

Today

- ◆ **Course overview**
- ◆ **Program analysis introduction**

Organization

- ◆ **Lectures**
 - **Core program analysis material**
 - **Application and tools**
- ◆ **Assignments**
 - **First assignment is already on the web**
 - **Due in one week**
- ◆ **Code reviews**
- ◆ **Maybe we'll read a few papers**
- ◆ **Project**

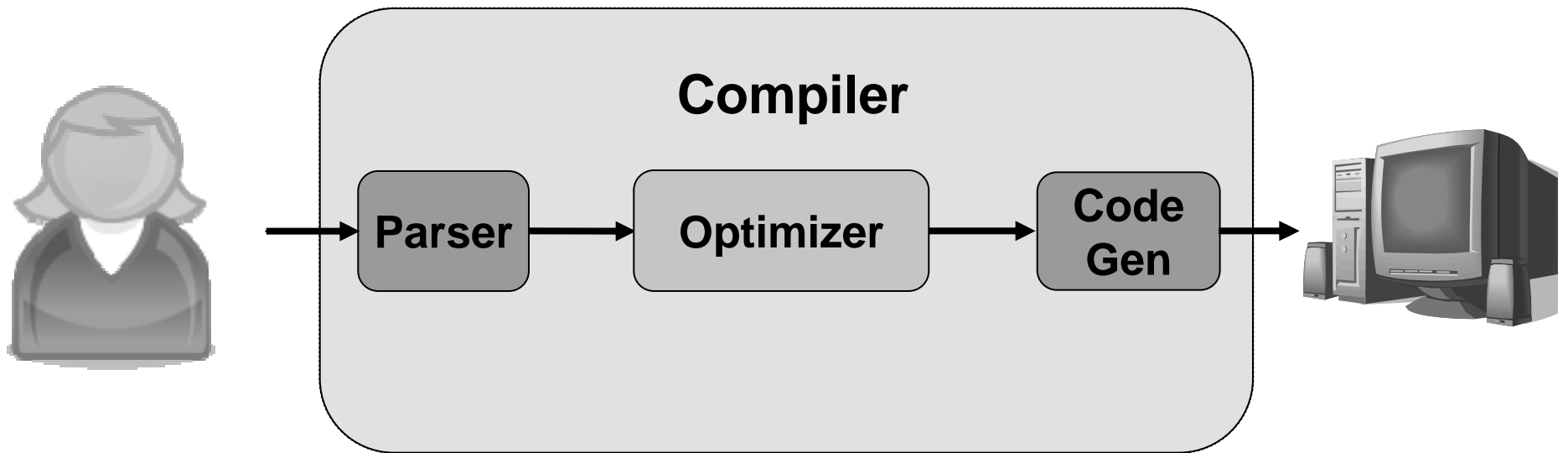
Projects

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

What is “Advanced Compilers”?

- ◆ Many versions of this class could be taught...
- ◆ This one is about
 - Static analysis
 - Theory and practice
 - Using analysis results to drive transformations
 - Using analysis results to do other stuff

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 1: Improve some resource metric: memory usage, execution speed, etc.**
 - ◆ **Goal 2: Do not change the meaning of the program**
- ◆ **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**
 - **Efficient runtime monitoring systems**
 - **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 “I don’t know for sure if this property holds”**
 - **Proof could not be found**
- ◆ **Alternative: Return “definitely yes,” “definitely no,” and “I don’t know”**
 - **Why is the third alternative necessary?**

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

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

Reasoning About Programs

```
int foo (int x) {  
    return x++ + x++;  
}  
void bar (int i) {  
    a[a[i]] = 0;  
}
```

- ◆ What do these mean?
- ◆ An analyzer must have a clear idea about programs like this
 - Even if programmers don't

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 “strictly conforming” C program produces the same results on all compiler / platform combinations
- ◆ The C standard tells us what conforming programs look like
- ◆ But...
 - Standard is in English
 - The English is not always clear
 - It is undecidable whether a given program is conforming or strictly conforming
- ◆ We’ll spend a bit of time on language 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**
 - **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

- ◆ **Static analysis is broadly useful**
- ◆ **We'll learn how to do it**

- ◆ **Homework 1 is on the web**
 - **Due in one week**

- ◆ **Get on the course mailing list**