



Compilers

Semantics Overview

- We must specify for every Cool expression what happens when it is evaluated
 - This is the “meaning” of an expression
- The definition of a programming language:
 - The tokens \Rightarrow lexical analysis
 - The grammar \Rightarrow syntactic analysis
 - The typing rules \Rightarrow semantic analysis
 - The evaluation rules
 - \Rightarrow code generation and optimization

- We have specified evaluation rules indirectly
 - The compilation of Cool to a stack machine
 - The evaluation rules of the stack machine
- This is a complete description
 - Why isn't it good enough?

- Assembly-language descriptions of language implementation have irrelevant detail
 - Whether to use a stack machine or not
 - Which way the stack grows
 - How integers are represented
 - The particular instruction set of the architecture
- We need a complete description
 - But not an overly restrictive specification

- Many ways to specify semantics
 - All equally powerful
 - Some more suitable to various tasks than others
- Operational semantics
 - Describes program evaluation via execution rules
 - on an abstract machine
 - Most useful for specifying implementations
 - This is what we use for Cool

- Denotational semantics
 - Program's meaning is a mathematical function
- Axiomatic semantics
 - Program behavior described via logical formulae
 - If execution begins in state satisfying X , then it ends in state satisfying Y
 - X, Y formulas
 - Foundation of many program verification systems