Interprocedural Optimisation

Seminar Static Program Analysis

Barbara Dörr

Sources:
Übersetzerbau - Analyse und Transformation (H. Seidl, R. Wilhelm, S. Hack)
Principles of Program Analysis (F. Nielson, H.R. Nielson, C. Hankin)

12. März 2010
Forms of Program Optimisation

Program Optimisation

- **Intraprocedural Optimisation**: optimise each function separately
- **Interprocedural Optimisation**: explicitly model function calls

⇒ **Interprocedural Optimisation**: more demanding, but also more precise information

- Inlining
- Remove Last Call
Interprocedural vs. Intraprocedural

disadvantage of intraprocedural optimisation: context-insensitive optimisation:
cannot distinguish between different calls (information is combined from all call sites)
→ imprecise information

interprocedural optimisation: context-sensitive optimisation:
different calls reached with different contexts $\delta_1$ and $\delta_2$
→ information obtained clearly related to $\delta_1$ and $\delta_2$
⇒ more precise, but more costly
Introduction

Simple Interprocedural Optimisations

Operational Semantic

Functional Approach

Related Approaches

Summary
Program Representation

intraprocedural

→ program represented by a control flow graph:

```
y <- 1;
while (x>1){
  y <- x*y;
  x <- x-1;
}
```
**Program Representation**

interprocedural

→ program represented by a set of control flow graphs;

\[
f() \]

\[
\text{main}() \{ \\
    b \leftarrow 3; \\
    f(); \\
    M[17] \leftarrow \text{ret}; \\
\}
\]

\[
f()\{ \\
    A \leftarrow b; \\
    \text{if } (A \leq 1) \text{ ret } \leftarrow 1; \\
    \text{else } \{ \\
        b \leftarrow A-1; \\
        f(); \\
        \text{ret } \leftarrow A \times \text{ret}; \\
    \}
\}
\]
Edge Annotations

\((x \ldots \text{variable}, \; e \ldots \text{arithmetic expression})\)

**edge effects - intraprocedural:**

- **Test:**
  - NonZero\((e)\)
  - Zero\((e)\)

- **Assignment:**
  - \(x \leftarrow e\)

- **Load:**
  - \(x \leftarrow M[e]\)

- **Store:**
  - \(M[e_1] \leftarrow e_2\)

- **Empty Statement:**
  - ;

**additional edge effect - interprocedural:**

- **Function Call:**
  - \(f()\)
Introduction

Simple Interprocedural Optimisations

Operational Semantic

Functional Approach

Related Approaches

Summary
Inlining

Inlining:
copy function body to calling point

Problems:

▶ function has to be statically known
▶ local variables of calling function must not be modified
  → rename local variables
▶ recursive functions
  → identified from call graph

→

▶ inlining only for leave functions (without calls)
▶ inlining only for non-recursive functions
Inlining

Call Graph

**Call Graph:**

- **nodes ~ functions**
- **edges ~ between function** $f_1$ and function $f_2$, if $f_1$ calls $f_2$

```c
main() {
    b <- 3;
    f();
    M[17] <- ret;
}

f(){
    A <- b;
    if (A <=1) ret <- 1;
    else {
        b <- A-1;
        f();
        ret <- A*ret;
    }
}
```
Inlining

Call Graph

```plaintext
abs()
  b_1 <- b;
  b_2 <- -b;
  max();
}

max()
  if (b_1 < b_2) ret <- b_2;
  else ret <- b_1;
```
Inlining

transformation PI:

$u \xrightarrow{f()} v$

copy of $f$

$A_f = 0; A \in \text{Loc}$
Inlining example

```
abs()
{  
b_1 <- b;
b_2 <- -b;
}

max()
{  
  if (b_1 < b_2) ret <- b_2;
  else ret <- b_1;
}
```

```
abs()
{  
b_1 <- b;
b_2 <- -b;
  if (b_1 < b_2) ret <- b_2;
  else ret <- b_1;
}
```
Remove Last Calls

→ no own stack frame needed; only replace local variables (unconditional jump to function body)
! only possible if local variables of calling function are not accessible any more

**transformation LC:**

\[ f() : \]

\[ A = 0; \ (A \in Loc) \]

\[ f() : \]
Remove Last Calls

eexample

\[
f()\{
  \text{if} (b_2 \leq 1) \text{ret} \leftarrow b_1; \\
  \text{else} \{
    b_1 \leftarrow b_1 \times b_2; \\
    b_2 \leftarrow b_2 - 1; \\
    f();
  \}
}\]

\[
f()\{
  \_f: \text{if} (b_2 \leq 1) \text{ret} \leftarrow b_1; \\
  \text{else} \{
    b_1 \leftarrow b_1 \times b_2; \\
    b_2 \leftarrow b_2 - 1; \\
    \text{goto } \_f;
  \}
}\]
Introduction

Simple Interprocedural Optimisations

Operational Semantic

Functional Approach

Related Approaches

Summary
Operational Semantic

intraprocedural

- computations are described by **paths** through the control flow graph
- computations transform the current program state
- **program state**: \( s = (\rho, \mu) \) with
  \[ \rho : \text{Vars} \rightarrow \text{int} \] ... value of variables
  \[ \mu : \mathbb{N} \rightarrow \text{int} \] ... content of memory
- edge \( k = (u, \text{lab}, v) \)
  ... entry node \( u \), exit node \( v \), edge annotation \( \text{label} \)
- **edge effect**: transformation \([k]\) on program states defined by the edge \( k \)
  \[ [k] = [\text{lab}] \]
Operational Semantic

Edge Effects - intraprocedural

\[
\begin{align*}
[; ] (\rho, \mu) &= (\rho, \mu) \\
[\text{NonZero}(e)] (\rho, \mu) &= (\rho, \mu), \quad \text{if} \ [e]_{\rho} \neq 0 \\
[\text{Zero}(e)] (\rho, \mu) &= (\rho, \mu), \quad \text{if} \ [e]_{\rho} = 0 \\
[x \leftarrow e] (\rho, \mu) &= \left(\rho \oplus \{x \mapsto [e]_{\rho}\}, \mu\right) \\
x \leftarrow M[e] \quad (\rho, \mu) &= \left(\rho \oplus \{x \mapsto \mu([e]_{\rho})\}, \mu\right) \\
M[e_1] \leftarrow e_2 \quad (\rho, \mu) &= \left(\rho, \mu \oplus \{[e_1]_{\rho} \mapsto [e_2]_{\rho}\}\right)
\end{align*}
\]
Stack Representation

Call Stack:

```
main() {
    b <- 3;
    f();
    M[17] <- ret;
}
```

```
f(){
    A <- b;
    if (A <= 1) ret <- 1;
    else {
        b <- A-1;
        f();
        ret <- A*ret;
    }
}
```
Stack Representation

**call stack:**

- describes called and not yet finished functions
- basis of operational semantic

\[
\text{config} = \text{stack} \times \text{globals} \times \text{store} \\
\text{globals} = \text{Glob} \rightarrow \mathbb{Z} \\
\text{store} = \mathbb{N} \rightarrow \mathbb{Z} \\
\text{stack} = \text{frame} \cdot \text{frame}^* \\
\text{frame} = \text{point} \times \text{locals} \\
\text{locals} = \text{Loc} \rightarrow \mathbb{Z}
\]

! function body is a scope with own local variables
Modeling of Function Call

- call \( k = (u, f(), v) \): \( !\rho_f = \{ x \mapsto 0 | x \in \text{Loc} \} \)

\[
\begin{align*}
(\sigma \cdot \left[ (u, \rho_{\text{Loc}}) \right], \rho_{\text{Glob}}, \mu) & \vdash (\sigma \cdot \left[ (v, \rho_{\text{Loc}}) \cdot (u_f, \rho_f) \right], \rho_{\text{Glob}}, \mu) \\
\end{align*}
\]

- effect of function itself
- return from call:

\[
\begin{align*}
(\sigma \cdot \left[ (v, \rho_{\text{Loc}}) \cdot (r_f, -) \right], \rho_{\text{Glob}}, \mu) & \vdash (\sigma \cdot \left[ (v, \rho_{\text{Loc}}) \right], \rho_{\text{Glob}}, \mu) \\
\end{align*}
\]

\(\sigma\) \ldots stack
\(\rho_{\text{Glob}}\) \ldots global variables
\(\mu\) \ldots store
\((u, \rho_{\text{Loc}})\) \ldots frame \((\text{point} \times \text{locals})\)
Path Effects

\[ \pi : ((u, \rho_{Loc}), \rho_{Glob}, \mu) \rightsquigarrow ((v, \rho'_{Loc}), \rho'_{Glob}, \mu') \]

Path \( \pi \) defines a partial function \( \llbracket \pi \rrbracket \), that transforms \( ((u, \rho_{Loc}), \rho_{Glob}, \mu) \) into \( ((v, \rho'_{Loc}), \rho'_{Glob}, \mu') \)

\( \Rightarrow \) compute transformation inductive over the structure of the path:

\[ \llbracket \pi k \rrbracket = \llbracket k \rrbracket \circ \llbracket \pi \rrbracket \]

for a normal edge \( k \) (composition of edge effects)
Paths Effects

- **same-level**: all entered functions are also left again

\[ \pi = \pi_1 \langle f \rangle \pi_2 \langle \backslash f \rangle \]

\[ \begin{array}{cccc}
05 & A & \rightarrow & 1 \\
07 & A & \rightarrow & 2 \\
08 & A & \rightarrow & 3 \\
02 & & & \\
05 & A & \rightarrow & 1 \\
08 & A & \rightarrow & 2 \\
08 & A & \rightarrow & 3 \\
02 & & &
\end{array} \]

→ height of the stack stays the same

\[ [\pi_1 \langle f \rangle \pi_2 \langle \backslash f \rangle] = H([\pi_2]) \circ [\pi_1] \]

with

\[ H(g)(\rho_{Loc}, \rho_{Glob}, \mu) = \text{let} (\rho'_{Loc}, \rho'_{Glob}, \mu') = g(0, \rho_{Glob}, \mu) \]

\[ \text{in} (\rho_{Loc}, \rho'_{Glob}, \mu') \]
Path Effects

- computation that reaches a program point:
  \( \pi \langle f \rangle \pi' \) with \( \pi, \pi' \) is same-level

\[
\begin{align*}
\llbracket \pi \langle f \rangle \pi' \rrbracket \left( \rho_{\text{Loc}}, \rho_{\text{Glob}}, \mu \right) &= \text{let } \left( -, \rho'_{\text{Glob}}, \mu' \right) = \llbracket \pi \rrbracket \left( \rho_{\text{Loc}}, \rho_{\text{Glob}}, \mu \right) \\
&\quad \text{in } \llbracket \pi' \rrbracket \left( 0, \rho'_{\text{Glob}}, \mu' \right)
\end{align*}
\]
Program Analysis

\( \mathbb{D} \) ... **lattice**
→ all possible sets of analysis information that may hold at a program point

*idea*: collect information along all paths leading to a program point to yield analysis information that holds there

→ transformation of analysis information along edge \( k \) according to **abstract edge effect** \([k] \# : \mathbb{D} \rightarrow \mathbb{D}\)
Program Analysis

interprocedural

\textbf{enter}^\#: D \rightarrow D

\rightarrow \text{initialise information for the starting point of a function}

\textbf{combine}^\#: D^2 \rightarrow D

\rightarrow \text{combines information at the end of function body and information before entering the function}

\Rightarrow [k]^#D = \text{combine}^#(D, [f]^#(\text{enter}^#D))
Example: Copy Propagation

intraprocedural

**Copy Propagation:**
computes for variable \( x \) at each program point the set of variables that contain the same value
→ usage may be replaced by usage of \( x \)

abstract edge effects: \( ([k]^{\#} : \mathbb{D} \to \mathbb{D}) \)

\[
\begin{align*}
[x \leftarrow e]^{\#} V &= \{x\} \\
[x \leftarrow M[e]]^{\#} V &= \{x\} \\
[z \leftarrow y]^{\#} V &= (y \in V)? V \cup \{z\} : V\setminus\{z\} , \quad x \not\equiv z, y \in Vars \\
[z \leftarrow r]^{\#} V &= V\setminus\{z\} , \quad x \not\equiv z, r \notin Vars
\end{align*}
\]
Example: Copy Propagation

interprocedural

- all variables global:

  \[
  \text{enter}^\# V = V \\
  \text{combine}^\# (V_1, V_2) = V_2
  \]

- with local variables:

  •: auxiliary local variable to store value of \( x \) before the function call

  \[
  \text{enter}^\# V = V \cap Glob \cup \{\bullet\} \\
  \text{combine}^\# (V_1, V_2) = (V_2 \cap Glob) \cup ((\bullet \in V_2) ? V_1 \cap Loc_{\bullet} : \emptyset) \\
  \text{with } Loc_{\bullet} = Loc \cup \{\bullet\}
  \]
Abstract Effect of Function $f$

$\rightarrow [f] \#$: upper bound for abstract effect $[\pi] \#$ of every same-level computation $\pi$ for $f$

$\rightarrow$ approximated via

$[start_f] \# \supseteq \text{Id}$

$[v] \# \supseteq H[\#(f)] \circ [u] \#$,

$k = (u, f() , v)$ function call

$[v] \# \supseteq [k] \# \circ [u] \#$,

$k = (u, \text{lab}, v)$ normal edge

$[f] \# \supseteq [stop_f] \#$

with $[v] \# : \mathbb{D} \rightarrow \mathbb{D}$ describes effects of all same-level computations from the beginning of $f$ to program point $v$
Abstract Effects of Function $f$

right side of inequalities is monotone
$\rightarrow$ system of inequalities has smallest solution

$[.]$ be the smallest solution of the system of inequalities

1. $[v] \supseteq [\pi]$  
   $\forall$ same-level computations $\pi$ from $start_f$ to $v$

2. $[f] \supseteq [\pi]$  
   $\forall$ same-level computations $\pi$ of $f$

$\Rightarrow$ every solution of the system of inequalities can be used to approximate the abstract effect of a function call
Problems

- not always closed representation of monotone functions in the system of inequalities
- infinite ascending chains

⇒ in the case of copy propagation:
  - complete lattice $\mathbb{V} = \{ \mathcal{V} \subseteq \text{Vars.} | x \in \mathcal{V} \}$ is **atomic**
  - edge effects are **distributive** ($\rightarrow$ monotone)
  - no infinite ascending chains: only finitely many variables

→ compact representation of monotone functions exists:

$$g(\mathcal{V}) = b \cup \bigsqcup \{ h(a) | a \in A \land a \subseteq \mathcal{V} \}$$

with $h : A \rightarrow \mathbb{V}$, $b \in \mathbb{V}$, $A \subseteq \mathbb{V}$
Abstract Effects of Function $f$

ex. Copy Propagation

```plaintext
main() {
    A <- M[0];
    if (A) print();
    b <- A;
    work();
    ret <- 1-ret;
}

work() {
    A <- b;
    if (A) work();
    ret <- A;
}
```
Abstract Effects of Function $f$

ex. Copy Propagation

$\text{Vars}_\bullet = \{A, b, \text{ret}, \bullet\}$, investigate $b$

$\Rightarrow$

$\left\lceil A \leftarrow b \right\rceil^\# C = C \cup \{A\}$

$:= g_1(C)$

$\left\lceil \text{ret} \leftarrow A \right\rceil^\# C = (A \in C)? (C \cup \{\text{ret}\}) : (C\backslash\{\text{ret}\})$

$:= g_2(C)$
Abstract Effects of Function $f$

ex. Copy Propagation

represent edge effects $g_1, g_2$ by $(h_1, Vars_{\bullet}), (h_2, Vars_{\bullet})$:

(enumerable for finite lattice)

<table>
<thead>
<tr>
<th></th>
<th>$h_1$</th>
<th>$h_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>{$b, \text{ret}, \bullet$}</td>
<td>$Vars_{\bullet}$</td>
<td>{$b, \bullet$}</td>
</tr>
<tr>
<td>{$b, A, \bullet$}</td>
<td>{$b, A, \bullet$}</td>
<td>$Vars_{\bullet}$</td>
</tr>
<tr>
<td>{$b, A, \text{ret}$}</td>
<td>{$b, A, \text{ret}$}</td>
<td>{$b, A, \text{ret}$}</td>
</tr>
</tbody>
</table>

$g_1 (C) = C \cup \{A\}$

$g_2 (C) = (A \in C)? (C \cup \{\text{ret}\}) : (C\{\text{ret}\})$
Abstract Effects of Function $f$

ex. Copy Propagation

$C$: set of variables that initially have the same value as $b$

work():

\[
\begin{align*}
7 &\quad A \leftarrow b \\
8 &\quad \text{NonZero}(A) \\
9 &\quad \text{Zero}(A) \\
10 &\quad \text{work()} \\
11 &\quad \text{ret} \leftarrow A
\end{align*}
\]

\[
\begin{align*}
[A \leftarrow b] &\quad \# C = C \cup \{A\} := g_1(C) \\
[\text{ret} \leftarrow A] &\quad \# C = (A \in C)? (C \cup \{\text{ret}\}) : (C \setminus \{\text{ret}\}) := g_2(C)
\end{align*}
\]
Abstract Effects of Function $f$

**ex. Copy Propagation**

$C$: set of variables that initially have the same value as $b$

$$\text{work}()$$

$7$ → $\text{ID}(C)$

$A \leftarrow b$

$8$

$\text{Zero}(A)$

$\text{NonZero}(A)$

$9$

$\text{work}()$

$10$

$\text{ret} \leftarrow A$

$11$

$$[A \leftarrow b] C = C \cup \{A\} : = g_1(C)$$

$$[\text{ret} \leftarrow A] C = (A \in C)? (C \cup \{\text{ret}\}) : (C \setminus \{\text{ret}\}) : = g_2(C)$$
Abstract Effects of Function $f$

**ex. Copy Propagation**

Let $C$ be the set of variables that initially have the same value as $b$.

```
work():

7 -> ID(C)
A <- b

8 -> g1(C)
NonZero(A)
Zero(A)

9 -> g1(C)

10 -> work()

11 -> ret <- A
```

Given:

\[
\begin{align*}
[A \leftarrow b] & \# C = C \cup \{A\} := g_1(C) \\
[ret \leftarrow A] & \# C = (A \in C)? (C \cup \{ret\}) : (C \setminus \{ret\}) := g_2(C)
\end{align*}
\]
Abstract Effects of Function $f$

ex. Copy Propagation

$C$: set of variables that initially have the same value as $b$

```
work():

7 → ID(C)

A ← b
8 → g1(C)

NonZero(A)
9 → g1(C)

Zero(A)
10 → work()

ret ← A
11
```

$$[[A \leftarrow b]]^C = C \cup \{A\} := g_1(C)$$

$$[[ret \leftarrow A]]^C = (A \in C)? (C \cup \{ret\}) : (C\setminus\{ret\}) := g_2(C)$$
Abstract Effects of Function \( f \)

**ex. Copy Propagation**

\( C \): set of variables that initially have the same value as \( b \)

\[
\begin{align*}
A &\leftarrow b \\
g_1(C) &\Rightarrow ID(C) \\
NonZero(A) &\rightarrow g_1(C) \\
Zero(A) &\rightarrow g_1(C) \\
work() &\rightarrow g_1(C) \\
ret &\leftarrow A \\
\end{align*}
\]

\[
\begin{align*}
[A \leftarrow b] &\# C = C \cup \{A\} := g_1(C) \\
[ret \leftarrow A] &\# C = (A \in C)? (C \cup \{ret\}) : (C \setminus \{ret\}) := g_2(C)
\end{align*}
\]
Abstract Effects of Function $f$

**ex. Copy Propagation**

$C$: set of variables that initially have the same value as $b$

$work()$

\[
\begin{align*}
7 & \quad \rightarrow \quad \text{Id}(C) \\
A & \leftarrow b \\
8 & \quad \rightarrow \quad g_1(C) \\
\text{NonZero}(A) & \quad \rightarrow \quad g_1(C) \\
\text{Zero}(A) & \quad \rightarrow \quad g_1(C) \\
9 & \quad \rightarrow \quad g_1(C) \\
\text{work()} & \quad \rightarrow \quad g_1(C) \\
10 & \quad \rightarrow \quad g_1(C) \\
\text{ret} & \leftarrow A \\
11 & \quad \rightarrow \quad g_2 \circ g_1(C) = C \cup \{A, \text{ret}\} =: g_3(C)
\end{align*}
\]

\[
\begin{align*}
[A & \leftarrow b] \# C = C \cup \{A\} := g_1(C) \\
[\text{ret} & \leftarrow A] \# C = (A \in C)?(C \cup \{\text{ret}\}) : (C\setminus\{\text{ret}\}) := g_2(C)
\end{align*}
\]
Abstract Effects of Function $f$

ex. Copy Propagation

$C$: set of variables that initially have the same value as $b$

work():

1. $A \leftarrow b$
2. $A \leftarrow b$
3. $A \leftarrow b$
4. $A \leftarrow b$
5. $A \leftarrow b$
6. $A \leftarrow b$
7. $A \leftarrow b$
8. $A \leftarrow b$
9. $A \leftarrow b$
10. $A \leftarrow b$
11. $A \leftarrow b$

First approximation for call of work:

$\text{combine}^\# (C, g_3 (\text{enter}^\# (C))) = C \cup \{\text{ret}\} := g_4 (C)$
Abstract Effects of Function \( f \)

ex. Copy Propagation

\( C \): set of variables that initially have the same value as \( b \)

\( \text{work}() \):

1. \( A \leftarrow b \)
2. \( \text{NonZero}(A) \)
3. \( \text{Zero}(A) \)

\( \text{ID}(C) \) \rightarrow \( g_1(C) \)

\( \text{g}_2 \circ \text{g}_1(C) = C \cup \{A, \text{ret}\} =: g_3(C) \)

First approximation for call of \( \text{work}() \):

\( \text{combine}^\#(C, g_3(\text{enter}^\#(C))) = C \cup \{\text{ret}\} := g_4(C) \)
Abstract Effects of Function $f$

ex. Copy Propagation

$C$: set of variables that initially have the same value as $b$

work():

1. $A \leftarrow b$
2. $A \leftarrow b$
3. $A \leftarrow b$
4. $A \leftarrow b$
5. $A \leftarrow b$
6. $A \leftarrow b$
7. $A \leftarrow b$
8. $A \leftarrow b$
9. $A \leftarrow b$
10. $A \leftarrow b$
11. $A \leftarrow b$

first approximation for call of work:

$\text{combine}^\#(C, g_3(\text{enter}^\#(C))) = C \cup \{\text{ret}\} := g_4(C)$
Abstract Effects of Function $f$

ex. Copy Propagation

$C$: set of variables that initially have the same value as $b$

work():

7

$A \leftarrow b$

8

NonZero($A$)

$g_1(C)$

9

Zero($A$)

$g_1(C)$

10

work()

$g_1(C)$

11

$ret \leftarrow A$

$g_3(C) = C \cup \{A, ret\}$

first approximation for call of work:

$\text{combine}^\#(C, g_3(\text{enter}^\#(C))) = C \cup \{ret\} := g_4(C)$
Abstract Effects of Function \( f \)

**ex. Copy Propagation**

\( C \): set of variables that initially have the same value as \( b \)

work():

1. \( A \leftarrow b \)
2. \( \text{NonZero}(A) \)
3. \( \text{Zero}(A) \)
4. \( \text{work}() \)
5. \( \text{ret} \leftarrow A \)

\[ g_3(C) = C \cup \{A, \text{ret}\} \]

first approximation for call of \( \text{work} \):

\[ \text{combine}^\#(C, g_3(\text{enter}^\#(C))) = C \cup \{\text{ret}\} := g_4(C) \]
Abstract Effects of Function $f$

ex. Copy Propagation

$C$: set of variables that initially have the same value as $b$

$\text{work}()$:

1. $A \leftarrow b$
2. $\text{NonZero}(A)$
3. $\text{Zero}(A)$
4. $\text{work}()$
5. $\text{ID}(C)$
6. $g_1(C)$
7. $\text{ID}(C)$
8. $g_1(C)$
9. $g_1(C)$
10. $g_1(C)$
11. $g_1(C)$

fixpoint reached after first iteration:

$\text{work}$ approximated by $g_4(C) = C \cup \{\text{ret}\}$
Coindidence Theorem

- ∃ same-level computation from \( start_f \) to \( v \) \( \forall v \in f \), edge effects and transformation \( H^\# \) are distributive

\[ [v]^\# = \bigsqcup \{ [\pi]^\# | \pi \in \mathcal{T}_v \} \forall v \in f \]
\( (\mathcal{T}_v \ldots \text{set of all same-level computations from } start_f \text{ to } v) \)

- enter\(^\# \) distributive, combine\(^\# \) \( (x_1, x_2) = h_1(x_1) \sqcup h_2(x_2) \)

\[ H^\# (\bigsqcup \mathcal{F}) = \bigsqcup \{ H^\# (g) | g \in \mathcal{F} \} \]
Coincidence Theorem

**ex. Copy Propagation**

\[ \text{enter}^\# V = V \cap \text{Glob} \cup \{\bullet\} \]

\[ \rightarrow \text{distributive} \]

\[ \text{combine}^\# (V_1, V_2) = (V_2 \cap \text{Glob}) \cup (\bullet \in V_2)\,? V_1 \cap \text{Loc} : \emptyset \]

\[ = ((V_1 \cap \text{Loc}_\bullet) \cup \text{Glob}) \cap \\
(V_2 \cap \text{Glob}) \cup \text{Loc}_\bullet) \cap \\
(Glob \cup (\bullet \in V_2)\,? \text{Vars}_\bullet : \text{Glob}) \]

\[ \rightarrow \text{intersection of distributive functions of first and second argument} \]

\[ \Rightarrow \text{coincidence theorem holds for copy propagation} \]
Interprocedural Reachability

effects $[f]#$ are approximated
→ compute for program point $u$ a safe approximation of
property $\mathcal{D}[u]$ that holds when $u$ is reached

$$
\begin{align*}
\mathcal{D}[\text{start}_{\text{main}}] & \equiv \text{enter}^#(d_0) \\
\mathcal{D}[\text{start}_{f}] & \equiv \text{enter}^#(\mathcal{D}[u]), \\
& \quad (u,f(),v) \text{ calling edge} \\
\mathcal{D}[v] & \equiv \text{combine}^#(\mathcal{D}[u],[f]^#(\text{enter}^#(\mathcal{D}[u]))), \\
& \quad (u,f(),v) \text{ calling edge} \\
\mathcal{D}[v] & \equiv [k]^#(\mathcal{D}[u]), \\
& \quad k = (u,lab,v) \text{ normal edge}
\end{align*}
$$
Interprocedural Reachability

smallest solution for system of inequalities exists because of monotonicity and it holds:

$$\mathcal{D}[v] \supseteq [\pi]'#d_0$$

for all paths that reach $v$

($d_0 \in \mathcal{D}$: information at the beginning of program execution)

for distributive abstract edge effects and distributive transformation $H'$:

$$\mathcal{D}[v] = \bigcup\{[\pi]'#d_0 | \pi \in \mathcal{P}_v\}$$

with $\mathcal{P}_v$ ... set of all paths that reach $v$
Interprocedural Reachability

example

```plaintext
main():
    A ← M[0]
    Zero(A)
    NonZero(A)
    print()
    b ← A  
    Zero(A)
    work()
    ret ← 1 - ret

work():
    A ← b
    NonZero(A)
    Zero(A)
    work()
    ret ← A
```

\[ D[start_{main}] \sqcup \{b\]
Interprocedural Reachability

**example**

```
main():

0 → {b}

A ← M[0]

1 → [A ← M[0]]#(D[0]) = D[0]

NonZero(A)

Zero(A)

2

print()

3 → b ← A

4

work()

5

ret ← 1 − ret

6

work() approximated by g_{4}(C) = C \cup \{\text{ret}\}

7

work():

A ← b

8

NonZero(A)

Zero(A)

9

work()

10 → ret ← A

11

combine #(D[4], J_f K#(enter#(D[4])))

J_ret ← ret − 1

⇒ within the call of work: global var. b may be used instead of local var. A
```
Interprocedural Reachability

example

main():

0 → \{b\}

A ← M[0]

1 → \{b\}

Zero(A)

2 → [NonZero(A)]\#(D[1]) = D[1]Zero(A)

print()

3 → b ← A

4 → work()

5 → ret ← 1 - ret

6 →

work():

7 → A ← b

8 → NonZero(A)

9 → work()

10 → ret ← A

11 →

D[0] \subseteq \{b\}
Interprocedural Reachability

example

main():

0 \rightarrow \{b\}

A \leftarrow M[0]

1 \rightarrow \{b\}

NonZero(A)

2 \rightarrow \{b\}

Zero(A)

3 \rightarrow \text{[Zero]} \# (D[1]) \cap \text{[print()]} \# (D[2])

b \leftarrow A

4

work()

5

ret \leftarrow 1 - ret

6

work approximated by:

g_4(C) = C \cup \{\text{ret}\}

work():

7

A \leftarrow b

8

NonZero(A)

9

Zero(A)

10

work()

11

ret \leftarrow A

\Rightarrow within the call of work:

global var. b may be used instead of local var. A.
Interprocedural Reachability

example

main():

0 → \{b\}
A ← M[0]

1 → \{b\}
NonZero (A)

2 → \{b\}
Zero (A)
print()

3 → \{b\}
b ← A

4 → \{b\}
work()

5 → \{b\}

6 → \{b\}

work():

7 → A ← b

8 → NonZero (A)

9 → Zero (A)

10 → ret ← A

11 → \{b\}
Interprocedural Reachability

**example**

**main()**:
- Node 0: \( A \leftarrow M[0] \) → \{b\}
- Node 1: \( A \leftarrow M[0] \) → \{b\}
- Node 2: \( \text{NonZero}(A) \) → \{b\}
- Node 3: \( b \leftarrow A \) → \{b\}
- Node 4: \( b \leftarrow A \) → \{b\}
- Node 5: \( \text{work}() \)
- Node 6: \( \text{ret} \leftarrow 1 - \text{ret} \)

**work()**:
- Node 7: \( \text{enter}#(D[4]) = \{b, \bullet\} \)
- Node 8: \( A \leftarrow b \)
- Node 9: \( \text{NonZero}(A) \)
- Node 10: \( \text{Zero}(A) \) → \{b\}
- Node 11: \( \text{work}() \)
Interprocedural Reachability

example

main():

0 \rightarrow \{b\}

1 \rightarrow \{b\}

2 \rightarrow \{b\}

3 \rightarrow \{b\}

4 \rightarrow \{b\}

5 \rightarrow \{b\}

6 \rightarrow \{b\}

A \leftarrow M[0]

b \leftarrow A

print()

work()

work approximated by

g_4(C) = C \cup \{ret\}

work():

7 \rightarrow \{b\}

8 \rightarrow \{b\}

9 \rightarrow \{b\}

10 \rightarrow \{b\}

11 \rightarrow \{b\}

A \leftarrow b

A \leftarrow b\#(D[7])

ret \leftarrow A

⇒ within the call of work:
global var. b may be used instead of local var. A
Interprocedural Reachability

Example

main():

0 → \{b\}
A ← M[0]

1 → \{b\}
NonZero (A)

2 → \{b\}
Zero (A)

3 → \{b\}
print()

b ← A

4 → \{b\}
work()

5 → \{b\}

6 

work():

7 → \{b\}
A ← b

8 → \{b, A, •\}
NonZero (A)

9 → [NonZero (A)] # (D[8])
Zero (A)

10 → \{b, A, •\}
work()

11 → \{b\}

12 → \{b\}
ret ← A

D[0] → \{b\}
ret ← 1 - ret

⇒ within the call of work:
global var. b may be used instead of local var. A.

41 / 54
Interprocedural Reachability

eample

main():

0  \rightarrow \{b\}

A \leftarrow M[0]

1  \rightarrow \{b\}

Zero (A)

NonZero (A)

2  \rightarrow \{b\}

print ()

3  \rightarrow \{b\}

b \leftarrow A

4  \rightarrow \{b\}

work ()

5  \rightarrow \{b\}

ret \leftarrow 1 - ret

6

work ():

7  \rightarrow \{b\}

A \leftarrow b

8  \rightarrow \{b, A, \bullet\}

NonZero (A)

Zero (A)

9  \rightarrow \{b, A, \bullet\}

work ()

10

[Zero (A)] \# (D[8]) \cap [work] \# (D[9])

[Zero (A)] \# (D[8])

11

work approximated by
g_4 (C) = C \cup \{ret\}
Interprocedural Reachability

element

main():

- 0: {b}
- 1: {b}
- 2: {b}
- 3: {b}
- 4: {b}
- 5: {b}
- 6: {b}

A ← M[0]

NonZero(A)

Zero(A)

print()

b ← A

work()

ret ← 1 – ret

work():

- 7: {b}
- 8: {b, A, •}
- 9: {b, A, •}
- 10: {b, A, •}
- 11: [ret ← A] # (D[10])

A ← b

NonZero(A)

Zero(A)

work()

ret ← A

[ret ← A] # (D[10])
Interprocedural Reachability

example

main():

0
A ← M[0]

1
A ← M[0]

2
NonZero(A)
Zero(A)

b ← A

3
print()

4
b ← A

5
work()

6
ret ← 1 - ret

7
work():

8
A ← b

9
NonZero(A)
Zero(A)

10
work()

11
ret ← A


⇒ within the call of work:

global var. b may be used instead of local var. A
Interprocedural Reachability

example

main():

0 → \{b\}

A ← M[0]

1 → \{b\}

NonZero(A)

2 → \{b\}

Zero(A)

print()

3 → \{b\}

b ← A

4 → \{b\}

work()

5 → \{b, ret\}

ret ← 1 – ret

6 → \{ ret ← ret – 1 \} # (D[5])

work():

7 → \{b\}

A ← b

8 → \{b, A, \bullet\}

NonZero(A)

9 → \{b, A, \bullet\}

Zero(A)

work()

10 → \{b, A, \bullet\}

ret ← A

11 → \{b, A, \bullet, ret\}
Interprocedural Reachability

example

main():

\[ A \leftarrow M[0] \]

\[ \text{Zero} (A) \]

\[ 2 \rightarrow \{b\} \]

\[ \text{NonZero} (A) \]

\[ 1 \rightarrow \{b\} \]

\[ \text{print} () \]

\[ 3 \rightarrow \{b\} \]

\[ b \leftarrow A \]

\[ 4 \rightarrow \{b\} \]

\[ \text{work} () \]

\[ 5 \rightarrow \{b, \text{ret}\} \]

\[ \text{ret} \leftarrow 1 - \text{ret} \]

\[ 6 \rightarrow \{b\} \]

work():

\[ 7 \rightarrow \{b\} \]

\[ A \leftarrow b \]

\[ 8 \rightarrow \{b, A, \bullet\} \]

\[ \text{NonZero} (A) \]

\[ 9 \rightarrow \{b, A, \bullet\} \]

\[ \text{Zero} (A) \]

\[ 10 \rightarrow \{b, A, \bullet\} \]

\[ \text{work} () \]

\[ 11 \rightarrow \{b, A, \bullet, \text{ret}\} \]

\[ \Rightarrow \text{within the call of work:} \]

\[ \text{global var.} \ b \ \text{may be used instead of local var.} \ A \]
Introduction

Simple Interprocedural Optimisations

Operational Semantic

Functional Approach

Related Approaches

Summary
Demand-Driven Interprocedural Analysis

sometimes: lattice not finite, functions cannot be represented in a compact form
→ only analyse calls in situations that really occur

! this is the case e.g. for constant propagation

→ use local fixpoint algorithm:
only compute solutions for certain inequalities;
only solve part of the system that is needed therefor
Demand-Driven Interprocedural Analysis

system of inequalities

\[ D[v, a] \; \subseteq \; a, \]
\( \quad v \) entry point

\[ D[v, a] \; \subseteq \; \text{combine}^\# (D[u, a], D[f, \text{enter}^\# (D[u, a]))), \]
\( (u, f(), v) \) calling edge

\[ D[v, a] \; \subseteq \; [lab]^\# (D[u, a]), \]
\( k = (u, lab, v) \) normal edge

\[ D[f, a] \; \subseteq \; D[\text{stop}_f, a] \]

with \( D[f, a] \) ... abstract state when reaching program point \( v \)

of a function called in abstract state \( a \) \( (D[f, a] \sim [v]^\# (a)) \)

\( \Rightarrow \) compute \( D[\text{main}, \text{enter}^\# (d_0)] \)
Demand-Driven Interprocedural Analysis

ex. Constant Propagation

**Constant Propagation:**
move as many computations as possible from runtime to compile time
complete lattice: $\mathbb{D} = (\text{Vars} \rightarrow \mathbb{Z}^\top) \bot$
$\rightarrow !$ not finite

```
enter# D = \begin{cases} 
\bot & D = \bot \\
D \oplus \{A \mapsto T | A \text{ local}\} & \text{otherwise}
\end{cases}
```

```
combine# (D_1, D_2) = \begin{cases} 
\bot & D_1 = \bot \vee D_2 = \bot \\
D_1 \oplus \{b \mapsto D_2 (b) | b \text{ global}\} & \text{otherwise}
\end{cases}
```
Constant Propagation

Abstract Edge Effects - intraprocedural

\[
\begin{align*}
\llbracket ; \rrbracket^#D &= D \\
\llbracket \text{NonZero}(e) \rrbracket^#D &= \begin{cases} 
\bot & \text{if } 0 = \llbracket e \rrbracket^#D \\
D & \text{otherwise}
\end{cases} \\
\llbracket \text{Zero}(e) \rrbracket^#D &= \begin{cases} 
\bot & \text{if } 0 \nsubseteq \llbracket e \rrbracket^#D \\
D & \text{if } 0 \subseteq \llbracket e \rrbracket^#D
\end{cases} \\
\llbracket x \leftarrow e \rrbracket^#D &= D \oplus \{ x \mapsto \llbracket e \rrbracket^#D \} \\
\llbracket x \leftarrow M[e] \rrbracket^#D &= D \oplus \{ x \mapsto \top \} \\
\llbracket M[e_1] \leftarrow e_2 \rrbracket^#D &= D
\end{align*}
\]
Demand-Driven Interprocedural Analysis

ex. Constant Propagation

\[ d_0 = \{ A \mapsto \top, \; b \mapsto \top, \; \text{ret} \mapsto \top \} \]

main():

\[ d_1 = \{ A \mapsto \top, \; b \mapsto 0, \; \text{ret} \mapsto \top \} \]

work():

<table>
<thead>
<tr>
<th>A</th>
<th>b</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Call-String-Approach

→ compute set of all reachable call stacks
  ! restrict call stacks to fixed size $d$
→ (complexity increases with depth)

here: call stack of depth 0
→ function call as unconditional jump
Call-String-Approach

system of inequalities

\[ \mathcal{D}[start_{\text{main}}] \sqsubseteq \text{enter}\#(d_0) \]
\[ \mathcal{D}[start_f] \sqsubseteq \text{enter}\#(\mathcal{D}[u]), \]
\[ (u, f(), v) \text{ calling edge} \]
\[ \mathcal{D}[v] \sqsubseteq \text{combine}\#(\mathcal{D}[u], \mathcal{D}[v]), \]
\[ (u, f(), v) \text{ calling edge} \]
\[ \mathcal{D}[v] \sqsubseteq [lab]\#(\mathcal{D}[u]), \]
\[ k = (u, lab, v) \text{ normal edge} \]
\[ \mathcal{D}[f] \sqsubseteq \mathcal{D}[\text{stop}_f] \]
Call-String-Approach

ex. Copy Propagation

main():

A ← 0

work():

A ← b

Zero(A)

NonZero(A)

print()

b ← A

work()

ret ← 1 − ret

interprocedural supergraph
Call-String-Approach
ex. Copy Propagation

$\mathcal{D}[5] \sqsubseteq \text{combine}^\#: (\mathcal{D}[4], \mathcal{D}[\text{work}])$

$\mathcal{D}[7] \sqsubseteq \text{enter}^\#: (\mathcal{D}[4])$

$\mathcal{D}[7] \sqsubseteq \text{enter}^\#: (\mathcal{D}[9])$

$\mathcal{D}[10] \sqsubseteq \text{combine}^\#: (\mathcal{D}[9], \mathcal{D}[\text{work}])$
Call-String-Approach

ex. Copy Propagation

! for depth 0: impossible paths may occur
Introduction

Simple Interprocedural Optimisations

Operational Semantic

Functional Approach

Related Approaches

Summary
Summary

- Interprocedural Analysis is an extension of intraprocedural analysis which takes into account the calling context of functions.
- Interprocedural Analysis is more demanding than intraprocedural analysis, but yields more precise results.
- **Functional Approach**: approximate abstract effect of function call by solving system of inequalities describing the edge effects within the function.
- Lattice of possible analysis solutions has to fulfill certain properties to ensure that the analysis terminates.