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
  - optimise function calls without explicit modelling e.g.
    - **Inlining**
    - **Remove Last Call**

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

Interprocedural vs. Intraprocedural

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

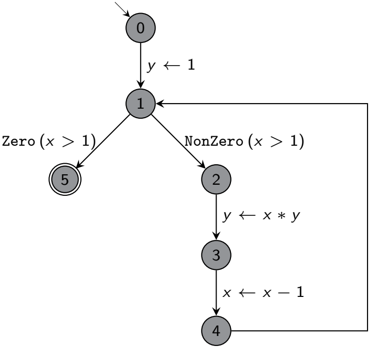
interprocedural optimisation:
context-sensitive optimisation:
different calls reached with different contexts δ_1 and δ_2
$\rightarrow$ information obtained clearly related to δ_1 and δ_2
$\Rightarrow$ more precise, but more costly

Program Representation

intraprocedural

→ program represented by a **control flow graph**:

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



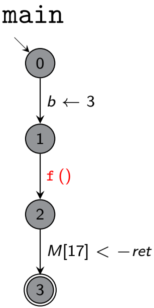
Program Representation

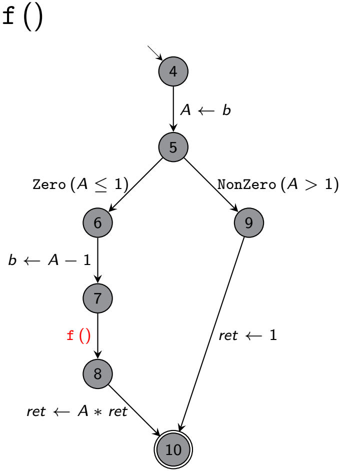
interprocedural

$\rightarrow$ program represented by a set of control flow graphs;

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

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





Edge Annotations

(x ... variable, e ... 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` ()

Inlining

inlining:
copy function body to calling point

problems:

- function has to be statically known
- local variables of calling function must not be modified
  $\rightarrow$ rename local variables
- recursive functions
  $\rightarrow$ identified from **call graph**

$\rightarrow$

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

Inlining

Call Graph

Call Graph:

nodes ∼ functions

edges ∼ between function $\texttt{f}_1$ and function $\texttt{f}_2$, if $\texttt{f}_1$ calls $\texttt{f}_2$

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

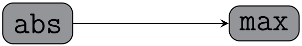
```
f(){
   A <- b;
   if (A <=1) ret <- 1;
   else {
       b <- A-1;
       f();
       ret <- A*ret;
   }
}
```

Inlining

Call Graph

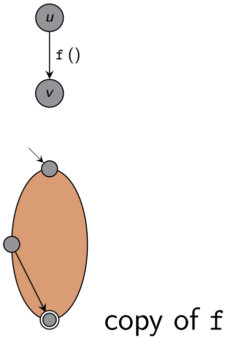
```
abs(){
    b_1 <- b;
    b_2 <- -b;
    max();
}
```

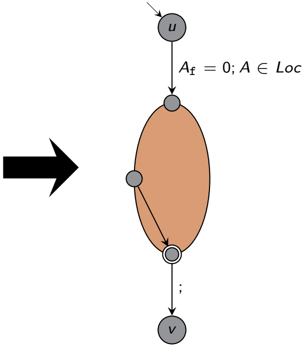
```
max(){
    if (b_1 < b_2) ret <- b_2;
    else ret <- b_1;
}
```



Inlining

transformation PI:





Inlining

example

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

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

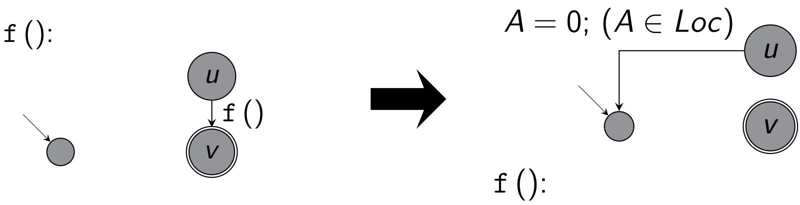
→

```
abs(){
    b_1 <- b;
    b_2 <- -b;
    if (b_1 < b_2) ret <- b2;
    else ret <- b_1;
}
```

Remove Last Calls

$\rightarrow$ 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:



Remove Last Calls

example

```
f(){
    if (b_2 <= 1) ret <- b_1;
    else {
          b_1 <- b_1*b_2;
          b_2 <- b_2 - 1;
          f();
    }
}
```

➡

```
f(){
    _f: if (b_2 <= 1) ret <- b_1;
        else {
               b_1 <- b_1*b_2;
               b_2 <- b_2 - 1;
               goto _f;
        }
}
```

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
  $\rho : \mathit{Vars} \to \mathsf{int}$ ... value of variables
  $\mu : \mathbb{N} \to \mathsf{int}$ ... content of memory
- edge $k = (u, \mathit{lab}, v)$
  ... entry node u, exit node v, edge annotation *label*
- **edge effect**: transformation $[\![k]\!]$ on program states defined by the edge k
  $[\![k]\!] = [\![\mathit{lab}]\!]$

Operational Semantic

Edge Effects - intraprocedural

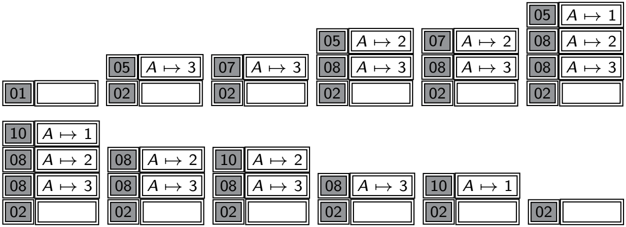
$$
\begin{aligned}
[\![;]\!]\,(\rho,\mu) &= (\rho,\mu) \\
[\![\texttt{NonZero}\,(e)]\!]\,(\rho,\mu) &= (\rho,\mu)\,, \\
&\quad \text{if } [\![e]\!]\rho \neq 0 \\
[\![\texttt{Zero}\,(e)]\!]\,(\rho,\mu) &= (\rho,\mu)\,, \\
&\quad \text{if } [\![e]\!]\rho = 0 \\
[\![x \leftarrow e]\!]\,(\rho,\mu) &= \left(\boxed{\rho \oplus \{x \mapsto [\![e]\!]\rho\}}, \mu\right) \\
[\![x \leftarrow M[e]]\!]\,(\rho,\mu) &= \left(\boxed{\rho \oplus \{x \mapsto \mu\,([\![e]\!]\rho)\}}, \mu\right) \\
[\![M[e_1] \leftarrow e_2]\!]\,(\rho,\mu) &= \left(\rho, \boxed{\mu \oplus \{[\![e_1]\!]\rho \mapsto [\![e_2]\!]\rho\}}\right)
\end{aligned}
$$

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

$$
\begin{aligned}
\mathit{config} &= \mathit{stack} \times \mathit{globals} \times \mathit{store} \\
\mathit{globals} &= \mathit{Glob} \to \mathbb{Z} \\
\mathit{store} &= \mathbb{N} \to \mathbb{Z} \\
\mathit{stack} &= \mathit{frame} \cdot \mathit{frame}^* \\
\mathit{frame} &= \mathit{point} \times \mathit{locals} \\
\mathit{locals} &= \mathit{Loc} \to \mathbb{Z}
\end{aligned}
$$

! function body is a scope with own local variables

Modeling of Function Call

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

$$\underbrace{\left(\sigma \cdot \boxed{(u, \rho_{Loc})}, \rho_{Glob}, \mu\right)}_{config} \quad \vdash \quad \left(\sigma \cdot \boxed{(v, \rho_{\mathrm{Loc}}) \cdot (u_{\mathrm{f}}, \rho_{\mathrm{f}})}, \rho_{Glob}, \mu\right)$$

- effect of function itself
- return from call:

$$\left(\sigma \cdot \boxed{(v, \rho_{Loc}) \cdot (r_{\mathrm{f}}, \_)}, \rho_{Glob}, \mu\right) \quad \vdash \quad \left(\sigma \cdot \boxed{(v, \rho_{Loc})}, \rho_{Glob}, \mu\right)$$

σ	...	stack
ρ_{Glob}	...	global variables
μ	...	store
(u, ρ_{Loc})	...	frame (*point* $\times$ *locals*)

Path Effects

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

path π defines a partial function $[\![\pi]\!]$, that transforms $((u, \rho_{Loc}), \rho_{Glob}, \mu)$ into $((v, \rho'_{Loc}), \rho'_{Glob}, \mu')$

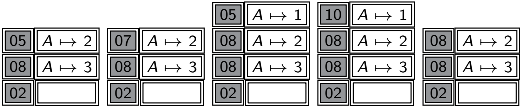
$\Rightarrow$ compute transformation inductive over the structure of the path:

$$[\![\pi k]\!] = [\![k]\!] \circ [\![\pi]\!]$$

for a normal edge k (**composition of edge effects**)

Paths Effects

- **same-level**: all entered functions are also left again
  $\pi = \pi_1\langle\texttt{f}\rangle\pi_2\langle\backslash\texttt{f}\rangle$



$\rightarrow$ height of the stack stays the same

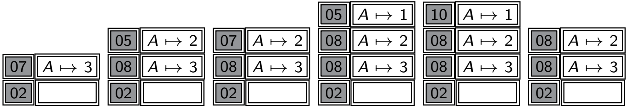
$$[\![\pi_1\langle\texttt{f}\rangle\pi_2\langle\backslash\texttt{f}\rangle]\!] \quad = \quad H\left([\![\pi_2]\!]\right) \circ [\![\pi_1]\!]$$

with

$$H\left(g\right)\left(\rho_{Loc}, \rho_{Glob}, \mu\right) \quad = \quad \textbf{let}\left(\rho'_{Loc}, \rho'_{Glob}, \mu'\right) = g\left(\underline{0}, \rho_{Glob}, \mu\right) \\ \textbf{in}\left(\rho_{Loc}, \rho'_{Glob}, \mu'\right)$$

Path Effects

- **computation that reaches a program point**: $\pi\langle \mathtt{f}\rangle\pi'$ with π, π' is same-level



$$\begin{aligned}\llbracket\pi\langle \mathtt{f}\rangle\pi'\rrbracket\,(\rho_{Loc}, \rho_{Glob}, \mu) \quad &= \quad \textbf{let}\,\left(\_, \rho'_{Glob}, \mu'\right) = \llbracket\pi\rrbracket\,(\rho_{Loc}, \rho_{Glob}, \mu) \\ &\qquad \textbf{in}\llbracket\pi'\rrbracket\,\left(\underline{0}, \rho'_{Glob}, \mu'\right)\end{aligned}$$

Program Analysis

$\mathbb{D}$ ... **lattice**
$\rightarrow$ 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

$\rightarrow$ transformation of analysis information along edge k according to **abstract edge effect** $[\![k]\!]^{\#} : \mathbb{D} \rightarrow \mathbb{D}$

Program Analysis

interprocedural

$\texttt{enter}^{\#} : \mathbb{D} \to \mathbb{D}$

$\to$ initialise information for the starting point of a function

$\texttt{combine}^{\#} : \mathbb{D}^2 \to \mathbb{D}$

$\to$ combines information at the end of function body and information before entering the function

$$\Rightarrow [\![k]\!]^{\#} D = \texttt{combine}^{\#} \left(D, [\![\texttt{f}]\!]^{\#} \left(\texttt{enter}^{\#} D\right)\right)$$

Example: Copy Propagation

intraprocedural

Copy Propagation:

computes for variable x at each program point the set of variables that contain the same value

$\rightarrow$ usage may be replaced by usage of x

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

$$
\begin{array}{rcl}
[\![x \leftarrow e]\!]^\# V & = & \{x\} \\
[\![x \leftarrow M[e]]\!]^\# V & = & \{x\} \\
[\![z \leftarrow y]\!]^\# V & = & (y \in V) ? V \cup \{z\} : V \backslash \{z\}, \\
 & & x \not\equiv z, y \in \mathit{Vars} \\
[\![z \leftarrow r]\!]^\# V & = & V \backslash \{z\}, \\
 & & x \not\equiv z, r \notin \mathit{Vars}
\end{array}
$$

Example: Copy Propagation

interprocedural

- all variables global:

$$\texttt{enter}^{\#} V = V$$
$$\texttt{combine}^{\#} (V_1, V_2) = V_2$$

- with local variables:

  $\bullet$: auxiliary local variable to store value of x before the function call

$$\texttt{enter}^{\#} V = V \cap Glob \cup \{\bullet\}$$
$$\texttt{combine}^{\#} (V_1, V_2) = (V_2 \cap Glob) \cup ((\bullet \in V_2) ? V_1 \cap Loc_\bullet : \emptyset)$$

with $Loc_\bullet = Loc \cup \{\bullet\}$

Abstract Effect of Function $\texttt{f}$

→ $[\![\texttt{f}]\!]^\#$: upper bound for abstract effect $[\![\pi]\!]^\#$ of every same-level computation π for $\texttt{f}$

→ approximated via

$$
\begin{array}{rcl}
[\![start_{\texttt{f}}]\!]^\# & \sqsupseteq & \texttt{Id} \\
[\![v]\!]^\# & \sqsupseteq & H^\#\left([\![\texttt{f}]\!]^\#\right) \circ [\![u]\!]^\#, \\
 & & k = \left(u, \texttt{f}\,()\,, v\right) \text{ function call} \\
[\![v]\!]^\# & \sqsupseteq & [\![k]\!]^\# \circ [\![u]\!]^\#, \\
 & & k = \left(u, lab, v\right) \text{ normal edge} \\
[\![\texttt{f}]\!]^\# & \sqsupseteq & [\![stop_{\texttt{f}}]\!]^\#
\end{array}
$$

with $[\![v]\!]^\# : \mathbb{D} \to \mathbb{D}$ describes effects of all same-level computations from the beginning of $\texttt{f}$ 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. $[\![v]\!]^{\#} \sqsupseteq [\![\pi]\!]^{\#}$
   $\forall$ same-level computations π from $start_{\texttt{f}}$ to v
2. $[\![\texttt{f}]\!]^{\#} \sqsupseteq [\![\pi]\!]^{\#}$
   $\forall$ same-level computations π 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

$\Rightarrow$ in the case of copy propagation:

- complete lattice $\mathbb{V} = \{V \subseteq Vars_\bullet | x \in V\}$ is **atomic**
- edge effects are **distributive** ($\rightarrow$ monotone)
- no infinite ascending chains: only finitely many variables

$\rightarrow$ compact representation of monotone functions exists:

$$g(V) \;=\; b \sqcup \bigsqcup \{h(a) \,|\, a \in A \wedge a \sqsubseteq V\}$$

with $h : A \rightarrow \mathbb{V}, b \in \mathbb{V}, A \subseteq \mathbb{V}$

Abstract Effects of Function `f`

ex. Copy Propagation

```
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

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

$\Rightarrow$

$$
\begin{aligned}
[\![A \leftarrow b]\!]^\# C &= C \cup \{A\} \\
&:= g_1(C) \\
[\![\text{ret} \leftarrow A]\!]^\# C &= (A \in C)?\,(C \cup \{ret\}) : (C \backslash \{ret\}) \\
&:= g_2(C)
\end{aligned}
$$

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)

	h_1	h_2
$\{b, ret, \bullet\}$	$Vars_\bullet$	$\{b, \bullet\}$
$\{b, A, \bullet\}$	$\{b, A, \bullet\}$	$Vars_\bullet$
$\{b, A, ret\}$	$\{b, A, ret\}$	$\{b, A, ret\}$

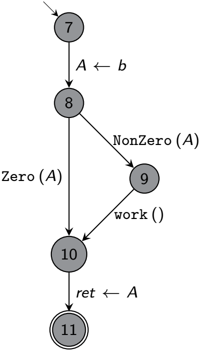
$$
\begin{aligned}
g_1(C) &= C \cup \{A\} \\
g_2(C) &= (A \in C) ? (C \cup \{ret\}) : (C \backslash \{ret\})
\end{aligned}
$$

Abstract Effects of Function `f`

ex. Copy Propagation

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

`work`():



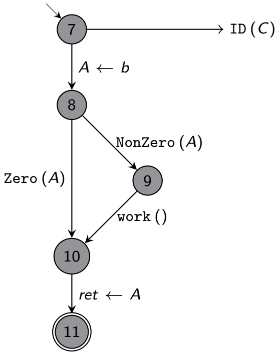
$$[\![A \leftarrow b]\!]^{\#} C = C \cup \{A\} := g_1 (C)$$
$$[\![\text{ret} \leftarrow A]\!]^{\#} C = (A \in C) ? (C \cup \{ret\}) : (C \backslash \{ret\}) := g_2 (C)$$

Abstract Effects of Function f

ex. Copy Propagation

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

work ():



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

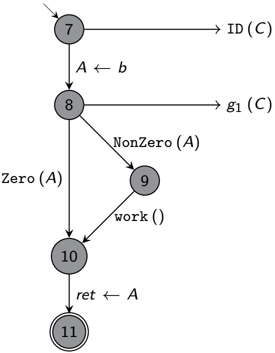
$[\![\text{ret} \leftarrow A]\!]^{\#} C = (A \in C)?\,(C \cup \{ret\}) : (C \backslash \{ret\}) := g_2(C)$

Abstract Effects of Function `f`

ex. Copy Propagation

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

`work` ():



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

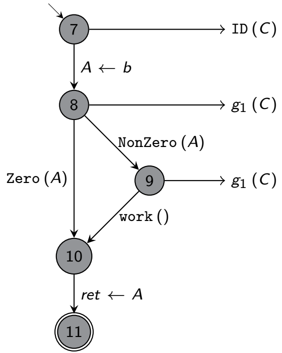
$[\![\mathrm{ret} \leftarrow A]\!]^{\#} C = (A \in C) ? (C \cup \{ret\}) : (C \backslash \{ret\}) := g_2(C)$

Abstract Effects of Function f

ex. Copy Propagation

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

work ():



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

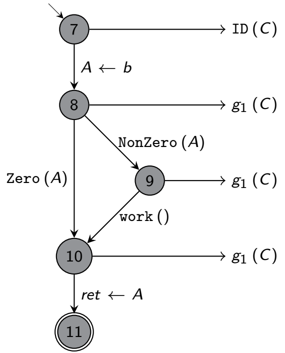
$[\![\text{ret} \leftarrow A]\!]^{\#} C = (A \in C)?\,(C \cup \{ret\}) : (C \backslash \{ret\}) := g_2(C)$

Abstract Effects of Function `f`

ex. Copy Propagation

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

`work` ():



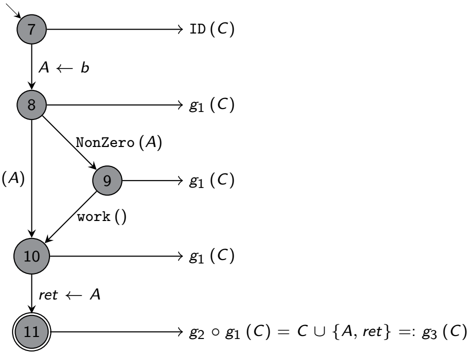
$$[\![A \leftarrow b]\!]^{\#} C = C \cup \{A\} := g_1(C)$$
$$[\![\text{ret} \leftarrow A]\!]^{\#} C = (A \in C)?\,(C \cup \{ret\}) : (C \backslash \{ret\}) := g_2(C)$$

Abstract Effects of Function f

ex. Copy Propagation

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

work ():



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

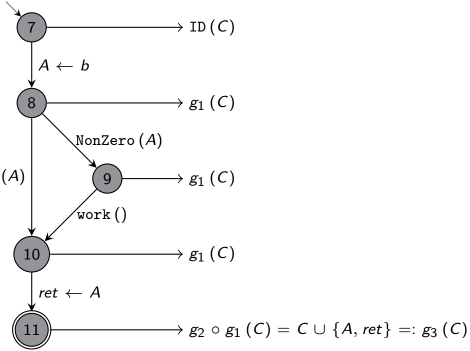
$$[\![\text{ret} \leftarrow A]\!]^{\#} C = (A \in C)?\,(C \cup \{ret\}) : (C \backslash \{ret\}) := g_2(C)$$

Abstract Effects of Function `f`

ex. Copy Propagation

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

`work` ():



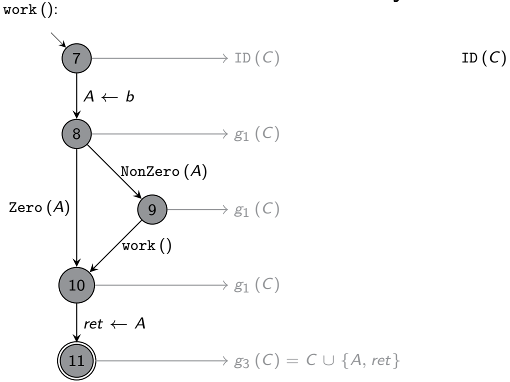
first approximation for call of `work`:

$\texttt{combine}^{\#}\left(C, g_3\left(\texttt{enter}^{\#}(C)\right)\right) = C \cup \{ret\} := g_4(C)$

Abstract Effects of Function `f`

ex. Copy Propagation

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



first approximation for call of `work`:

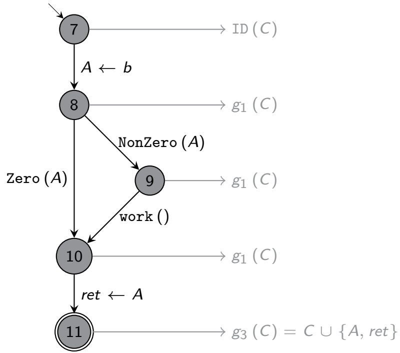
$$\texttt{combine}^{\#}\left(C, g_3\left(\texttt{enter}^{\#}(C)\right)\right) = C \cup \{ret\} := g_4(C)$$

Abstract Effects of Function `f`

ex. Copy Propagation

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

`work ()`:



$\texttt{ID}(C)$

$g_1(C)$

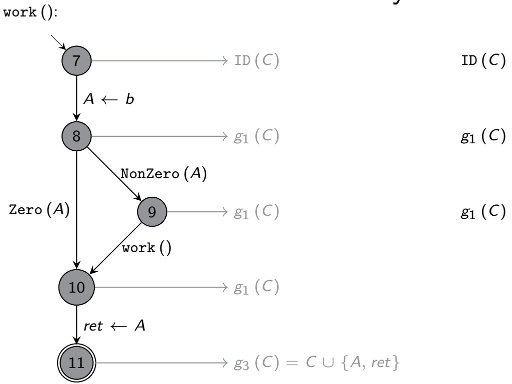
first approximation for call of `work`:

$$\texttt{combine}^{\#}\left(C, g_3\left(\texttt{enter}^{\#}(C)\right)\right) = C \cup \{ret\} := g_4(C)$$

Abstract Effects of Function f

ex. Copy Propagation

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



first approximation for call of work:

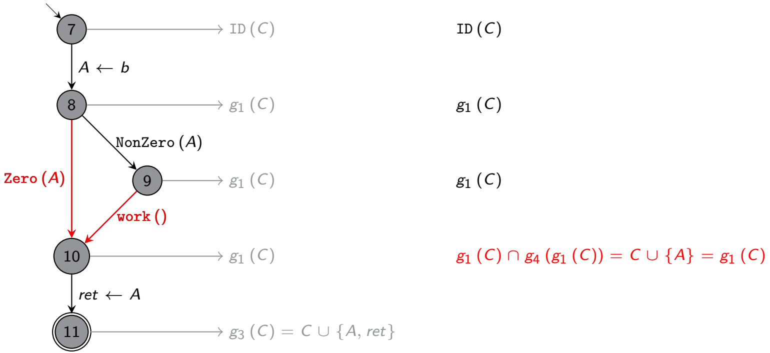
$$\texttt{combine}^{\#}\left(C, g_3\left(\texttt{enter}^{\#}(C)\right)\right) = C \cup \{ret\} := g_4(C)$$

Abstract Effects of Function f

ex. Copy Propagation

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

work ():



first approximation for call of work:

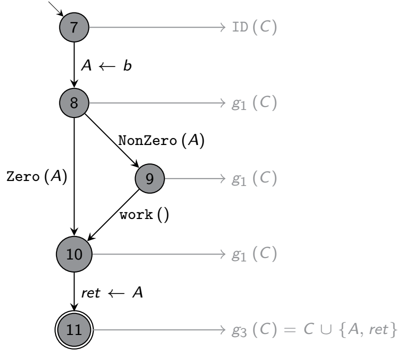
$$\texttt{combine}^{\#}\left(C, g_3\left(\texttt{enter}^{\#}(C)\right)\right) = C \cup \{ret\} := g_4(C)$$

Abstract Effects of Function `f`

ex. Copy Propagation

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

`work ()`:



$\text{ID}(C)$

$g_1(C)$

$g_1(C)$

$g_1(C) \cap g_4(g_1(C)) = C \cup \{A\} = g_1(C)$

$g_2 \circ g_1(C) = C \cup \{A, ret\} =: g_3(C)$

fixpoint reached after first iteration:
`work` approximated by $g_4(C) = C \cup \{ret\}$

Coincidence Theoreme

- $\exists$ same-level computation from $start_{\mathtt{f}}$ to v $\forall v \in \mathtt{f}$, edge effects and transformation $H^{\#}$ are distributive

  $\Rightarrow [\![v]\!]^{\#} = \bigsqcup\{[\![\pi]\!]^{\#} | \pi \in \mathcal{T}_v\} \forall v \in \mathtt{f}$
  ($\mathcal{T}_v$...set of all same-level computations from $start_{\mathtt{f}}$ to v)

- $\mathtt{enter}^{\#}$ distributive, $\mathtt{combine}^{\#}(x_1, x_2) = h_1(x_1) \sqcup h_2(x_2)$
  $\Rightarrow H^{\#}$ distributive: $H^{\#}(\bigsqcup \mathcal{F}) = \bigsqcup\{H^{\#}(g) | g \in \mathcal{F}\}$

Coincidence Theoreme

ex. Copy Propagation

$\texttt{enter}^{\#}\, V = V \cap Glob \cup \{\bullet\}$
$\rightarrow$ distributive

$$
\begin{aligned}
\texttt{combine}^{\#}\,(V_1, V_2) &= (V_2 \cap Glob) \cup (\bullet \in V_2)?\, V_1 \cap Loc : \emptyset \\
&= ((V_1 \cap Loc_{\bullet}) \cup Glob) \cap \\
&\quad ((V_2 \cap Glob) \cup Loc_{\bullet}) \cap \\
&\quad (Glob \cup (\bullet \in V_2)?\, Vars_{\bullet} : Glob)
\end{aligned}
$$

$\rightarrow$ intersection of distributive functions of first and second argument

$\Rightarrow$ coincidence theoreme holds for copy propagation

Interprocedural Reachability

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

$$
\begin{aligned}
\mathcal{D}[start_{\texttt{main}}] &\sqsupseteq \texttt{enter}^\#(d_0) \\
\mathcal{D}[start_{\texttt{f}}] &\sqsupseteq \texttt{enter}^\#(\mathcal{D}[u])\,, \\
&\quad (u, \texttt{f}\,()\,, v) \text{ calling edge} \\
\mathcal{D}[v] &\sqsupseteq \texttt{combine}^\#\left(\mathcal{D}[u], [\![\texttt{f}]\!]^\#\left(\texttt{enter}^\#(\mathcal{D}[u])\right)\right), \\
&\quad (u, \texttt{f}\,()\,, v) \text{ calling edge} \\
\mathcal{D}[v] &\sqsupseteq [\![k]\!]^\#(\mathcal{D}[u])\,, \\
&\quad k = (u, lab, v) \text{ normal edge}
\end{aligned}
$$

Interprocedural Reachability

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

$$\mathcal{D}[v] \sqsupseteq [\![\pi]\!]^{\#} d_0$$

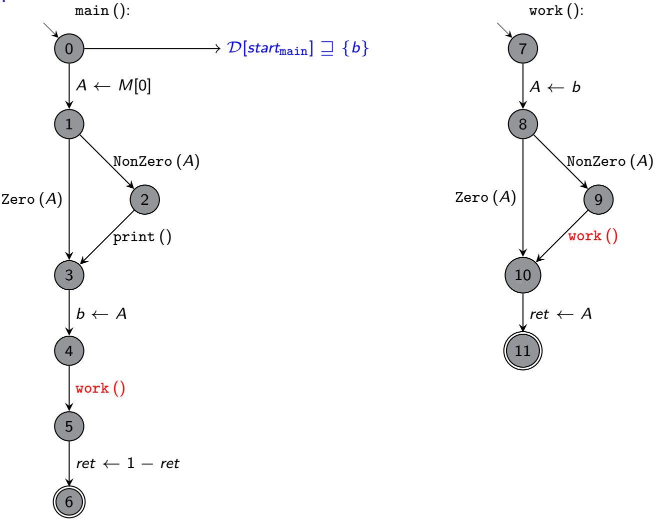
for all paths that reach v
($d_0 \in \mathbb{D}$: information at the beginning of program execution)
for distributive abstract edge effects and distributive transformation $H^{\#}$:

$$\mathcal{D}[v] = \bigsqcup \{[\![\pi]\!]^{\#} d_0 | \pi \in \mathcal{P}_v\}$$

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

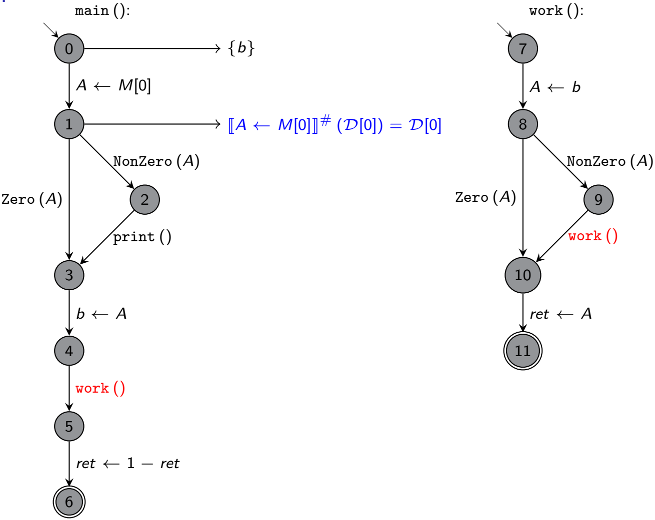
Interprocedural Reachability

example



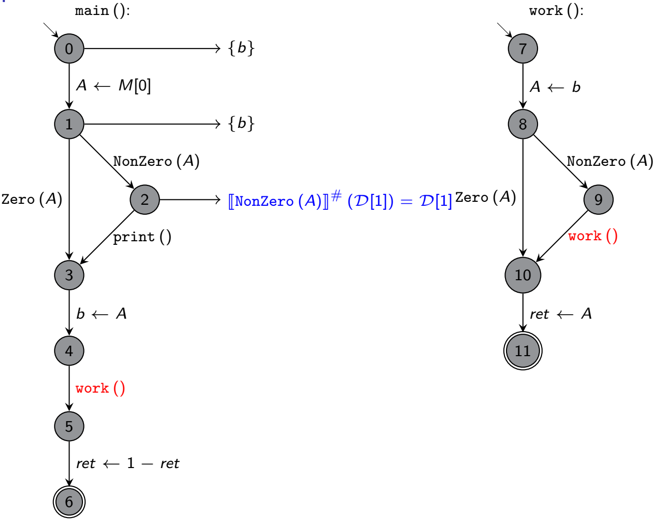
Interprocedural Reachability

example



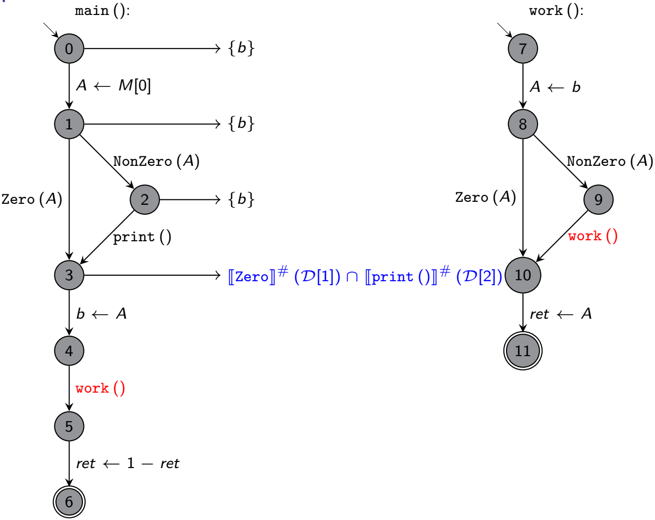
Interprocedural Reachability

example



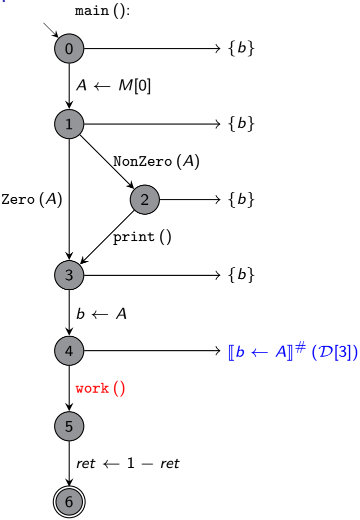
Interprocedural Reachability

example



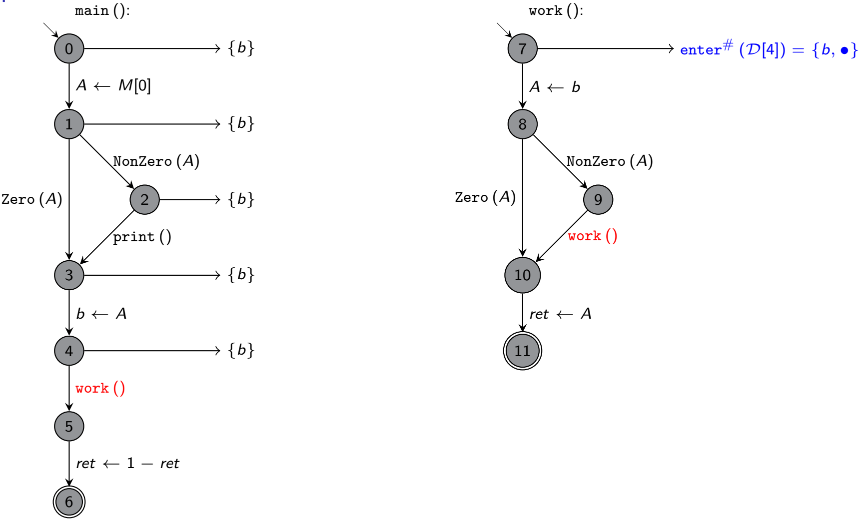
Interprocedural Reachability

example



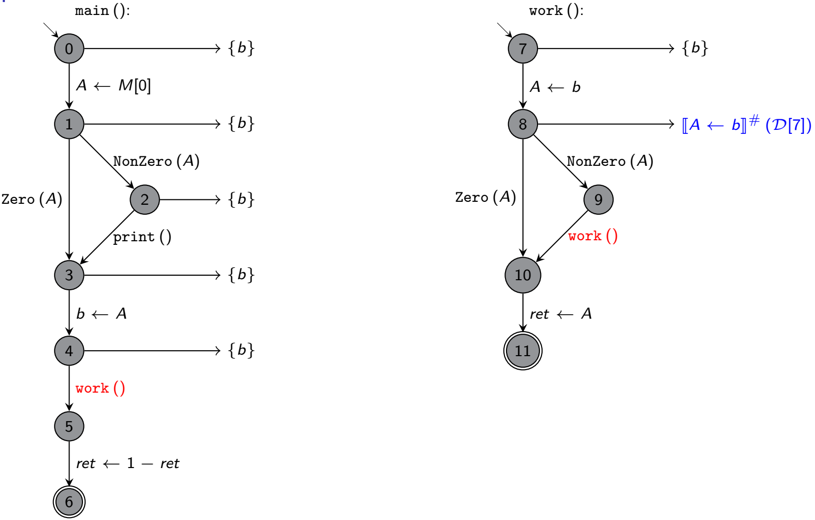
Interprocedural Reachability

example



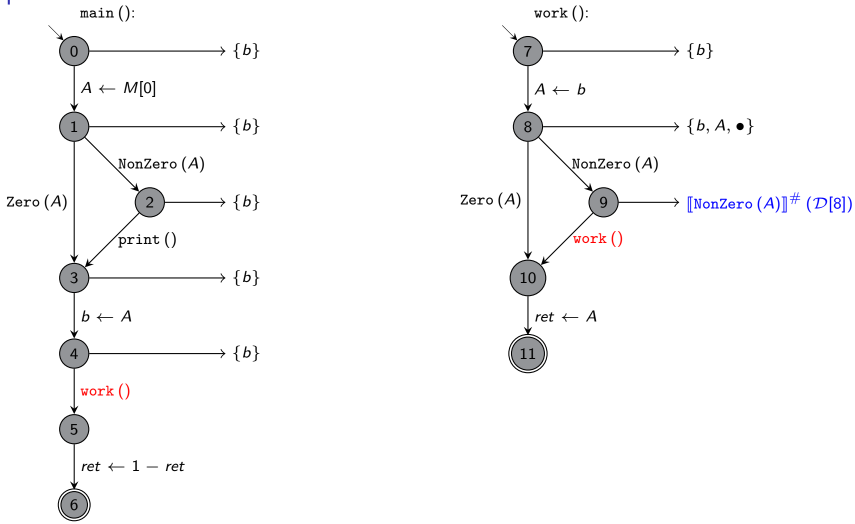
Interprocedural Reachability

example



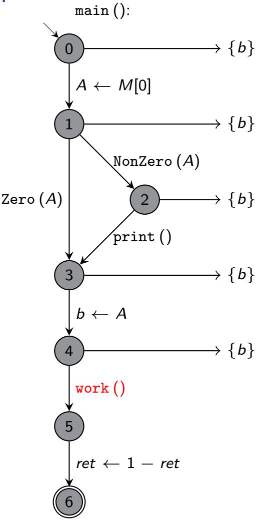
Interprocedural Reachability

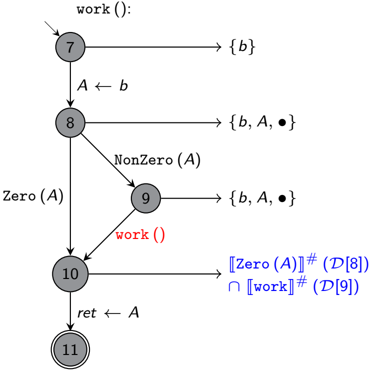
example



Interprocedural Reachability

example

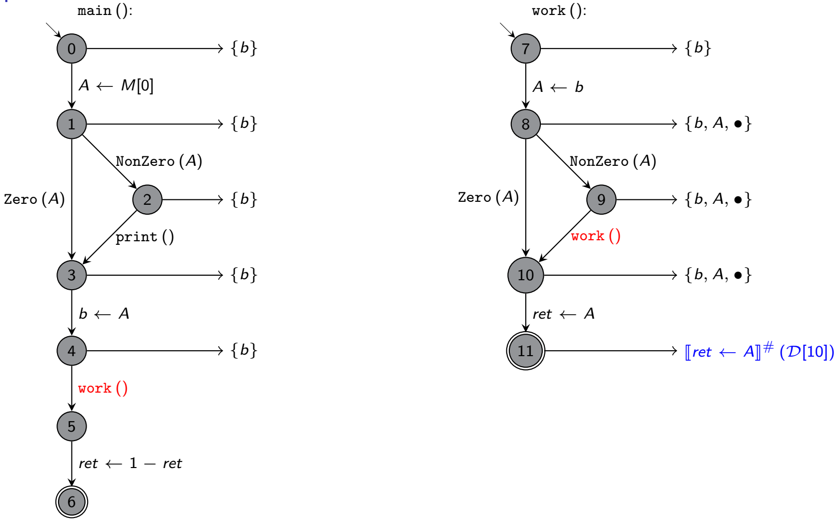




work approximated by $g_4(C) = C \cup \{ret\}$

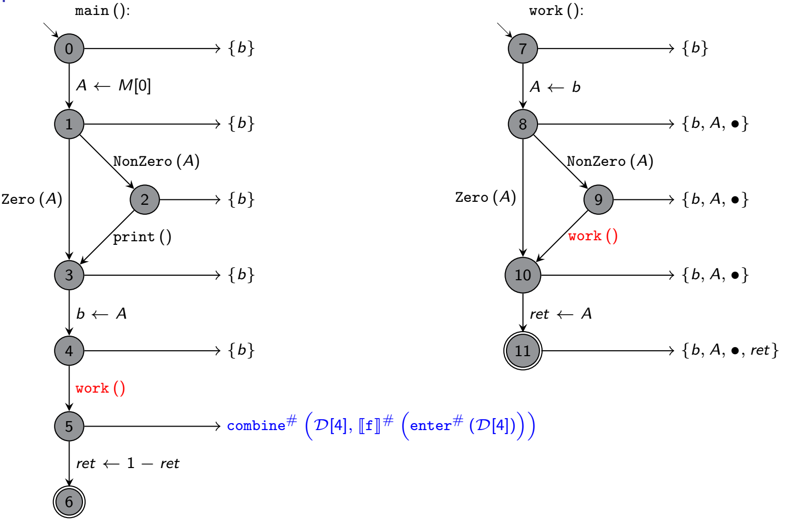
Interprocedural Reachability

example



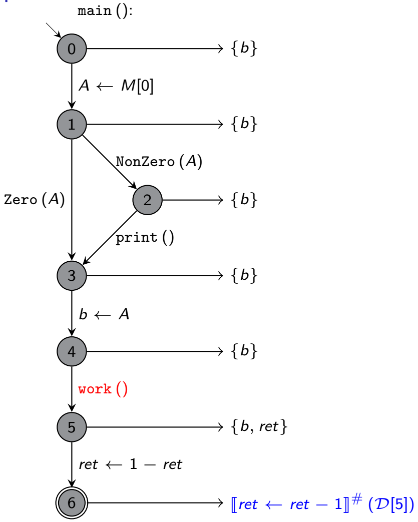
Interprocedural Reachability

example



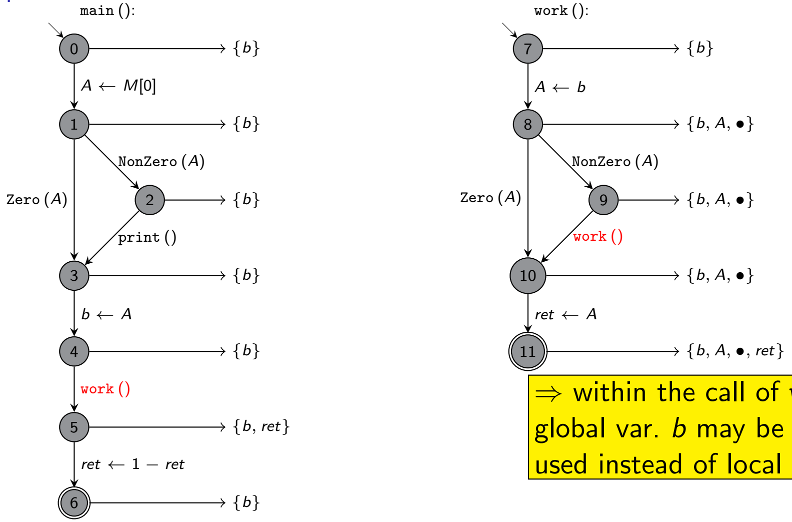
Interprocedural Reachability

example



Interprocedural Reachability

example



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

Demand-Driven Interprocedural Analysis

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

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

$\rightarrow$ 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

$$\begin{array}{lll}
\mathcal{D}[v, a] & \sqsupseteq & a, \\
 & & v \text{ entry point} \\
\mathcal{D}[v, a] & \sqsupseteq & \texttt{combine}^{\#}\left(\mathcal{D}[u, a], \mathcal{D}[\texttt{f}, \texttt{enter}^{\#}\left(\mathcal{D}[u, a]\right)]\right), \\
 & & \left(u, \texttt{f}\left(\right), v\right) \text{ calling edge} \\
\mathcal{D}[v, a] & \sqsupseteq & [\![lab]\!]^{\#}\left(\mathcal{D}[u, a]\right), \\
 & & k = \left(u, lab, v\right) \text{ normal edge} \\
\mathcal{D}[\texttt{f}, a] & \sqsupseteq & \mathcal{D}[stop_{\texttt{f}}, a]
\end{array}$$

with $\mathcal{D}[\texttt{f}, a]$ ... abstract state when reaching program point v of a function called in abstract state a ($\mathcal{D}[\texttt{f}, a] \sim [\![v]\!]^{\#}(a)$)

$\Rightarrow$ compute $\mathcal{D}[\texttt{main}, \texttt{enter}^{\#}(d_0)]$

Demand-Driven Interprocedural Analysis

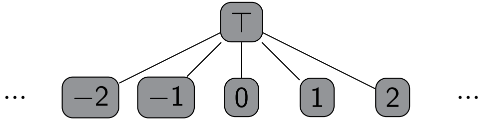
ex. Constant Propagation

Constant Propagation:

move as many computations as possible from runtime to compile time

complete lattice: $\mathbb{D} = \left(Vars \to \mathbb{Z}^\top\right)_\bot$

$\to$! not finite



$$\texttt{enter}^\# D = \begin{cases} \bot & D = \bot \\ D \oplus \{A \mapsto \top \mid A \text{ local}\} & \text{otherwise} \end{cases}$$

$$\texttt{combine}^\# (D_1, D_2) = \begin{cases} \bot & D_1 = \bot \vee D_2 = \bot \\ D_1 \oplus \{b \mapsto D_2(b) \mid b \text{ global}\} & \text{otherwise} \end{cases}$$

Constant Propagation

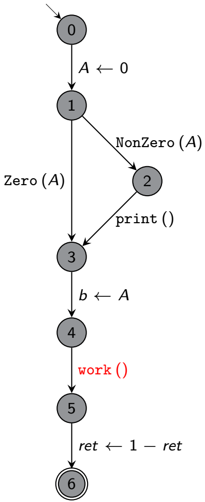
Abstract Edge Effects - intraprocedural

$$
\begin{aligned}
[\![;]\!]^{\#} D &= D \\
[\![\texttt{NonZero}\,(e)]\!]^{\#} D &= \begin{cases} \bot & \text{if } 0 = [\![e]\!]^{\#} D \\ D & \text{otherwise} \end{cases} \\
[\![\texttt{Zero}\,(e)]\!]^{\#} D &= \begin{cases} \bot & \text{if } 0 \not\sqsubseteq [\![e]\!]^{\#} D \\ D & \text{if } 0 \sqsubseteq [\![e]\!]^{\#} D \end{cases} \\
[\![x \leftarrow e]\!]^{\#} D &= D \oplus \{ x \mapsto [\![e]\!]^{\#} D \} \\
[\![x \leftarrow M[e]]\!]^{\#} D &= D \oplus \{ x \mapsto \top \} \\
[\![M[e_1] \leftarrow e_2]\!]^{\#} D &= D
\end{aligned}
$$

Demand-Driven Interprocedural Analysis

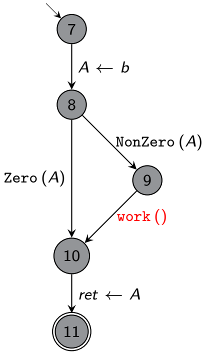
ex. Constant Propagation

$d_0 = \{A \mapsto \top, b \mapsto \top, ret \mapsto \top\}$
main ():



$d_1 = \{A \mapsto \top, b \mapsto 0, ret \mapsto \top\}$

work ():



	A	b	ret
$0, d_0$	$\top$	$\top$	$\top$
$1, d_0$	0	$\top$	$\top$
$2, d_0$		$\bot$	
$3, d_0$	0	$\top$	$\top$
$4, d_0$	0	0	$\top$
$7, d_1$	$\top$	0	$\top$
$8, d_1$	0	0	$\top$
$9, d_1$		$\bot$	
$10, d_1$	0	0	$\top$
$11, d_1$	0	0	0
$5, d_0$	0	0	0
$6, d_0$	0	0	1
main, d_0	0	0	1

Call-String-Approach

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

here: call stack of depth 0
$\rightarrow$ function call as unconditional jump

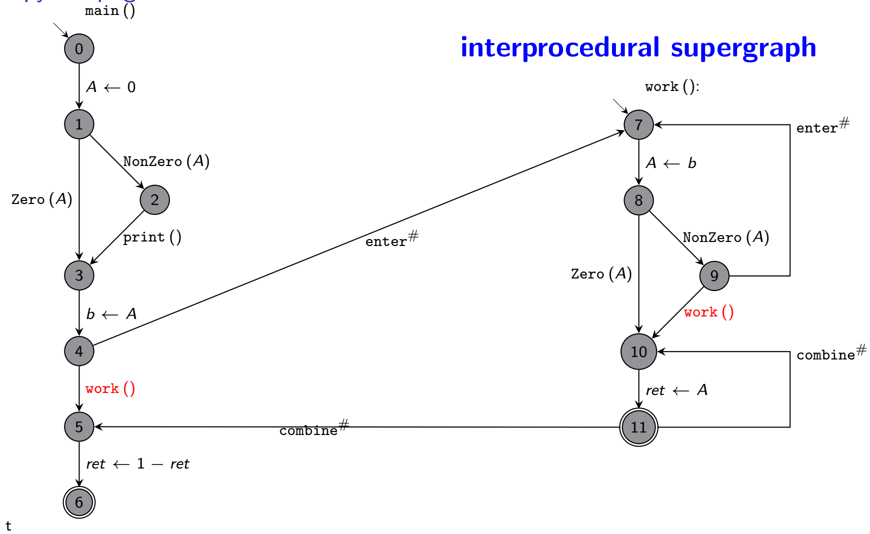
Call-String-Approach

system of inequalities

$$
\begin{array}{rcl}
\mathcal{D}[start_{\texttt{main}}] & \sqsupseteq & \text{enter}^{\#}\,(d_0) \\
\mathcal{D}[start_{\texttt{f}}] & \sqsupseteq & \text{enter}^{\#}\,(\mathcal{D}[u])\,, \\
 & & (u, \texttt{f}\,()\,, v)\;\; \text{calling edge} \\
\mathcal{D}[v] & \sqsupseteq & \texttt{combine}^{\#}\,(\mathcal{D}[u], \mathcal{D}[v])\,, \\
 & & (u, \texttt{f}\,()\,, v)\;\; \text{calling edge} \\
\mathcal{D}[v] & \sqsupseteq & [\![lab]\!]^{\#}\,(\mathcal{D}[u])\,, \\
 & & k = (u, lab, v)\;\; \text{normal edge} \\
\mathcal{D}[\texttt{f}] & \sqsupseteq & \mathcal{D}[stop_{\texttt{f}}]
\end{array}
$$

Call-String-Approach

ex. Copy Propagation



Call-String-Approach

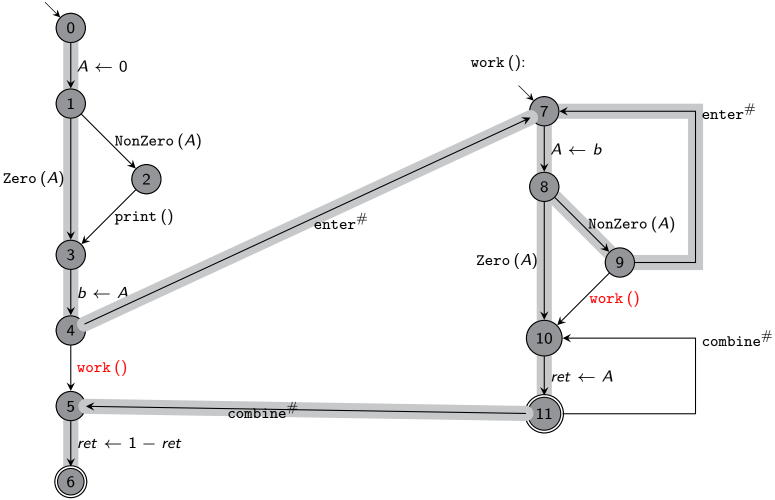
ex. Copy Propagation

$$
\begin{aligned}
\mathcal{D}[5] &\sqsupseteq \texttt{combine}^{\#}(\mathcal{D}[4], \mathcal{D}[\texttt{work}]) \\
\mathcal{D}[7] &\sqsupseteq \texttt{enter}^{\#}(\mathcal{D}[4]) \\
\mathcal{D}[7] &\sqsupseteq \texttt{enter}^{\#}(\mathcal{D}[9]) \\
\mathcal{D}[10] &\sqsupseteq \texttt{combine}^{\#}(\mathcal{D}[9], \mathcal{D}[\texttt{work}])
\end{aligned}
$$

Call-String-Approach

ex. Copy Propagation

! for depth 0: impossible paths may occur



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 fullfill certain properties to ensure that the analysis terminates