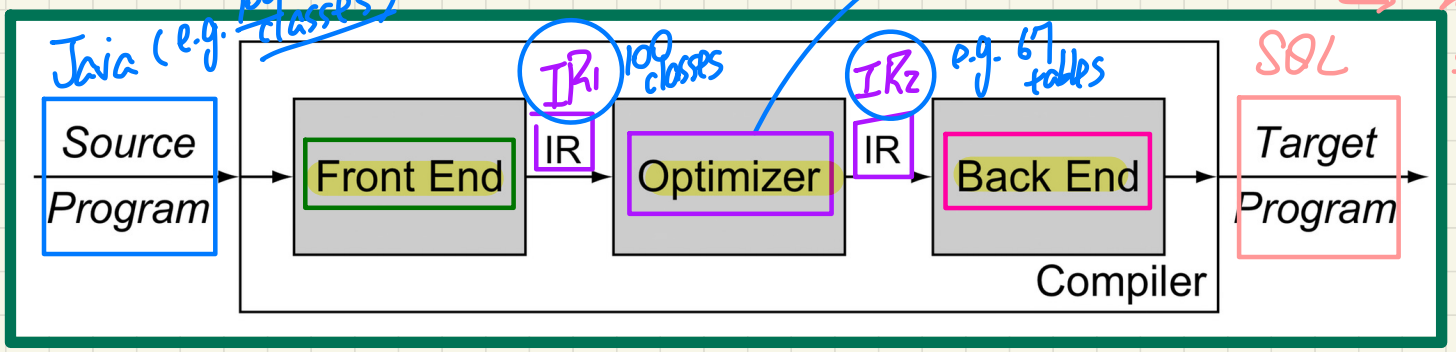
Lecture 3 - May 13

Overview of Compilation

Infrastructure:
Front-End, Optimizer, Back-End
Stages:
Lexical, Syntactical, Semantic Analyses
Routines:
Scanning vs. Parsing

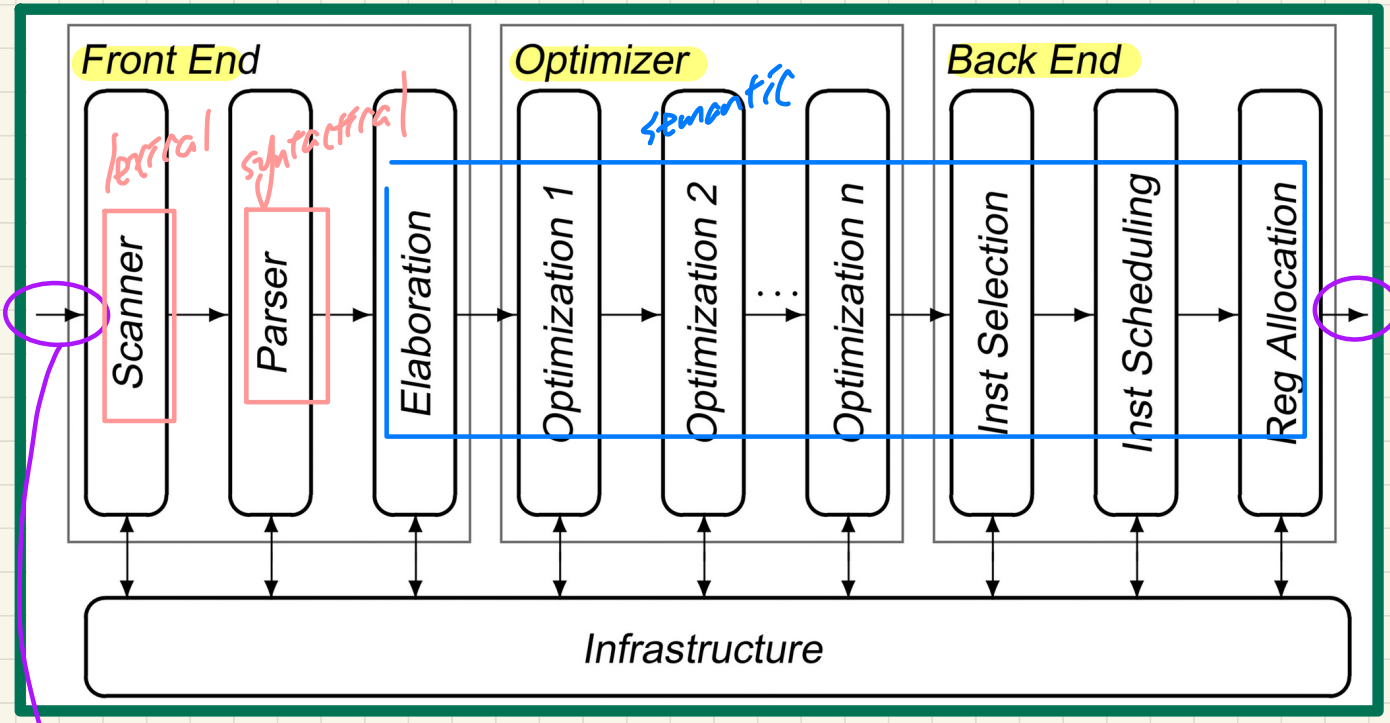
Compiler: Typical Infrastructure (2)

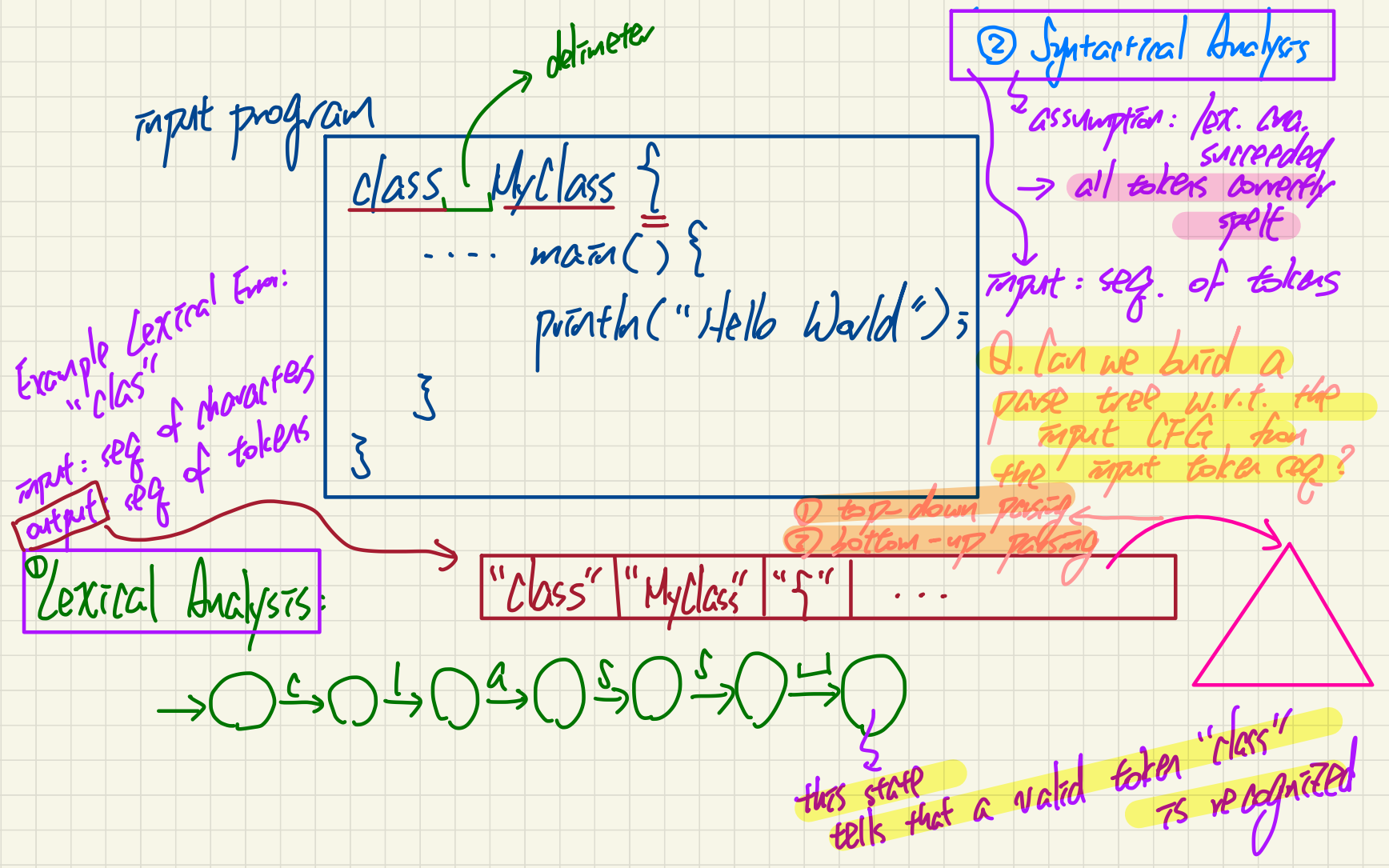


Q. What does the behaviour of the target program depend upon?

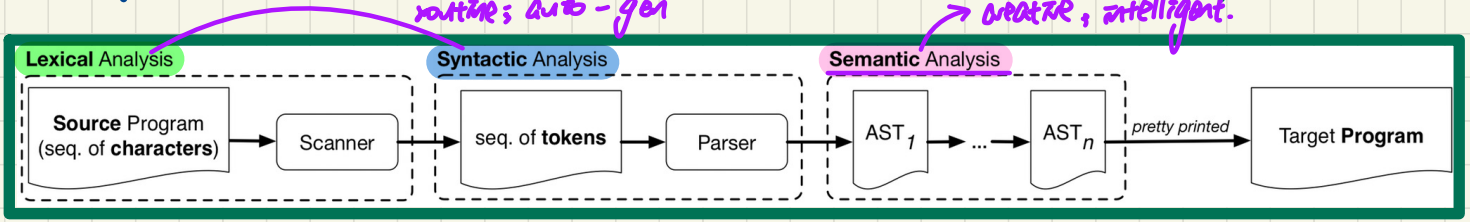
1. front-end accurately represents the input program in IR_1
2. optimizer preserves the semantics/meaning from IR_1 to IR_2
3. back-end accurately represents the (optimized) IR_2 in output.

Example Compiler 1: Infrastructure




input program
delimeter
class MyClass {
.... main() {
printf("Hello World");
}
}
② Syntactical Analysis
assumption: lex. ana. succeeded
→ all tokens correctly spelt
input: seq. of tokens
Q. Can we build a parse tree w.r.t. the input CFG from the input token seq?
① top-down parsing
② bottom-up parsing
Example Lexical Error:
"clas"
input: seq of characters
output: seq of tokens
① Lexical Analysis
"class" | "MyClass" | "{" | ...
c l a s s
this state tells that a valid token "class" is recognized

Compiler Infrastructure: Scanner, Parser, Optimizer



Analogy: Compare Compilation to Essay Writing

Introduction

Contemporary technologies in today's information society are not merely an institutional system, instead, they are a system of material objects designed by those who intend to exercise the social requirements and their hegemonic purposes: command, control, and exploitation. In this essay, one main thesis – contemporary technologies are not neutral – will be revealed by first looking at how Feenberg's notions of dialectical technological rationality and technical code provide a generic template for explaining how technologies can combine the social and political requirements under a particular capitalist social context, and then examining two different standings on arguing the "un-neutrality" of technologies: While Margolis and Resnick argue for the ethical ideas, Winner, Goodman, McDermott, and Robins and Webster argue against the blamable messages embedded within technologies.

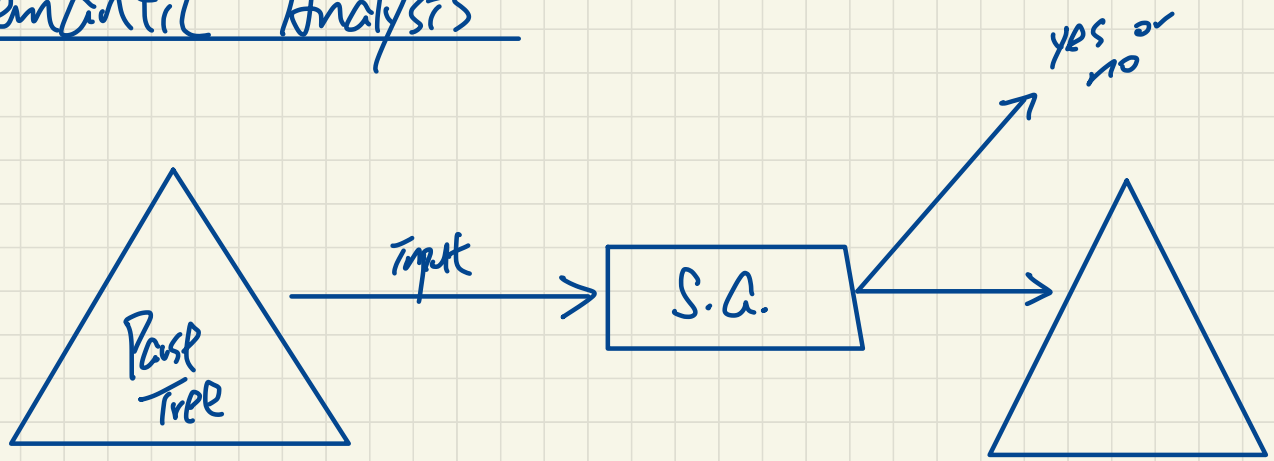
Summaries of Arguments from Sources

In his work, Cressman (2004) describes how Feenberg develops his notions of dialectical technological rationality and his concept of the technical code based on Marx's technological ambivalence and Marcuse's technological rationality. Feenberg's technical code can be defined as the general rule of integrating social requirements and the technical advancement into a single technological artifact, which frequently binds technological applications to hegemonic purposes (Cressman 2004). Based on Marx's notion of "design critique" of technology, Feenberg claims that the contemporary social system of capitalism has shaped the sort of technology we are using and even guides what we will have in the future. A capitalist system mainly requires the control over the majority of the working class, and hence division of the labour force is implemented, and

- words → lexical (spellings)
- sentences → syntactic
  - e.g. I love tennis. ✓
  - e.g. I tennis. ✗
- meaning → consistency, contradiction

I love tennis.
I hate racquet sport.

Semantic Analysis



parse tree (PT)
≈
abstract syntax tree (AST)

while-Loop: Context-Free Grammar (CFG)

```
WhileLoop   ::=  WHILE LPAREN BoolExpr RPAREN LCBRAC Impl RCBRAC
Impl        ::=
              |  Instruction SEMICOL Impl
```

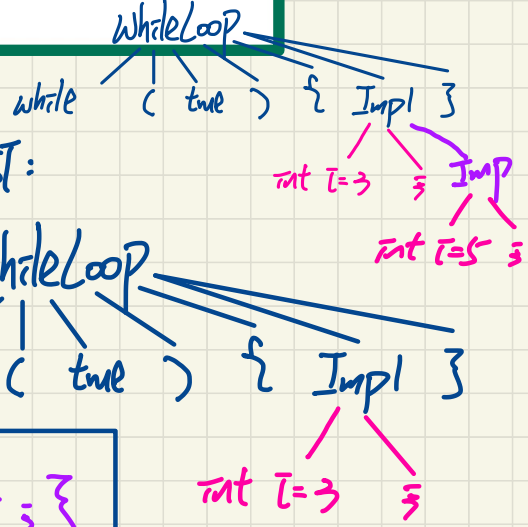
recursive def.

Input (1): `while(true) { int i = 3; }`

pass syntactical analysis

Input (2): `while(true) { int i = 3; int i = 5; }`

PT/AST:



(1) & (2) passed to semantic analysis: e.g. type checking

(2) declared twice → not semantically correct

Compiler Infrastructure: AST-to-AST Optimizer (1)

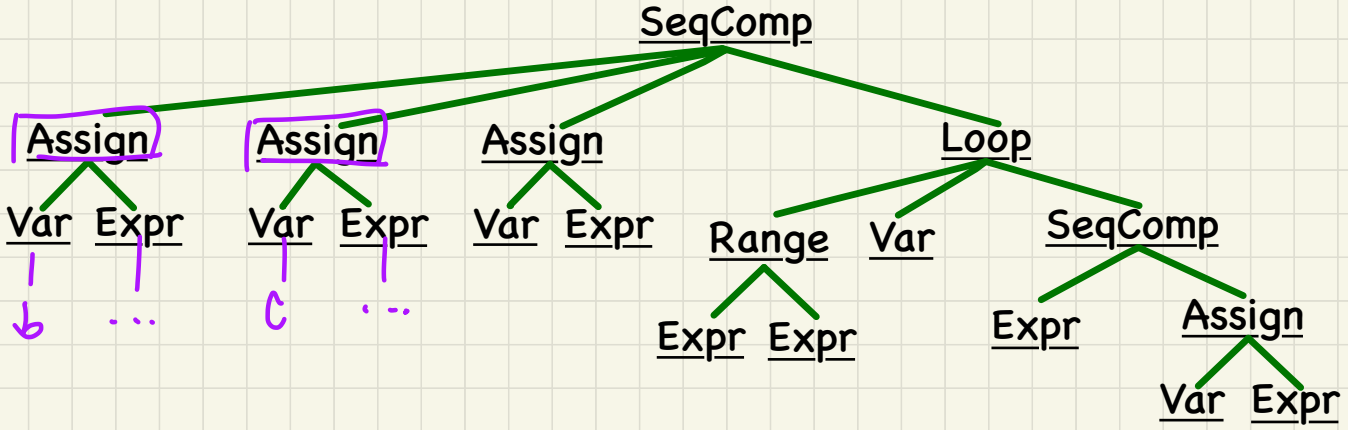
```
b := ... ; c := ... ; a := ...
across 1 |..| n is i
  loop
    read d
    a := a * 2 * b * c * d
  end
```

1, 2, 3, ..., n

remains "invariant" between iterations.

→ syntactically correct.

AST of input program:



Compiler Infrastructure: AST-to-AST Optimizer (2)

```
b := ... ; c := ... ; a := ...
temp := 2 * b * c
across 1 |..| n is i
  loop
    read d
    a := a * temp * d
  end
```

temp's value calculated only once outside the loop.

AST of output program:

Compiler Infrastructure: AST-to-AST Optimizer (3)

commuting diagram.

Q. How should the various artifacts be connected?

```
b := ... ; c := ... ; a := ...
across i |..| n is i
  loop
    read d
    a := a * 2 * b * c * d
  end
```

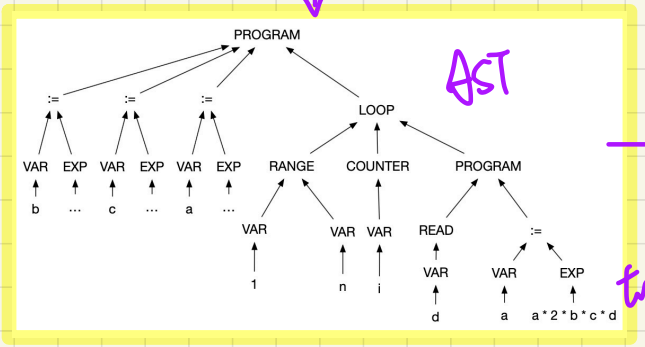
input (intermediate prog)

```
b := ... ; c := ... ; a := ...
temp := 2 * b * c
across i |..| n is i
  loop
    read d
    a := a * temp * d
  end
```

parsed

prog-to-prog compilation (ext. use)

pretty-printing.



(optimization)

IR-to-IR transformation