how do i do research

matthias felleisen, racketeer.
plt. northeastern
how do I do research?

would

if I were you

how do I do research?
\[ \sigma_c^2 < \left( \frac{\sigma}{\phi_f} \right) - \sigma_\mu^2 + \frac{3ln\epsilon_f}{\pi^2 \beta_f} \]
how do I work with my PhD students?

how would I do research if I were you?

how do I do research?
how do i relate to my PhD students
I have *never*, ever hired a PhD student. Period.

Instead my students and I find a topic we both love.
Functional I/O (ICFP '09)

Typed Racket (ICFP '10)

Compiler Coaching (OOPSLA '12)

Laziness, what is it (good for)? (JFP 1996)
And that’s what’s called ‘doing research.’
how would I do research if I were you
Two Case Studies

Asumu Takikawa

Tony Garnock-Jones

Kuhn, *The Structure of Scientific Revolution*
Types for Classes

Typed Racket
(ICFP '10)
dozens of classes, 100s of methods, and he equipped all of them with “lightweight contracts”
Previous Topic: Contracts for Classes & Objects
A Positive (Self-perpetuating) Feedback Loop
truly functional GUIs

messages as events

communicating worlds

from freshman programs to systems

actors

networks

publish subscribe

failures!

message brokers
Functional I/O & Communicating Worlds

Networking Systems

DNS
Proxy
SSH
Server
Chat
Room
TCP
Stack

DSL for comm. actors
Topic: *Coordinated Concurrent Functional Language*

- CCFL over Racket
- CCFL over JavaScript

**Does it specialize?**
- **Does it generalize?**
- **Is it performant?**

**Design**
- The Network Calculus

**Implement**
- Base actors in distinct languages

**Evaluate**
- Coordination over the “real” network

**Improve**
What is the cost of breaking open a new field?

5 years

6.5 years
how did I do research as a PhD student
My Story

Dan Friedman

(f (g (call/cc k))
  =
  (k (λ (x) (f (g x)))))

Go, implement it. See what happens.
What does it mean to implement equations

I had read that paper. ... in two hours.

I read it again. NOT 4 hours

I spent 4 MONTHS studying this paper.
What does it mean to implement equations
What did four months of reading yield

How do calculi correspond to eval?
- start from an abstract syntax
- identify values & programs
- define basic notion of reduction
- inductively generate theories
- eval-> and eval=
- Church & Rosssser Thm.
- Thm. eval-> = eval=
- Standard Reduction Theorem
- Thm: eval-standard = eval->
My dissertation: “This” works for imperative features, too.

How do calculi correspond to eval?
- start from an abstract syntax
- identify values & programs
- define basic notion of reduction
- inductively generate theories
- eval-> and eval=
- Church & Rosser Thm.
- Thm. eval-> = eval=
- Standard Reduction Theorem
- Thm: eval-standard = eval->
Lessons

Know to distinguish the good from the bad in your advisor’s suggestions.

Good paper require ‘deep study’ not just a ‘reading.’

Really good paper are ‘research programs’ not just results.
how do I do research now
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
problem I can solve
More papers does not mean better researcher.
Think *big*, think long-term.

Lesson

Good researchers say “no” to many problems. They focus on those that they care about.
My Long-term Projects

How can programmers design programs systematically? (1985)

How do types fit into untyped languages? (1988)

How do you teach 12, 14, 16 year olds programming and what benefit does this have? (1995, last day of POPL)

What is linguistic power and why is a DSL better than an algorithm? (1985)
What do such long-term projects look like?

How do you launch long-term projects?
What do such long-term projects look like?
### The “Gradual Typing” Dissertations

<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>Mike Fagan</td>
<td>Soft Typing (*** )</td>
</tr>
<tr>
<td>1994</td>
<td>Andrew Wright</td>
<td>Practical Soft Typing</td>
</tr>
<tr>
<td>1998</td>
<td>Cormac Flanagan</td>
<td>Componental SBA</td>
</tr>
<tr>
<td>2002</td>
<td>Robby Findler</td>
<td>Higher-order Contracts</td>
</tr>
<tr>
<td>2005</td>
<td>Philippe Meunier</td>
<td>Modular SBA from Contracts</td>
</tr>
<tr>
<td>2006</td>
<td>Sam Tobin-H. (2010)</td>
<td>From Scripts to Programs</td>
</tr>
<tr>
<td>2012</td>
<td>Stevie Strickland</td>
<td>Contracts for First-class Classes</td>
</tr>
<tr>
<td>2015</td>
<td>Asumu Takikawa</td>
<td>Types for First-class Classes</td>
</tr>
<tr>
<td>Year</td>
<td>Event</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>F’1992</td>
<td>Teaching my first introductory programming course</td>
<td></td>
</tr>
<tr>
<td>1993/94</td>
<td>@ CMU, “Bob is teaching it all wrong”</td>
<td></td>
</tr>
<tr>
<td>Jan 1995</td>
<td>Launch TeachScheme! — FP and algebra in high schools</td>
<td></td>
</tr>
<tr>
<td>Dec 1995</td>
<td>The Dump (~1,000 pages) — re-focusing on explicit design</td>
<td></td>
</tr>
<tr>
<td>1996-2001</td>
<td>HtDP/1e — Programs as Functions</td>
<td></td>
</tr>
<tr>
<td>2002-2005</td>
<td>Designing, implementing, evaluating</td>
<td></td>
</tr>
<tr>
<td>2007-2015</td>
<td>HtDP, 2nd ed. — Programs are not functions</td>
<td></td>
</tr>
</tbody>
</table>
What is linguistic power and why is a DSL better than an algorithm? (1985)

<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1985</td>
<td>with Kohlbecker et al</td>
<td>Hygienic Maros</td>
</tr>
<tr>
<td>1986</td>
<td>with Bruce Duba</td>
<td>Macros in Phases</td>
</tr>
<tr>
<td>88/89</td>
<td>John Greiner, Steve Weeks</td>
<td>Programming Abstract Syntax</td>
</tr>
<tr>
<td>1991</td>
<td>Todd Yonkers</td>
<td>Extensible Syntax</td>
</tr>
<tr>
<td>1994</td>
<td>Matthew Flatt</td>
<td>Connecting DSLs into Applications</td>
</tr>
<tr>
<td>95/97</td>
<td>PLT</td>
<td>Teaching languages</td>
</tr>
<tr>
<td>95/99</td>
<td>Shriram Krishnamurthi</td>
<td>Parameterizing over Language</td>
</tr>
<tr>
<td>2002</td>
<td>Matthew Flatt</td>
<td>You want it when?</td>
</tr>
<tr>
<td>03/08</td>
<td>Ryan Culpepper</td>
<td>Protecting Macros</td>
</tr>
<tr>
<td>08/09</td>
<td>Ryan Culpepper</td>
<td>Debugging Macros</td>
</tr>
</tbody>
</table>
How do you launch long-term projects?
Sometimes you stumble into a topic.
### The “Gradual Typing” Dissertations

How do types fit into untyped languages? (1988)

<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>Mike Fagan</td>
<td>Soft Typing (***).</td>
</tr>
<tr>
<td>1994</td>
<td>Andrew Wright</td>
<td>Practical Soft Typing</td>
</tr>
<tr>
<td>1998</td>
<td>Cormac Flanagan</td>
<td>Componential SBA</td>
</tr>
<tr>
<td>2001</td>
<td>Robby Findler</td>
<td>Higher-order Contracts</td>
</tr>
<tr>
<td>2005</td>
<td>Philippe Meunier</td>
<td>Modular SBA from Contracts</td>
</tr>
<tr>
<td>2006</td>
<td>Sam Tobin-H. (2010)</td>
<td>From Scripts to Programs</td>
</tr>
<tr>
<td>2012</td>
<td>Stevie Strickland</td>
<td>Contracts for First-class Classes</td>
</tr>
<tr>
<td>2015</td>
<td>Asumu Takikawa</td>
<td>Types for First-class Classes</td>
</tr>
</tbody>
</table>
Sometimes it is love at first sight.
How can programmers design programs systematically?

How do you teach 12, 14, 16 year olds programming? What benefit does it have? (1995, last day of POPL)

<table>
<thead>
<tr>
<th>Year</th>
<th>Event/Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1985</td>
<td>The Little Lisper, 2nd ed.</td>
</tr>
<tr>
<td>F'1992</td>
<td>Teaching my first introductory programming course</td>
</tr>
<tr>
<td>1993/94</td>
<td>@ CMU, “Bob’s teaching it all wrong”</td>
</tr>
<tr>
<td>Jan 1995</td>
<td>Launch TeachScheme! — FP and algebra in high schools</td>
</tr>
<tr>
<td>Dec 1995</td>
<td>The Dump (~1,000 pages) — re-focusing on explicit design</td>
</tr>
<tr>
<td>1996-2001</td>
<td>HtDP/1e — Programs as Functions</td>
</tr>
<tr>
<td>2002-2005</td>
<td>Designing, implementing, evaluating “Functional I/O”</td>
</tr>
<tr>
<td>2007-2015</td>
<td>HtDP, 2nd ed. — Programs are not just “functional I/O”</td>
</tr>
</tbody>
</table>

We knew what we had to do: software, curriculum, teaching.

An “entertaining” thought.

Cormac asked the one critical question.
Sometimes it develops as a necessity.
What is linguistic power and why is a DSL better than an algorithm? (1985)

<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1985</td>
<td>with Kohlbecker et al</td>
<td>Hygienic Maros</td>
</tr>
<tr>
<td>1986</td>
<td>with Bruce Duba</td>
<td>Macros in Phases</td>
</tr>
<tr>
<td>88/89</td>
<td>John Greiner, Steve Weeks</td>
<td>Programming Abstract Syntax</td>
</tr>
<tr>
<td>1991</td>
<td>Todd Yonkers</td>
<td>Extensible Syntax</td>
</tr>
<tr>
<td>1994</td>
<td>Matthew Flatt</td>
<td>Connecting DSLs into Applications</td>
</tr>
<tr>
<td>95/97</td>
<td>PLT</td>
<td>Teaching languages</td>
</tr>
<tr>
<td>95/99</td>
<td>Shriram Krishnamurthi</td>
<td>Parameterizing over Language</td>
</tr>
<tr>
<td>2002</td>
<td>Matthew Flatt</td>
<td>You want it when?</td>
</tr>
<tr>
<td>03/08</td>
<td>Ryan Culpepper</td>
<td>Protecting Macros</td>
</tr>
<tr>
<td>08/09</td>
<td>Ryan Culpepper</td>
<td>Debugging Macros</td>
</tr>
</tbody>
</table>
what to remember?
As a student, you need to
— develop a sense of the landscape
— follow your heart
— plan out design, implementation, evaluation.

As a researcher, I
— look for long-term projects
— follow my heart
— use teaching (for the 99%) for inspiration
— develop dissertation-size goals
— plan for hand-over
— and have my eyes open for new ideas.

No matter what, keep in mind that the number of your papers is *unrelated* to the quality of your work.
The End