Visualizing Compiler Passes with LastPass*
Paulette, Braxton, and Jonathan (PB & J)

* formerly SecondPass†
† formerly FirstPass‡
‡ formerly Untitled Compiler Pass Visualization
Visualizing Compiler Passes with LastPass*

what?

* formerly SecondPass†
† formerly FirstPass‡
‡ formerly Untitled Compiler Pass Visualization
What are compiler passes?

A Very Nice Program

...compile

how?

Machine Gunk
What are compiler passes?

A Very Nice Program

…

pass

A Very Nice Program With Small Change

…
What are compiler passes?

A Very Nice Program With Small Change

\[ \cdots \]  

pass

\[ \cdots \]  

A Very Nice Program With Small Change
Multi-pass compiler

A Very Nice Program

...  pass

A Very Nice Program With Small Change

...  pass

...  pass

Machine Gunk
Goal of CPSC 411

A High-Level Functional Program

... pass ...

A High-Level Functional Program With Small Change

... pass ...

... pass ...

Assembly
understanding an individual pass
understanding an individual pass
analyzing its effect on different program expressions
Tasks

understanding an individual pass
analyzing its effect on different program expressions
identifying compiled expressions in context
identifying and comparing complex compiler passes
identifying and comparing complex compiler passes
comparing amount of generated code
Tasks

identifying and comparing complex compiler passes
comparing amount of generated code
identifying structural changes
CPSC 411 reference compiler: 33 passes!
LastPass: 2 passes
**LastPass Scope**

1. **a-normalize**
   
   \( (\ast \ (\ + \ 2 \ 3) \ 5) \)
   
   \( \text{let} \ (\text{tmp} \ (\ + \ 2 \ 3)) \)
   
   \( \ (\ast \ \text{tmp} \ 5) \)

2. **select-instructions**
   
   \( \text{set!} \ \text{tmp}.99 \ 2 \)
   
   \( \text{set!} \ \text{tmp}.100 \ (\ + \ \text{tmp}.99 \ 3) \)
   
   \( \text{set!} \ \text{tmp} \ \text{tmp}.100 \)
   
   \( \text{set!} \ \text{tmp}.101 \ (\ast \ \text{tmp} \ 5) \)
   
   \( \text{set!} \ \text{rax} \ \text{tmp}.101 \)
   
   \( \text{jump} \ \text{ra}.98 \ \text{rbp} \ \text{rax} \)
Data: Programs

“(+ 2 3)”

Abstract Syntax Tree (AST)
directed acyclic graph

user input

internally

visually

(+ 2 3)
Data: Pass

input AST

output AST
3 program ASTs
source program AST
output AST from a-normalize
output AST from select-instructions

2 sets of directed edges
a-normalize and select-instructions

exactly what we want to visualize!
false positives and negatives on complex passes

- a-normalize doesn’t move a lot of code, but requires a new style of programming to implement

scale, 33 >> 2

pinning overload

- pinning too many flows blends colours
LastPass: A tool for CPSC 411

query multiple test programs

see an overview of compilation changes

compare programs after any number of passes

track how expressions change in context
Thank you!

https://se.cs.ubc.ca/compiler-viz/index.html