Getting Started in Programming Language Design Research

Object-Oriented and Imperative Languages

William Cook
Univertisy of Texas at Austin
More about Me

• I've lived in nine states
  – MI, PA, IL, MA, FL, LA, RI, CA, TX

• Worked many jobs
  – HP, Apple, BAM!, Net-It, Allegis, UT
  – (ask about the $60M in venture capital)

• Married w/8-year old son Miles
  – My wife Robin is a user-interface designer

• Research interests
  – Programming languages & databases
  – Structured concurrency/workflow
  – Model-driven & feature-oriented programming
Picking Topics

• Train your “spider senses”
  – As undergraduates, we work around problems
    • Also true of most programming tasks
  – As grad students, focus on the problems

• Do something
  ... even if it doesn't seem big enough
  – In the doing, you may bump into a bigger problem

• Solve someone else’s problem
  – Someone else = person outside PL
  – Learn about other areas in CS, and outside CS
Picking Topics

• Best: direct contact with problem area
  – Get your hands dirty

• Theory
  – Theory does not tell you what to do
  – It helps guide/constraint/analyze

• Learn 2 or 3 things deeply
  – Opportunities are found at the interfaces

• Find problem and solution together
Criteria Different by Topic

• **Types**
  – Proof of soundness (mechanically checked!)

• **Language runtime (systems)**
  – Implementation, performance

• **Language design papers**
  – Good motivation, examples

• **Analysis papers**
  – Proof, implementation, complexity, performance

• **Garbage Collection**
  – careful experiment design and measurement
Use the Tools

- Semantics and Analysis
- Language Theory Substrates
- Implementation Substrates
- Verification and mechanization
- (Runtimes... in another talk)
- Practical Tools
- Evaluation + Packaging
Foundation

• **Denotational Semantics**
  - good for intuition
  - $\lambda$-calculus

• **Operational Semantics**
  - Small-step
    - good for proofs
  - Large-step
    - natural, easy interpreters
    - recent proof techniques
Combination of Two Areas

- Denotational Semantics + Objects

![Diagram showing relationships between objects and modifications]

- Object: $A$
- Modification: $\Delta(A)$
- Self-reference: $Y(G)$
- Inheritance: $\Delta(Y(G))$

$\Delta \circ G$
The Tools

- **Abstract Interpretation**
  - compute over abstract values - types, properties, states

- **Static analysis**
  - lightweight formal methods
  - typing
  - shape analysis
  - ownership

- **Model checking**

- **Partial evaluation**
Language Substrates

- **Featherweight Java (Pierce’s book)**
  - functional subset of java

- **Lightweight Java**
  - [http://www.cl.cam.ac.uk/~rs456/lj/](http://www.cl.cam.ac.uk/~rs456/lj/)
  - imperative, true subset of java
  - comes with formalization in Isabelle/HOL

- **ClassicJava**
  - imperative
  - also support for mixins
Implementing Java Extensions

• Polyglot
  – Widely used
  – Complex plug-in model

• JastAdd
  – Newer
  – Based on declarative attribute grammars
Thought Tools

- **Galois connections**
  - loose isomorphism between ordered sets
- **Fixed points**
  - recursion and induction
- **Linear Types**
  - control over resources
- **Abstract data types and algebra**
  - Contrast with objects
- **Bisimulation**
  - equivalence of processes
- **Datalog**
  - data query/transformation
- **Attribute grammars**
  - declarative static analysis
- **Category theory**
  - theory of structure (some say: “content-free”)
Language Environments

• Implementations are convincing

• Ott tool
  – Generates Isabelle/HOL specs (an Latex)
  – Includes formalization of Lightweight Java

• PLT Redex
  – Based on PLT Scheme
  – domain-specific language (DLS)
  – specifying and debugging operational semantics

• Eclipse
  – For refactoring/Development tools
Mechanical Proof Checking

• Theorem provers
  – Isabelle/HOL
  – Twelf
  – ACL2
  – PVS

• Specification languages
  – Alloy, Maude, Z

• POPLMark Challenge
  – Challenge problems for “mechanizing meta-theory”
Practical Tools

- **Latex**
  - Inference Rules: Pierce’s “bcprules.sty”
  - Presentations: PP or Latex??

- **Graphics**
  - OpenOffice → eps, IPE → esp

- **Use CVS for collaboration**
  - Latex too --- 5 author papers
  - Eclipse interface

- **Unison file sync**

- **Shell script!**
  - running tests
  - gathering results
Learn new skills

You are here because you are good at something

To be successful, need to be good at a range of skills
Tasks

- Managing a small business
- Presenting your work
- Get funding
- Starting projects
- Accounting
- How to really dig into unfamiliar territory
- Finding topics to work on
- How to skim
- Managing a team

Understand your work style

~~

Compensate for deficiencies
my profile

• Quick thinking, sometimes too quick
• A programmer at heart
• Use theory as a tool
• Relentless, creative, intuitive
• Know what can be proven
• Proofs themselves are not easy
• Struggle with email (time management)
• Not a “born” writer, bad spelling
• How do you think I compensate?
What do you have to produce to get a PhD?
No,
it’s not your thesis
It is you