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

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Static Analysis Check</th>
</tr>
</thead>
</table>
| **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

Source program → Parsing, lexical analysis → Intermediate representation → Code generation, optimization

- Analyze intermediate representation, perform additional analysis on the results
- Use this information for the applications

Target code → Code execution
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
Example
1. $a := b + c$

2. $a = b + c$;

• Grammar for 1
  - stmtlist $\rightarrow$ stmt $|$ stmt stmtlist
  - stmt $\rightarrow$ assign $|$ if-then $|$ ...
  - assign $\rightarrow$ ident "$=\$" ident binop ident
  - binop $\rightarrow$ "+" $|$ "-" $|$ ...

• Grammar for 2
  - stmtlist $\rightarrow$ stmt ";" $|$ stmt ";" stmtlist
  - stmt $\rightarrow$ assign $|$ if-then $|$ ...
  - assign $\rightarrow$ ident "$=\$" ident binop ident
  - binop $\rightarrow$ "+" $|$ "-" $|$ ...
Parse Trees

Example
1. \( a := b + c \)
2. \( a = b + c; \)

Parse Tree for 1

```
stmtlist
  | stmt
  | assign
  | ident "=" ident binop ident
  | a "=" b "+" c
```

Parse Tree for 2

```
stmtlist
  | stmt
  | assign
  | ident "=" ident binop ident
  | a "=" b "+" c
```

Identifiers:
- ident: a, b, c

Binary Operators:
- binop: "=" and "+"

Syntax of a statement:
- A statement is a list of statements terminated with a semicolon.
- A statement can be an assignment, which is a variable followed by an operator and another variable, or a simple statement.

Patterns:
- A pattern for a statement list is a list of statements.
- A pattern for a statement is an assignment or a simple statement.
- An assignment is a variable followed by an operator and another variable.
- A simple statement is a variable followed by a semicolon.
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
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:
        write ("one")
        break
    case 2:
        write ("two")
        break
    case 3:
        write ("three")
        break
    default
        write ("Other")
endswitch
end Trivial
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")
        break
endswitch
end Trivial