Static Program Analysis
Part I of IV

Automated Static Analysis

- A static analyzer is a software tool for source code text processing
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V&V/Testing team
- Very effective as an aid to inspections.
- A supplement to but not a replacement for inspections

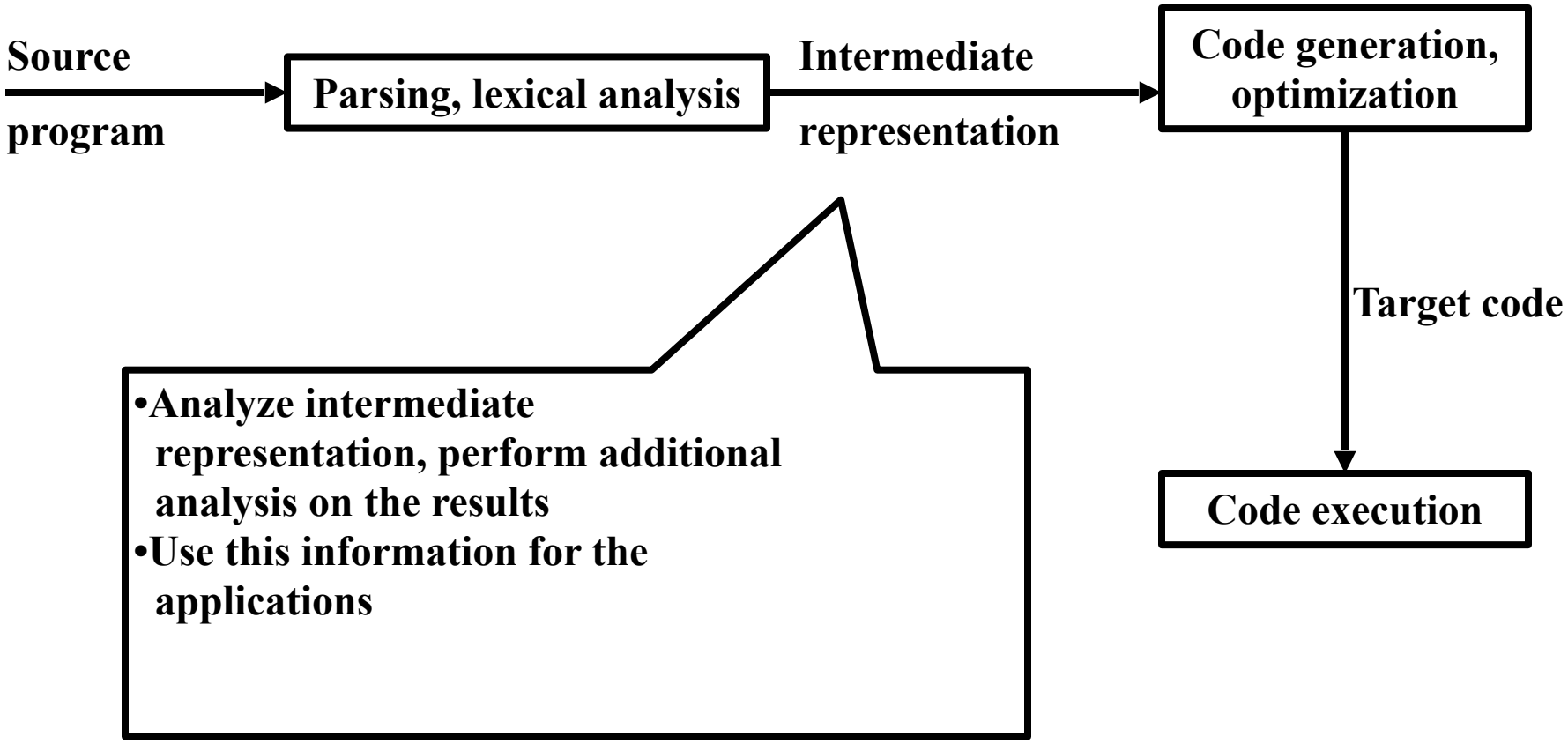
Types of Static Analysis Checks

Fault Type	Static Analysis Check
Data	• Variables used before initialization • Variables declared but never used • Variables assigned twice but never used between assignments • Possible array bound violations • Undeclared variables
Control	• Unreachable code • Unconditional branches into loops
I/O	• Variables output twice with no intervening assignment
Interface	• Parameter type mismatches • Parameter number mismatches • Non-usage of results of functions • Uncalled functions
Storage management	• Unassigned pointers • Pointer arithmetic

Static Models of the Source Code

- Low level
  - Source code text
- Intermediate level
  - Symbol table
  - Parse tree
- High level
  - Control flow
  - Data flow
  - Program Dependency Graph
- Design Level
  - Class diagram
  - Sequence diagram

Starting Point for Static Analysis



Intermediate Representation

- Parse (derivation) Tree & Symbol Table
- Concrete Parse Tree
  - Concrete (derivation) tree shows structure *and* is language-specific issues
  - Parse tree represents concrete syntax
- Abstract Syntax Tree/Graph (AST)/(ASG)
  - Abstract Syntax Tree shows only structure
  - Represents abstract syntax

AST vs Parse Tree

Example

1. a := b + c

2. a = b + c;

- Grammar for 1
  - stmtlist → stmt | stmt stmtlist
  - stmt → assign | if-then | …
  - assign → ident “:=“ ident binop ident
  - binop → “+” | “-” | …

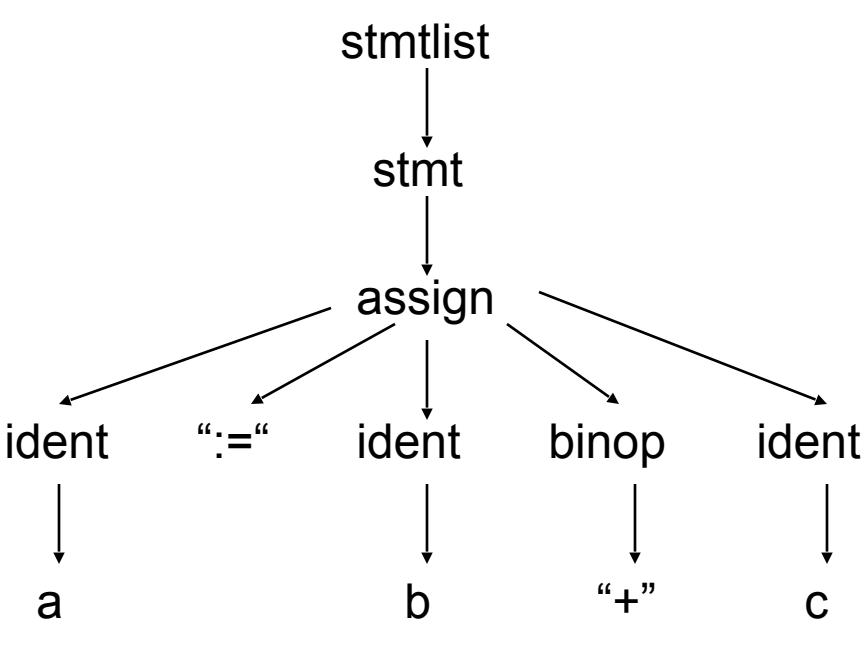
- Grammar for 2
  - stmtlist → stmt “;” | stmt”;” stmtlist
  - stmt → assign | if-then | …
  - assign → ident “=“ ident binop ident
  - binop → “+” | “-” | …

Parse Trees

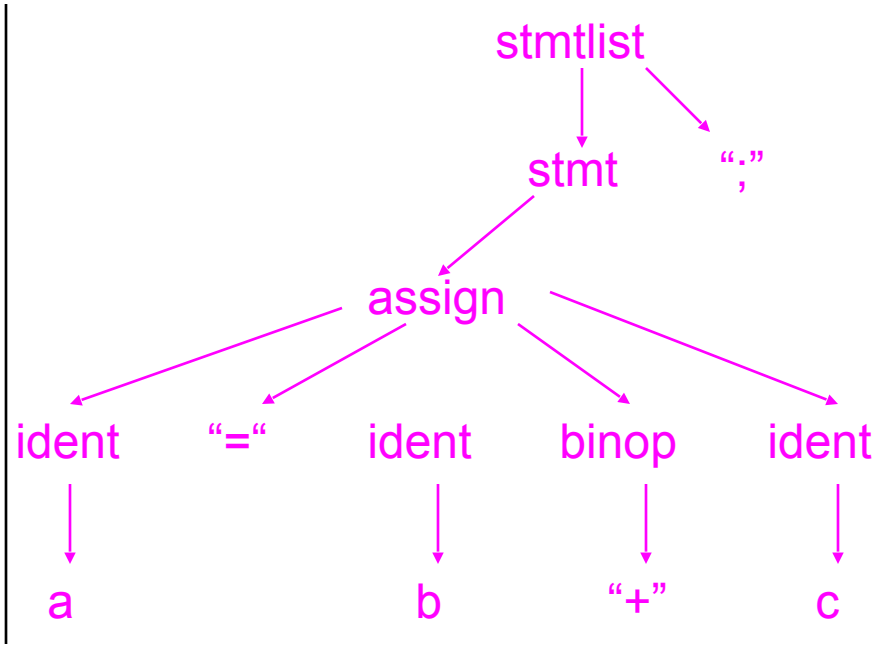
Example

1. a := b + c
2. a = b + c;

Parse Tree for 1



Parse Tree for 2

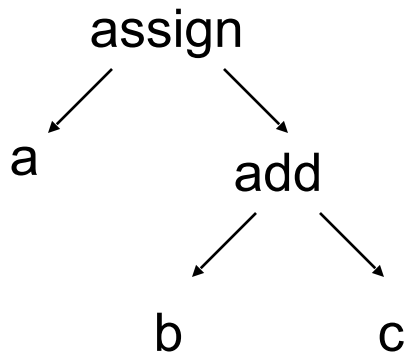


AST

Example

1. a := b + c
2. a = b + c;

Abstract syntax tree for 1 and 2

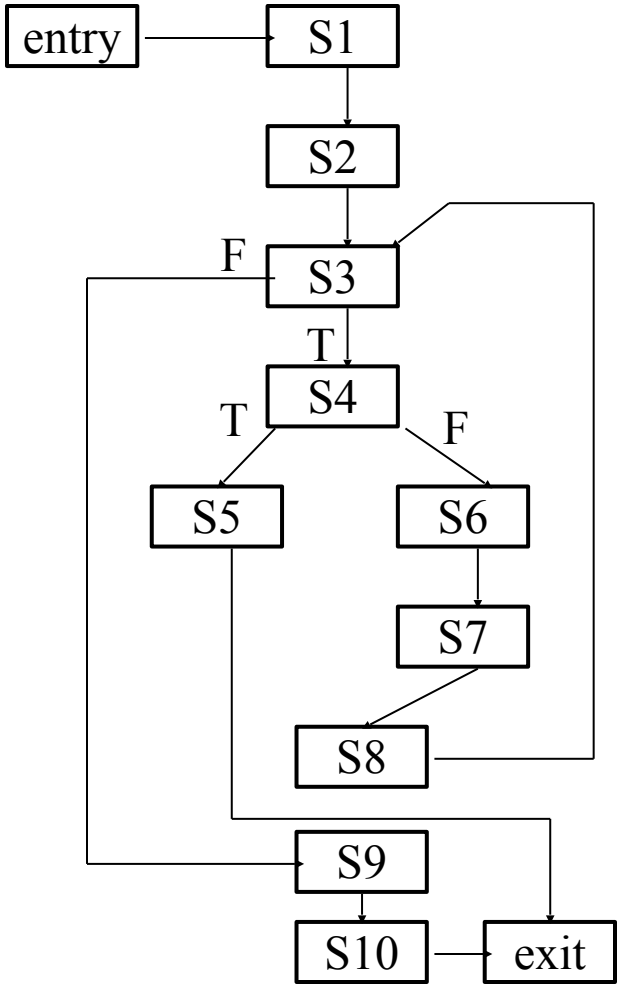


Intermediate to High level

- Given
  - Source code
  - AST
  - Symbol table
- One can construct
  - Call graphs
  - Control flow graph
  - Data flow
  - Slices

Control Flow Analysis (CF)

```
Procedure AVG
S1    count = 0
S2    fread(fptr, n)
S3    while (not EOF) do
S4       if (n < 0)
S5          return (error)
         else
S6          nums[count] = n
S7          count ++
         endif
S8       fread(fptr, n)
       endwhile
S9    avg = mean(nums,count)
S10   return(avg)
```

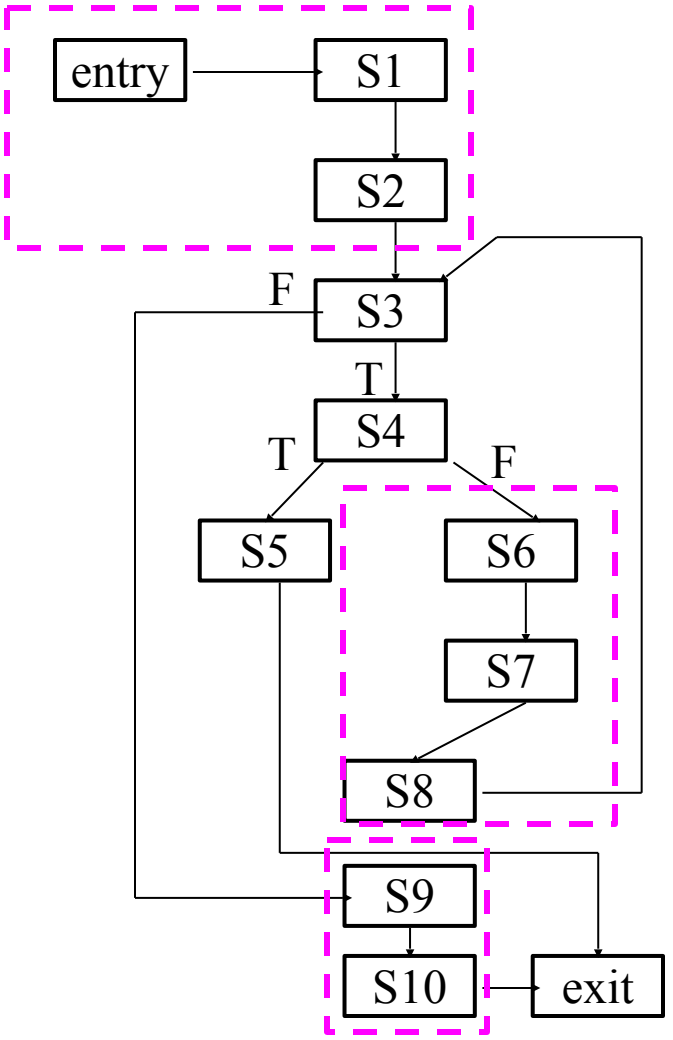


Computing Control Flow

- Basic blocks can be identified in the AST
- Basic blocks are straight line sequence of statements with no branches in or out.
- A basic block may or may not be “maximal”
- For compiler optimizations, maximal basic blocks are desirable
- For software engineering tasks, basic blocks that represent one source code statement are often used

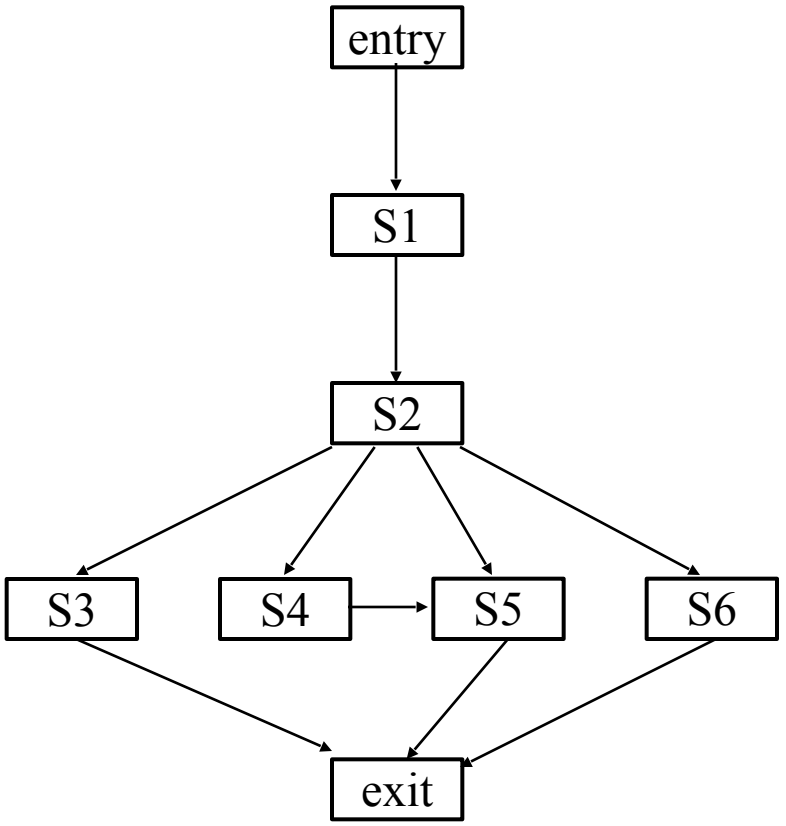
Computing Control Flow

```
Procedure AVG
S1    count = 0
S2    fread(fptr, n)
S3    while (not EOF) do
S4       if (n < 0)
S5          return (error)
         else
S6          nums[count] = n
S7          count ++
         endif
S8       fread(fptr, n)
       endwhile
S9    avg = mean(nums,count)
S10   return(avg)
```



Computing Control Flow

```
Procedure Trivial
S1    read (n)
S2    switch (n)
        case 1:
S3        write (“one”)
          break
        case 2:
S4        write (“two”)
        case 3:
S5        write (“three”)
          break
        default
S6        write (“Other”)
      endswitch
end Trivial
```



Computing Control Flow

```
Procedure Trivial
S1    read (n)
S2    switch (n)
        case 1:
S3        write (“one”)
          break
        case 2:
S4        write (“two”)
        case 3:
S5        write (“three”)
          break
        default
S6        write (“Other”)
      endswitch
end Trivial
```