Interprocedural Data Flow Analysis

Static Program Analysis

Christian Hammer

18. Juni 2014
Interprocedural Reaching Definitions

(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)  }
Interprocedural Reaching Definitions

(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)  }
Interprocedural Reaching Definitions

(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 a = 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 Reaching Definitions

(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) }

Global Variables
- definition of a (line 5) reaches 6–7 but not 8–11 (killed by 14 through call in 7), also reaches 13–14 but not 15–22
- definition of c in line 9 reaches 13–22 through call in line 10.

- More complex are the definitions of global b: the definition in line 6 cannot reach lines 8–10 or 21, as line 19 kills it — any call of p must execute line 19 to terminate the recursion. Also, the definition in line 19 reaches line 13–19, as it might reach the call in line 10 by procedure p returning from the call in line 7.

- The variable d is global and only visible inside procedure p. the definition in line 16 may reach lines 13–16 because of the call in line 17. Through procedure p returning from the call in line 7, both definitions (line 13 and 16) may reach lines 8–10 and therefore also line 13–16 and 18–22.

- Locals like z are (usually) only visible in procedures they are defined in. Call-by-value parameters are like locals, with a definition at the procedure entry: x is defined in line 12.

- Call-by-reference introduces a simple form of aliasing and makes otherwise invisible variables available in called procedures.
6.2 Interprocedural Realizable Paths

In the intraprocedural case all paths in the CFG were assumed to be executable and therefore realizable. In the interprocedural case this is more complicated: The individual procedures of a program are represented in control flow graphs \( G_p = (N_p, E_p, n_{s_p}, n_{e_p}) \) for each procedure \( p \). An interprocedural control flow graph (ICFG) is a directed graph \( G = (N?, E?, n_{s_0}, n_{e_0}) \), where \( N? = S_p N_p \) and \( E? = E C [S_p E_p] \). One procedure \( q \) is the program’s main procedure, its START and EXIT nodes are the main START and EXIT nodes: \( n_{s_0} = n_{s_q} \) and \( n_{e_0} = n_{e_q} \).

The calls are represented by call and return edges in \( E_C \): A call edge \( e \in E_C \) is going from a call node \( n_2 \in N_p \) to the \( \text{START} \) node \( n_{s_q} \) of the called procedure \( q \). A return edge \( e \in E_C \) is going from the \( \text{EXIT} \) node \( n_{e_q} \) of the called procedure \( q \) back to the immediate successor of the call node \( n_2 \in N_p \).

Example 6.2: Figure 6.2 shows the ICFG for the reaching definition example. Note that there are control flow edges between call nodes and their immediate successors.

If any path through the ICFG is assumed to be a realizable path, data flow analysis will become imprecise, as clearly unrealizable paths can be traversed: Consider the definition of global \( c \) in line/node 9, which reaches the called procedure via the call edge at line/node 10. All paths through \( p \) are free of definitions for \( c \) and the definition gets propagated along the return edges: via

1. There are two common variants: First, the immediate successor of a call node is an explicitly defined return node. Second, the return edge is going from the EXIT node to the call node itself.
- **Control Flow Graph** $G_p = (N_p, E_p, n^s_p, n^e_p)$ for each procedure $p$. An interprocedural control flow graph (ICFG) is a directed graph $G = (N^*, E^*, n^s_0, n^e_0)$, where $N^* = \bigcup_p N_p$ and $E^* = E^C \cup \bigcup_p E_p$

- **call and return edges** in $E^C$: A call edge $e \in E^C$ is going from a call node $n \in N_p$ to the START node $n^s_q$ of the called procedure $q$. A return edge $e \in E^C$ is going from the EXIT node $n^e_q$ of the called procedure $q$ back to the immediate successor of the call node $n \in N_p$

- **unrealizable paths** possible if leaving a function on a different node than the call’s successor
6.2 Interprocedural Realizable Paths

In the intraprocedural case all paths in the CFG were assumed to be executable and therefore realizable. In the interprocedural case this is more complicated:

The individual procedures of a program are represented in control flow graphs \( G_p = (N_p, E_p, n_{s_p}, n_{e_p}) \) for each procedure \( p \). An interprocedural control flow graph (ICFG) is a directed graph \( G = (N, E, n_s, n_e) \), where

\[
N = S_p N_p
\]

and

\[
E = E_C \cup E_s
\]

One procedure \( q \) is the program's main procedure, its START and EXIT nodes are the main START and EXIT nodes:

\[
n_s = n_{s_q} \quad \text{and} \quad n_e = n_{e_q}
\]

The calls are represented by call and return edges in \( E_C \): A call edge \( e \in E_C \) is going from a call node \( n \) to the START node \( n_{s_q} \) of the called procedure \( q \). A return edge \( e \in E_C \) is going from the EXIT node \( n_{e_q} \) of the called procedure \( q \) back to the immediate successor of the call node \( n \).

Example 6.2: Figure 6.2 shows the ICFG for the reaching definition example. Note that there are control flow edges between call nodes and their immediate successors.

If any path through the ICFG is assumed to be a realizable path, data flow analysis will become imprecise, as clearly unrealizable paths can be traversed: Consider the definition of global \( c \) in line/node 9, which reaches the called procedure via the call edge at line/node 10. All paths through \( p \) are free of definitions for \( c \) and the definition gets propagated along the return edges: via

1. There are two common variants: First, the immediate successor of a call node is an explicitly defined return node. Second, the return edge is going from the EXIT node to the call node itself.
Inter-procedurally Realizable Path

Edges from $E_p$ are marked with the empty word $\varepsilon$ and edges from $E_C^\ell$ are marked according to their source and target nodes.

An interprocedurally realizable path $I$ is an interprocedurally right- or left-balanced path.
Interprocedural Reachability, Witness

- A node $n$ is interprocedurally reachable from node $m$, iff an interprocedurally realizable path from $m$ to $n$ in the ICFG exists, written as $m \rightarrow^* R n$.
- A sequence $\langle n_1, \ldots, n_k \rangle$ of nodes is called an interprocedurally (realizable) witness, iff $n_k$ is interprocedurally reachable from $n_1$ via an interprocedurally realizable path $p = \langle m_1, \ldots, m_l \rangle$ with:
  - $m_1 = n_1$, $m_l = n_k$, and
  - $\forall 1 \leq i < k \exists x, y: x < y \land m_x = n_i \land m_y = n_i + 1$. 
6.2 Interprocedural Realizable Paths

In the intraprocedural case all paths in the CFG were assumed to be executable and therefore realizable. In the interprocedural case this is more complicated:

The individual procedures of a program are represented in control flow graphs $G_p = (N_p, E_p, n_s, n_e)$ for each procedure $p$. An interprocedural control flow graph (ICFG) is a directed graph $G = (N?, E?, n_s?, n_e?)$, where $N? = S_p N_p$ and $E? = E_C [S_p E_p]$. One procedure $q$ is the program's main procedure, its START and EXIT nodes are the main START and EXIT nodes: $n_s? = n_s q$ and $n_e? = n_e q$.

The calls are represented by call and return edges in $E_C$: A call edge $e_2 \in E_C$ is going from a call node $n_2 \in N_p$ to the START node $n_s q$ of the called procedure $q$. A return edge $e_2 \in E_C$ is going from the EXIT node $n_e q$ of the called procedure $q$ back to the immediate successor of the call node $n_2 \in N_p$.

Example 6.2: Figure 6.2 shows the ICFG for the reaching definition example. Note that there are control flow edges between call nodes and their immediate successors.

If any path through the ICFG is assumed to be a realizable path, data flow analysis will become imprecise, as clearly unrealizable paths can be traversed: consider the definition of global $c$ in line/node 9, which reaches the called procedure via the call edge at line/node 10. All paths through $p$ are free of definitions for $c$ and the definition gets propagated along the return edges: via.

There are two common variants: First, the immediate successor of a call node is an explicitly defined return node. Second, the return edge is going from the EXIT node to the call node itself.
6.2 Interprocedural Realizable Paths

In the intraprocedural case all paths in the CFG were assumed to be executable and therefore realizable. In the interprocedural case this is more complicated: The individual procedures of a program are represented in control flow graphs \( G_p = (N_p, E_p, n_{sp}, n_{ep}) \) for each procedure \( p \). An interprocedural control flow graph (ICFG) is a directed graph \( G = (N?, E?, n_s, n_e) \), where \( N? = S_p N_p \) and \( E? = E_C[S_p E_p] \). One procedure \( q \) is the program's main procedure, its \( n_s = n_{sp}q \) and \( n_e = n_{ep}q \).

The calls are represented by call and return edges in \( E_C \): A call edge \( e \in E_C \) is going from a call node \( n \in N_p \) to the \( \text{START} \) node \( n_{sp}q \) of the called procedure \( q \). A return edge \( e \in E_C \) is going from the \( \text{EXIT} \) node \( n_{ep}q \) of the called procedure \( q \) back to the immediate successor of the call node \( n \).

**Example 6.2:** Figure 6.2 shows the ICFG for the reaching definition example. Note that there are control flow edges between call nodes and their immediate successors.

If any path through the ICFG is assumed to be a realizable path, data flow analysis will become imprecise, as clearly unrealizable paths can be traversed: Consider the definition of global \( c \) in line/node 9, which reaches the called procedure via the call edge at line/node 10. All paths through \( p \) are free of definitions for \( c \) and the definition gets propagated along the return edges: via 1

There are two common variants: First, the immediate successor of a call node is an explicitly defined return node. Second, the return edge is going from the \( \text{EXIT} \) node to the call node itself.