A puzzle and a dilemma

Suppose I want to capture/catch/slay crashing programs before they have a chance to run (=static type safety).

Do I let this one go?

```ocaml
let c = ref (fun x -> x) in
c := (fun x -> x + 1)
c := (fun x -> not x)
!c true
```

Meta-Level Information

- Please interrupt at any time!
- It's completely ok to say:
  - I don't understand. Please say it another way.
  - Slow down! Wait, I want to read that!
- Discussion, not lecture

Getting to Know You: “I, ..., wonder …”

Distraction-Free Classroom

- Let's turn off our cell phones and wi-fi

Distraction-Free Classroom

- Laptop users, please sit in the back rows or side

... just imagine that we have class at 30,000 feet
Introductions: Your guide this semester

• Office hours: TR 10:45am-11:45am or when the door is open

Instruction Philosophy

• Do you drive?

Who is the better instructor?

<table>
<thead>
<tr>
<th>Instructor</th>
<th>Pass Rate on Driver's Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>98%</td>
</tr>
<tr>
<td>#2</td>
<td>88%</td>
</tr>
</tbody>
</table>

Who is the better instructor?

<table>
<thead>
<tr>
<th>Instructor</th>
<th>Pass Rate on Driver's Test</th>
<th>Accident Rate in Next Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>98%</td>
<td>20%</td>
</tr>
<tr>
<td>#2</td>
<td>88%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

Instructor #1 focuses on the exam, while instructor #2 focuses on the ideas and skills the student needs for the future.

Focusing on guiding towards understanding with...

• Discussion, discussion, discussion
  - Readings
  - Reflection with "a moment's thought"
  - Homeworks that count on discussion to form new understanding—driven by you!
  - "Redo" policy

About you?

• What do you want to get out of this class?
Today

• What this course is and is not
• Tell some stories
• Goals for this course
• Requirements and grading
• Course summary

• Convince you that PL is cool and useful

“Isn’t PL a solved problem?”

• We have lots of programming languages. Go home?
• What do you think this course is about?

“Isn’t PL a solved problem?”

“Isn’t PL a solved problem?”

New and Better Compilers?
A Dismal View of PL Research

Programming Languages

- Touches most other areas of CS
  - Theory: DFAs, TMs, language theory (e.g., LALR)
  - Systems: system calls, memory management
  - Arch: compiler targets, optimizations, stack frames
  - Numerics: FORTRAN, IEEE FP, Matlab
  - AI: theorem proving, search
  - DB: SQL, transactions
  - Networking: packet filters, protocols
  - Graphics: OpenGL, LaTeX, PostScript
  - Security: buffer overruns, .NET, bytecode, PCC,
  - Computational Biology: pathway models
  - Software Engineering: software quality, development tools
  - Human Computer Interaction: development tools
- Both theory (math) and practice (engineering)

Overarching Theme

- I assert (and shall convince you) that
- PL is one of the most vibrant and active areas of CS research today
  - It is both theoretical and practical
  - It intersects most other CS areas
- You will be able to use PL techniques in your own projects

Goals

Goal 1
Learn to use advanced PL techniques

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 will be able to describe it formally.

Discussion: Language Design

• Languages are adopted to fill a void
  - Enable a previously difficult/impossible application
  - Orthogonal to language design quality (almost)
• Training is the dominant adoption cost
  - Languages with many users are replaced rarely
  - But easy to start in a new niche. Examples:

Why so many languages?

• Many languages were created for specific applications
• Application domains have distinctive (and conflicting) needs
  - which leads to a proliferation of languages.
• Examples:
  - Artificial intelligence: symbolic computation (Lisp, Prolog)
  - Scientific Computing: high performance (Fortran)
  - Business: report generation (COBOL)
  - Systems Programming: low-level access (C)
  - Scripting (Perl, Python, TCL)
  - Distributed systems: mobile computation (Java)
  - Special purpose languages: ...

Why so many languages?

• Examples:
  - AI: symbolic computation (Lisp, Prolog)
  - Scientific Computing: high performance (Fortran)
  - Business: report generation (COBOL)
  - Systems Programming: low-level access (C)
  - Scripting (Perl, Python, TCL)
  - Distributed Systems: mobile computation (Java)
  - Web (PHP)
  - Special purpose languages: ...

Language Paradigms

Loose classification of languages.
• Imperative (Examples? Notion of Computation?)
Language Paradigms

Loose classification of languages.
- Other paradigms with which you have experience?

Language Paradigms

- Imperative
  - Fortran, Algol, Cobol, C, Pascal
- Functional
  - Lisp, Scheme, ML, Haskell
- Object oriented
  - Smalltalk, Eiffel, Self, C++, Java, C#, Javascript
- Logic
  - Prolog
- Concurrent
  - CSP, dialects of the above languages
- Special purpose
  - TEX, Postscript, TrueType, sh, HTML, make

What makes a good language?

- “A good language is one people use”?

What makes a good language?

- Goals almost always conflict.
- Examples:
  - Safety checks cost something in either compilation or execution time.
  - Type systems restrict programming style in exchange for strong guarantees.

What are good language features?

- Simplicity (syntax and semantics)
- Readability
- Safety
- Support for programming large systems
- Efficiency (of execution and compilation)

Designing good languages is hard
Story: The Clash of Two Features

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

Polymorphism (Informal)

- Code that works uniformly on various types of data
- Examples of function signatures:
  - `length : α list → int` (takes an argument of type “list of α”, returns an integer, for any type α)
  - `head : α list → 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: `τ ref`
  - `x : int ref` “x is a pointer to an integer”
- Expressions:
  - `ref : τ → τ ref` (allocate a cell to store a τ, like malloc)
  - `le : when e : τ ref` (read through a pointer, like *e)
  - `e := e' with e : τ ref and e' : τ` (write through a pointer, like *e = e’)
- Works just as you might expect

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 : α → α</code></td>
</tr>
<tr>
<td><code>val c = ref id</code></td>
<td><code>c : (α → α) ref</code></td>
</tr>
<tr>
<td><code>fun inc(x) = x + 1</code></td>
<td><code>inc : int → int</code></td>
</tr>
<tr>
<td><code>c := inc</code></td>
<td>Ok, since <code>c : (int → int) ref</code></td>
</tr>
<tr>
<td><code>(c) (true)</code></td>
<td>Ok, since <code>c : (bool → bool) ref</code></td>
</tr>
</tbody>
</table>

Reconciling Polymorphism and References

- Type system fails to prevent a type error!
- Commonly accepted solution today:
  - value restriction: generalize only the type of values
  - easy to use, simple proof of soundness
  - many “failed fixes”
- To see what went wrong we need to understand semantics, type systems, polymorphism and references

Story: 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
Recall Goal 2

When (not if) you design a language, it will avoid the mistakes of the past, and you will be able to describe it formally.

Goal 3

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

Most Important Goal

Have Lots of Fun!

Requirements

Prerequisites

- "Deep and broad programming experience"  
  - exposure to various language constructs, computational models, and their meaning (e.g., CSCI 3155)  
  - ideal: undergraduate compilers (e.g., CSCI 4555)

- "Mathematical maturity"  
  - we’ll use formal notation to describe the meaning of programs

- If you are an undergraduate or from another department, please see me.

Logistics

- Website  
  http://www.cs.colorado.edu/~bec/courses/csci5535-f13/  
  - readings, assignments, etc.
- Piazza: discussion
- Moodle: grades
- Office hours  
  - TR 10:45am-11:45am or when door is open  
  - ECOT 621
Assignments

- Reading and participation (each meeting)
- Weekly homework (for ~half semester)
- Final exam
- Final project

Reading and Participation

- ~2 papers/book chapter, each meeting
  - Spark class discussion, post/bring questions
- "A moment's thought" on Piazza
  - Post ≥1 substantive comment, question, or answer for each lecture
  - Due before the next meeting
  - Distance students participate more online!

What is "substantive"?

- "Less than a blog post but more than a tweet."
- Some examples:
  - Questions
  - Thoughtful answers
  - Clarification of some point
  - What you think is the main point in the reading set.
  - An idea of how some work could be improved
- Intent: take a moment to reflect on the day's reading/discussion (not to go scour the web)

Homework and Exam

- Homework/Problem Sets
  - Where the "real" learning happens
  - "Math" (logic) + "Programming" (Ocaml)
  - Encourage mastery: "redos"
  - Due Fridays/Saturdays
  - Collaborate with peers (but acknowledge!)
- Final Exam

Final Project

- Options:
  - Research project
  - Literature survey
  - Implementation project
- Write a ~5-8 page paper (conference-like)
- Give a ~15-20 minute presentation
- On a topic of your choice
  - Ideal: integrate PL with your research
- ~Pair projects

Course Summary
Course At-A-Glance

- **Part I: Language Specification**
  - Semantics = Describing programs
  - Evaluation strategies, imperative languages
- **Part II: Language Design**
  - Types = Classifying programs
  - Typed $\lambda$-calculus, functional languages
- **Part III: Applications**

Core Topics

- **Semantics**
  - Operational semantics
    - rules for execution on an abstract machine
    - useful for implementing a compiler or interpreter
  - Axiomatic semantics
    - logical rules for reasoning about the behavior of a program
    - useful for proving program correctness
  - Abstract interpretation
    - application: program analysis
- **Types**
  - $\lambda$-calculus
    - tiny language to study core issues in isolation

Possible Special Topics

- Software model checking
- Object-oriented languages
- Types for low-level languages
- Types for resource management
- Shape analysis

What do you want to hear about?

First Topic: Model Checking

- **Verify properties or find bugs in software**
- Take an important program (e.g., a device driver)
- Merge it with a property (e.g., no deadlocks)
- Transform the result into a boolean program
- Use a model checker to exhaustively explore the resulting state space
  - Result 1: program provably satisfies property
  - Result 2: program violates property "right here on line 92,376!"

For Next Time

- Join the Piazza and introduce yourself
  - Write a few sentences on why you are taking this course
- Read the two articles on SLAM
  - see the website under "Schedule"