CS614: Advanced Compilers

Intermediate Representations

Manas Thakur
CSE, IIT Bombay

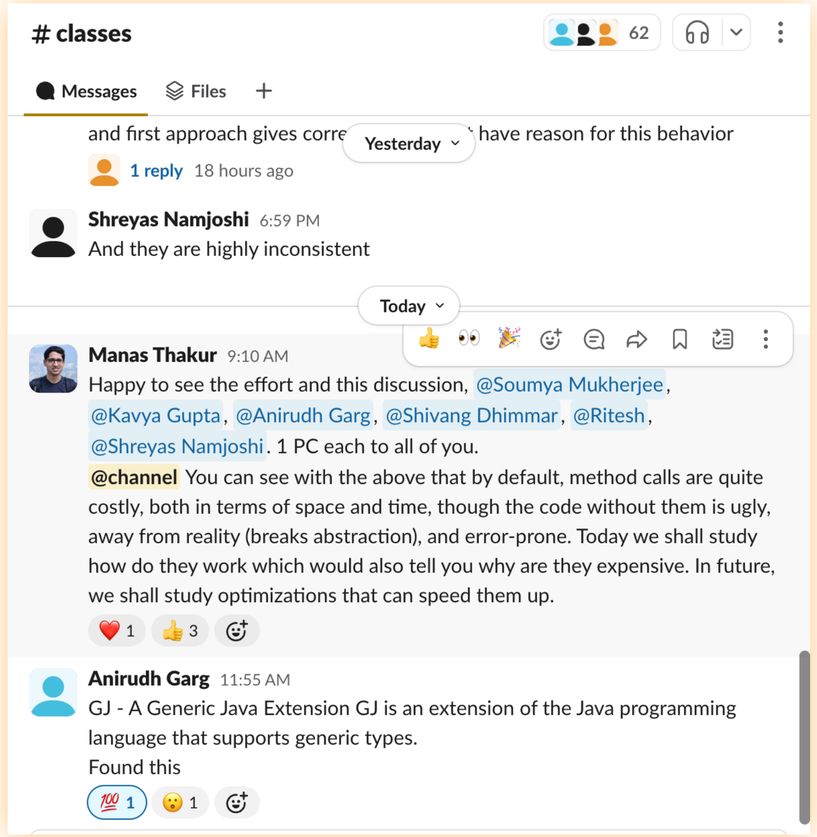
Spring 2025

Things we learnt in the last class

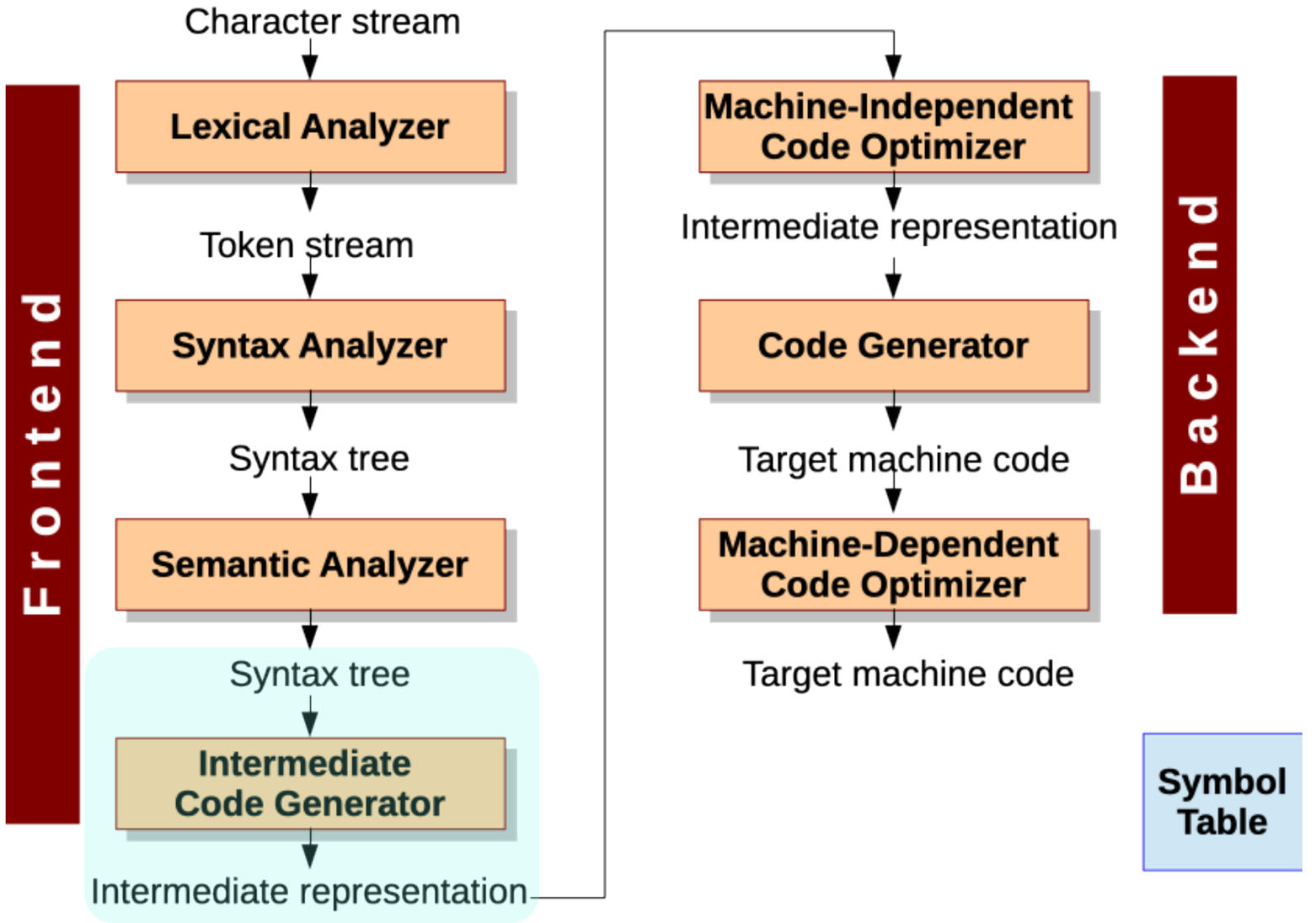
- Review of the first 3 stages of a compiler
- BNF notation of context-free grammars
- AST generation and traversal
- A few good OO design principles
- A bit of Java fun

  Experimentation complements Theory

- Calculator using JavaCC, JTB and Visitor Pattern
- Slack can give you PCs



Typical phases in a compiler



Roles of IRs

- Act as a glue between front-end and back-end
  - Or source and machine codes
- Lower abstraction from source level
  - To make life simple
- Maintain some high-level information
  - To keep life interesting
- Make the dream of `m+n` components for `m` languages and `n` platforms look like a possibility
  - Scala to Java Bytecode, for example
- Enable machine-independent optimization
  - Next phase

Typical Kinds of IRs

- Structural
  - Graph oriented
  - Heavily used in IDEs, source-to-source translators
  - Tend to be large

Examples:
ASTs, DAGs

- Linear
  - Pseudo-code for an abstract machine
  - Level of abstraction varies
  - Simple, compact data structures

Examples:
3AC, Bytecode

- Hybrid
  - Combination of graphs and linear code

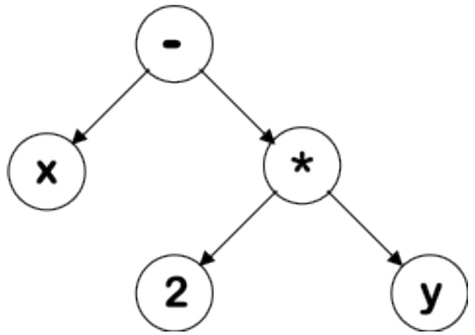
Examples:
CFGs, Sea of nodes

Abstract Syntax Tree (AST)

- Parse tree with some intermediate nodes removed

```
x - 2 * y
```

becomes

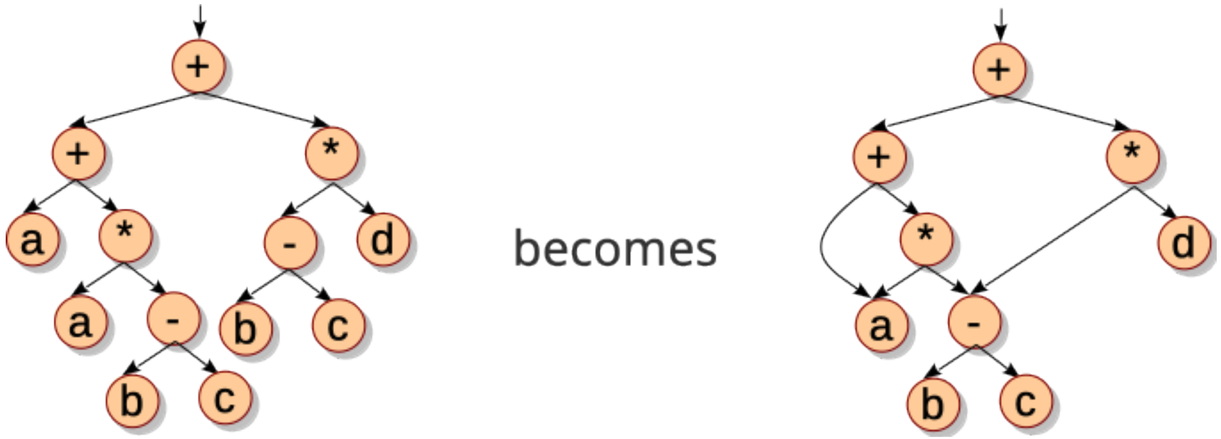


- **Advantages:**
  - Easy to evaluate
    - Postfix form: `x 2 y * -`
    - Useful for interpretation
  - Source code can be reconstructed
    - Helpful in program understanding

Directed Acyclic Graph (DAG)

- AST with a unique node for each value

```
a + a * (b - c) + (b - c) * d
```



- **Advantages:**
  - Compact (reduces redundancy)
  - Won't have to evaluate the same expression twice

Three-Address Code (TAC or 3AC)

- At most
  - three addresses (names/constants) in the instruction
  - one operator on the right hand side of assignment
- General statement form: `x = y op z`
- Longer expressions are simplified by introducing temporaries

```
z = x - 2 * y
```

becomes

```
t1 = 2 * y
t2 = x - t1
z  = t2
```

or

```
t1 = 2 * y
z  = x - t1
```

- **Advantages:**
  - Easy to understand
  - Names for intermediate values

Convert to 3AC

```
r = a + a * (b – c) + (b – c) * d
```

```
t1 = b - c
t2 = t1 * d
t3 = b – c
t4 = a * t3
t5 = t4 + t2
r  = a + t5
```

```
if (x < y) S1 else S2
```

```
t1 = x < y
if !t1 goto L1
S1
goto L2
L1: S2
L2:
```

```
while (x < 10) S1
```

```
L1: c = x < 10
t = !c
if !t goto L2
S1
goto L1
L2:
```

IR Generation for OO Languages: **Java Case Study**

- Field Resolution
- Method Resolution
- Inheritance and its Effects

Fields are bound statically, methods dynamically!

Next Class

Case Study on Java Bytecode