Interprocedural Data Flow Analysis

Uday Khedker

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

October 2015

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
- Value context based approach

Part 2

Issues in Interprocedural Analysis

Interprocedural Analysis: Overview

- Extends the scope of data flow analysis across procedure boundaries

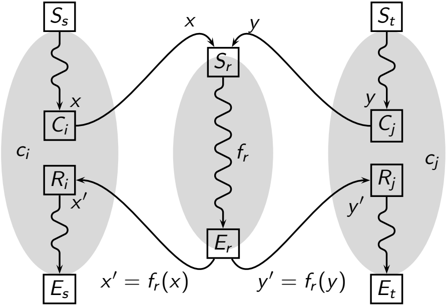
  Incorporates the effects of

  - 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

Inherited and Synthesized Data Flow Information



Data Flow Information	
x	Inherited by procedure r from call site c_i in procedure s
y	Inherited by procedure r from call site c_j in procedure t
x'	Synthesized by procedure r in s at call site procedure c_i
y'	Synthesized by procedure r in t at call site procedure c_j

Inherited and Synthesized Data Flow Information

- Example of uses of inherited data flow information

  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?

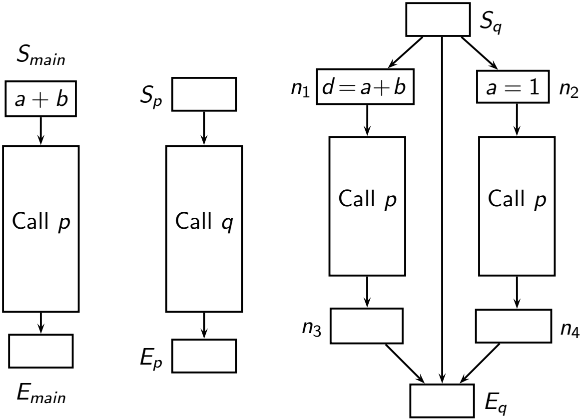
- Examples of uses of synthesized data flow information

  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)

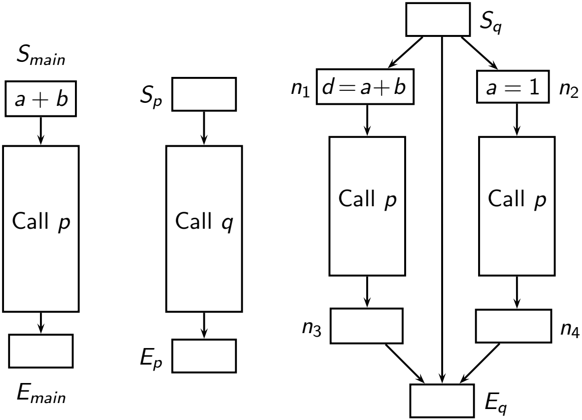
- Most of the above questions may have a *May* or *Must* qualifier

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



Supergraphs of procedures

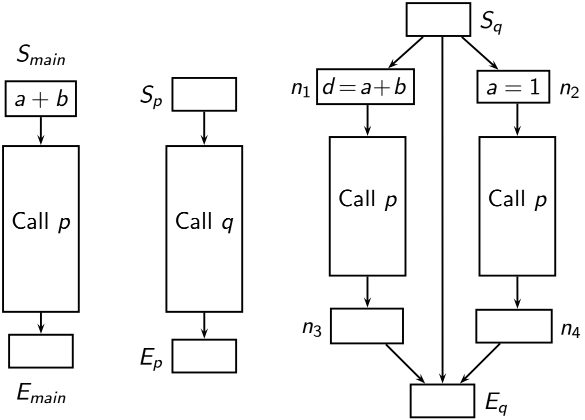
Program Representation for Interprocedural Data Flow Analysis: Call Multi-Graph



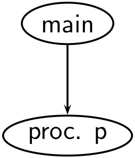
Supergraphs of procedures

Call multi-graph

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

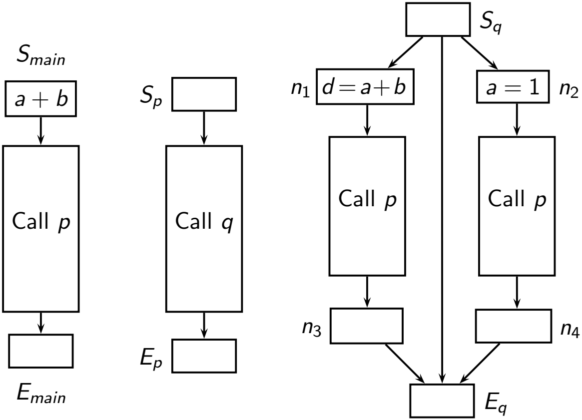


Supergraphs of procedures

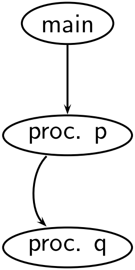


Call multi-graph

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

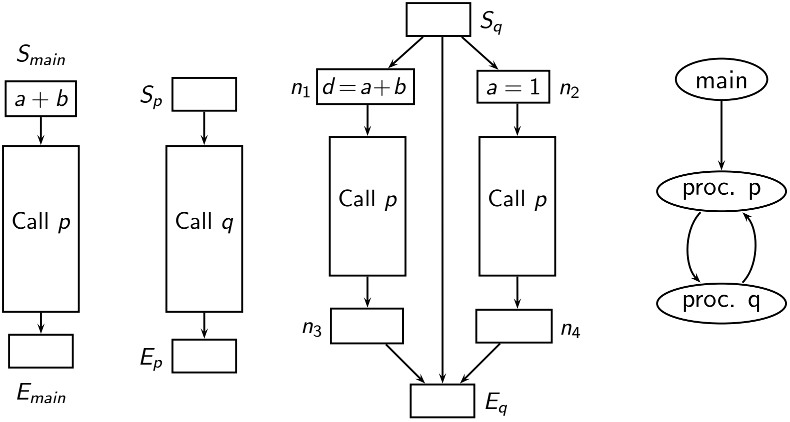


Supergraphs of procedures



Call multi-graph

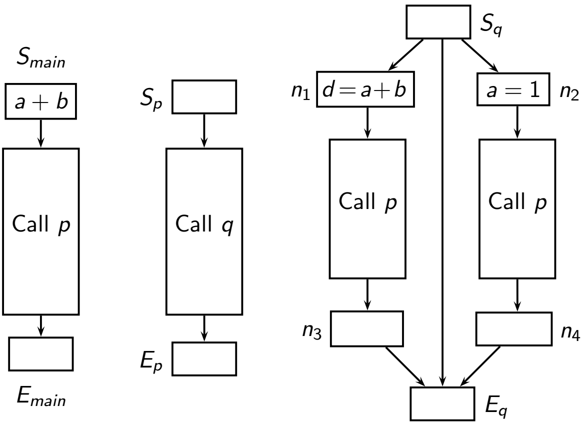
Program Representation for Interprocedural Data Flow Analysis: Call Multi-Graph



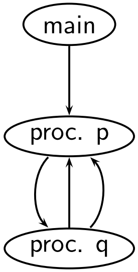
Supergraphs of procedures

Call multi-graph

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

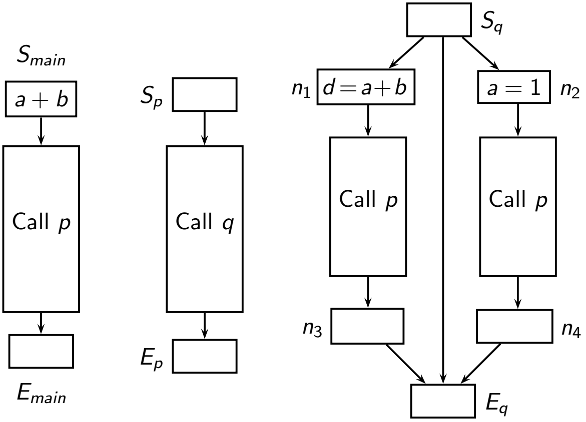


Supergraphs of procedures

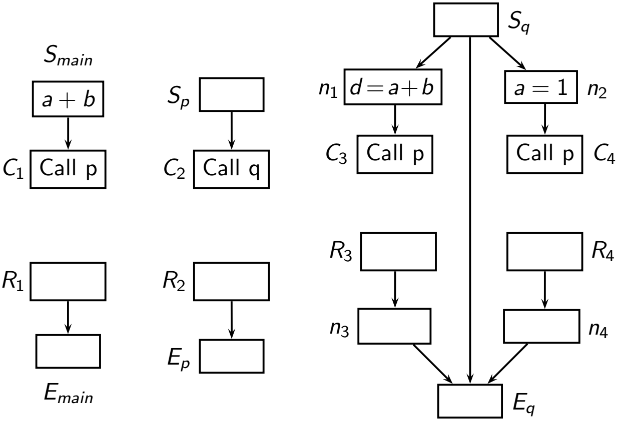


Call multi-graph

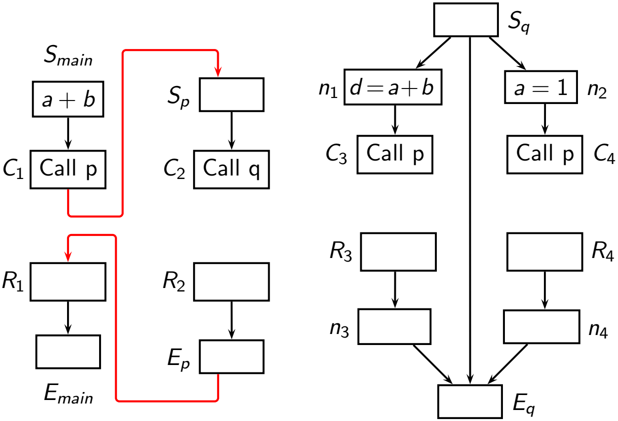
Program Representation for Interprocedural Data Flow Analysis: Supergraph



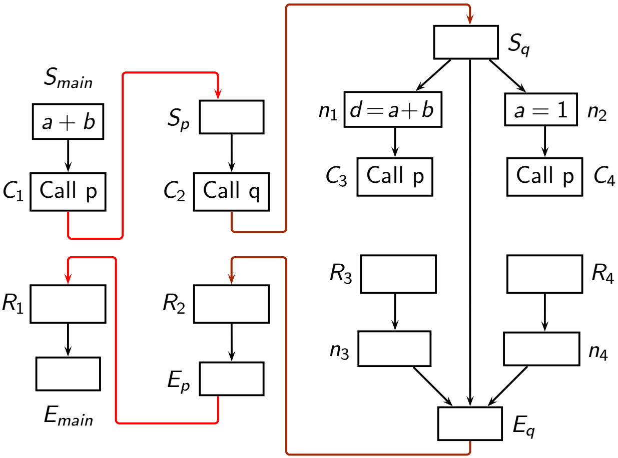
Program Representation for Interprocedural Data Flow Analysis: Supergraph



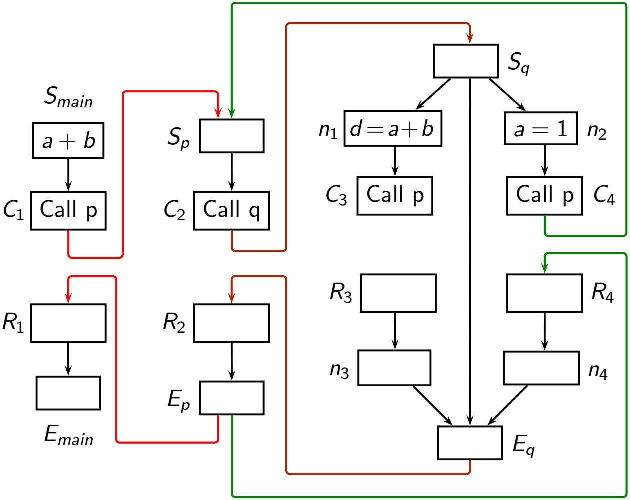
Program Representation for Interprocedural Data Flow Analysis: Supergraph



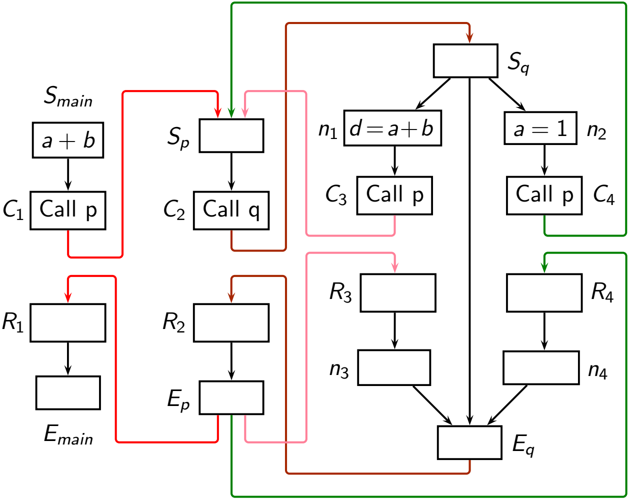
Program Representation for Interprocedural Data Flow Analysis: Supergraph



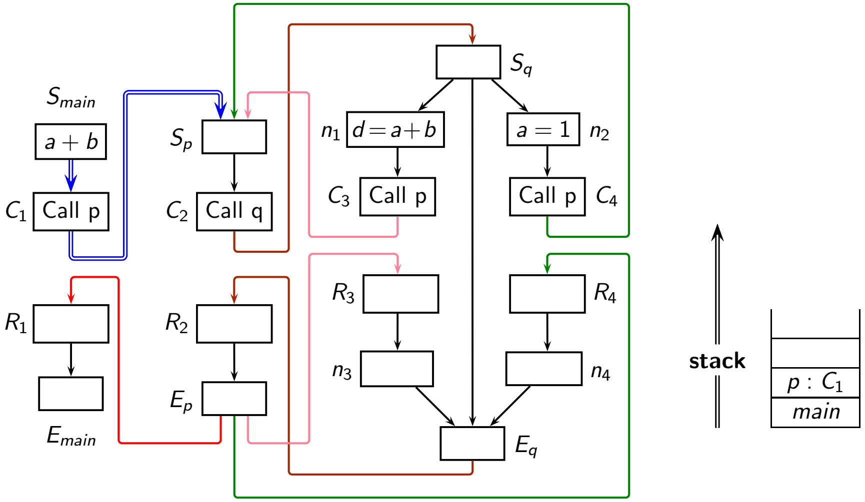
Program Representation for Interprocedural Data Flow Analysis: Supergraph



Program Representation for Interprocedural Data Flow Analysis: Supergraph

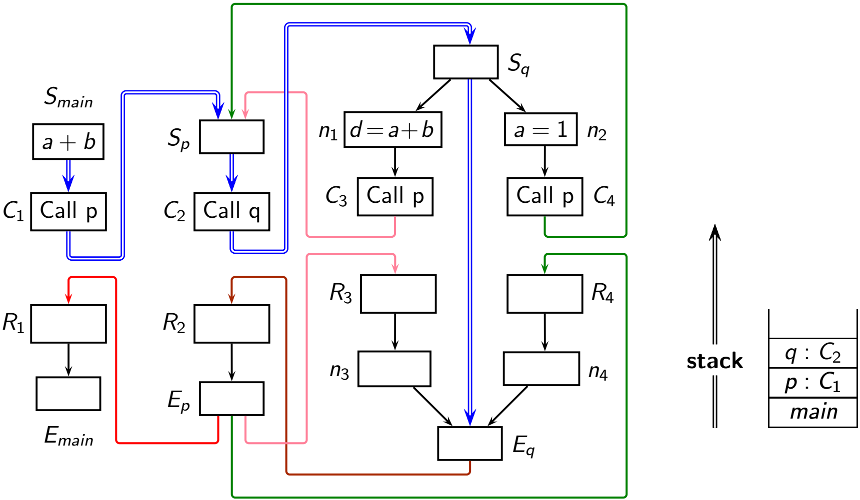


Validity of Interprocedural Control Flow Paths



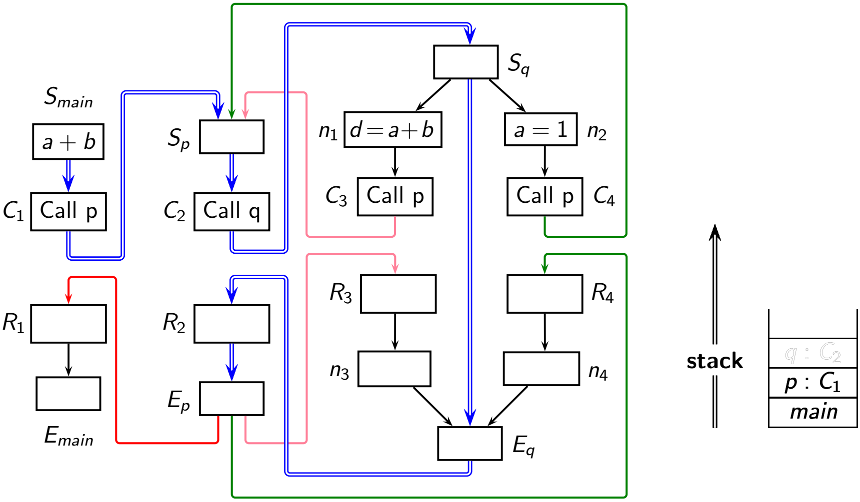
Interprocedurally valid control flow path

Validity of Interprocedural Control Flow Paths



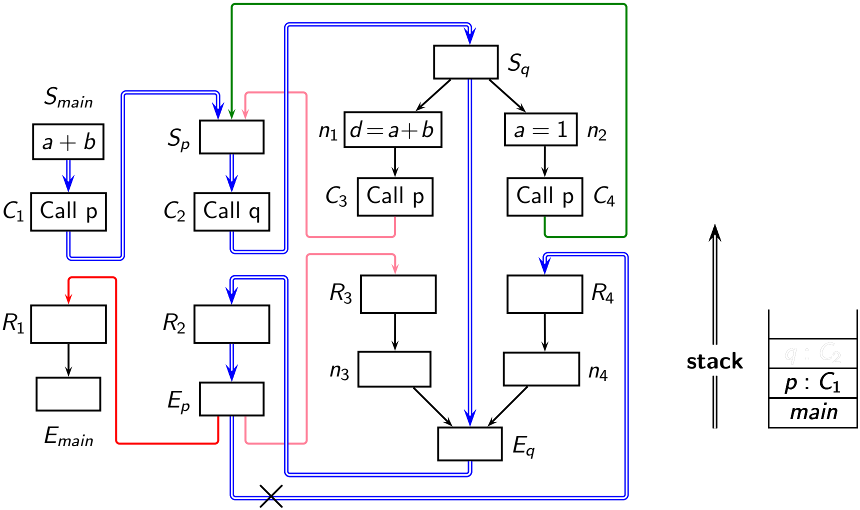
Interprocedurally valid control flow path

Validity of Interprocedural Control Flow Paths



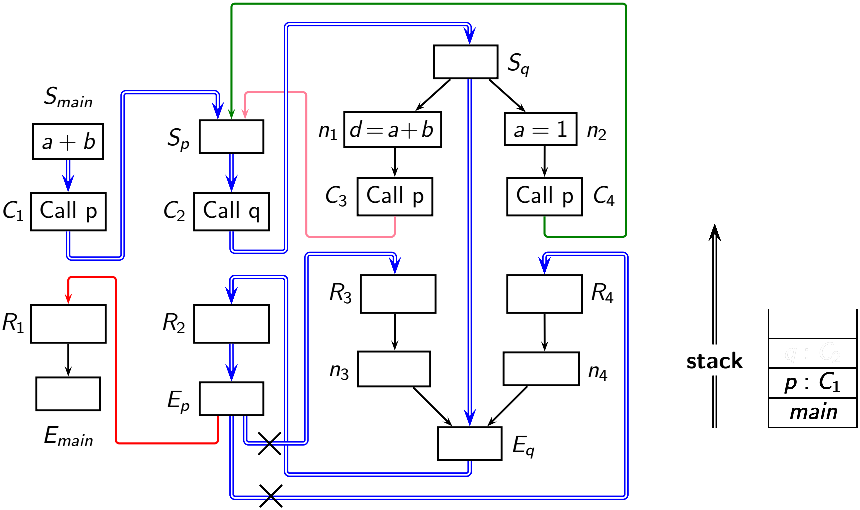
Interprocedurally valid control flow path

Validity of Interprocedural Control Flow Paths



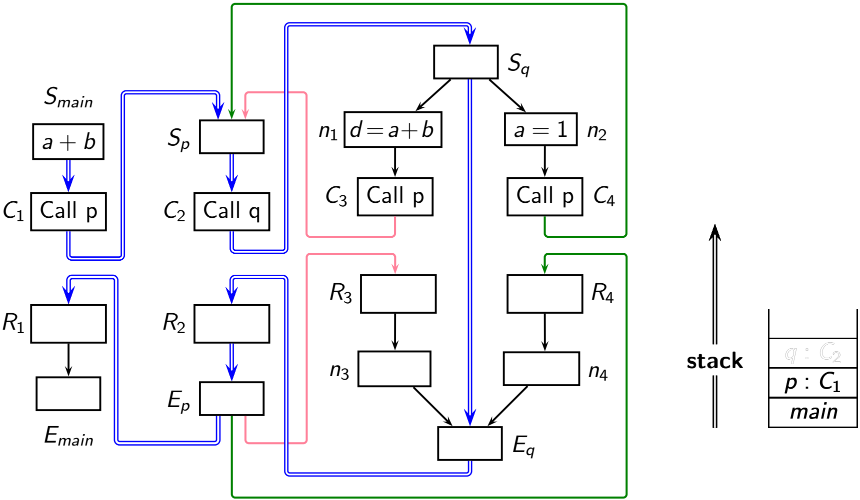
Interprocedurally invalid control flow path

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

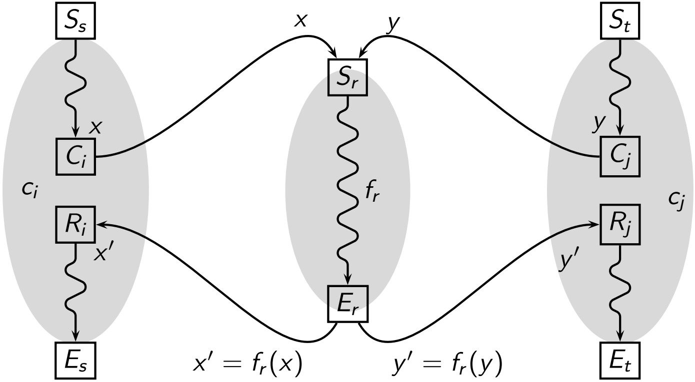
- Flow sensitive analysis:
  Considers intraprocedurally valid paths
- Context sensitive analysis:
  Considers interprocedurally valid paths
- For maximum statically attainable precision , analysis must be both flow and context sensitive

Flow and Context Sensitivity

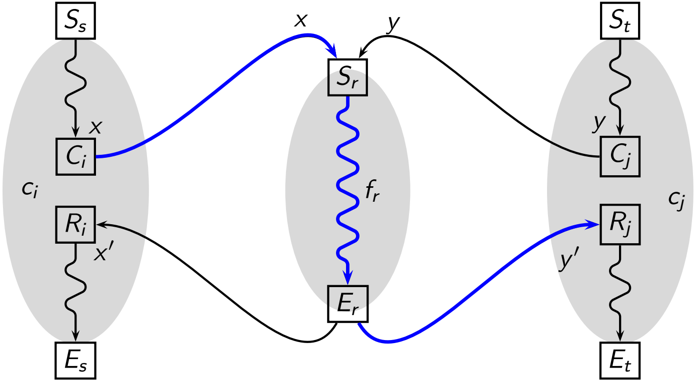
- Flow sensitive analysis:
  Considers intraprocedurally valid paths
- Context sensitive analysis:
  Considers interprocedurally valid paths
- For maximum statically attainable precision, analysis must be both flow and context sensitive

MFP computation restricted to valid paths only

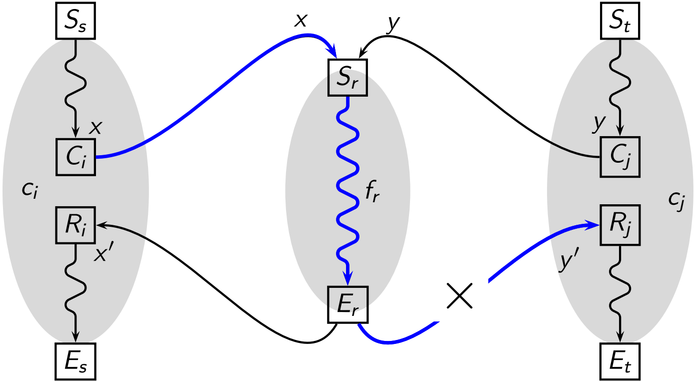
Context Sensitivity in Interprocedural Analysis



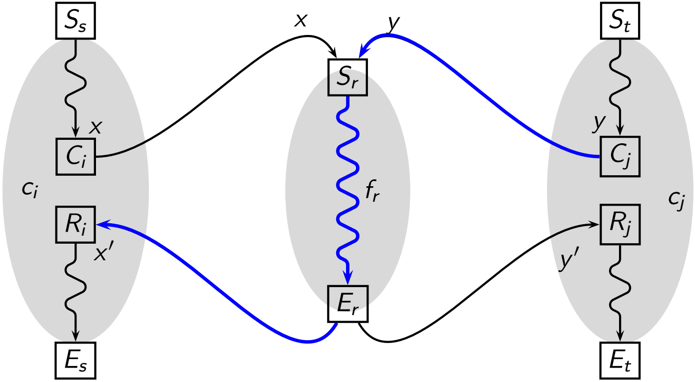
Context Sensitivity in Interprocedural Analysis



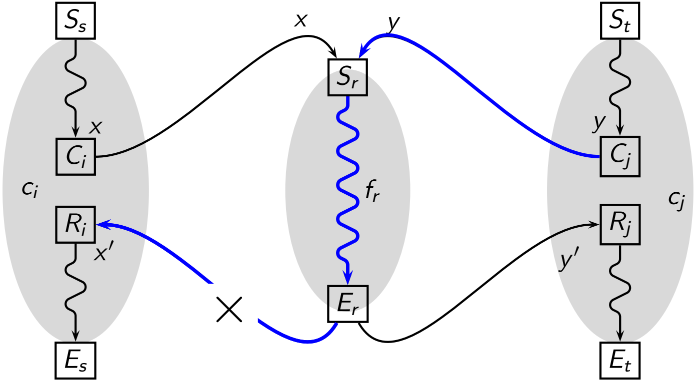
Context Sensitivity in Interprocedural Analysis



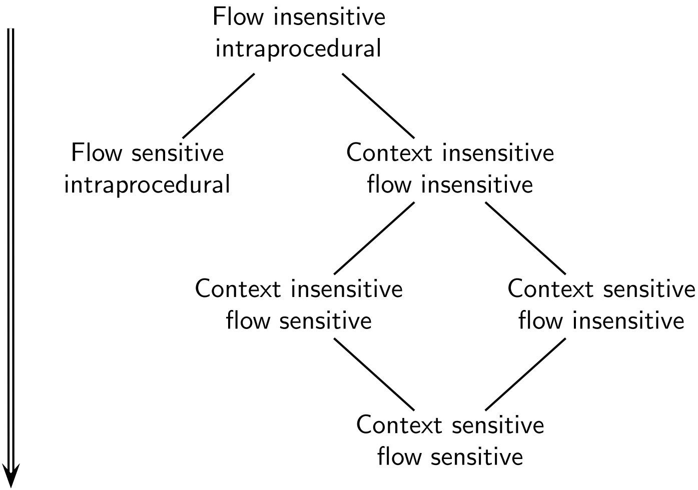
Context Sensitivity in Interprocedural Analysis



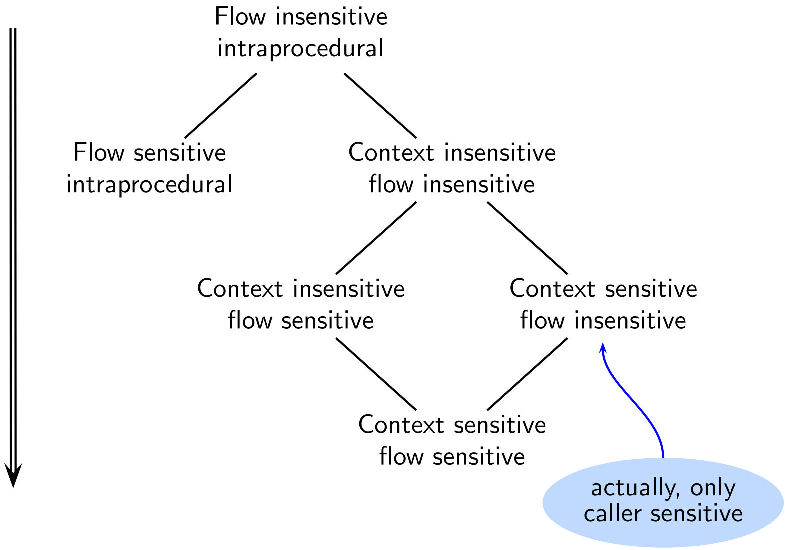
Context Sensitivity in Interprocedural Analysis



Increasing Precision in Data Flow Analysis



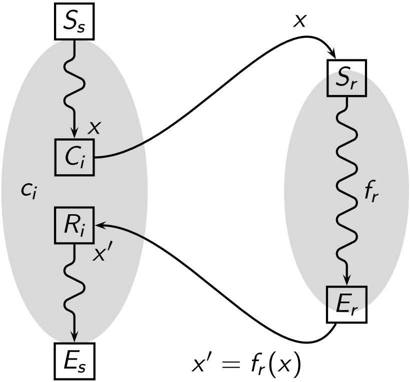
Increasing Precision in Data Flow Analysis



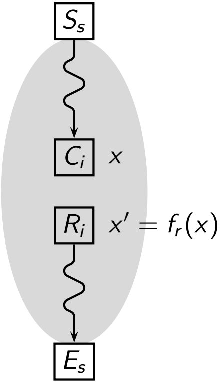
Part 3

Classical Functional Approach

Functional Approach



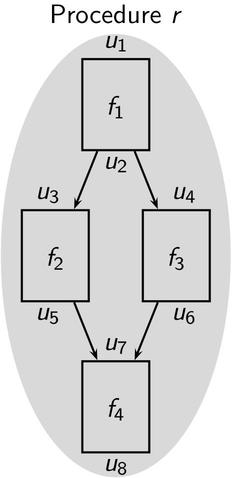
Functional Approach



- Compute summary flow functions for each procedure
- Use summary flow functions as the flow function for a call block

Notation for Summary Flow Function

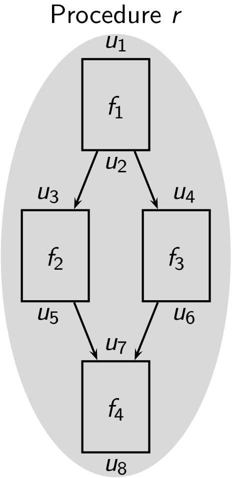
For simplicity forward flow is assumed



Notation for Summary Flow Function

For simplicity forward flow is assumed

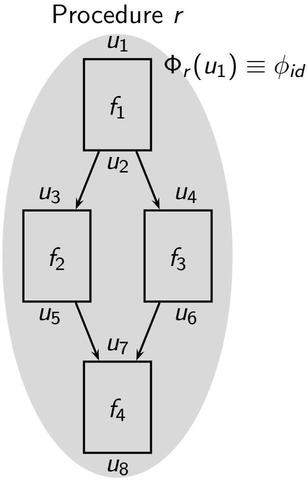
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Notation for Summary Flow Function

For simplicity forward flow is assumed

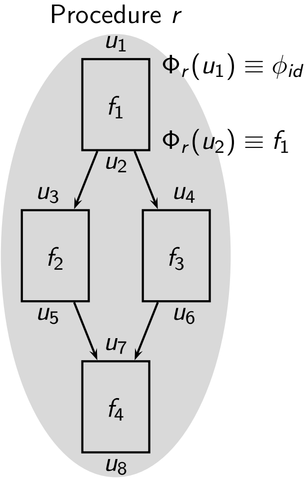
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Notation for Summary Flow Function

For simplicity forward flow is assumed

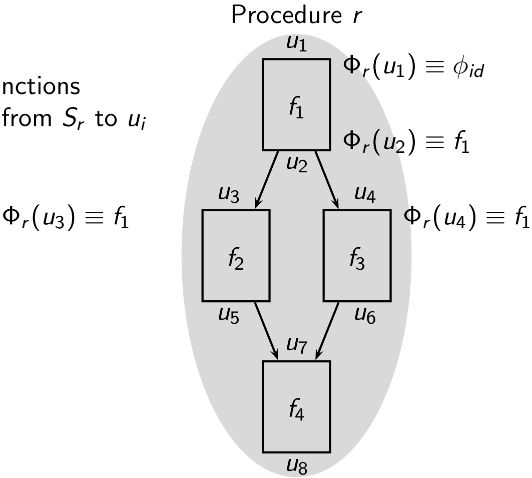
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Notation for Summary Flow Function

For simplicity forward flow is assumed

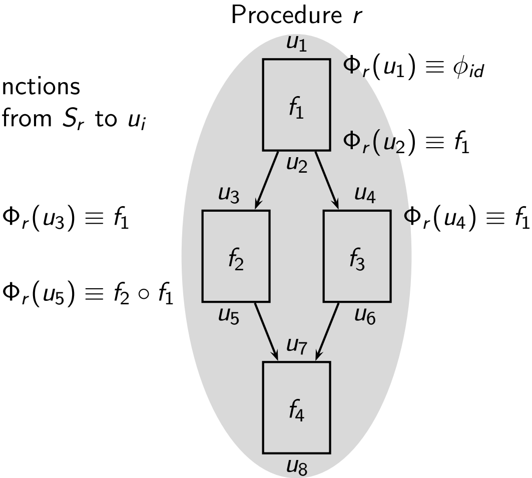
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Notation for Summary Flow Function

For simplicity forward flow is assumed

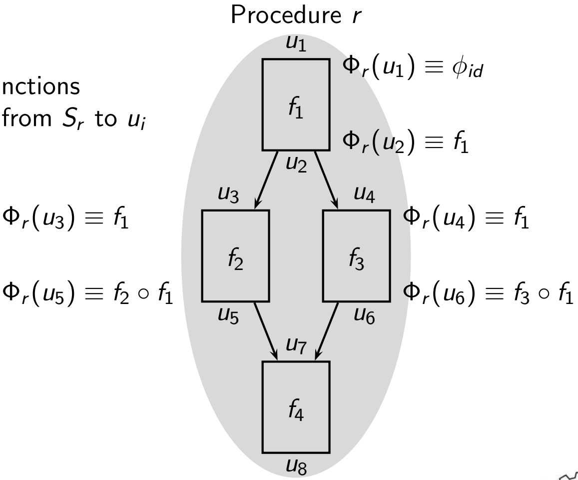
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



Notation for Summary Flow Function

For simplicity forward flow is assumed

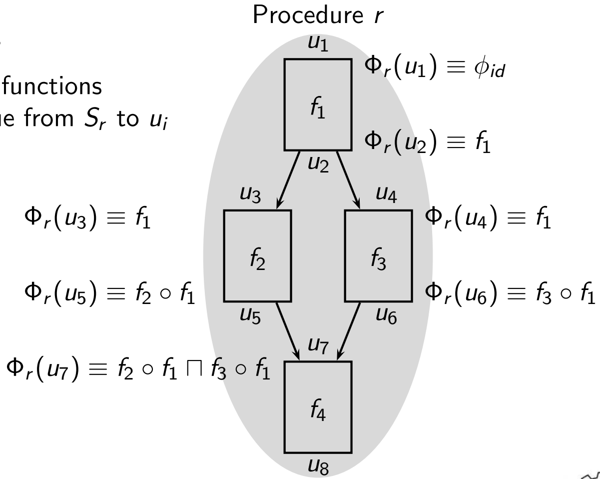
- u_i: Program points
- f_i: Node flow functions
- $\Phi_r(u_i)$: Summary flow functions mapping data flow value from S_r to u_i



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

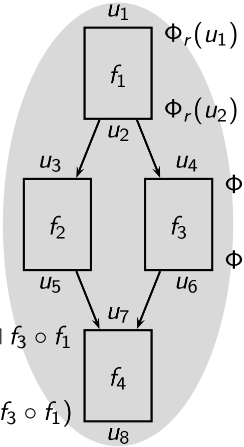


Notation for Summary Flow Function

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

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

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

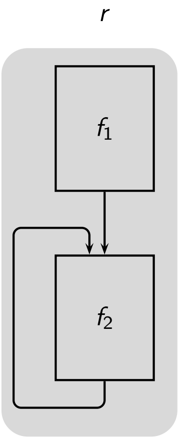
$$\Phi_r(O_n) = \begin{cases} \Phi_s(u) \circ \Phi_r(I_n) & \text{if } n \text{ calls procedure } s \text{ and } u \text{ is } O_{E_s} \\ f_n \circ \Phi_r(I_n) & \text{otherwise} \end{cases}$$

The summary flow function of a given procedure r

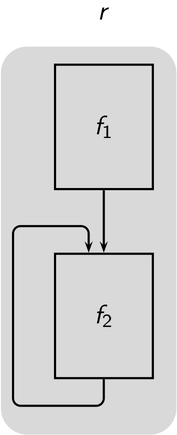
- is influenced by summary flow functions of the callees of r
- is not influenced by summary flow functions of the callers of r

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

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

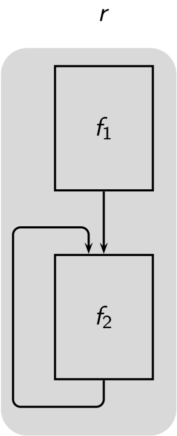
$\Phi_r(u_2) = f_1$

$\Phi_r(u_3) = f_1$

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

Constructing Summary Flow Functions Iteratively

Iteration #2



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

$\Phi_r(u_2) = f_1$

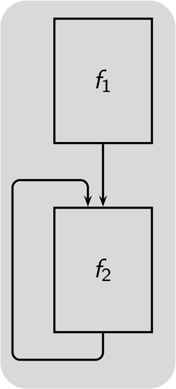
$\Phi_r(u_3) = f_1 \sqcap f_2 \circ f_1$

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

Constructing Summary Flow Functions Iteratively

r

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)

Lattice of data flow values	All possible flow functions	Lattice of flow functions
$\widehat{\top} = \emptyset \downarrow \widehat{\bot} = \{a\}$	(table below)	$\widehat{\phi}_{\top} \downarrow \widehat{\phi}_{id} \downarrow \widehat{\phi}_{\bot}$

Gen_n	Kill_n	$\widehat{f}_n$	$\widehat{f}_n(x),\ \forall x \in \{\widehat{\top}, \widehat{\bot}\}$
$\emptyset$	$\emptyset$	$\widehat{\phi}_{id}$	x
$\emptyset$	$\{a\}$	$\widehat{\phi}_{\top}$	$\widehat{\top}$
$\{a\}$	$\emptyset$	$\widehat{\phi}_{\bot}$	$\widehat{\bot}$
$\{a\}$	$\{a\}$		

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

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

Reducing Function Compositions in Bit Vector Frameworks

Kill_n denoted by K_n and Gen_n denoted by G_n

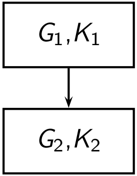
$$\begin{aligned} f_3(\text{x}) &= f_2(f_1(\text{x})) \\ &= f_2\big((\text{x} - K_1) \cup G_1\big) \end{aligned}$$

$$\boxed{G_1, K_1}$$

Reducing Function Compositions in Bit Vector Frameworks

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

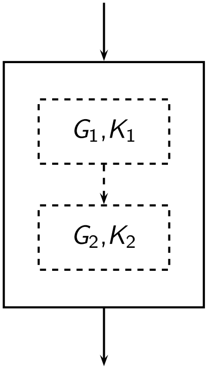
$$
\begin{aligned}
f_3(x) &= f_2(f_1(x)) \\
&= f_2\big((x - K_1) \cup G_1\big) \\
&= \Big(\big((x - K_1) \cup G_1\big) - K_2\Big) \ \cup \ G_2
\end{aligned}
$$



Reducing Function Compositions in Bit Vector Frameworks

Kill_n denoted by K_n and Gen_n denoted by G_n

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



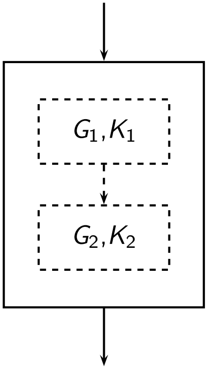
Reducing Function Compositions in Bit Vector Frameworks

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

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

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)

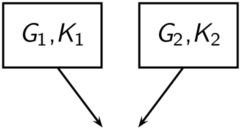
Kill$_n$ denoted by K_n and Gen$_n$ denoted by G_n

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

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

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 (1)

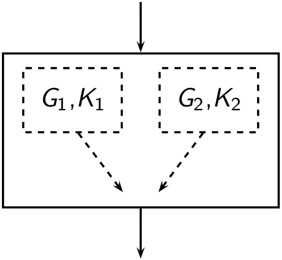
Kill_n denoted by K_n and Gen_n denoted by G_n

- 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)

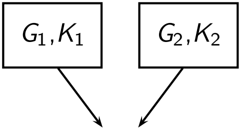
Kill_n denoted by K_n and Gen_n denoted by G_n

- 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)

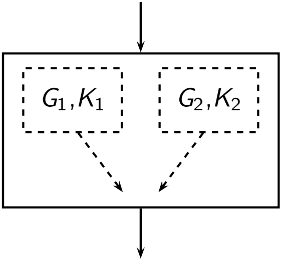
Kill_n denoted by K_n and Gen_n denoted by G_n

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

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

Hence,

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



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

Lattice of data flow values

$$\begin{tikzcd}
 & \top = \emptyset \arrow[dl] \arrow[dr] & \\
\{a\} \arrow[dr] & & \{b\} \arrow[dl] \\
 & \bot = \{a, b\} &
\end{tikzcd}$$

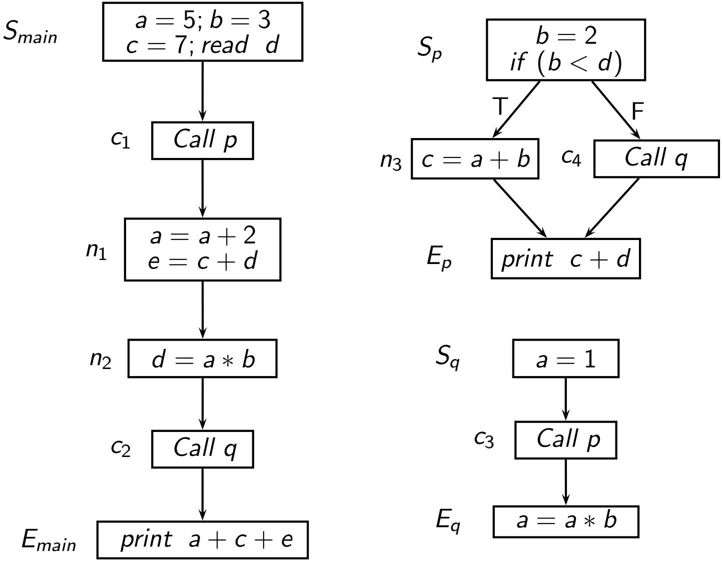
All possible flow functions

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

Lattice of flow functions

$$\begin{tikzcd}
 & & \phi_{\top\top} \arrow[dl] \arrow[dr] & & \\
 & \phi_{\top I} \arrow[dl] \arrow[dr] & & \phi_{I\top} \arrow[dl] \arrow[dr] & \\
\phi_{\top\bot} \arrow[dr] & & \phi_{II} \arrow[dl] \arrow[dr] & & \phi_{\bot\top} \arrow[dl] \\
 & \phi_{I\bot} \arrow[dr] & & \phi_{\bot I} \arrow[dl] & \\
 & & \phi_{\bot\bot} & &
\end{tikzcd}$$

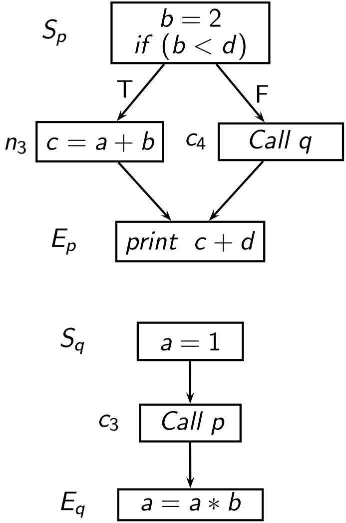
An Example of Interprocedural Liveness Analysis



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}	$\{c, d\}$	$\emptyset$		
	$\Phi_p(n_3)$	$f_{n_3} \circ \Phi_p(E_p)$	$\{a, b, d\}$	$\{c\}$		
	$\Phi_p(c_4)$	$f_q \circ \Phi_p(E_p) = \phi_\top$	$\emptyset$	$\{a, b, c, d, e\}$	$\{d\}$	$\{a, b, c\}$
	$\Phi_p(S_p)$	$f_{S_p} \circ (\Phi_p(n_3) \sqcap \Phi_p(c_4))$	$\{a, d\}$	$\{b, c\}$		
	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	
$\Phi_p(E_p)$	$BI_p \cup \{c, d\}$
$\Phi_p(n_3)$	$\left(BI_p - \{c\}\right) \cup \{a, b, d\}$
$\Phi_p(c_4)$	$\left(BI_p - \{a, b, c\}\right) \cup \{d\}$
$\Phi_p(S_p)$	$\left(BI_p - \{b, c\}\right) \cup \{a, d\}$
$\Phi_q(E_q)$	$\left(BI_q - \{a\}\right) \cup \{a, b\}$
$\Phi_q(c_3)$	$\left(BI_q - \{a, b, c\}\right) \cup \{a, d\}$
$\Phi_q(S_q)$	$\left(BI_q - \{a, b, c\}\right) \cup \{d\}$

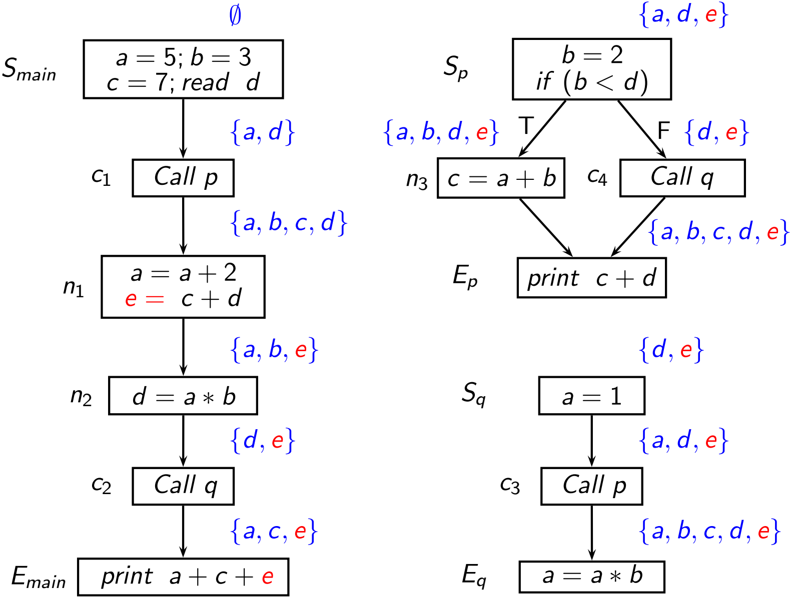
Result of Interprocedural Liveness Analysis

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}	$\Phi_m(c_2)$	$\left(BI_m - \{a, b, c\}\right) \cup \{d, e\}$	$\{d, e\}$
In_{n_2}	$\Phi_m(n_2)$	$\left(BI_m - \{a, b, c, d\}\right) \cup \{a, b, e\}$	$\{a, b, e\}$
In_{n_1}	$\Phi_m(n_1)$	$\left(BI_m - \{a, b, c, d, e\}\right) \cup \{a, b, c, d\}$	$\{a, b, c, d\}$
In_{c_1}	$\Phi_m(c_1)$	$\left(BI_m - \{a, b, c, d, e\}\right) \cup \{a, d\}$	$\{a, d\}$
In_{S_m}	$\Phi_m(S_m)$	$BI_m - \{a, b, c, d, e\}$	$\emptyset$

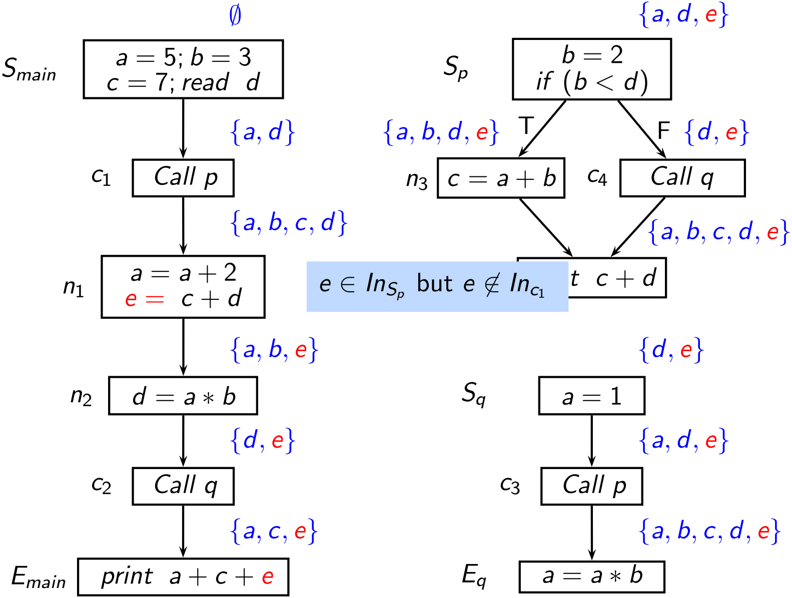
Result of Interprocedural Liveness Analysis

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}	$\Phi_p(n_3)$	$\left(BI_p - \{c\}\right) \cup \{a, b, d\}$	$\{a, b, d, e\}$
In_{c_4}	$\Phi_p(c_4)$	$\left(BI_p - \{a, b, c\}\right) \cup \{d\}$	$\{d, e\}$
In_{S_p}	$\Phi_p(S_p)$	$\left(BI_p - \{b, c\}\right) \cup \{a, d\}$	$\{a, d, e\}$
Procedure q, $BI = \{a, b, c, d, e\}$			
In_{E_q}	$\Phi_q(E_q)$	$\left(BI_q - \{a\}\right) \cup \{a, b\}$	$\{a, b, c, d, e\}$
In_{c_3}	$\Phi_q(c_3)$	$\left(BI_q - \{a, b, c\}\right) \cup \{a, d\}$	$\{a, d, e\}$
In_{S_q}	$\Phi_q(S_q)$	$\left(BI_q - \{a, b, c\}\right) \cup \{d\}$	$\{d, e\}$

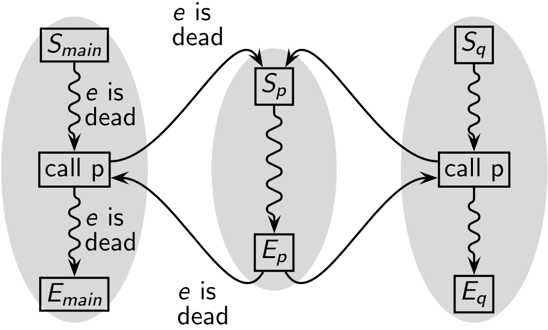
Context Sensitivity of Interprocedural Liveness Analysis



Context Sensitivity of Interprocedural Liveness Analysis

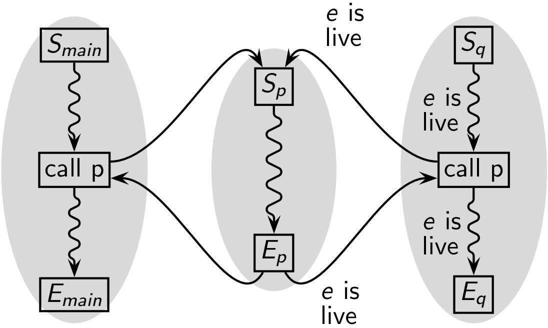


Explaining Context Sensitivity



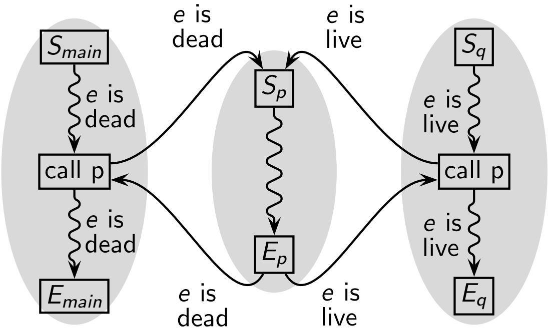
- Flow function of procedure p is identity with respect to variable e

Explaining Context Sensitivity



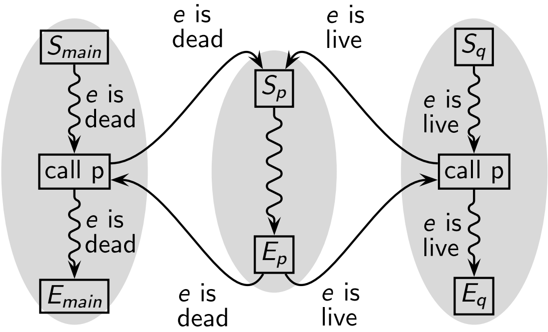
- Flow function of procedure p is identity with respect to variable e

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



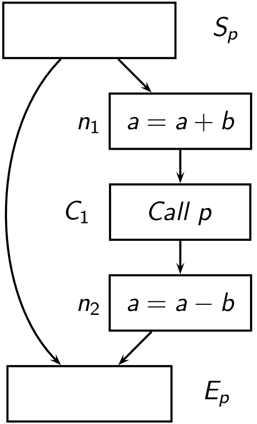
- Flow function of procedure p is identity with respect to variable e
- Is e live in the body of procedure p?
  - During the analysis: Depends on the calling context
  - After the analysis: Yes (static approximation across all executions)
- Distinction between caller's effect on callee and callee's effect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

Tutorial Problem #1

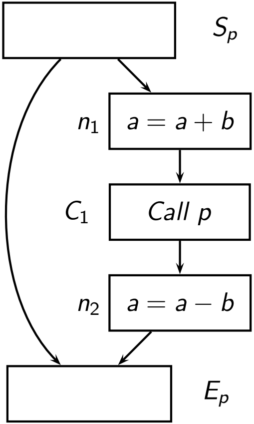
Perform interprocedural live variables analysis for the following program

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

Tutorial Problem #2: Summary Flow Function for Constant Propagation

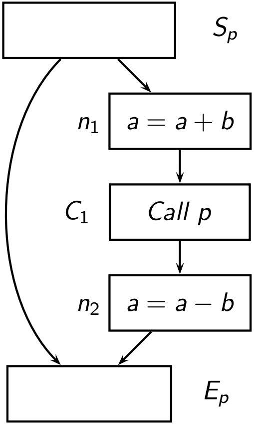


Tutorial Problem #2: Summary Flow Function for Constant Propagation



	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

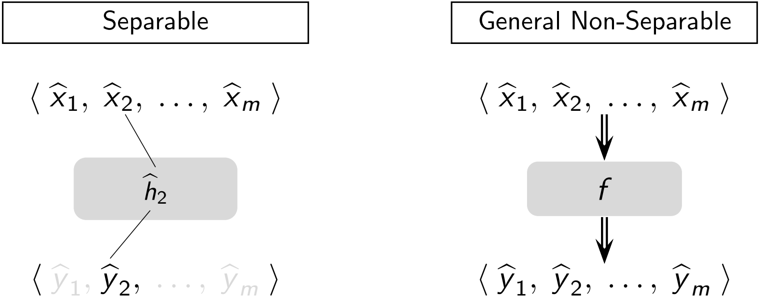
Overall Flow Function and Component Function

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

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 $\widehat{x}_i$



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

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

$$\Downarrow$$

$$f$$

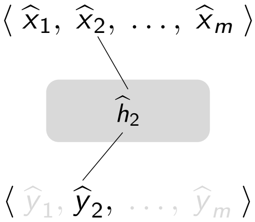
$$\Downarrow$$

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

Overall Flow Function and Component Function

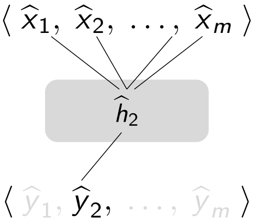
- Overall flow function $f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$
- Component function: $\widehat{h}_i$ which computes the value of $\widehat{x}_i$

Separable



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

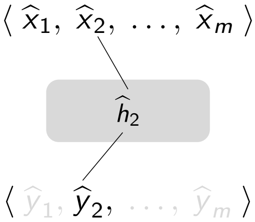
General Non-Separable



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 $\widehat{x}_i$

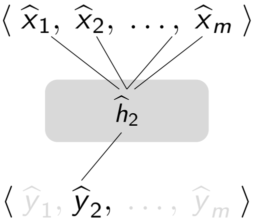
Separable



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

Example: All bit vector frameworks

General Non-Separable



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

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

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

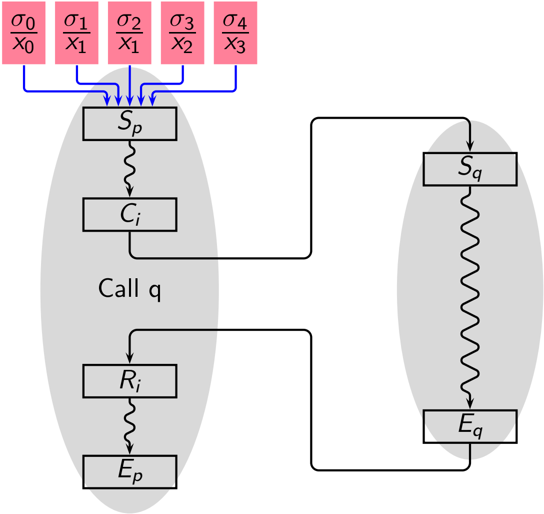
- We can reduce the amount of effort by using

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

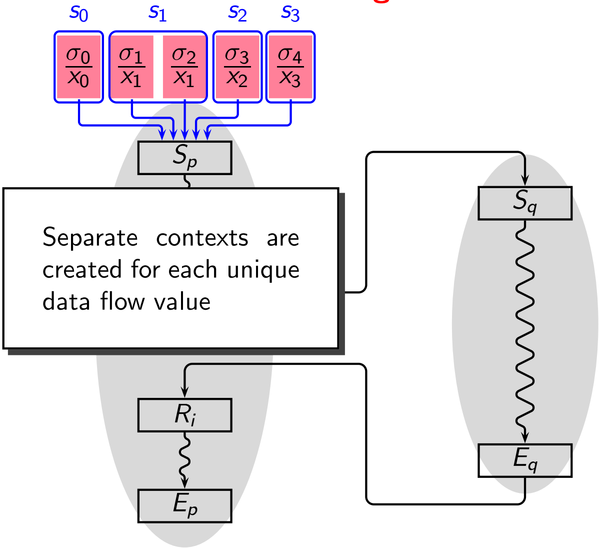
Interprocedural Data Flow Analysis Using Value Contexts

- 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 $\mapsto$ 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

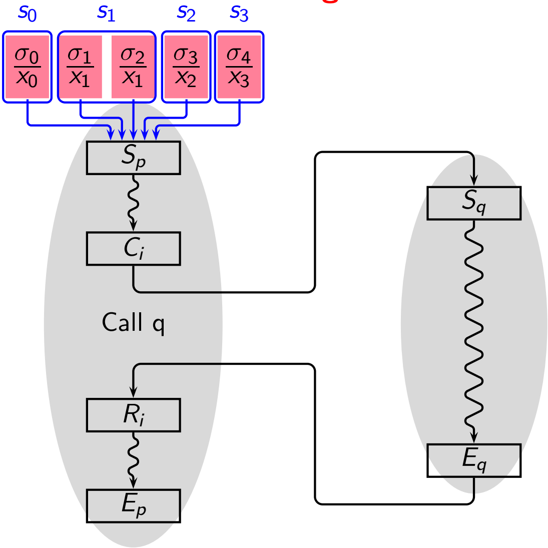


Understanding Value Contexts

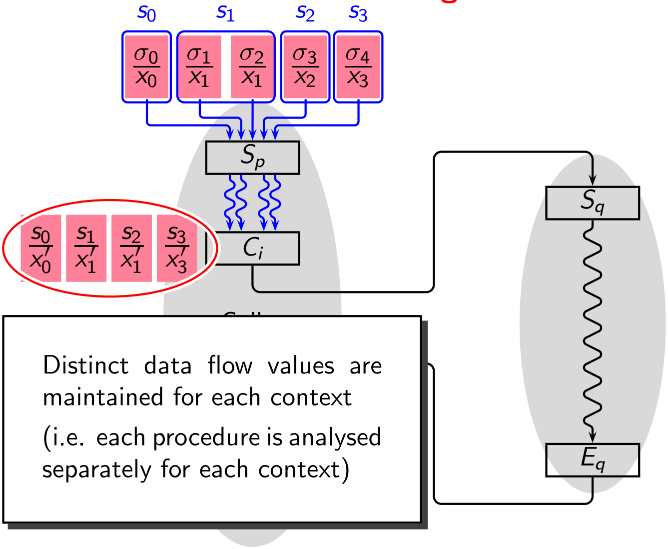


Separate contexts are created for each unique data flow value

Understanding Value Contexts



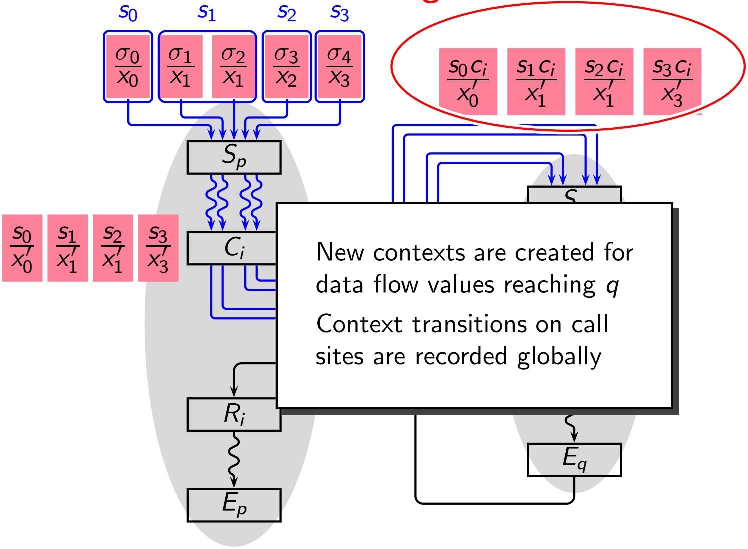
Understanding Value Contexts



Distinct data flow values are maintained for each context

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

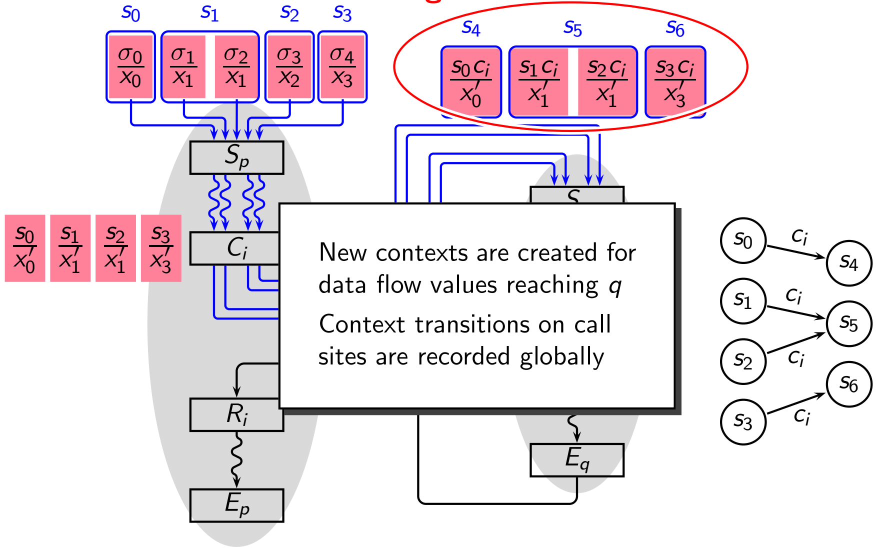
Understanding Value Contexts



New contexts are created for data flow values reaching q

Context transitions on call sites are recorded globally

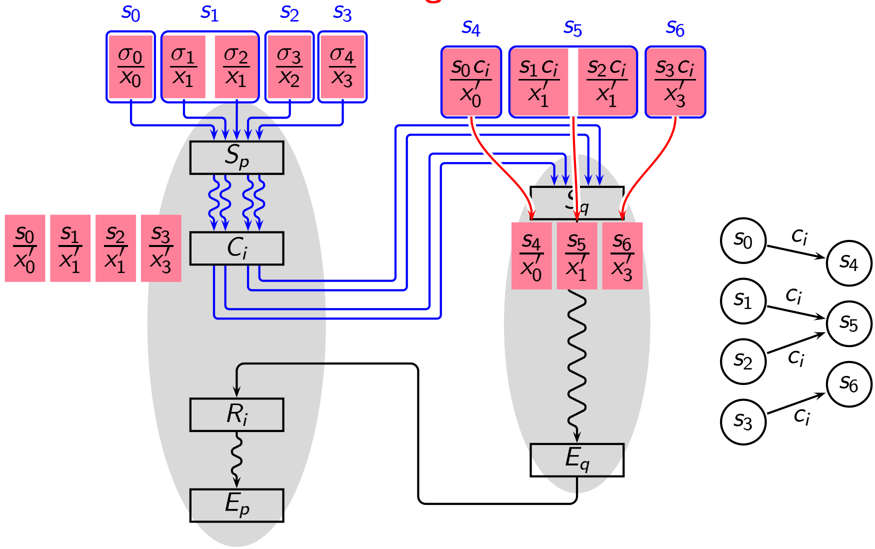
Understanding Value Contexts



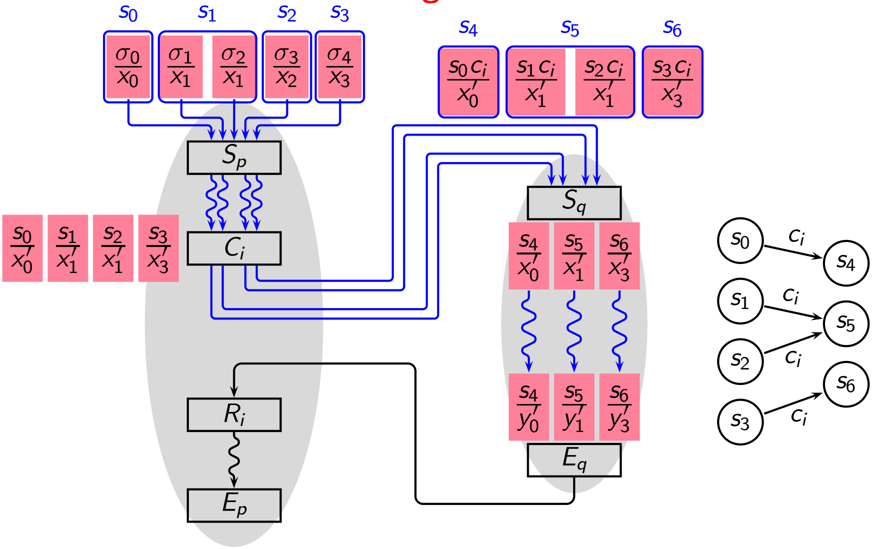
New contexts are created for data flow values reaching q

Context transitions on call sites are recorded globally

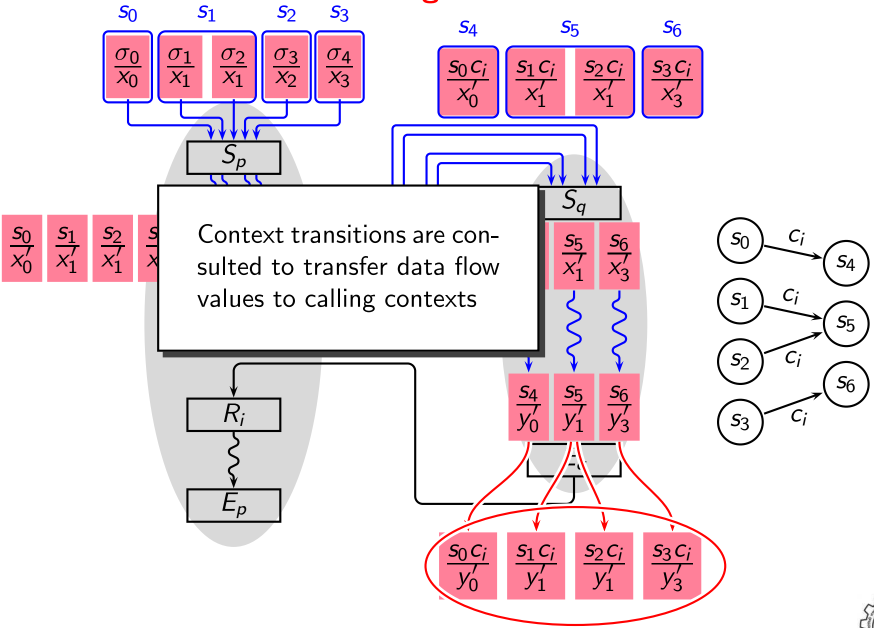
Understanding Value Contexts



Understanding Value Contexts

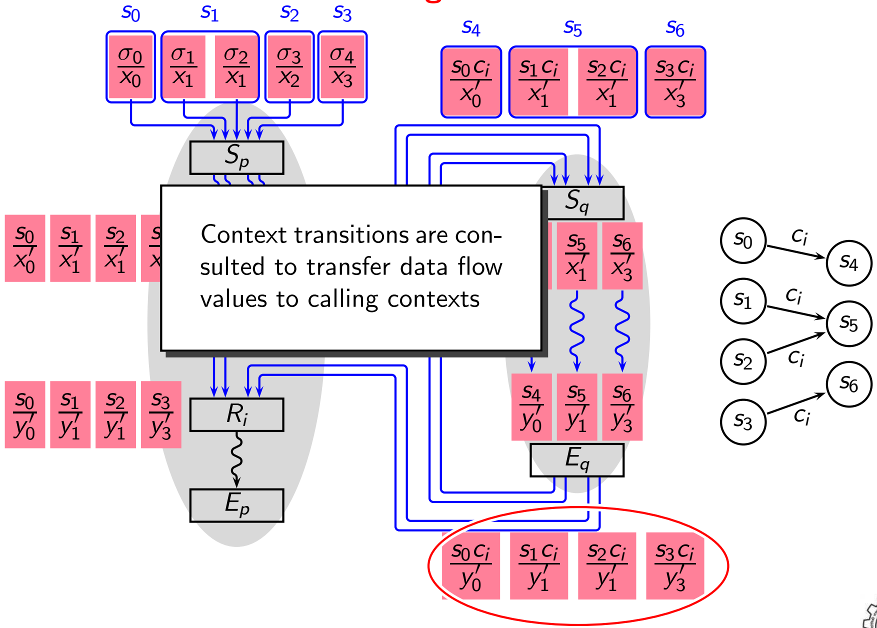


Understanding Value Contexts

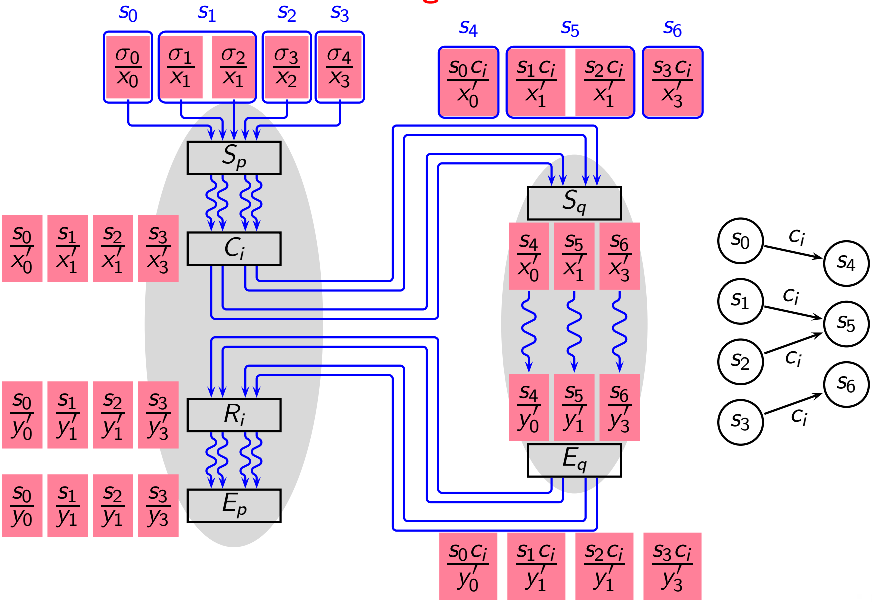


Context transitions are consulted to transfer data flow values to calling contexts

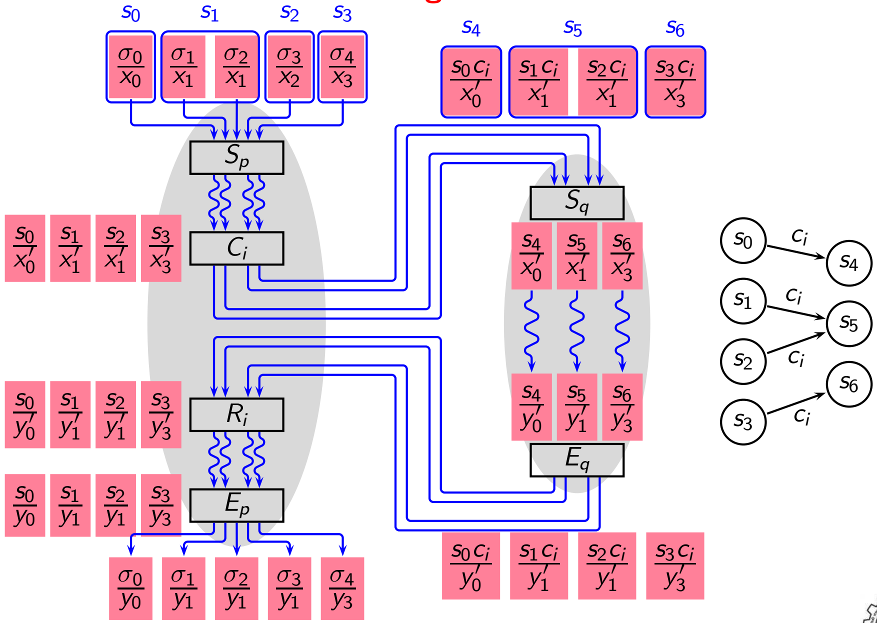
Understanding Value Contexts



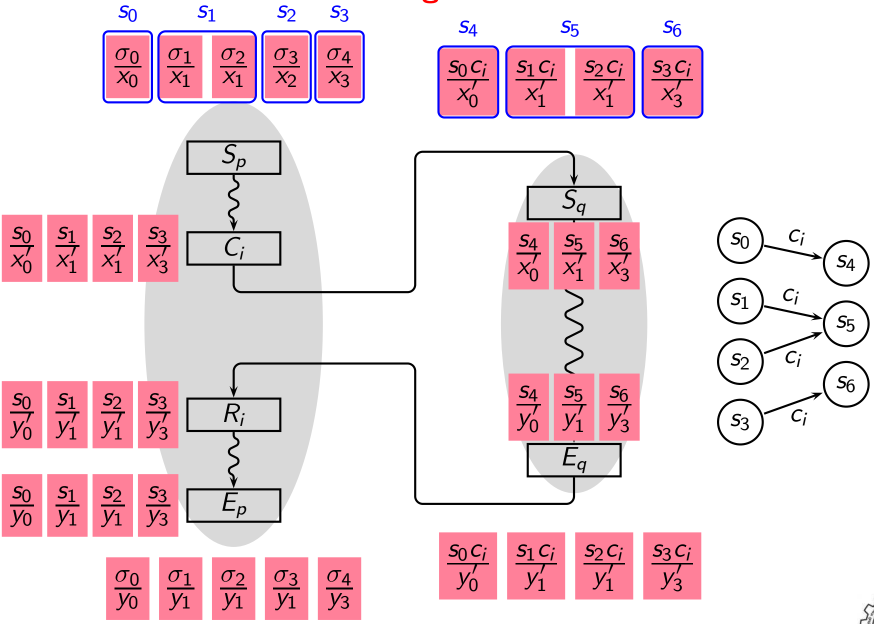
Understanding Value Contexts



Understanding Value Contexts



Understanding Value Contexts



Defining Value Contexts

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

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

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
$X \vert m$	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
$X \xrightarrow{C_i} Y$	Transition from context X to context Y at call site C_i

Interprocedural Data Flow Analysis Using Value Contexts

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

Interprocedural Data Flow Analysis Using Value Contexts

- Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
  - For all other nodes

Interprocedural Data Flow Analysis Using Value Contexts

- 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)
    - If it does not exist
  - If $n = E_p$
  - For all other nodes

Interprocedural Data Flow Analysis Using Value Contexts

- 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)
      - record the transition $X \xrightarrow{C_i} Y$
      - $Out_{C_i}[X] = Out_{C_i}[X] \sqcap exitValue(Y)$
      - if there is a change, add $X|m$, $\forall m \in succ(C_i)$ to WL
    - If it does not exist
  - If $n = E_p$
  - For all other nodes

Interprocedural Data Flow Analysis Using Value Contexts

- 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)
      - 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
    - If it does not exist
      - create a new context $Y = \langle p,v \rangle$
      - initialize $exitValue(Y) = \top$
      - record the transition $X \overset{C_i}{\to} Y$
      - add entries $Y|m$ for all nodes m of procedure p to WL
  - If $n = E_p$
  - For all other nodes

Interprocedural Data Flow Analysis Using Value Contexts

- Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
    - Set $exitValue(X) = v$
  - For all other nodes

Interprocedural Data Flow Analysis Using Value Contexts

- Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
    - Set $exitValue(X) = v$
    - Find out all transitions $Z \xrightarrow{C_j} X$
      - Set $Out_{C_j}[Z] = v$
      - 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

- Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
  - For all other nodes
    - Set $Out_n[X] = f_n(v)$

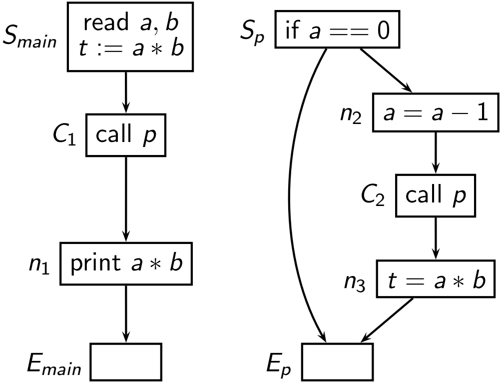
Interprocedural Data Flow Analysis Using Value Contexts

- Select $X|n$ from WL. Compute In_n. Let $X.v$ be in In_n
  - If $n = C_i$ calling procedure p
  - If $n = E_p$
  - 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

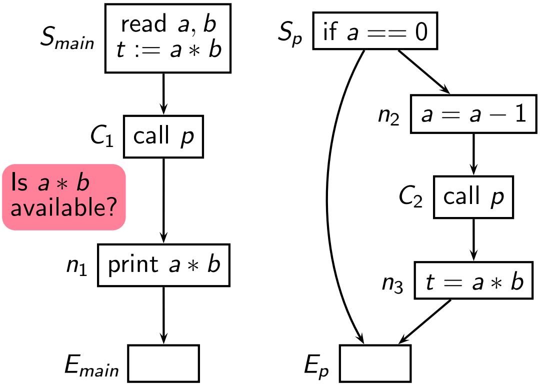
Interprocedural Data Flow Analysis Using Value Contexts

- 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)
      - 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
    - If it does not exist
      - create a new context $Y = \langle p,v\rangle$
      - initialize $exitValue(Y) = \top$
      - record the transition $X \overset{C_i}{\to} Y$
      - add entries $Y|m$ for all nodes m of procedure p to WL
  - If $n = E_p$
    - Set $exitValue(X) = v$
    - Find out all transitions $Z \overset{C_j}{\to} X$
      - Set $Out_{C_j}[Z] = v$
      - 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

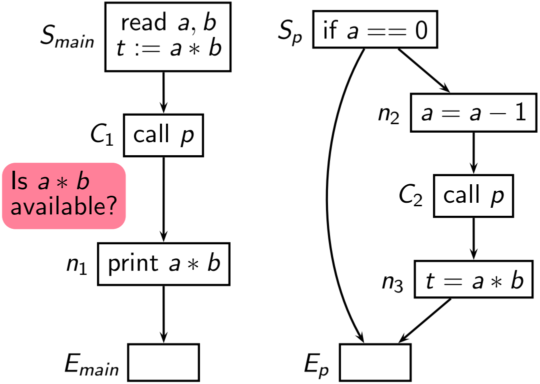
Available Expressions Analysis Using Value Contexts



Available Expressions Analysis Using Value Contexts

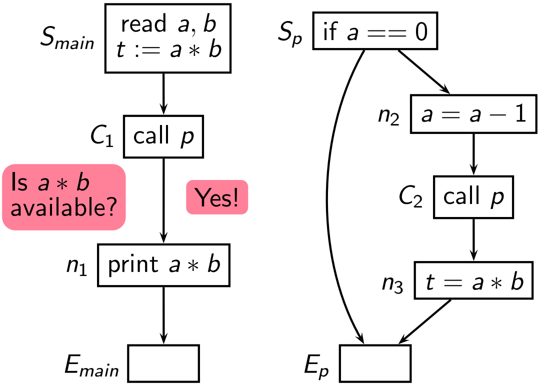


Available Expressions Analysis Using Value Contexts



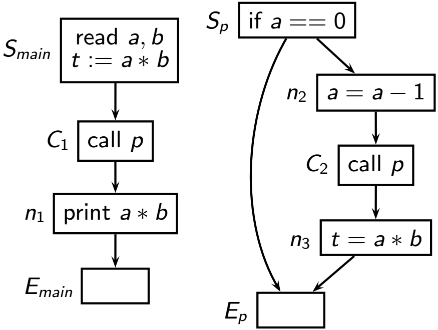

```
int a, b, t;
void p()
{  if (a == 0)
   {  a = a-1;
      p();
      t = a*b;
   }
}
```

Available Expressions Analysis Using Value Contexts

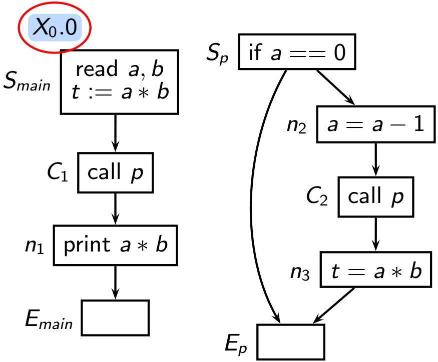



```
int a, b, t;
void p()
{  if (a == 0)
   {  a = a-1;
      p();
      t = a*b;
   }
}
```

Available Expressions Analysis Using Value Contexts



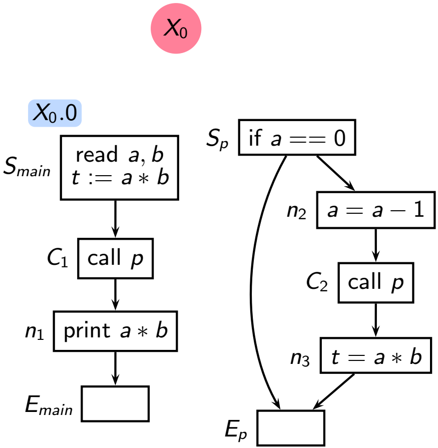
Available Expressions Analysis Using Value Contexts



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 $\top = 1$

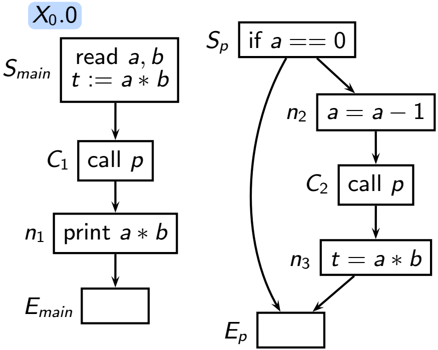
Initialize the work list with all nodes in procedure main for X_0

Available Expressions Analysis Using Value Contexts

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

X_0

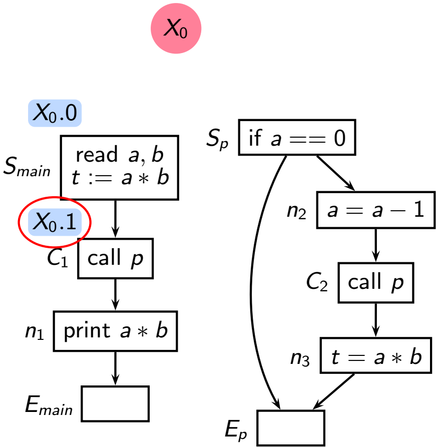
Context	*exitValue*
$X_0 = \langle \text{main,0} \rangle$	1



Compute the data flow values for S_m for context X_0

Available Expressions Analysis Using Value Contexts

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

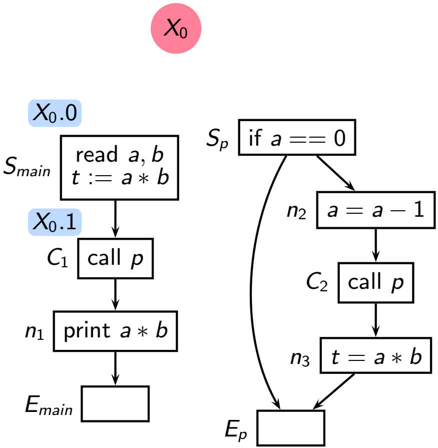


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1

Compute the data flow values for S_m for context X_0

Available Expressions Analysis Using Value Contexts

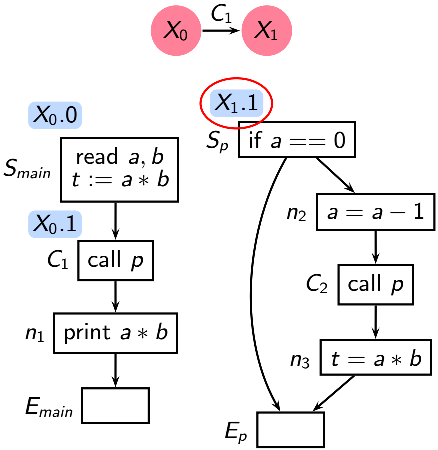
$WL = [X_0|C_1, X_0|n_1, X_0|E_m]$



Context	*exitValue*
$X_0 = \langle$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

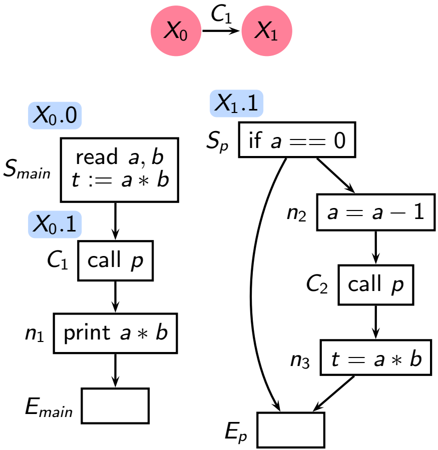
Record the transition to X_1

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

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

Available Expressions Analysis Using Value Contexts

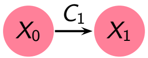
$WL = [X_1|S_p, X_1|n_2, X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



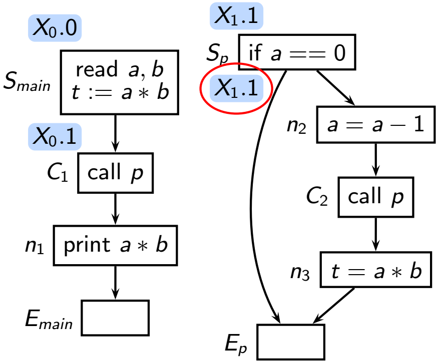
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

Available Expressions Analysis Using Value Contexts

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

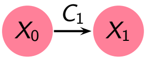


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

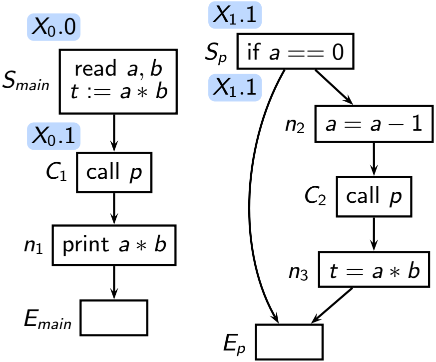


Available Expressions Analysis Using Value Contexts

$WL = [\color{red}{X_1|n_2}, X_1|C_2, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$

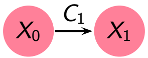


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

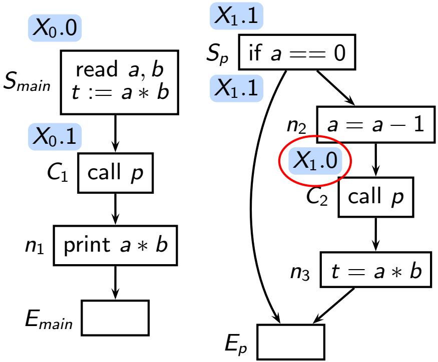


Available Expressions Analysis Using Value Contexts

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

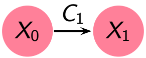


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

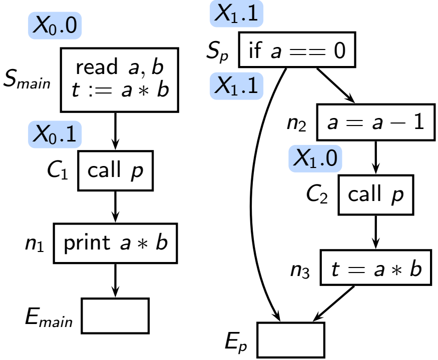


Available Expressions Analysis Using Value Contexts

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

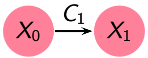


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1

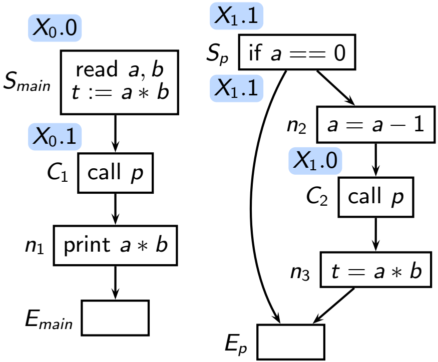


Available Expressions Analysis Using Value Contexts

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

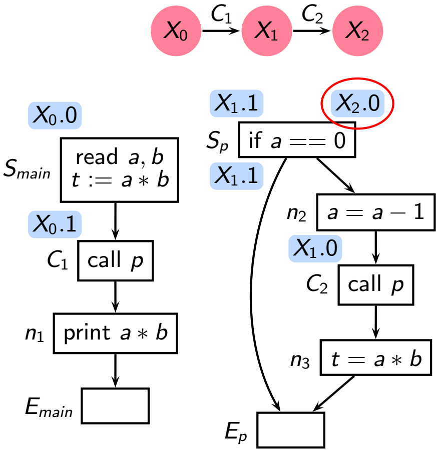


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1



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

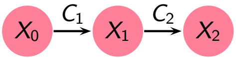
Record the transition to X_2

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

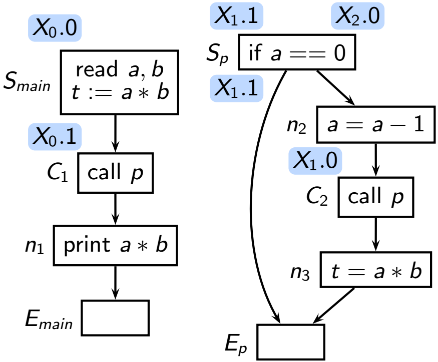
Add all nodes of procedure p to the work list for X_2

Available Expressions Analysis Using Value Contexts

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

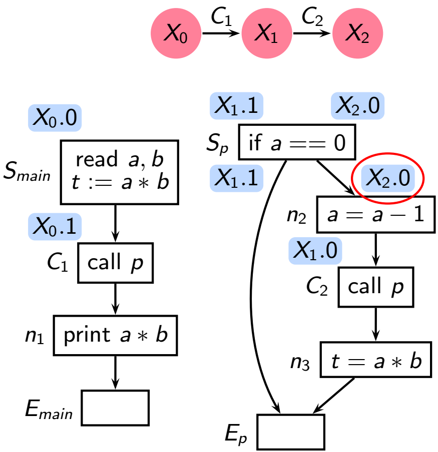


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1



Available Expressions Analysis Using Value Contexts

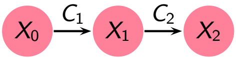
$WL = [X_2|S_p, X_2|n_2, X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



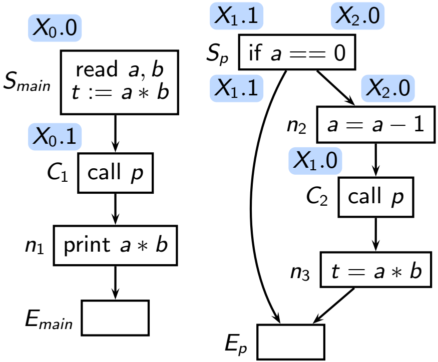
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

Available Expressions Analysis Using Value Contexts

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

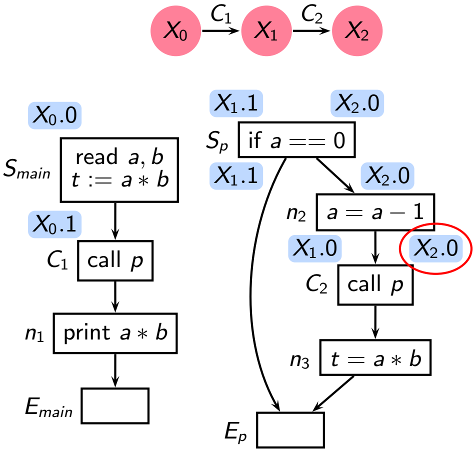


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1



Available Expressions Analysis Using Value Contexts

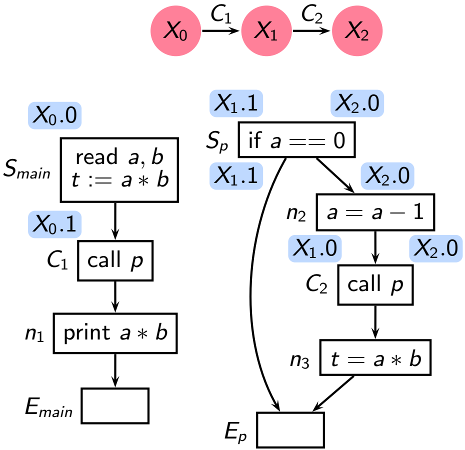
$WL = [X_2|n_2, X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



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

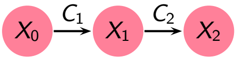
$WL = [X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m]$



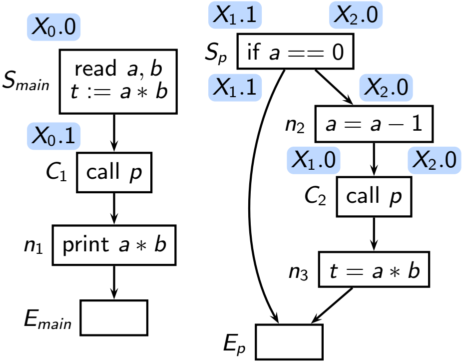
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

Available Expressions Analysis Using Value Contexts

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



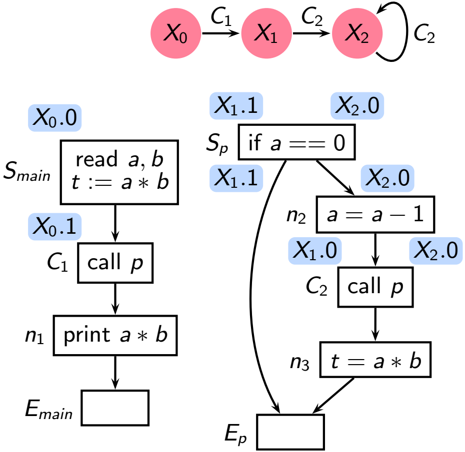
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1



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

Available Expressions Analysis Using Value Contexts

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



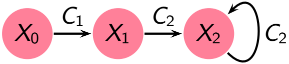
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

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

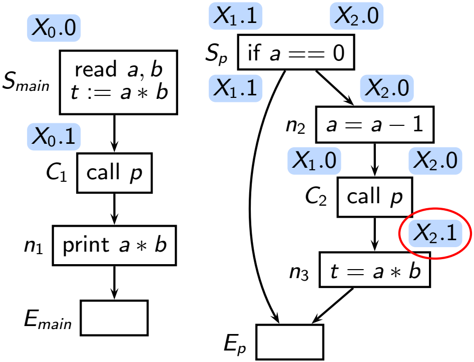
Record the transition from context X_2 to itself

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



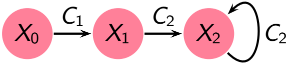
p has context X_2 with value 0 so no need to create a new context

Record the transition from context X_2 to itself

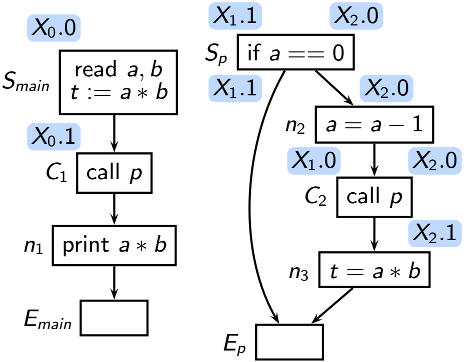
Use the *exitValue*(X_2) to compute $Out_{C_2}[X_2]$

Available Expressions Analysis Using Value Contexts

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

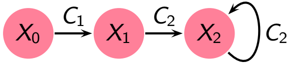


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

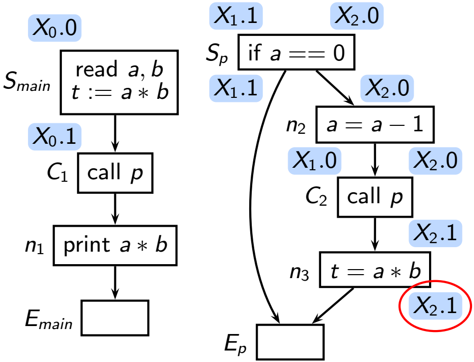


Available Expressions Analysis Using Value Contexts

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

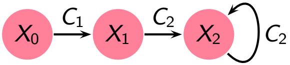


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

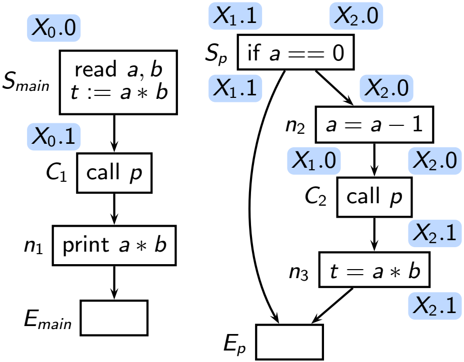


Available Expressions Analysis Using Value Contexts

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

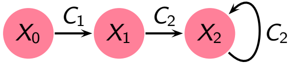


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1

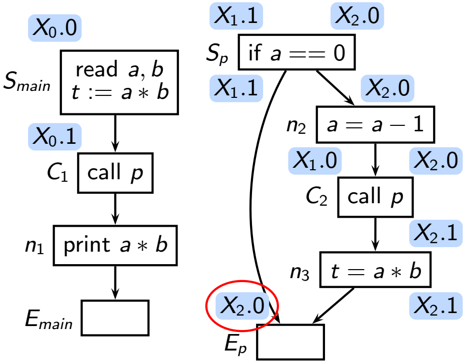


Available Expressions Analysis Using Value Contexts

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



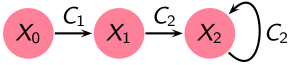
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	1



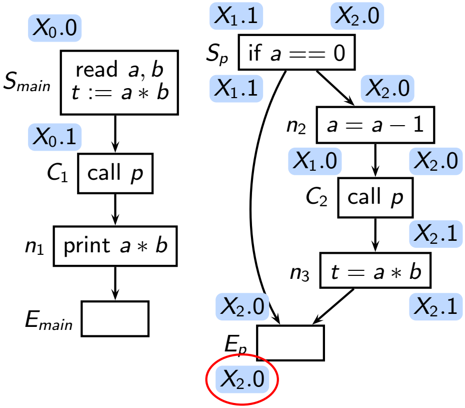
At E_p the values from S_p and n_3 are merged for context X_2

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

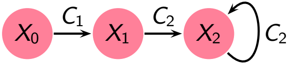


At E_p the values from S_p and n_3 are merged for context X_2

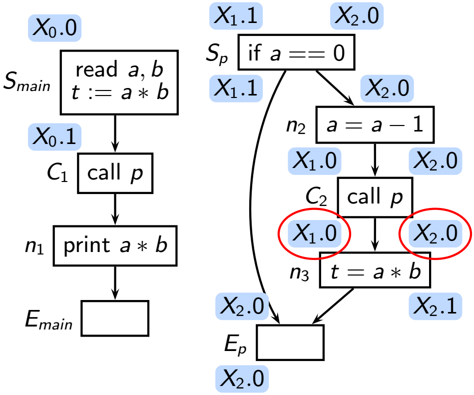
$exitValue(X_2)$ is set to 0

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



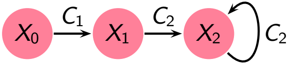
At E_p the values from S_p and n_3 are merged for context X_2

$exitValue(X_2)$ is set to 0

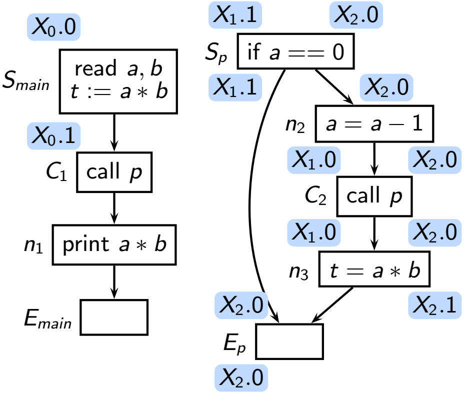
Since X_2 has transitions $X_1 \xrightarrow{C_2} X_2$ and $X_2 \xrightarrow{C_2} X_2$, $Out_{C_2}[X_1]$ and $Out_{C_2}[X_2]$ become 0

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

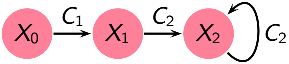
$exitValue(X_2)$ is set to 0

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

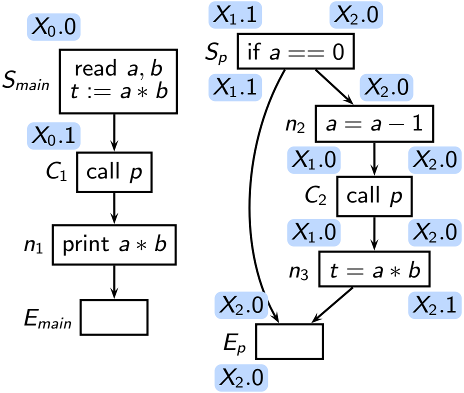
Since $Out_{C_2}[X_2]$ changes, $X_2|n_3$ is added to the work list

Available Expressions Analysis Using Value Contexts

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



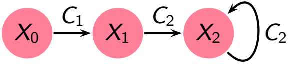
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



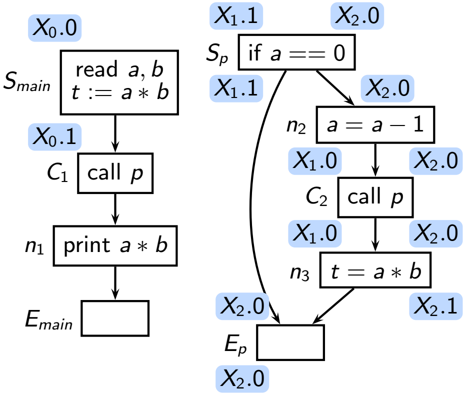
There is no change in $Out_{n_3}[X_2]$

Available Expressions Analysis Using Value Contexts

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



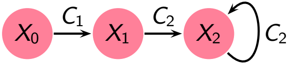
Context	*exitValue*
$X_0 = \langle \text{main},0 \rangle$	1
$X_1 = \langle \text{p},1 \rangle$	1
$X_2 = \langle \text{p},0 \rangle$	0



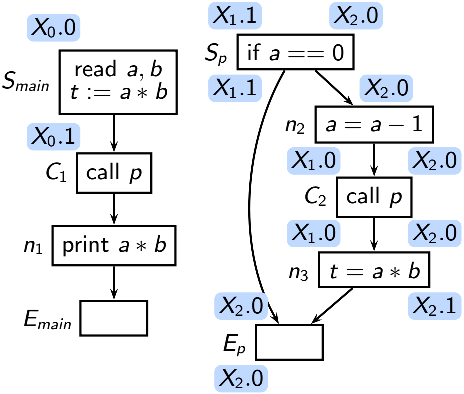
There is no change in $Out_{n_3}[X_2]$

Available Expressions Analysis Using Value Contexts

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



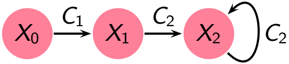
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



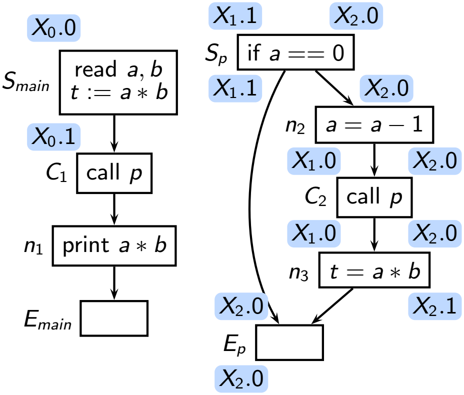
There is no change in $Out_{n_3}[X_1]$ also

Available Expressions Analysis Using Value Contexts

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



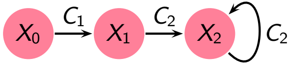
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



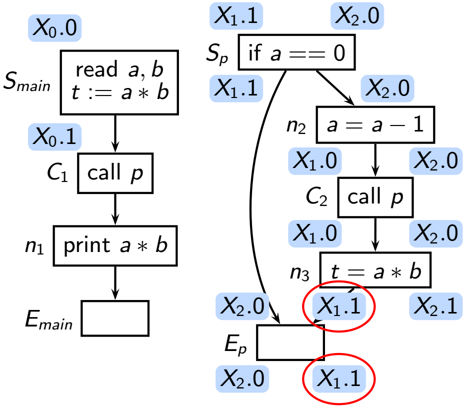
There is no change in $Out_{n_3}[X_1]$ also

Available Expressions Analysis Using Value Contexts

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



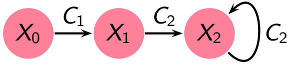
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



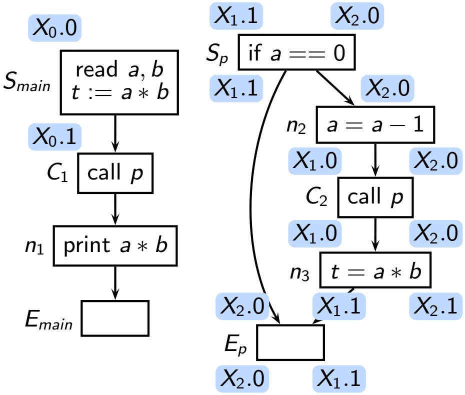
At E_p the values from S_p and n_3 are merged for context X_1

Available Expressions Analysis Using Value Contexts

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



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

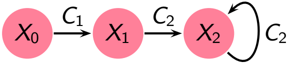


At E_p the values from S_p and n_3 are merged for context X_1

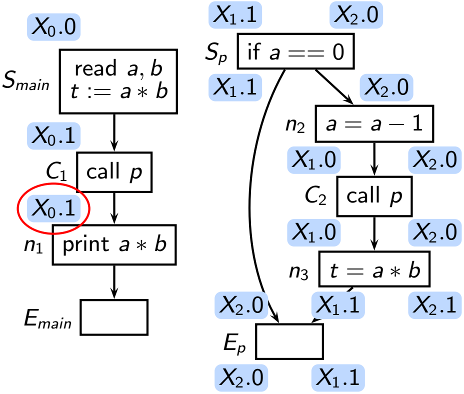
exitValue(X_1) remains 1

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



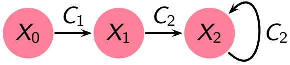
At E_p the values from S_p and n_3 are merged for context X_1

$exitValue(X_1)$ remains 1

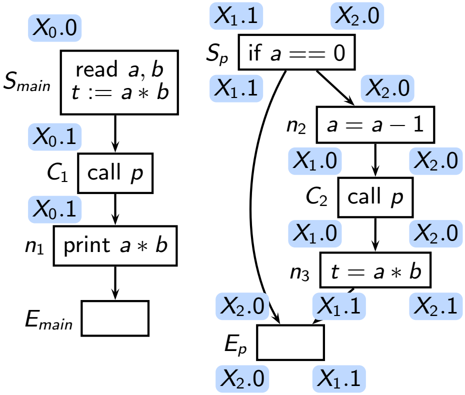
Since X_1 has transition $X_0 \xrightarrow{C_1} X_1$, $Out_{C_1}[X_0]$ becomes 1

Available Expressions Analysis Using Value Contexts

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

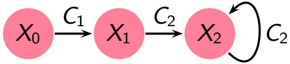


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

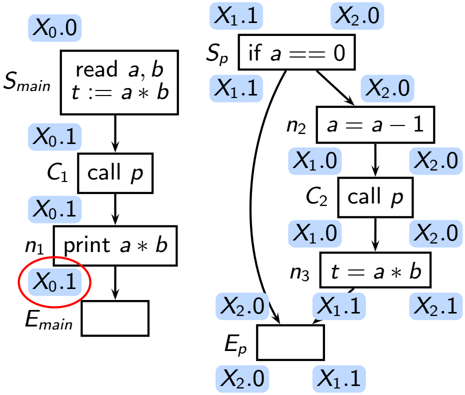


Available Expressions Analysis Using Value Contexts

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

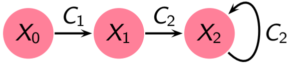


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

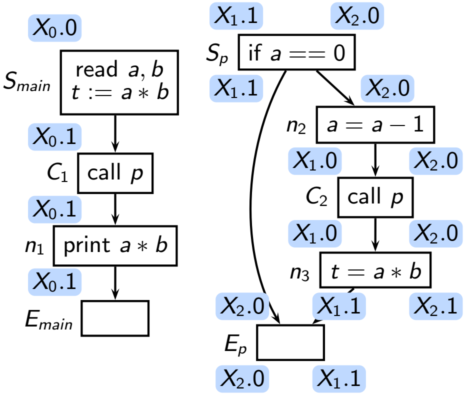


Available Expressions Analysis Using Value Contexts

$WL = [X_0 | E_m]$

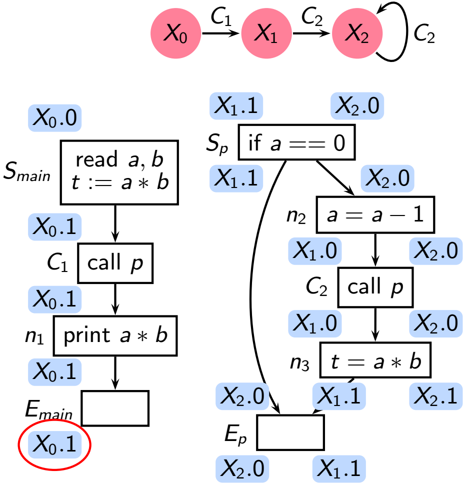


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0



Available Expressions Analysis Using Value Contexts

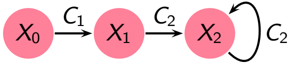
$WL = [X_0 | E_m]$



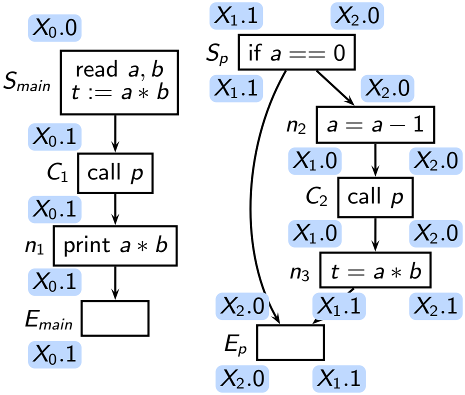
Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

Available Expressions Analysis Using Value Contexts

$WL = [X_0 | E_m]$

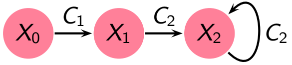


Context	*exitValue*
$X_0 = \langle$main,0$\rangle$	1
$X_1 = \langle$p,1$\rangle$	1
$X_2 = \langle$p,0$\rangle$	0

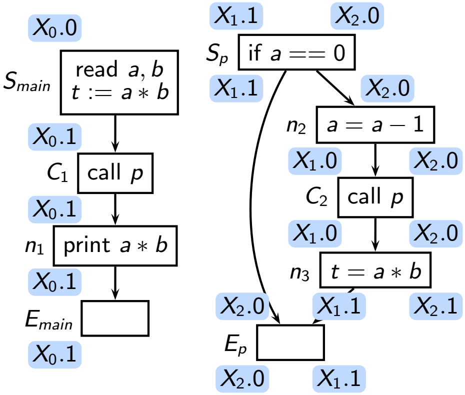


Available Expressions Analysis Using Value Contexts

$WL = [\ ]$



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)

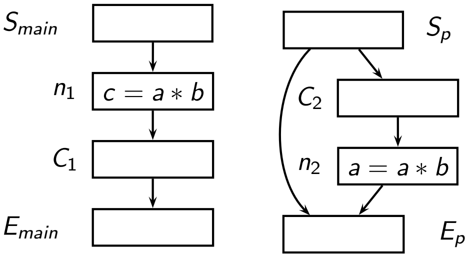
S. No.	Work List	Sel. node	Data flow value	New context	New trans.	exit value	Addition to the work list
1				$X_0 = \langle m,0 \rangle$		$X_0.1$	$X_0 \vert S_m, X_0 \vert C_1, X_0 \vert n_1, X_0 \vert E_m$
2	$X_0 \vert S_m, X_0 \vert C_1, X_0 \vert n_1, X_0 \vert E_m$	S_m	$Out_{S_m}[X_0] = 1$				
3	$X_0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

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

<pre>main()
{
   p();
}</pre>
	<pre>p()
	{ while (...)
	   { printf ("%d\n",a);
	      p();
	   }
	}</pre>

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;
}
```

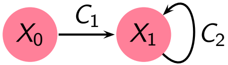
```
q()
{
  /*{d,e}*/
  a = 1;
  /*{a,d,e}*/
  p();
  /*{a,b,c,d,e}*/
  a = a*b;
}
```

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	3973×10^3	8	7	2.37
bit_gray	53	5	11	7	100000+	31374	2705×10^3	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	432×10^3	14	4	0.43
fourinarow	676	17	45	5	510	158	397.76	46	7	1.86
sim	1146	13	45	8	100000+	33546	1427×10^3	211	105	234.16
181_mcf	1299	17	24	6	32789	32767	484×10^3	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.

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