Programming Languages Research at the University of Colorado, Boulder
PL research at CU has breadth!

How do we effectively express computation? language design, type systems, logic

How do we assist reasoning about programs? program analysis, development tools

How do we make programs run efficiently? performance analysis, compilation

How do we get reliable, secure software? verification, model checking
PL researchers at CU collaborate!

- Language design
- Gradual Programming
- Dynamic Algorithmic Complexity
- Preventing Resource Exhaustion Attacks
- Program analysis
- Performance analysis
- Verification

You?

Devin Coughlin
Application: Auto Code Improvement

Suggestion: using Java.util.HashMap instead of Java.util.List will give you a speedup. Click here to see more details.

Have I used the right data structures for my tasks?
Computational Complexity

AVL Trees, Fibonacci heaps, $O(n^2)$, P=NP?

Memory hierarchy: Caches, page faults, register allocation,..

Have I used the right data structures?
Have I used the right data structures?

Class MyContainer

void addElement(Element x);

Element chooseElt (...);

Element findMatch (...);

void printSorted (...);

<table>
<thead>
<tr>
<th>Function</th>
<th>Hashtable</th>
<th>Balanced Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>addElement</td>
<td>$O(1)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>chooseElt</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>findMatch</td>
<td>$O(N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>findMaximum</td>
<td>$O(N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>printSorted</td>
<td>$O(N \log N)$</td>
<td>$O(N)$</td>
</tr>
</tbody>
</table>
How is the library being used?

Usage Profile for MyContainer

<table>
<thead>
<tr>
<th>Function</th>
<th>Usage Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>addElement</td>
<td>70%</td>
</tr>
<tr>
<td>chooseElt</td>
<td>12%</td>
</tr>
<tr>
<td>findMatch</td>
<td>5%</td>
</tr>
<tr>
<td>findMax</td>
<td>10%</td>
</tr>
<tr>
<td>prettyPrint</td>
<td>3%</td>
</tr>
</tbody>
</table>

Conclusion: Use HashTable

But wait, what would your systems professor say?
Dynamic Complexity Estimation

Parameterized Unit Tests:
Design unit test suites to simulate usage pattern and vary input size.

Runtime System with Performance Monitoring

\[1.2n^2mk + 3.5nm - 351\]
What function to fit?

• Static analysis of *complexity trends*.  
  - Using invariant + ranking function generation.

```
for (i=0; i < N; ++i)
  for (j=0; j < i; ++j)
    foo(x[i][j], N);
```

```
while (i < N)
  if (...) 
    i = i * 3;
  else 
    i = i * 2;
```

\[ O(N^2) \]  \[ O(N^2) \]  \[ O(\log N) \]
Application: Dynamic Algo. Selection

Dynamically select the best algorithm.

Strassen’s Matrix Multiplication Algorithm

“AP Computer Science” algorithm

$O(N^{2.38...})$

$N > 100000$

$O(N^3)$
Application: System Security

- *Denial of Service Attacks* can exploit high complexity worst case.

<table>
<thead>
<tr>
<th></th>
<th>Worst-Case</th>
<th>Average-Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Sort</td>
<td>$O(N^2)$</td>
<td>$O(N \log N)$</td>
</tr>
</tbody>
</table>

Malicious User ➔ Remote Server (run quicksort over user input)
Challenges and Opportunities

Practice

Runtime Monitoring
Static Analysis
Specification Formalisms
Compiler Optimizations

Tools that the programmer can use.

Theory

Complexity
Linear Programming
Monte-Carlo Simulations
Randomized Algorithms

Exciting Ideas

Tools that the programmer can use.

Practice

Runtime Monitoring
Static Analysis
Specification Formalisms
Compiler Optimizations

Tools that the programmer can use.

Theory

Complexity
Linear Programming
Monte-Carlo Simulations
Randomized Algorithms

Exciting Ideas
PL research at CU is successful!

**PLDI 2010** (2)  Toronto, Canada

Khoo, Chang, Foster. *Mixing Type Checking and Symbolic Evaluation.*

**POPL 2010** (2)  Madrid, Spain

Harris, Sankaranarayanan, Ivancic, Gupta. *Program Analysis via Satisfiability Modulo Path Programs.*

**ESOP 2010**  Cyprus

Laviron, Chang, Rival. *Separating Shape Graphs.*

< 20% acceptance rate  < 27% acceptance rate
PL research at CU is successful!

**ASPLOS 2009**  
Washington, DC  
Mytkowicz, Diwan, Hauswirth, Sweeney. *Producing wrong data without doing anything obviously wrong!*

**OOPSLA 2009**  
(2) Orlando  
von Dincklage, Diwan. *Optimizing programs with intended semantics.*  
Mytkowicz, Coughlin, Diwan. *Inferred call-path profiling.*

**ASE 2009**  
Auckland, New Zealand  

**CAV 2009**  
Grenoble, France  
Kanade, Alur, Sankaranarayanan et al. *Generating and analyzing symbolic traces of Simulink/Stateflow models.*
PL research at CU is **successful**!

**ESOP 2009**  
York, UK  
Siek, Garcia, Taha. *Exploring the design space of higher-order casts.*

**TACAS 2009**  
York, UK  
Kahlon, Sankaranarayanan, Gupta. *Semantic reduction of thread interleavings in concurrent programs.*

**CC 2009**  
York, UK  
Knights, Mytkowicz, Sweeney, Mozer, Diwan. *Blind optimization for exploiting architectural features.*

and more ...

Papers ⇒ Travel + PhD
PL research at CU has 
world-wide collaborations!

Collaborators ⇒
Internships and Jobs
PL students have *interned* at …
After *graduation*, PL students have gone to ...
The PL group has **fun** together!

Group meetings at the **Boulder Tea House** twice a month

Travel to **conferences** (Todd at OOPSLA’09)

Our mentoring: Guide you to research that **excites** you!
Our group

Devin
Weiyu

Postdoc
Todd

PhD
Robert
Jonathan

Daniel

MS
Amer
Jeremy

Faculty

You?
You?

BS
James

Evan
Sriram
Some of our other research projects

- Understanding performance
- Program metamorphosis
- Lightweight data collection
- Blind optimization
- Algorithmic optimizations
- Validating architectural simulators
- Using non-linear dynamics to understand computer systems
- Tools for teaching programming languages
- End-user program analysis
- Post-mortem analysis and error reporting
- Security policies for power-grids
- Analysis of web languages
- Modeling and validating building security policies
- Confident program analysis
- Checking low-level code
- Generic programming
- Meta-programming
- Gradual type checking
- High-level optimizations for memory efficiency
- Finding bugs in parallel programs
- Cyber-physical systems verification
- And soon projects created by you!