Programming Languages
Topic of Ultimate Mastery

Wes Weimer
CS 6610
http://www.cs.virginia.edu/~weimer/6610
Reasonable Initial Skepticism
Today’s Class

• Vague Historical Context
• Goals For This Course
• Requirements and Grading
• Course Summary

• Convince you that PL is useful
Meta-Level Information

- Please interrupt at any time!
- Completely cromulent queries:
  - I don’t understand: please say it another way.
  - Slow down, you talk too fast!
  - Wait, I want to read that!
  - I didn’t get joke X, please explain.
What Have You Done For Us Lately?

• Isn’t PL a solved problem?
  - PL is an old field within Computer Science
  - 1920’s: “computer” = “person”
  - 1936: Church’s Lambda Calculus (= PL!)
  - 1937: Shannon’s digital circuit design
  - 1940’s: first digital computers
  - 1950’s: FORTRAN (= PL!)
  - 1958: LISP (= PL!)
  - 1960’s: Unix
  - 1972: C Programming Language
  - 1981: TCP/IP
  - 1985: Microsoft Windows
  - 1992: Ultima Underworld / Wolfenstein 3D

“... a prestigious line of work with a long and glorious tradition.” - Vizzini
Don’t We Already Have Compilers?
Dismal View Of PL Research

DOCTOR FUN

I WANT

C++

I WANT

Java

or C#

NOW

THEN

Progress

19 Mar 97

Copyright © 1997 David Farley, d-farley@tezcreek.com
http://sunsite.unc.edu/Dave/ddrfun.html

Opinions expressed herein are solely those of the author.

This cartoon is made available on the Internet for personal viewing only.
Parts of Computer Science

- **CS** = \((\text{Math} \times \text{Logic}) + \text{Engineering}\)
  - Science (from Latin *scientia* - knowledge) refers to a system of acquiring knowledge based on empiricism, experimentation, and methodological naturalism - aimed at finding out the truth.

- We rarely actually do this in CS
  - “CS theory” = Math (logic)
  - “Systems” = Engineering (bridge building)
Programming Languages

- Best of both worlds: Theory and Practice!
  - Only pure CS theory is more primal
- Touches most other CS areas
  - **Theory**: DFAs, PDAs, TMs, language theory (e.g., LALR)
  - **Systems**: system calls, assembler, memory management
  - **Arch**: compiler targets, optimizations, stack frames
  - **Numerics**: FORTRAN, IEEE FP, Matlab, loop nest optim.
  - **AI**: theorem proving, machine learning, search
  - **DB**: SQL, persistent objects, modern linkers
  - **Networking**: packet filters, protocols, even Ruby on Rails
  - **Graphics**: OpenGL, LaTeX, PostScript, even Logo (= LISP)
  - **Security**: buffer overruns, .net, bytecode, PCC, ...
  - **Software Engineering**: bug finding, refactoring, types, ...
Overarching Theme

• I assert (and shall convince you) that

• PL is one of the most vibrant and active areas of CS research today
  - It has theoretical and practical meatiness
  - It intersects most other CS areas

• You will be able to use PL techniques in your own projects
Goal #1

- Learn to use advanced PL techniques
Useful Complex Knowledge

• A proof of the fundamental theorem of calculus
• A proof of the max-flow min-cut theorem
• Nifty Tree node insertion (e.g., B-Trees, AVL, Red-Black)
• The code for the Fast Fourier Transform
• And so on ...
No Useless Memorization

• I will not waste your time with useless memorization
• This course will cover complex subjects
• I will teach their details to help you understand them the first time
• But you will never have to memorize anything low-level
• Rather, learn to apply broad concepts
Goal #2

- When (not if) you design a language, it will avoid the mistakes of the past and you’ll be able to describe it formally.
Story: The Clash of Two Features

- **Real story** about bad programming language design
- **Cast includes** famous scientists
- **ML (’82)** is a functional language with polymorphism and monomorphic references (i.e. pointers)
- **Standard ML (’85)** innovates by adding polymorphic reference
- It took **10 years to fix** the “innovation”
Polymorphism (Informal)

- Code that works uniformly on various types of data

- Examples of function signatures:
  
  \[
  \text{length} : \alpha \text{ list} \rightarrow \text{int} \quad \text{(takes an argument of type “list of } \alpha \text{”, returns an integer, for any type } \alpha)\]
  
  \[
  \text{head} : \alpha \text{ list} \rightarrow \alpha
  \]

- Type inference:
  - generalize all elements of the input type that are not used by the computation
References in Standard ML

• Like “updateable pointers” in C

• Type constructor: \texttt{ptr }\tau
  - \texttt{x : ptr int } \quad \text{=== “ }x\text{ is a pointer to an integer”}

• Expressions:
  
  \texttt{alloc : }\tau \rightarrow \text{ptr }\tau \quad \text{(allocate a cell to store a }\tau) \\
  \texttt{*e } : \tau \text{ when }e : \text{ptr }\tau \quad \text{(read through a pointer)} \\
  \texttt{*e1 := e2 } \text{ with }e1 : \text{ptr }\tau \text{ and }e2 : \tau \\
  \quad \quad \quad \quad \quad \text{(write through a pointer)}

• Works just as you might expect …
Polymorphic References: A Major Pain

Consider the following program fragment:

**Code**

```plaintext
fun id(x) = x
val c = alloc id
print (*c) ("hi")
print (*c) (5)
```
Polymorphic References: A Major Pain

Consider the following program fragment:

```
fun id(x) = x
val c = alloc id
fun inc(x) = x + 1
*c := inc
print (*c) (6)
```
Polymorphic References: A Major Pain

Consider the following program fragment:

Code

fun id(x) = x
val c = alloc id
fun inc(x) = x + 1
*c := inc
(*c) ("hi")
## Polymorphic References: A Major Pain

Consider the following program fragment:

<table>
<thead>
<tr>
<th>Code</th>
<th>Type inference</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fun id(x) = x</code></td>
<td><code>id : \( \alpha \rightarrow \alpha \)</code> (for any ( \alpha ))</td>
</tr>
<tr>
<td><code>val c = alloc id</code></td>
<td><code>c : ptr (\( \alpha \rightarrow \alpha \))</code> (for any ( \alpha ))</td>
</tr>
<tr>
<td><code>fun inc(x) = x + 1</code></td>
<td><code>inc : int \rightarrow int</code></td>
</tr>
<tr>
<td><code>*c := inc</code></td>
<td>Ok, since <code>c : ptr (int \rightarrow int)</code></td>
</tr>
<tr>
<td><code>(*c) (&quot;hi&quot;)</code></td>
<td>Ok, <code>c : ptr (string \rightarrow string)</code></td>
</tr>
</tbody>
</table>
Reconciling Polymorphism and References

- Type system **fails to prevent a type error!**
- Common solution:
  - value restriction: generalize only the type of values!
    - easy to use, simple proof of soundness
- X Features $\Rightarrow X^2$ Complication
- To see what went wrong we needed to understand semantics, type systems, polymorphism and references
Story 2: Java Bytecode Subroutines

- **Java bytecode** programs contain **subroutines** (jsr) that run in the caller’s stack frame *(why?)*
- jsr complicates the formal semantics of bytecodes
  - Several verifier bugs were in code implementing jsr
  - 30% of typing rules, 50% of soundness proof due to jsr
- **It is not worth it:**
  - In 650K lines of Java code, 230 subroutines, saving 2427 bytes, or 0.02%
  - 13 times more space could be saved by renaming the language back to Oak
  - [In 1994], the language was renamed “Java” after a trademark search revealed that the name “Oak” was used by a manufacturer of video adapter cards.
Recall Goal #2

• When (not if) you design a language, it will avoid the mistakes of the past and you’ll be able to describe it formally
Goal #3

• Understand current PL research (PLDI, POPL, OOPSLA, TOPLAS, ...) and technology transfer (MS, Intel, ...).
Final Goal: Fun
This 1960 Daniel Keyes sci-fi novel is told as a "progris riport" from the point-of-view of Charlie Gordon as he takes an experimental intelligence-enhancing treatment. The treatment is temporary. The book won the Hugo and Nebula awards.
This Sri Lanka-born, British computer scientist is best known for his development of QuickSort, a logic for verifying program correctness, the monitor approach to mutual exclusion, and the formalism of Communicating Sequential Processes. In 2009 he apologized for inventing the null reference:

- I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.
Prerequisites

• Undergraduate PL/compilers course?  
  - No

• “Mathematical maturity”
Assignments

- Short Homework Assignments (5)
- Long Homework Assignment (1)
- Daily Reading (1-2 papers per class)
- Final Project
Homework Problem Sets

- Some material can be “mathy”
- Much like Calculus, practice is handy
- **Short**: ~3 theory + 1 coding per HW
- You have one week to do each one
  - Usually available in advance …
- **Long**: analysis of real C programs
- **NB**: I will offer suggestions and comments on your **English prose**.
Final Project

• Literature survey, implementation project, or research project
• Write a 5-page paper (akin to PLDI)
• Give a ~10 minute presentation
• On the topic of your choice
  - I will help you find a topic (many examples)
  - Best: integrate PL with your current research
Reading Quizzes: Let's Vote!

• A key problem:
  - If I never check, graduate students will not do the reading.

• A related desire:
  - Graduate students often wish that someone would make them do the reading.

• Proposal:
  - “One Word” reading quizzes
Piazza Forum

- Ask questions about assignments
- Compare concerns about papers

- Votes
  - Alternate opt-in grading contract: you are not responsible for homeworks or reading quizzes, only the final project, but your highest possible grade is an A- or B+
How Hard Is This Class?
In 1930, the Republican-controlled House of Representatives, in an effort to alleviate the effects of the... Anyone? Anyone? ... the Great Depression, passed the ... Anyone? Anyone? The tariff bill? The Hawley-Smoot Tariff Act? Which, anyone? Raised or lowered? ... raised tariffs, in an effort to collect more revenue for the federal government. Did it work? Anyone? Anyone know the effects?
Key Features of PL

Now that the NDA is up, we're seeing a lot of people out there unhappy with the D&D Online experience we've delivered. Let's take a look at our key features list, see if we didn't miss something.

Alright, dungeons. Where are we at?

Dungeons & Dragons
Online key Features:
- Dungeons
- Dragons

They're in, sir.

Dragons? Come on, guys. Don't tell me we forgot dragons.

I think... Jim was on Dragons.

Good. Glad to hear it. And dragons?

No, no – I was on Dungeons.
Programs and Languages

- What are they trying to do?
- Are they doing it?
- Are they making some other mistake?
- Were they hard to write?
- Could we make it easier?
- Should you run them?
- How should you run them?
- How can I run them faster?
Programs and Languages

• Languages
  - Why are they annoying?
  - How could we make them better?
  - What tasks can they make easier?
  - What cool features might we add?
  - Can we stop mistakes before they happen?
  - Do we need new paradigms?
  - How can we help out My Favorite Domain?
Common PL Research Tasks

- Design a new language feature
- Design a new type system / checker
- Design a new program analysis
- Find bugs in programs
- (Help people to) Fix bugs in programs
- Transform programs (source or assembly)
- Interpret and execute programs
- Prove things about programs
- Optimize programs
Grand Unified Theory

- Design a new type system
- Your type-checker becomes a bug-finder
  - No type errors $\implies$ proof that program is safe
  - Type error $\implies$ bug may exist in program
    - Fault localization and automated program repair

- Design a new language feature
  - To prevent the sort of mistakes you found

- Write a source-to-source transform
  - Your new feature now works on existing code
CS 6610 - Core Topics

- Model Checking
- Operational semantics
- Provers and proofs
- Type theory
- Verification conditions
- Symbolic Execution
- Invariant Detection
- Abstract interpretation
- Machine Learning
- Lambda Calculus

"So, by a vote of 8 to 2 we have decided to skip the Industrial Revolution completely, and go right into the Electronic Age."
Special Topics

• Communications and Concurrency
• Fault Localization, Bug Isolation
• Automated Program Repair

• What do you want to hear about?
Weimer's Research Group

• I am looking to pick up 0 or 1 new grads
  - Only students who plan to get a PhD
  - I have plenty of unspent funding

• Take Grad PL and do well (impress me)

• Pick an ambitious semester project

• Expectations: 1-2 new submissions / year
  - Kinga: 1 journal + 5 conf + PhD in 4 years, now professor at GMU
  - Ray: 1 journal + 8 conf (1 best paper) + research internship + grant in 6 years, now at Google
  - Pieter: 2 journal + 9 conf (1 best paper) + 3 research internships in 6 years, now at Facebook
  - Claire: 4 journal + 7 conf (3 best papers) + research internship in 6 years, now a professor at CMU
First Topic: Model Checking

- Verify critical properties of software or find bugs
- Take an important program (e.g., a device driver)
- Merge it with a property (e.g., no deadlocks, asynchronous IRP handling, BSD sockets, database transactions, ...)
- **Transform** the result into a *boolean program*
  - Same control flow, but only boolean variables
- Use a **model checker** to explore the resulting *state space*
  - Result 1: program *provably satisfies property*
  - Result 2: program *violates property right here on line 92,376!*

![Comic Strip](image.png)
Example ( ) {
    do{
        lock();
        old = new;
        q = q->next;
        if (q != NULL){
            q->data = new;
            unlock();
            new ++;
        }
    } while(new != old);
    unlock();
    return;
}
Example Program

```c
Example () {
    do{
        lock();
        old = new;
        q = q->next;
        if (q != NULL) {
            q->data = new;
            unlock();
            new ++;
        }
    } while(new != old);
    unlock();
    return;
}
```

Is this program correct?

What does correct mean?
- Doing no evil?
- Doing some good?

How do we determine if a program is correct?
Verification by **Model Checking**

**Example** () {
1:    do {
        lock();
        old = new;
        q = q->next;
    2:    if (q != NULL) {
    3:        q->data = new;
        unlock();
        new ++;
    }
    4: } while (new != old);
5:    unlock();
    return;
}

1. (Finite State) Program
2. State Transition Graph
3. Reachability

- Pgm → Finite state model
- State explosion
+ State Exploration
+ Counterexamples

**Precise** [SPIN, SMV, Bandera, JPF ]
For Our Next Exciting Episode

- See webpage under “Lectures”
- Read the two articles
- Peruse the HW and Project pages