CISPA Center for IT-Security, Privacy and Accountability

CISPA Center for IT-Security, Privacy and Accountability

UNIVERSITÄT DES SAARLANDES

Interprocedural Data Flow Analysis

Static Program Analysis

Christian Hammer

18. Juni 2014

SPONSORED BY THE

Federal Ministry of Education and Research

max planck institut informatik

German Research Center for Artificial Intelligence

Max Planck Institute for Software Systems

Interprocedural Reaching Definitions

CISPA
Center for IT-Security, Privacy and Accountability

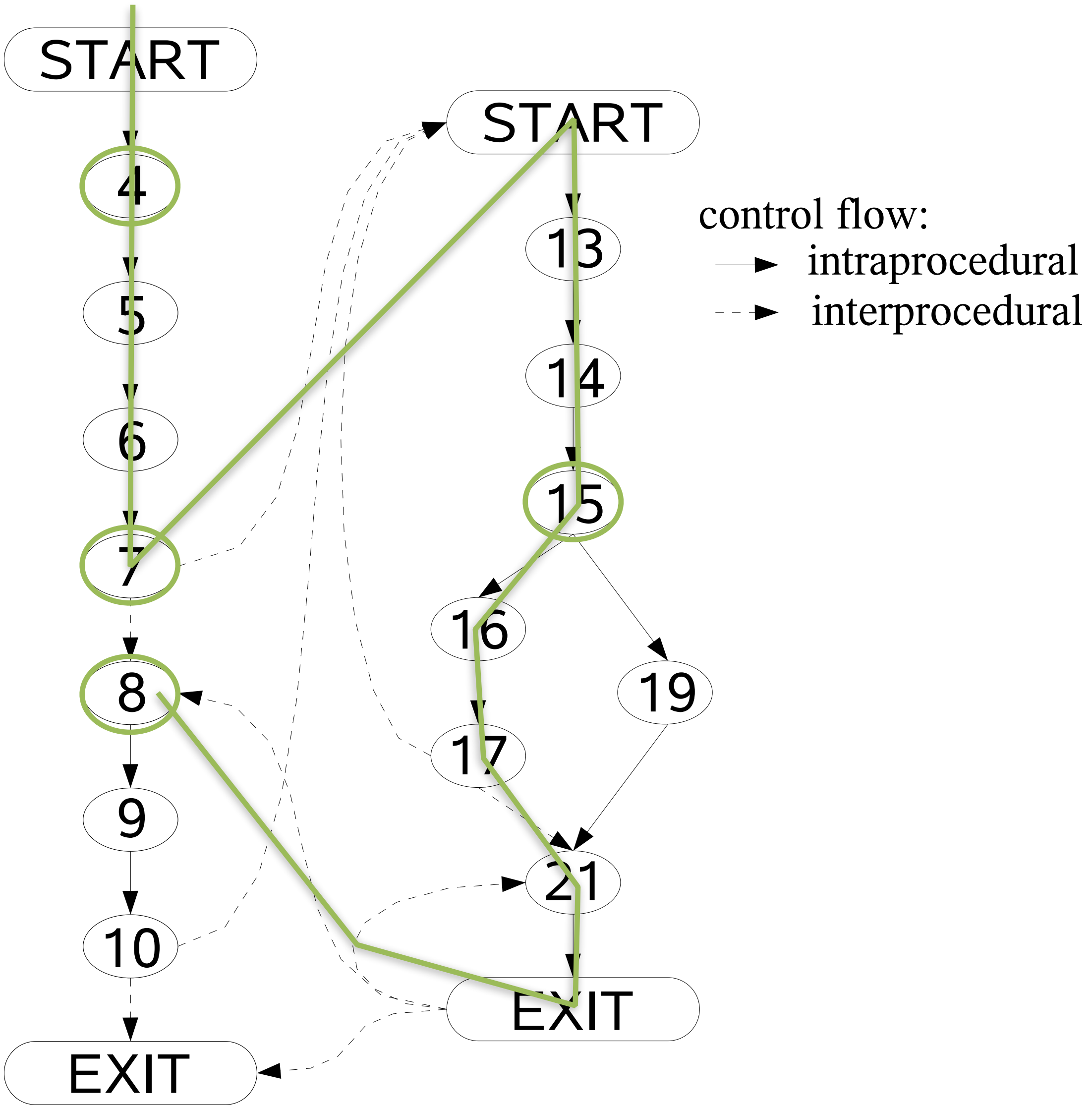
Global Variables

```
(1)  int a, b, c;

(3)  void q () {
(4)    int z=1;
(5)    a=2;
(6)    b=3;
(7)    p(4, z);
(8)    z=a;
(9)    c=5;
(10)   p(6, c);
(11) }

(12) void p(int x,int &y) {
(13)   static int d = 6;
(14)   a=c;
(15)   if(x) {
(16)     d=7;
(17)     p(8, x);
(18)   } else {
(19)     b=9;
(20)   }
(21)   y =0;
(22) }
```

call-by-value

call-by-reference

Interprocedural Control Flow Graph



Analyzing Interprocedural Programs

- $RD_{IMOP}(n) = \bigcup_{p=\langle n^s_0, \ldots , n\rangle} [p](\emptyset)$
- where p are inter procedurally realizable paths (impossible in general)
- interprocedural minimal-fixed-point (IMFP) solution is computed
- However, impossible to check for interprocedurally realizable paths

- Procedures can be inlined
  – replace calls by the called procedure
  – resulting program can be analyzed like an intraprocedural one
  – not possible in the presence of recursion
  – even without the size of the inlined programs may grow exponentially
  – not feasible in practice

Analyzing Interprocedural Programs (cont.)

- Compute effects of procedures
  - represented in a transfer function
  - maps flow information at a call site from the call to the return
  - call statements are ordinary statements with transfer functions
  - intraprocedural techniques can be applied

- Explicit encoding of calling context of a procedure
  - procedure is analyzed for each calling context separately
  - in the presence of recursion the set of calling contexts may be infinite
  - depending on the encoding of the calling context

Effect Calculation

C I S P A
Center for IT-Security, Privacy and Accountability

- functional approach [SP81]
- maps the data flow information at the entry of a procedure to the information that holds at the exit
- computed function can be used in the transfer functions at the call statements
- intraprocedural data flow analysis can then be used in a second pass
- first pass is a data flow analysis where the data flow information are functions and the transfer functions are function compositions
- For some data flow problems the resulting data flow information is infinite function compositions and therefore not computable
- For a large class of data flow problems these computed functions reduce to simple mappings where the composition can be computed instantly

Context Encoding

- call strings capture the “history” of calls that lead to a node n
- abstraction of the call stack
- lattice elements combine calling context and intraprocedural data flow facts
- transfer functions extended to handle the additional calling context
- length of the call strings can be limited to a certain length k
- call string longer than k are shortened such that the “oldest” elements are removed first
- overcomes limitations of recursion
- maybe imprecise

Call Strings

- calling context $c \in C$ encoded through data flow facts that hold at the entry to procedure $p \in P$
- data flow facts c′ at the exit of the procedure stored in mapping $C \times P \rightarrow C$
- At every call node n of a procedure p the data flow facts c are then bound to data flow facts c′ = bind(c) that hold at the entry node of p
- If the effect of p for c′ has already been computed, it can be reused from the mapping which contains the data flow facts c′′ holding at the exit of p
- After back-binding the effect to the call site, the effect $c''' = \text{bind}^{-1}(c'')$ holds at the exit of the call node n

Interprocedural Data Dependence

- Let $G = (N^*, E^*, n^s{}_0, n^e{}_0)$ be an ICFG. A node $m \in N^*$ is data dependent on node $n \in N^*$, if
  - there is an interprocedurally matched path p from n to m in the ICFG,
  - there is a variable v, with $v \in def(n)$ and $v \in ref(m)$, and
  - for all nodes $k \neq n$ of path p, $v \notin def(k)$ holds.

- At call sites the global variables are modeled as call-by-value-result parameters, which is correct without call-by-reference parameters and aliasing
- GMOD(p): the set of all variables that might be modified if procedure p is called.
- GREF(p): the set of all variables that might be referenced if procedure p is called.

Effect Calculation

- $\text{bind}^{-1}(S, p) = S - \text{locals}(p)$
- $\text{GMOD}(n) = \text{bind}^{-1}(\text{GMOD}(p))$
- $\text{GREF}(n) = \text{bind}^{-1}(\text{GREF}(p))$

- $\text{GMOD}(q) = \text{IMOD}(q) \cup \bigcup_{p \in \text{calls}(q)} \text{bind}^{-1}(\text{GMOD}(p), p)$
- $\text{GREF}(q) = \text{IREF}(q) \cup \bigcup_{p \in \text{calls}(q)} \text{bind}^{-1}(\text{GREF}(p), p)$

- $\text{def}(n) = \text{GMOD}(n)$
- $\text{ref}(n) = \text{GMOD}(n) \cup \text{GREF}(n)$

Example Interprocedural Data Dependences

CISPA
Center for IT-Security, Privacy and Accountability