Programming Languages and Compilers (CS 421)



William Mansky

http://courses.engr.illinois.edu/cs421/

Based in part on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, Elsa Gunter, and Dennis Griffith

The Big Picture

- OCaml/Functional Programming
- Continuation Passing Style
- Type checking/inference
- Unification
- Lexing
- Parsing
- Operational Semantics
- Lambda Calculus
- Hoare Logic



Anatomy of an Interpreter

Program Text

Lexing regular expressions/lex

Token Buffer

Parsing BNF grammar/yacc

AST

Static Semantics

Correct AST

type checking/inference, unification possibly Hoare Logic!

Dynamic Semantics operational semantics

Program Output

Anatomy of a Compiler

Program Text Lexing regular expressions/lex Token Buffer Parsing BNF grammar/yacc **AST** type checking/inference, unification **Static Semantics** possibly Hoare Logic! **Correct AST** denotational semantics **Translation** possibly CPS! IR **Optimization** Optimized IR Code Generation (denotational semantics again) Assembly/Machine Code (or lambda calculus!)

4

What can we do now?

- Write functional programs (which parallelize well and compute math quickly)
- Build compilers and interpreters
- Specify and implement programming languages
- Prove programs correct
- Compare various approaches to PL syntax and semantics

Formal Methods

- The study of proving programs correct
- But what is a program?
 - Anything we can model formally/mathematically
 - Computer programs, machines, workflows
 - Compilers, air traffic control protocols, security protocols, simulators, Java programs
- And how do we prove it correct?
 - Syntax, semantics, formal logic, automatic/interactive provers

Program Text

Lexing

Token Buffer

Parsing

AST

Static Semantics

Correct AST

Translation

IR

Optimization

Optimized IR

Code Generation

"do the same thing"

Assembly/Machine Code

Program Text

Lexing

Token Buffer

Parsing

AST

Static Semantics

Correct AST

Translation

IR

Optimization

Optimized IR

Code Generation

have the same semantics?

Assembly/Machine Code

Program Text

Lexing

Token Buffer

Parsing

AST

Static Semantics

Correct AST

Translation

IR

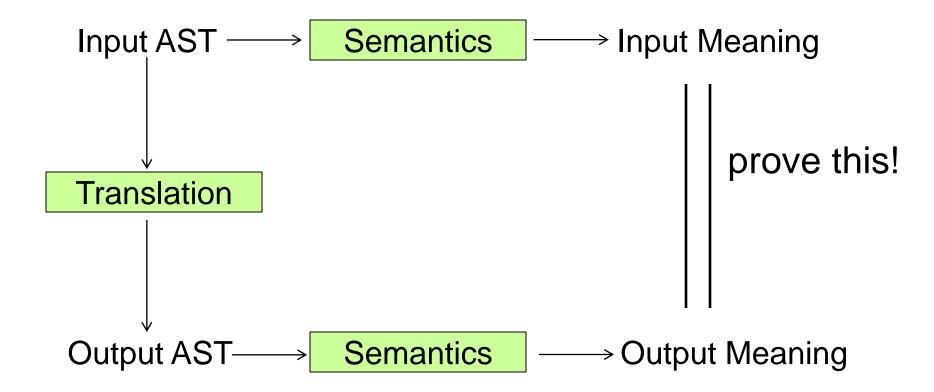
Optimization

Optimized IR

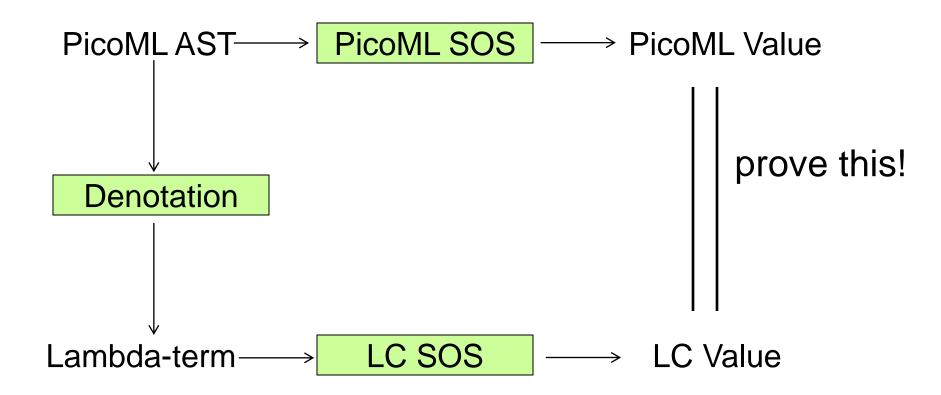
Code Generation

have the same semantics

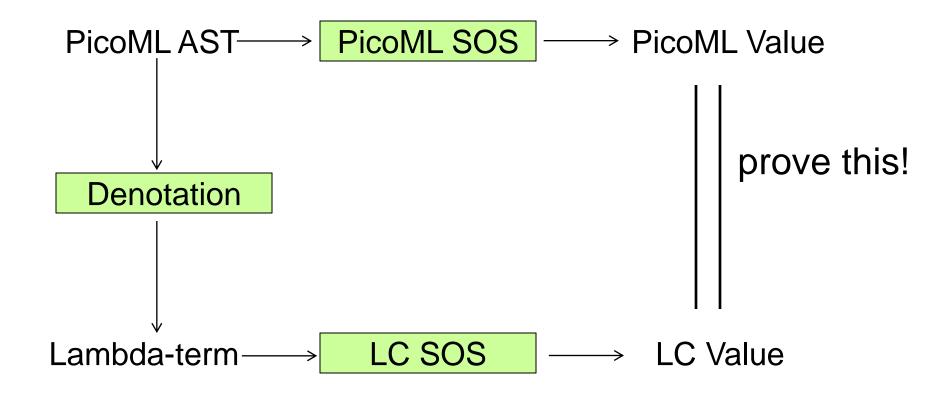
Assembly/Machine Code



Verifying a Compiler – Example

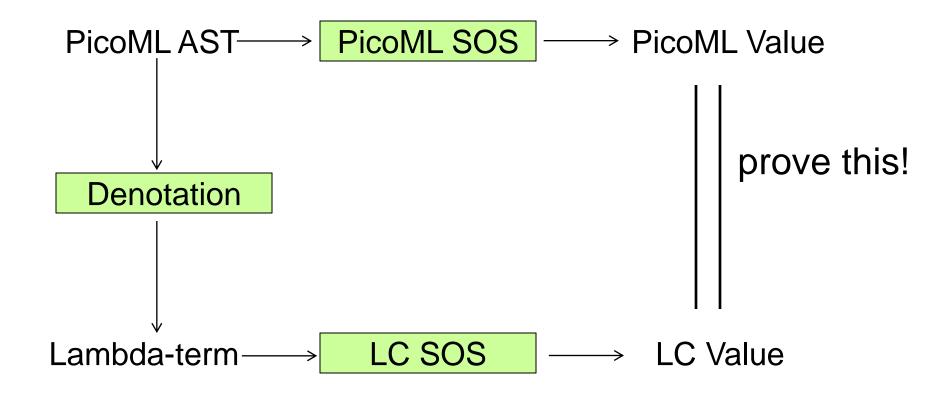


Verifying a Compiler – Example



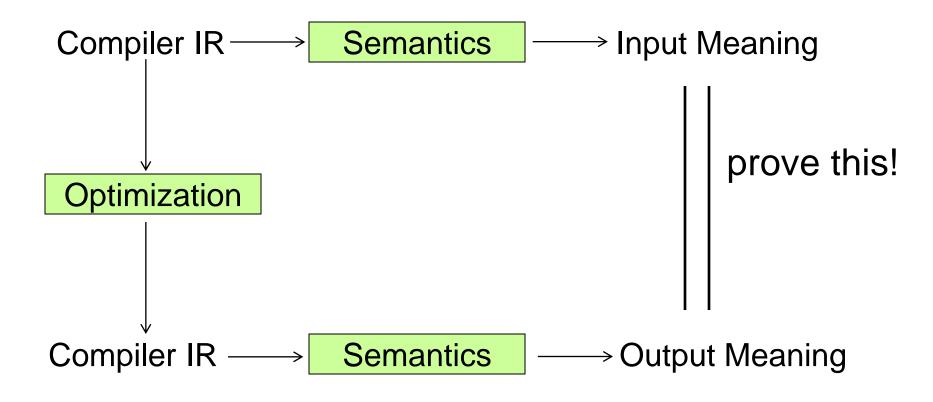
Can we prove this for every program?

Verifying a Compiler – Example



Can we prove this for every program? Yes.

Verifying a Compiler – Harder Case



Can we prove this for every program?

Control Flow Graphs

- ASTs are good for complex expressions
- What if we care more about control flow than expression structure?
- Graphs with commands on nodes, edges showing control flow

4

Control Flow Graphs

$$0: x := 1$$

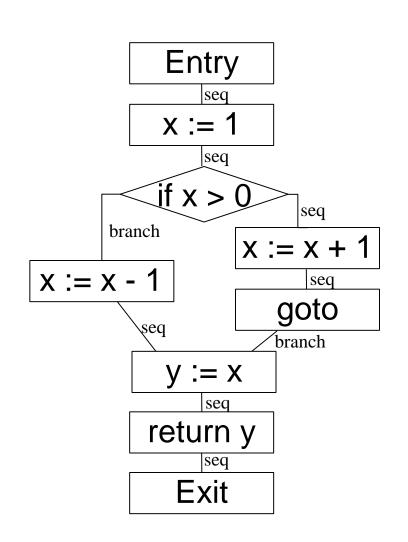
1: if
$$x > 0$$
 goto 4

2:
$$x := x + 1$$

4:
$$x := x - 1$$

5:
$$y := x$$

6: return y



Semantics on Control Flow Graphs

- Configurations are (G, n, m), where G is a graph and n is the currently executing node
- Small-step for moving between nodes, bigstep within a single node

E.g. (Assignment Rule):

$$\frac{|\text{abel}(G, n) = I := E \quad (E, m) \downarrow v \quad \text{out}(n) = n'}{(G, n, m) \rightarrow (G, n', m[x \leftarrow v])}$$

Verifying a Compiler – Harder Case

