Interprocedural Data Flow Analysis

Uday Khedker

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

October 2017

Part 1

About These Slides

Copyright

These slides constitute the lecture notes for CS618 Program Analysis course at IIT Bombay and have been made available as teaching material accompanying the book:

- Uday Khedker, Amitabha Sanyal, and Bageshri Karkare.

  (Indian edition published by Ane Books in 2013) *Data Flow Analysis: Theory and Practice*. CRC Press (Taylor and Francis Group). 2009.

Apart from the above book, some slides are based on the material from the following books

- S. S. Muchnick and N. D. Jones. *Program Flow Analysis*. Prentice Hall Inc. 1981.

Outline

- Issues in interprocedural analysis
- Functional approach
- Classical call strings approach
- Value context based approach

Part 2

Issues in Interprocedural Analysis

Interprocedural Analysis: Overview

- Extends the scope of data flow analysis across procedure boundaries

  Incorporates the effects of

  - procedure calls in the caller procedures, and
  - calling contexts in the callee procedures

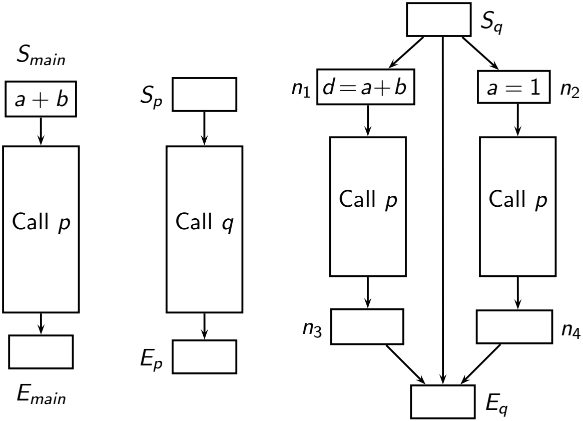
- Approaches :

  - Generic : Call strings approach, functional approach
  - Problem specific : Alias analysis, Points-to analysis, Partial redundancy elimination, Constant propagation

Why Interprocedural Analysis?

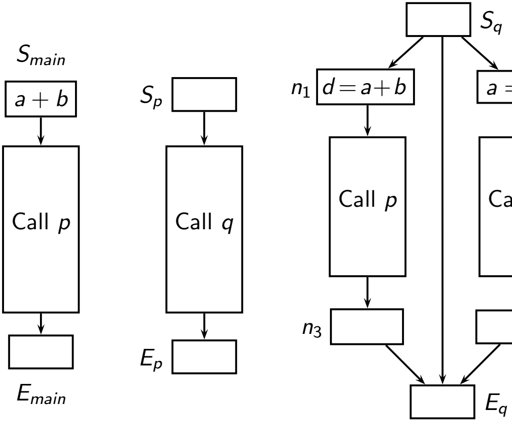
- Answering questions about formal parameters and global variables:
  - Which variables are constant?
  - Which variables aliased with each other?
  - Which locations can a pointer variable point to?
- Answering questions about side effects of a procedure call:
  - Which variables are defined or used by a called procedure? (Could be local/global/formal variables)
- Most of the above questions may have a *May* or *Must* qualifier

Program Representation for Interprocedural Data Flow Analysis: Call Multi-Graph



Supergraphs of procedures

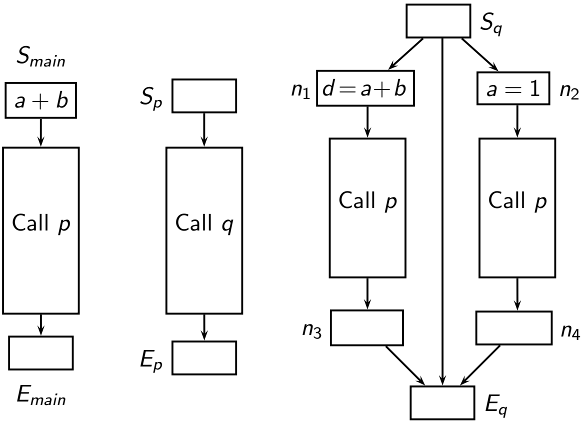
Program Representation for Interprocedural Data Flow Analysis: Call Multi-Graph



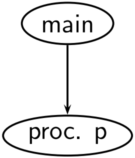
Supergraphs of procedures

Call multi-graph

Program Representation for Interprocedural Data Flow Analysis: Call Multi-Graph

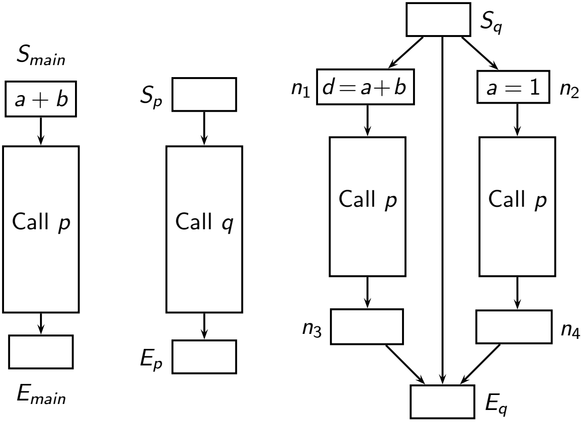


Supergraphs of procedures

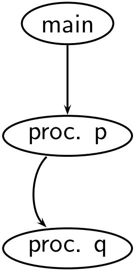


Call multi-graph

Program Representation for Interprocedural Data Flow Analysis: Call Multi-Graph

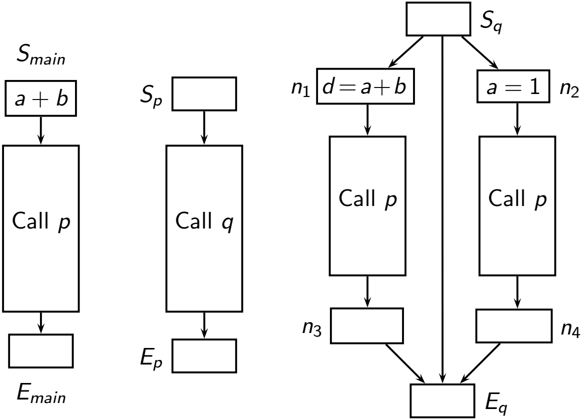


Supergraphs of procedures

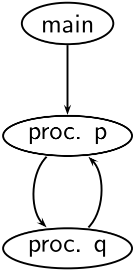


Call multi-graph

Program Representation for Interprocedural Data Flow Analysis: Call Multi-Graph

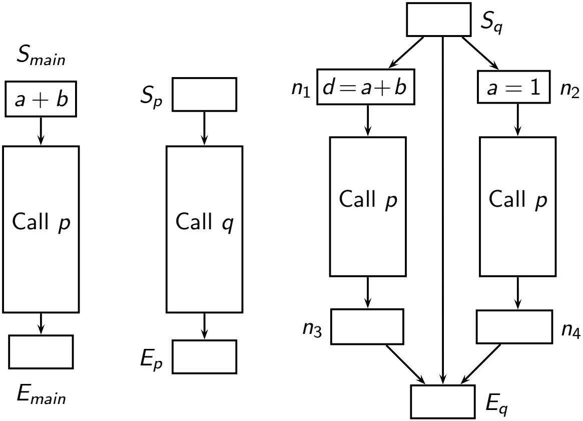


Supergraphs of procedures

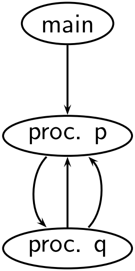


Call multi-graph

Program Representation for Interprocedural Data Flow Analysis: Call Multi-Graph

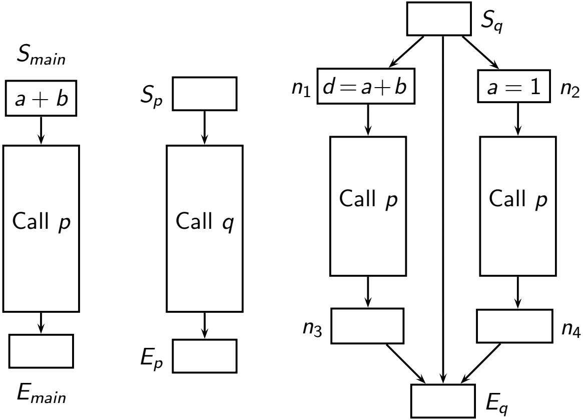


Supergraphs of procedures

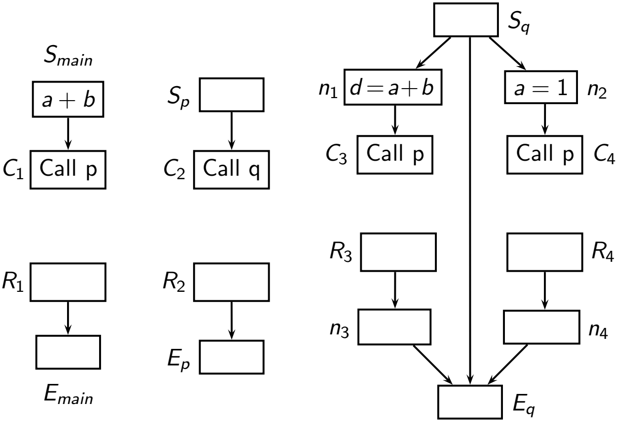


Call multi-graph

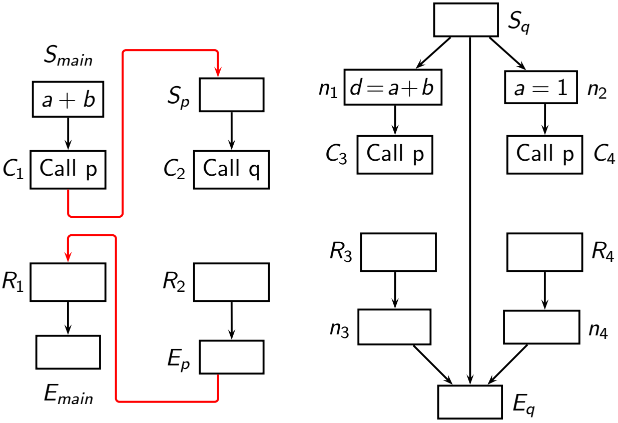
Program Representation for Interprocedural Data Flow Analysis: Supergraph



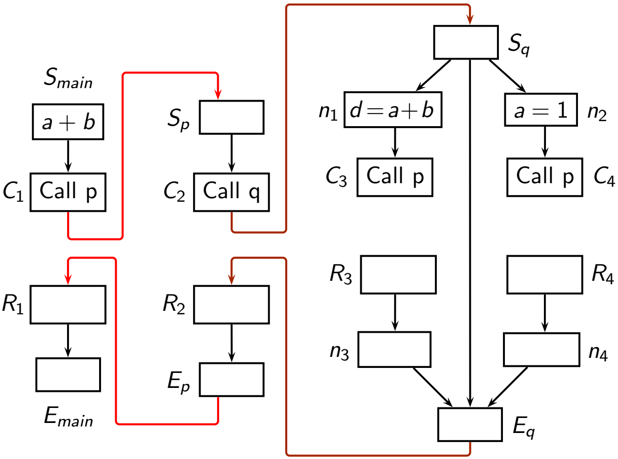
Program Representation for Interprocedural Data Flow Analysis: Supergraph



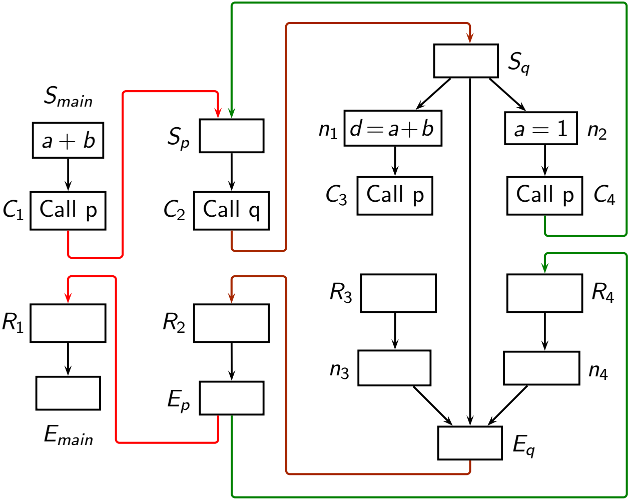
Program Representation for Interprocedural Data Flow Analysis: Supergraph



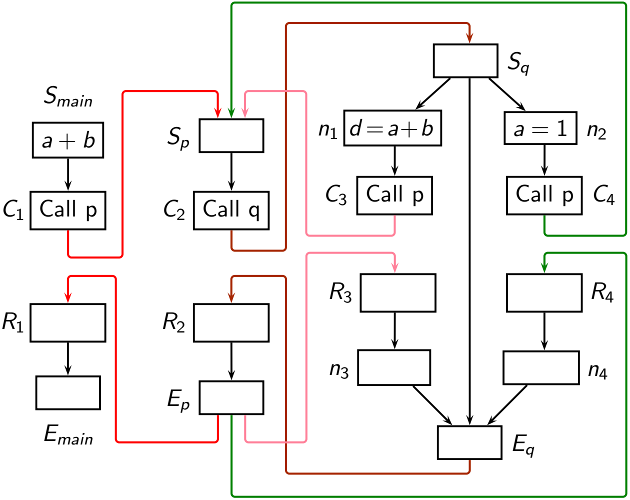
Program Representation for Interprocedural Data Flow Analysis: Supergraph



Program Representation for Interprocedural Data Flow Analysis: Supergraph



Program Representation for Interprocedural Data Flow Analysis: Supergraph

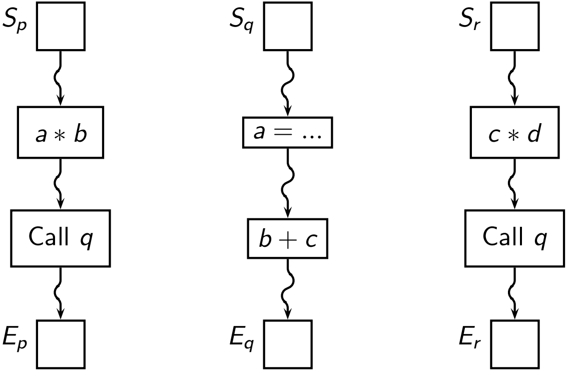


Top-down Vs. Bottom-up Interprocedural Analysis

- Bottom-up approach
  - Traverses the call graph bottom up
  - Computes a parameterized summary of each callee
  - Can be viewed as procedure inlining
    Summary is inlined at the all site, not the entire procedure body
- Top-down approach
  - Traverses the call graph top down
  - Needs to visit a procedure separately for every calling context
  - Can be viewed as procedure inlining

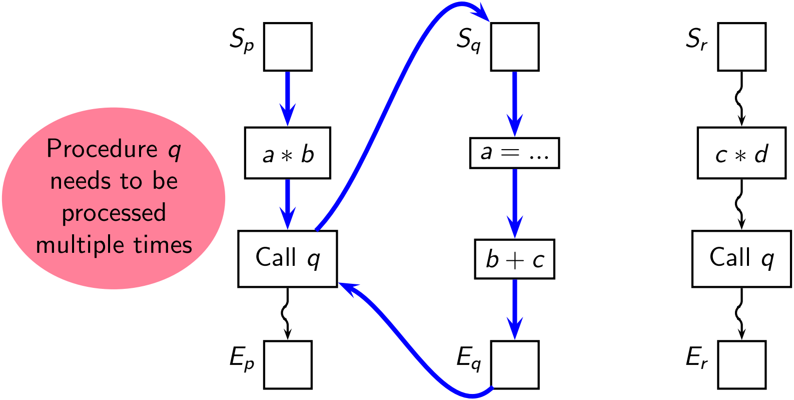
Top-down Vs. Bottom-up Interprocedural Analysis

Top-down Analysis for Available Expressions Analysis



Top-down Vs. Bottom-up Interprocedural Analysis

Top-down Analysis for Available Expressions Analysis



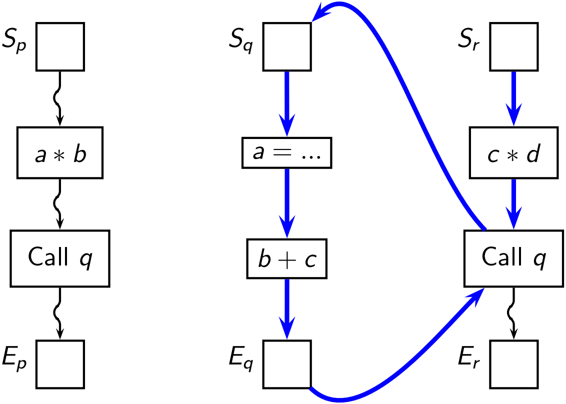
Expression $b + c$ is available in procedure p

Expression $a * b$ is not available in procedure p

Top-down Vs. Bottom-up Interprocedural Analysis

Top-down Analysis for Available Expressions Analysis

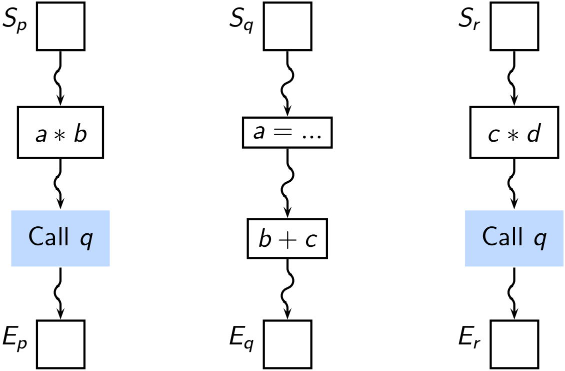
Procedure q needs to be processed multiple times



Expressions $b + c$ and $c * d$ are available in procedure r

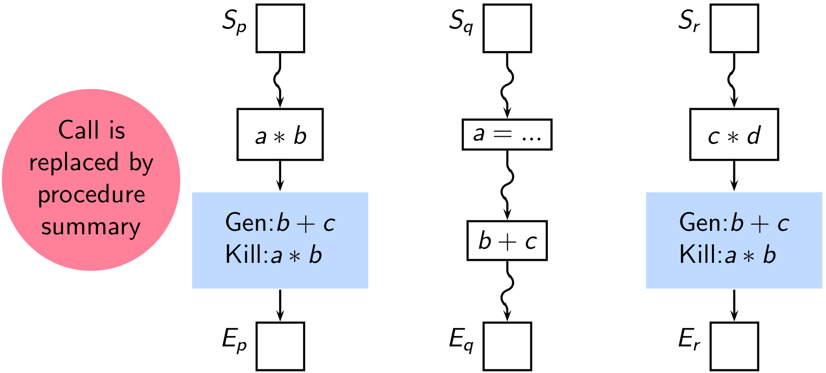
Top-down Vs. Bottom-up Interprocedural Analysis

Bottom-Up Analysis for Available Expressions Analysis



Top-down Vs. Bottom-up Interprocedural Analysis

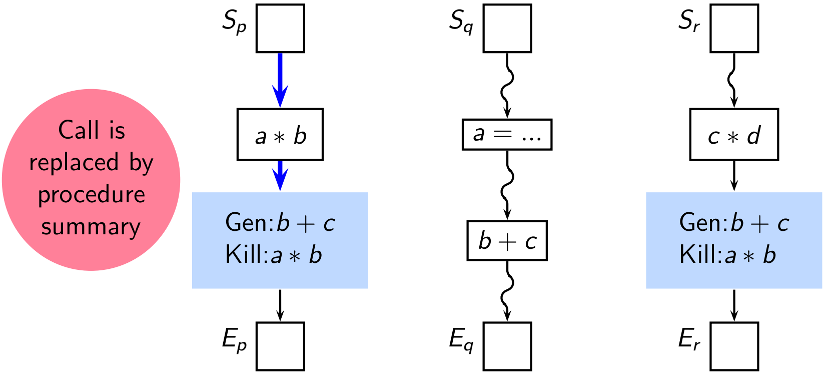
Bottom-Up Analysis for Available Expressions Analysis



Using procedure summary of g at call sites

Top-down Vs. Bottom-up Interprocedural Analysis

Bottom-Up Analysis for Available Expressions Analysis

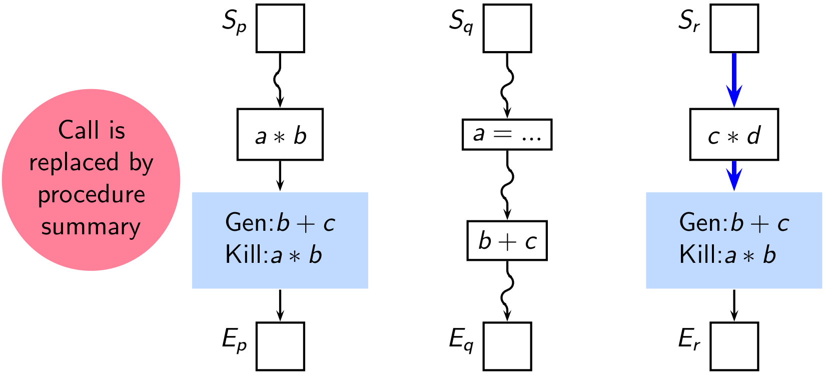


Expression $b + c$ is available in procedure p

Expression $a * b$ is not available in procedure p

Top-down Vs. Bottom-up Interprocedural Analysis

Bottom-Up Analysis for Available Expressions Analysis

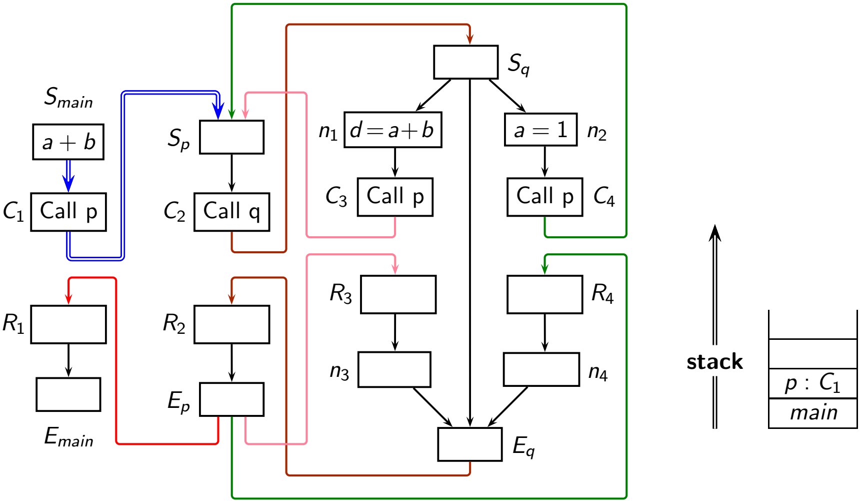


Expressions $b + c$ and $c * d$ are available in procedure r

Issues in Top-down Vs. Bottom-up Interprocedural Analysis

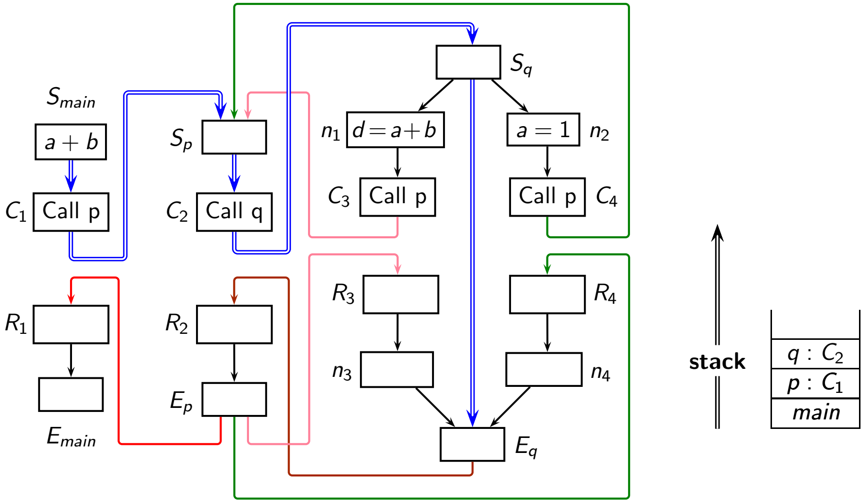
- Bottom-up approach
  - Compact representation
  - Information may depend on the calling context
- Top-down approach
  - Exponentially large number of calling contexts
  - Many contexts may have no effect on the procedure

Validity of Interprocedural Control Flow Paths



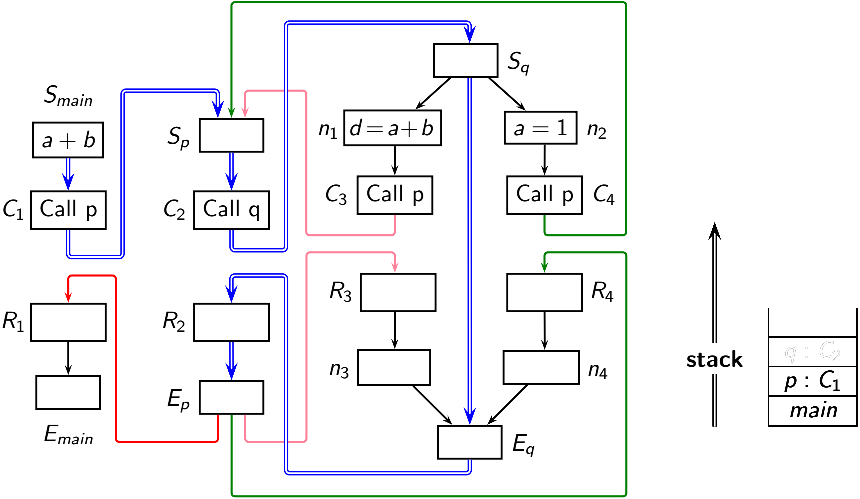
Interprocedurally valid control flow path

Validity of Interprocedural Control Flow Paths



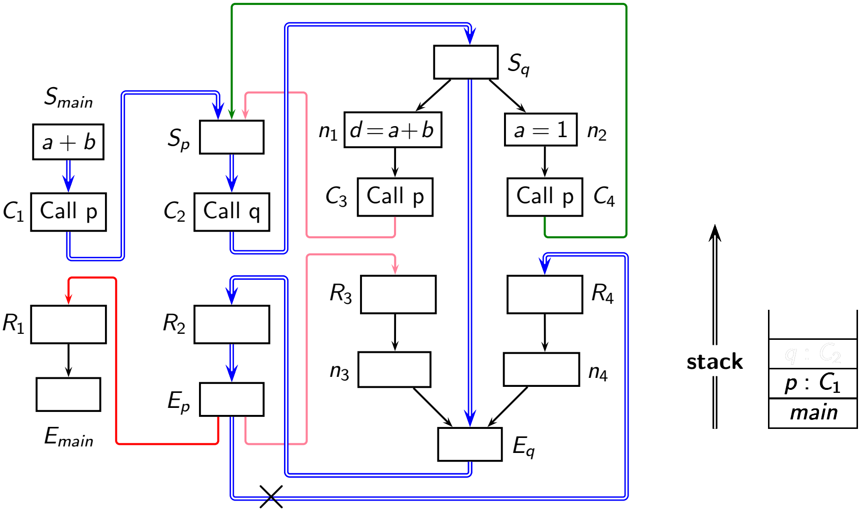
Interprocedurally valid control flow path

Validity of Interprocedural Control Flow Paths



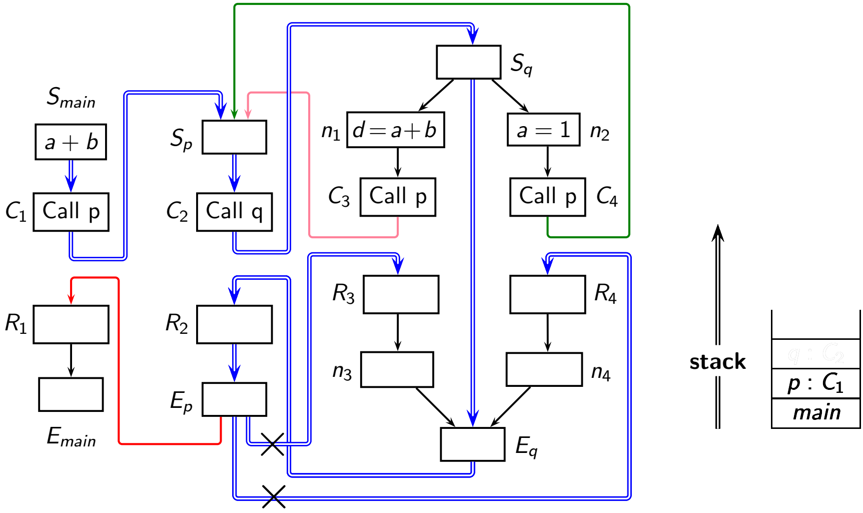
Interprocedurally valid control flow path

Validity of Interprocedural Control Flow Paths



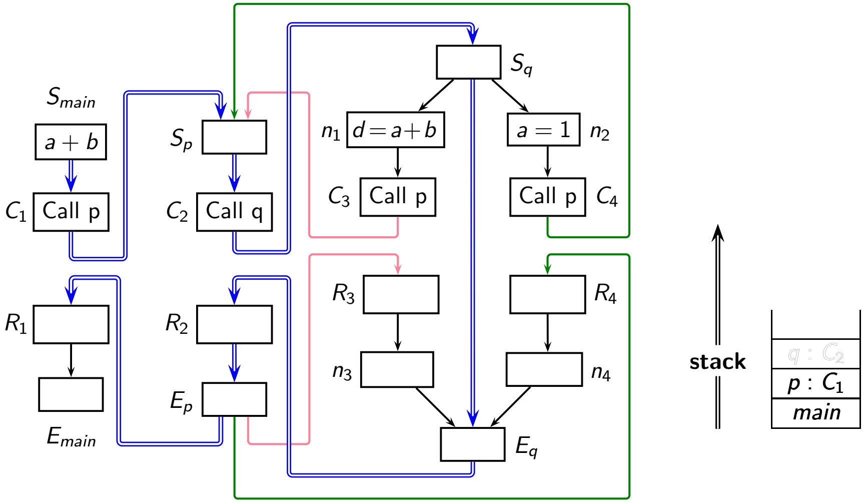
Interprocedurally invalid control flow path

Validity of Interprocedural Control Flow Paths



Interprocedurally invalid control flow path

Validity of Interprocedural Control Flow Paths



Interprocedurally valid control flow path

Soundness, Precision, and Efficiency of Data Flow Analysis

- Data flow analysis uses static representation of programs to compute summary information along paths

Soundness, Precision, and Efficiency of Data Flow Analysis

- Data flow analysis uses static representation of programs to compute summary information along paths
- *Ensuring Soundness*. All valid paths must be covered

Soundness, Precision, and Efficiency of Data Flow Analysis

A path which represents legal control flow

- Data flow analysis uses static representation of programs to compute summary information along paths
- *Ensuring Soundness*. All valid paths must be covered

Soundness, Precision, and Efficiency of Data Flow Analysis

A path which represents legal control flow

- Data flow analysis uses static representation of programs to compute summary information along paths
- *Ensuring Soundness*. All valid paths must be covered
- *Ensuring Precision*. Only valid paths should be covered

Soundness, Precision, and Efficiency of Data Flow Analysis

A path which represents legal control flow

- Data flow analysis uses static representation of programs to compute summary information along paths
- *Ensuring Soundness*. All valid paths must be covered
- *Ensuring Precision*. Only valid paths should be covered

Subject to merging data flow values at shared program points without creating invalid paths

Soundness, Precision, and Efficiency of Data Flow Analysis

A path which represents legal control flow

- Data flow analysis uses static representation of programs to compute summary information along paths
- *Ensuring Soundness*. All valid paths must be covered
- *Ensuring Precision*. Only valid paths should be covered
- *Ensuring Efficiency*. Only relevant valid paths should be covered

Subject to merging data flow values at shared program points without creating invalid paths

Soundness, Precision, and Efficiency of Data Flow Analysis

A path which represents legal control flow

- Data flow analysis uses static representation of programs to compute summary information along paths
- *Ensuring Soundness*. All valid paths must be covered
- *Ensuring Precision*. Only valid paths should be covered
- *Ensuring Efficiency*. Only relevant valid paths should be covered

Subject to merging data flow values at shared program points without creating invalid paths

A path which yields information that affects the summary information

Flow and Context Sensitivity

- Flow sensitive analysis:

  Considers intraprocedurally valid paths

Flow and Context Sensitivity

- Flow sensitive analysis:

  Considers intraprocedurally valid paths

- Context sensitive analysis:

  Considers interprocedurally valid paths

Flow and Context Sensitivity

- Flow sensitive analysis:
  Considers intraprocedurally valid paths
- Context sensitive analysis:
  Considers interprocedurally valid paths
- For maximum statically attainable precision , analysis must be both flow and context sensitive

Flow and Context Sensitivity

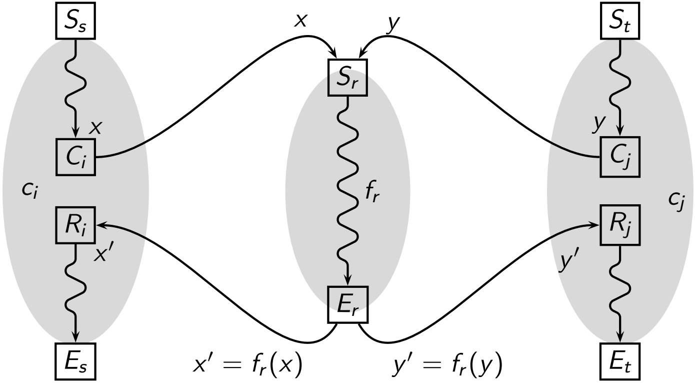
- Flow sensitive analysis:
  Considers intraprocedurally valid paths

- Context sensitive analysis:
  Considers interprocedurally valid paths

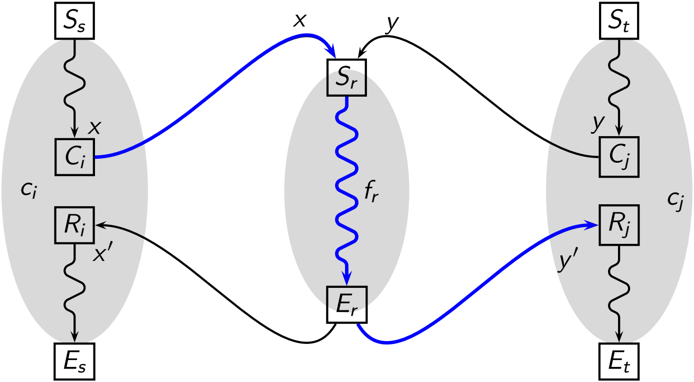
- For maximum statically attainable precision, analysis must be both flow and context sensitive

MFP computation restricted to valid paths only

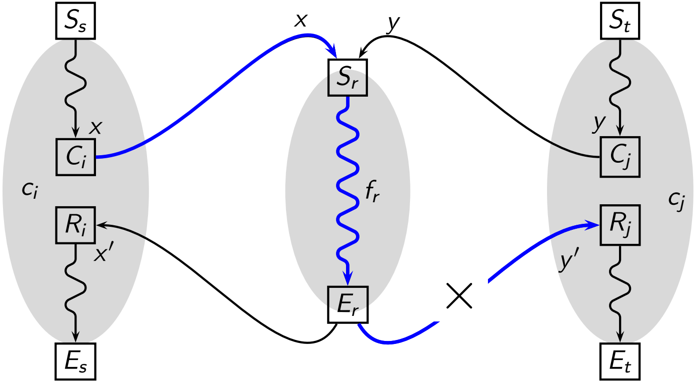
Context Sensitivity in Interprocedural Analysis



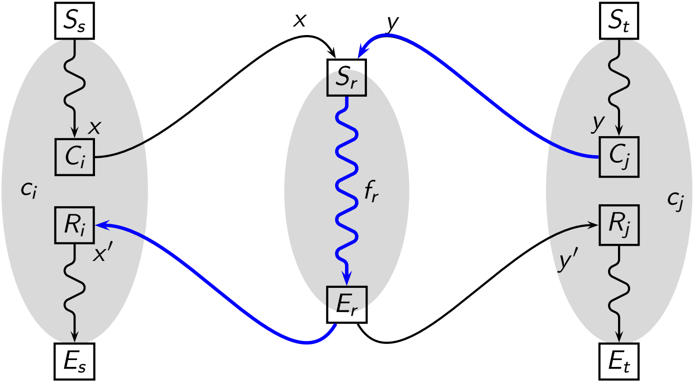
Context Sensitivity in Interprocedural Analysis



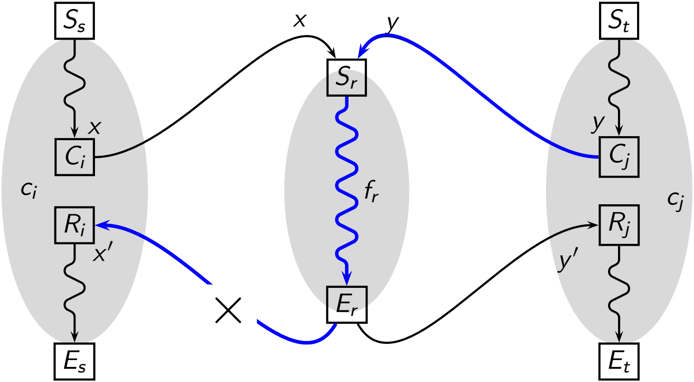
Context Sensitivity in Interprocedural Analysis



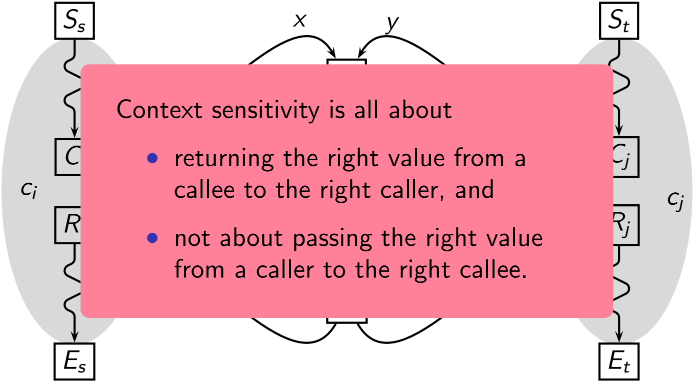
Context Sensitivity in Interprocedural Analysis



Context Sensitivity in Interprocedural Analysis



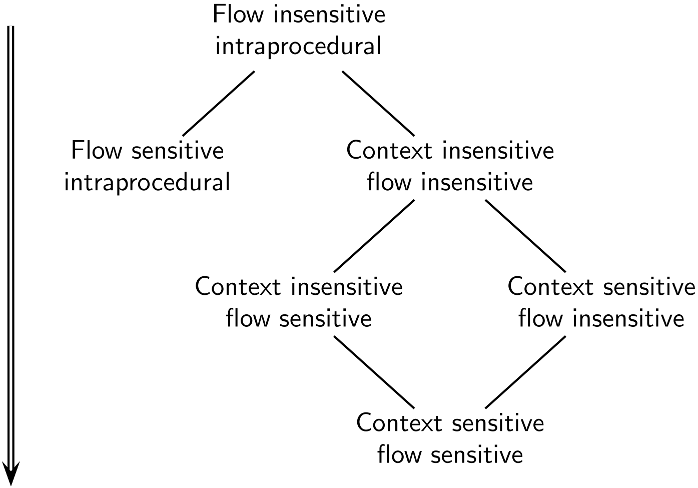
Context Sensitivity in Interprocedural Analysis



Context sensitivity is all about

- returning the right value from a callee to the right caller, and
- not about passing the right value from a caller to the right callee.

Increasing Precision in Data Flow Analysis



Flow insensitive intraprocedural

Flow sensitive intraprocedural

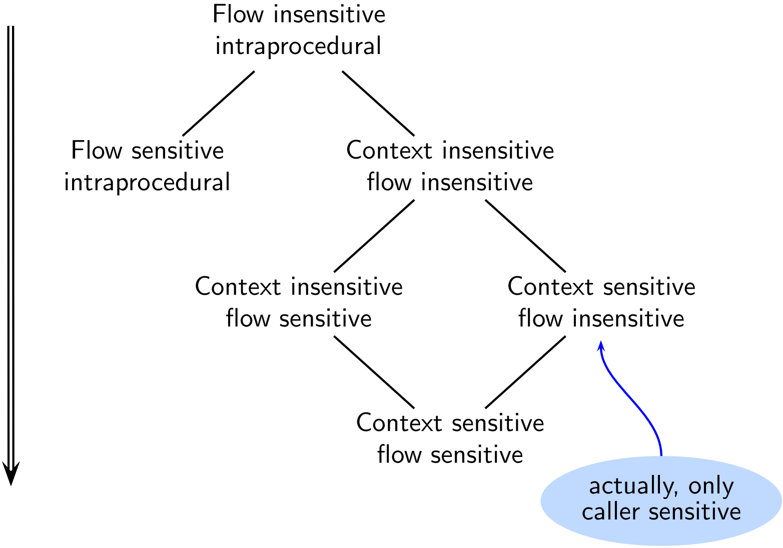
Context insensitive flow insensitive

Context insensitive flow sensitive

Context sensitive flow insensitive

Context sensitive flow sensitive

Increasing Precision in Data Flow Analysis



Flow insensitive intraprocedural

Flow sensitive intraprocedural

Context insensitive flow insensitive

Context insensitive flow sensitive

Context sensitive flow insensitive

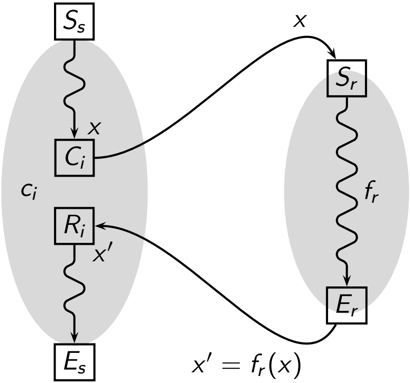
Context sensitive flow sensitive

actually, only caller sensitive

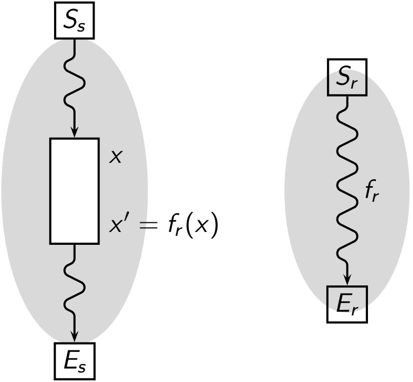
Part 3

Classical Functional Approach

Functional Approach



Functional Approach

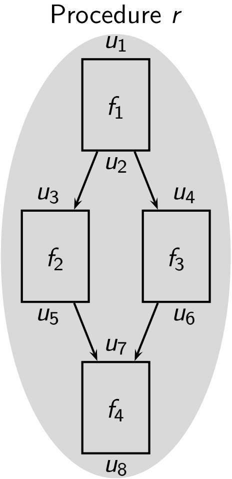


- Bottom-up Approach
- Compute summary flow functions for each procedure
- Use summary flow functions as the flow function for a call block
- Main challenge:

  Appropriate representation for summary flow functions

Notation for Summary Flow Function

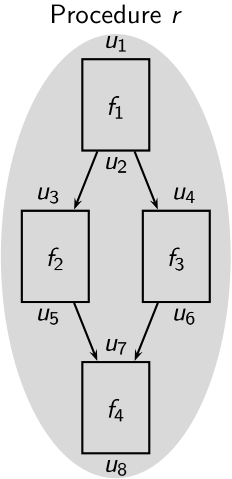
For simplicity forward flow is assumed



Notation for Summary Flow Function

For simplicity forward flow is assumed

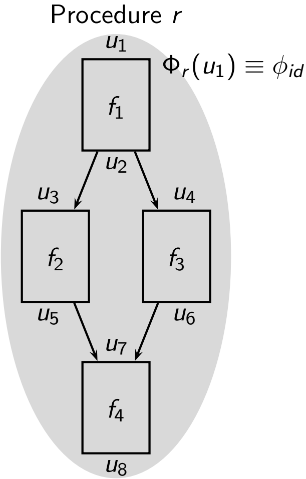
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Notation for Summary Flow Function

For simplicity forward flow is assumed

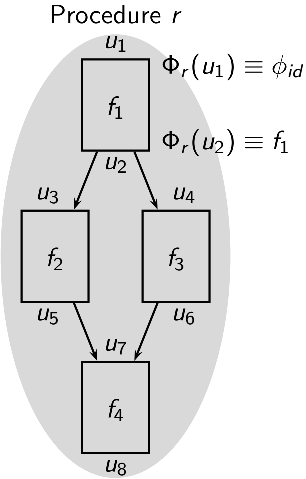
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Notation for Summary Flow Function

For simplicity forward flow is assumed

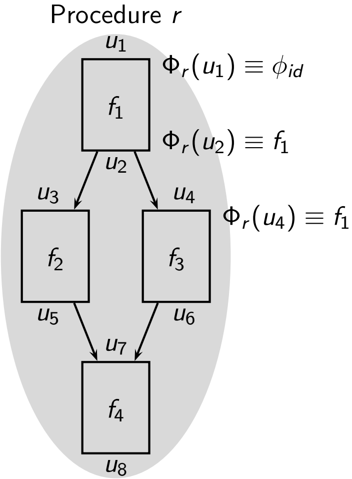
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Notation for Summary Flow Function

For simplicity forward flow is assumed

- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



$\Phi_r(u_1) \equiv \phi_{id}$

$\Phi_r(u_2) \equiv f_1$

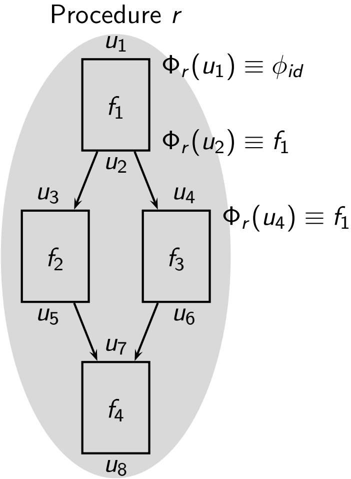
$\Phi_r(u_3) \equiv f_1$

$\Phi_r(u_4) \equiv f_1$

Notation for Summary Flow Function

For simplicity forward flow is assumed

- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



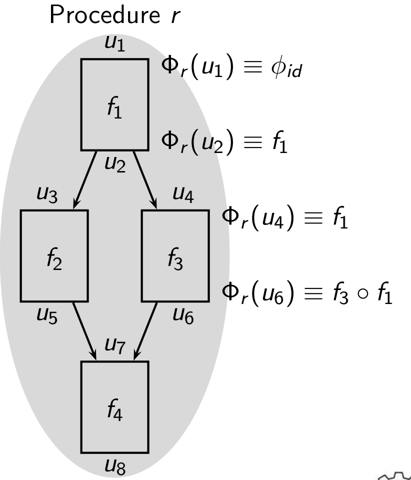
$\Phi_r(u_3) \equiv f_1$

$\Phi_r(u_5) \equiv f_2 \circ f_1$

Notation for Summary Flow Function

For simplicity forward flow is assumed

- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



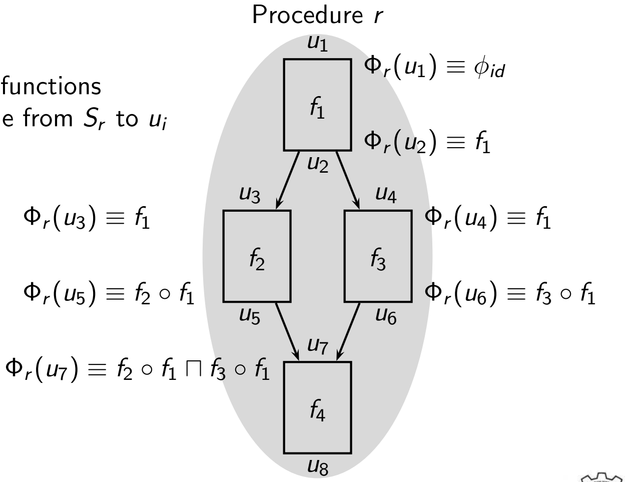
$\Phi_r(u_3) \equiv f_1$

$\Phi_r(u_5) \equiv f_2 \circ f_1$

Notation for Summary Flow Function

For simplicity forward flow is assumed

- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Procedure r

$\Phi_r(u_1) \equiv \phi_{id}$

$\Phi_r(u_2) \equiv f_1$

$\Phi_r(u_3) \equiv f_1$

$\Phi_r(u_4) \equiv f_1$

$\Phi_r(u_5) \equiv f_2 \circ f_1$

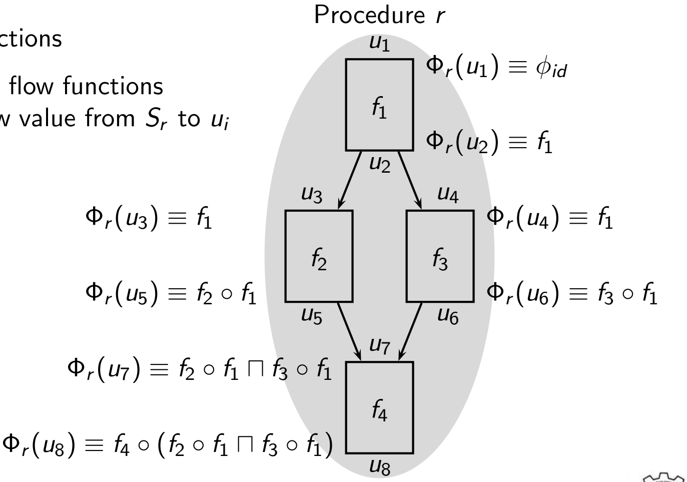
$\Phi_r(u_6) \equiv f_3 \circ f_1$

$\Phi_r(u_7) \equiv f_2 \circ f_1 \sqcap f_3 \circ f_1$

Notation for Summary Flow Function

For simplicity forward flow is assumed

- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Procedure r

$\Phi_r(u_1) \equiv \phi_{id}$

$\Phi_r(u_2) \equiv f_1$

$\Phi_r(u_3) \equiv f_1$

$\Phi_r(u_4) \equiv f_1$

$\Phi_r(u_5) \equiv f_2 \circ f_1$

$\Phi_r(u_6) \equiv f_3 \circ f_1$

$\Phi_r(u_7) \equiv f_2 \circ f_1 \sqcap f_3 \circ f_1$

$\Phi_r(u_8) \equiv f_4 \circ (f_2 \circ f_1 \sqcap f_3 \circ f_1)$

Equations for Constructing Summary Flow Functions

For simplicity forward flow is assumed. I_n is Entry of n, O_n is Exit of n

$$\Phi_r(I_n) = \begin{cases} \phi_{id} & \text{if } n \text{ is } S_r \\ \displaystyle\bigsqcap_{p \in pred(n)} \Big(\Phi_r(O_p)\Big) & \text{otherwise} \end{cases}$$

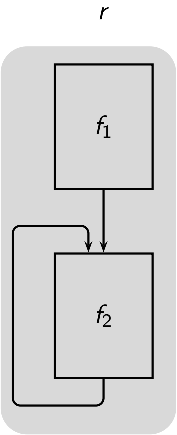
$$\Phi_r(O_n) = \begin{cases} \Phi_s(u) \circ \Phi_r(I_n) & \text{if } n \text{ calls procedure } s \\ & \text{and } u \text{ is } O_{E_s} \\ f_n \circ \Phi_r(I_n) & \text{otherwise} \end{cases}$$

The summary flow function of a given procedure *r*

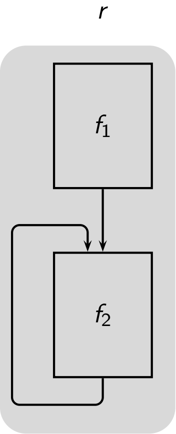
- is influenced by summary flow functions of the callees of *r*
- is not influenced by summary flow functions of the callers of *r*

Fixed point computation may be required in the presence of loops or recursion

Constructing Summary Flow Functions Iteratively



Constructing Summary Flow Functions Iteratively



Iteration #1

$\Phi_r(u_1) = \phi_{id}$

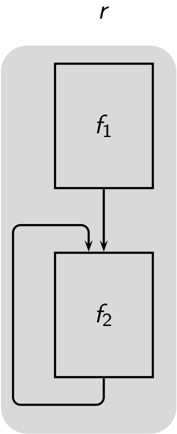
$\Phi_r(u_2) = f_1$

$\Phi_r(u_3) = f_1$

$\Phi_r(u_4) = f_2 \circ f_1$

Constructing Summary Flow Functions Iteratively

Iteration #2



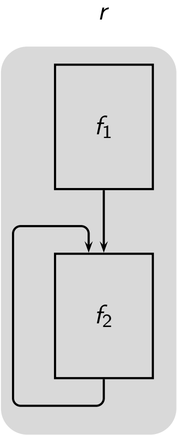
$\Phi_r(u_1) = \phi_{id}$

$\Phi_r(u_2) = f_1$

$\Phi_r(u_3) = f_1 \sqcap f_2 \circ f_1$

$\Phi_r(u_4) = f_2 \circ (f_1 \sqcap f_2 \circ f_1)$

Constructing Summary Flow Functions Iteratively



Iteration #3

$\Phi_r(u_1) = \phi_{id}$

$\Phi_r(u_2) = f_1$

$\Phi_r(u_3) = f_1 \sqcap f_2 \circ (f_1 \sqcap f_2 \circ f_1)$

$\Phi_r(u_4) = f_2 \circ (f_1 \sqcap f_2 \circ (f_1 \sqcap f_2 \circ f_1))$

Termination is possible only if all function compositions and confluences can be reduced to a finite set of functions

Lattice of Flow Functions for Live Variables Analysis

Component functions (i.e. for a single variable)

<table>
<tr><th>Lattice of data flow values</th><th>All possible flow functions</th><th>Lattice of flow functions</th></tr>
<tr>
<td>$$\begin{tikzcd} \widehat{\top} = \emptyset \arrow[d] \\ \widehat{\bot} = \{a\} \end{tikzcd}$$</td>
<td>
<table>
<tr><th>Gen_n</th><th>Kill_n</th><th>$\widehat{f}_n$</th><th>$\widehat{f}_n(x),\ \forall x \in \{\widehat{\top}, \widehat{\bot}\}$</th></tr>
<tr><td>$\emptyset$</td><td>$\emptyset$</td><td>$\widehat{\phi}_{id}$</td><td>x</td></tr>
<tr><td>$\emptyset$</td><td>$\{a\}$</td><td>$\widehat{\phi}_{\top}$</td><td>$\widehat{\top}$</td></tr>
<tr><td>$\{a\}$</td><td>$\emptyset$</td><td rowspan="2">$\widehat{\phi}_{\bot}$</td><td rowspan="2">$\widehat{\bot}$</td></tr>
<tr><td>$\{a\}$</td><td>$\{a\}$</td></tr>
</table>
</td>
<td>$$\begin{tikzcd} \widehat{\phi}_{\top} \arrow[d] \\ \widehat{\phi}_{id} \arrow[d] \\ \widehat{\phi}_{\bot} \end{tikzcd}$$</td>
</tr>
</table>

Reducing Component Flow Functions for Live Variables Analysis

Let $\widehat{\phi} \in \{\widehat{\phi}_\top, \widehat{\phi}_{id}, \widehat{\phi}_\bot\}$ and $x \in \{1, 0\}$. Then,

- $\widehat{\phi}_\top \sqcap \widehat{\phi} = \widehat{\phi}$ (because $0 + x = x$)
- $\widehat{\phi}_\bot \sqcap \widehat{\phi} = \widehat{\phi}_\bot$ (because $1 + x = 1$)
- $\widehat{\phi}_\top \circ \widehat{\phi} = \widehat{\phi}_\top$ (because $\widehat{\phi}_\top$ is a constant function)
- $\widehat{\phi}_\bot \circ \widehat{\phi} = \widehat{\phi}_\bot$ (because $\widehat{\phi}_\bot$ is a constant function)
- $\widehat{\phi}_{id} \circ \widehat{\phi} = \widehat{\phi}$ (because $\widehat{\phi}_{id}$ is the identity function)

Reducing Function Compositions in Bit Vector Frameworks

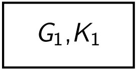
Kill_n denoted by K_n and Gen_n denoted by G_n

$$f_3(\mathrm{x}) = f_2(f_1(\mathrm{x}))$$

Reducing Function Compositions in Bit Vector Frameworks

Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

$$\begin{aligned} f_3(\mathsf{x}) &= f_2(f_1(\mathsf{x})) \\ &= f_2\big((\mathsf{x} - K_1) \cup G_1\big) \end{aligned}$$

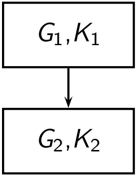


G_1, K_1

Reducing Function Compositions in Bit Vector Frameworks

Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

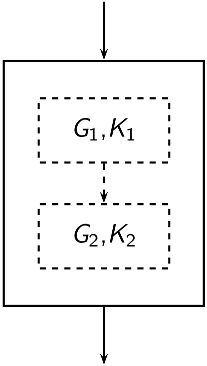
$$\begin{aligned} f_3(\mathrm{x}) &= f_2(f_1(\mathrm{x})) \\ &= f_2\big((\mathrm{x} - K_1) \cup G_1\big) \\ &= \Big(\big((\mathrm{x} - K_1) \cup G_1\big) - K_2\Big) \ \cup \ G_2 \end{aligned}$$



Reducing Function Compositions in Bit Vector Frameworks

Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

$$\begin{aligned} f_3(\text{x}) &= f_2(f_1(\text{x})) \\ &= f_2\big((\text{x} - K_1) \cup G_1\big) \\ &= \big(\big((\text{x} - K_1) \cup G_1\big) - K_2\big) \;\cup\; G_2 \\ &= \big(\text{x} - (K_1 \cup K_2)\big) \;\cup\; (G_1 - K_2) \;\cup\; G_2 \end{aligned}$$



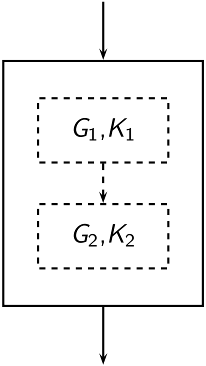
Reducing Function Compositions in Bit Vector Frameworks

Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

$$\begin{aligned} f_3(x) &= f_2(f_1(x)) \\ &= f_2\big((x - K_1) \cup G_1\big) \\ &= \big(((x - K_1) \cup G_1\big) - K_2\big) \ \cup \ G_2 \\ &= \big(x - (K_1 \cup K_2)\big) \ \cup \ (G_1 - K_2) \ \cup \ G_2 \end{aligned}$$

Hence,

$$\begin{aligned} K_3 &= K_1 \cup K_2 \\ G_3 &= (G_1 - K_2) \ \cup \ G_2 \end{aligned}$$



Reducing Bit Vector Flow Function Confluences (1)

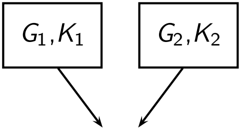
Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

- When $\sqcap$ is $\cup$,

$$\begin{aligned} f_3(\text{x}) &= f_2(\text{x}) \cup f_1(\text{x}) \\ &= \big((\text{x} - K_2) \cup G_2\big) \;\cup\; \big((\text{x} - K_1) \cup G_1\big) \\ &= \big(\text{x} - (K_1 \cap K_2)\big) \;\cup\; \big(G_1 \cup G_2\big) \end{aligned}$$

Hence,

$$\begin{aligned} K_3 &= K_1 \cap K_2 \\ G_3 &= G_1 \cup G_2 \end{aligned}$$



Reducing Bit Vector Flow Function Confluences (1)

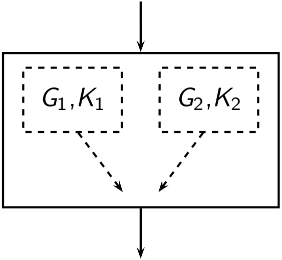
Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

- When $\sqcap$ is $\cup$,

$$\begin{aligned} f_3(x) &= f_2(x) \cup f_1(x) \\ &= \big((x - K_2) \cup G_2\big) \;\cup\; \big((x - K_1) \cup G_1\big) \\ &= \big(x - (K_1 \cap K_2)\big) \;\cup\; \big(G_1 \cup G_2\big) \end{aligned}$$

Hence,

$$\begin{aligned} K_3 &= K_1 \cap K_2 \\ G_3 &= G_1 \cup G_2 \end{aligned}$$



Reducing Bit Vector Flow Function Confluences (2)

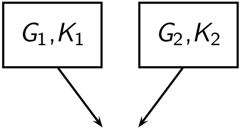
Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

- When $\sqcap$ is $\cap$,

$$\begin{aligned} f_3(x) &= f_2(x) \cap f_1(x) \\ &= \big((x - K_2) \cup G_2\big) \ \cap \ \big((x - K_1) \cup G_1\big) \\ &= \big(x - (K_1 \cup K_2)\big) \ \cup \ \big(G_1 \cap G_2\big) \end{aligned}$$

Hence,

$$\begin{aligned} K_3 &= K_1 \cup K_2 \\ G_3 &= G_1 \cap G_2 \end{aligned}$$



Reducing Bit Vector Flow Function Confluences (2)

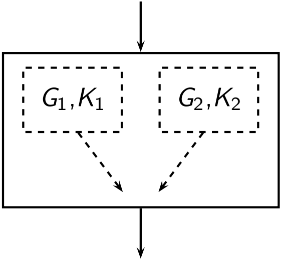
Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

- When $\sqcap$ is $\cap$,

$$\begin{aligned} f_3(\mathrm{x}) &= f_2(\mathrm{x}) \cap f_1(\mathrm{x}) \\ &= \big((\mathrm{x} - K_2) \cup G_2\big) \;\cap\; \big((\mathrm{x} - K_1) \cup G_1\big) \\ &= \big(\mathrm{x} - (K_1 \cup K_2)\big) \;\cup\; \big(G_1 \cap G_2\big) \end{aligned}$$

Hence,

$$\begin{aligned} K_3 &= K_1 \cup K_2 \\ G_3 &= G_1 \cap G_2 \end{aligned}$$

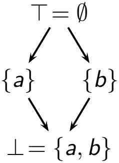


Lattice of Flow Functions for Live Variables Analysis

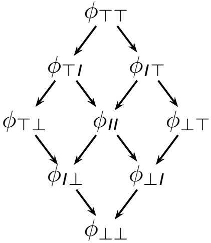
Flow functions for two variables

- Product of lattices for independent variables (because of separability)

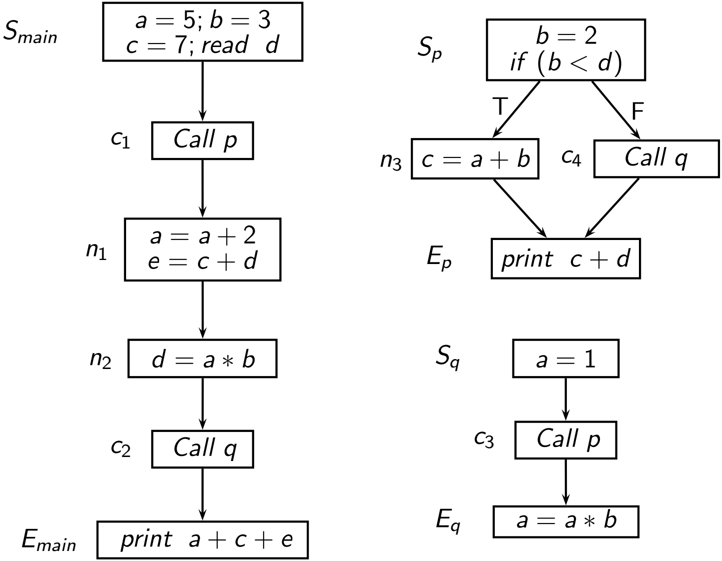
Lattice of data flow values	All possible flow functions	Lattice of flow functions



Gen_n	Kill_n	f_n	Gen_n	Kill_n	f_n
$\emptyset$	$\emptyset$	ϕ_{II}	$\{b\}$	$\emptyset$	$\phi_{I\bot}$
$\emptyset$	$\{a\}$	$\phi_{\top I}$	$\{b\}$	$\{a\}$	$\phi_{\top\bot}$
$\emptyset$	$\{b\}$	$\phi_{I\top}$	$\{b\}$	$\{b\}$	$\phi_{I\bot}$
$\emptyset$	$\{a, b\}$	$\phi_{\top\top}$	$\{b\}$	$\{a, b\}$	$\phi_{\top\bot}$
$\{a\}$	$\emptyset$	$\phi_{\bot I}$	$\{a, b\}$	$\emptyset$	$\phi_{\bot\bot}$
$\{a\}$	$\{a\}$	$\phi_{\bot I}$	$\{a, b\}$	$\{a\}$	$\phi_{\bot\bot}$
$\{a\}$	$\{b\}$	$\phi_{\bot\top}$	$\{a, b\}$	$\{b\}$	$\phi_{\bot\bot}$
$\{a\}$	$\{a, b\}$	$\phi_{\bot\top}$	$\{a, b\}$	$\{a, b\}$	$\phi_{\bot\bot}$



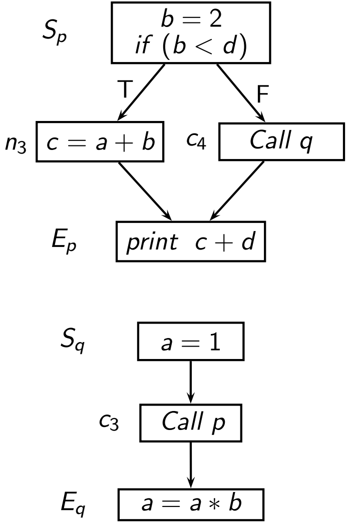
An Example of Interprocedural Liveness Analysis



Summary Flow Functions for Interprocedural Liveness Analysis

Proc.	Flow Function	Defining Expression	Iteration #1		Changes in iteration #2	
			Gen	Kill	Gen	Kill
p	$\Phi_p(E_p)$	f_{E_p}	$\{c, d\}$	$\emptyset$		
	$\Phi_p(n_3)$	$f_{n_3} \circ \Phi_p(E_p)$	$\{a, b, d\}$	$\{c\}$		
	$\Phi_p(c_4)$	$f_q \circ \Phi_p(E_p) = \phi_\top$	$\emptyset$	$\{a, b, c, d, e\}$	$\{d\}$	$\{a, b, c\}$
	$\Phi_p(S_p)$	$f_{S_p} \circ \left(\Phi_p(n_3) \sqcap \Phi_p(c_4)\right)$	$\{a, d\}$	$\{b, c\}$		
	f_p	$\Phi_p(S_p)$	$\{a, d\}$	$\{b, c\}$		
q	$\Phi_q(E_q)$	f_{E_q}	$\{a, b\}$	$\{a\}$		
	$\Phi_q(c_3)$	$f_p \circ \Phi_q(E_q)$	$\{a, d\}$	$\{a, b, c\}$		
	$\Phi_q(S_q)$	$f_{S_q} \circ \Phi_q(c_3)$	$\{d\}$	$\{a, b, c\}$		
	f_q	$\Phi_q(S_q)$	$\{d\}$	$\{a, b, c\}$		

Computed Summary Flow Functions



Summary Flow Function	
$\Phi_p(E_p)$	$BI_p \cup \{c, d\}$
$\Phi_p(n_3)$	$\left(BI_p - \{c\}\right) \cup \{a, b, d\}$
$\Phi_p(c_4)$	$\left(BI_p - \{a, b, c\}\right) \cup \{d\}$
$\Phi_p(S_p)$	$\left(BI_p - \{b, c\}\right) \cup \{a, d\}$
$\Phi_q(E_q)$	$\left(BI_q - \{a\}\right) \cup \{a, b\}$
$\Phi_q(c_3)$	$\left(BI_q - \{a, b, c\}\right) \cup \{a, d\}$
$\Phi_q(S_q)$	$\left(BI_q - \{a, b, c\}\right) \cup \{d\}$

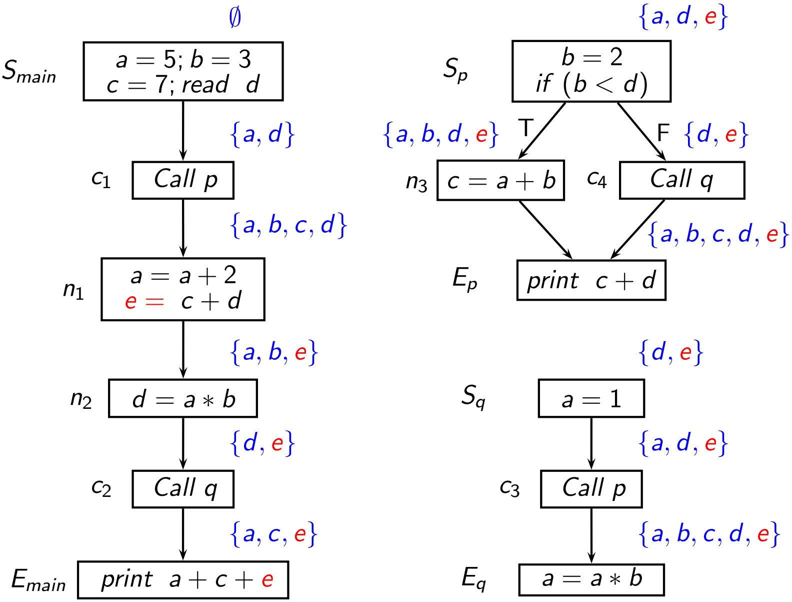
Result of Interprocedural Liveness Analysis

Data flow variable	Summary flow function		Data flow value
	Name	Definition	
Procedure *main*, $BI = \emptyset$			
In_{E_m}	$\Phi_m(E_m)$	$BI_m \cup \{a, c, e\}$	$\{a, c, e\}$
In_{c_2}	$\Phi_m(c_2)$	$\big(BI_m - \{a, b, c\}\big) \cup \{d, e\}$	$\{d, e\}$
In_{n_2}	$\Phi_m(n_2)$	$\big(BI_m - \{a, b, c, d\}\big) \cup \{a, b, e\}$	$\{a, b, e\}$
In_{n_1}	$\Phi_m(n_1)$	$\big(BI_m - \{a, b, c, d, e\}\big) \cup \{a, b, c, d\}$	$\{a, b, c, d\}$
In_{c_1}	$\Phi_m(c_1)$	$\big(BI_m - \{a, b, c, d, e\}\big) \cup \{a, d\}$	$\{a, d\}$
In_{S_m}	$\Phi_m(S_m)$	$BI_m - \{a, b, c, d, e\}$	$\emptyset$

Result of Interprocedural Liveness Analysis

Data flow variable	Summary flow function		Data flow value
	Name	Definition	
Procedure p, $BI = \{a, b, c, d, e\}$			
In_{E_p}	$\Phi_p(E_p)$	$BI_p \cup \{c, d\}$	$\{a, b, c, d, e\}$
In_{n_3}	$\Phi_p(n_3)$	$\big(BI_p - \{c\}\big) \cup \{a, b, d\}$	$\{a, b, d, e\}$
In_{c_4}	$\Phi_p(c_4)$	$\big(BI_p - \{a, b, c\}\big) \cup \{d\}$	$\{d, e\}$
In_{S_p}	$\Phi_p(S_p)$	$\big(BI_p - \{b, c\}\big) \cup \{a, d\}$	$\{a, d, e\}$
Procedure q, $BI = \{a, b, c, d, e\}$			
In_{E_q}	$\Phi_q(E_q)$	$\big(BI_q - \{a\}\big) \cup \{a, b\}$	$\{a, b, c, d, e\}$
In_{c_3}	$\Phi_q(c_3)$	$\big(BI_q - \{a, b, c\}\big) \cup \{a, d\}$	$\{a, d, e\}$
In_{S_q}	$\Phi_q(S_q)$	$\big(BI_q - \{a, b, c\}\big) \cup \{d\}$	$\{d, e\}$

Context Sensitivity of Interprocedural Liveness Analysis



$\emptyset$

S_{main}: $a = 5; b = 3$ / $c = 7;$ *read* d

$\{a, d\}$

c_1: *Call p*

$\{a, b, c, d\}$

n_1: $a = a + 2$ / $e = c + d$

$\{a, b, e\}$

n_2: $d = a * b$

$\{d, e\}$

c_2: *Call q*

$\{a, c, e\}$

E_{main}: *print* $a + c + e$

$\{a, d, e\}$

S_p: $b = 2$ / *if* $(b < d)$

T $\{a, b, d, e\}$

F $\{d, e\}$

n_3: $c = a + b$

c_4: *Call q*

$\{a, b, c, d, e\}$

E_p: *print* $c + d$

$\{d, e\}$

S_q: $a = 1$

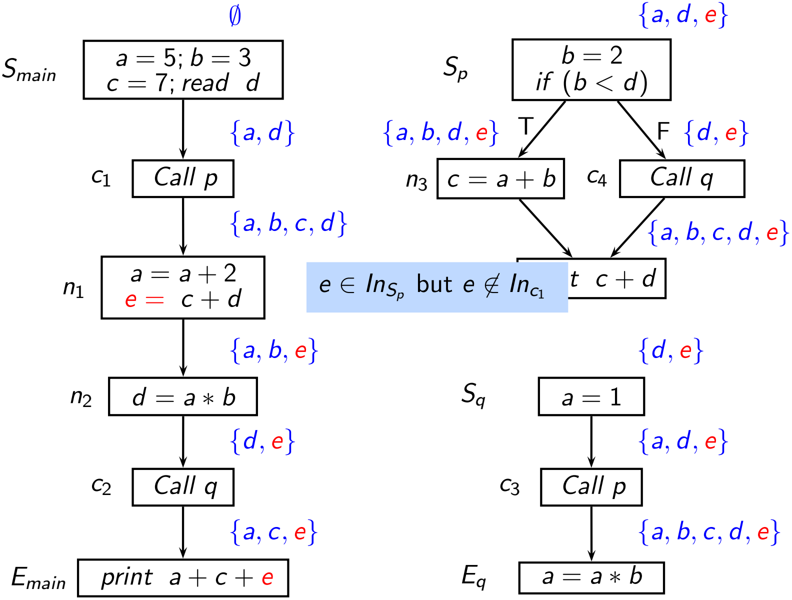
$\{a, d, e\}$

c_3: *Call p*

$\{a, b, c, d, e\}$

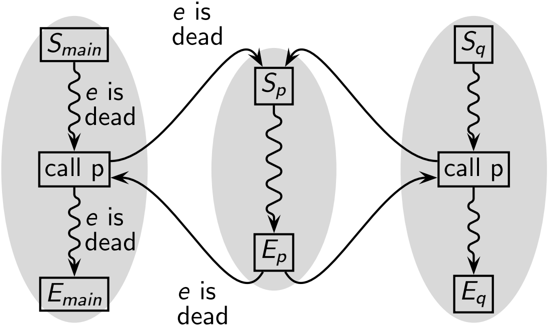
E_q: $a = a * b$

Context Sensitivity of Interprocedural Liveness Analysis



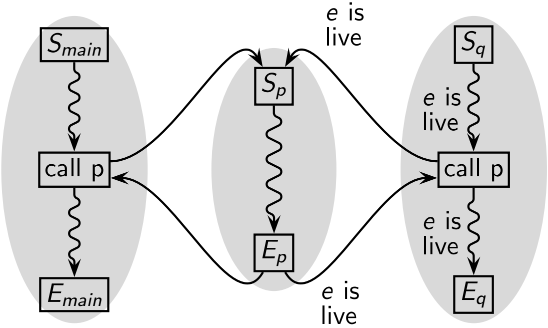
$e \in In_{S_p}$ but $e \notin In_{c_1}$

Explaining Context Sensitivity



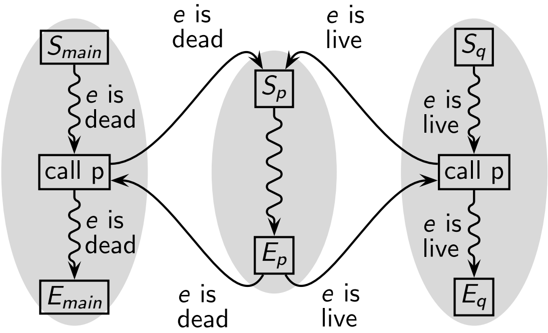
- Flow function of procedure p is identity with respect to variable e

Explaining Context Sensitivity



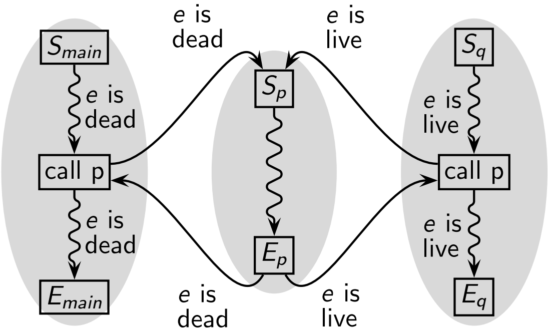
- Flow function of procedure *p* is identity with respect to variable *e*

Explaining Context Sensitivity



- Flow function of procedure p is identity with respect to variable e
- Is e live in the body of procedure p?
  - During the analysis: Depends on the calling context
  - After the analysis: Yes (static approximation across all executions)

Explaining Context Sensitivity



- Flow function of procedure p is identity with respect to variable e
- Is e live in the body of procedure p?
  - During the analysis: Depends on the calling context
  - After the analysis: Yes (static approximation across all executions)
- Distinction between caller's effect on callee and callee's effect on caller

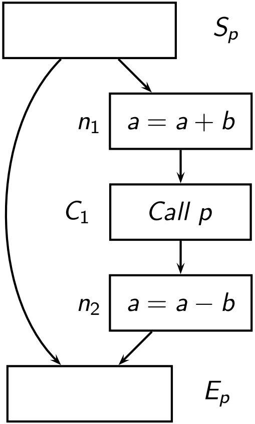
Tutorial Problem #1

Perform interprocedural live variables analysis for the following program

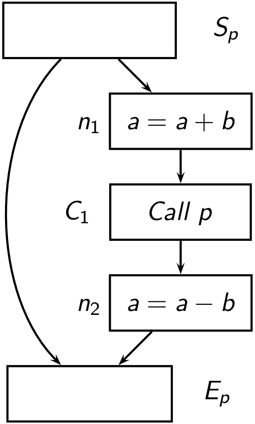
```
main()
{
   p();
}
```

```
p()
{  while (c < 10)
   {  p();
      a = a*b;
   }
}
```

Tutorial Problem #2: Summary Flow Function for Constant Propagation

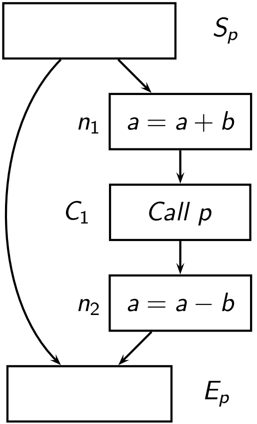


Tutorial Problem #2: Summary Flow Function for Constant Propagation



	Iter. #1	Iter. #2
$[\Phi_p(S_p)](\langle v_a, v_b\rangle)$	$\langle v_a, v_b\rangle$	$\langle v_a, v_b\rangle$
$[\Phi_p(n_1)](\langle v_a, v_b\rangle)$	$\langle v_a + v_b, v_b\rangle$	$\langle v_a + v_b, v_b\rangle$
$[\Phi_p(C_1)](\langle v_a, v_b\rangle)$	$\langle \widehat{\top}, \widehat{\top}\rangle$	$\langle v_a + v_b, v_b\rangle$
$[\Phi_p(n_2)](\langle v_a, v_b\rangle)$	$\langle \widehat{\top}, \widehat{\top}\rangle$	$\langle v_a, v_b\rangle$
$[\Phi_p(E_p)](\langle v_a, v_b\rangle)$	$\langle v_a, v_b\rangle$	$\langle v_a, v_b\rangle$
$f_p(\langle v_a, v_b\rangle)$	$\langle v_a, v_b\rangle$	$\langle v_a, v_b\rangle$

Tutorial Problem #2: Summary Flow Function for Constant Propagation



	Iter. #1	Iter. #2
$[\Phi_p(S_p)](\langle v_a, v_b\rangle)$	$\langle v_a, v_b\rangle$	$\langle v_a, v_b\rangle$
$[\Phi_p(n_1)](\langle v_a, v_b\rangle)$	$\langle v_a + v_b, v_b\rangle$	$\langle v_a + v_b, v_b\rangle$
$[\Phi_p(C_1)](\langle v_a, v_b\rangle)$	$\langle \widehat{\top}, \widehat{\top}\rangle$	$\langle v_a + v_b, v_b\rangle$
$[\Phi_p(n_2)](\langle v_a, v_b\rangle)$	$\langle \widehat{\top}, \widehat{\top}\rangle$	$\langle v_a, v_b\rangle$
$[\Phi_p(E_p)](\langle v_a, v_b\rangle)$	$\langle v_a, v_b\rangle$	$\langle v_a, v_b\rangle$
$f_p(\langle v_a, v_b\rangle)$	$\langle v_a, v_b\rangle$	$\langle v_a, v_b\rangle$

Will this work always?

Tutorial Problem #3

- Is a*b available on line 18? Line 6?
- Perform available expressions analysis by constructing the summary flow function for procedure p

```
1.   main()
2.   {
3.        c = a*b;
4.        p();
5.        a = a*b;
6.   }
```

```
 7.  p()
 8.  {   if (...)
 9.      {   a = a*b;
10.          p();
11.      }
12.      else if (...)
13.      {   c = a * b;
14.          p();
15.          c = a;
16.      }
17.      else
18.          ; /* ignore */
19.  }
```

Limitations of Functional Approach to Interprocedural Data Flow Analysis

Problems with constructing summary flow functions

Limitations of Functional Approach to Interprocedural Data Flow Analysis

Problems with constructing summary flow functions

- Reducing expressions defining flow functions may not be possible in the presence of dependent parts
- May work for some instances of some problems but not for all
- Hence basic blocks in pointer analysis and constant propagation contain a single statement

Overall Flow Function and Component Function

- Overall flow function $f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$
- Component function: $\widehat{h}_i$ which computes the value of $\widehat{x}_i$

Overall Flow Function and Component Function

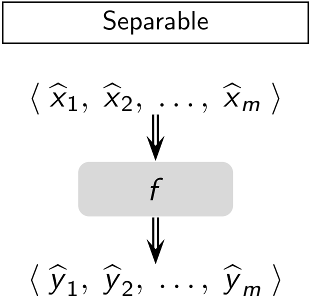
- Overall flow function $f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$
- Component function: $\widehat{h}_i$ which computes the value of $\widehat{x}_i$

Separable

General Non-Separable

Overall Flow Function and Component Function

- Overall flow function $f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$
- Component function: $\widehat{h}_i$ which computes the value of $\widehat{x}_i$

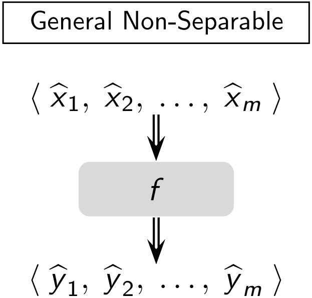


Separable

$\langle\, \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m \,\rangle$

f

$\langle\, \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m \,\rangle$



General Non-Separable

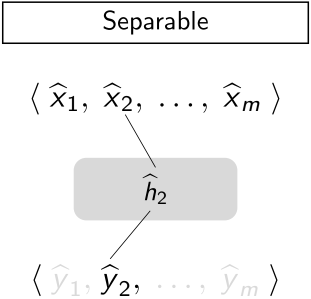
$\langle\, \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m \,\rangle$

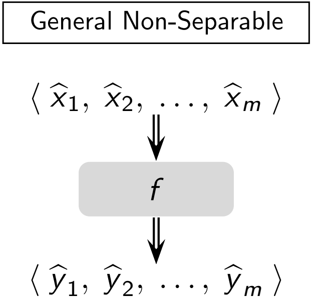
f

$\langle\, \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m \,\rangle$

Overall Flow Function and Component Function

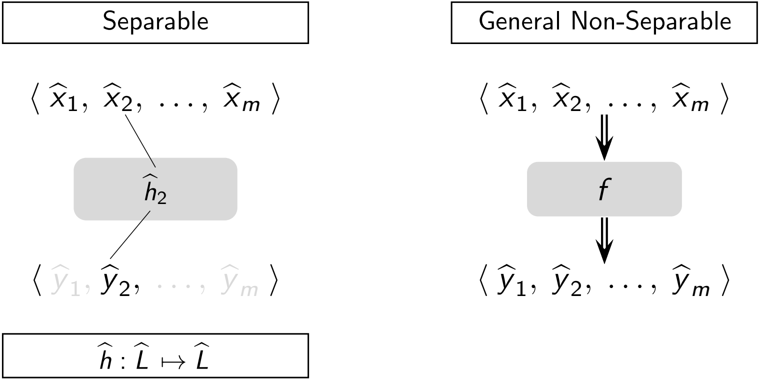
- Overall flow function $f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$
- Component function: $\widehat{h}_i$ which computes the value of $\widehat{x}_i$


Separable
$\langle \widehat{x}_1, \widehat{x}_2, \ldots, \widehat{x}_m \rangle$
$\widehat{h}_2$
$\langle \widehat{y}_1, \widehat{y}_2, \ldots, \widehat{y}_m \rangle$


General Non-Separable
$\langle \widehat{x}_1, \widehat{x}_2, \ldots, \widehat{x}_m \rangle$
f
$\langle \widehat{y}_1, \widehat{y}_2, \ldots, \widehat{y}_m \rangle$

Overall Flow Function and Component Function

- Overall flow function $f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$
- Component function: $\widehat{h}_i$ which computes the value of $\widehat{x}_i$



Separable

$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$

$\widehat{h}_2$

$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$

$\widehat{h} : \widehat{L} \mapsto \widehat{L}$

General Non-Separable

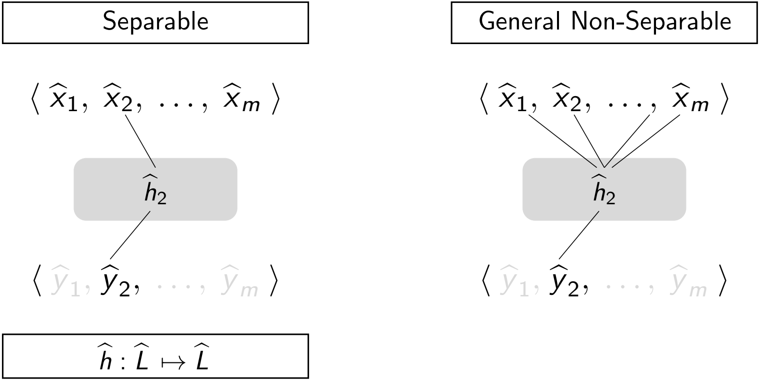
$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$

f

$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$

Overall Flow Function and Component Function

- Overall flow function $f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$
- Component function: $\widehat{h}_i$ which computes the value of $\widehat{x}_i$



Separable

General Non-Separable

$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$

$\widehat{h}_2$

$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$

$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$

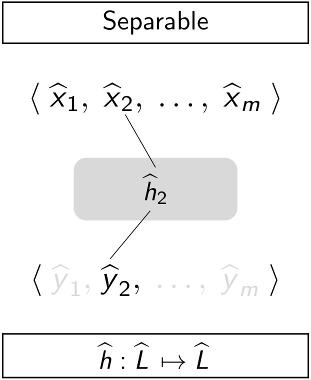
$\widehat{h}_2$

$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$

$\widehat{h} : \widehat{L} \mapsto \widehat{L}$

Overall Flow Function and Component Function

- Overall flow function $f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$
- Component function: $\widehat{h}_i$ which computes the value of $\widehat{x}_i$



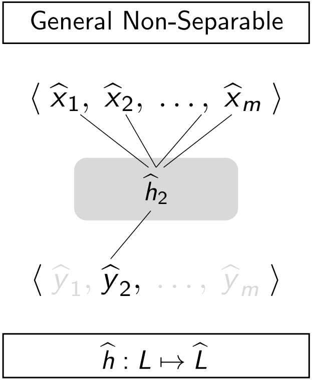
Separable

$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$

$\widehat{h}_2$

$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$

$\widehat{h} : \widehat{L} \mapsto \widehat{L}$



General Non-Separable

$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$

$\widehat{h}_2$

$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$

$\widehat{h} : L \mapsto \widehat{L}$

Example: All bit vector frameworks

Example: Points-To Analysis

Entity Functions in Points-to Analysis

Statement with $a \in L\_locations$	Entity functions		Closed under composition?
$\ldots = null$	Constant	$\widehat{L} \mapsto \widehat{L}$	Yes
$\ldots = \&b$	Constant	$\widehat{L} \mapsto \widehat{L}$	Yes
$\ldots = b$	Identity	$\widehat{L} \mapsto \widehat{L}$	Yes
$\ldots = *b$	?	$L \mapsto \widehat{L}$	No

Entity Functions in Constant Propagation

Statement	Entity functions		Closed under composition?
$a = 5$	Constant	$\widehat{L} \mapsto \widehat{L}$	Yes
$a = b$	Constant	$\widehat{L} \mapsto \widehat{L}$	Yes
$a = b + 5$	Linear	$\widehat{L} \mapsto \widehat{L}$	Yes
$a = b + c$	?	$L \mapsto \widehat{L}$	No

Enumeration Based Functional Approach

- Instead of constructing flow functions, remember the mapping $x \mapsto y$ as input output values
- Reuse output value of a flow function when the same input value is encountered again

Enumeration Based Functional Approach

- Instead of constructing flow functions, remember the mapping $x \mapsto y$ as input output values
- Reuse output value of a flow function when the same input value is encountered again

Requires the number of values to be finite

Part 4

Classical Call Strings Approach

Classical Full Call Strings Approach

Most general, flow and context sensitive method

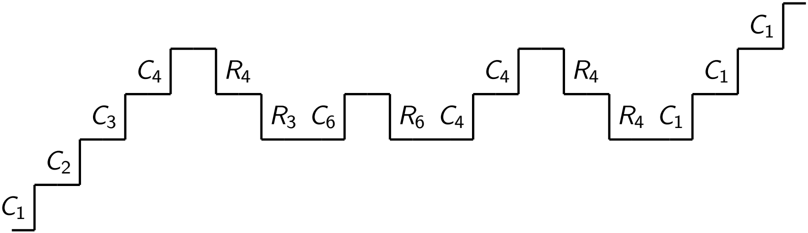
- Remember call history

  Information should be propagated *back* to the correct point

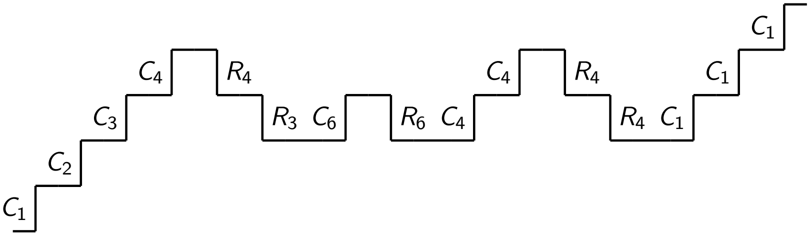
- Call string at a program point:
  - Sequence of *unfinished calls* reaching that point
  - Starting from the S_{main}

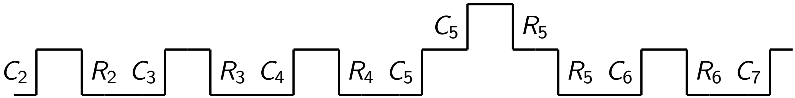
  A snap-shot of call stack in terms of call sites

Interprocedural Validity and Calling Contexts

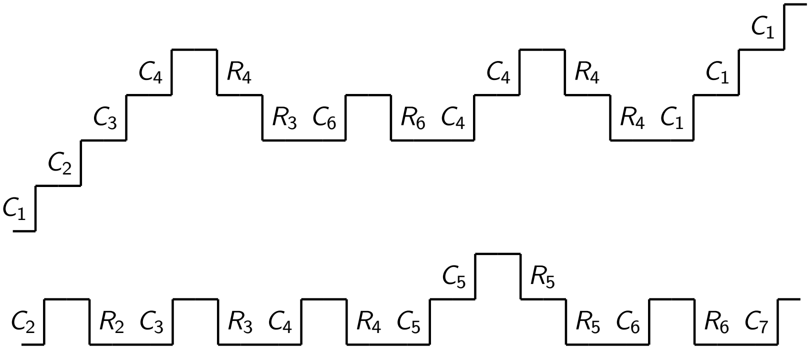


Interprocedural Validity and Calling Contexts



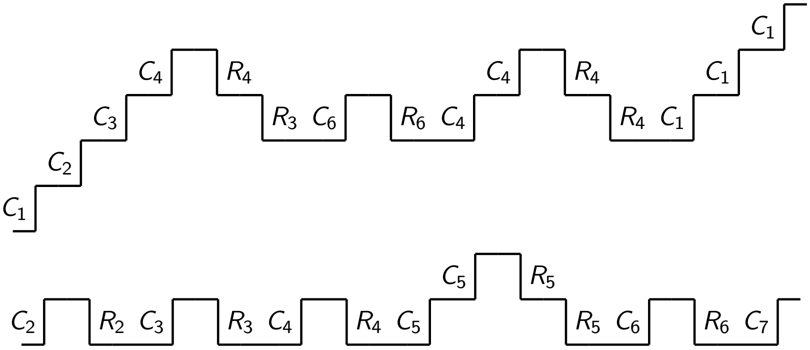


Interprocedural Validity and Calling Contexts



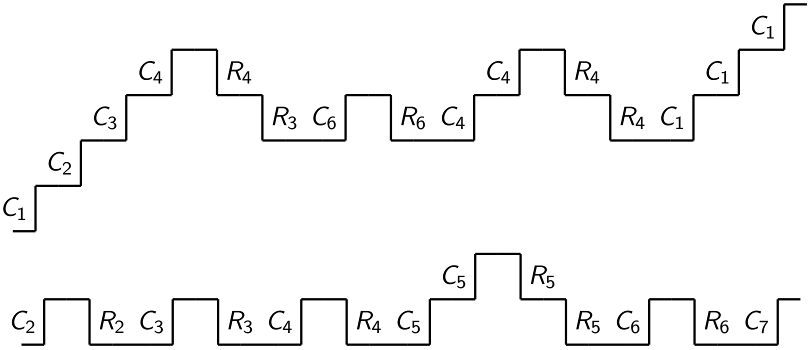
- “You can descend only as much as you have ascended!”

Interprocedural Validity and Calling Contexts



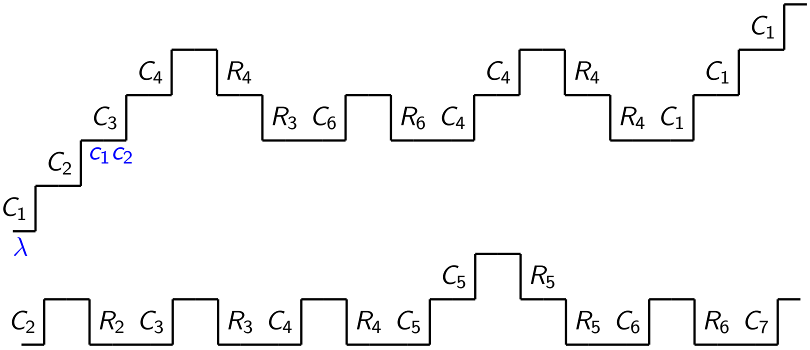
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step

Interprocedural Validity and Calling Contexts



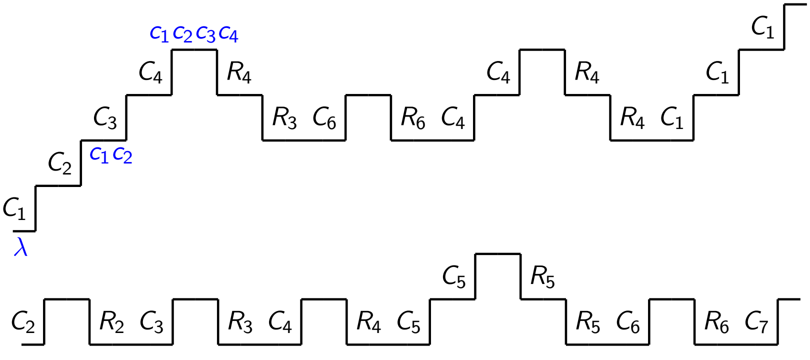
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



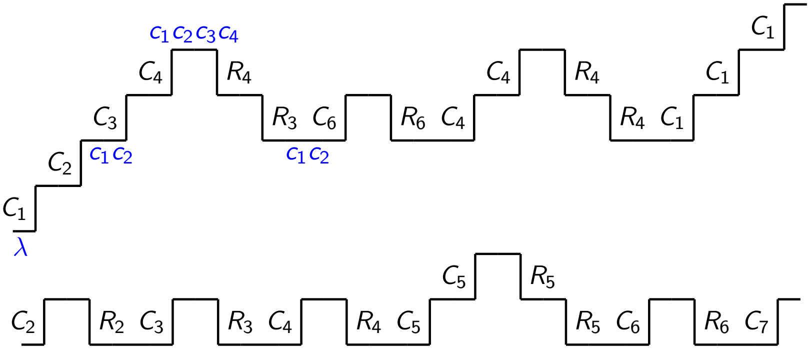
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



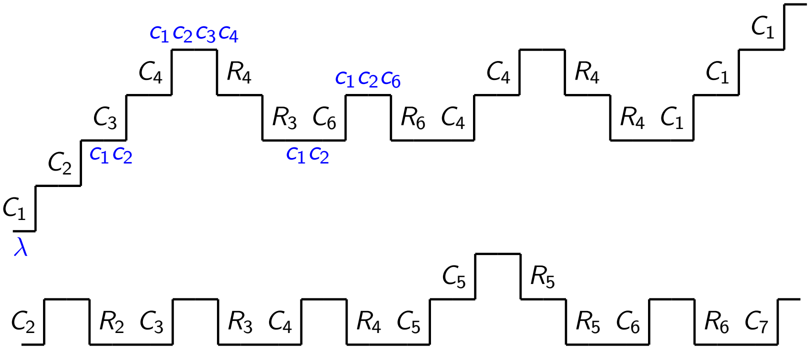
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



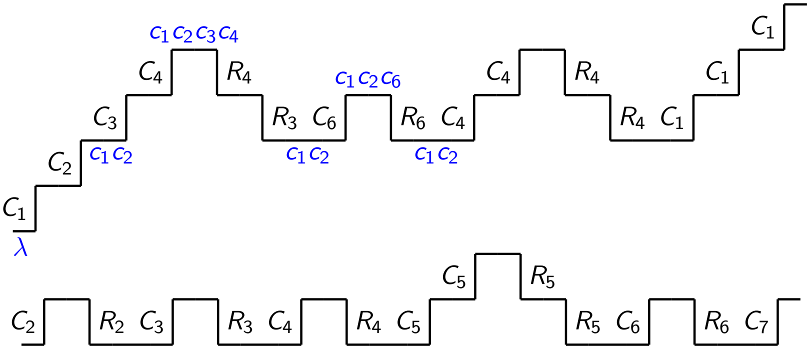
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



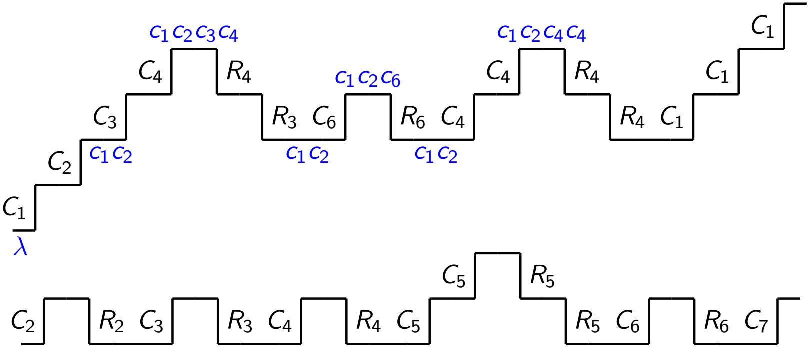
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



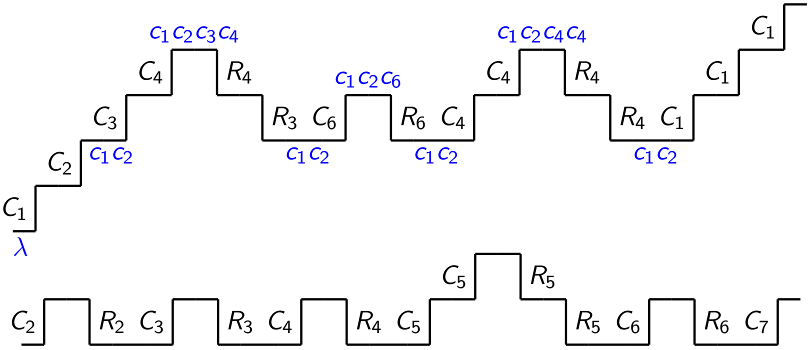
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



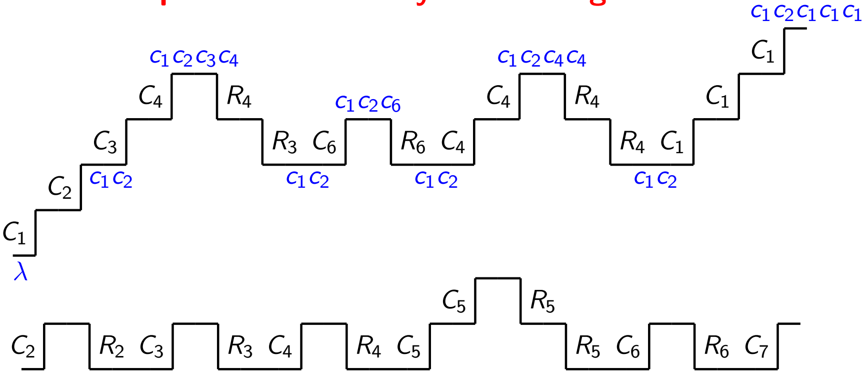
- “You can descend only as much as you have ascended!”
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



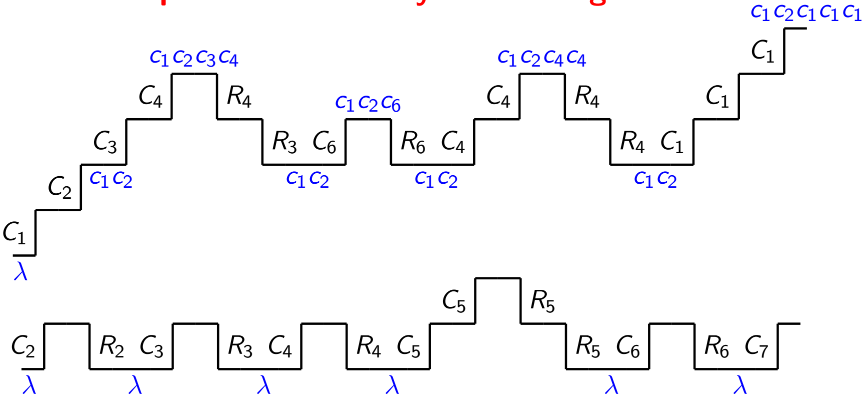
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



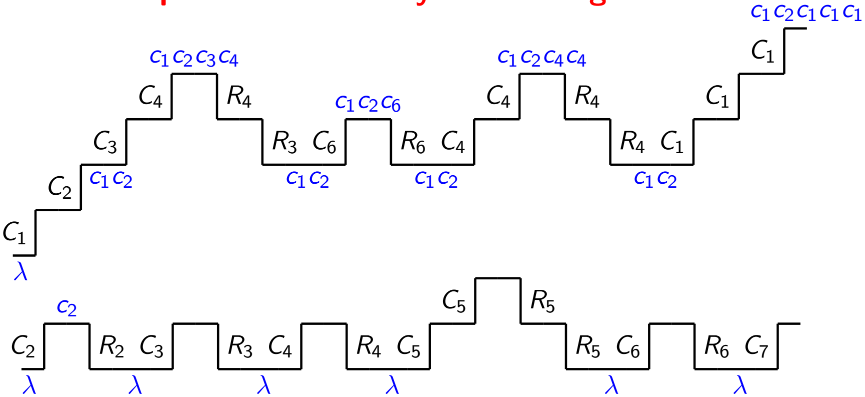
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



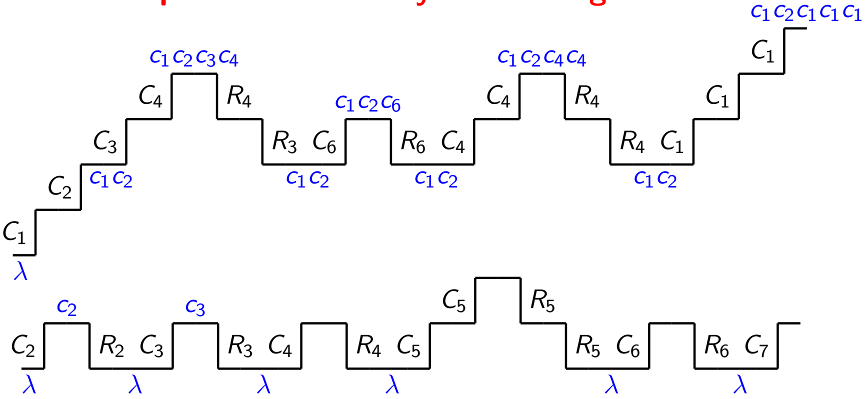
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



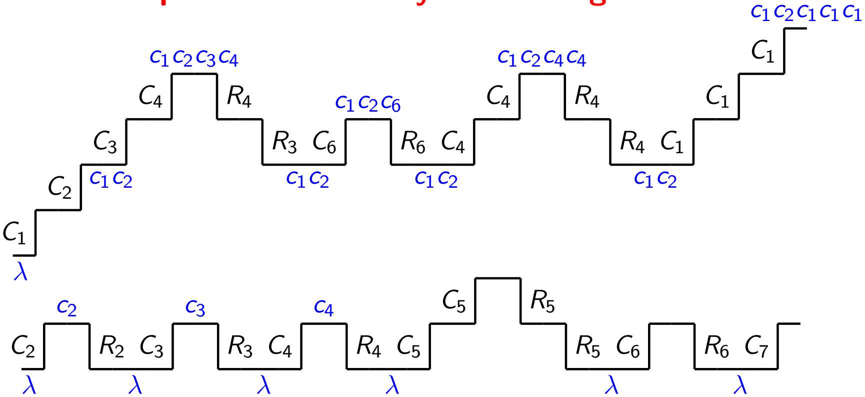
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



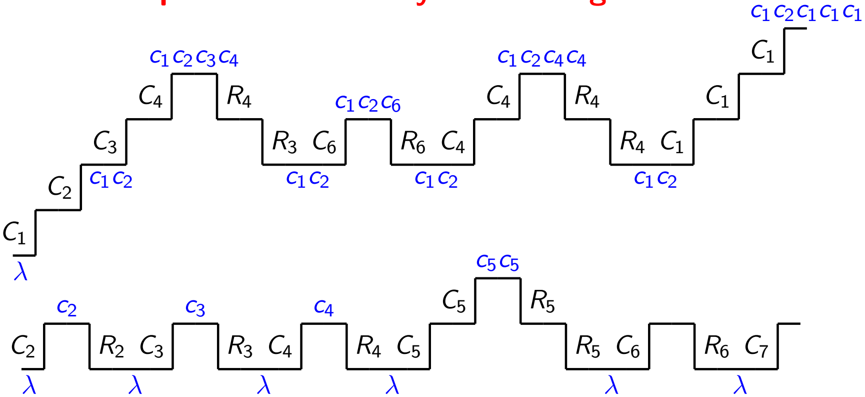
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



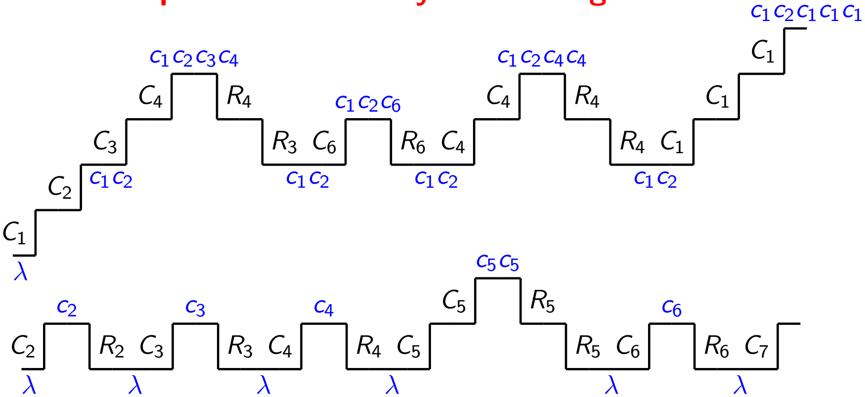
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



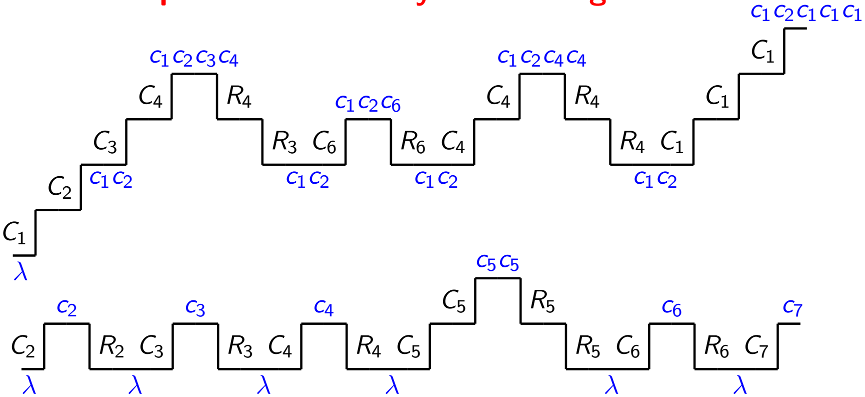
- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Validity and Calling Contexts



- "You can descend only as much as you have ascended!"
- Every descending step must match a corresponding ascending step
- Calling context is represented by the remaining descending steps

Interprocedural Data Flow Analysis Using Call Strings

- Augmented data flow information
  - IN_n and OUT_n are partial maps from call strings to L
  - The final data flow information at a program point is

$$\begin{aligned} In_n &= \bigsqcap_{\langle \sigma, x \rangle \in \text{IN}_n} x \\ Out_n &= \bigsqcap_{\langle \sigma, x \rangle \in \text{OUT}_n} x \end{aligned}$$

(glb of data flow values for all call strings)

- Flow functions to manipulate tagged data flow information
  - Intraprocedural edges manipulate data flow value x
  - Interprocedural edges manipulate call string σ

Augmented Data Flow Equations: Computing IN_n

$$\text{IN}_n = \begin{cases} \langle \lambda, BI \rangle & n \text{ is a } S_{main} \\ \biguplus_{p \in pred(n)} \text{OUT}_p & \text{otherwise} \end{cases}$$

where we merge underlying data flow values only if the contexts are same

Augmented Data Flow Equations: Computing IN_n

$$\text{IN}_n = \begin{cases} \langle \lambda, BI \rangle & n \text{ is a } S_{main} \\ \biguplus\limits_{p \in pred(n)} \text{OUT}_p & \text{otherwise} \end{cases}$$

where we merge underlying data flow values only if the contexts are same

$$\begin{aligned} \Gamma_1 \uplus \Gamma_2 \;=\; \big\{ \langle \sigma, z \rangle \mid\; & \langle \sigma, x \rangle \in \Gamma_1 \wedge \langle \sigma, y \rangle \in \Gamma_2 \;\Rightarrow\; z = x \sqcap y, \\ & \langle \sigma, x \rangle \in \Gamma_1 \wedge \langle \sigma, y \rangle \notin \Gamma_2 \;\Rightarrow\; z = x, \\ & \langle \sigma, x \rangle \notin \Gamma_1 \wedge \langle \sigma, y \rangle \in \Gamma_2 \;\Rightarrow\; z = y \;\big\} \end{aligned}$$

Augmented Data Flow Equations: Computing OUT_n

- Call node C_i
  - Append c_i to every σ
  - Propagate the data flow values unchanged

Augmented Data Flow Equations: Computing OUT_n

- Call node C_i
  - Append c_i to every σ
  - Propagate the data flow values unchanged
- Return node R_i
  - If the last call site is c_i, remove it and propagate the data flow value unchanged
  - Block other data flow values

Augmented Data Flow Equations: Computing OUT$_n$

- Call node C_i
  - Append c_i to every σ
  - Propagate the data flow values unchanged

Ascend

- Return node R_i
  - If the last call site is c_i, remove it and propagate the data flow value unchanged
  - Block other data flow values

Descend

Augmented Data Flow Equations: Computing OUT_n

- Call node C_i
  - Append c_i to every σ
  - Propagate the data flow values unchanged

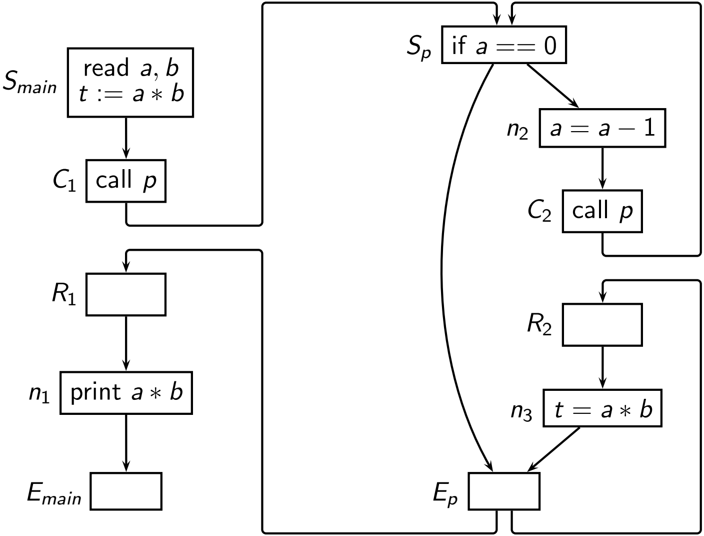
Ascend

- Return node R_i
  - If the last call site is c_i, remove it and propagate the data flow value unchanged
  - Block other data flow values

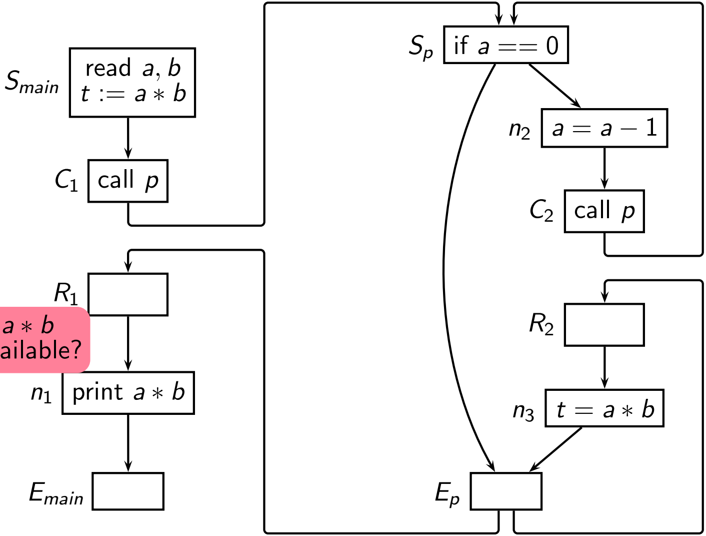
Descend

$$\text{OUT}_n(X) \quad = \quad \begin{cases} \left\{ \langle \sigma \cdot c_i, \text{x} \rangle \mid \langle \sigma, \text{x} \rangle \in \text{IN}_n \right\} & n \text{ is } C_i \\ \left\{ \langle \sigma, \text{x} \rangle \mid \langle \sigma \cdot c_i, \text{x} \rangle \in \text{IN}_n \right\} & n \text{ is } R_i \\ \left\{ \langle \sigma, f_n(\text{x}) \rangle \mid \langle \sigma, \text{x} \rangle \in \text{IN}_n \right\} & \text{otherwise} \end{cases}$$

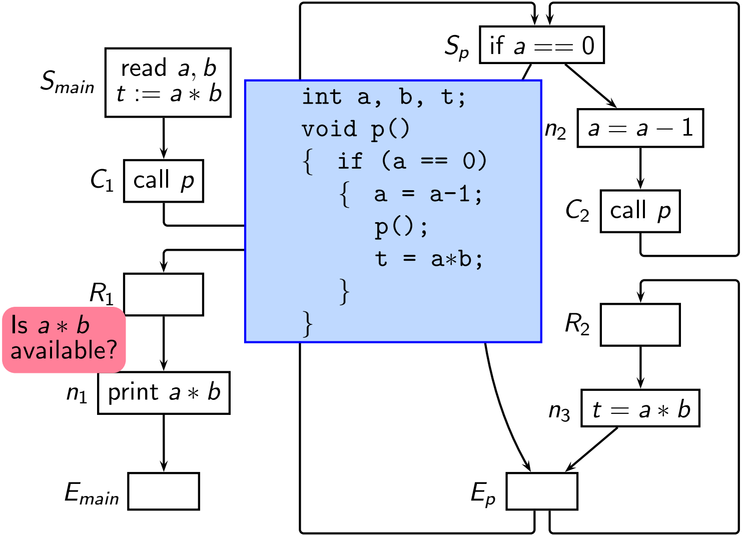
Available Expressions Analysis Using Call Strings Approach



Available Expressions Analysis Using Call Strings Approach

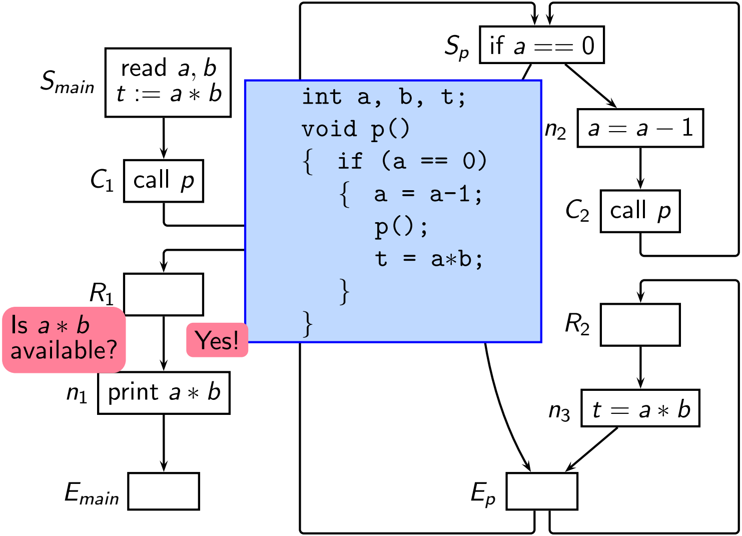


Available Expressions Analysis Using Call Strings Approach



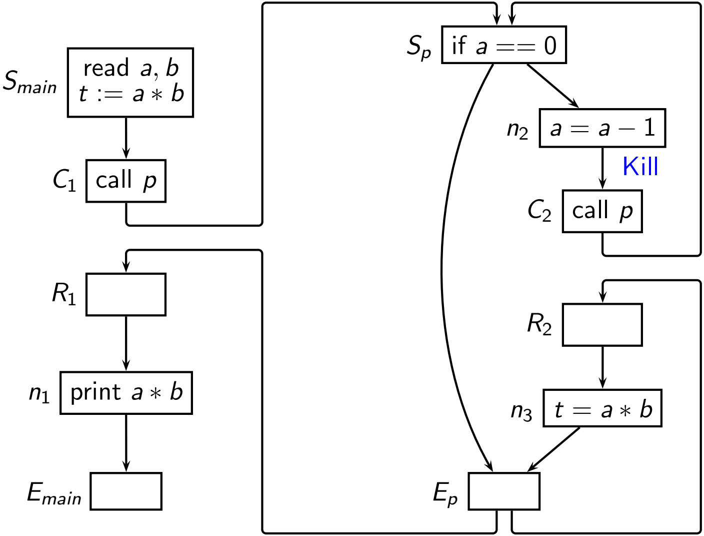
```
int a, b, t;
void p()
{  if (a == 0)
   {  a = a-1;
      p();
      t = a*b;
   }
}
```

Available Expressions Analysis Using Call Strings Approach

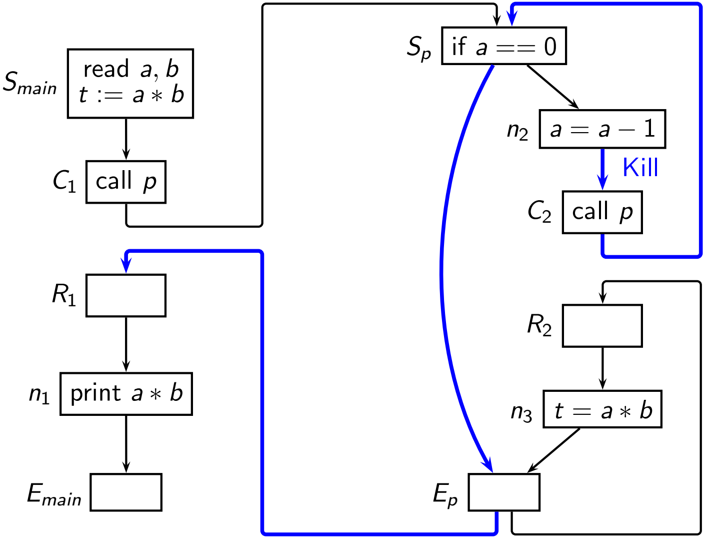


```
int a, b, t;
void p()
{  if (a == 0)
   {  a = a-1;
      p();
      t = a*b;
   }
}
```

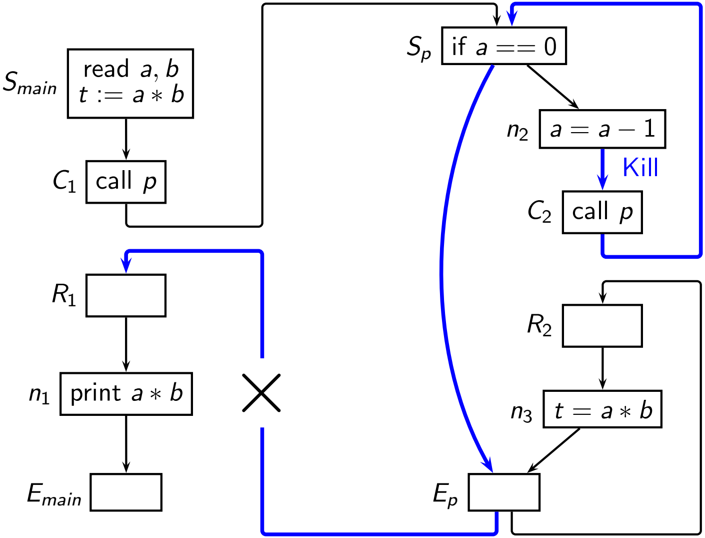
Available Expressions Analysis Using Call Strings Approach



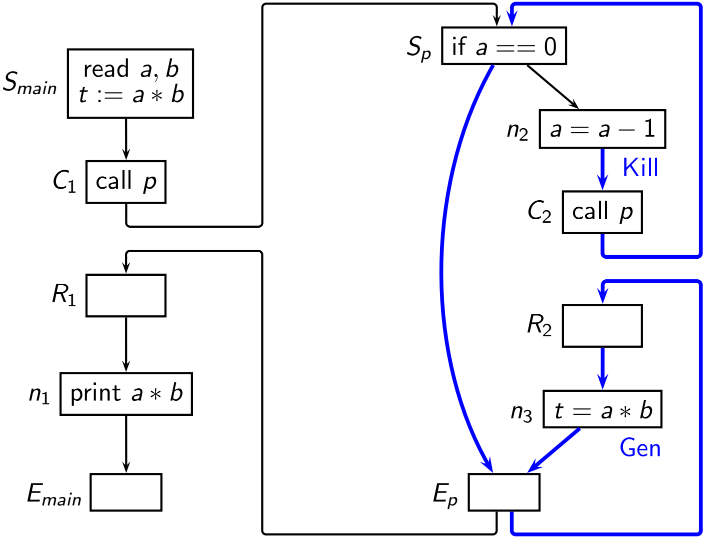
Available Expressions Analysis Using Call Strings Approach



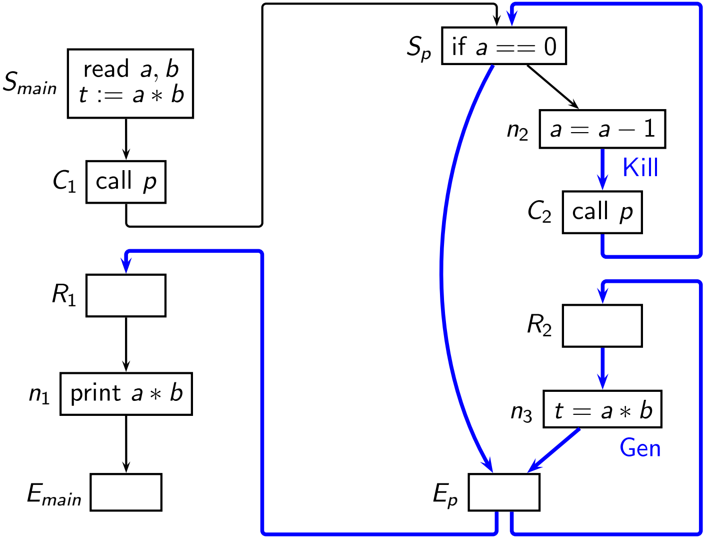
Available Expressions Analysis Using Call Strings Approach



Available Expressions Analysis Using Call Strings Approach

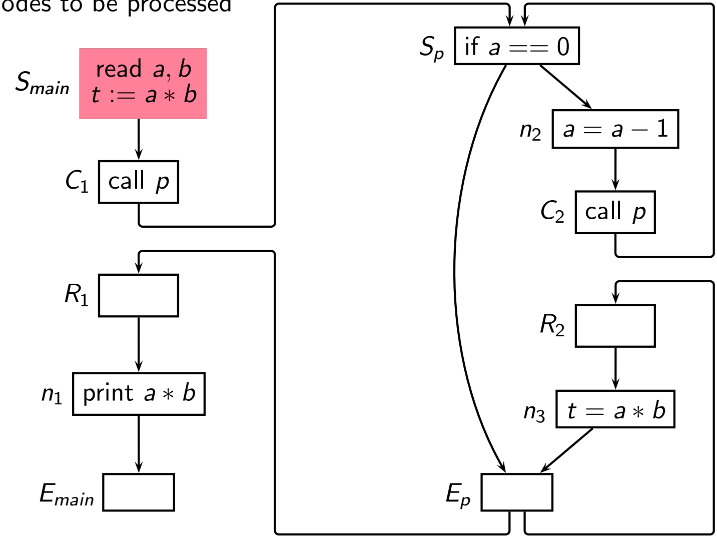


Available Expressions Analysis Using Call Strings Approach



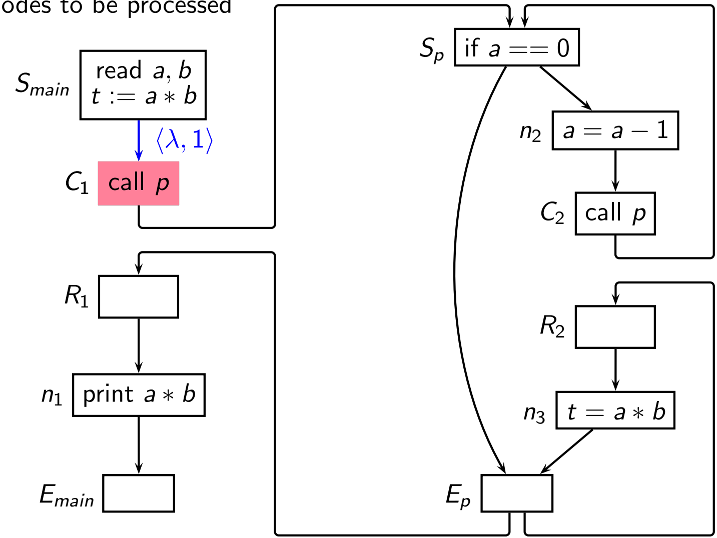
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed



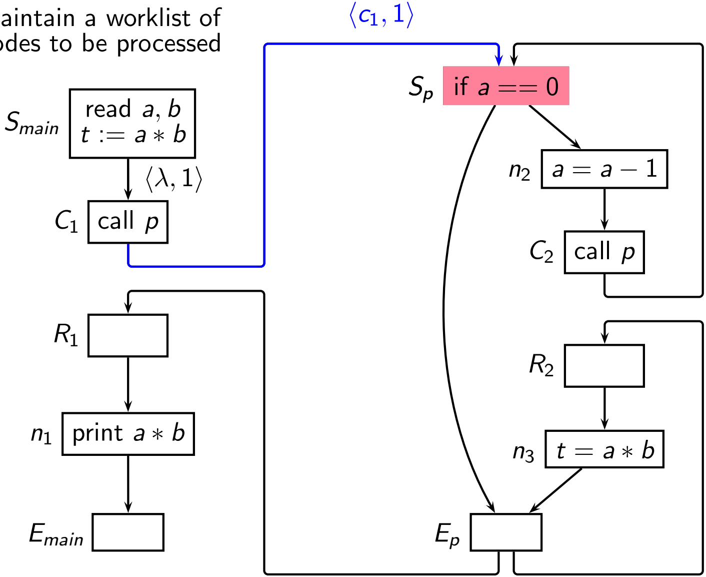
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed



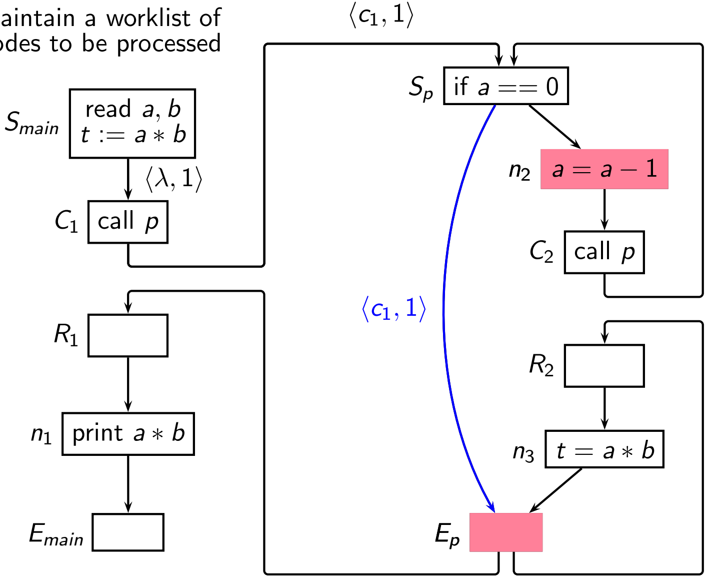
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed



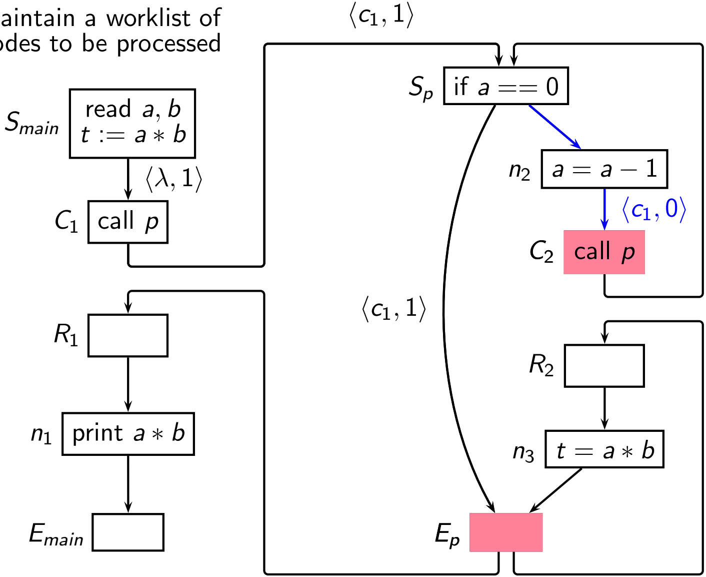
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed



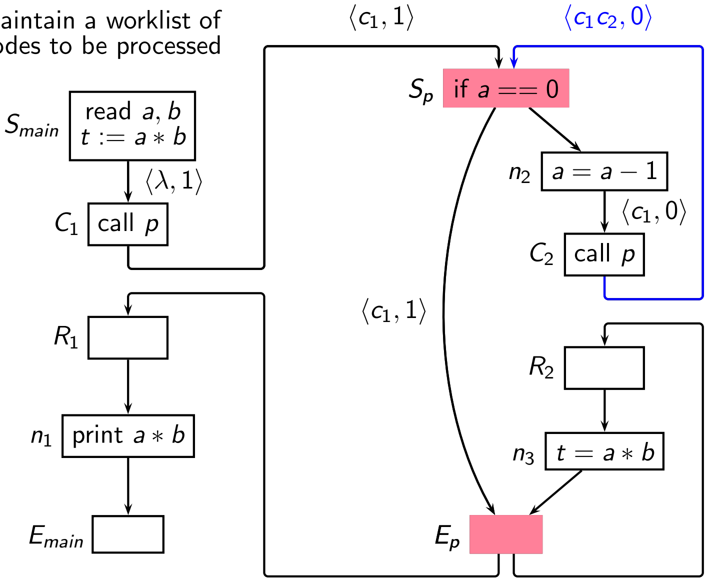
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed



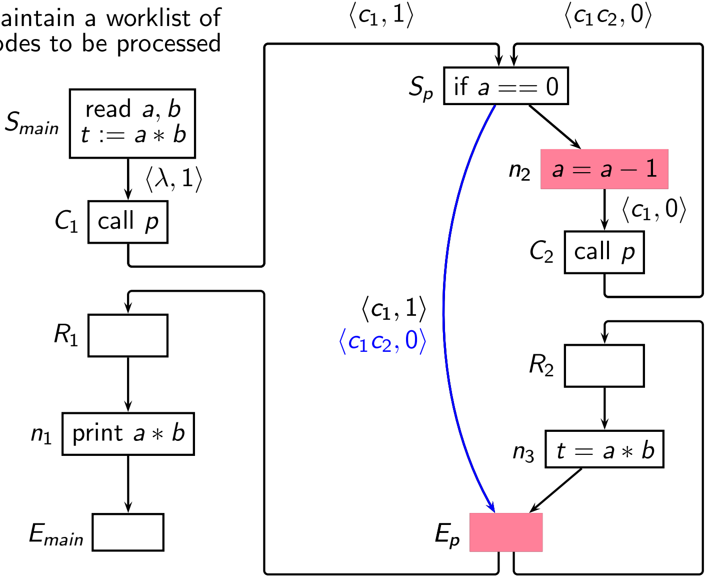
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed



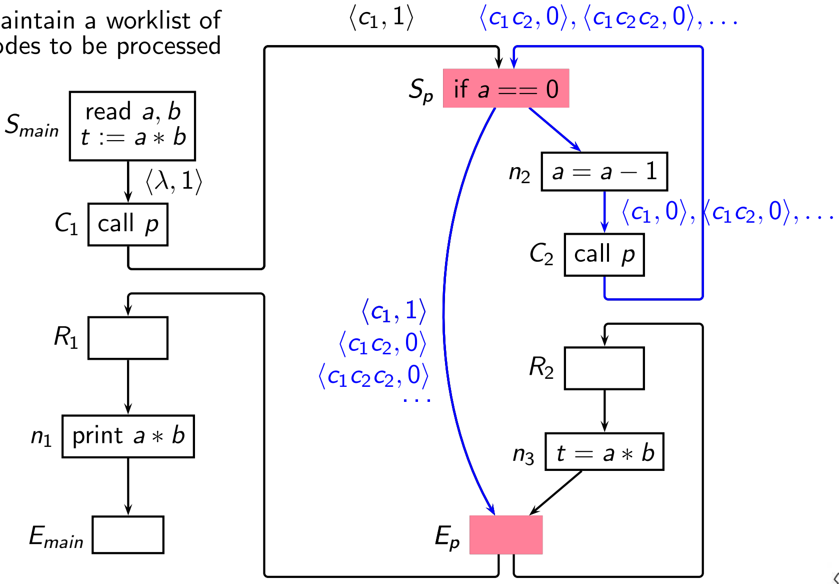
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed



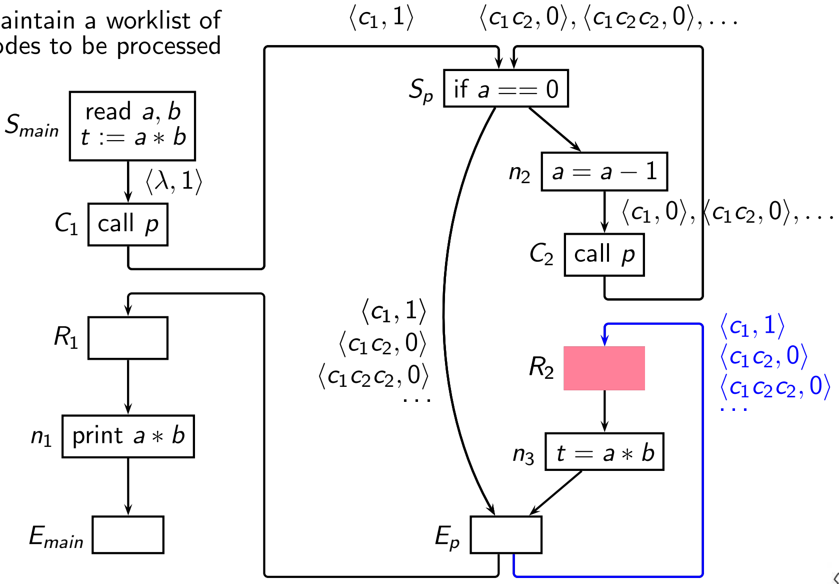
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed


$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle, \langle c_1 c_2 c_2, 0\rangle, \ldots$
S_{main}
read a, b
$t := a * b$
$\langle \lambda, 1\rangle$
C_1
call p
S_p
if $a == 0$
n_2
$a = a - 1$
$\langle c_1, 0\rangle, \langle c_1 c_2, 0\rangle, \ldots$
C_2
call p
R_1
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle$
$\langle c_1 c_2 c_2, 0\rangle$
$\ldots$
R_2
n_1
print $a * b$
n_3
$t = a * b$
E_{main}
E_p

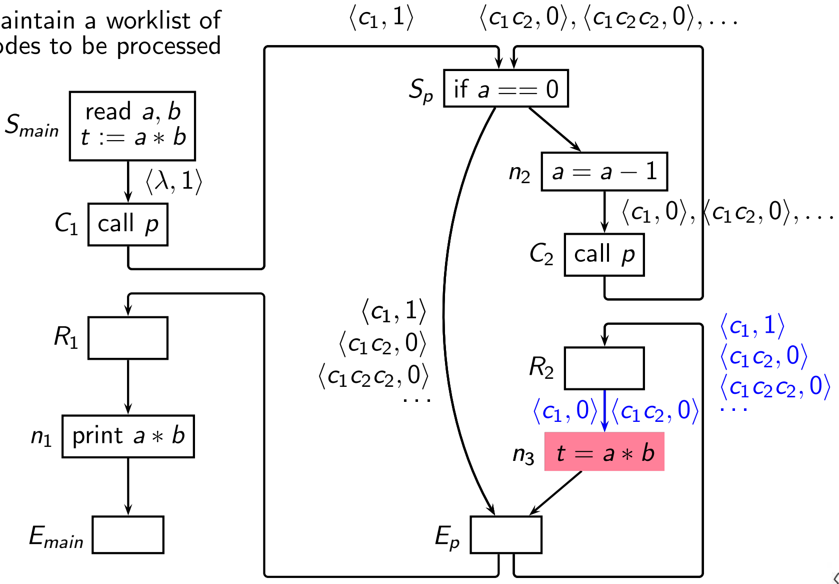
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed


$\langle c_1, 1\rangle$
$\langle c_1c_2, 0\rangle, \langle c_1c_2c_2, 0\rangle, \ldots$
S_p
if $a == 0$
S_{main}
read a, b
$t := a * b$
$\langle \lambda, 1\rangle$
n_2
$a = a - 1$
$\langle c_1, 0\rangle, \langle c_1c_2, 0\rangle, \ldots$
C_1
call p
C_2
call p
R_1
$\langle c_1, 1\rangle$
$\langle c_1c_2, 0\rangle$
$\langle c_1c_2c_2, 0\rangle$
$\ldots$
R_2
$\langle c_1, 1\rangle$
$\langle c_1c_2, 0\rangle$
$\langle c_1c_2c_2, 0\rangle$
$\ldots$
n_1
print $a * b$
n_3
$t = a * b$
E_{main}
E_p

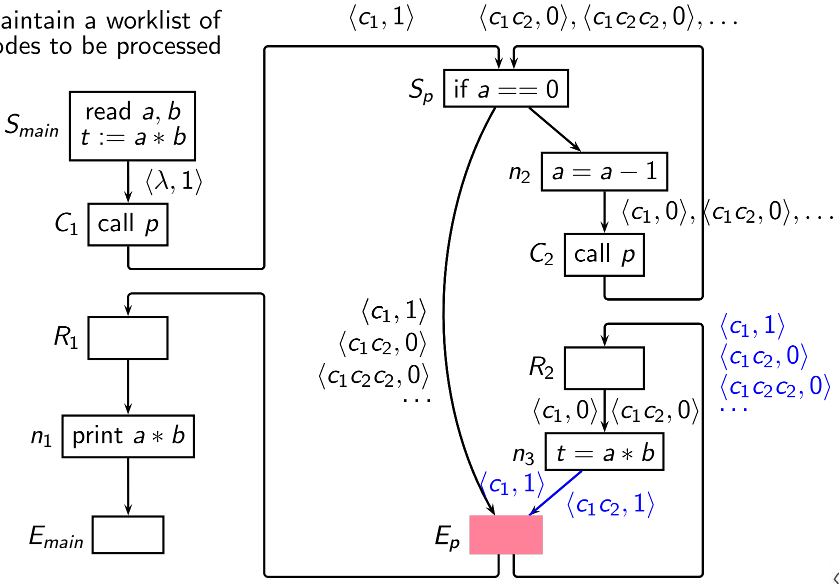
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed


$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle, \langle c_1 c_2 c_2, 0\rangle, \ldots$
S_p
if $a == 0$
S_{main}
read a, b
$t := a * b$
$\langle \lambda, 1\rangle$
C_1
call p
n_2
$a = a - 1$
$\langle c_1, 0\rangle, \langle c_1 c_2, 0\rangle, \ldots$
C_2
call p
R_1
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle$
$\langle c_1 c_2 c_2, 0\rangle$
$\ldots$
R_2
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle$
$\langle c_1 c_2 c_2, 0\rangle$
$\ldots$
$\langle c_1, 0\rangle$
$\langle c_1 c_2, 0\rangle$
n_1
print $a * b$
n_3
$t = a * b$
E_{main}
E_p

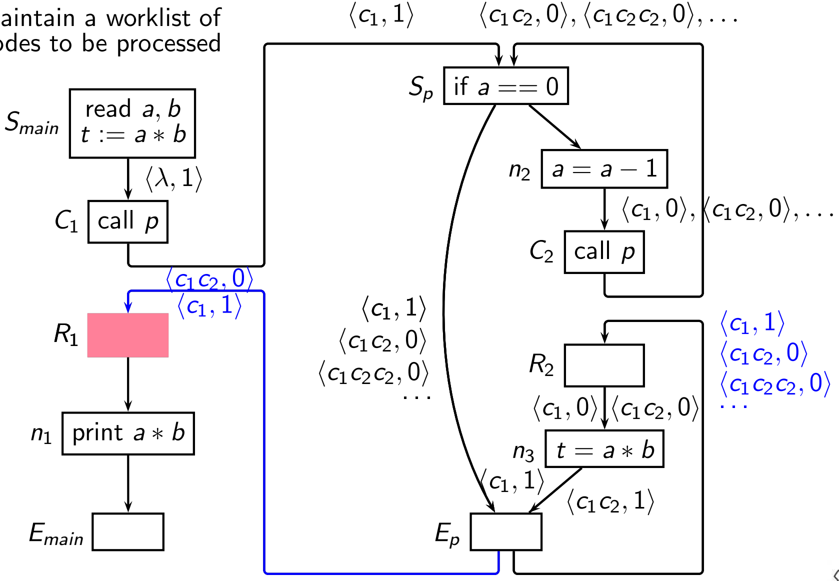
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed


S_{main}
read a, b
$t := a * b$
$\langle \lambda, 1\rangle$
C_1
call p
R_1
n_1
print $a * b$
E_{main}
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle, \langle c_1 c_2 c_2, 0\rangle, \ldots$
S_p
if $a == 0$
n_2
$a = a - 1$
$\langle c_1, 0\rangle, \langle c_1 c_2, 0\rangle, \ldots$
C_2
call p
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle$
$\langle c_1 c_2 c_2, 0\rangle$
$\ldots$
R_2
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle$
$\langle c_1 c_2 c_2, 0\rangle$
$\ldots$
$\langle c_1, 0\rangle$
$\langle c_1 c_2, 0\rangle$
n_3
$t = a * b$
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 1\rangle$
E_p

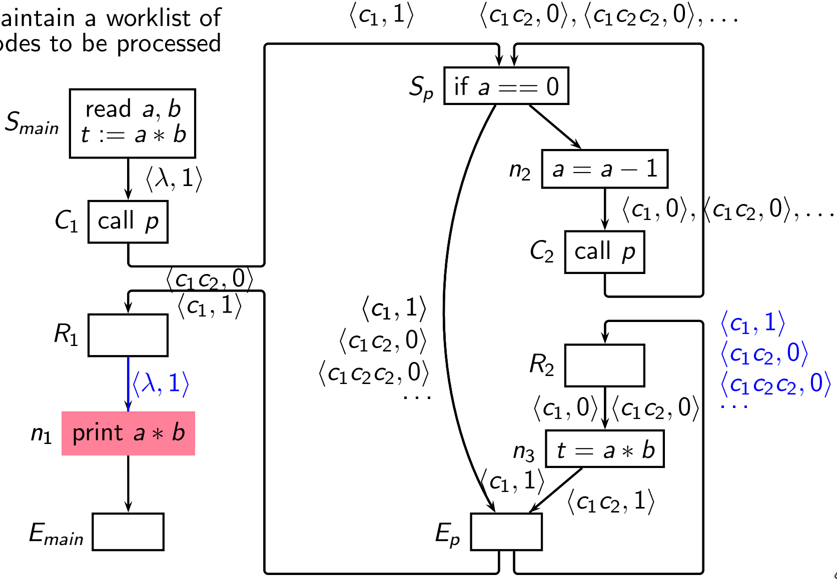
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed


$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle, \langle c_1 c_2 c_2, 0\rangle, \ldots$
S_p
if $a == 0$
S_{main}
read a, b
$t := a * b$
n_2
$a = a - 1$
$\langle \lambda, 1\rangle$
C_1
call p
$\langle c_1, 0\rangle, \langle c_1 c_2, 0\rangle, \ldots$
C_2
call p
$\langle c_1 c_2, 0\rangle$
$\langle c_1, 1\rangle$
R_1
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle$
$\langle c_1 c_2 c_2, 0\rangle$
$\ldots$
R_2
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 0\rangle$
$\langle c_1 c_2 c_2, 0\rangle$
$\ldots$
$\langle c_1, 0\rangle$
$\langle c_1 c_2, 0\rangle$
n_1
print $a * b$
n_3
$t = a * b$
$\langle c_1, 1\rangle$
$\langle c_1 c_2, 1\rangle$
E_{main}
E_p

Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed


$\langle c_1, 1\rangle$
$\langle c_1c_2, 0\rangle, \langle c_1c_2c_2, 0\rangle, \ldots$
S_p
if $a == 0$
S_{main}
read a, b
$t := a * b$
$\langle \lambda, 1\rangle$
n_2
$a = a - 1$
C_1
call p
$\langle c_1, 0\rangle, \langle c_1c_2, 0\rangle, \ldots$
C_2
call p
$\langle c_1c_2, 0\rangle$
$\langle c_1, 1\rangle$
R_1
$\langle c_1, 1\rangle$
$\langle c_1c_2, 0\rangle$
$\langle c_1c_2c_2, 0\rangle$
$\ldots$
$\langle c_1, 1\rangle$
$\langle c_1c_2, 0\rangle$
$\langle c_1c_2c_2, 0\rangle$
$\ldots$
R_2
$\langle \lambda, 1\rangle$
$\langle c_1, 0\rangle$
$\langle c_1c_2, 0\rangle$
n_1
print $a * b$
n_3
$t = a * b$
$\langle c_1, 1\rangle$
$\langle c_1c_2, 1\rangle$
E_{main}
E_p

Tutorial Problem #1

Perform available expressions analysis for the following program

```
main()
{
    a = b*c;

    p();      /* C1 */

    d = b*c;  /* avail b*c? */

    q();      /* C2 */
}
```

```
p()
{
}


q()
{
    b = 5;

    p();   /* C3 */
}
```

The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

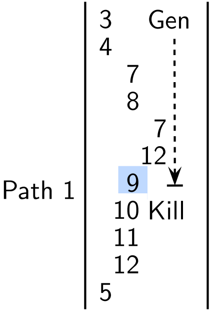
```
int a,b,c;
void main()
{   c = a*b;
    p();
}
void p()
{  if (...)
   { p();
/*Is a*b available?*/
     a = a*b;
   }
}
```

The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

Recursive calls: 1

```
 1. int a,b,c;
 2. void main()
 3. {   c = a*b;
 4.     p();
 5. }
 6. void p()
 7. {  if (...)
 8.    {  p();
 9. /*Is a*b available?*/
10.       a = a*b;
11.    }
12. }
```

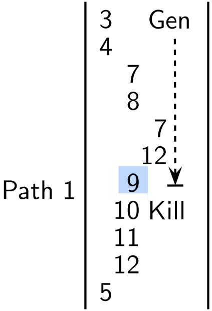


The Need for Multiple Occurrences of a Call Site

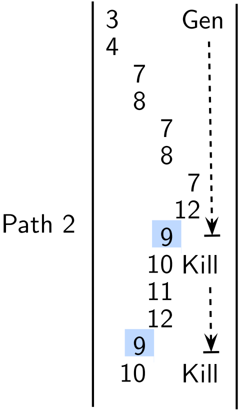
Even if data flow values in cyclic call sequence do not change

```
 1. int a,b,c;
 2. void main()
 3. {   c = a*b;
 4.     p();
 5. }
 6. void p()
 7. {  if (...)
 8.    {  p();
 9. /*Is a*b available?*/
10.       a = a*b;
11.    }
12. }
```

Recursive calls: 1



Recursive calls: 2



The Need for Multiple Occurrences of a Call Site

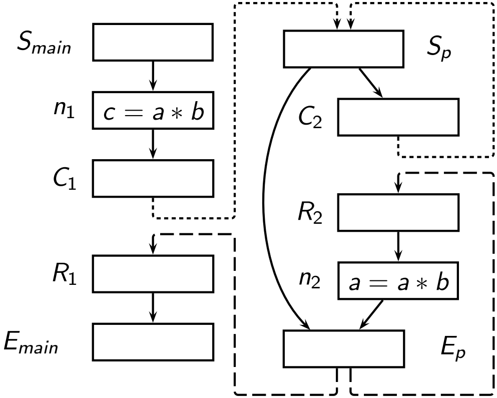
Even if data flow values in cyclic call sequence do not change

```
int a,b,c;
void main()
{   c = a*b;
    p();
}
void p()
{  if (...)
   {  p();
   Is a*b available?
      a = a*b;
   }
}
```

The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

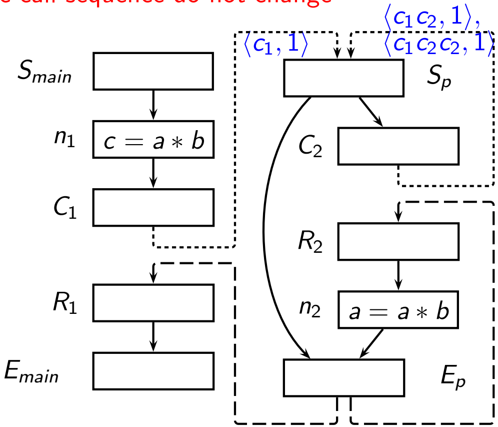
```
int a,b,c;
void main()
{   c = a*b;
    p();
}
void p()
{  if (...)
   {  p();
      Is a*b available?
        a = a*b;
   }
}
```



The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

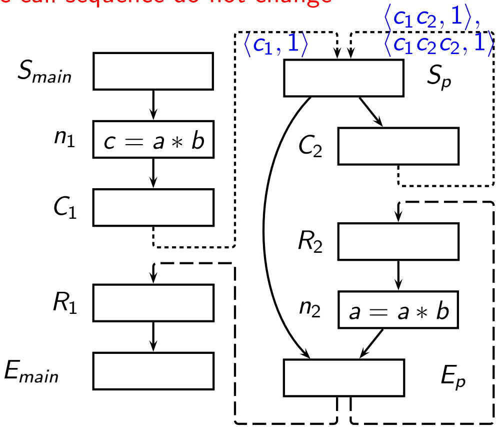
```
int a,b,c;
void main()
{   c = a*b;
    p();
}
void p()
{  if (...)
   {  p();
      Is a*b available?
        a = a*b;
   }
}
```



The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

```
int a,b,c;
void main()
{   c = a*b;
    p();
}
void p()
{  if (...)
   {  p();
      Is a*b available?
         a = a*b;
   }
}
```



- Interprocedurally valid IFP

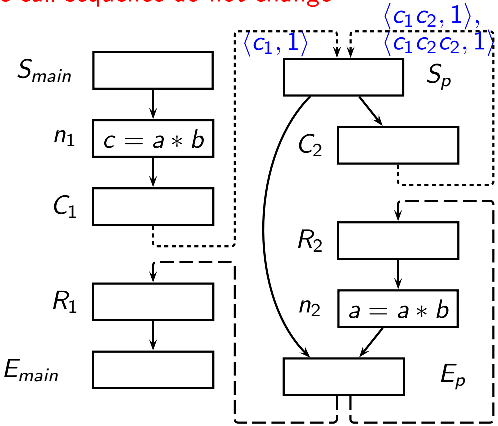
Kill

n_2, E_p, R_2, n_2

The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

```
 1. int a,b,c;
 2. void main()
 3. {   c = a*b;
 4.     p();
 5. }
 6. void p()
 7. {  if (...)
 8.    {  p();
 9.       Is a*b available?
10.         a = a*b;
11.    }
12. }
```



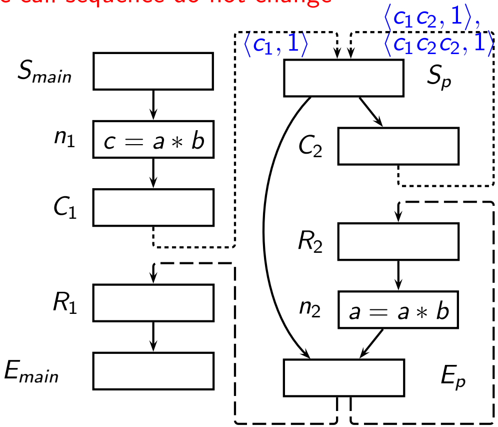
- Interprocedurally valid IFP

$$\underbrace{C_2, S_p, E_p, R_2}, \overset{\text{Kill}}{n_2}, E_p, R_2, n_2$$

The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

```
 1. int a,b,c;
 2. void main()
 3. {   c = a*b;
 4.     p();
 5. }
 6. void p()
 7. {  if (...)
 8.    {  p();
 9.       Is a*b available?
10.          a = a*b;
11.    }
12. }
```



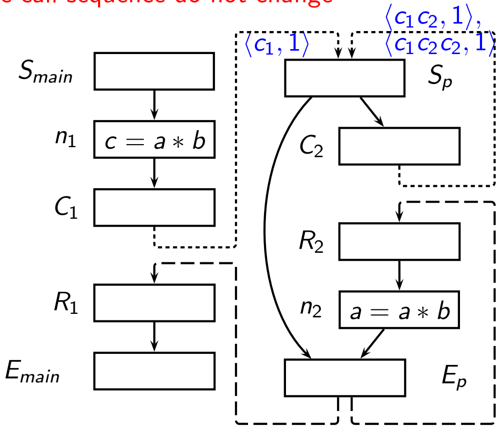
- Interprocedurally valid IFP

$$\underbrace{C_2, S_p, \underbrace{C_2, S_p, E_p, R_2}, \overset{\text{Kill}}{n_2}, E_p, R_2}, n_2$$

The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

```
int a,b,c;
void main()
{   c = a*b;
    p();
}
void p()
{  if (...)
   {  p();
   Is a*b available?
         a = a*b;
   }
}
```



- Interprocedurally valid IFP

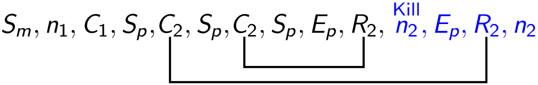
$S_m, n_1, C_1, S_p, C_2, S_p, C_2, S_p, E_p, R_2, \overset{\text{Kill}}{n_2}, E_p, R_2, n_2$

The Need for Multiple Occurrences of a Call Site

Even if data flow values in cyclic call sequence do not change

In terms of staircase diagram

- Interprocedurally valid IFP



$S_m, n_1, C_1, S_p, C_2, S_p, C_2, S_p, E_p, R_2, \overset{\text{Kill}}{n_2}, E_p, R_2, n_2$

The Need for Multiple Occurrences of a Call Site

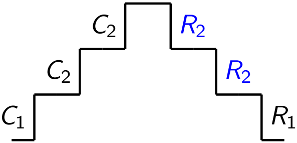
Even if data flow values in cyclic call sequence do not change

In terms of staircase diagram

- Interprocedurally valid IFP

$$S_m, n_1, C_1, S_p, C_2, S_p, C_2, S_p, E_p, R_2, \overset{\text{Kill}}{n_2}, E_p, R_2, n_2$$

- You cannot descend twice, unless you ascend twice



The Need for Multiple Occurrences of a Call Site

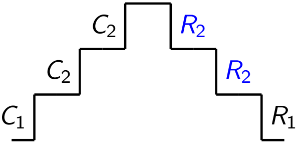
Even if data flow values in cyclic call sequence do not change

In terms of staircase diagram

- Interprocedurally valid IFP

$$S_m, n_1, C_1, S_p, C_2, S_p, C_2, S_p, E_p, R_2, \overset{\text{Kill}}{n_2}, E_p, R_2, n_2$$

- You cannot descend twice, unless you ascend twice



- Even if the data flow values do not change while ascending, you need to ascend because they may change while descending

Tutorial Problem #2

Is a∗b available on line 18 in the following program? On line 15? Construct its supergraph and argue in terms of interprocedurally valid paths

```
1.   main()
2.   {
3.       c = a*b;
4.       p();
5.       a = a*b;
6.   }
```

```
 7.   p()
 8.   {   if (...)
 9.       {   a = a*b;
10.           p();
11.       }
12.       else if (...)
13.       {   c = a * b;
14.           p();
15.           c = a;
16.       }
17.       else
18.           ; /* ignore */
19.   }
```

Terminating Call String Construction

- For non-recursive programs: Number of call strings is finite

Terminating Call String Construction

- For non-recursive programs: Number of call strings is finite
- For recursive programs: Number of call strings could be infinite
  Fortunately, the problem is decidable for finite lattices

Terminating Call String Construction

- For non-recursive programs: Number of call strings is finite
- For recursive programs: Number of call strings could be infinite

  Fortunately, the problem is decidable for finite lattices

  - All call strings upto the following length *must be* constructed

Terminating Call String Construction

- For non-recursive programs: Number of call strings is finite
- For recursive programs: Number of call strings could be infinite

  Fortunately, the problem is decidable for finite lattices

  - All call strings upto the following length *must be* constructed
    - $K \cdot (|L| + 1)^2$ for general bounded frameworks

      (L is the overall lattice of data flow values)

Terminating Call String Construction

- For non-recursive programs: Number of call strings is finite
- For recursive programs: Number of call strings could be infinite

  Fortunately, the problem is decidable for finite lattices

  - All call strings upto the following length *must be* constructed
    - $K \cdot (|L| + 1)^2$ for general bounded frameworks
      (L is the overall lattice of data flow values)
    - $K \cdot (|\widehat{L}| + 1)^2$ for separable bounded frameworks
      ($\widehat{L}$ is the component lattice for an entity)

Terminating Call String Construction

- For non-recursive programs: Number of call strings is finite
- For recursive programs: Number of call strings could be infinite

  Fortunately, the problem is decidable for finite lattices

  - All call strings upto the following length *must be* constructed
    - $K \cdot (|L| + 1)^2$ for general bounded frameworks
      (L is the overall lattice of data flow values)
    - $K \cdot (|\widehat{L}| + 1)^2$ for separable bounded frameworks
      ($\widehat{L}$ is the component lattice for an entity)
    - $K \cdot 3$ for bit vector frameworks

Terminating Call String Construction

- For non-recursive programs: Number of call strings is finite

- For recursive programs: Number of call strings could be infinite

  Fortunately, the problem is decidable for finite lattices

  - All call strings upto the following length *must be* constructed
    - $K \cdot (|L| + 1)^2$ for general bounded frameworks
      (L is the overall lattice of data flow values)
    - $K \cdot (|\widehat{L}| + 1)^2$ for separable bounded frameworks
      ($\widehat{L}$ is the component lattice for an entity)
    - $K \cdot 3$ for bit vector frameworks
    - 3 occurrences of any call site in a call string for bit vector frameworks

⇒ Not a bound but prescribed necessary length

Terminating Call String Construction

- For non-recursive programs: Number of call strings is finite
- For recursive programs: Number of call strings could be infinite

  Fortunately, the problem is decidable for finite lattices

  - All call strings upto the following length *must be* constructed
    - $K \cdot (|L| + 1)^2$ for general bounded frameworks
      (L is the overall lattice of data flow values)
    - $K \cdot (|\widehat{L}| + 1)^2$ for separable bounded frameworks
      ($\widehat{L}$ is the component lattice for an entity)
    - $K \cdot 3$ for bit vector frameworks
    - 3 occurrences of any call site in a call string for bit vector frameworks

⇒ Not a bound but prescribed necessary length

⇒ Large number of long call strings

Classical Call String Length

- Notation
    - $IVP(n, m)$: Interprocedurally valid path from block n to block m
    - $CS(\rho)$: Number of call nodes in ρ that do not have the matching return node in ρ
      (length of the call string representing $IVP(n, m)$)
- Claim

  Let $M = K \cdot (|L| + 1)^2$ where K is the number of distinct call sites in any call chain

  Then, for any $\rho = IVP(S_{main}, m)$ such that

  $$CS(\rho) > M,$$

  $\exists\ \rho' = IVP(S_{main}, m)$ such that

  $$CS(\rho') \leq M, \text{ and } f_{\rho}(BI) = f_{\rho'}(BI)$$

$\Rightarrow$ ρ, the longer path, is redundant for data flow analysis

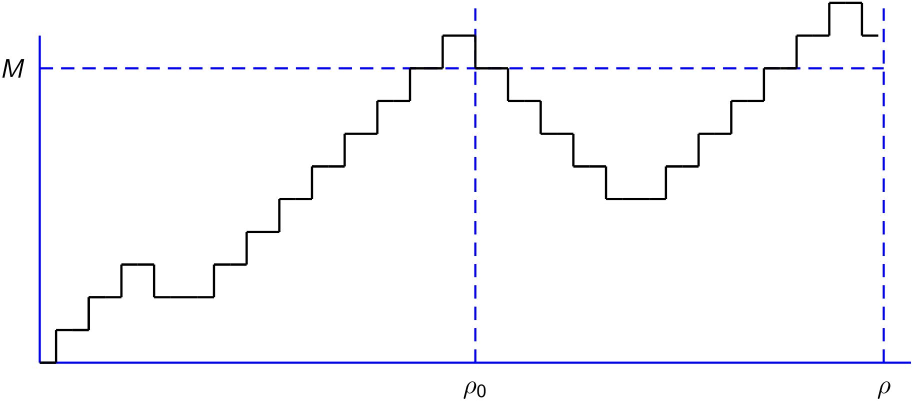
Classical Call String Length

Sharir-Pnueli [1981]

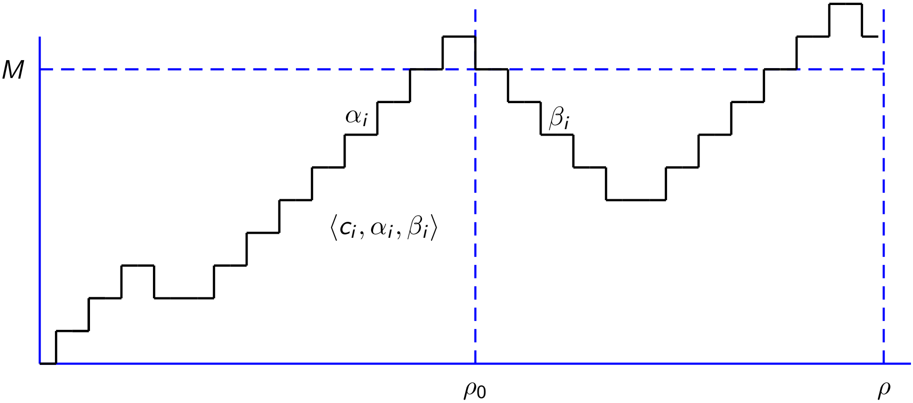
- Consider the smallest prefix ρ_0 of ρ such that $CS(\rho_0) > M$
- Consider a triple $\langle c_i, \alpha_i, \beta_i \rangle$ where
  - α_i is the data flow value reaching call node C_i along ρ and
  - β_i is the data flow value reaching the corresponding return node R_i along ρ

    If R_i is not in ρ, then $\beta_i = \Omega$ (undefined)

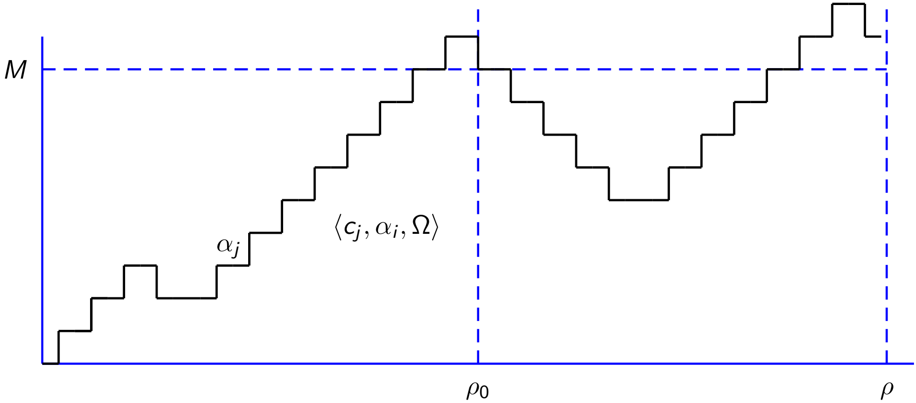
Classical Call String Length



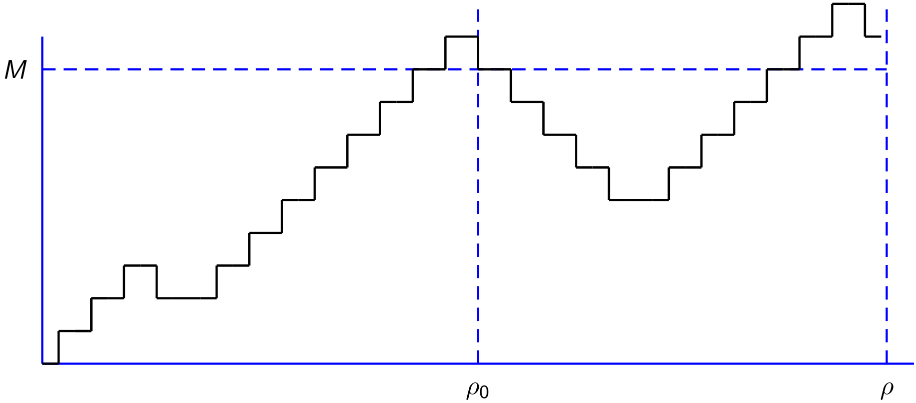
Classical Call String Length



Classical Call String Length

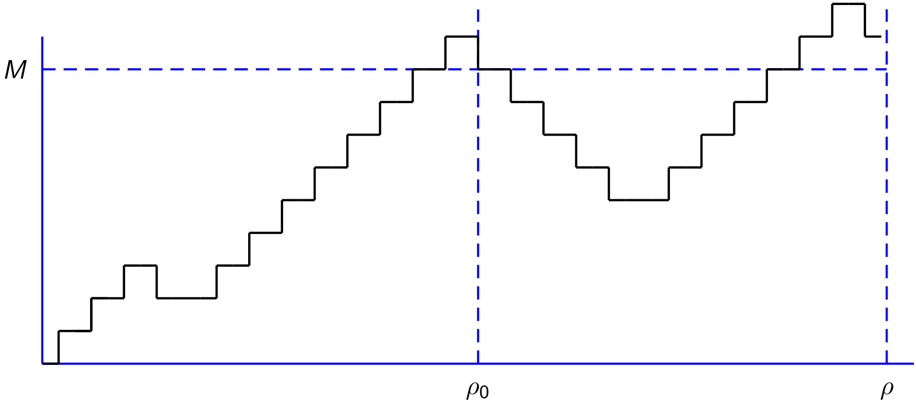


Classical Call String Length



- Number of distinct triples $\langle c_i, \alpha_i, \beta_i \rangle$ is $M = K \cdot (|L| + 1)^2$.

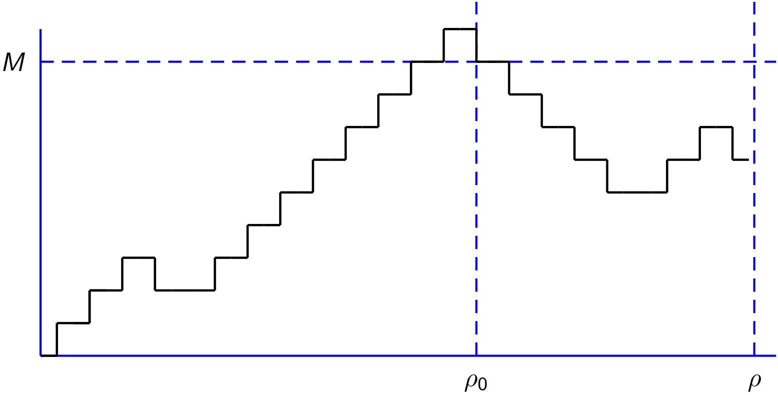
Classical Call String Length



- Number of distinct triples $\langle c_i, \alpha_i, \beta_i \rangle$ is $M = K \cdot (|L| + 1)^2$.
- There are at least two calls from the same call site that have the same effect on data flow values

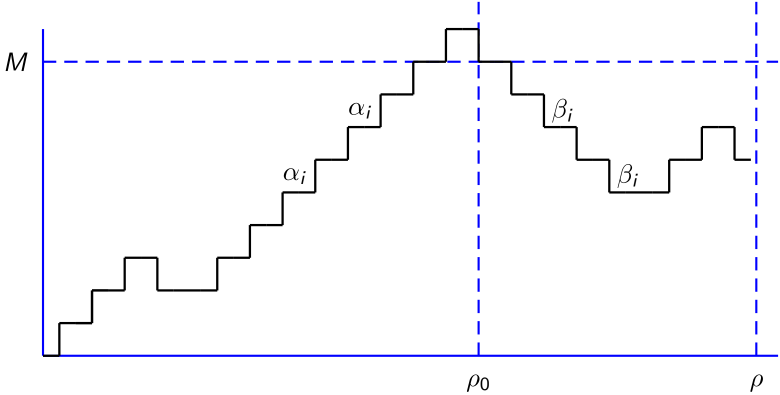
Classical Call String Length

When β_i is not Ω



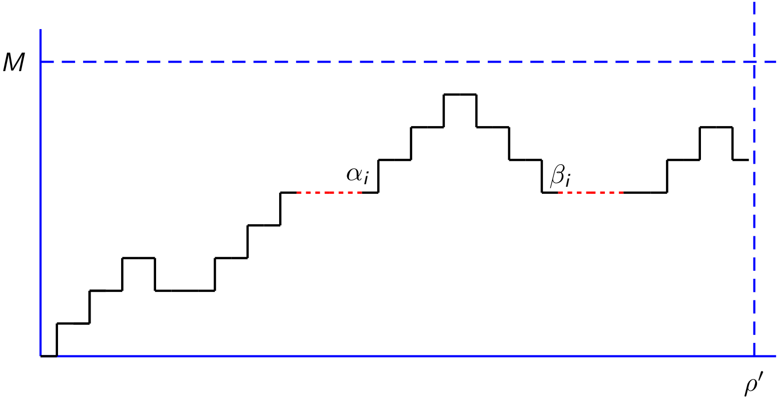
Classical Call String Length

When β_i is not Ω



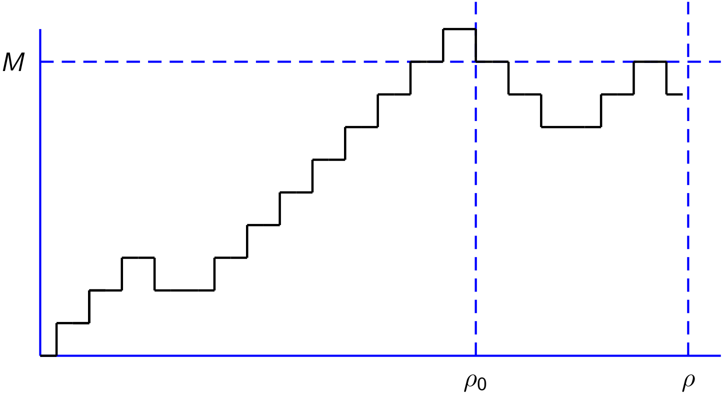
Classical Call String Length

When β_i is not Ω



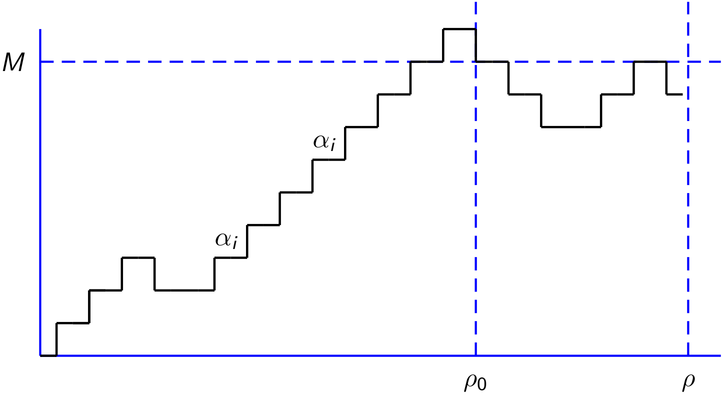
Classical Call String Length

When β_i is Ω



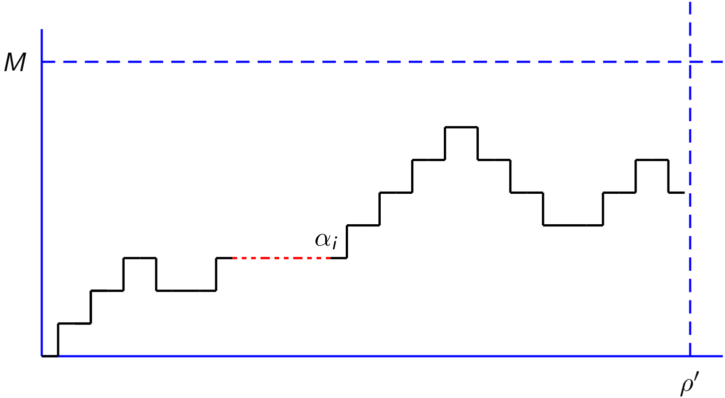
Classical Call String Length

When β_i is Ω



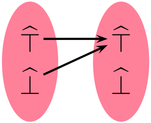
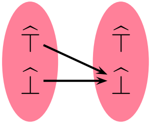
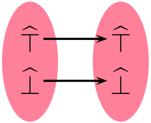
Classical Call String Length

When β_i is Ω



Tighter Bound for Bit Vector Frameworks

- $\widehat{L}$ is $\{0,1\}$, L is $\{0,1\}^m$
- $\widehat{\sqcap}$ is either boolean AND or boolean OR
- $\widehat{\top}$ and $\widehat{\bot}$ are 0 or 1 depending on $\widehat{\sqcap}$.
- $\widehat{h}$ is a *bit function* and could be one of the following:

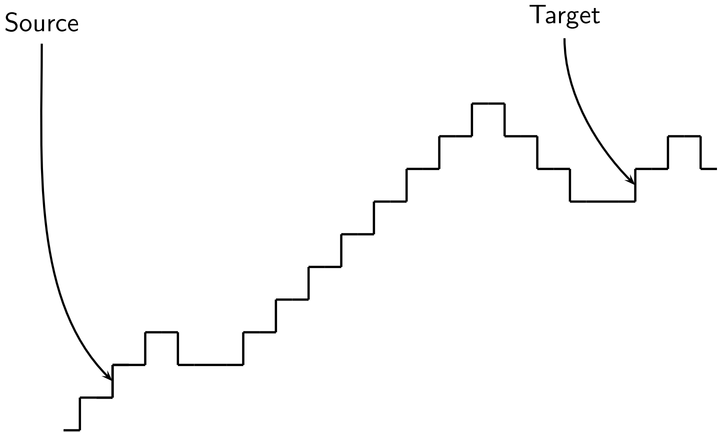
Raise	*Lower*	*Propagate*
 $\widehat{\top}$ $\widehat{\bot}$ $\widehat{\top}$ $\widehat{\bot}$	 $\widehat{\top}$ $\widehat{\bot}$ $\widehat{\top}$ $\widehat{\bot}$	 $\widehat{\top}$ $\widehat{\bot}$ $\widehat{\top}$ $\widehat{\bot}$

Tighter Bound for Bit Vector Frameworks

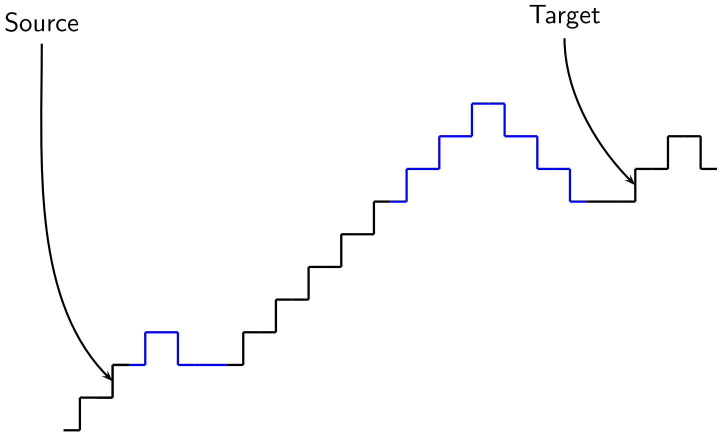
Karkare Khedker 2007

- Validity constraints are imposed by the presence of return nodes
- For every cyclic path consisting on Propagate functions, there exists an acyclic path consisting of Propagate functions
- Source of information is a Raise or Lower function
- Target is a point reachable by a series of Propagate functions
- Identifies interesting path segments that we need to consider for determining a sufficient set of call strings

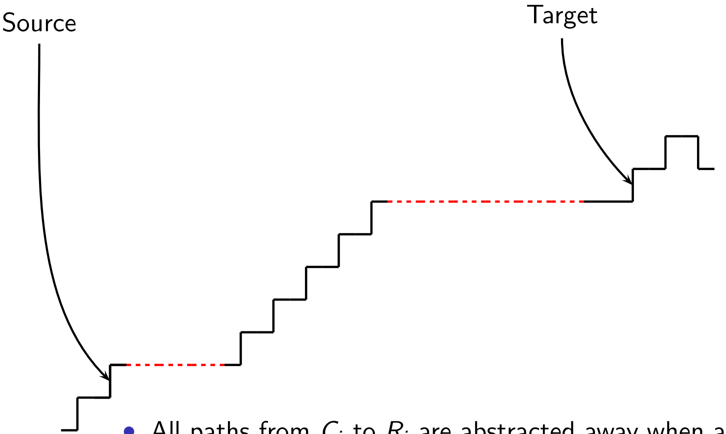
Relevant Path Segments for Tigher Bound for Bit Vector Frameworks



Relevant Path Segments for Tigher Bound for Bit Vector Frameworks



Relevant Path Segments for Tigher Bound for Bit Vector Frameworks



- All paths from C_i to R_i are abstracted away when a call node C_j is reached after R_i
- Consider maximal interprocedurally valid paths in which there is no path from a return node to a call node

Relevant Path Segments for Tigher Bound for Bit Vector Frameworks

Consider all four combinations

- Case A: Source is a call node and target is a call node
- Case B: Source is a call node and target is a return node
- Case C: Source is a return node and target is also a return node
- Case D: Source is a return node and target is a call node:
  Not relevant

Tighter Length for Bit Vector Frameworks

Case A:
Source is a call node and target is also a call node $P(I_{\rightsquigarrow} C_S \rightsquigarrow C_T)$

- No return node, no validity constraints
- Paths $P(I_{\rightsquigarrow} C_S)$ and Paths $P(C_S \rightsquigarrow C_T)$ can be acyclic
- A call node may be common to both segments
- At most 2 occurrences of a call site

Tighter Length for Bit Vector Frameworks

Case B:

Source is a call node C_S and target is some return node R_T

Tighter Length for Bit Vector Frameworks

Case B:
Source is a call node C_S and target is some return node R_T

- $P(I_{\rightsquigarrow} C_S \rightsquigarrow \textcircled{C_T} \rightsquigarrow R_T)$

- $P(I_{\rightsquigarrow} \textcircled{C_T} \rightsquigarrow C_S \rightsquigarrow R_S \rightsquigarrow R_T)$

Tighter Length for Bit Vector Frameworks

Case B:

Source is a call node C_S and target is some return node R_T

- $P(I_{\rightsquigarrow} C_S \rightsquigarrow \enclose{circle}{C_T} \rightsquigarrow R_T)$
  - Call strings are derived from the paths $P(I_{\rightsquigarrow} C_S \rightsquigarrow C_T \rightsquigarrow C_L)$ where C_L is the last call node

Tighter Length for Bit Vector Frameworks

Case B:
Source is a call node C_S and target is some return node R_T

- $P(I_{\rightsquigarrow} C_S \rightsquigarrow \textcircled{C_T} \rightsquigarrow R_T)$
  - Call strings are derived from the paths $P(I_{\rightsquigarrow} C_S \rightsquigarrow C_T \rightsquigarrow C_L)$ where C_L is the last call node
  - Thus there are three acyclic segments $P(I_{\rightsquigarrow} C_S)$, $P(C_S \rightsquigarrow C_T)$, and $P(C_T \rightsquigarrow C_L)$

Tighter Length for Bit Vector Frameworks

Case B:
Source is a call node C_S and target is some return node R_T

- $P(I_{\rightsquigarrow} C_S \rightsquigarrow \enclose{circle}{C_T} \rightsquigarrow R_T)$
  - Call strings are derived from the paths $P(I_{\rightsquigarrow} C_S \rightsquigarrow C_T \rightsquigarrow C_L)$ where C_L is the last call node
  - Thus there are three acyclic segments $P(I_{\rightsquigarrow} C_S)$, $P(C_S \rightsquigarrow C_T)$, and $P(C_T \rightsquigarrow C_L)$
  - A call node may be shared in all three
    $\Rightarrow$ At most 3 occurrences of a call site

Tighter Length for Bit Vector Frameworks

Case B:
Source is a call node C_S and target is some return node R_T

- $P(I_{\rightsquigarrow} C_S \rightsquigarrow \enclose{circle}{C_T} \rightsquigarrow R_T)$
  - Call strings are derived from the paths $P(I_{\rightsquigarrow} C_S \rightsquigarrow C_T \rightsquigarrow C_L)$ where C_L is the last call node
  - Thus there are three acyclic segments $P(I_{\rightsquigarrow} C_S)$, $P(C_S \rightsquigarrow C_T)$, and $P(C_T \rightsquigarrow C_L)$
  - A call node may be shared in all three
    $\Rightarrow$ At most 3 occurrences of a call site
- $P(I_{\rightsquigarrow} \enclose{circle}{C_T} \rightsquigarrow C_S \rightsquigarrow R_S \rightsquigarrow R_T)$
  - C_T is required because of validity constraints

Tighter Length for Bit Vector Frameworks

Case B:
Source is a call node C_S and target is some return node R_T

- $P(I_{\rightsquigarrow} C_S \rightsquigarrow \enclose{circle}{C_T} \rightsquigarrow R_T)$
  - Call strings are derived from the paths $P(I_{\rightsquigarrow} C_S \rightsquigarrow C_T \rightsquigarrow C_L)$ where C_L is the last call node
  - Thus there are three acyclic segments $P(I_{\rightsquigarrow} C_S)$, $P(C_S \rightsquigarrow C_T)$, and $P(C_T \rightsquigarrow C_L)$
  - A call node may be shared in all three $\Rightarrow$ At most 3 occurrences of a call site
- $P(I_{\rightsquigarrow} \enclose{circle}{C_T} \rightsquigarrow C_S \rightsquigarrow R_S \rightsquigarrow R_T)$
  - C_T is required because of validity constraints
  - Call strings are derived from the paths $P(I_{\rightsquigarrow} C_T \rightsquigarrow C_S \rightsquigarrow C_L)$ where C_L is the last call node

Tighter Length for Bit Vector Frameworks

Case B:
Source is a call node C_S and target is some return node R_T

- $P(I_{\leadsto} C_S \leadsto \textcircled{C_T} \leadsto R_T)$
  - Call strings are derived from the paths $P(I_{\leadsto} C_S \leadsto C_T \leadsto C_L)$ where C_L is the last call node
  - Thus there are three acyclic segments $P(I_{\leadsto} C_S)$, $P(C_S \leadsto C_T)$, and $P(C_T \leadsto C_L)$
  - A call node may be shared in all three
    $\Rightarrow$ At most 3 occurrences of a call site
- $P(I_{\leadsto} \textcircled{C_T} \leadsto C_S \leadsto R_S \leadsto R_T)$
  - C_T is required because of validity constraints
  - Call strings are derived from the paths $P(I_{\leadsto} C_T \leadsto C_S \leadsto C_L)$ where C_L is the last call node
  - Again, there are three acyclic segments and at most 3 occurrences of a call site

Tighter Length for Bit Vector Frameworks

Case C:
Source is a return node R_S and target is also some return node R_T

- $P(I_{\rightsquigarrow} C_T \rightsquigarrow C_S \rightsquigarrow R_S \rightsquigarrow R_T)$
- C_T and C_S are required because of validity constraints
- Call strings are derived from the paths $P(I_{\rightsquigarrow} C_T \rightsquigarrow C_S \rightsquigarrow C_L)$ where C_L is the last call node
- Again, there are three acyclic segments and at most 3 occurrences of a call site

Classical Approximate Call Strings Approach

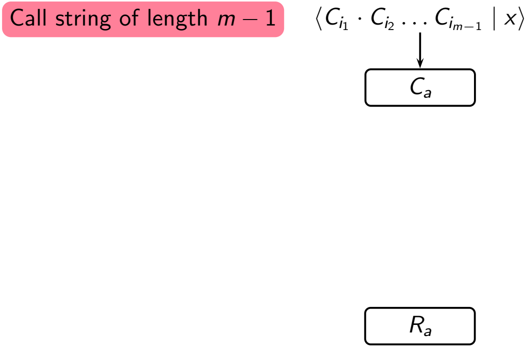
- Maintain call string suffixes of upto a given length m

C_a

R_a

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m

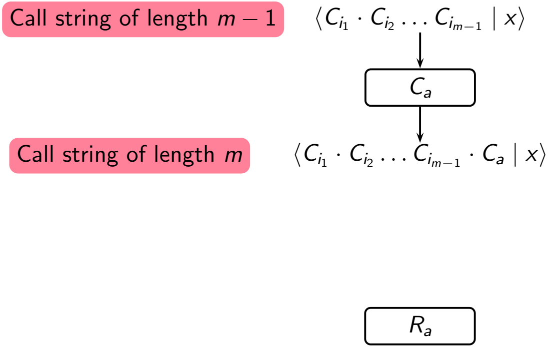


Call string of length $m-1$

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \mid x \rangle$

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



Call string of length $m - 1$ $\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \mid x \rangle$

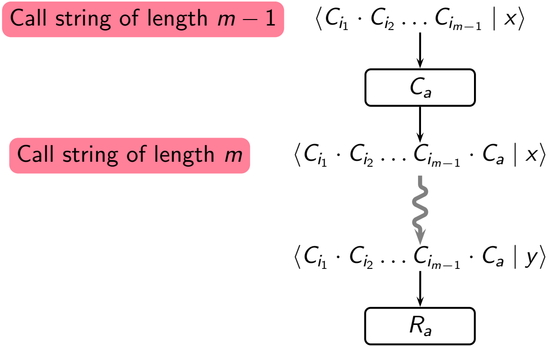
C_a

Call string of length m $\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \cdot C_a \mid x \rangle$

R_a

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



Call string of length $m-1$ $\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \mid x \rangle$

C_a

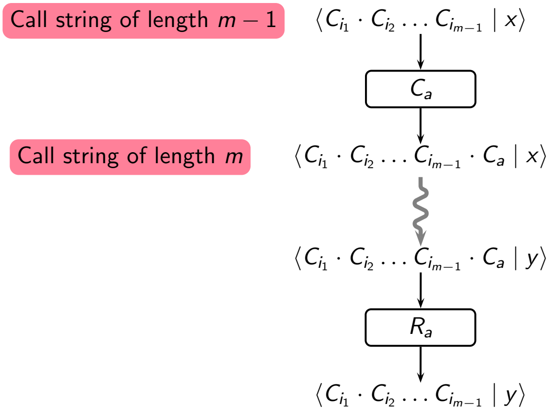
Call string of length m $\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \cdot C_a \mid x \rangle$

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \cdot C_a \mid y \rangle$

R_a

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



Call string of length $m - 1$

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \mid x \rangle$

C_a

Call string of length m

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \cdot C_a \mid x \rangle$

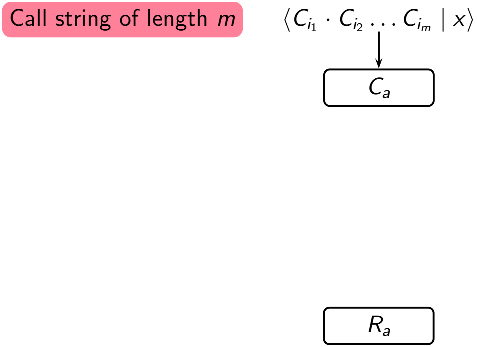
$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \cdot C_a \mid y \rangle$

R_a

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_{m-1}} \mid y \rangle$

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m

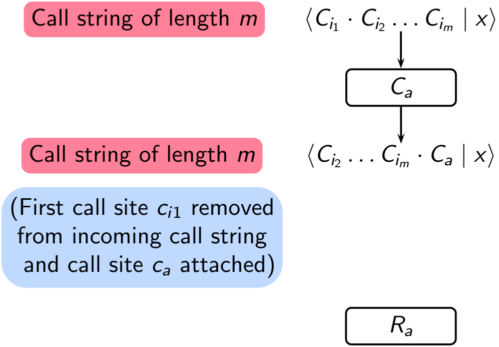


Call string of length m

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x \rangle$

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



Call string of length m

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x \rangle$

C_a

Call string of length m

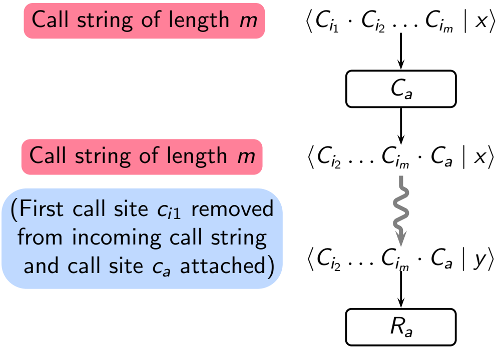
$\langle C_{i_2} \ldots C_{i_m} \cdot C_a \mid x \rangle$

(First call site c_{i1} removed from incoming call string and call site c_a attached)

R_a

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



Call string of length m

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x \rangle$

Call string of length m

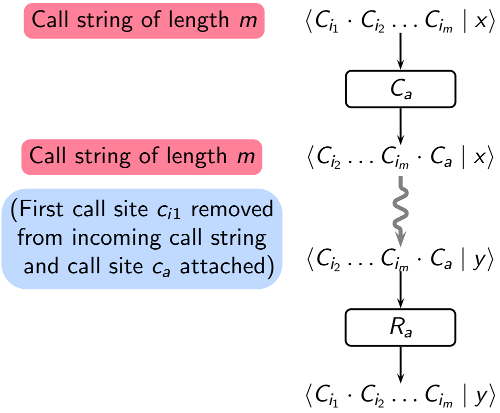
$\langle C_{i_2} \ldots C_{i_m} \cdot C_a \mid x \rangle$

(First call site c_{i1} removed from incoming call string and call site c_a attached)

$\langle C_{i_2} \ldots C_{i_m} \cdot C_a \mid y \rangle$

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



Call string of length m

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x \rangle$

C_a

Call string of length m

$\langle C_{i_2} \ldots C_{i_m} \cdot C_a \mid x \rangle$

(First call site c_{i1} removed from incoming call string and call site c_a attached)

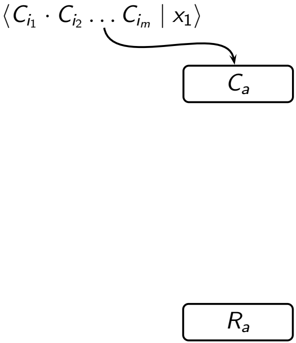
$\langle C_{i_2} \ldots C_{i_m} \cdot C_a \mid y \rangle$

R_a

$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid y \rangle$

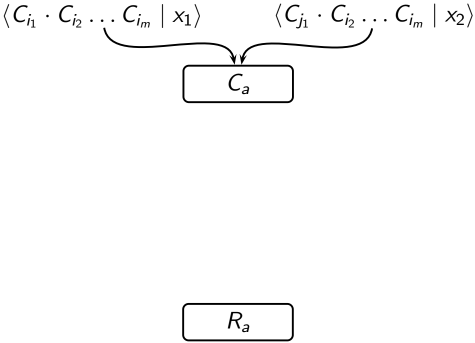
Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m


$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_1 \rangle$
C_a
R_a

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length *m*



$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_1 \rangle$

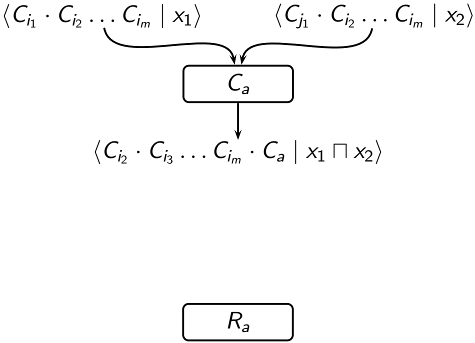
$\langle C_{j_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_2 \rangle$

C_a

R_a

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



$\langle C_{i_1} \cdot C_{i_2} \dots C_{i_m} \mid x_1 \rangle$

$\langle C_{j_1} \cdot C_{i_2} \dots C_{i_m} \mid x_2 \rangle$

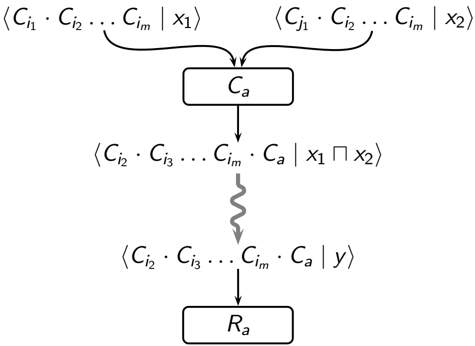
C_a

$\langle C_{i_2} \cdot C_{i_3} \dots C_{i_m} \cdot C_a \mid x_1 \sqcap x_2 \rangle$

R_a

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_1 \rangle$

$\langle C_{j_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_2 \rangle$

C_a

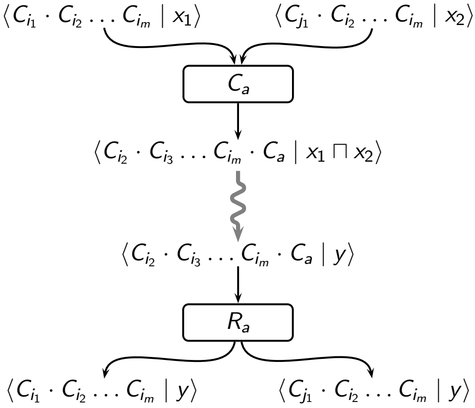
$\langle C_{i_2} \cdot C_{i_3} \ldots C_{i_m} \cdot C_a \mid x_1 \sqcap x_2 \rangle$

$\langle C_{i_2} \cdot C_{i_3} \ldots C_{i_m} \cdot C_a \mid y \rangle$

R_a

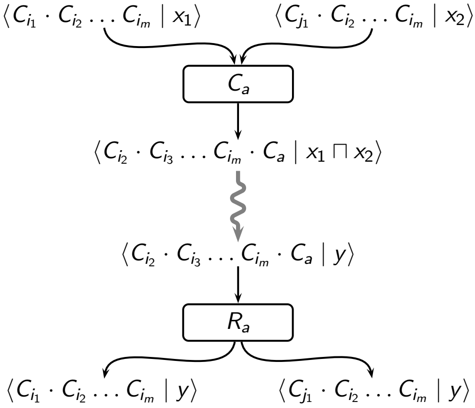
Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m


$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_1 \rangle$
$\langle C_{j_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_2 \rangle$
C_a
$\langle C_{i_2} \cdot C_{i_3} \ldots C_{i_m} \cdot C_a \mid x_1 \sqcap x_2 \rangle$
$\langle C_{i_2} \cdot C_{i_3} \ldots C_{i_m} \cdot C_a \mid y \rangle$
R_a
$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid y \rangle$
$\langle C_{j_1} \cdot C_{i_2} \ldots C_{i_m} \mid y \rangle$

Classical Approximate Call Strings Approach

- Maintain call string suffixes of upto a given length m



$\langle C_{i_1} \cdot C_{i_2} \dots C_{i_m} \mid x_1 \rangle$

$\langle C_{j_1} \cdot C_{i_2} \dots C_{i_m} \mid x_2 \rangle$

C_a

$\langle C_{i_2} \cdot C_{i_3} \dots C_{i_m} \cdot C_a \mid x_1 \sqcap x_2 \rangle$

$\langle C_{i_2} \cdot C_{i_3} \dots C_{i_m} \cdot C_a \mid y \rangle$

R_a

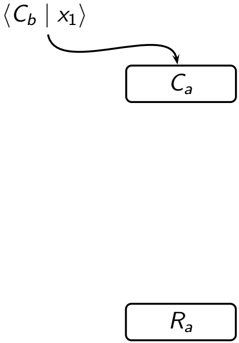
$\langle C_{i_1} \cdot C_{i_2} \dots C_{i_m} \mid y \rangle$

$\langle C_{j_1} \cdot C_{i_2} \dots C_{i_m} \mid y \rangle$

- Practical choices of m have been 1 or 2

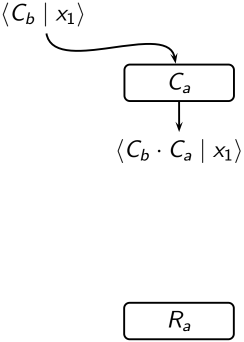
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$



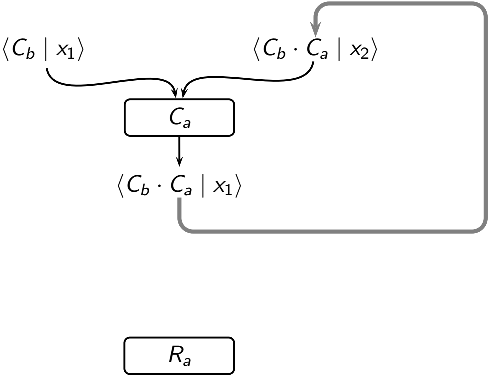
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$


$\langle C_b \mid x_1 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle$
R_a

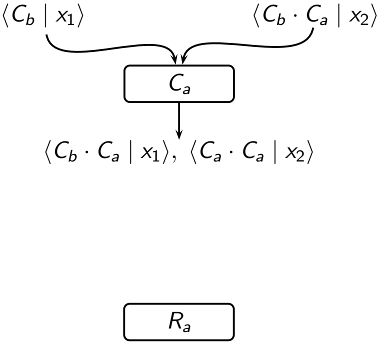
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$


$\langle C_b \mid x_1 \rangle$
$\langle C_b \cdot C_a \mid x_2 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle$
R_a

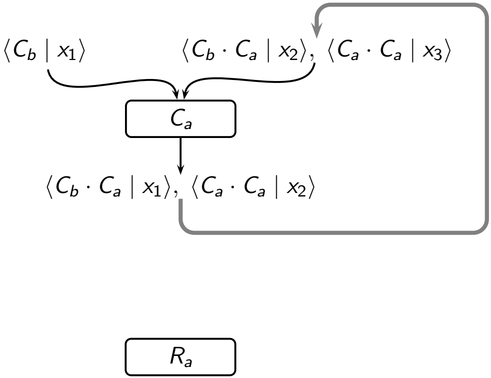
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$


$\langle C_b \mid x_1 \rangle$
$\langle C_b \cdot C_a \mid x_2 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle, \langle C_a \cdot C_a \mid x_2 \rangle$
R_a

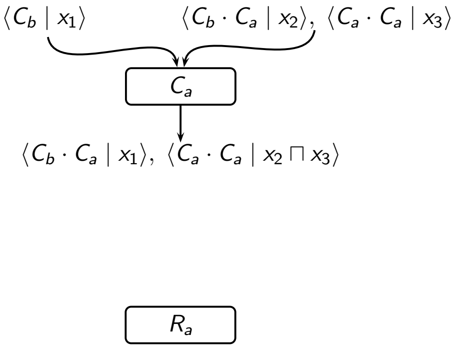
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$


$\langle C_b \mid x_1 \rangle$
$\langle C_b \cdot C_a \mid x_2 \rangle$, $\langle C_a \cdot C_a \mid x_3 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle$, $\langle C_a \cdot C_a \mid x_2 \rangle$
R_a

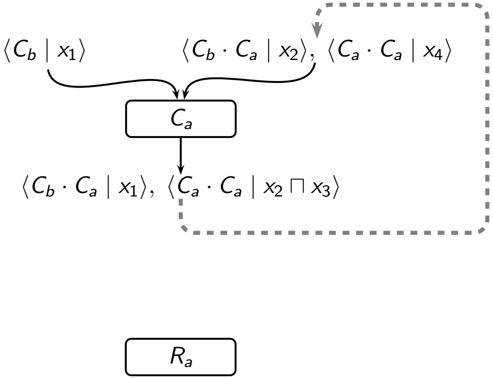
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$


$\langle C_b \mid x_1 \rangle$
$\langle C_b \cdot C_a \mid x_2 \rangle, \langle C_a \cdot C_a \mid x_3 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle, \langle C_a \cdot C_a \mid x_2 \sqcap x_3 \rangle$
R_a

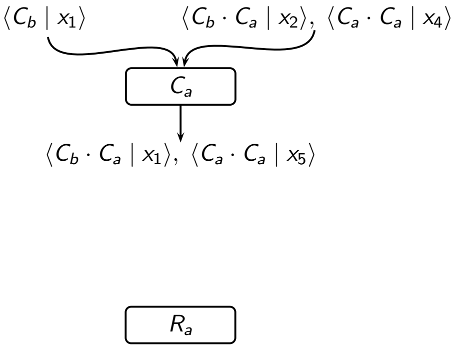
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$


$\langle C_b \mid x_1 \rangle$
$\langle C_b \cdot C_a \mid x_2 \rangle$, $\langle C_a \cdot C_a \mid x_4 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle$, $\langle C_a \cdot C_a \mid x_2 \sqcap x_3 \rangle$
R_a

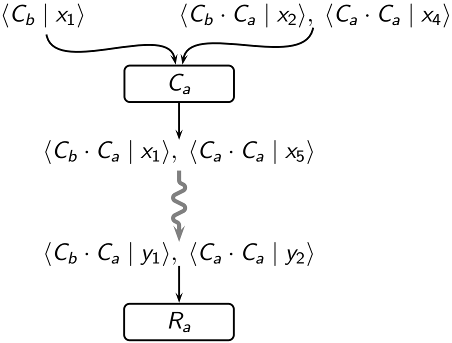
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$


$\langle C_b \mid x_1 \rangle$
$\langle C_b \cdot C_a \mid x_2 \rangle, \langle C_a \cdot C_a \mid x_4 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle, \langle C_a \cdot C_a \mid x_5 \rangle$
R_a

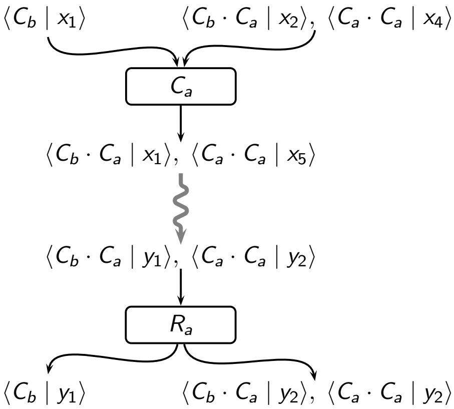
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$


$\langle C_b \mid x_1 \rangle$
$\langle C_b \cdot C_a \mid x_2 \rangle$, $\langle C_a \cdot C_a \mid x_4 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle$, $\langle C_a \cdot C_a \mid x_5 \rangle$
$\langle C_b \cdot C_a \mid y_1 \rangle$, $\langle C_a \cdot C_a \mid y_2 \rangle$
R_a

Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$



$\langle C_b \mid x_1 \rangle$

$\langle C_b \cdot C_a \mid x_2 \rangle, \langle C_a \cdot C_a \mid x_4 \rangle$

C_a

$\langle C_b \cdot C_a \mid x_1 \rangle, \langle C_a \cdot C_a \mid x_5 \rangle$

$\langle C_b \cdot C_a \mid y_1 \rangle, \langle C_a \cdot C_a \mid y_2 \rangle$

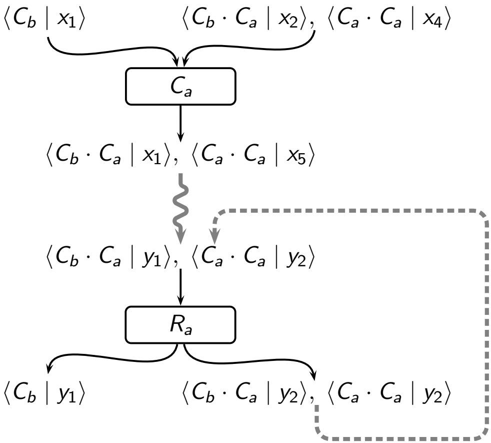
R_a

$\langle C_b \mid y_1 \rangle$

$\langle C_b \cdot C_a \mid y_2 \rangle, \langle C_a \cdot C_a \mid y_2 \rangle$

Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$



$\langle C_b \mid x_1 \rangle$
$\langle C_b \cdot C_a \mid x_2 \rangle, \langle C_a \cdot C_a \mid x_4 \rangle$
C_a
$\langle C_b \cdot C_a \mid x_1 \rangle, \langle C_a \cdot C_a \mid x_5 \rangle$
$\langle C_b \cdot C_a \mid y_1 \rangle, \langle C_a \cdot C_a \mid y_2 \rangle$
R_a
$\langle C_b \mid y_1 \rangle$
$\langle C_b \cdot C_a \mid y_2 \rangle, \langle C_a \cdot C_a \mid y_2 \rangle$

Part 5

IPDFA Using Value Contexts

Value Contexts: Key Ideas

Consider call chains σ_1 and σ_2 reaching S_p

- Data flow value invariant:

  If the data flow reaching S_p along σ_1 and σ_2 are identical, then

Value Contexts: Key Ideas

Consider call chains σ_1 and σ_2 reaching S_p

- Data flow value invariant:

  If the data flow reaching S_p along σ_1 and σ_2 are identical, then

  - the data flow values reaching E_p for the two contexts will also be identical

Value Contexts: Key Ideas

Consider call chains σ_1 and σ_2 reaching S_p

- Data flow value invariant:

  If the data flow reaching S_p along σ_1 and σ_2 are identical, then

  - the data flow values reaching E_p for the two contexts will also be identical

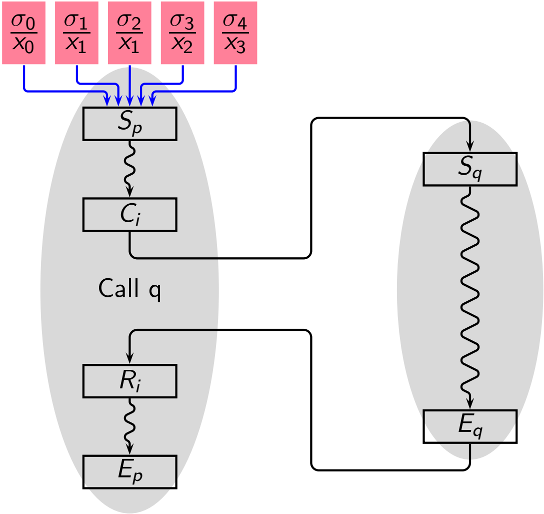
- We can reduce the amount of effort by using

  - Data flow values at S_p as value contexts
  - Maintaining distinct data flow values in p for each value context

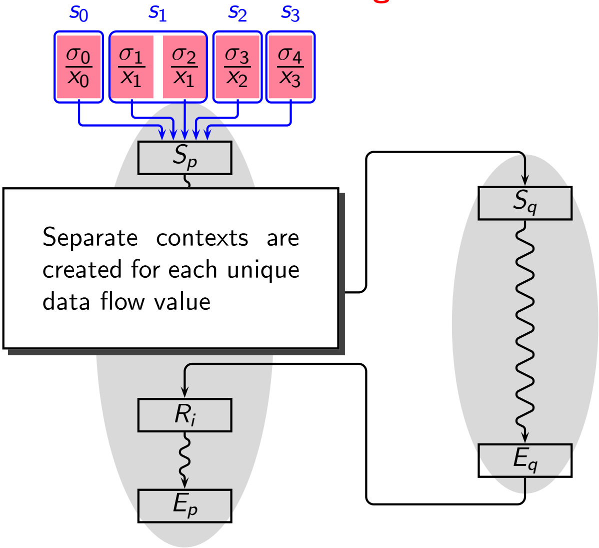
Interprocedural Data Flow Analysis Using Value Contexts

- A value context is defined by a particular input data flow value reaching a procedure
- It is used to enumerate the summary flow functions in terms of (input ↦ output) pairs
- In order to compute these pairs, data flow analysis within a procedure is performed separately for each context (i.e. input data flow value)
- When a new call to a procedure is encounterd, the pairs are consulted do decide if the procedure needs to be analysed again
    - If it was already analysed once for the input value, output can be directly processed
    - Otherwise, a new context is created and the procedure is analysed for this new context

Understanding Value Contexts

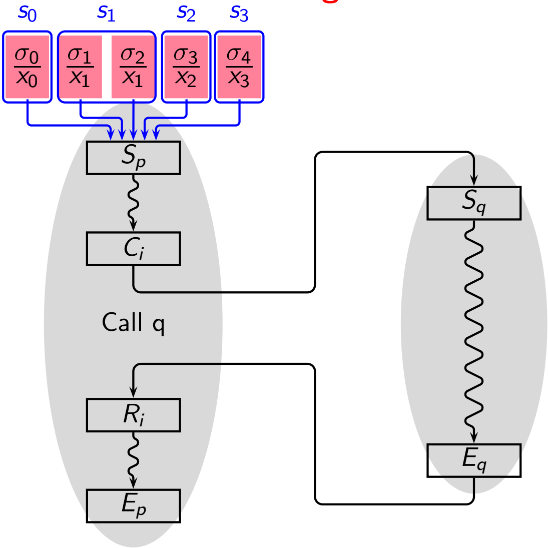


Understanding Value Contexts

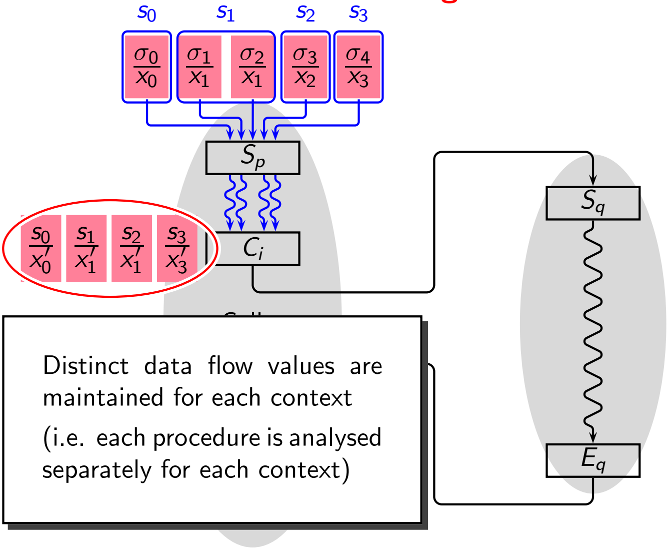


Separate contexts are created for each unique data flow value

Understanding Value Contexts



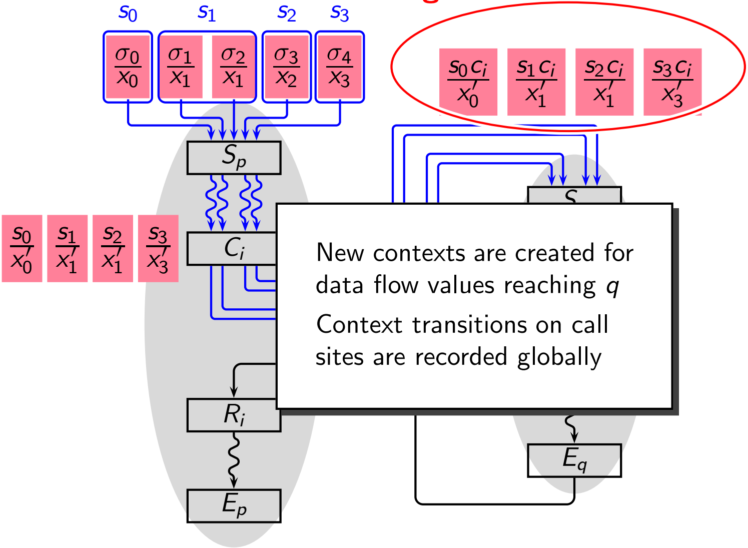
Understanding Value Contexts



Distinct data flow values are maintained for each context

(i.e. each procedure is analysed separately for each context)

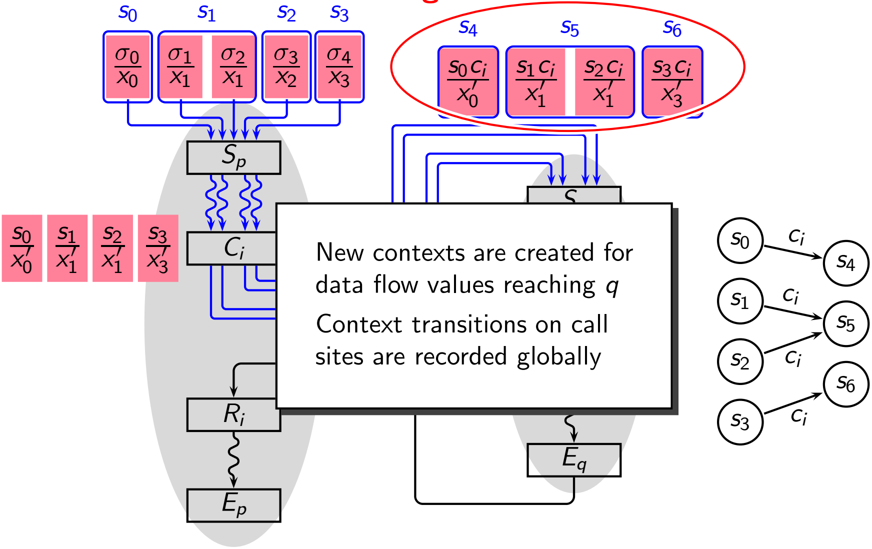
Understanding Value Contexts



New contexts are created for data flow values reaching q

Context transitions on call sites are recorded globally

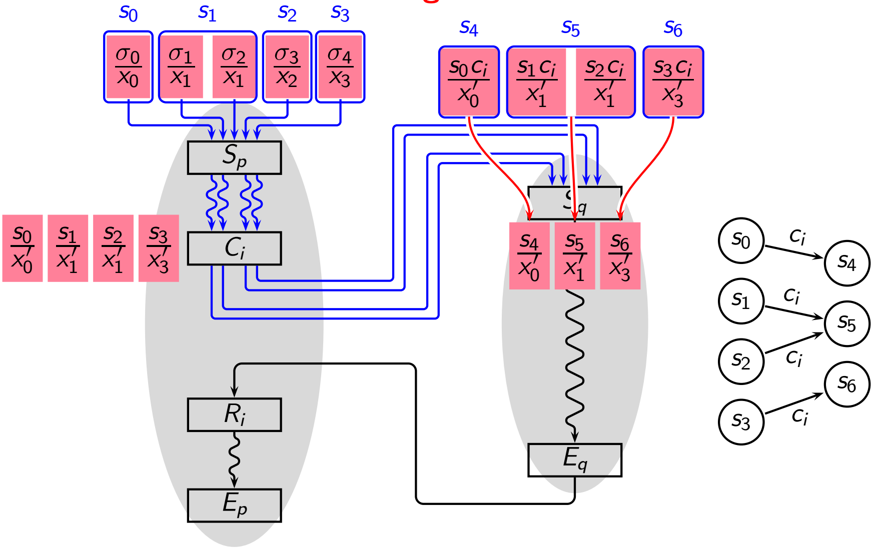
Understanding Value Contexts



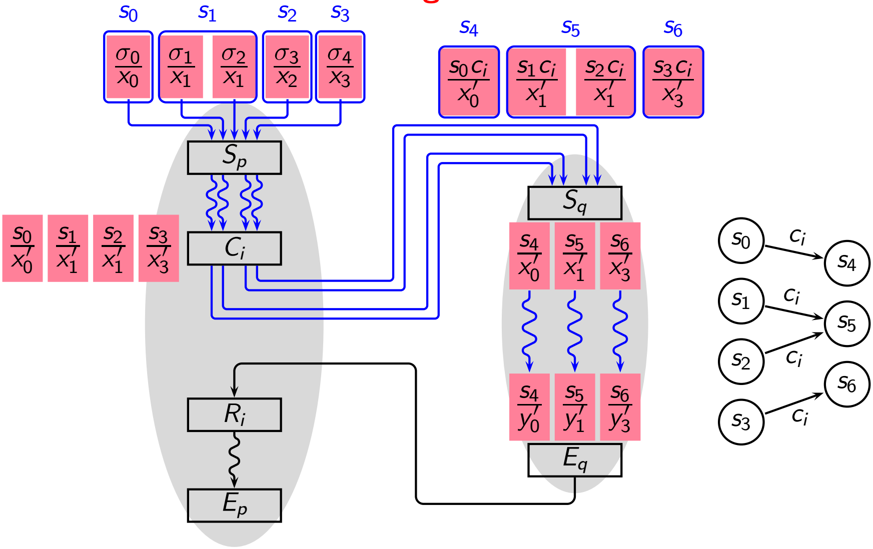
New contexts are created for data flow values reaching q

Context transitions on call sites are recorded globally

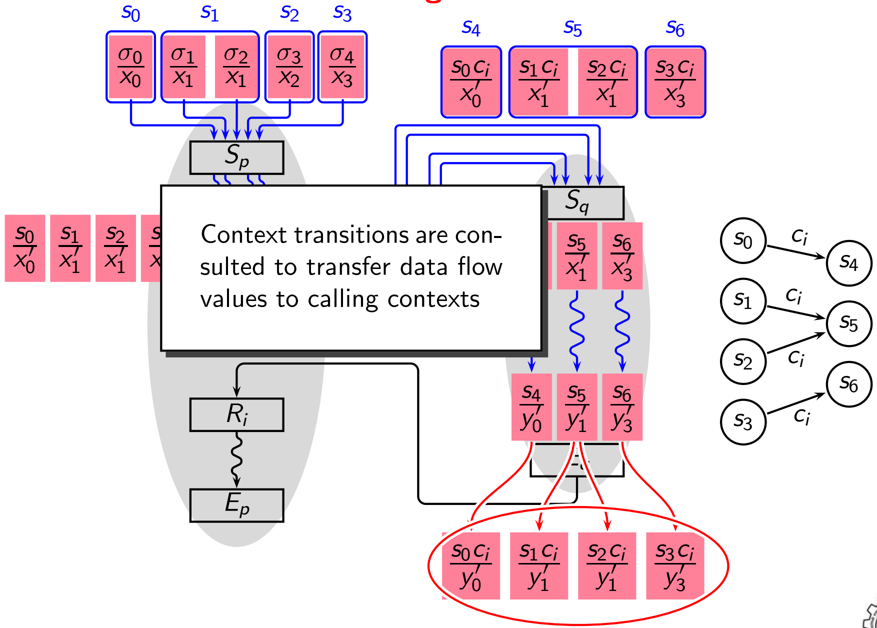
Understanding Value Contexts



Understanding Value Contexts

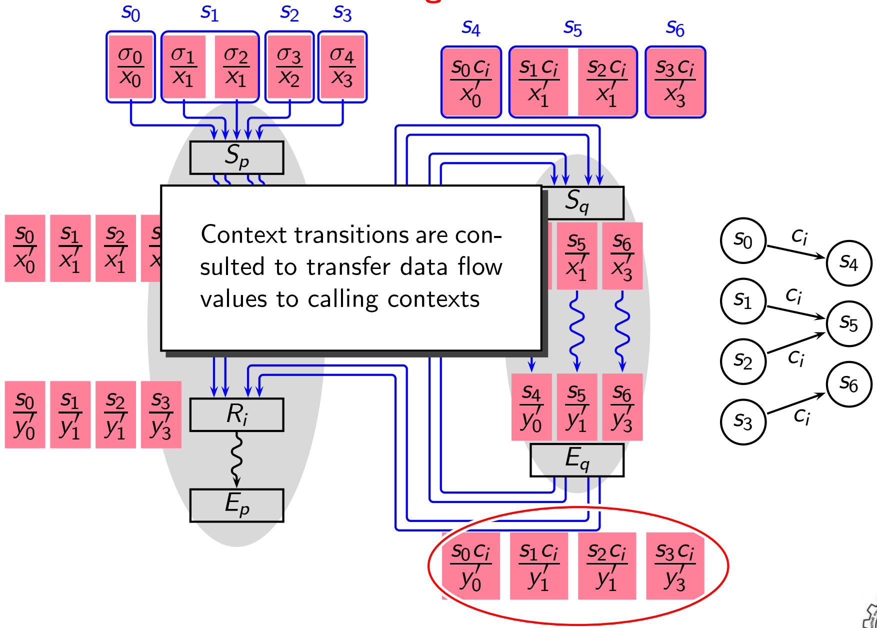


Understanding Value Contexts



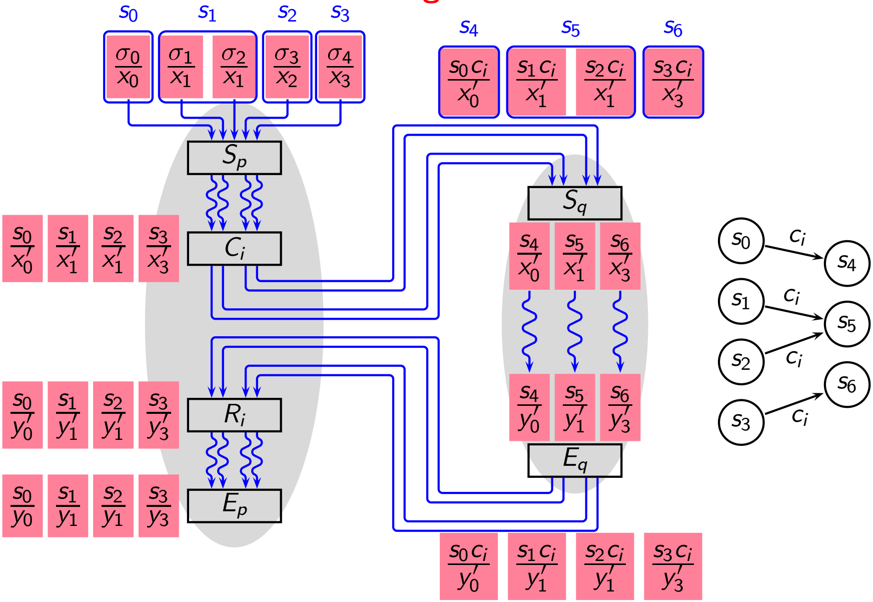
Context transitions are consulted to transfer data flow values to calling contexts

Understanding Value Contexts

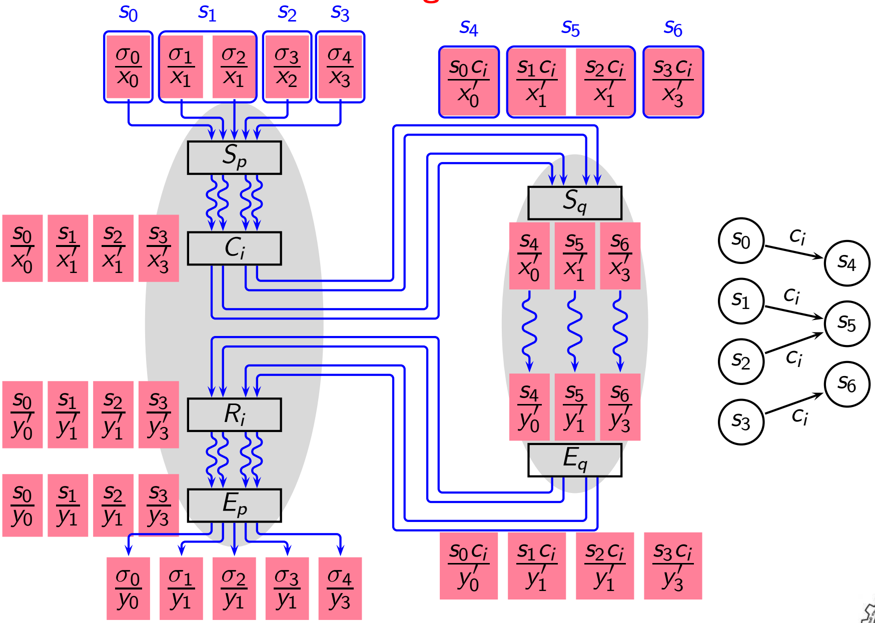


Context transitions are consulted to transfer data flow values to calling contexts

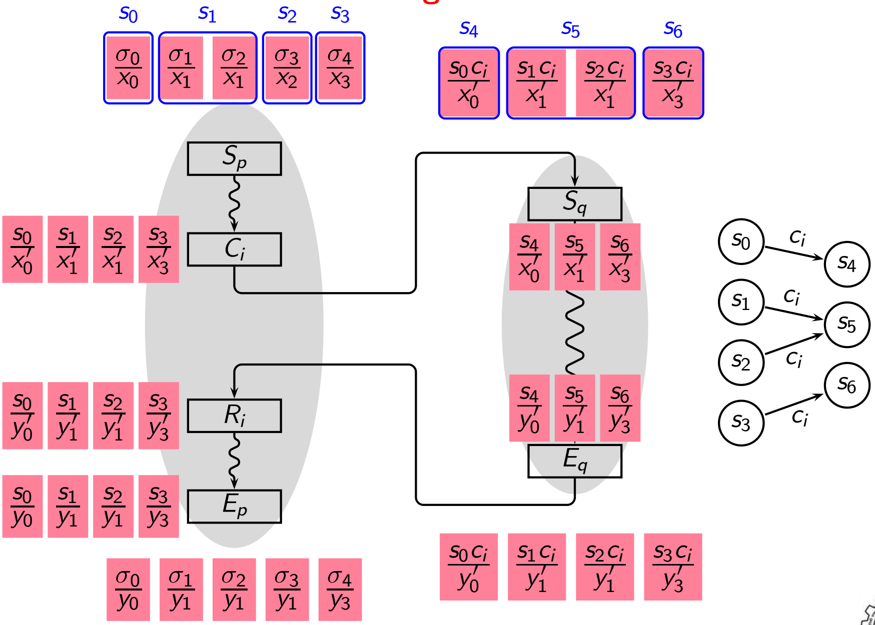
Understanding Value Contexts



Understanding Value Contexts



Understanding Value Contexts



Defining Value Contexts

- The set of value contexts is $VC = Procs \times L$

  A value context $X = \langle proc, entryValue \rangle \in VC$
  where $proc \in Procs$ and $entryValue \in L$

Defining Value Contexts

- The set of value contexts is $VC = Procs \times L$

  A value context $X = \langle proc, entryValue \rangle \in VC$
  where $proc \in Procs$ and $entryValue \in L$

- Supporting functions (CS is the set of call sites)
  - $exitValue : VC \mapsto L$
  - $transitions : (VC \times CS) \mapsto VC$

Defining Value Contexts

- The set of value contexts is $VC = Procs \times L$

  A value context $X = \langle proc, entryValue \rangle \in VC$
  where $proc \in Procs$ and $entryValue \in L$

- Supporting functions (CS is the set of call sites)
  - $exitValue : VC \mapsto L$ eg. $exitValue(X) = v$
  - $transitions : (VC \times CS) \mapsto VC$ eg. $X \xrightarrow{C_i} Y$

Interprocedural Data Flow Analysis Using Value Contexts

- The method works with a collection of control flow graphs

  No need of supergraph

  - No need to distinguish between C_i and R_i
  - No need of call $(C_i \rightarrow S_p)$ and return $(E_p \rightarrow E_i)$ edges
- Maintain a work list WL of entries $\langle context, node \rangle$

  (in reverse post order of nodes within a procedure for forward flows)
- Notation:

$\langle p,v \rangle$	Context for procedure p with data flow value v
$X \mid m$	Work list entry for context X for node m
$X.v$	Data flow value in context X is v
$Out_m[X]$	Data flow value of context X in Out_m
$X \overset{C_i}{\rightarrow} Y$	Transition from context X to context Y at call site C_i

Interprocedural Data Flow Analysis Using Value Contexts: An Overview

- Select $X|n$ from WL. Compute In_n.

Interprocedural Data Flow Analysis Using Value Contexts: An Overview

- Select $X|n$ from WL. Compute In_n.
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
  - If n is some other node

Interprocedural Data Flow Analysis Using Value Contexts: An Overview

- Select $X|n$ from WL. Compute In_n.
  - If $n = C_i$ calling procedure p
    Propagate In_n to appropriate value context of the callee procedure p
  - If $n = E_p$
  - If n is some other node

Interprocedural Data Flow Analysis Using Value Contexts: An Overview

- Select $X|n$ from WL. Compute In_n.
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
    Propagate In_n to appropriate value contexts of the callers of p
  - If n is some other node

Interprocedural Data Flow Analysis Using Value Contexts: An Overview

- Select $X|n$ from WL. Compute In_n.
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
  - If n is some other node
    Compute Out_n

Interprocedural Data Flow Analysis Using Value Contexts: An Overview

- Select $X|n$ from WL. Compute In_n.
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
  - If n is some other node

Update WL

Interprocedural Data Flow Analysis Using Value Contexts: An Overview

- Select $X|n$ from WL. Compute In_n.
  - If $n = C_i$ calling procedure p
    Propagate In_n to appropriate value context of the callee procedure p
  - If $n = E_p$
    Propagate In_n to appropriate value contexts of the callers of p
  - If n is some other node
    Compute Out_n

  Update WL

Interprocedural Data Flow Analysis Using Value Contexts: An Overview

- Select $X|n$ from WL. Compute In_n.
    - If $n = C_i$ calling procedure p
      Propagate In_n to appropriate value context of the callee procedure p
    - If $n = E_p$
      Propagate In_n to appropriate value contexts of the callers of p
    - If n is some other node
      Compute Out_n

  Update WL
- Repeat until WL is empty

Interprocedural Data Flow Analysis Using Value Contexts (2)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

Interprocedural Data Flow Analysis Using Value Contexts (2)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = C_i$ calling procedure p

Interprocedural Data Flow Analysis Using Value Contexts (2)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = C_i$ calling procedure p
    - If some context $\langle p,v \rangle$ exists (say Y) /* p is the callee */
    - If it does not exist

Interprocedural Data Flow Analysis Using Value Contexts (2)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = C_i$ calling procedure p
  - If some context $\langle p,v\rangle$ exists (say Y) /* p is the callee */
    - record the transition $X \overset{C_i}{\rightarrow} Y$
    - $Out_{C_i}[X] = Out_{C_i}[X] \sqcap exitValue(Y)$
    - if there is a change, add $X|m$, $\forall m \in succ(C_i)$ to WL
  - If it does not exist

Interprocedural Data Flow Analysis Using Value Contexts (2)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = C_i$ calling procedure p
  - If some context $\langle p, v \rangle$ exists (say Y) /* p is the callee */
  - If it does not exist
    - create a new context $Y = \langle p, v \rangle$ /* p is the callee */
    - initialize $exitValue(Y) = \top$
    - record the transition $X \overset{C_i}{\to} Y$
    - initialize $Out_m[Y] = \top$ for all nodes m of procedure p
    - add entries $Y|m$ for all nodes m of procedure p to WL

Interprocedural Data Flow Analysis Using Value Contexts (2)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = C_i$ calling procedure p
  - If some context $\langle p,v\rangle$ exists (say Y) /* p is the callee */
    - record the transition $X \overset{C_i}{\rightarrow} Y$
    - $Out_{C_i}[X] = Out_{C_i}[X] \sqcap exitValue(Y)$
    - if there is a change, add $X|m$, $\forall m \in succ(C_i)$ to WL
  - If it does not exist
    - create a new context $Y = \langle p,v\rangle$ /* p is the callee */
    - initialize $exitValue(Y) = \top$
    - record the transition $X \overset{C_i}{\rightarrow} Y$
    - initialize $Out_m[Y] = \top$ for all nodes m of procedure p
    - add entries $Y|m$ for all nodes m of procedure p to WL

Interprocedural Data Flow Analysis Using Value Contexts (3)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

Interprocedural Data Flow Analysis Using Value Contexts (3)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = E_p$

- For all other nodes

Interprocedural Data Flow Analysis Using Value Contexts (3)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = E_p$
  - Set $exitValue(X) = v$ /* E_p is an empty block */
- For all other nodes

Interprocedural Data Flow Analysis Using Value Contexts (3)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = E_p$
  - Set $exitValue(X) = v$ /* E_p is an empty block */
  - Find out all transitions $Z \xrightarrow{C_j} X$
    - Set $Out_{C_j}[Z] = Out_{C_j}[Z] \sqcap v$
    - If there is a change, add $Z|m$, $\forall m \in succ(C_j)$ to WL
- For all other nodes

Interprocedural Data Flow Analysis Using Value Contexts (3)

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = E_p$

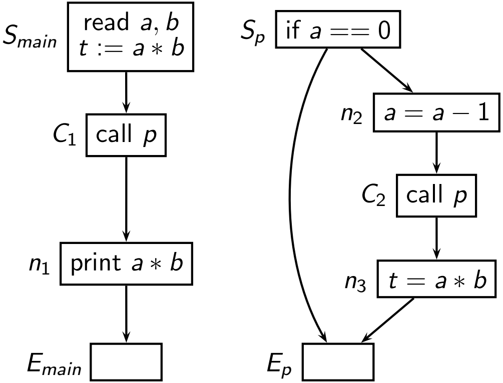
- For all other nodes
    - Set $Out_n[X] = f_n(v)$

Interprocedural Data Flow Analysis Using Value Contexts (3)

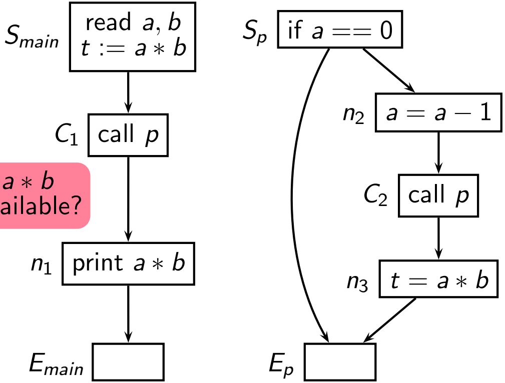
Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = E_p$
  - Set $exitValue(X) = v$ /* E_p is an empty block */
  - Find out all transitions $Z \overset{C_j}{\rightarrow} X$
    - Set $Out_{C_j}[Z] = Out_{C_j}[Z] \sqcap v$
    - If there is a change, add $Z|m$, $\forall m \in succ(C_j)$ to WL
- For all other nodes
  - Set $Out_n[X] = f_n(v)$
  - If there is a change, add $X|m$, $\forall m \in succ(n)$ to WL

Available Expressions Analysis Using Value Contexts

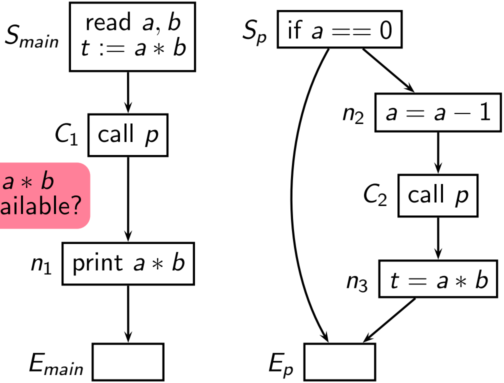


Available Expressions Analysis Using Value Contexts



Is $a * b$ available?

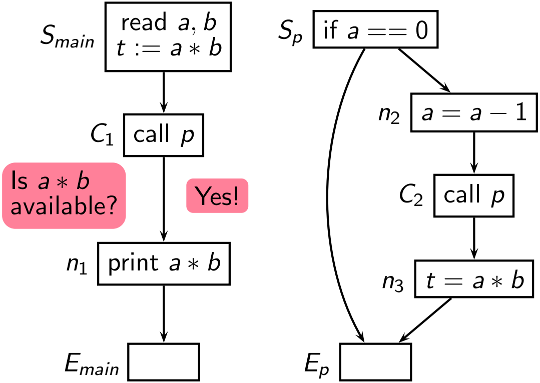
Available Expressions Analysis Using Value Contexts



Is $a * b$ available?

```
int a, b, t;
void p()
{  if (a == 0)
   {  a = a-1;
      p();
      t = a*b;
   }
}
```

Available Expressions Analysis Using Value Contexts



S_{main}: read a, b; $t := a * b$

C_1: call p

n_1: print $a * b$

E_{main}

S_p: if $a == 0$

n_2: $a = a - 1$

C_2: call p

n_3: $t = a * b$

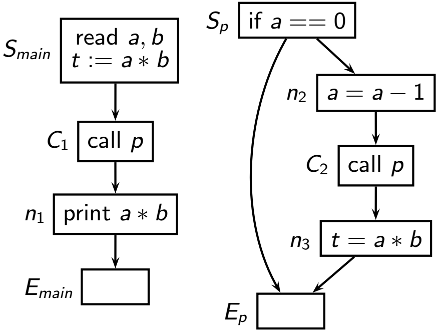
E_p

Is $a * b$ available?

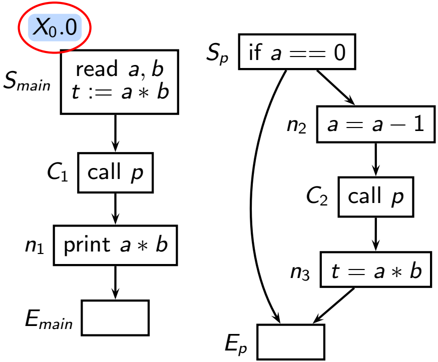
Yes!

```
int a, b, t;
void p()
{  if (a == 0)
   {  a = a-1;
      p();
      t = a*b;
   }
}
```

Available Expressions Analysis Using Value Contexts



Available Expressions Analysis Using Value Contexts

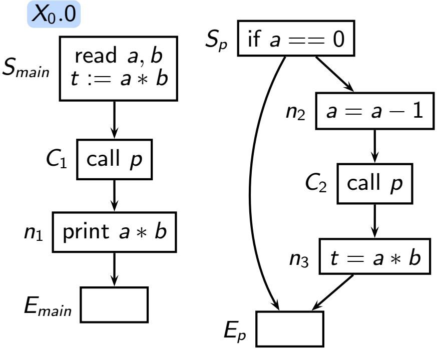


Create a new context X_0 with BI which is 0 for available expressions analysis

Available Expressions Analysis Using Value Contexts

$WL = [X_0|S_m, X_0|C_1, X_0|n_1, X_0|E_m]$

Context	*exitValue*
$X_0 = \langle \text{main}, 0 \rangle$	1



Create a new context X_0 with *BI* which is 0 for available expressions analysis

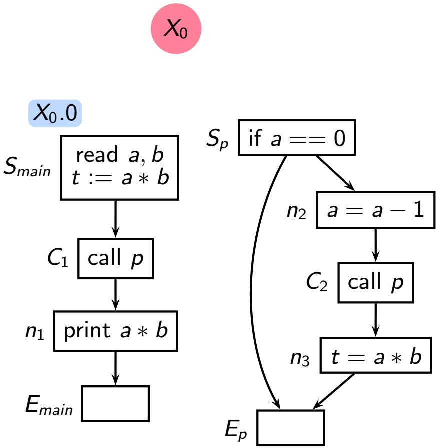
Initialize $exitValue(X_0)$ to $\top = 1$

Initialize the work list with all nodes in procedure main for X_0

Initialize $Out_n[X_0]$ for all n in main to $\top$

Available Expressions Analysis Using Value Contexts

$WL = [X_0|S_m, X_0|C_1, X_0|n_1, X_0|E_m]$

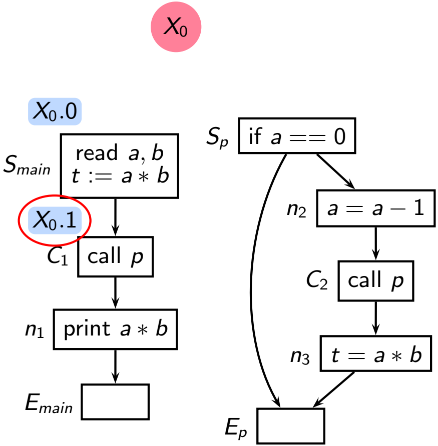


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1

Compute the data flow values for S_m for context X_0

Available Expressions Analysis Using Value Contexts

$WL = [X_0|C_1, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle \text{main}, 0 \rangle$	1

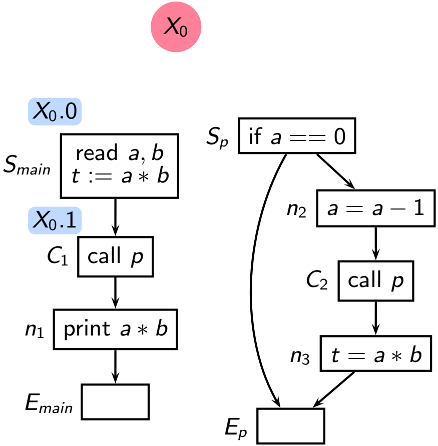
Compute the data flow values for S_m for context X_0

It does not change

Available Expressions Analysis Using Value Contexts

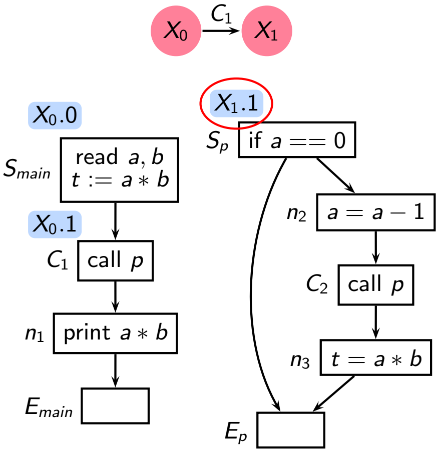
$WL = [X_0|C_1, X_0|n_1, X_0|E_m]$

Context	*exitValue*
$X_0 = \langle \text{main}, 0\rangle$	1



Available Expressions Analysis Using Value Contexts

$WL = [X_1|S_p, X_1|n_2, X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

Create a new context X_1 with entry value 1

Record the transition to X_1

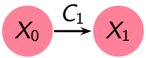
Initialize $exitValue(X_1)$ to $\top = 1$

Add all nodes of procedure p to the work list for X_1

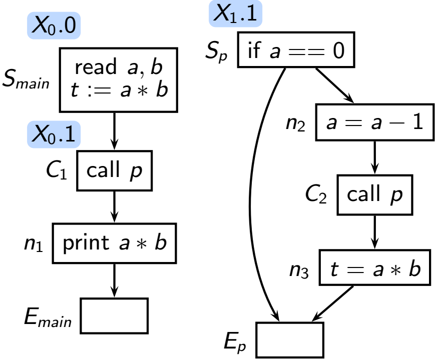
Initialize $Out_n[X_1]$ for all n in p to $\top$

Available Expressions Analysis Using Value Contexts

$WL = [X_1|S_p, X_1|n_2, X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$

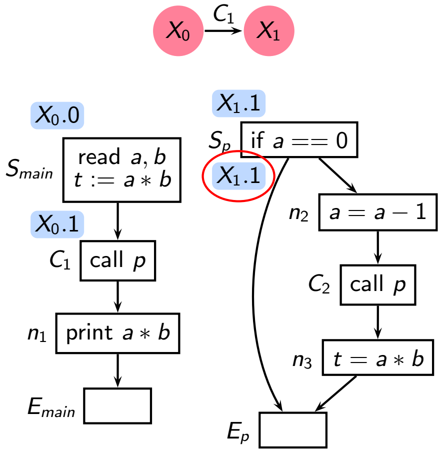


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1



Available Expressions Analysis Using Value Contexts

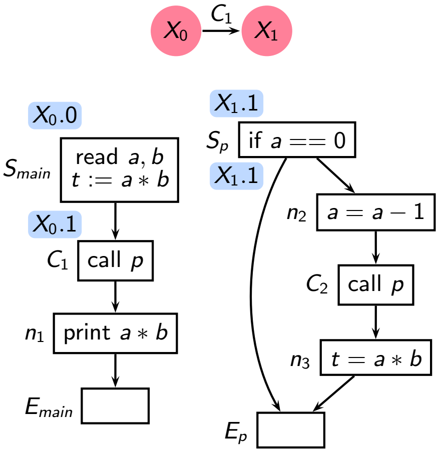
$WL = [X_1|n_2, X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

Available Expressions Analysis Using Value Contexts

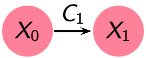
$WL = [X_1|n_2, X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



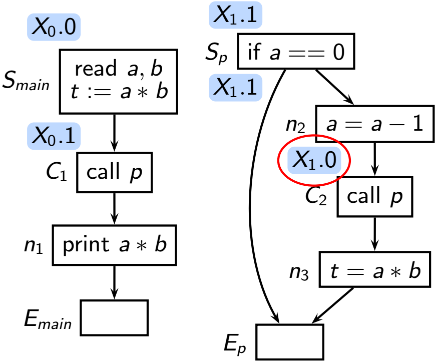
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

Available Expressions Analysis Using Value Contexts

$WL = [X_1|n_2, X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$

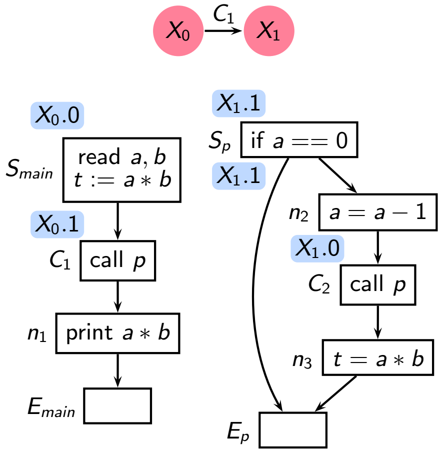


Context	*exitValue*
$X_0 = \langle \text{main,0} \rangle$	1
$X_1 = \langle \text{p,1} \rangle$	1



Available Expressions Analysis Using Value Contexts

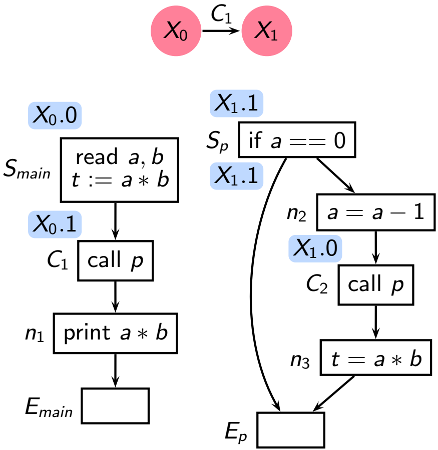
$WL = [X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

Available Expressions Analysis Using Value Contexts

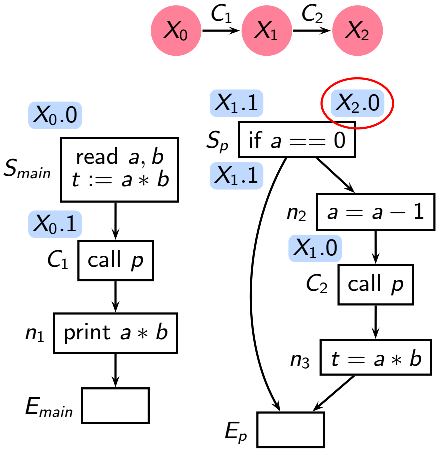
$WL = [X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

Available Expressions Analysis Using Value Contexts

$WL = [X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

Since there is no context for p with value 0, create context X_2

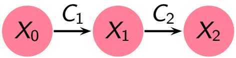
Record the transition to X_2

Initialize $exitValue(X_2)$ to $\top = 1$

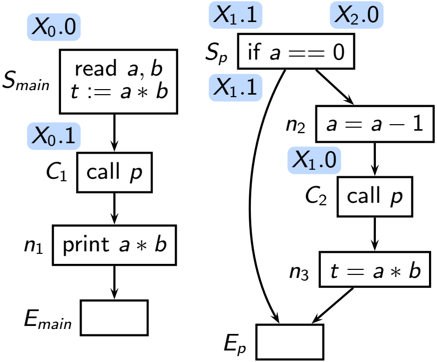
Add all nodes of procedure p to the work list for X_2 Initialize $Out_n[X_2]$ for all n in p to $\top$

Available Expressions Analysis Using Value Contexts

$WL = [X_2|S_p, X_2|n_2, X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$

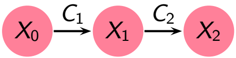


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

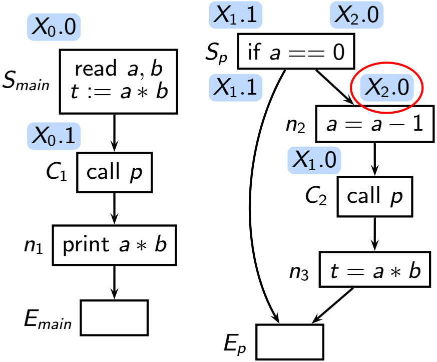


Available Expressions Analysis Using Value Contexts

$$WL = [X_2|S_p, X_2|n_2, X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$$

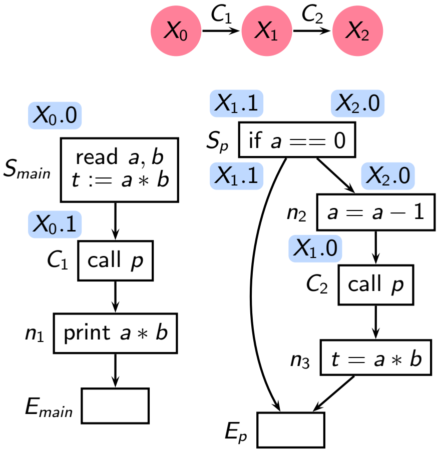


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1



Available Expressions Analysis Using Value Contexts

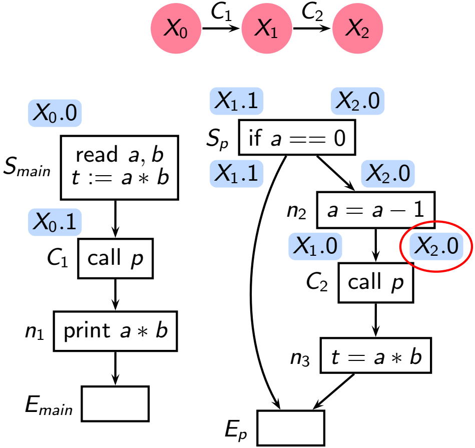
$$WL = [X_2|n_2, X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

Available Expressions Analysis Using Value Contexts

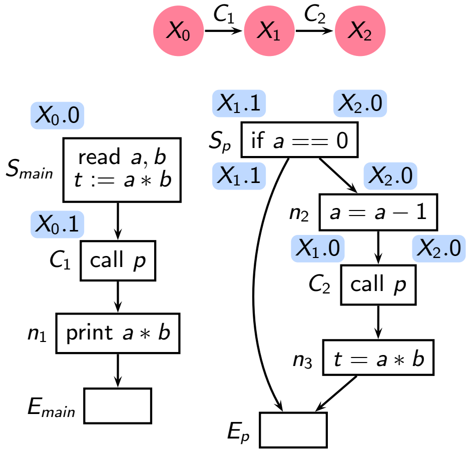
$WL = [X_2|n_2, X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

Available Expressions Analysis Using Value Contexts

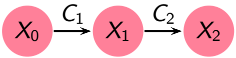
$WL = [X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



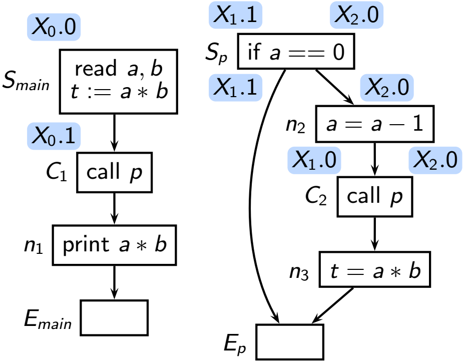
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

Available Expressions Analysis Using Value Contexts

$WL = [X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



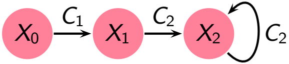
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1



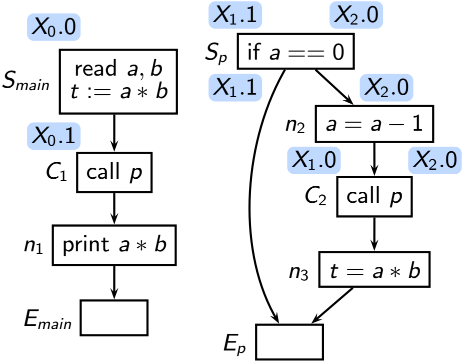
p has context X_2 with value 0 so no need to create a new context

Available Expressions Analysis Using Value Contexts

$WL = [X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

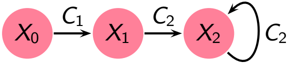


p has context X_2 with value 0 so no need to create a new context

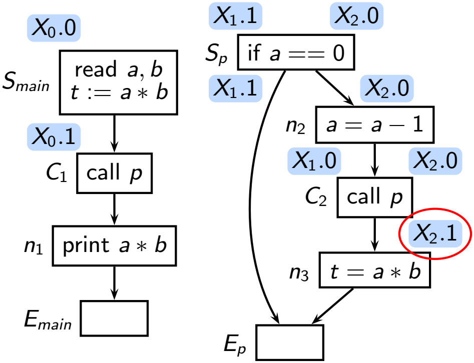
Record the transition from context X_2 to itself

Available Expressions Analysis Using Value Contexts

$$WL = [X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1



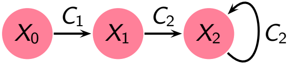
p has context X_2 with value 0 so no need to create a new context

Record the transition from context X_2 to itself

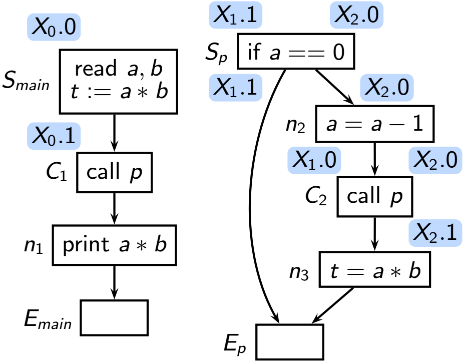
Use the *exitValue*(X_2) to compute $Out_{C_2}[X_2]$

Available Expressions Analysis Using Value Contexts

$WL = [X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$

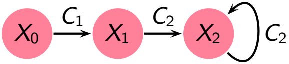


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

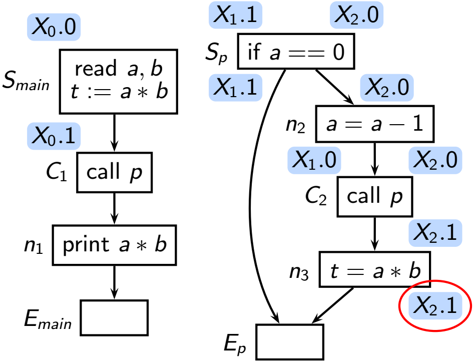


Available Expressions Analysis Using Value Contexts

$WL = [X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$

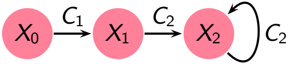


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

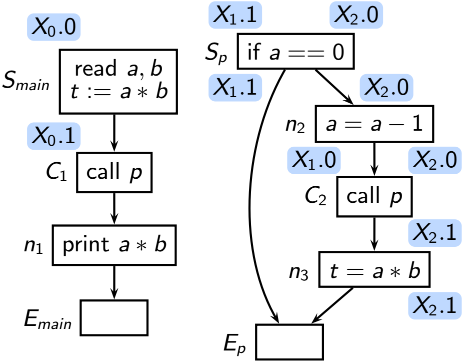


Available Expressions Analysis Using Value Contexts

$WL = [X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$

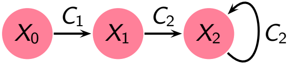


Context	*exitValue*
$X_0 = \langle \text{main},0 \rangle$	1
$X_1 = \langle \text{p},1 \rangle$	1
$X_2 = \langle \text{p},0 \rangle$	1

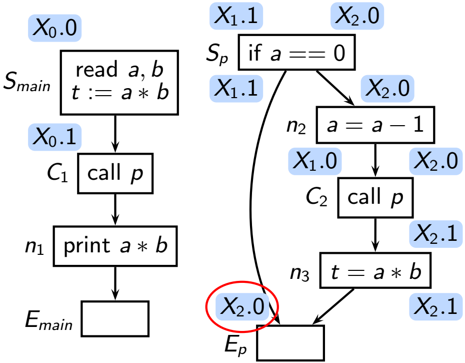


Available Expressions Analysis Using Value Contexts

$WL = [X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



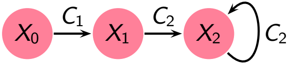
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1



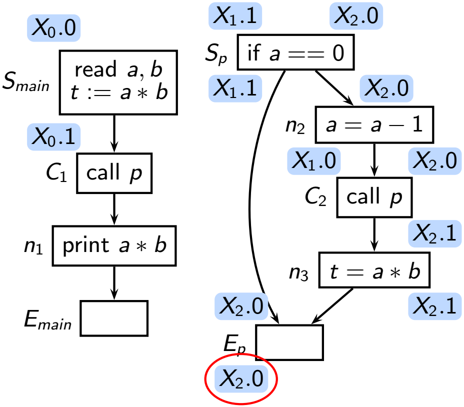
At E_p the values from S_p and n_3 are merged for context X_2

Available Expressions Analysis Using Value Contexts

$WL = [X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

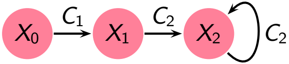


At E_p the values from S_p and n_3 are merged for context X_2

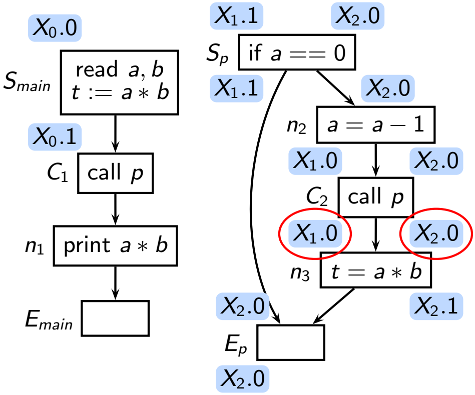
$exitValue(X_2)$ is set to 0

Available Expressions Analysis Using Value Contexts

$WL = [X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle \text{main},0 \rangle$	1
$X_1 = \langle \text{p},1 \rangle$	1
$X_2 = \langle \text{p},0 \rangle$	0



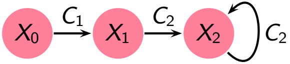
At E_p the values from S_p and n_3 are merged for context X_2

exitValue(X_2) is set to 0

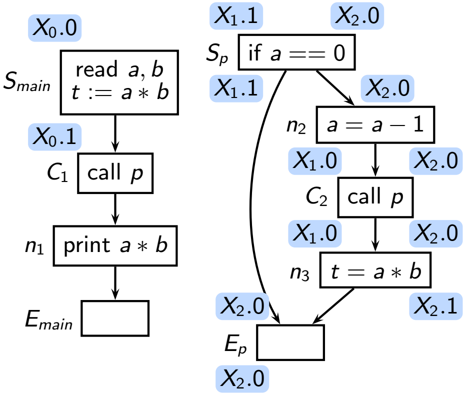
Since X_2 has transitions $X_1 \xrightarrow{C_2} X_2$ and $X_2 \xrightarrow{C_2} X_2$, $Out_{C_2}[X_1]$ and $Out_{C_2}[X_2]$ become 0

Available Expressions Analysis Using Value Contexts

$WL = [X_2|n_3, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



At E_p the values from S_p and n_3 are merged for context X_2

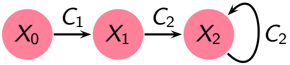
$exitValue(X_2)$ is set to 0

Since X_2 has transitions $X_1 \xrightarrow{C_2} X_2$ and $X_2 \xrightarrow{C_2} X_2$, $Out_{C_2}[X_1]$ and $Out_{C_2}[X_2]$ become 0

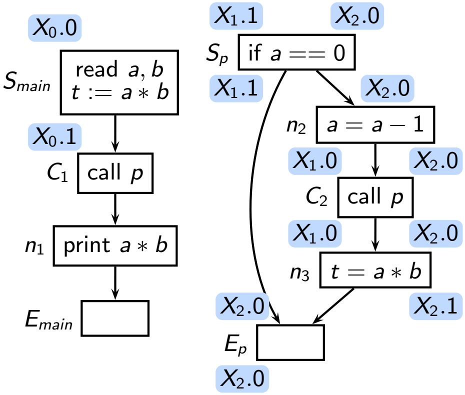
Since $Out_{C_2}[X_2]$ changes, $X_2|n_3$ is added to the work list

Available Expressions Analysis Using Value Contexts

$WL = [X_2|n_3, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



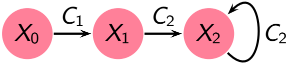
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



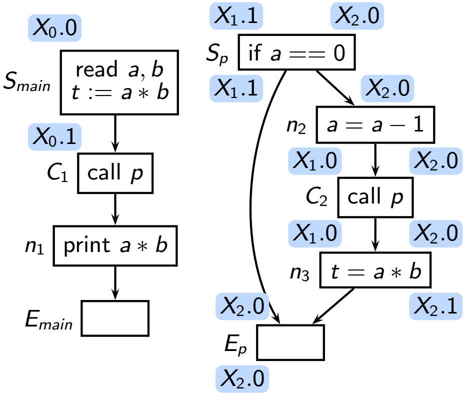
There is no change in $Out_{n_3}[X_2]$ (because it was initialized to $\top$)

Available Expressions Analysis Using Value Contexts

$WL = [X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



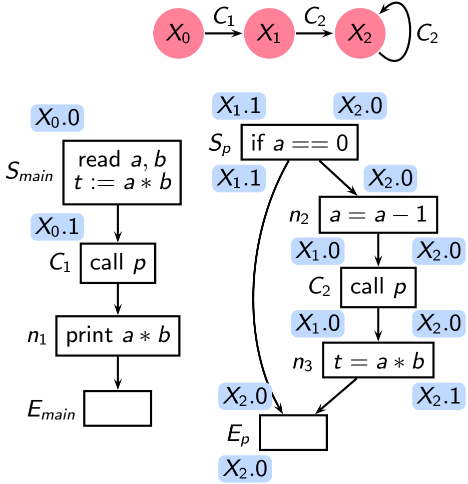
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



There is no change in $Out_{n_3}[X_2]$ (because it was initialized to $\top$)

Available Expressions Analysis Using Value Contexts

$WL = [X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$

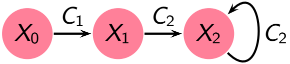


Context	*exitValue*
$X_0 = \langle \text{main},0\rangle$	1
$X_1 = \langle \text{p},1\rangle$	1
$X_2 = \langle \text{p},0\rangle$	0

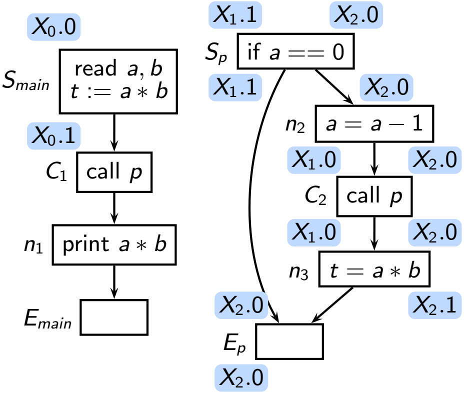
There is no change in $Out_{n_3}[X_1]$ either

Available Expressions Analysis Using Value Contexts

$WL = [X_1|E_p, X_0|n_1, X_0|E_m]$



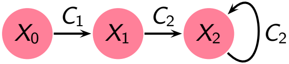
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



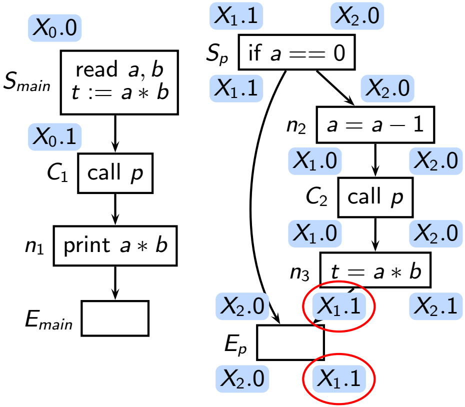
There is no change in $Out_{n_3}[X_1]$ either

Available Expressions Analysis Using Value Contexts

$WL = [X_1|E_p, X_0|n_1, X_0|E_m]$



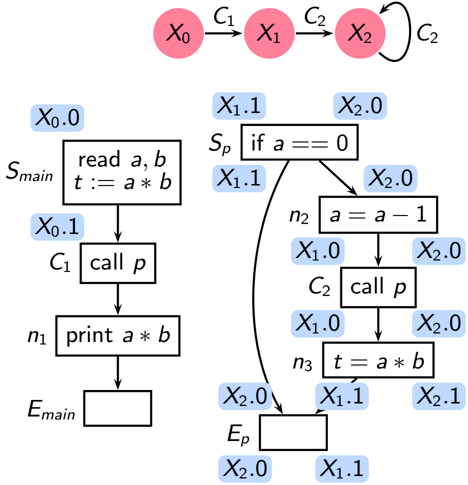
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



At E_p the values from S_p and n_3 are merged for context X_1

Available Expressions Analysis Using Value Contexts

$WL = [X_1|E_p, X_0|n_1, X_0|E_m]$



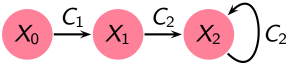
Context	*exitValue*
$X_0 = \langle \text{main},0\rangle$	1
$X_1 = \langle \text{p},1\rangle$	1
$X_2 = \langle \text{p},0\rangle$	0

At E_p the values from S_p and n_3 are merged for context X_1

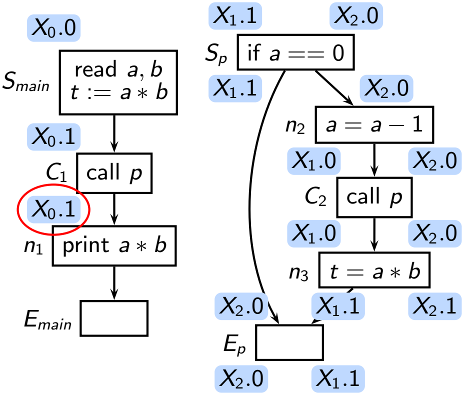
$exitValue(X_1)$ remains 1

Available Expressions Analysis Using Value Contexts

$WL = [X_1|E_p, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



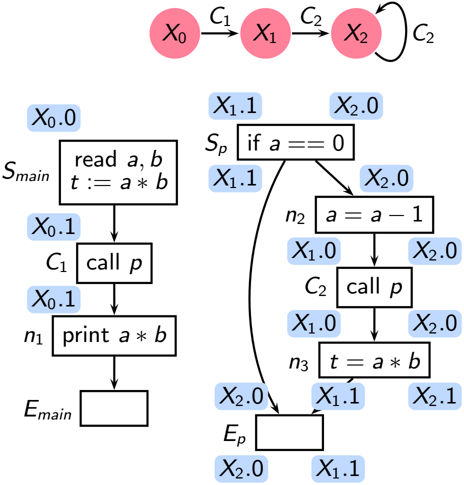
At E_p the values from S_p and n_3 are merged for context X_1

$exitValue(X_1)$ remains 1

Since X_1 has transition $X_0 \xrightarrow{C_1} X_1$, $Out_{C_1}[X_0]$ becomes 1

Available Expressions Analysis Using Value Contexts

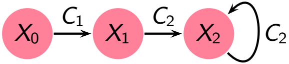
$WL = [X_0|n_1, X_0|E_m]$



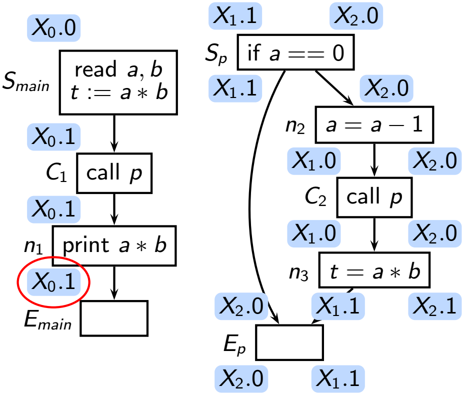
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

Available Expressions Analysis Using Value Contexts

$WL = [X_0|n_1, X_0|E_m]$

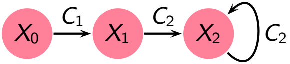


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

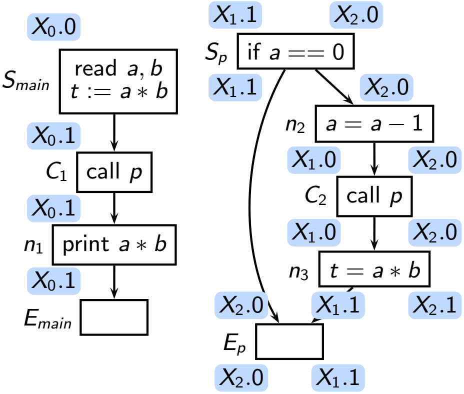


Available Expressions Analysis Using Value Contexts

$WL = [X_0 | E_m]$

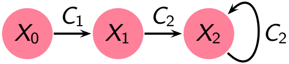


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

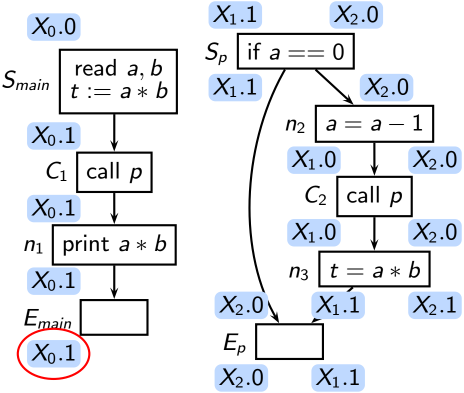


Available Expressions Analysis Using Value Contexts

$WL = [X_0 | E_m]$

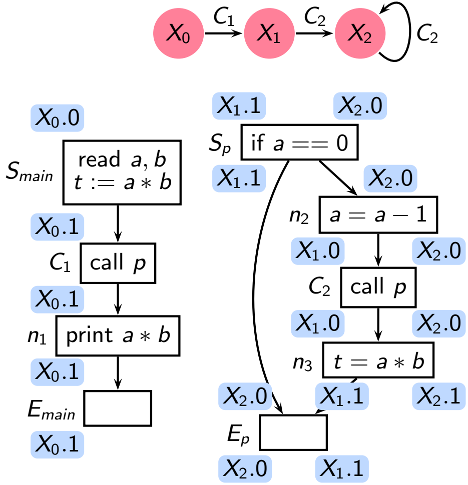


Context	*exitValue*
$X_0 = \langle \text{main}, 0 \rangle$	1
$X_1 = \langle \text{p}, 1 \rangle$	1
$X_2 = \langle \text{p}, 0 \rangle$	0



Available Expressions Analysis Using Value Contexts

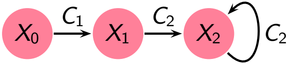
$WL = [X_0 | E_m]$



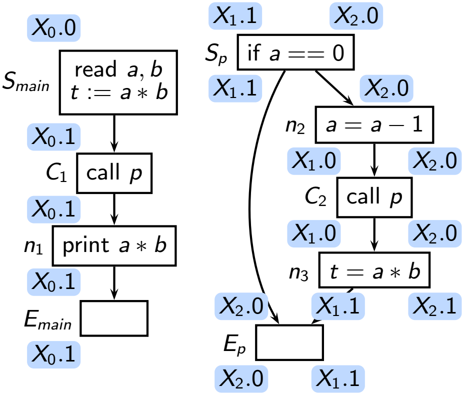
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

Available Expressions Analysis Using Value Contexts

$WL = [\ ]$



Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



Work list is empty and the analysis is over

A Trace of Value Context Based Analysis (1)

S. No.	Work List	Sel. node	Data flow value	New context	New trans.	exit value	Addition to the work list
1				$X_0 = \langle m, 0 \rangle$		$X_0.1$	$X_0 \| S_m, X_0 \| C_1, X_0 \| n_1, X_0 \| E_m$
2	$X_0 \| S_m, X_0 \| C_1, X_0 \| n_1, X_0 \| E_m$	S_m	$Out_{S_m}[X_0] = 1$				
3	$X_0 \| C_1, X_0 \| n_1, X_0 \| E_m$	C_1		$X_1 = \langle p, 1 \rangle$	$X_0 \xrightarrow{C_1} X_1$	$X_1.1$	$X_1 \| S_p, X_1 \| n_2, X_1 \| C_2, X_1 \| n_3, X_1 \| E_p$
4	$X_1 \| S_p, X_1 \| n_2, X_1 \| C_2, X_1 \| n_3, X_1 \| E_p, X_0 \| n_1, X_0 \| E_m$	S_p	$Out_{S_p}[X_1] = 1$				
5	$X_1 \| n_2, X_1 \| C_2, X_1 \| n_3, X_1 \| E_p, X_0 \| n_1, X_0 \| E_m$	n_2	$Out_{n_2}[X_1] = 0$				
6	$X_1 \| C_2, X_1 \| n_3, X_1 \| E_p, X_0 \| n_1, X_0 \| E_m$	C_2		$X_2 = \langle p, 0 \rangle$	$X_1 \xrightarrow{C_2} X_2$	$X_2.1$	$X_2 \| S_p, X_2 \| n_2, X_2 \| C_2, X_2 \| n_3, X_2 \| E_p$
7	$X_2 \| S_p, X_2 \| n_2, X_2 \| C_2, X_2 \| n_3, X_2 \| E_p, X_1 \| n_3, X_1 \| E_p, X_0 \| n_1, X_0 \| E_m$	S_p	$Out_{S_p}[X_2] = 0$				

A Trace of Value Context Based Analysis (2)

S. No.	Work List	Sel. node	Data flow value	New context	New trans.	exit value	Addition to the work list
8	$X_2\|n_2, X_2\|C_2, X_2\|n_3, X_2\|E_p, X_1\|n_3, X_1\|E_p, X_0\|n_1, X_0\|E_m$	n_2	$Out_{n_2}[X_2] = 0$				
9	$X_2\|C_2, X_2\|n_3, X_2\|E_p, X_1\|n_3, X_1\|E_p, X_0\|n_1, X_0\|E_m$	C_2	$Out_{C_2}[X_2] = 1$		$X_2 \xrightarrow{C_2} X_2$		
10	$X_2\|n_3, X_2\|E_p, X_1\|n_3, X_1\|E_p, X_0\|n_1, X_0\|E_m$	n_3	$Out_{n_3}[X_2] = 1$				
11	$X_2\|E_p, X_1\|n_3, X_1\|E_p, X_0\|n_1, X_0\|E_m$	E_p	$Out_{E_p}[X_2] = 0$ $Out_{C_2}[X_2] = 0$ $Out_{C_2}[X_1] = 0$			$X_2.0$	$X_2\|n_3$
12	$X_2\|n_3, X_1\|n_3, X_1\|E_p, X_0\|n_1, X_0\|E_m$	n_3	No change				
13	$X_1\|n_3, X_1\|E_p, X_0\|n_1, X_0\|E_m$	n_3	$Out_{n_3}[X_1] = 1$				
14	$X_1\|E_p, X_0\|n_1, X_0\|E_m$	E_p	$Out_{E_p}[X_1] = 1$ $Out_{C_1}[X_0] = 1$			$X_1.1$	
15	$X_0\|n_1, X_0\|E_m$	n_1	$Out_{n_1}[X_0] = 1$				
16	$X_0\|E_m$	E_m	$Out_{E_m}[X_0] = 1$				

Merging ExitValue with Previous Out Value at the Call Site

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = C_i$ calling procedure p
  - If some context $\langle p,v \rangle$ exists (say Y) /* p is the callee */
    - record the transition $X \overset{C_i}{\to} Y$
    - $Out_{C_i}[X] = Out_{C_i}[X] \sqcap exitValue(Y)$
    - if there is a change, add $X|m$, $\forall m \in succ(C_i)$ to WL

Merging ExitValue with Previous Out Value at the Call Site

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

- If $n = C_i$ calling procedure p
  - If some context $\langle p,v \rangle$ exists (say Y) /* p is the callee */
    - record the transition $X \overset{C_i}{\to} Y$
    - $Out_{C_i}[X] = \boxed{Out_{C_i}[X] \sqcap} exitValue(Y)$
    - if there is a change, add $X|m$, $\forall m \in succ(C_i)$ to WL

Merging ExitValue with Previous Out Value at the Call Site

Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n

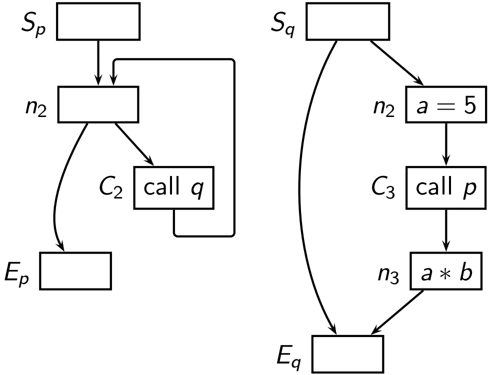
- If $n = C_i$ calling procedure p
    - If some context $\langle p,v \rangle$ exists (say Y) /* p is the callee */
        - record the transition $X \overset{C_i}{\rightarrow} Y$
        - $Out_{C_i}[X] = \boxed{Out_{C_i}[X] \ \sqcap} exitValue(Y)$
        - if there is a change, add $X|m$, $\forall m \in succ(C_i)$ to WL

Analogy:

- At the intraprocedural level, we merge the values at the entry of a loop to compute the glb across all iterations of the loop
- At the interprocedural level, we want to compute the glb across repeated calls at the same call site (perhaps in a loop)

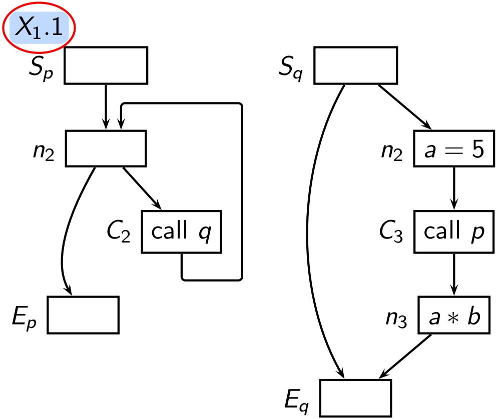
Partially Available Expressions Analysis Using Value Contexts

This example illustrates non-termination of analysis if the *exitValue* is not merged with the previous *Out* value



We assume that procedure main calls procedure p (and not q) and the expression $a * b$ is partially available on entry to p

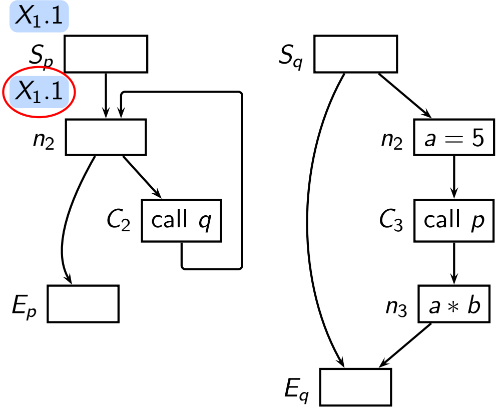
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1\rangle$	0

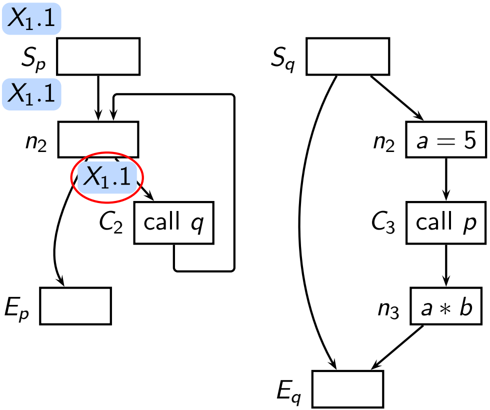
We create context X_1 for entry value 1 with *exitValue* as 0 ($\top$ for partially available expressions analysis)

Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0

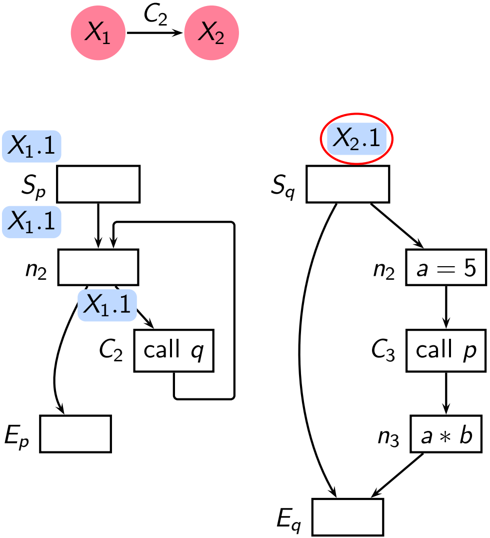
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p, 1 \rangle$	0

Value 1 reaches q and a new context must be created for it

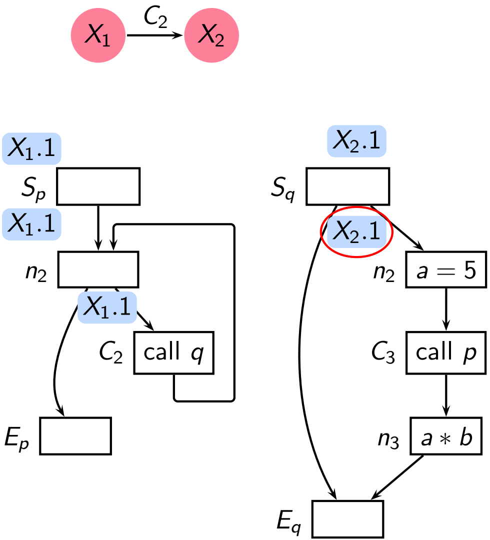
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0

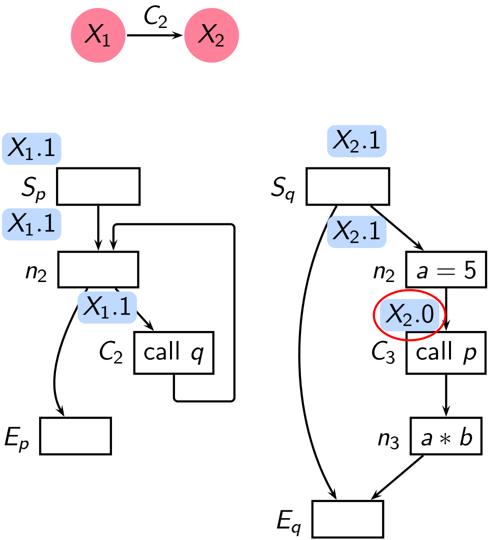
We create context X_2 for value 1 reaching q and record a transition from X_1 to X_2 on C_2

Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1\rangle$	0
$X_2 = \langle q,1\rangle$	0

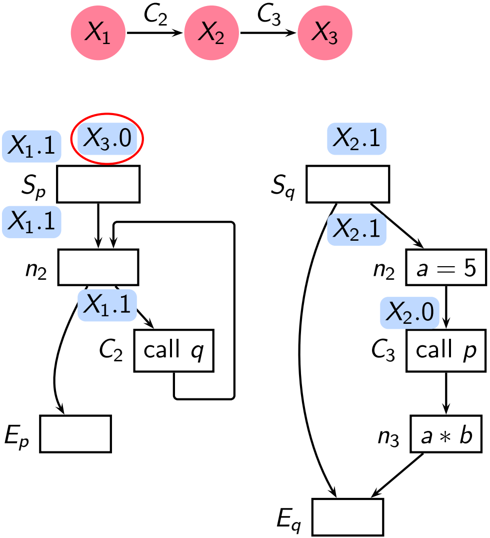
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0

The expression is killed in node n_2 and data flow value 0 reaches the call site C_3 that calls p

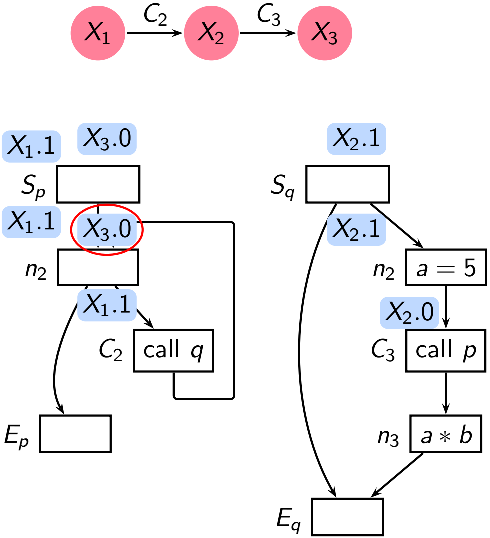
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0

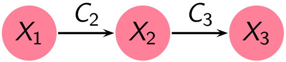
We create context X_3 for the new value (0) reaching p and record transition from X_2 to X_3 on C_3

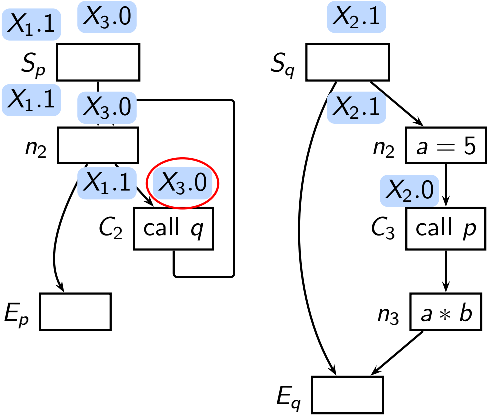
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p, 1\rangle$	0
$X_2 = \langle q, 1\rangle$	0
$X_3 = \langle p, 0\rangle$	0

Partially Available Expressions Analysis Using Value Contexts

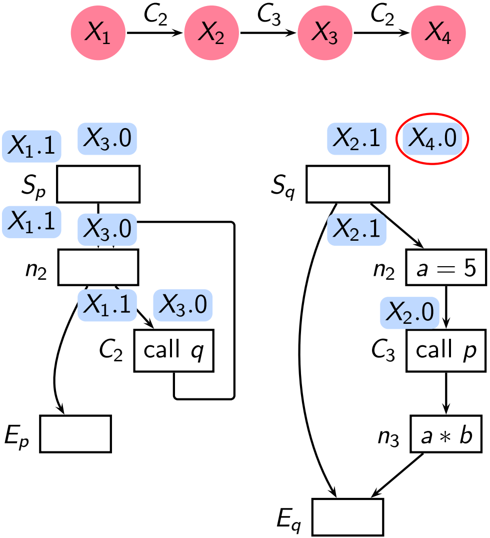




Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0

And now the value 0 reaches q at call site C_2

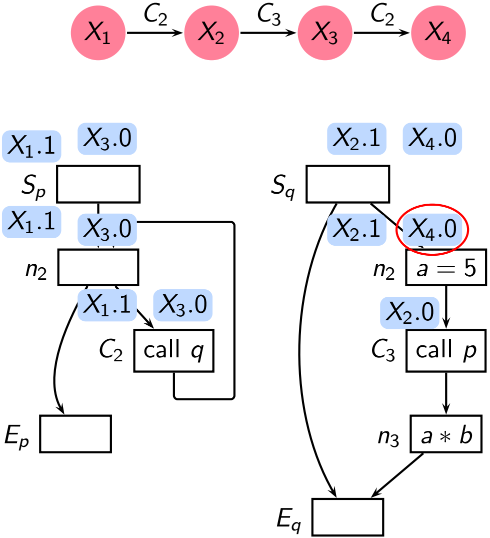
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	0

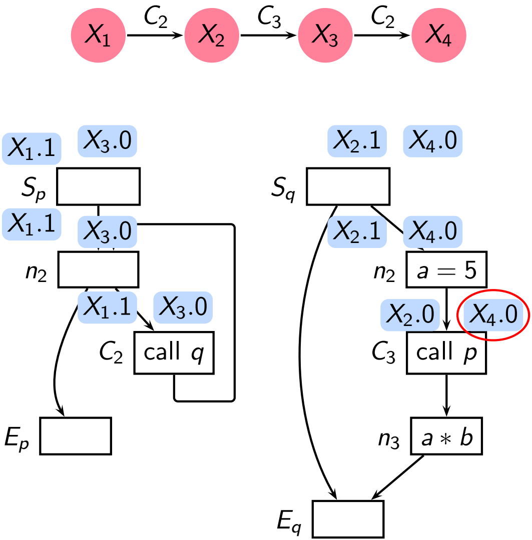
We create context X_4 for the new value (0) reaching p and record transition from X_3 to X_4 on C_2

Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	0

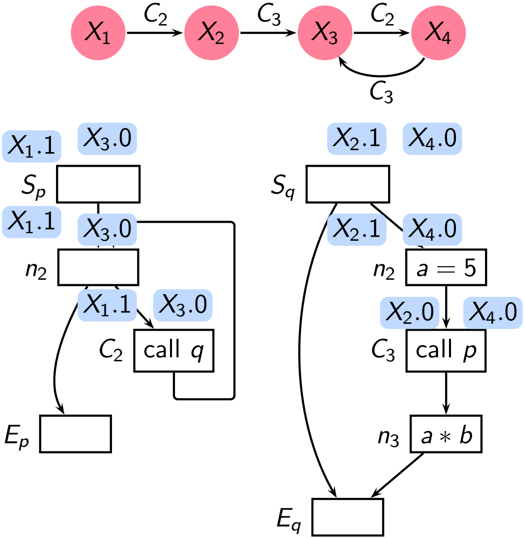
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	0

And now the value 0 reaches p at call site C_3

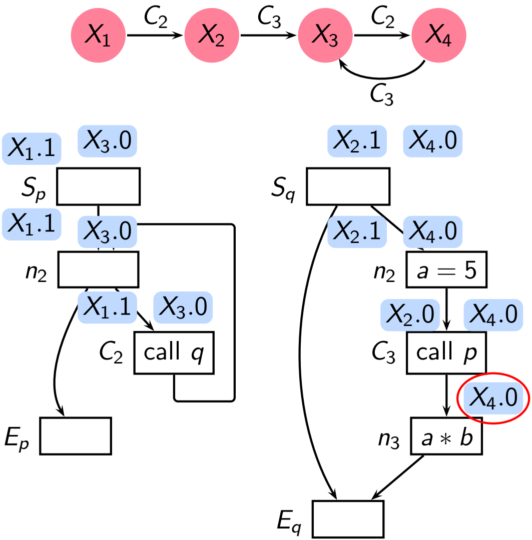
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p{,}1\rangle$	0
$X_2 = \langle q{,}1\rangle$	0
$X_3 = \langle p{,}0\rangle$	0
$X_4 = \langle q{,}0\rangle$	0

We already have context X_3 with entry value 0 for p so no need to analyse p again

Partially Available Expressions Analysis Using Value Contexts

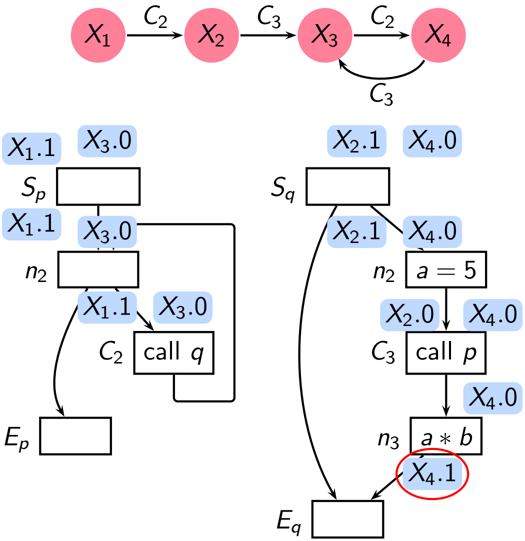


Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	0

We use the *exitValue* for X_3 to compute Out_{C_3} for the context X_4 (because of the transition $X_4 \xrightarrow{C_3} X_3$)

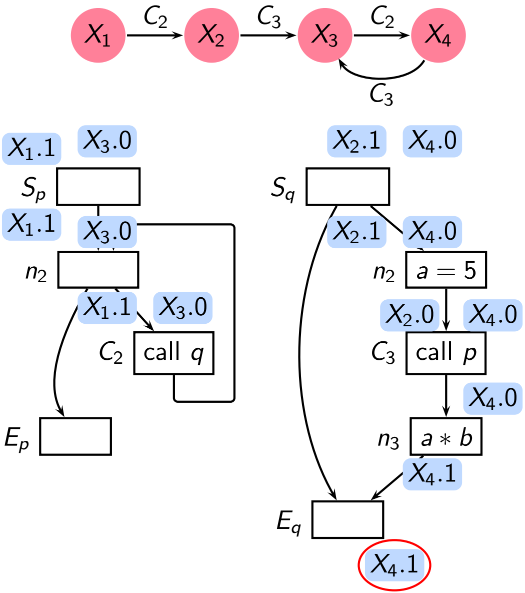
The analysis of p is not yet over for any context, and so we get the $\top$ value

Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	0

Partially Available Expressions Analysis Using Value Contexts

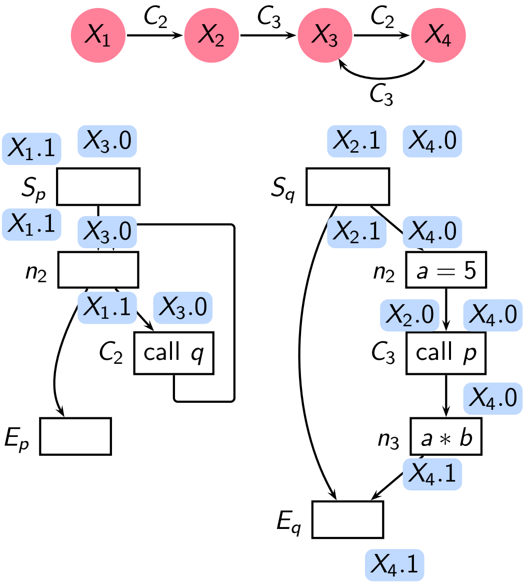


Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	0

The analysis of q for X_4 is now and the *exitValue* of X_4 becomes 1

This change in X_4 must be propagated to X_3 in the caller p (identified from the transition $X_3 \xrightarrow{C_2} X_4$)

Partially Available Expressions Analysis Using Value Contexts

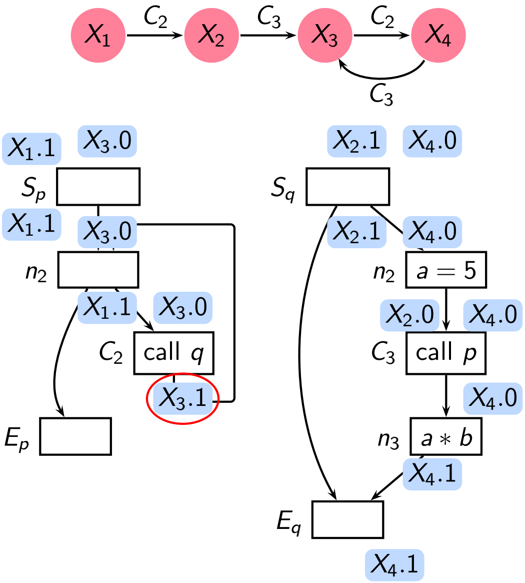


Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	1

The analysis of q for X_4 is now and the *exitValue* of X_4 becomes 1

This change in X_4 must be propagated to X_3 in the caller p (identified from the transition $X_3 \xrightarrow{C_2} X_4$)

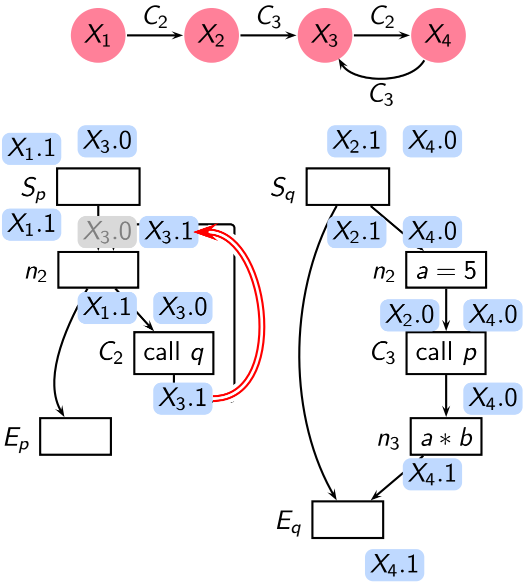
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p, 1\rangle$	0
$X_2 = \langle q, 1\rangle$	0
$X_3 = \langle p, 0\rangle$	0
$X_4 = \langle q, 0\rangle$	1

Out_{C_2} becomes 1 for X_3 which changes the value at In_{n_2} for X_3 to 1

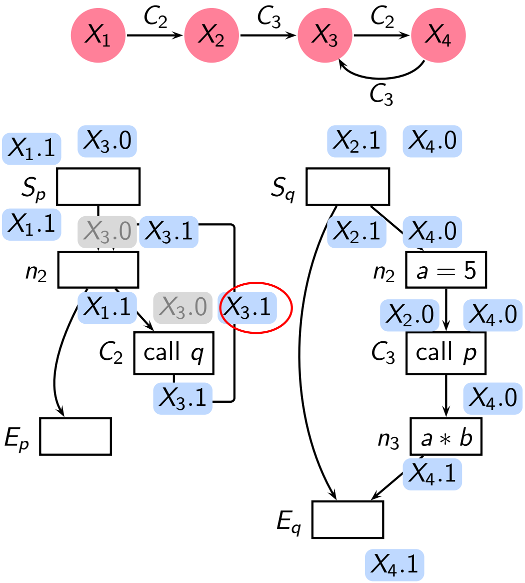
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	1

Out_{C_2} becomes 1 for X_3 which changes the value at In_{n_2} for X_3 to 1

Partially Available Expressions Analysis Using Value Contexts

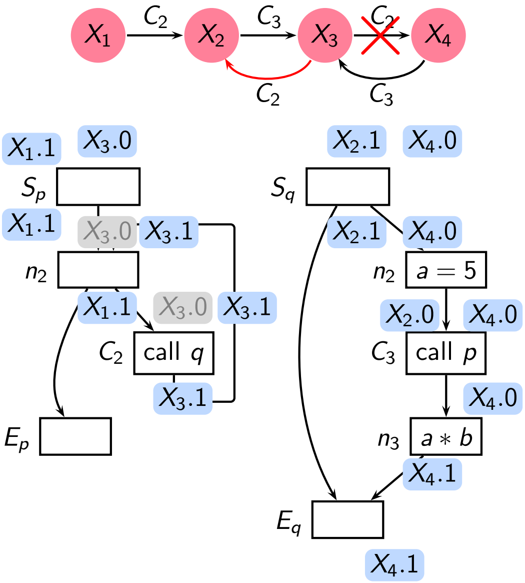


Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	1

In_{C_2} becomes 1 for X_3

Since we have a context for q with entry value 1 (X_2), we remove the transition $X_3 \xrightarrow{C_2} X_4$ and add the transition $X_3 \xrightarrow{C_2} X_2$

Partially Available Expressions Analysis Using Value Contexts

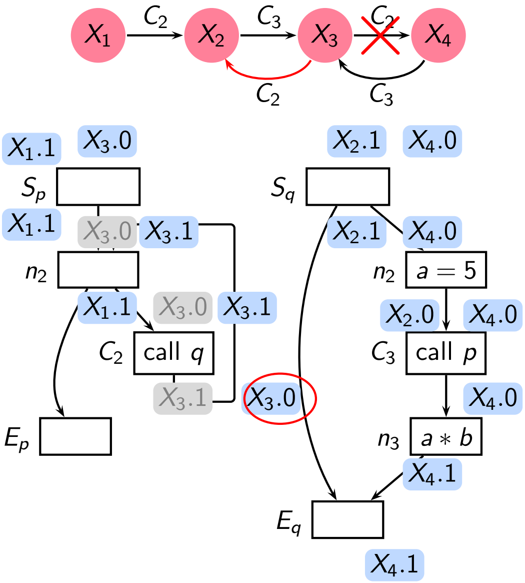


Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	1

In_{C_2} becomes 1 for X_3

Since we have a context for q with entry value 1 (X_2), we remove the transition $X_3 \xrightarrow{C_2} X_4$ and add the transition $X_3 \xrightarrow{C_2} X_2$

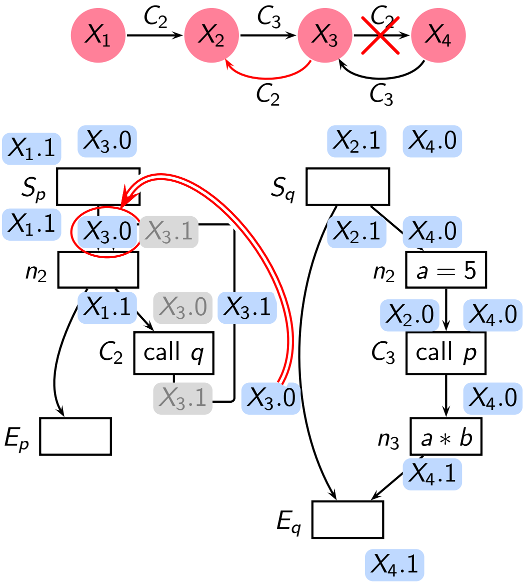
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1\rangle$	0
$X_2 = \langle q,1\rangle$	0
$X_3 = \langle p,0\rangle$	0
$X_4 = \langle q,0\rangle$	1

We use the *exitValue* of X_2 to compute the value of X_3 in Out_{C_2}

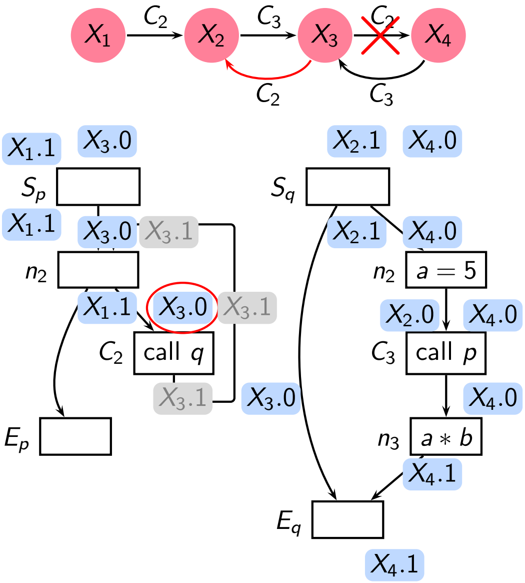
Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p, 1\rangle$	0
$X_2 = \langle q, 1\rangle$	0
$X_3 = \langle p, 0\rangle$	0
$X_4 = \langle q, 0\rangle$	1

The value of X_3 in In_{n_2} once again becomes 0

Partially Available Expressions Analysis Using Value Contexts

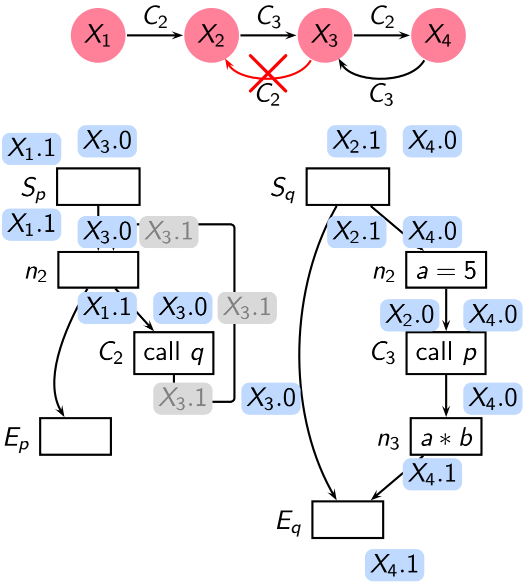


Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	1

The value of X_3 in In_{C_2} once again becomes 0

The transition from X_3 needs to be restored to $X_3 \xrightarrow{C_2} X_4$ removing the transition $X_3 \xrightarrow{C_2} X_2$

Partially Available Expressions Analysis Using Value Contexts

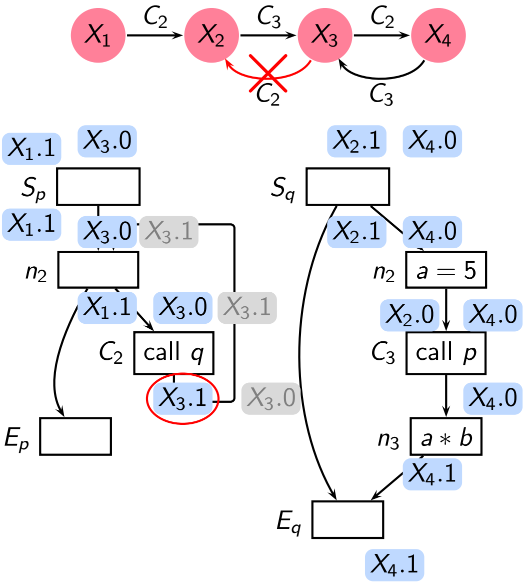


Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	1

The value of X_3 in In_{C_2} once again becomes 0

The transition from X_3 needs to be restored to $X_3 \xrightarrow{C_2} X_4$ removing the transition $X_3 \xrightarrow{C_2} X_2$

Partially Available Expressions Analysis Using Value Contexts

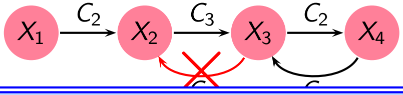


Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0
$X_2 = \langle q,1 \rangle$	0
$X_3 = \langle p,0 \rangle$	0
$X_4 = \langle q,0 \rangle$	1

We use the *exitValue* of X_4 to compute Out_{C_2} for X_3 which once again becomes 0

Thus we are back to the same situation

Partially Available Expressions Analysis Using Value Contexts



Context	*exitValue*
$X_1 = \langle p,1 \rangle$	0

- The process would not terminate so long as the processing of the nodes in the loop continues
- If the work list organization allows processing of E_p, then the *exitValue* of X_3 will also change to 1 which will lead to termination
- Our underlying flow functions are monotonic and a fixed point exists; non-termination is caused by the algorithm because its progress depends on the order of the nodes in the work list
- We avoid this problem by taking a meet at the exit of call nodes when the exit values of existing contexts are used at the call sites in the callers

$X_4.1$

Defining Value Context Method Using Data Flow Equations

- The overall data flow values Γ are sets of $X.v$ where X is a context and $v \in L$ is the underlying data flow value.
- We merge underlying data flow values only if the contexts are same

$$\begin{aligned}\Gamma_1 \uplus \Gamma_2 \;=\; \big\{X.w \mid\; & X.u \in \Gamma_1 \;\wedge\; X.v \in \Gamma_2 \;\Rightarrow\; w = u \sqcap v,\\ & X.u \in \Gamma_1 \;\wedge\; X.v \notin \Gamma_2 \;\Rightarrow\; w = u,\\ & X.u \notin \Gamma_1 \;\wedge\; X.v \in \Gamma_2 \;\Rightarrow\; w = v\;\big\}\end{aligned}$$

  Effectively, if a context does not exist in Γ, its value is $\top$ in Γ

- Data flow variables for node n in procedure p are $In(p, n)$ and $Out(p, n)$
- The flow function for node n in procedure p is $f(p, n)$

Defining Value Context Method Using Data Flow Equations

We assume the following auxiliary functions

- Function *context* maintains the context information

  $\textit{context}(p, v)$ returns the context of procedure p for entry value v

  If no such context exists, the function creates a new context and returns it

- Function $\textit{exitValue}(X)$ returns the exit valus of context X

  If context X does not exist, the function returns $\top \in L$

- Function *gpred* extends the predcessor relation pred (which is local to a procedure) to a global level across procedures

$$\textit{gpred}(p, n) = \begin{cases} \{(q, m) \mid \text{ call site } m \text{ in } q \text{ calls } p\} & n \text{ is } S_p \\ \{(p, m) \mid m \in \textit{pred}(n)\} & \text{otherwise} \end{cases}$$

Defining Value Context Method Using Data Flow Equations

We define data flow equations for a forward data flow analysis

$$In(p,n) = \begin{cases} \{X.v \mid X = context(p,v), Y.v \in In(q,m), \\ \qquad (q,m) \in gpred(p,n)\} & n \text{ is } S_p \\ \\ \biguplus_{(p,m)\in gpred(p,n)} Out(p,m) & \text{otherwise} \end{cases}$$

$$Out(p,n) = \begin{cases} Out(p,n) \uplus \{X.v \mid X.v' \in In(p,m), \\ \qquad Y = context(q,v'), \\ \qquad v = exitValue(Y)\ \} & n \text{ calls } q \\ \\ \{X.v \mid X.v' \in In(p,m), v = f(p,n)(v')\} & \text{otherwise} \end{cases}$$

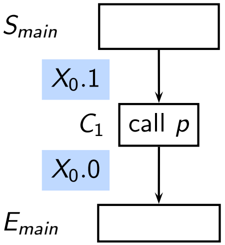
Value Contexts and Interprocedurally Valid Paths

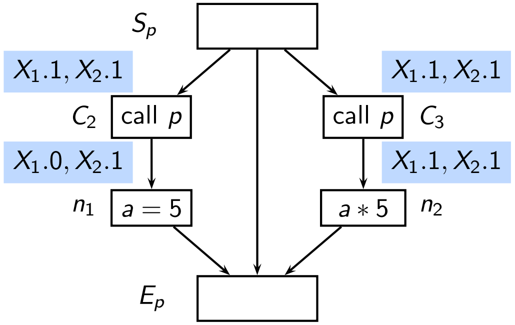
The role of value contexts in context sensitivity

- Value contexts preserve interprocedurally valid paths
- Value contexts consider only interprocedurally valid paths

We explain this with the help of an example by illustrating paths using a staircase diagram

Value Contexts and Interprocedurally Valid Paths: Example

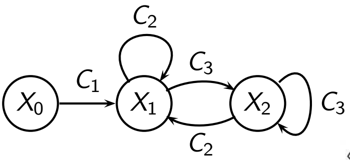




Context Transition Table

Context	*exitValue*
X_0 : $\langle \text{main}, 0 \rangle$	1
X_1 : $\langle p, 0 \rangle$	1
X_2 : $\langle p, 1 \rangle$	1

Context Transition Graph



Value Contexts and Interprocedurally Valid Paths: Example

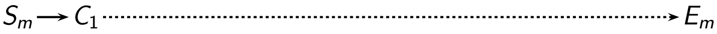
We explain the data flow value at the entry of C_2 by dividing the paths into the following two categories:

A. Paths in which the innermost recursion is along the call at C_2.

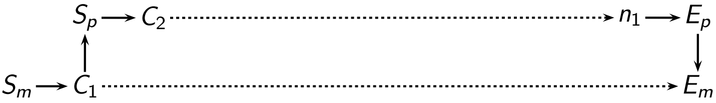
B. Paths in which the innermost recursion is along the call at C_3.

We draw the staircase diagrams of the example paths in the two categories

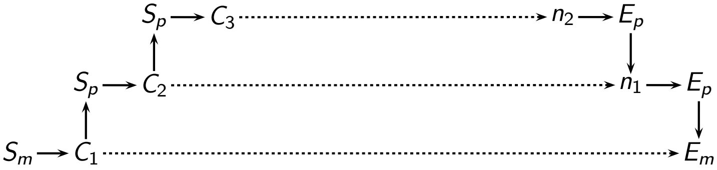
Innermost Recursion Along the Call at C_2



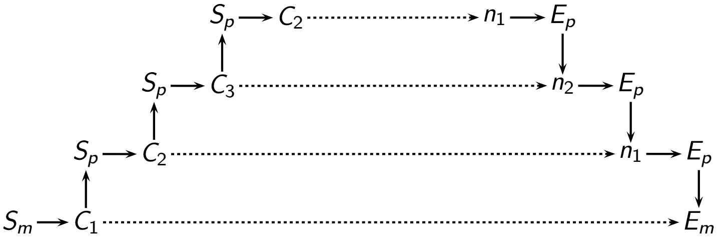
Innermost Recursion Along the Call at C_2



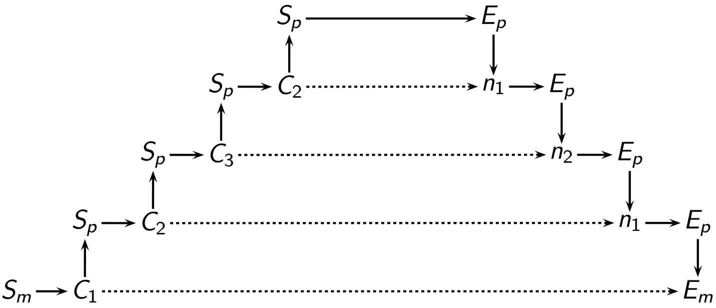
Innermost Recursion Along the Call at C_2



Innermost Recursion Along the Call at C_2

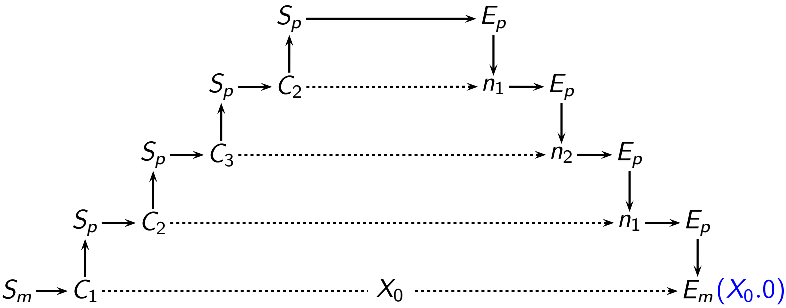


Innermost Recursion Along the Call at C_2



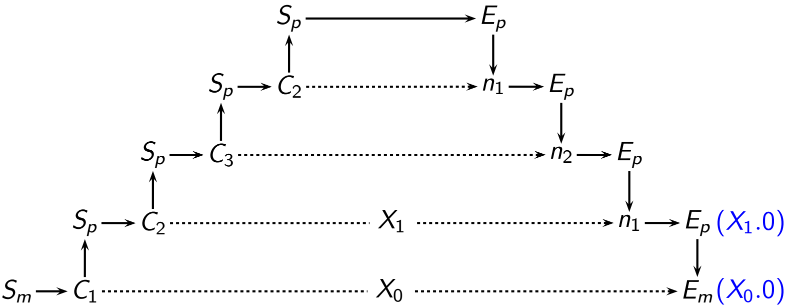
Innermost Recursion Along the Call at C_2

$BI$is 0



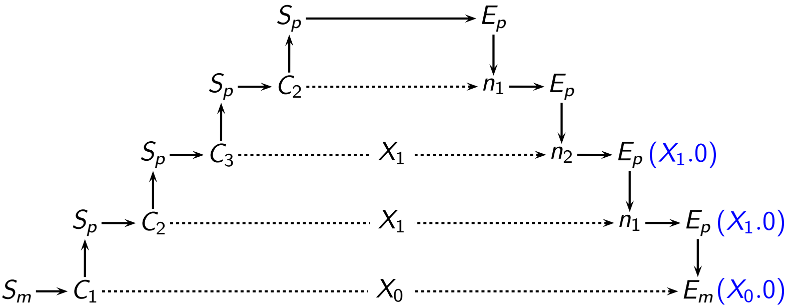
Innermost Recursion Along the Call at C_2

BI is 0



Innermost Recursion Along the Call at C_2

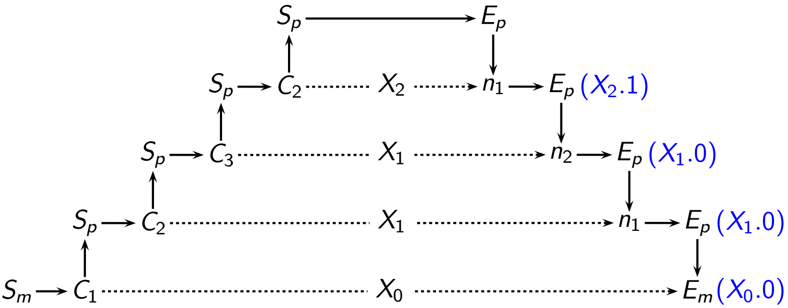
n_1 kills the liveness of a
New context is not required



Innermost Recursion Along the Call at C_2

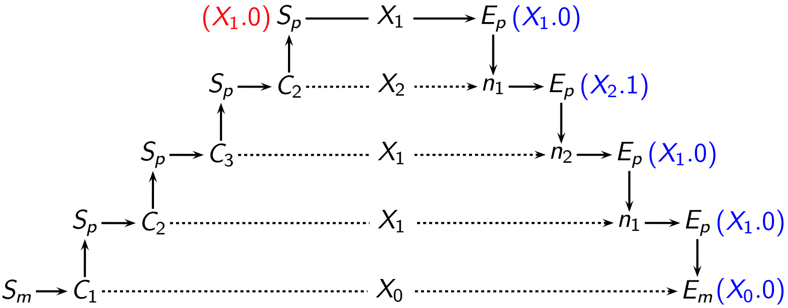
n_2 generates the liveness of a

New context is required



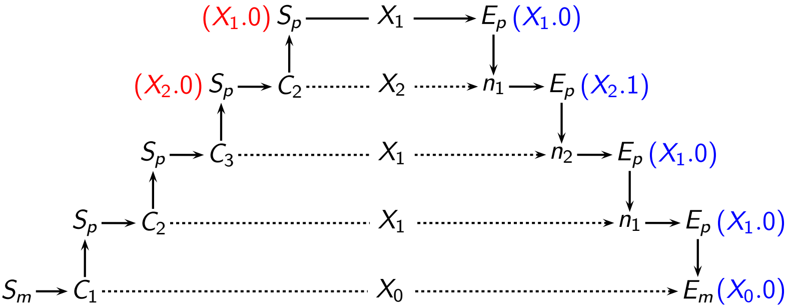
Innermost Recursion Along the Call at C_2

exitValue of X_1 is 0



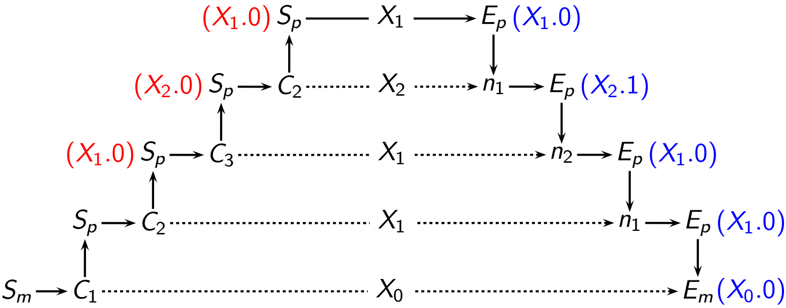
Innermost Recursion Along the Call at C_2

exitValue of X_2 is 0



Innermost Recursion Along the Call at C_2

exitValue of X_1 remains 0



$(X_1.0)$ S_p X_1 E_p $(X_1.0)$

$(X_2.0)$ S_p C_2 X_2 n_1 E_p $(X_2.1)$

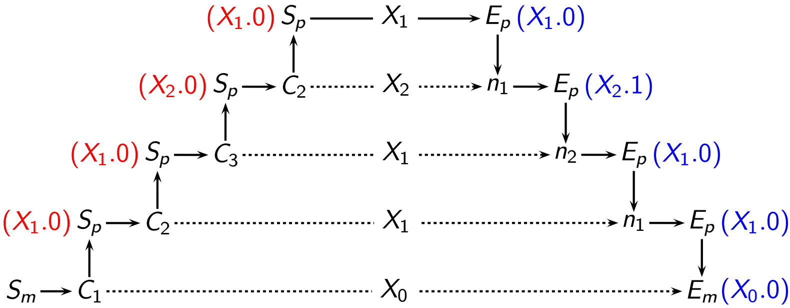
$(X_1.0)$ S_p C_3 X_1 n_2 E_p $(X_1.0)$

S_p C_2 X_1 n_1 E_p $(X_1.0)$

S_m C_1 X_0 E_m $(X_0.0)$

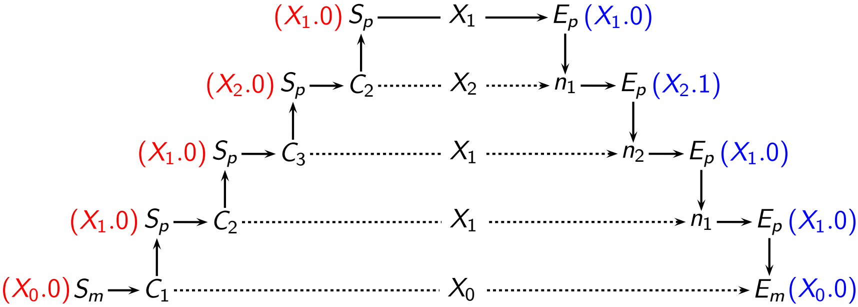
Innermost Recursion Along the Call at C_2

exitValue of X_1 remains 0



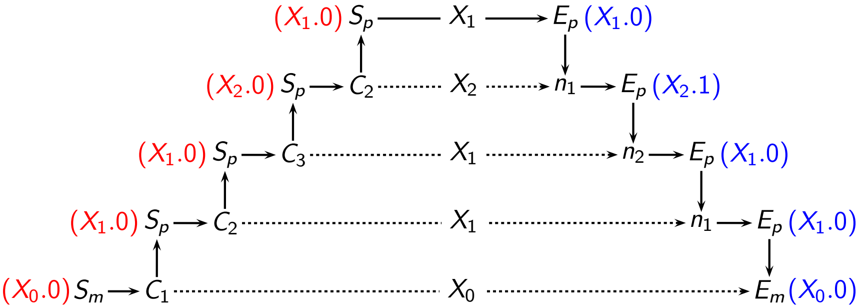
Innermost Recursion Along the Call at C_2

exitValue of X_0 is 0

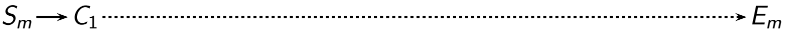


Innermost Recursion Along the Call at C_2

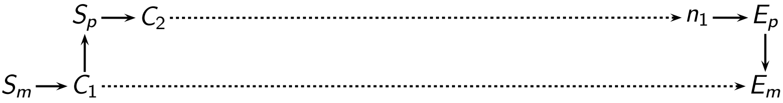
For this example, the innermost call determines the *exitValue* of contexts



Innermost Recursion Along the Call at C_3



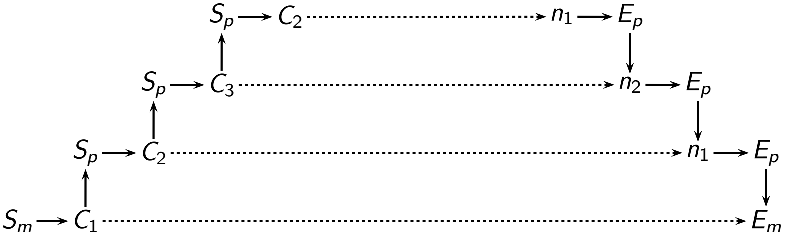
Innermost Recursion Along the Call at C_3



Innermost Recursion Along the Call at C_3

$$\begin{tikzcd}
 & & S_p \arrow[r] & C_3 \arrow[r, dotted] & n_2 \arrow[r] & E_p \arrow[d] & \\
 & S_p \arrow[r] & C_2 \arrow[u] \arrow[rrr, dotted] & & & n_1 \arrow[r] & E_p \arrow[d] \\
S_m \arrow[r] & C_1 \arrow[u] \arrow[rrrrr, dotted] & & & & & E_m
\end{tikzcd}$$

Innermost Recursion Along the Call at C_3



Innermost Recursion Along the Call at C_3

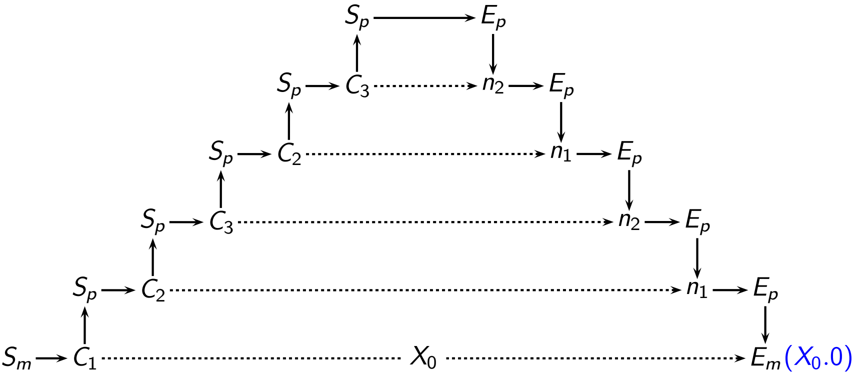
$$\begin{tikzcd}
 & & & & S_p \arrow[r] & C_3 \arrow[r, dashed] & n_2 \arrow[r] & E_p \arrow[d] & & & \\
 & & & S_p \arrow[r] & C_2 \arrow[u] \arrow[rrr, dashed] & & & n_1 \arrow[r] & E_p \arrow[d] & & \\
 & & S_p \arrow[r] & C_3 \arrow[u] \arrow[rrrrr, dashed] & & & & & n_2 \arrow[r] & E_p \arrow[d] & \\
 & S_p \arrow[r] & C_2 \arrow[u] \arrow[rrrrrrr, dashed] & & & & & & & n_1 \arrow[r] & E_p \arrow[d] \\
S_m \arrow[r] & C_1 \arrow[u] \arrow[rrrrrrrrr, dashed] & & & & & & & & & E_m
\end{tikzcd}$$

Innermost Recursion Along the Call at C_3

$$\begin{tikzcd}
 & & & & & S_p \arrow[r] & E_p \arrow[d] & & & & \\
 & & & & S_p \arrow[r] & C_3 \arrow[u] \arrow[r, dotted] & n_2 \arrow[r] & E_p \arrow[d] & & & \\
 & & & S_p \arrow[r] & C_2 \arrow[u] \arrow[rrr, dotted] & & & n_1 \arrow[r] & E_p \arrow[d] & & \\
 & & S_p \arrow[r] & C_3 \arrow[u] \arrow[rrrrr, dotted] & & & & & n_2 \arrow[r] & E_p \arrow[d] & \\
 & S_p \arrow[r] & C_2 \arrow[u] \arrow[rrrrrrr, dotted] & & & & & & & n_1 \arrow[r] & E_p \arrow[d] \\
S_m \arrow[r] & C_1 \arrow[u] \arrow[rrrrrrrrr, dotted] & & & & & & & & & E_m
\end{tikzcd}$$

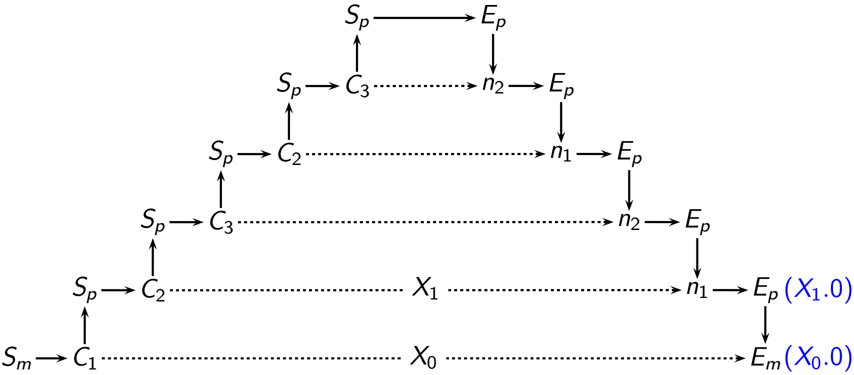
Innermost Recursion Along the Call at C_3

$BI$is 0



Innermost Recursion Along the Call at C_3

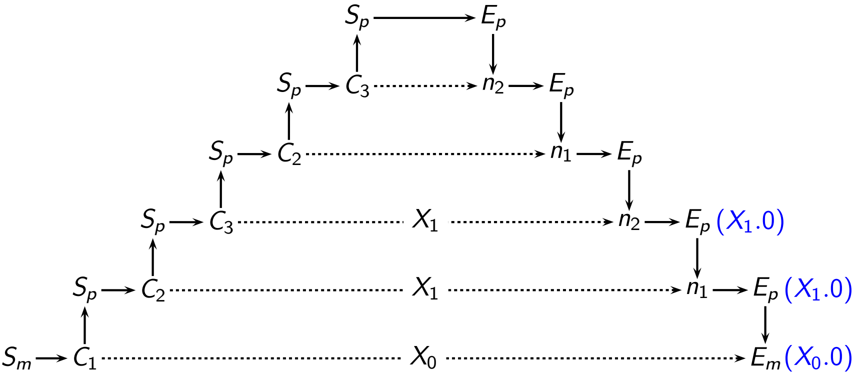
n_1 kills the liveness of a
New context is not required



Innermost Recursion Along the Call at C_3

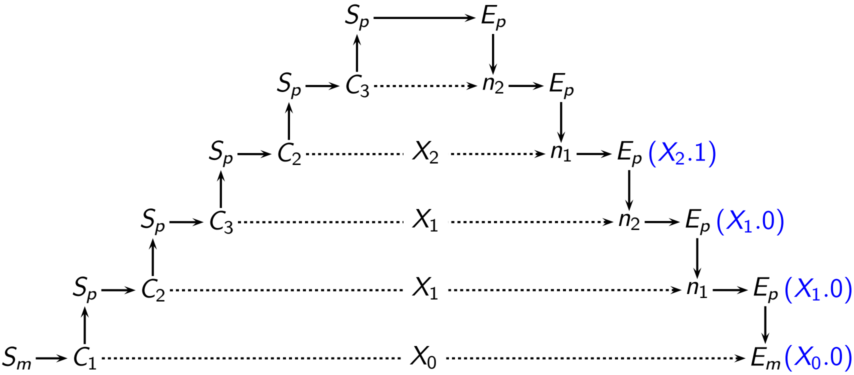
n_1 kills the liveness of a

New context is not required



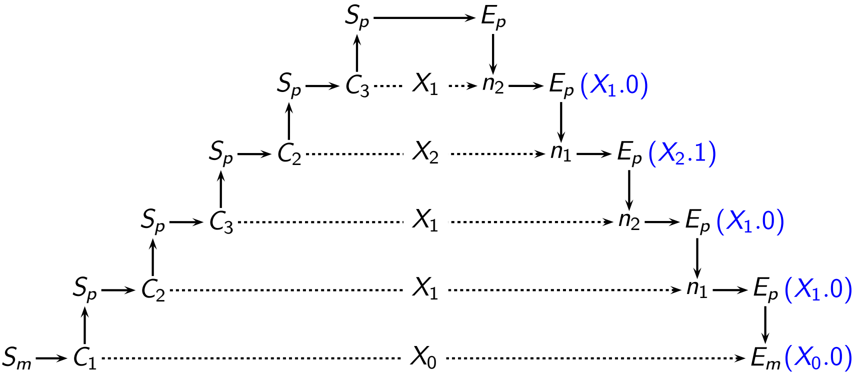
Innermost Recursion Along the Call at C_3

n_2 generates the liveness of a
New context is required



Innermost Recursion Along the Call at C_3

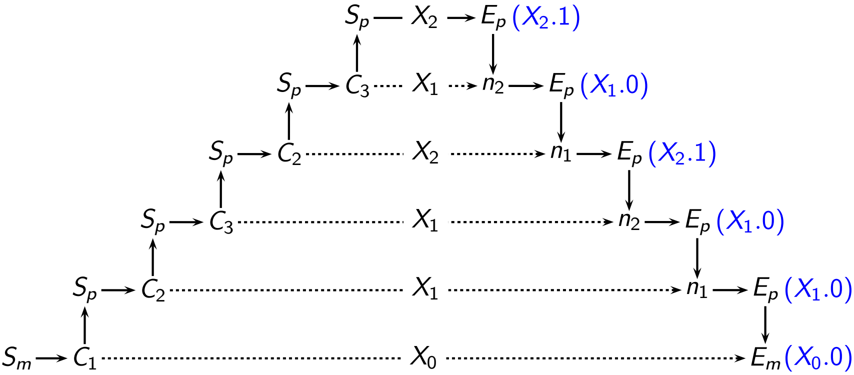
n_1 kills the liveness of a
New context is not required



Innermost Recursion Along the Call at C_3

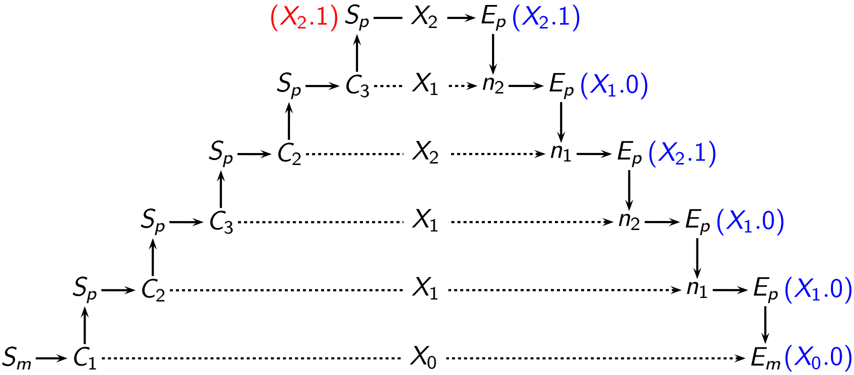
n_2 generates the liveness of a

New context is not required


S_p — X_2 → E_p $(X_2.1)$
S_p → C_3 ····· X_1 ···→ n_2 → E_p $(X_1.0)$
S_p → C_2 ········ X_2 ········→ n_1 → E_p $(X_2.1)$
S_p → C_3 ········ X_1 ········→ n_2 → E_p $(X_1.0)$
S_p → C_2 ········ X_1 ········→ n_1 → E_p $(X_1.0)$
S_m → C_1 ········ X_0 ········→ E_m $(X_0.0)$

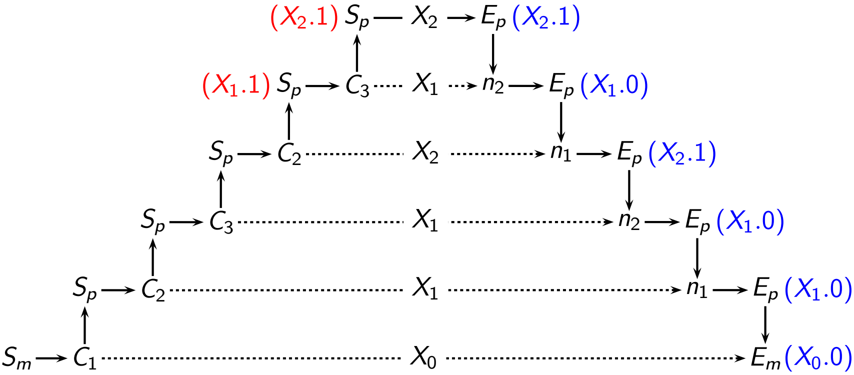
Innermost Recursion Along the Call at C_3

exitValue of X_2 is 1 (after merging with previous value 0)


$(X_2.1)$ S_p — X_2 → E_p $(X_2.1)$
S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
S_p → C_2 X_2 n_1 → E_p $(X_2.1)$
S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
S_p → C_2 X_1 n_1 → E_p $(X_1.0)$
S_m → C_1 X_0 E_m $(X_0.0)$

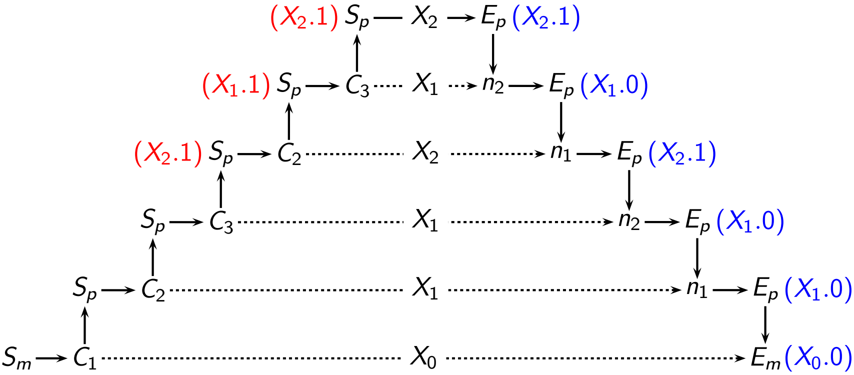
Innermost Recursion Along the Call at C_3

exitValue of X_1 is 1 (after merging with previous value 0)


$(X_2.1)$ S_p — X_2 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
S_p → C_2 X_2 n_1 → E_p $(X_2.1)$
S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
S_p → C_2 X_1 n_1 → E_p $(X_1.0)$
S_m → C_1 X_0 E_m $(X_0.0)$

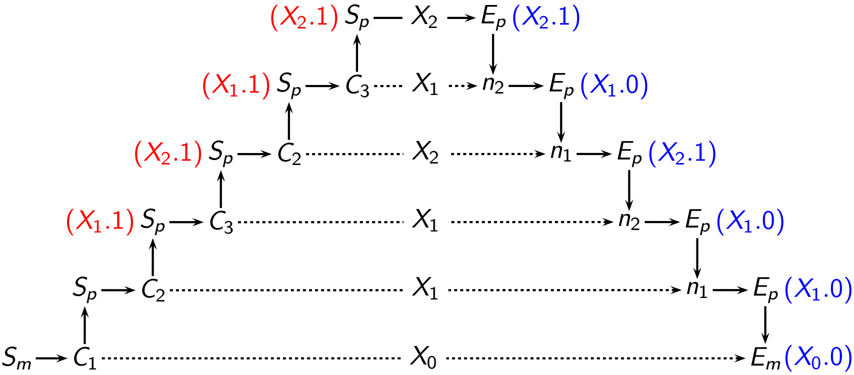
Innermost Recursion Along the Call at C_3

exitValue of X_2 remains 1


$(X_2.1)$ S_p — X_2 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
$(X_2.1)$ S_p → C_2 X_2 n_1 → E_p $(X_2.1)$
S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
S_p → C_2 X_1 n_1 → E_p $(X_1.0)$
S_m → C_1 X_0 E_m $(X_0.0)$

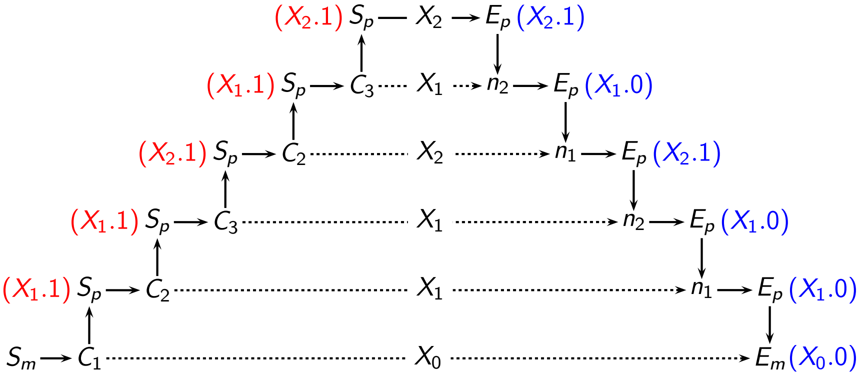
Innermost Recursion Along the Call at C_3

exitValue of X_1 remains 1


$(X_2.1)$ S_p — X_2 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
$(X_2.1)$ S_p → C_2 X_2 n_1 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
S_p → C_2 X_1 n_1 → E_p $(X_1.0)$
S_m → C_1 X_0 E_m $(X_0.0)$

Innermost Recursion Along the Call at C_3

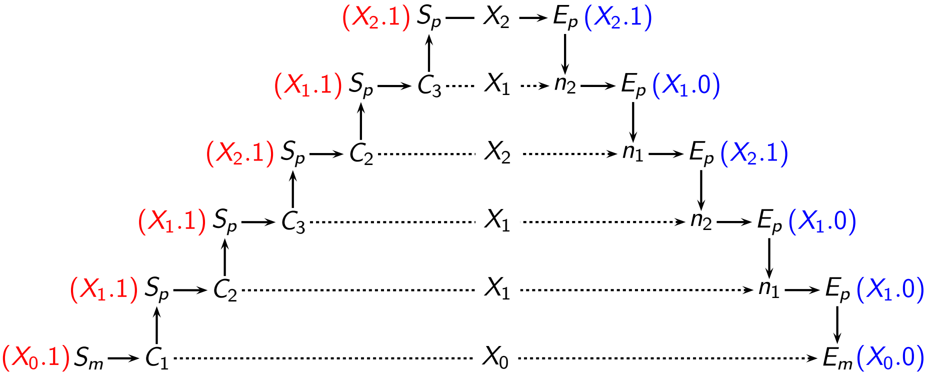
exitValue of X_1 remains 1



$(X_2.1)$ S_p — X_2 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
$(X_2.1)$ S_p → C_2 X_2 n_1 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
$(X_1.1)$ S_p → C_2 X_1 n_1 → E_p $(X_1.0)$
S_m → C_1 X_0 E_m $(X_0.0)$

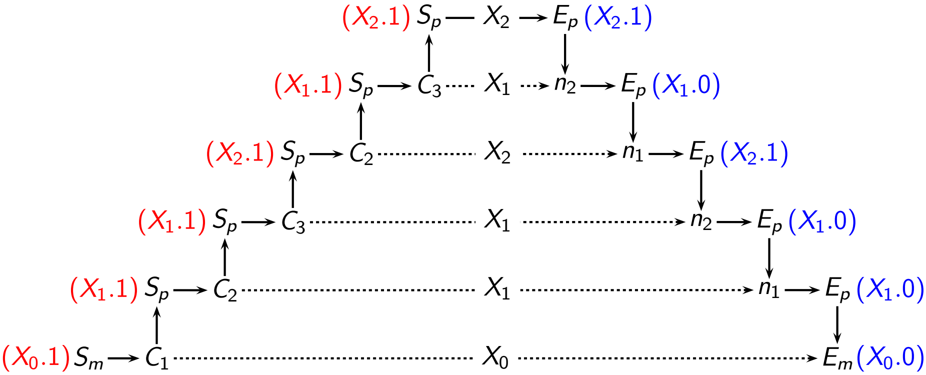
Innermost Recursion Along the Call at C_3

exitValue of X_0 is 1 (after merging with previous value 0)


$(X_2.1)$ S_p — X_2 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 ⋯ X_1 ⋯→ n_2 → E_p $(X_1.0)$
$(X_2.1)$ S_p → C_2 ⋯ X_2 ⋯→ n_1 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 ⋯ X_1 ⋯→ n_2 → E_p $(X_1.0)$
$(X_1.1)$ S_p → C_2 ⋯ X_1 ⋯→ n_1 → E_p $(X_1.0)$
$(X_0.1)$ S_m → C_1 ⋯ X_0 ⋯→ E_m $(X_0.0)$

Innermost Recursion Along the Call at C_3

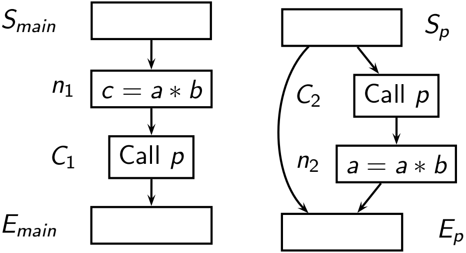
Again, the innermost call determines the *exitValue* of contexts
The final values at the entry of C_3 are 1 (union of 1 and 0)



$(X_2.1)$ S_p — X_2 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
$(X_2.1)$ S_p → C_2 X_2 n_1 → E_p $(X_2.1)$
$(X_1.1)$ S_p → C_3 X_1 n_2 → E_p $(X_1.0)$
$(X_1.1)$ S_p → C_2 X_1 n_1 → E_p $(X_1.0)$
$(X_0.1)$ S_m → C_1 X_0 E_m $(X_0.0)$

Tutorial Problem #1 for Value Contexts

```
int a,b,c;
void main()
{   c = a*b;
    p();
}
void p()
{  if (...)
   {  p();
   Is a*b available?
         a = a*b;
   }
}
```



Tutorial Problem #2 for Value Contexts

Perform interprocedural live variables analysis using value contexts

```
main()
{
   p();
}
```

```
p()
{  while (...)
   {  printf ("%d\n",a);
      p();
   }
}
```

Observe the change in edges in the transition diagram

Tutorial Problem #3 for Value Contexts

Perform interprocedural available expressions analysis using value contexts

```
main()
{
   c = a*b;
   p();
}
```

```
p()
{
   while (a > b)
   {
       p();
       a = a*b;
   }
}
```

Observe the change in edges in the transition diagram

Tutorial Problem #4 for Value Contexts

Perform interprocedural available expressions analysis using value contexts

```
main()
{
        c = a*b;
        p();
        a = a*b;
}
```

```
p()
{   if (...)
        {   a = a*b;
            p();
        }
        else if (...)
        {   c = a * b;
            p();
            c = a;
        }
        else
            ; /* ignore */
}
```

Tutorial Problem #5 for Value Contexts

Perform interprocedural live variables analysis using value contexts

```
main()
{
  a = 5; b = 3;
  c = 7; d = 2;
  p();
  a = a + 2;
  e = c+d;
  d = a*b;
  q();
  print a+c+e;
}
```

```
p()
{
  b = 2;
  if (b<d)
     c = a+b;
  else
     q();
  print c+d;
}
```

```
q()
{
  a = 1;
  p();
  a = a*b;
}
```

Context sensitivity: e is live on entry to p but not before its call in main

Result of Tutorial #5

```
main()
{
  a = 5; b = 3;
  c = 7; d = 2;
  /*{a,d}*/
  p();
  /*{a,b,c,d}*/
  a = a + 2;
  e = c+d;
  /*{a,b,e}*/
  d = a*b;
  /*{d,e}*/
  q();
  /*{a,c,e}*/
  print a+c+e;
}
```

```
p()
{  /*{a,d,e}*/
  b = 2;
  if (b<d)
    /*{a,b,d,e}*/
     c = a+b;
  else
    /*{d,e}*/
     q();
  /*{a,b,c,d,e}*/
  print c+d;
}
```

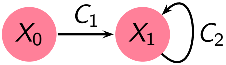
```
q()
{
  /*{d,e}*/
  a = 1;
  /*{a,d,e}*/
  p();
  /*{a,b,c,d,e}*/
  a = a*b;
}
```

Tutorial Problem #6: Interprocedural Points-to Analysis

```
main()
{  x = &y;
   z = &x;
   y = &z;
   p(); /* C1 */
}

p()
{  if (...)
   {  p(); /* C2 */
      x = *x;
   }
}
```

Value contexts method requires three contexts as shown below in the transition diagram



Reaching Definitions Analysis in GCC 4.0

Program	LoC	$\#F$	$\#C$	$3K$ length bound				Proposed Approach		
				K	#CS	Max	Time	#CS	Max	Time
hanoi	33	2	4	4	100000+	99922	3973 $\times 10^3$	8	7	2.37
bit_gray	53	5	11	7	100000+	31374	2705 $\times 10^3$	17	6	3.83
analyzer	288	14	20	2	21	4	20.33	21	4	1.39
distray	331	9	21	6	96	28	322.41	22	4	1.11
mason	350	9	13	8	100000+	22143	432 $\times 10^3$	14	4	0.43
fourinarow	676	17	45	5	510	158	397.76	46	7	1.86
sim	1146	13	45	8	100000+	33546	1427 $\times 10^3$	211	105	234.16
181_mcf	1299	17	24	6	32789	32767	484 $\times 10^3$	41	11	5.15
256_bzip2	3320	63	198	7	492	63	258.33	406	34	200.19

- LoC is the number of lines of code,
- $\#F$ is the number of procedures,
- $\#C$ is the number of call sites,
- $\#CS$ is the number of call strings
- Max denotes the maximum number of call strings reaching any node.
- Analysis time is in milliseconds.

(Implementation was carried out by Seema Ravandale.)

Some Observations

- Compromising on precision may not be necessary for efficiency.
- Separating the necessary information from redundant information is much more significant.
- Data flow propagation in real programs seems to involve only a small subset of all possible values.

  Much fewer changes than the theoretically possible worst case number of changes.
- A precise modelling of the process of analysis is often an eye opener.

Some Observations

- Compromising on precision may not be necessary for efficiency.
- Separating the necessary information from redundant information is much more significant.
- Data flow propagation in real programs seems to involve only a small subset of all possible values.

  Much fewer changes than the theoretically possible worst case number of changes.
- A precise modelling of the process of analysis is often an eye opener.

$$\#\text{ distinct tagged values} = \text{Min}\ (\#\text{ actual contexts},\ \#\text{ actual data flow values})$$