Lecture 2. Program Representation

Wei Le

2014.9
Frequently Used Program Representations in Program Analysis

- Abstract Syntax Tree (AST)
- Control Flow Graph (CFG), Inter-Procedural Control Flow Graph
- Call Graph
- Dependency Graphs
- Research Related to Program Representation
Building Abstract Syntax Trees

Lexical analysis (scanner, lexer, lexical analyzer): from a string to tokens

Syntactical analysis or parsing (parser): from tokens to trees, determine whether a program is part of the language
Abstract Syntax Tree (AST)

- Abstract syntax trees or syntax trees, represents the hierarchical syntactic structure of the source code.

AST does not contain all information from the source code. E.g., spacing, comments, brackets, parentheses.

Any ambiguity has been resolved: E.g., $a + b + c$ produces the same AST as $(a + b) + c$.
AST and Program Analysis

- Profile program structure information: how many loops are there in the code?
- Help instrumentation: e.g., insert printf after a call
- Design a specification language: Flex and Bison: two important open source tools to generate scanner and parser
- Demo: srcxml

(Example from Maprle)
A call graph for a program is a set of nodes and edges such that

1. There is one node for each procedure in the program.

2. There is one node for each call site, that is, a place in the program where a procedure is invoked.

3. If call site $c$ may call procedure $p$, then there is an edge from the node for $c$ to the node for $p$. 
int (*pf)(int);

int fun1(int x) {
    if (x < 10)
        return (*pf)(x+1);
    else
        return x;
}

int fun2(int y) {
    pf = &fun1;
    return (*pf)(y);
}

void main() {
    pf = &fun2;
    (*pf)(5);
}
Control Flow Graphs

A directed graph, where
- each node represents a statement
- each edge represents a control flow transfer

Variations of Control Flow Graphs
- We usually don’t include declarations (e.g., int x;)
- May want a unique entry and exit node: All nodes without a predecessor should be pointed to by entry; All nodes without a successor should point to exit
- May group statements into basic blocks: A sequence of instructions with no branches into or out of the block
Control Flow Graphs: Examples

Program

```
x = z-2;
y = 2*z;
if (c) {
    x = x+1;
y = y+1;
}
else {
    x = x-1;
y = y-1;
}
z = x+y;
```

Control Flow Graph

```
B1
  x = z-2;
y = 2*z;
if (c)
B2
  x = x+1;
y = y+1;
B3
  x = x-1;
y = y-1;
B4
  z = x+y;
```

```
while(1)

r = poll(ar, fd, -1);
ar[0].revents

if (r < 0)

fd = uid_accept(sock, &creds);

if (fd < 0)

/* successfully obtain sockets */
acpid_add_client(fd, buf);

/* process event */
clean_exit(EXIT_SUCCESS);
```
Inter-procedural Control Flow Graphs

A control flow graph (CFG) of a procedure is a graph $G = (N, E)$, where the nodes in $N$ represent statements of the procedure and the edges in $E$ represent the transfer of the control between two statements. Two distinguished nodes $\text{entry} \in N$ and $\text{exit} \in N$ represent the unique entry and exit of the procedure.

An interprocedural control flow graph (ICFG) of a program is a collection of control flow graphs $\{G_i\}$ such that $G_i$ represents a procedure in the program. Suppose $\text{call}(s)$ represents the procedure called from a callsite $s$. Then for each callsite $n$ in an ICFG, there exists an edge from $n$ to the entry of the procedure $\text{call}(n)$, and also there exists an edge from the exit of $\text{call}(n)$ to $n$.
CFG vs. AST

- Both structural
- CFG is simpler and specifies explicitly the control flow information
- AST is more faithful
Program Dependence Graphs (PDG) [1]

Node: statements
Edge: control and data dependence edges

- Control dependence between two statement nodes exists if one statement controls the execution of the other (e.g. through if- or while-statements).
- Data dependence between two statement nodes exists if a definition of a variable at one statement might reach the usage of the same variable at another statement.
Control Dependence

Definition 2. A node $V$ is \emph{post-dominated} by a node $W$ in $G$ if every directed path from $V$ to STOP (not including $V$) contains $W$.

Note that this definition of post-dominance does not include the initial node on the path. In particular, a node never post-dominates itself.

Definition 3. Let $G$ be a control flow graph. Let $X$ and $Y$ be nodes in $G$. $Y$ is \emph{control dependent} on $X$ iff

1. there exists a directed path $P$ from $X$ to $Y$ with any $Z$ in $P$ (excluding $X$ and $Y$) post-dominated by $Y$ and
2. $X$ is not post-dominated by $Y$.

If $Y$ is control dependent on $X$ then $X$ must have two exits. Following one of the exits from $X$ always results in $Y$ being executed, while taking the other exit may result in $Y$ not being executed. Condition 1 can be satisfied by a path consisting
Control Dependence Graph

(a)

(b)
System Dependence Graphs (SPDG)

```
program Main
    sum := 0;
    i := 1;
    while i < 11 do
        call A (sum, i)
    od
    end(sum, i)

procedure A (x, y)
    call Add(x, y);
    call Increment(y)
    return

procedure Add (a, b)
    a := a + b
    return

procedure Increment (z)
    call Add (z, 1)
    return
```
Applications of PDG

- Program Integration
- Debugging
- Automatic Parallelization
Research Related Topics: MVICFG-2014

(a) 4 Versions of Programs

(b) MVICFG for the 4 Versions

(c) Constructing an MVICFG
Figure 2: MVICFG: the Union of ICFGs
Karl J. Ottenstein and Linda M. Ottenstein.
The program dependence graph in a software development environment.