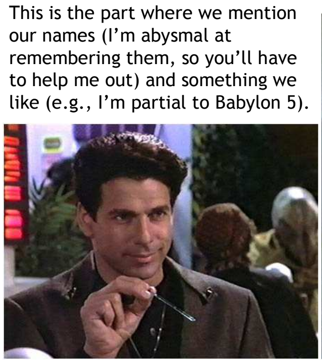
"Who Are You?"

This is the part where we mention our names (I'm abysmal at remembering them, so you'll have to help me out) and something we like (e.g., I'm partial to Babylon 5).



Programming Languages
Topic of Ultimate Mastery

Wes Weimer

CS 655 – TR 5:00-6:15 @ MEC 339

http://www.cs.virginia.edu/~weimer/655

Reasonable Initial Skepticism

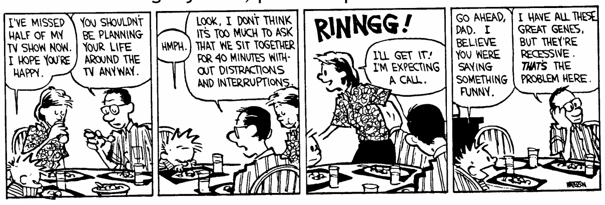
Today's Class

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

- Convince you that PL is useful

Meta-Level Information

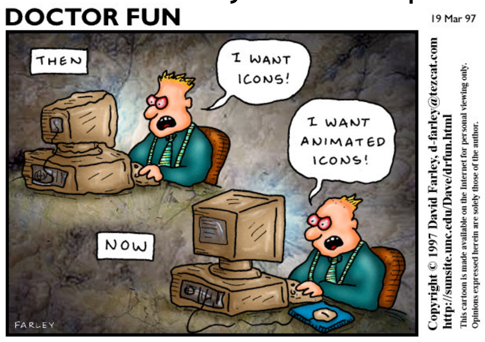
- Please interrupt at any time!
- Completely cromulent queries:
  - I don't understand: please say it another way.
  - Slow down, you talk too fast!
  - Wait, I want to read that!
  - I don't get joke X, please explain.


I'VE MISSED HALF OF MY TV SHOW NOW. I HOPE YOU'RE HAPPY.
YOU SHOULDN'T BE PLANNING YOUR LIFE AROUND THE TV ANYWAY.
HMPH.
LOOK, I DON'T THINK IT'S TOO MUCH TO ASK THAT WE SIT TOGETHER FOR 40 MINUTES WITHOUT DISTRACTIONS AND INTERRUPTIONS.
RINNGG!
I'LL GET IT! I'M EXPECTING A CALL.
GO AHEAD, DAD. I BELIEVE YOU WERE SAYING SOMETHING FUNNY.
I HAVE ALL THESE GREAT GENES, BUT THEY'RE RECESSIVE. THAT'S THE PROBLEM HERE.
WATTERSON

What Have You Done For Us Lately?

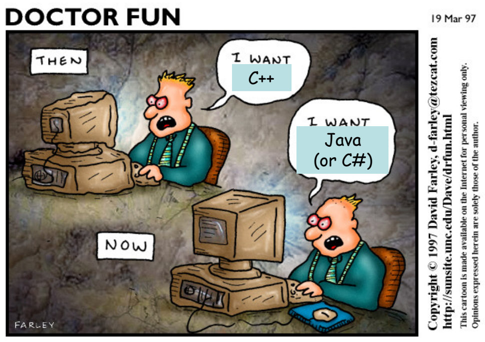
- Isn't PL a solved problem?
  - PL is an old field within Computer Science
  - 1920's: "computer" = "person"
  - 1936: Church's Lambda Calculus (= PL!)
  - 1937: Shannon's digital circuit design
  - 1940's: first digital computers
  - 1950's: FORTRAN (= PL!)
  - 1958: LISP (= PL!)
  - 1960's: Unix
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

Don’t We Already Have Compilers?


DOCTOR FUN
19 Mar 97
THEN
I WANT ICONS!
I WANT ANIMATED ICONS!
NOW
FARLEY
Copyright © 1997 David Farley, d-farley@tezcat.com
http://sunsite.unc.edu/Dave/drfun.html
This cartoon is made available on the Internet for personal viewing only.
Opinions expressed herein are solely those of the author.

Progress

Dismal View Of PL Research


DOCTOR FUN
19 Mar 97
THEN
I WANT C++
I WANT Java (or C#)
NOW
FARLEY
Copyright © 1997 David Farley, d-farley@tezcat.com
http://sunsite.unc.edu/Dave/drfun.html
This cartoon is made available on the Internet for personal viewing only.
Opinions expressed herein are solely those of the author.

Progress

Parts of Computer Science

- CS = (Math × Logic) + 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
  - AI: theorem proving, ML, 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: obvious

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:

  length : α list $\rightarrow$ int (takes an argument of type list of α, returns an integer, for any type α)

  head : α list $\rightarrow$ α

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

References in Standard ML

- Like "updatable pointers" in C
- Type constructor: ptr τ
- Expressions:

  alloc : $\tau \rightarrow$ ptr τ (allocate a cell to store a τ)

  *e : τ when e : ptr τ (read through a pointer)

  *e := e' with e : ptr τ and e' : τ (write through a pointer)

- Works just as you might expect

Polymorphic References: A Major Pain

Consider the following program fragment:

Code	Type inference	
fun id(x) = x	id : $\alpha \to \alpha$	(for any α)
val c = alloc id	c : ptr ($\alpha \to \alpha$)	(for any α)
fun inc(x) = x + 1	inc : int $\to$ int	
*c := inc	Ok, since c : ptr (int $\to$ int)	
(*c) ("hi")	Ok, since c : ptr (string $\to$ string)	

Reconciling Polymorphism and References

- Type system fails to prevent a type error!
- Solutions (examples):
  - weak type variables:
    - polymorphic variables whose instantiation is restricted
    - difficult to use, several failed proofs of soundness
  - value restriction: generalize only the type of values!
    - easy to use, simple proof of soundness

Story: Java Bytecode Subroutines

- Java bytecode programs contain subroutines (jsr) that run in the caller's stack frame
- 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

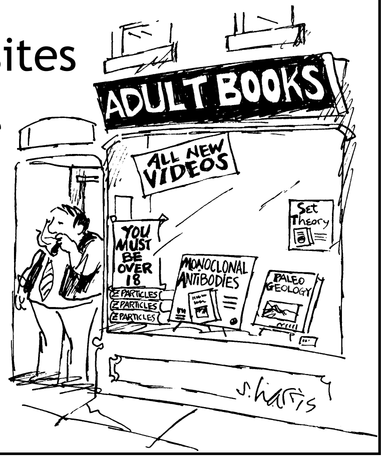
Goal #3

- Understand current PL research (PLDI, POPL, OOPSLA, …)

Final Goal: Fun

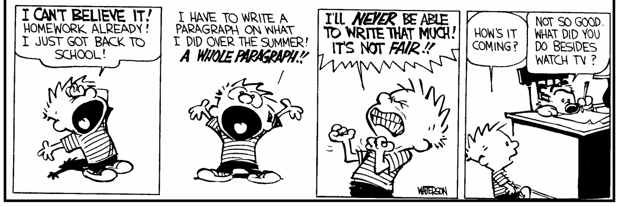
Prerequisites

- Undergraduate compilers course
- “Mathematical maturity”



Assignments

- Short Homework Assignments (~7)
- Scribe Notes (25 / |class_size|)
- Final Project
- Final Exam
- Course Evaluations


I CAN'T BELIEVE IT! HOMEWORK ALREADY! I JUST GOT BACK TO SCHOOL!
I HAVE TO WRITE A PARAGRAPH ON WHAT I DID OVER THE SUMMER! A WHOLE PARAGRAPH!!
I'LL NEVER BE ABLE TO WRITE THAT MUCH! IT'S NOT FAIR!!
HOW'S IT COMING?
NOT SO GOOD. WHAT DID YOU DO BESIDES WATCH TV?
WATTERSON

Homework Problem Sets

- Some material can be “mathy”
- Much like in Calculus, practice is handy
- Short: 3 problems / HW
  - This course has no TA
  - I don’t want to be grading …
- You have one week to do each one

Scribe Notes

- Long-standing CS tradition
- Every class one or two students take detailed notes and latex them up
- I post the notes
- Provides
  - a formal set of course note
  - practice (paper writing, typesetting)
  - study guides

Final Project

- Literature survey, implementation project, or research project
- Write a 10-page paper (a la PLDI)
- Give a 10-15 minute presentation
- On the topic of your choice
  - I will help you find a topic (many examples)
  - Best: integrate PL with your current research

Final Exam

- I reserve the right to make threatening noises about a take-home final exam
- But there’s no TA …
  - Draw your own conclusions
  - Notably, if everyone does quite well on all other assignments, the final becomes unnecessary

How Hard Is This Class?

This Shall Be Avoided

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.
NERF THI
Alright, dungeons. Where are we at?
Dungeons & Dragons Online key Features:
-Dungeons
-Dragons
They're in, sir.
Good. Glad to hear it. And dragons?
Dragons? Come on, guys. Don't tell me we forgot dragons.
I think... Jim was on Dragons.
No, no – I was on Dungeons.
©2006 Mike Krahulik and Jerry Holkins

Programs and Languages

- Programs
  - 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
- Design a new feature “meaning”
- 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 $\Rightarrow$ proof program is safe
- Design a new language feature
- To prevent the sort of mistakes you found
- Write a source-to-source transform
- Your new feature works on existing code

CS 655 - Core Topics

- Operational semantics
- Type theory
- Verification conditions
- Abstract interpretation
- Lambda Calculus
- Concurrency
- Type systems

"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

- Program Verification Using Counterexample-Guided Abstraction Refinement
- Type Systems For Resource Management
- Possibly: Bug Finding And Verification In The Real World (Coverity guest lecture)
- What do you want to hear about?

In Our Next Exciting Episode

$$\frac{\langle e, \sigma\rangle \Downarrow n}{\langle x := e, \sigma\rangle \Downarrow \sigma[x := n]}$$

Def: $\sigma[x:= n](x) = n$
$\sigma[x:= n](y) = \sigma(y)$

$$\frac{\langle b, \sigma\rangle \Downarrow \text{false}}{\langle \text{while } b \text{ do } c, \sigma\rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma\rangle \Downarrow \text{true} \quad \langle c;\ \text{while } b \text{ do } c, \sigma\rangle \Downarrow \sigma'}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma'}$$

Meat Today? Judgments

hypotheses therefore result

$\langle e, \sigma\rangle \Downarrow n$

$\{Z\} \models Z \wedge ((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$

$\Theta; \Delta; \Sigma; \Gamma \vdash x : \Gamma(x)$

Can really show anything. Varies by judgment.

Rules of Inference

$$\frac{\text{Hypothesis}_1 \ \ldots\ \text{Hypothesis}_N}{\text{Conclusion}}$$

$$\frac{\Gamma \vdash b : \text{bool} \qquad \Gamma \vdash e1 : \tau \qquad \Gamma \vdash e2 : \tau}{\Gamma \vdash \text{if } b \text{ then } e1 \text{ else } e2 : \tau}$$

- For any given proof system, a finite number of rules of inference (or schema) are listed somewhere
- Rule instances should be easily checked
- What is the definition of "NP"?

Derivation

$$\dfrac{\dfrac{\dfrac{\Gamma(x) = \mathit{int}}{\Gamma \vdash x : \mathit{int}}\,\text{var} \quad \dfrac{}{\Gamma \vdash 3 : \mathit{int}}\,\text{int}}{\Gamma \vdash x > 3 : \mathit{bool}}\,\text{gt} \qquad \dfrac{\dfrac{\Gamma(x) = \mathit{int}}{\Gamma \vdash x : \mathit{int}}\,\text{var} \quad \dfrac{\dfrac{\Gamma(x) = \mathit{int}}{\Gamma \vdash x : \mathit{int}}\,\text{var} \quad \dfrac{}{\Gamma \vdash 1 : \mathit{int}}\,\text{int}}{\Gamma \vdash x - 1 : \mathit{int}}\,\text{sub}}{\Gamma \vdash x := x - 1}\,\text{assign}}{\Gamma \vdash \texttt{while}\ x > 3\ \texttt{do}\ x := x - 1\ \texttt{done}}\,\text{while}$$

- Tree-structured (conclusion at bottom)
- May include multiple sorts of rules-of-inference
- Could be constructed, typically are not

For Next Time

- See webpage under "Lectures"
- Read Winskel Chapter 2
- Read Hoare article
- Peruse the optional readings
  - You know you want to
- Be ready to do scribe scheduling