Lecture 04 (Outline of a Compiler)

1 Semantic Analysis

1. try and understand the meaning
2. Undecidable - similar to equivalence checking
3. Limited semantic analysis
    - catch inconsistencies
    - no effort at understanding the meaning

Some use cases are:

1. Scoping
2. Type checking

2 Optimization

Modify program without changing meaning

1. Run faster
2. Use less memory
3. Reduce power
4. Reduce network messages
5. Reduce disk accesses

2.1 Precise Semantic Modelling

1. $(2 \times i)/2 \not\Rightarrow i$ because of overflow
2. $z = 0 \times y \not\Rightarrow z = 0$ if y is a float
3. $for(i = 0; i < n + 1; i + +) \not\Rightarrow for(i = 0; i <= n; i + +)$ if i is unsigned
4. $for(i = 0; i < n + 1; i + +) \Rightarrow for(i = 0; i <= n; i + +)$ if i is signed

3 Code Generation

1. ISA aware
2. Some optimization phases are also ISA aware

4 Compiler Steps

1. Lexing
2. Parsing
3. Semantic analysis
4. Optimization
5. Code generation