Precise Interprocedural Dataflow Analysis via Graph Reachability

Thomas Reps      Susan Horwitz      Mooly Sagiv
POPL 1995

Presenter: Ben Greenman
IFDS

Thomas Reps  Susan Horwitz  Mooly Sagiv

POPL 1995

Presenter: Ben Greenman
The IFDS "framework"

- a model for dataflow problems
- a *uniform solution* to these problems
- a polynomial-time *algorithm*
Reaching Definitions

What statements are affected by a given definition?

there is a path from \( x := e' \) to \( e \)  
x is not re-defined along the path

\( x := e' \) reaches \( e \)
Available Expressions

What expressions can a statement re-use?

- there is a path from \( x := \ldots \ e' \ldots ; \) to \( e \)
- no variable in \( e' \) is re-defined along the path

\( e' \) is available at \( e \)
Live Variable Analysis

What variables are referenced at/after a statement?

there is a path from \(e\) to \(e'\)  \(v\) is referenced at \(e'\)

\(v\) is live at \(e\)
Possibly-Uninitialized Variables

Which variables may be null at a given statement?

\[ x := e'; \]

does not appear on any path to \( e \)

\[ x \text{ may be } \text{null at } e \]

\[ x := e'; \]
is the most recent binding

\[ \exists y \in e'. \]
y may be null at \( e' \)

\[ x \text{ may be } \text{null at } e \]
Program Slicing

"The algorithm described in this paper yields an improved interprocedural-slicing algorithm ... 6x as fast as the Horwitz-Reps-Binkley algorithm."

Speeding up Slicing
Reps, Horwitz, Sagiv, Rosay; FSE '94
Interprocedural Slicing Using Dependence Graphs  
Susan Horwitz, Thomas Reps, David Binkley

The Program Summary Graph and Flow-Sensitive Interprocedural Data-Flow Analysis  
David Callahan

Interprocedural Side-Effect Analysis in Linear Time  
Keith D. Cooper, Ken Kennedy
Possibly-Uninitialized Variables

Which variables may be null at a given statement?

\[ x := e'; \]

does not appear on any path to \( e \).

\[ x \text{ may be null at } e \]

\[ x := e'; \]

is the most recent binding.

\[ \exists y \in e'. \]

\[ y \text{ may be null at } e' \]

\[ x \text{ may be null at } e \]
int g;

void P(int a) {
    if (a > 0) {
        read(g);
        a := a - g;
        P(a);
    }
}

enter P

if (a > 0)

read(g);

a := a - g;

P(a);

exit P
int g;

void P(int a) {
    if (a > 0) {
        read(g);
        a := a - g;
        P(a);
    }
}

enter P

if (a > 0)

read(g);

a := a - g;

P(a);

exit P

{ g }

{ g }

{ g }
int g;

void P(int a) {
    if (a > 0) {
        read(g);
        a := a - g;
        P(a);
    }
}
Kildall, POPL 1973

"Meet over all paths"

\[ \prod_{f_1 \ldots f_n \in \text{AllPaths}} \{ g \} \]

\[ \{ a \} \quad \{ g \} \quad \{ a, g \} \]

Infinite Set!

\[ \{ a \} \quad \{ g \} \quad \{ a, g \} \]

\[ \lambda S. S - \{ g \} \]

\[ \lambda S. ? \]

\[ \lambda S. S \]

\[ \{ g \} \]

\[ \{ g \} \]

\[ \{ g \} \]

\[ \{ g \} \]
Meet over all valid paths

- Calls & Returns must match
Meet over all valid paths

- Calls & Returns must match
- Enforced by call & ret nodes
Meet over all valid paths

- Calls & Returns must match
- Enforced by call & ret nodes
- Track local variables with a call-to-return edge
if (a > 0)
    a := a - g;
call P(a);
read(g);
ret P;
int g;
void main(void) {
  int x;
  read(x);
  P(x);
}

The "supergraph"

The given code snippet defines a function `P` with an `if` statement and a call to another procedure. It also includes a `main` function that reads an integer `x` and calls `P(x)`.

The diagram illustrates the control flow of the program, with nodes representing function calls and statements, and edges indicating the flow of control. The diagram includes annotations showing the context of function invocations and the effects of the program's execution.
The "supergraph"

```c
int g;

void main(void) {
    int x;
    read(x);
    P(x);
}
```

Each procedure can have a different $D$ and lattice
An IFDS problem instance

- $G$ = a supergraph
- $D$ = a finite set (determines a lattice)
- $F$ = a set of distributive functions over the lattice
- $M$ = a map from edges in $G$ to functions in $F$
- $\cap$ = meet operator on the lattice
A few "IFDS" problems

- Reaching definitions
- Available Expressions
- Live Variable Analysis
- Possibly-Uninitialized Variables
- Type Analysis
\[
\prod_{i=1}^{n} f_i \quad \text{if } (a > 0) \\
\text{read}(g) ; \quad \text{call } P(a) ; \\
a := a - g ; \\
\text{ret } P ; \quad \text{exit } P
\]

\[
\lambda S. S - \{g\} \\
\lambda S. S \quad \lambda S. S \quad \lambda S. ? \\
\lambda S. S \quad \lambda S. S \quad \lambda S. S \\
\lambda S. S \quad \lambda S. S \quad \lambda S. S \\
\lambda S. S \quad \lambda S. S \quad \lambda S. S \\
\lambda S. S \quad \lambda S. S \quad \lambda S. S \\
\lambda S. S \quad \lambda S. S \quad \lambda S. S \\
\lambda S. S \quad \lambda S. S \quad \lambda S. S
\]

... since each \( f_i \) is distributive

\[
f_i (D) = \{ f_i (D_1), \ldots, f_i (D_k) \}
\]

... therefore

\[
f_n (\ldots (f_1(D)) \ldots) = \{ f_n (\ldots (f_1(D_1)) \ldots ) , \ldots , f_n (\ldots (f_1(D_k)) \ldots ) \}
\]

In general, apply "path function" to a subset of \( D \)
\[ \prod \{ \text{\textit{fn} ( ... ( \text{\textit{f1} ( D ) ) ) ... )} \} \]

\text{\textit{f1} ... \text{\textit{fn}} \in \text{All Valid Paths}}

\ldots \text{since each \textit{fi} is distributive}

\text{\textit{fi} (D) = \{ \text{\textit{fi} ( D_1 ), \ldots , \text{\textit{fi} ( D_k )} \} }

\ldots \text{therefore}

\text{\textit{fn} ( ... ( \text{\textit{f1} (D) )}) ...) = \{ \text{\textit{fn} ( ... ( \text{\textit{f1} (D_1) )}) ...) }

\ldots_k \}
\[ \prod_{g} \left( \ldots \left( f_1 \left( D \right) \right) \ldots \right) \]

\[ f_1 \ldots f_n \in \text{All Valid Paths} \]

... since each \( f_i \) is distributive

\[ f_i \left( D \right) = \{ f_i \left( D_1 \right), \ldots, f_i \left( D_k \right) \} \]

... therefore

\[ f_n \left( \ldots \left( f_1 \left( D \right) \right) \ldots \right) = \{ f_n \left( \ldots \left( f_1 \left( D_1 \right) \right) \ldots \right) \]

\[ \ldots \} \]
\[ \prod_{f_1 \ldots f_n \in \text{All Valid Paths}} f_n (\ldots (f_1(D))\ldots) \]

... since each \( f_i \) is distributive

\[ f_i(D) = \{ f_i(D_1), \ldots, f_i(D_k) \} \]

... therefore

\[ f_n (\ldots (f_1(D))\ldots) = \{ f_n (\ldots (f_1(D_1))\ldots) \ldots \} \]

\[ a := a - g; \]

\[ \lambda S. \quad \text{if } a \in S \]

\[ \quad \text{or } g \in S \]

\[ \quad \text{then } S \cup \{a\} \]

\[ \quad \text{else } S - \{a\} \]
\[
\prod_{f_i \text{ distributive}} \left( f_1 (D) \right) = \left\{ f_1 (D_1), \ldots, f_i (D_k) \right\}
\]

... therefore

\[
f_n (\ldots (f_1 (D)) \ldots) = \{ f_n (\ldots (f_1 (D_1)) \ldots) \}
\]

For any \( f \),

\[
0 \rightarrow 0 \text{ if } f(0) = y
\]

\[
x \rightarrow y \text{ if } f(x) = y \text{ and } f(0) \neq y
\]
int g;

void main(void) {
    int x;
    read(x);
    P(x);
}

void P(int a) {
    if (a > 0) {
        read(g);
        a := a - g;
        P(a);
    }
}
"Tabulation" Algorithm

1. keep a worklist of **Path Edges**
   - (suffixes of valid paths)

2. build set of **Summary Edges**
   - (side effects of a procedure call)

3. result = meet over valid paths
Init
(lines 1-4)

Path Edge

Summary Edge:
Case \( n \not\in \text{Call}, \ n \not\in \text{Exit} \) (lines 31-33)

**Path Edge**

\( \text{main 0} \rightarrow \text{main 0} \)

\( \text{main 0} \rightarrow n1 \{x,g\} \)

**Summary Edge:**
Case $n \not\in \text{Call}, n \not\in \text{Exit}$
(lines 31-33)

Path Edge:
- main 0 $\rightarrow$ main 0
- main 0 $\rightarrow$ n1 \{x,g\}
- main 0 $\rightarrow$ n2 \{g\}

Summary Edge:
Case $n \in \text{Call}$
(lines 13-20)

Path Edge:

- $\text{main 0} \rightarrow \text{main 0}$
- $\text{main 0} \rightarrow n1 \{x,g\}$
- $\text{main 0} \rightarrow n2 \{g\}$
- $\text{sp} \{g\} \rightarrow \text{sp} \{g\}$
- $\text{main 0} \rightarrow n3 \{g\}$

Summary Edge:
Case $n \not\in \text{Call}, n \not\in \text{Exit}$
(lines 31-33)

Path Edge

<table>
<thead>
<tr>
<th>main 0</th>
<th>main 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>main 0</td>
<td>n1 {x,g}</td>
</tr>
<tr>
<td>main 0</td>
<td>n2 {g}</td>
</tr>
<tr>
<td>sp {g}</td>
<td>sp {g}</td>
</tr>
<tr>
<td>main 0</td>
<td>n3 {g}</td>
</tr>
<tr>
<td>main 0</td>
<td>exit {g}</td>
</tr>
</tbody>
</table>

Summary Edge:
Case n \(\not\in\) Call, n \(\not\in\) Exit
(lines 31-33)

Path Edge:

- main 0 \(\rightarrow\) main 0
- main 0 \(\rightarrow\) n1 \{x, g\}
- main 0 \(\rightarrow\) n2 \{g\}
- sp \{g\} \(\rightarrow\) sp \{g\}
- main 0 \(\rightarrow\) n3 \{g\}
- main 0 \(\rightarrow\) exit \{g\}

Summary Edge:
Case \( n \not\in \text{Call}, n \not\in \text{Exit} \) (lines 31-33)
Case \( n \notin \text{Call}, \ n \notin \text{Exit} \) (lines 31-33)
Case \( n \in \text{Exit} \) (lines 21-25)

**Path Edge**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>main 0</td>
<td>main 0</td>
</tr>
<tr>
<td>main 0</td>
<td>n1 {x,g}</td>
</tr>
<tr>
<td>main 0</td>
<td>n2 {g}</td>
</tr>
<tr>
<td>sp {g}</td>
<td>sp {g}</td>
</tr>
<tr>
<td>main 0</td>
<td>n3 {g}</td>
</tr>
<tr>
<td>main 0</td>
<td>exit {g}</td>
</tr>
<tr>
<td>sp {g}</td>
<td>n4 {g}</td>
</tr>
<tr>
<td>sp {g}</td>
<td>exit {g}</td>
</tr>
<tr>
<td>sp {g}</td>
<td>n5 {g}</td>
</tr>
</tbody>
</table>

**Summary Edge:**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>n2 {g}</td>
<td>n3 {g}</td>
</tr>
</tbody>
</table>
Case $n \in \text{Exit}$
(lines 25-32)

Path Edge:

- main 0 → main 0
- main 0 → n1 \(\{x,g\}\)
- main 0 → n2 \(\{g\}\)
- sp \(\{g\}\) → sp \(\{g\}\)
- main 0 → n3 \(\{g\}\)
- main 0 → exit \(\{g\}\)
- sp \(\{g\}\) → n4 \(\{g\}\)
- sp \(\{g\}\) → exit \(\{g\}\)
- sp \(\{g\}\) → n5 \(\{g\}\)
- main 0 → n3 \(\{g\}\)

Summary Edge:

- n2 \(\{g\}\) → n3 \(\{g\}\)
Case $n \in \text{Exit}$
(lines 25-32)

Path Edge:
- main 0 → main 0
- main 0 → n1 \{x,g\}
- main 0 → n2 \{g\}
- sp \{g\} → sp \{g\}
- main 0 → n3 \{g\}
- main 0 → exit \{g\}
- sp \{g\} → n4 \{g\}
- sp \{g\} → exit \{g\}
- sp \{g\} → n5 \{g\}
- main 0 → n3 \{g\}

Summary Edge:
- n2 \{g\} → n3 \{g\}

And so on ...
Algorithm II

- "4" ways to find Path Edges

1. call edge

2. return edge / Summary Edge

3. normal edge
Running Time

- **E** supergraph edges to explore
- **D** sources to explore from
- **D^2** exploded edges for each edge

<table>
<thead>
<tr>
<th>Class of F</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributive</td>
<td>O(ED^3)</td>
</tr>
<tr>
<td><em>h</em>-sparse</td>
<td>O(Call D^3 + hED^2)</td>
</tr>
<tr>
<td>Locally Separable</td>
<td>O(ED)</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Program</th>
<th># lines</th>
<th># proc.</th>
<th># calls</th>
<th># nodes</th>
<th># edges</th>
<th>D</th>
<th># n++</th>
<th># e++</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct-beauty</td>
<td>897</td>
<td>36</td>
<td>214</td>
<td>2188</td>
<td>2860</td>
<td>90</td>
<td>184k</td>
<td>221k</td>
</tr>
<tr>
<td>C-parser</td>
<td>1224</td>
<td>48</td>
<td>78</td>
<td>1637</td>
<td>1992</td>
<td>70</td>
<td>104k</td>
<td>112k</td>
</tr>
<tr>
<td>ratfor</td>
<td>1345</td>
<td>52</td>
<td>266</td>
<td>2239</td>
<td>2991</td>
<td>87</td>
<td>180k</td>
<td>218k</td>
</tr>
<tr>
<td>twig</td>
<td>2388</td>
<td>81</td>
<td>221</td>
<td>3692</td>
<td>4439</td>
<td>142</td>
<td>492k</td>
<td>561k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program</th>
<th>naive time (s)</th>
<th>naive # null</th>
<th>ifds time (s)</th>
<th>ifds # null</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct-beauty</td>
<td>1.58</td>
<td>583</td>
<td>4.83 (+ 3.25)</td>
<td>543 (- 40)</td>
</tr>
<tr>
<td>C-parser</td>
<td>0.54</td>
<td>127</td>
<td>0.7 (+ 0.16)</td>
<td>11 (- 116)</td>
</tr>
<tr>
<td>ratfor</td>
<td>1.46</td>
<td>998</td>
<td>3.15 (+ 1.69)</td>
<td>894 (- 104)</td>
</tr>
<tr>
<td>twig</td>
<td>5.04</td>
<td>775</td>
<td>5.45 (+ 0.41)</td>
<td>767 (- 8)</td>
</tr>
</tbody>
</table>
Precise Interprocedural Dataflow Analysis via Graph Reachability

- Evaluation
- Side effects, live variables, type analysis, ...
- Exploded Supergraph
- Tabulation Algorithm
- F → bipartite graph
Discussion

• What static analysis problems are / are not IFDS?

• The uninitialized variables problem is **cubic** in the # global variables, even if these are rarely used. Can we avoid this overhead?

• Could we allow a (restricted) GOTO?

• Can we add more context-sensitivity?

(Naeem, Lhoták, Rodriguez; CC'10)
Influence

- WALA

- SOOT (Bodden; SOAP '12)

- FLIX (Madsen, Yee, Lhoták; PLDI 2016)

- FlowDroid (Arzt, Rasthofer, Fritz, Bodden, Bartel, Klein, Le Traon, Octeau, McDaniel; PLDI 2014)