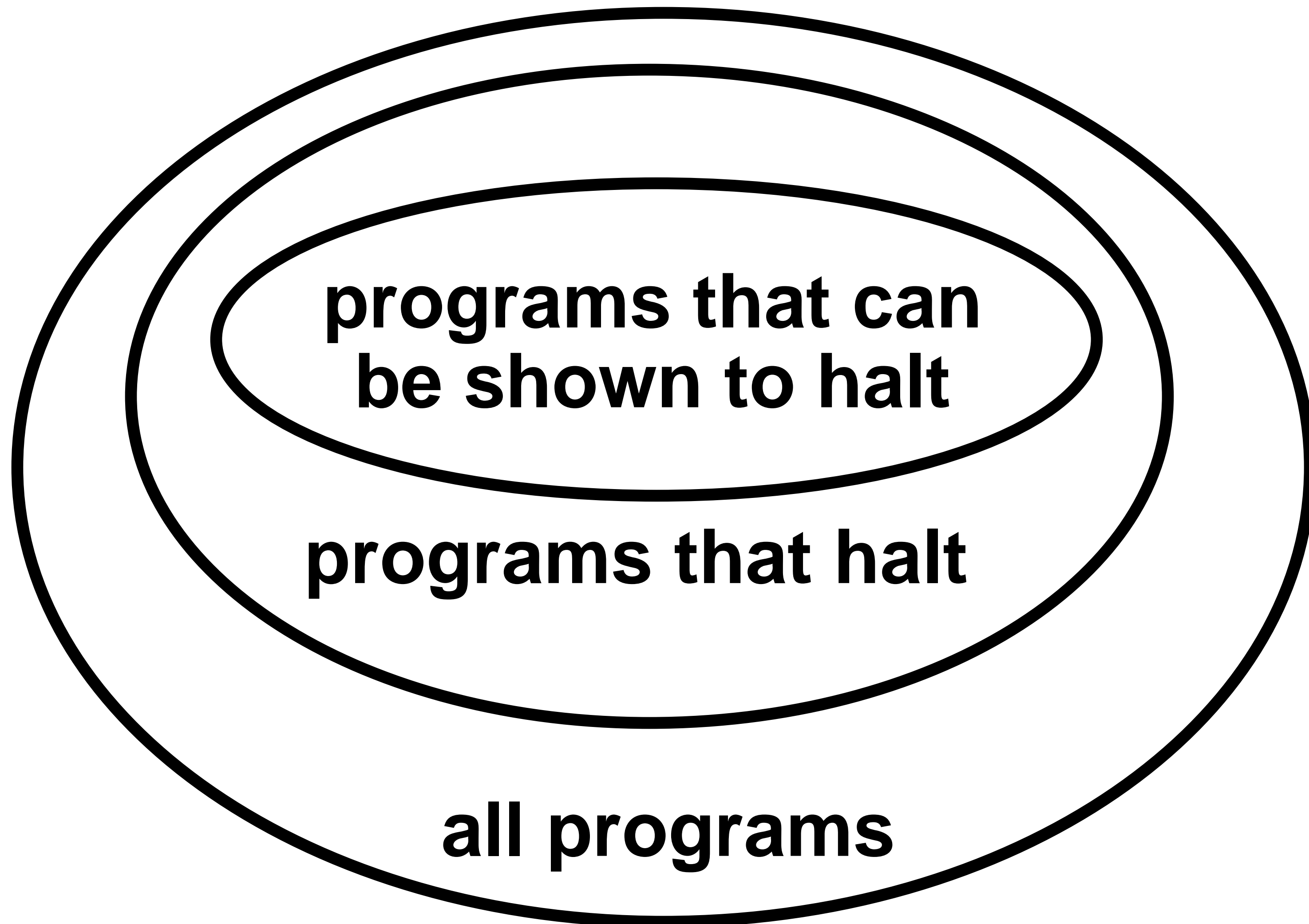
Challenge 1: Safety

- "Yes" means "absolutely yes, for all possible executions of the program"
  - Ideally, analyzer could emit a proof
    - What would this look like?
- "No" means "probably not but I don't know for sure"
  - Proof could not be found
- Note: Sometimes unsafe analyses are useful

Challenge 2: Precision

- **Imprecise analysis is easy**
  - **Just say “I don’t know” every time**
- **How much precision is necessary?**
  - **For example, do we need to prove that Z==17, or is it ok just to prove that Z > 0?**
- **Imprecision is inevitable**
  - **All interesting program analysis questions are undecidable when phrased precisely**

Precision Cont'd



Challenge 3: Efficiency

- Inherent conflict between precision and efficiency
  - How long to analyze all of Windows XP?
  - Many state-of-the-art analysis techniques do not scale to really big programs
- Claim: Only modular analyses truly scale
  - Even O(n) may be unacceptable if the analysis is not modular
  - Example of a modular analysis: Compute per-function summaries, then analyze the summaries together

Awful Example

```
int foo (void) {
    int x = 0;
    return x++ + x++;
}
```

- What does this function return?
- C compiler can pick an order in which to evaluate exprs
- An analyzer outside of the compiler must evaluate all possibilities

Language Semantics

- Semantics tells us what a program in the language means
  - “Mathematical model that predicts the behavior of a program.”
- Scope is limited – does not tell us:
  - If the program is correct
  - What system calls the program makes
  - What instructions the program uses
  - How fast the program is, or how much memory it uses
- The field of static analysis was developed to support optimization
  - But w/o language semantics
  - This led to confusing papers and broken optimizers

Loose Semantics

- A “conforming” C program produces the same results on all compiler / platform combinations
- The C standard tells us what conforming programs look like
- Not all legal programs conform

```
int x[10];
y = x[15];
```

- C semantics does not tell us what value is in y after the assignment
  - Of even if the program keeps running

Loose Semantics

- Claim: The compiler for a well-designed language can statically check for conformance
- Does C have this property?
  - No: It is undecidable whether a C program conforms
    - Oops
  - Optimizing C compilers break non-conforming programs without informing the user

More Semantics

- Analyses only apply to conforming programs
  - Most large programs will contain non-conforming behavior
- Unfortunate implication: Can't actually predict the behavior of a C program
  - That is, the mathematical model doesn't apply to most real programs
  - Thus the buffer overflow vulnerabilities that make life difficulty on the Internet
- In this course we'll be taking a pretty relaxed view of semantics

Course Topics

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

Course Topics

- **Analysis/Transformation Algorithms**
  - **Dataflow Analysis**
  - **Interprocedural analysis**
  - **Pointer analysis**
  - **Abstract interpretation**
  - **Constraint-based analysis**
  - **Interaction between transformations and analyses**

Course Topics

- Applications
  - Scalar optimizations
  - Loop optimizations
  - Program verification
  - Bug finding
  - Garbage collection

Analysis Issues

- How much of the program do we see?
  - All?
  - One file at a time?
  - One library at a time?
- Any additional inputs?
  - Human help?
  - Profile data?

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**

Summary

- **For next Tuesday: Read "Lecture Notes on Static Analysis," pages 1-34**