Interprocedural Data Flow Analysis

Uday Khedker

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

October 2017
Part 1

*About These Slides*
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:


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


*These slides are being made available under GNU FDL v1.2 or later purely for academic or research use.*
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

\[ S_{main} \)
\[ a + b \]
\[ Call \ p \]
\[ E_{main} \]

\[ S_p \]
\[ Call \ q \]
\[ E_p \]

\[ n_1 \]
\[ d = a + b \]
\[ Call \ p \]
\[ n_3 \]
\[ n_4 \]

\[ n_2 \]
\[ a = 1 \]

\[ S_q \]
\[ E_q \]
Program Representation for Interprocedural Data Flow Analysis: Supergraph

\[ S_{main} \]
\[ a + b \]
\[ C_1 \] Call p

\[ S_p \]
\[ C_2 \] Call q

\[ R_1 \]
\[ R_2 \]
\[ E_{main} \]

\[ n_1 \]
\[ d = a + b \]
\[ C_3 \] Call p

\[ n_2 \]
\[ a = 1 \]
\[ C_4 \] Call p

\[ R_3 \]
\[ n_3 \]
\[ E_q \]

\[ R_4 \]
\[ n_4 \]
Program Representation for Interprocedural Data Flow Analysis: Supergraph
Program Representation for Interprocedural Data Flow Analysis: Supergraph

\[ S_{main} \xrightarrow{a + b} C_1 \xrightarrow{\text{Call } p} S_p \]

\[ \xrightarrow{C_1} R_1 \rightarrow E_{main} \]

\[ S_p \xrightarrow{S_p} C_2 \xrightarrow{\text{Call } q} \]

\[ \xrightarrow{C_2} R_2 \rightarrow E_p \]

\[ n_1 \rightarrow d = a + b \]

\[ \xrightarrow{n_1 C_3} \text{Call } p \]

\[ \xrightarrow{n_1} n_3 \rightarrow d = a + b \]

\[ \xrightarrow{n_3} C_4 \xrightarrow{\text{Call } p} \]

\[ \xrightarrow{n_1} a = 1 \]

\[ \xrightarrow{n_1} n_2 \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]

\[ \xrightarrow{n_2} \]
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

\[
\begin{align*}
S_p & \rightarrow a \times b \rightarrow \text{Call } q \\
E_p & \rightarrow \text{Call } q
\end{align*}
\]

\[
\begin{align*}
S_q & \rightarrow a = \ldots \rightarrow b + c \\
E_q & \rightarrow \text{Call } q
\end{align*}
\]

\[
\begin{align*}
S_r & \rightarrow c \times d \\
E_r & \rightarrow \text{Call } q
\end{align*}
\]
Top-down vs. Bottom-up Interprocedural Analysis

Top-down Analysis for Available Expressions Analysis

Procedure $q$ needs to be processed multiple times

Expression $b + c$ is available in procedure $p$
Expression $a \times b$ 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 $c \times d$ are available in procedure $r$
Top-down Vs. Bottom-up Interprocedural Analysis

Bottom-Up Analysis for Available Expressions Analysis

\[ S_p \xrightarrow{a \times b} \text{Call } q \]

\[ S_q \xrightarrow{a = \ldots} b + c \]

\[ S_r \xrightarrow{c \times d} \text{Call } q \]
Top-down Vs. Bottom-up Interprocedural Analysis

Bottom-Up Analysis for Available Expressions Analysis

Call is replaced by procedure summary

Using procedure summary of g at call sites
Top-down Vs. Bottom-up Interprocedural Analysis

Bottom-Up Analysis for Available Expressions Analysis

Call is replaced by procedure summary

Expression $b + c$ is available in procedure $p$
Expression $a \times b$ is not available in procedure $p$
Top-down Vs. Bottom-up Interprocedural Analysis

Bottom-Up Analysis for Available Expressions Analysis

Call is replaced by procedure summary

Expressions \( b + c \) and \( c \ast d \) 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

Oct 2017
IIT Bombay
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

- 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

- 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

- 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

- 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

- 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 represents legal control flow.

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

\[ x' = f_r(x) \quad y' = f_r(y) \]
Context Sensitivity in Interprocedural Analysis

\[ S_s \rightarrow C_i \rightarrow R_i \rightarrow E_s \]  
\[ S_r \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \right.
Context Sensitivity in Interprocedural Analysis

- $S_s$
- $C_i$
- $R_i$
- $E_s$
- $S_r$
- $E_r$
- $C_j$
- $R_j$
- $E_t$
- $S_t$

Connections:
- $S_s \rightarrow C_i \rightarrow R_i \rightarrow E_s$
- $S_t \rightarrow C_j \rightarrow R_j \rightarrow E_t$
- $S_r \rightarrow E_r$
- $S_r \rightarrow \text{interprocedural transition}$
- $E_r \rightarrow \text{interprocedural transition}$
- $x', y', f_r$
- $c_i, c_j$
Context Sensitivity in Interprocedural Analysis

Oct 2017
Context Sensitivity in Interprocedural Analysis

\[ S_s \xrightarrow{x} C_i \xrightarrow{x'} R_i \xrightarrow{x} S_r \xrightarrow{y} S_t \]

\[ E_s \xrightarrow{c_i} R_i \xrightarrow{x'} E_r \]

\[ S_t \xrightarrow{y} C_j \xrightarrow{y'} R_j \xrightarrow{c_j} E_t \]
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
  - Flow insensitive
  - Context insensitive
    - Flow insensitive
    - Context sensitive
      - Flow sensitive
      - Context sensitive
        - Flow sensitive
        - actually, only caller sensitive
Part 3

Classical Functional Approach
Functional Approach

\[
x' = f_r(x)
\]
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 $u_i$

$$\Phi_r(u_1) \equiv \phi_{id}$$
$$\Phi_r(u_2) \equiv f_1$$
$$\Phi_r(u_3) \equiv f_1$$
$$\Phi_r(u_4) \equiv f_1$$
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 $u_i$

\[
\Phi_r(u_1) \equiv \phi_{id} \\
\Phi_r(u_2) \equiv f_1 \\
\Phi_r(u_3) \equiv f_1 \\
\Phi_r(u_5) \equiv f_2 \circ f_1 \\
\Phi_r(u_4) \equiv f_1 \\
\Phi_r(u_6) \equiv f_1 \\
\Phi_r(u_7) \equiv f_1 \\
\Phi_r(u_8) \equiv f_1
\]
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 \( u_i \)

\[
\begin{align*}
\Phi_r(u_1) & \equiv \phi_{id} \\
\Phi_r(u_2) & \equiv f_1 \\
\Phi_r(u_3) & \equiv f_1 \\
\Phi_r(u_5) & \equiv f_2 \circ f_1 \\
\Phi_r(u_6) & \equiv f_3 \circ f_1 \\
\Phi_r(u_7) & \equiv f_4 \\
\end{align*}
\]
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 \( u_i \)

\[
\begin{align*}
\Phi_r(u_1) & \equiv \phi_{id} \\
\Phi_r(u_2) & \equiv f_1 \\
\Phi_r(u_3) & \equiv f_1 \\
\Phi_r(u_5) & \equiv f_2 \circ f_1 \\
\Phi_r(u_7) & \equiv f_2 \circ f_1 \sqcap f_3 \circ f_1 \\
\Phi_r(u_4) & \equiv f_1 \\
\Phi_r(u_6) & \equiv f_3 \circ f_1 \\
\Phi_r(u_8) & \equiv f_4
\end{align*}
\]
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 \( u_i \)

\[
\begin{align*}
\Phi_r(u_1) & \equiv \phi_{id} \\
\Phi_r(u_2) & \equiv f_1 \\
\Phi_r(u_3) & \equiv f_1 \\
\Phi_r(u_4) & \equiv f_1 \\
\Phi_r(u_5) & \equiv f_2 \circ f_1 \\
\Phi_r(u_6) & \equiv f_3 \circ f_1 \\
\Phi_r(u_7) & \equiv f_2 \circ f_1 \sqcap f_3 \circ f_1 \\
\Phi_r(u_8) & \equiv f_4 \circ (f_2 \circ f_1 \sqcap f_3 \circ f_1)
\end{align*}
\]
Equations for Constructing Summary Flow Functions

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

\[
\Phi_r(I_n) = \begin{cases} 
\phi_{id} & \text{if } n \text{ is } S_r \\
\prod_{p \in \text{pred}(n)} (\Phi_r(O_p)) & \text{otherwise}
\end{cases}
\]

\[
\Phi_r(O_n) = \begin{cases} 
\Phi_s(u) \circ \Phi_r(I_n) & \text{if } n \text{ calls procedure } s \\
f_n \circ \Phi_r(I_n) & \text{otherwise}
\end{cases}
\]

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

\[ r \]

\[ f_1 \]

\[ f_2 \]
Constructing Summary Flow Functions Iteratively

Iteration #1

\[
\begin{align*}
\Phi_r(u_1) &= \phi_{id} \\
\Phi_r(u_2) &= f_1 \\
\Phi_r(u_3) &= f_1 \\
\Phi_r(u_4) &= f_2 \circ f_1
\end{align*}
\]
Constructing Summary Flow Functions Iteratively

\[ \Phi_r(u_1) = \phi_{id} \]
\[ \Phi_r(u_2) = f_1 \]
\[ \Phi_r(u_3) = f_1 \cap f_2 \circ f_1 \]
\[ \Phi_r(u_4) = f_2 \circ (f_1 \cap f_2 \circ f_1) \]
Constructing Summary Flow Functions Iteratively

Iteration #3

\[ \Phi_r(u_1) = \phi_{id} \]

\[ \Phi_r(u_2) = f_1 \]

\[ \Phi_r(u_3) = f_1 \cap f_2 \circ (f_1 \cap f_2 \circ f_1) \]

\[ \Phi_r(u_4) = f_2 \circ (f_1 \cap f_2 \circ (f_1 \cap f_2 \circ f_1)) \]

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>
<thead>
<tr>
<th>Lattice of data flow values</th>
<th>All possible flow functions</th>
<th>Lattice of flow functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\top$</td>
<td>$\top$</td>
<td>$\phi_{id}$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\phi_{\perp}$</td>
</tr>
</tbody>
</table>

$\top = \emptyset$

$\bot = \{a\}$

$\hat{f}_n$, $\hat{f}_n(x)$, $\forall x \in \{\top, \bot\}$

<table>
<thead>
<tr>
<th>Gen$_n$</th>
<th>Kill$_n$</th>
<th>$\hat{f}_n$</th>
<th>$\hat{f}_n(x)$, $\forall x \in {\top, \bot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\hat{\phi}_{id}$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>${a}$</td>
<td>$\hat{\phi}_\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>${a}$</td>
<td>$\emptyset$</td>
<td>$\hat{\phi}_\bot$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>${a}$</td>
<td>${a}$</td>
<td>$\hat{\phi}_\bot$</td>
<td></td>
</tr>
</tbody>
</table>
Reducing Component Flow Functions for Live Variables Analysis

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

- $\hat{\phi}_T \sqcap \hat{\phi} = \hat{\phi}$ (because $0 + x = x$)
- $\hat{\phi}_{\bot} \sqcap \hat{\phi} = \hat{\phi}_{\bot}$ (because $1 + x = 1$)
- $\hat{\phi}_T \circ \hat{\phi} = \hat{\phi}_T$ (because $\hat{\phi}_T$ is a constant function)
- $\hat{\phi}_{\bot} \circ \hat{\phi} = \hat{\phi}_{\bot}$ (because $\hat{\phi}_{\bot}$ is a constant function)
- $\hat{\phi}_{id} \circ \hat{\phi} = \hat{\phi}$ (because $\hat{\phi}_{id}$ is the identity function)
Reducing Function Compositions in Bit Vector Frameworks

$\text{Kill}_n$ denoted by $K_n$ and $\text{Gen}_n$ denoted by $G_n$

$$f_3(x) = f_2(f_1(x))$$
Reducing Function Compositions in Bit Vector Frameworks

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

\[
f_3(x) = f_2(f_1(x)) = f_2((x - K_1) \cup G_1)
\]
Reducing Function Compositions in Bit Vector Frameworks

$\text{Kill}_n$ denoted by $K_n$ and $\text{Gen}_n$ denoted by $G_n$

\[
f_3(x) = f_2(f_1(x)) = f_2((x - K_1) \cup G_1) = (((x - K_1) \cup G_1) - K_2) \cup G_2
\]
Reducing Function Compositions in Bit Vector Frameworks

\[ \text{Kill}_n \text{ denoted by } K_n \text{ and } \text{Gen}_n \text{ denoted by } G_n \]

\[
\begin{align*}
  f_3(x) &= f_2(f_1(x)) \\
        &= f_2((x - K_1) \cup G_1) \\
        &= (((x - K_1) \cup G_1) - K_2) \cup G_2 \\
        &= (x - (K_1 \cup K_2)) \cup (G_1 - K_2) \cup G_2
\end{align*}
\]
Reducing Function Compositions in Bit Vector Frameworks

Killₙ denoted by $Kₙ$ and Genₙ denoted by $Gₙ$

\[ f_3(x) = f_2(f_1(x)) \]
\[ = f_2((x - K_1) \cup G_1) \]
\[ = (((x - K_1) \cup G_1) - K_2) \cup G_2 \]
\[ = (x - (K_1 \cup K_2)) \cup (G_1 - K_2) \cup G_2 \]

Hence,

\[ K_3 = K_1 \cup K_2 \]
\[ G_3 = (G_1 - K_2) \cup G_2 \]
Reducing Bit Vector Flow Function Confluences (1)

Kill\(_n\) denoted by \(K_n\) and Gen\(_n\) denoted by \(G_n\)

- When \(\cap\) is \(\cup\),

\[
\begin{align*}
  f_3(x) &= f_2(x) \cup f_1(x) \\
       &= (x - K_2) \cup G_2 \cup (x - K_1) \cup G_1 \\
       &= (x - (K_1 \cap K_2)) \cup (G_1 \cup G_2)
\end{align*}
\]

Hence,

\[
\begin{align*}
  K_3 &= K_1 \cap K_2 \\
  G_3 &= G_1 \cup G_2
\end{align*}
\]
Reducing Bit Vector Flow Function Confluences (1)

Kill\(_n\) denoted by \(K_n\) and Gen\(_n\) denoted by \(G_n\)

- When \(\cap\) is \(\cup\),

\[
f_3(x) = f_2(x) \cup f_1(x) \\
= ((x - K_2) \cup G_2) \cup ((x - K_1) \cup G_1) \\
= (x - (K_1 \cap K_2)) \cup (G_1 \cup G_2)
\]

Hence,

\[
K_3 = K_1 \cap K_2 \\
G_3 = G_1 \cup G_2
\]
Reducing Bit Vector Flow Function Confluences (2)

Kill\(_n\) denoted by \(K\_n\) and Gen\(_n\) denoted by \(G\_n\)

- When \(\cap\) is \(\cap\),

\[
f_3(x) = f_2(x) \cap f_1(x)
= ((x - K_2) \cup G_2) \cap ((x - K_1) \cup G_1)
= (x - (K_1 \cup K_2)) \cup (G_1 \cap G_2)
\]

Hence,

\[
K_3 = K_1 \cup K_2
G_3 = G_1 \cap G_2
\]
Kill\(_n\) denoted by \(K_n\) and Gen\(_n\) denoted by \(G_n\)

- When \(\cap\) is \(\cap\),

\[
f_3(x) = f_2(x) \cap f_1(x) \\
= ((x - K_2) \cup G_2) \cap ((x - K_1) \cup G_1) \\
= (x - (K_1 \cup K_2)) \cup (G_1 \cap G_2)
\]

Hence,

\[
K_3 = K_1 \cup K_2 \\
G_3 = G_1 \cap G_2
\]
Lattice of Flow Functions for Live Variables Analysis

Flow functions for two variables

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

<table>
<thead>
<tr>
<th>Lattice of data flow values</th>
<th>All possible flow functions</th>
<th>Lattice of flow functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\top = \emptyset$</td>
<td>${a} \quad {\emptyset}$</td>
<td>$\phi_{\top \top}$</td>
</tr>
<tr>
<td>$\bot = {a, b}$</td>
<td>${b} \quad {{a}}$</td>
<td>$\phi_{\top \bot}$</td>
</tr>
<tr>
<td>${a}$</td>
<td>${b} \quad \phi_{\bot}$</td>
<td>$\phi_{\bot \top}$</td>
</tr>
<tr>
<td>${a}$</td>
<td>${a, b} \quad \phi_{\bot \bot}$</td>
<td>$\phi_{\bot}$</td>
</tr>
<tr>
<td>${a}$</td>
<td>${a}$</td>
<td>$\phi_{\bot \bot}$</td>
</tr>
<tr>
<td>${a}$</td>
<td>${b}$</td>
<td>$\phi_{\bot \bot}$</td>
</tr>
<tr>
<td>${a}$</td>
<td>${a, b}$</td>
<td>$\phi_{\bot \bot}$</td>
</tr>
<tr>
<td>${a}$</td>
<td>${a, b}$</td>
<td>$\phi_{\bot \bot}$</td>
</tr>
</tbody>
</table>
An Example of Interprocedural Liveness Analysis

$S_{main}$

\[ a = 5; \ b = 3 \]
\[ c = 7; \text{read } d \]

$\quad c_1$  
**Call p**

$\quad n_1$

\[ a = a + 2 \]
\[ e = c + d \]

$\quad n_2$

\[ d = a \ast b \]

$\quad c_2$

**Call q**

$E_{main}$

**print** $a + c + e$

$S_p$

\[ b = 2 \]
\[ \text{if } (b < d) \]

$\quad n_3$

\[ c = a + b \]

$\quad T$

$\quad F$

$\quad c_4$

**Call q**

$E_p$

**print** $c + d$

$S_q$

\[ a = 1 \]

$\quad c_3$

**Call p**

$E_q$

\[ a = a \ast b \]
# Summary Flow Functions for Interprocedural Liveness Analysis

<table>
<thead>
<tr>
<th>Proc.</th>
<th>Flow Function</th>
<th>Defining Expression</th>
<th>Iteration #1</th>
<th>Changes in iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Gen</td>
<td>Kill</td>
</tr>
<tr>
<td>(p)</td>
<td>(\Phi_p(E_p))</td>
<td>(f_{E_p})</td>
<td>{c, d}</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td></td>
<td>(\Phi_p(n_3))</td>
<td>(f_{n_3} \circ \Phi_p(E_p))</td>
<td>{a, b, d}</td>
<td>{c}</td>
</tr>
<tr>
<td></td>
<td>(\Phi_p(c_4))</td>
<td>(f_q \circ \Phi_p(E_p) = \phi_T)</td>
<td>(\emptyset)</td>
<td>{a, b, c, d, e}</td>
</tr>
<tr>
<td></td>
<td>(\Phi_p(S_p))</td>
<td>(f_{S_p} \circ (\Phi_p(n_3) \sqcap \Phi_p(c_4)))</td>
<td>{a, d}</td>
<td>{b, c}</td>
</tr>
<tr>
<td></td>
<td>(f_p)</td>
<td>(\Phi_p(S_p))</td>
<td>{a, d}</td>
<td>{b, c}</td>
</tr>
<tr>
<td>(q)</td>
<td>(\Phi_q(E_q))</td>
<td>(f_{E_q})</td>
<td>{a, b}</td>
<td>{a}</td>
</tr>
<tr>
<td></td>
<td>(\Phi_q(c_3))</td>
<td>(f_p \circ \Phi_q(E_q))</td>
<td>{a, d}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td></td>
<td>(\Phi_q(S_q))</td>
<td>(f_{S_q} \circ \Phi_q(c_3))</td>
<td>{d}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td></td>
<td>(f_q)</td>
<td>(\Phi_q(S_q))</td>
<td>{d}</td>
<td>{a, b, c}</td>
</tr>
</tbody>
</table>
Computed Summary Flow Functions

\[
\begin{align*}
S_p & \quad b = 2 \quad \text{if} \ (b < d) \\
E_p & \quad c = a + b \quad \text{print} \ c + d \\
S_q & \quad a = 1 \\
E_q & \quad a = a \times b \\
n_3 & \\
c_4 & \quad \text{Call q}
\end{align*}
\]

Summary Flow Function

<table>
<thead>
<tr>
<th>( \Phi_p(E_p) )</th>
<th>( BI_p \cup { c, d } )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Phi_p(n_3) )</td>
<td>( (BI_p - { c }) \cup { a, b, d } )</td>
</tr>
<tr>
<td>( \Phi_p(c_4) )</td>
<td>( (BI_p - { a, b, c }) \cup { d } )</td>
</tr>
<tr>
<td>( \Phi_p(S_p) )</td>
<td>( (BI_p - { b, c }) \cup { a, d } )</td>
</tr>
<tr>
<td>( \Phi_q(E_q) )</td>
<td>( (BI_q - { a }) \cup { a, b } )</td>
</tr>
<tr>
<td>( \Phi_q(c_3) )</td>
<td>( (BI_q - { a, b, c }) \cup { a, d } )</td>
</tr>
<tr>
<td>( \Phi_q(S_q) )</td>
<td>( (BI_q - { a, b, c }) \cup { d } )</td>
</tr>
</tbody>
</table>
# Result of Interprocedural Liveness Analysis

<table>
<thead>
<tr>
<th>Data flow variable</th>
<th>Summary flow function</th>
<th>Data flow value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$In_{E_m}$</td>
<td>$\Phi_m(E_m)$</td>
<td>$B_l_m \cup {a, c, e}$</td>
</tr>
<tr>
<td>$In_{c_2}$</td>
<td>$\Phi_m(c_2)$</td>
<td>$B_l_m - {a, b, c} \cup {d, e}$</td>
</tr>
<tr>
<td>$In_{n_2}$</td>
<td>$\Phi_m(n_2)$</td>
<td>$B_l_m - {a, b, c, d} \cup {a, b, e}$</td>
</tr>
<tr>
<td>$In_{n_1}$</td>
<td>$\Phi_m(n_1)$</td>
<td>$B_l_m - {a, b, c, d, e} \cup {a, b, c, d}$</td>
</tr>
<tr>
<td>$In_{c_1}$</td>
<td>$\Phi_m(c_1)$</td>
<td>$B_l_m - {a, b, c, d, e} \cup {a, d}$</td>
</tr>
<tr>
<td>$In_{S_m}$</td>
<td>$\Phi_m(S_m)$</td>
<td>$B_l_m - {a, b, c, d, e}$</td>
</tr>
</tbody>
</table>
### Result of Interprocedural Liveness Analysis

<table>
<thead>
<tr>
<th>Data flow variable</th>
<th>Summary flow function</th>
<th>Data flow value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Procedure p, ( Bl = {a, b, c, d, e} )</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( In_{E_p} )</td>
<td>( \Phi_p(E_p) )</td>
<td>( Bl_p \cup {c, d} )</td>
</tr>
<tr>
<td>( In_{n_3} )</td>
<td>( \Phi_p(n_3) )</td>
<td>( (Bl_p - {c}) \cup {a, b, d} )</td>
</tr>
<tr>
<td>( In_{c_4} )</td>
<td>( \Phi_p(c_4) )</td>
<td>( (Bl_p - {a, b, c}) \cup {d} )</td>
</tr>
<tr>
<td>( In_{S_p} )</td>
<td>( \Phi_p(S_p) )</td>
<td>( (Bl_p - {b, c}) \cup {a, d} )</td>
</tr>
<tr>
<td><strong>Procedure q, ( Bl = {a, b, c, d, e} )</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( In_{E_q} )</td>
<td>( \Phi_q(E_q) )</td>
<td>( (Bl_q - {a}) \cup {a, b} )</td>
</tr>
<tr>
<td>( In_{c_3} )</td>
<td>( \Phi_q(c_3) )</td>
<td>( (Bl_q - {a, b, c}) \cup {a, d} )</td>
</tr>
<tr>
<td>( In_{S_q} )</td>
<td>( \Phi_q(S_q) )</td>
<td>( (Bl_q - {a, b, c}) \cup {d} )</td>
</tr>
</tbody>
</table>
Context Sensitivity of Interprocedural Liveness Analysis

\[ S_{\text{main}} \]
\[ a = 5; b = 3 \]
\[ c = 7; \text{read } d \]
\[ \{a, d\} \]
\[ \text{Call } p \]
\[ \{a, b, c, d\} \]
\[ a = a + 2 \]
\[ e = c + d \]
\[ \{a, b, e\} \]
\[ \text{Call } q \]
\[ \{a, c, e\} \]
\[ E_{\text{main}} \]
\[ \text{print } a + c + e \]

\[ S_p \]
\[ b = 2 \]
\[ \text{if } (b < d) \]
\[ \{a, b, d, e\} \]
\[ T \]
\[ \{a, b, c, d, e\} \]
\[ \text{Call } q \]
\[ \{d, e\} \]
\[ E_p \]
\[ \text{print } c + d \]

\[ S_q \]
\[ a = 1 \]
\[ \{a, d, e\} \]
\[ \text{Call } p \]
\[ \{a, b, c, d, e\} \]
\[ \text{Call } q \]
\[ \{a, d, e\} \]
\[ \text{Call } p \]
\[ \{a, b, c, d, e\} \]
\[ \text{Call } q \]
\[ \{a, d, e\} \]
\[ \text{Call } p \]
\[ \{a, b, c, d, e\} \]
Context Sensitivity of Interprocedural Liveness Analysis

### States

- **S_{main}**
  - `a = 5; b = 3`
  - `c = 7; read d`
  - Transition: Call `p`
  - Next State: `{a, d}`

- **c_1**
  - Transition: Call `p`
  - Next State: `{a, b, c, d}`

- **n_1**
  - `a = a + 2`
  - `e = c + d`
  - Transition: `n_1`
  - Next State: `{a, b, e}`

- **n_2**
  - `d = a * b`
  - Transition: `n_2`
  - Next State: `{d, e}`

- **E_{main}**
  - `print a + c + e`

- **S_p**
  - `b = 2`
  - `if (b < d)`
  - Transition: `T`
  - Next State: `{a, b, d, e}`

- **n_3**
  - `c = a + b`
  - Transition: `n_3`
  - Next State: `{a, b, c, d, e}`

- **c_4**
  - Transition: Call `q`
  - Next State: `{a, b, c, d, e}`

- **S_q**
  - `a = 1`
  - Transition: `c_3`
  - Next State: `{a, d, e}`

- **c_3**
  - Transition: Call `p`
  - Next State: `{a, b, c, d, e}`

- **E_q**
  - `a = a * b`
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
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

\[ b = c + d \]

\[ a = a + b \]

\[ C_1 \text{ Call } p \]

\[ a = a - b \]

\[ S_p \]

\[ n_1 \]

\[ n_2 \]

\[ E_p \]

Oct 2017 IIT Bombay
Tutorial Problem #2: Summary Flow Function for Constant Propagation

\[ b = c + d \]

\[ n_1 \quad a = a + b \]

\[ n_2 \quad a = a - b \]

\[ C_1 \quad \text{Call } p \]

\[ E_p \]

<table>
<thead>
<tr>
<th></th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Phi_p(S_p) ]</td>
<td>[ v_a, v_b ]</td>
<td>[ v_a, v_b ]</td>
</tr>
<tr>
<td>[ \Phi_p(n_1) ]</td>
<td>[ v_a + v_b, v_b ]</td>
<td>[ v_a + v_b, v_b ]</td>
</tr>
<tr>
<td>[ \Phi_p(C_1) ]</td>
<td>[ \top, \top ]</td>
<td>[ v_a + v_b, v_b ]</td>
</tr>
<tr>
<td>[ \Phi_p(n_2) ]</td>
<td>[ \top, \top ]</td>
<td>[ v_a, v_b ]</td>
</tr>
<tr>
<td>[ \Phi_p(E_p) ]</td>
<td>[ v_a, v_b ]</td>
<td>[ v_a, v_b ]</td>
</tr>
<tr>
<td>[ f_p ]</td>
<td>[ v_a, v_b ]</td>
<td>[ v_a, v_b ]</td>
</tr>
</tbody>
</table>
Tutorial Problem #2: Summary Flow Function for Constant Propagation

\[ b = c + d \]

\[ a = a + b \]  \hspace{1cm} \text{Iter. #1} \hspace{1cm} \text{Iter. #2}

| \[ \Phi_p(S_p)(\langle v_a, v_b \rangle) \] | \[ \langle v_a, v_b \rangle \] | \[ \langle v_a, v_b \rangle \] |
| \[ \Phi_p(n_1)(\langle v_a, v_b \rangle) \] | \[ \langle v_a + v_b, v_b \rangle \] | \[ \langle v_a + v_b, v_b \rangle \] |
| \[ \Phi_p(C_1)(\langle v_a, v_b \rangle) \] | \[ \langle \top, \top \rangle \] | \[ \langle v_a + v_b, v_b \rangle \] |
| \[ \Phi_p(n_2)(\langle v_a, v_b \rangle) \] | \[ \langle \top, \top \rangle \] | \[ \langle v_a, v_b \rangle \] |
| \[ \Phi_p(E_p)(\langle v_a, v_b \rangle) \] | \[ \langle v_a, v_b \rangle \] | \[ \langle v_a, v_b \rangle \] |

\[ f_p(\langle v_a, v_b \rangle) \] | \[ \langle v_a, v_b \rangle \] | \[ \langle v_a, v_b \rangle \] |

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 $f : L \rightarrow L$ is $\langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle$
- Component function: $\hat{h}_i$ which computes the value of $\hat{x}_i$
Overall Flow Function and Component Function

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

| Separable | General Non-Separable |
Overall Flow Function and Component Function

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

Separable

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \xrightarrow{f} \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

General Non-Separable

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \xrightarrow{f} \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]
Overall Flow Function and Component Function

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

Separable

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ \hat{h}_2 \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

General Non-Separable

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ f \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]
Overall Flow Function and Component Function

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

Separable

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle
\]

\( \hat{h}_2 \)

\[
\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

General Non-Separable

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle
\]

\( f \)

\[
\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

\( \hat{h} : \hat{L} \mapsto \hat{L} \)
Overall Flow Function and Component Function

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

Separable

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

$\hat{h}_2$

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

General Non-Separable

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

$\hat{h}_2$

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

$\hat{h} : \hat{L} \mapsto \hat{L}$
Overall Flow Function and Component Function

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

**Separable**

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

$\hat{h}_2$

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

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

**General Non-Separable**

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

$\hat{h}_2$

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

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

Example: All bit vector frameworks

Example: Points-To Analysis
## Entity Functions in Points-to Analysis

<table>
<thead>
<tr>
<th>Statement with $a \in L_{locations}$</th>
<th>Entity functions</th>
<th>Closed under composition?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$... = \text{null}$</td>
<td>Constant</td>
<td>$\hat{L} \mapsto \hat{L}$</td>
</tr>
<tr>
<td>$... = &amp; b$</td>
<td>Constant</td>
<td>$\hat{L} \mapsto \hat{L}$</td>
</tr>
<tr>
<td>$... = b$</td>
<td>Identity</td>
<td>$\hat{L} \mapsto \hat{L}$</td>
</tr>
<tr>
<td>$... = \ast b$</td>
<td>?</td>
<td>$L \mapsto \hat{L}$</td>
</tr>
</tbody>
</table>
Entity Functions in Constant Propagation

<table>
<thead>
<tr>
<th>Statement</th>
<th>Entity functions</th>
<th>Closed under composition?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = 5$</td>
<td>Constant</td>
<td>$\hat{L} \mapsto \hat{L}$</td>
</tr>
<tr>
<td>$a = b$</td>
<td>Constant</td>
<td>$\hat{L} \mapsto \hat{L}$</td>
</tr>
<tr>
<td>$a = b + 5$</td>
<td>Linear</td>
<td>$\hat{L} \mapsto \hat{L}$</td>
</tr>
<tr>
<td>$a = b + c$</td>
<td>?</td>
<td>$L \mapsto \hat{L}$</td>
</tr>
</tbody>
</table>
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

Oct 2017 IIT Bombay
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
• “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
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
“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
  - $\text{IN}_n$ and $\text{OUT}_n$ are partial maps from call strings to $L$
  - The final data flow information at a program point is
    \[
    \text{In}_n = \bigcap_{\langle \sigma, x \rangle \in \text{IN}_n} x
    \]
    \[
    \text{Out}_n = \bigcap_{\langle \sigma, x \rangle \in \text{OUT}_n} x
    \]
    (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 $\sigma$
Augmented Data Flow Equations: Computing $\text{IN}_n$

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

where we merge underlying data flow values only if the contexts are same.
Augmented Data Flow Equations: Computing $\text{IN}_n$

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

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

\[
\Gamma_1 \uplus \Gamma_2 = \begin{cases} 
\langle \sigma, z \rangle & \text{if } \langle \sigma, x \rangle \in \Gamma_1 \land \langle \sigma, y \rangle \in \Gamma_2 \Rightarrow z = x \sqcap y, \\
\langle \sigma, x \rangle \in \Gamma_1 \land \langle \sigma, y \rangle \notin \Gamma_2 \Rightarrow z = x, \\
\langle \sigma, x \rangle \notin \Gamma_1 \land \langle \sigma, y \rangle \in \Gamma_2 \Rightarrow z = y
\end{cases}
\]
Augmented Data Flow Equations: Computing $\text{OUT}_n$

- Call node $C_i$
  - Append $c_i$ to every $\sigma$
  - Propagate the data flow values unchanged
Augmented Data Flow Equations: Computing $\text{OUT}_n$

- Call node $C_i$
  - Append $c_i$ to every $\sigma$
  - 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 $\text{OUT}_n$

- Call node $C_i$
  - Append $c_i$ to every $\sigma$
  - 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

Ascend

Descend
Augmented Data Flow Equations: Computing $\text{OUT}_n$

- **Call node $C_i$**
  - Append $c_i$ to every $\sigma$
  - 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

$$\text{OUT}_n(X) = \begin{cases} 
\{ \langle \sigma \cdot c_i, x \rangle \mid \langle \sigma, x \rangle \in \text{IN}_n \} & n \text{ is } C_i \\
\{ \langle \sigma, x \rangle \mid \langle \sigma \cdot c_i, x \rangle \in \text{IN}_n \} & n \text{ is } R_i \\
\{ \langle \sigma, f_n(x) \rangle \mid \langle \sigma, x \rangle \in \text{IN}_n \} & \text{otherwise}
\end{cases}$$
Available Expressions Analysis Using Call Strings Approach

$S_{main}$
- read $a, b$
- $t := a \times b$

$C_1$
- call $p$

$R_1$
- $n_1$
- print $a \times b$

$E_{main}$

$S_p$
- if $a == 0$

$n_2$
- $a = a - 1$

$C_2$
- call $p$

$R_2$
- $n_3$
- $t = a \times b$

$E_p$

Oct 2017 IIT Bombay
Available Expressions Analysis Using Call Strings Approach

\[ S_{main} \]
- Read \( a, b \)
- \( t := a * b \)

\[ C_1 \]
- Call \( p \)

\[ R_1 \]
- \( n_1 \) Print \( a * b \)
- Is \( a * b \) available?

\[ E_{main} \]

\[ S_p \]
- If \( a == 0 \)

\[ n_2 \]
- \( a = a - 1 \)

\[ C_2 \]
- Call \( p \)

\[ R_2 \]
- \( n_3 \) \( t = a * b \)

\[ E_p \]
Available Expressions Analysis Using Call Strings Approach

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

Is \( a \cdot b \) available?
Available Expressions Analysis Using Call Strings Approach

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

Yes!
Available Expressions Analysis Using Call Strings Approach

\[ S_{main} \]
read \textit{a, b}
t := \textit{a} \times \textit{b}

\[ C_1 \]
call \textit{p}

\[ S_p \]
if \textit{a} == 0

\[ n_2 \]
a = a - 1

\[ C_2 \]
call \textit{p}

\[ R_1 \]

\[ R_2 \]

\[ n_1 \]
print \textit{a} \times \textit{b}

\[ n_3 \]
t = \textit{a} \times \textit{b}

\[ E_{main} \]

\[ E_p \]

\[ n_2 \]
Kill

Oct 2017
Available Expressions Analysis Using Call Strings Approach

\[ S_{main} \rightarrow \text{read } a, b \rightarrow t := a \ast b \]

\[ C_1 \rightarrow \text{call } p \]

\[ R_1 \rightarrow \text{print } a \ast b \]

\[ E_{main} \rightarrow \]

\[ S_p \rightarrow \text{if } a == 0 \]

\[ n_2 \rightarrow a = a - 1 \]

\[ C_2 \rightarrow \text{call } p \]

\[ R_2 \rightarrow \]

\[ n_3 \rightarrow t = a \ast b \]

\[ E_p \rightarrow \]

\[ \text{Kill Oct 2017 IIT Bombay} \]
Available Expressions Analysis Using Call Strings Approach

$S_{main}$

- **read** $a, b$
- $t := a \times b$

$C_1$
- **call** $p$

$R_1$

$n_1$
- **print** $a \times b$

$E_{main}$

$S_p$

- **if** $a == 0$

$n_2$
- $a = a - 1$

$C_2$
- **call** $p$

$R_2$

$n_3$
- $t = a \times b$

$E_p$

Kill

Oct 2017
Available Expressions Analysis Using Call Strings Approach

\[ S_{main} \]
- read \( a, b \)
- \( t := a \times b \)

\[ C_1 \]
- call \( p \)

\[ R_1 \]
- \( n_1 \) print \( a \times b \)

\[ E_{main} \]

\[ S_p \]
- if \( a == 0 \)

\[ n_2 \]
- \( a = a - 1 \)

\[ C_2 \]
- call \( p \)

\[ R_2 \]
- \( n_3 \) \( t = a \times b \)

\[ E_p \]
- \( \text{Gen} \)

\[ \text{Kill} \]
Available Expressions Analysis Using Call Strings Approach

\[ S_{main} \]
\[ \text{read } a, b \]
\[ t := a \ast b \]
\[ C_1 \]
\[ \text{call } p \]

\[ E_{main} \]

\[ R_1 \]

\[ n_1 \]
\[ \text{print } a \ast b \]

\[ S_p \]
\[ \text{if } a == 0 \]

\[ C_2 \]
\[ \text{call } p \]

\[ E_p \]

\[ n_2 \]
\[ a = a - 1 \]

\[ n_3 \]
\[ t = a \ast b \]

\[ R_2 \]

\[ \text{Kill} \]
\[ \text{Gen} \]
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\[ S_{main} \]

\[ \text{read } a, b \]
\[ t := a \times b \]

\[ C_1 \]

\[ \text{call } p \]

\[ R_1 \]

\[ n_1 \]

\[ \text{print } a \times b \]

\[ E_{main} \]

\[ S_p \]

\[ \text{if } a == 0 \]

\[ n_2 \]

\[ a = a - 1 \]

\[ C_2 \]

\[ \text{call } p \]

\[ R_2 \]

\[ n_3 \]

\[ t = a \times b \]

\[ E_p \]
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\[ S_{\text{main}} \]
- read \( a, b \)
- \( t := a \times b \)

\[ C_1 \]
- call \( p \)

\[ R_1 \]

\[ n_1 \]
- print \( a \times b \)

\[ E_{\text{main}} \]

\[ S_p \]
- if \( a == 0 \)

\[ n_2 \]
- \( a = a - 1 \)

\[ C_2 \]
- call \( p \)

\[ R_2 \]

\[ n_3 \]
- \( t = a \times b \)

\[ E_p \]
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\[ S_{\text{main}} \]
- \( \text{read } a, b \)
- \( t := a \ast b \)

\[ C_1 \]
- call \( p \)

\[ R_1 \]

\[ n_1 \]
- print \( a \ast b \)

\[ E_{\text{main}} \]

\[ \langle \lambda, 1 \rangle \]

\[ \langle c_1, 1 \rangle \]

\[ S_p \]
- if \( a == 0 \)

\[ n_2 \]
- \( a = a - 1 \)

\[ C_2 \]
- call \( p \)

\[ R_2 \]

\[ n_3 \]
- \( t = a \ast b \)

\[ E_p \]
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\[ S_{main} \]
\[ \text{read } a, b \]
\[ t := a \ast b \]
\[ \langle \lambda, 1 \rangle \]
\[ C_1 \]
\[ \text{call } p \]
\[ \langle \lambda, 1 \rangle \]
\[ R_1 \]
\[ n_1 \]
\[ \text{print } a \ast b \]
\[ E_{main} \]
\[ \langle c_1, 1 \rangle \]
\[ \text{if } a == 0 \]
\[ S_p \]
\[ n_2 \]
\[ a = a - 1 \]
\[ C_2 \]
\[ \text{call } p \]
\[ \langle \lambda, 1 \rangle \]
\[ R_2 \]
\[ n_3 \]
\[ t = a \ast b \]
\[ \langle c_1, 1 \rangle \]
\[ E_p \]
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\( S_{main} \)

read \( a, b \)
\( t := a \ast b \)

\( \langle \lambda, 1 \rangle \)

\( C_1 \)
call \( p \)

\( \langle c_1, 1 \rangle \)

\( S_p \)

if \( a == 0 \)

\( n_2 \)
\( a = a - 1 \)

\( \langle c_1, 0 \rangle \)

\( C_2 \)
call \( p \)

\( \langle c_1, 1 \rangle \)

\( R_1 \)

\( n_1 \)
print \( a \ast b \)

\( \langle c_1, 1 \rangle \)

\( R_2 \)

\( n_3 \)
\( t = a \ast b \)

\( \langle c_1, 0 \rangle \)

\( E_{main} \)

\( \langle \lambda, 1 \rangle \)

\( \langle c_1, 0 \rangle \)

\( E_p \)

Maintain a worklist of nodes to be processed
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\( S_{main} \)
- \( \text{read } a, b \)
- \( t := a \ast b \)

\( C_1 \)
- \( \text{call } p \)

\( R_1 \)
- \( n_1 \) print \( a \ast b \)

\( E_{main} \)

\( S_p \)
- if \( a == 0 \)

\( C_2 \)
- \( \text{call } p \)

\( R_2 \)
- \( n_3 \) \( t = a \ast b \)

\( n_2 \) \( a = a - 1 \)

\( E_p \)
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\[ S_{\text{main}} \]
\[ \text{read } a, b \]
\[ t := a \times b \]
\[ \langle \lambda, 1 \rangle \]
\[ C_1 \]
\[ \text{call } p \]
\[ n_1 \]
\[ \text{print } a \times b \]
\[ E_{\text{main}} \]

\[ S_p \]
\[ \text{if } a == 0 \]
\[ \langle c_1, 1 \rangle \]
\[ n_2 \]
\[ a = a - 1 \]
\[ \langle c_1, 0 \rangle \]
\[ C_2 \]
\[ \text{call } p \]

\[ n_3 \]
\[ t = a \times b \]

\[ E_p \]
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

$S_{main}$
- read $a, b$
- $t := a \times b$

$C_1$
- call $p$

$n_1$
- print $a \times b$

$E_{main}$

$S_p$
- if $a == 0$

$n_2$
- $a = a - 1$

$C_2$
- call $p$

$n_3$
- $t = a \times b$

$R_1$

$R_2$

$\langle c_1, 1 \rangle$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle \lambda, 1 \rangle$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 1 \rangle$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 1 \rangle$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots$

$\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

$S_{main}$
- read $a, b$
- $t := a \times b$

$C_1$
- call $p$

$R_1$

$n_1$
- print $a \times b$

$E_{main}$

$S_p$
- if $a == 0$

$n_2$
- $a = a - 1$

$C_2$
- call $p$

$n_3$
- $t = a \times b$

$E_p$

$\langle c_1, 1 \rangle$

$\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_2, 0 \rangle, \ldots$

Oct 2017
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\[ S_{main} \]
- read \( a, b \)
- \( t := a \times b \)

\[ C_1 \]
- call \( p \)

\[ R_1 \]

\[ n_1 \]
- print \( a \times b \)

\[ E_{main} \]

\[ S_p \]
- if \( a == 0 \)

\[ n_2 \]
- \( a = a - 1 \)

\[ C_2 \]
- call \( p \)

\[ R_2 \]

\[ n_3 \]
- \( t = a \times b \)

\[ E_p \]
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

$S_{main}$
- read $a, b$
- $t := a \times b$
- $\langle \lambda, 1 \rangle$

$C_1$
- call $p$

$R_1$

$n_1$
- print $a \times b$

$E_{main}$

$S_p$
- if $a == 0$
- $\langle c_1, 1 \rangle$
- $\langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_2, 0 \rangle, \ldots$

$n_2$
- $a = a - 1$
- $\langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots$

$C_2$
- call $p$

$R_2$

$n_3$
- $t = a \times b$
- $\langle c_1, 1 \rangle$
- $\langle c_1, 0 \rangle$
- $\langle c_1 c_2, 0 \rangle$
- $\langle c_1 c_2 c_2, 0 \rangle$
- $\ldots$

$E_p$
- $\langle c_1 c_2, 1 \rangle$
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\[ S_{main} \]
\[ \text{read } a, b \]
\[ t := a \times b \]
\[ \langle \lambda, 1 \rangle \]
\[ C_1 \]
\[ \text{call } p \]
\[ \langle c_1, 1 \rangle \]
\[ \langle c_1 c_2, 0 \rangle \]
\[ \langle c_1 c_2 c_0, 0 \rangle \]
\[ E_{main} \]

\[ R_1 \]
\[ n_1 \]
\[ \text{print } a \times b \]

\[ S_p \]
\[ \text{if } a == 0 \]
\[ \langle c_1, 1 \rangle \]
\[ \langle c_1 c_2, 0 \rangle, \langle c_1 c_2 c_0, 0 \rangle, \ldots \]

\[ n_2 \]
\[ a = a - 1 \]
\[ \langle c_1, 0 \rangle, \langle c_1 c_2, 0 \rangle, \ldots \]

\[ C_2 \]
\[ \text{call } p \]
\[ \langle c_1, 1 \rangle \]
\[ \langle c_1 c_2, 0 \rangle \]
\[ \langle c_1 c_2 c_0, 0 \rangle \]
\[ \ldots \]

\[ R_2 \]
\[ n_3 \]
\[ t = a \times b \]
\[ \langle c_1, 0 \rangle \]
\[ \langle c_1 c_2, 0 \rangle \]
\[ \langle c_1 c_2 c_0, 0 \rangle \]
\[ \ldots \]

\[ E_p \]
\[ \langle c_1 c_2, 1 \rangle \]
Available Expressions Analysis Using Call Strings Approach

Maintain a worklist of nodes to be processed

\( S_{\text{main}} \)

- read \( a, b \)
- \( t := a \cdot b \)

\( C_1 \)
- call \( p \)

\( R_1 \)

\( n_1 \)
- print \( a \cdot b \)

\( E_{\text{main}} \)

\( S_p \)
- if \( a == 0 \)

\( n_2 \)
- \( a = a - 1 \)

\( C_2 \)
- call \( p \)

\( R_2 \)

\( n_3 \)
- \( t = a \cdot b \)

\( E_p \)
Tutorial Problem #1

Perform available expressions analysis for the following program

```c
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

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. {
4.   c = a * b;
5. }
6. void p()
7. {
8.   p();
9.  /*Is a*b available?*/
10. a = a * b;
11. }
12. }

Recursive calls: 1

Path 1

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

Oct 2017
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

Path 1

Path 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. }
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. }
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. }

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

Oct 2017
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

Oct 2017
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

\[ C_2, S_p, E_p, R_2, \text{Kill} n_2, E_p, R_2, n_2 \]
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. {
   4.     c = a*b;
   5. }
6. void p()
7. {
   8.     if (...)
   9.     {
         p();
   10.     }
   11.     Is a*b available?
   12. }

• Interprocedurally valid IFP

\[ C_2, S_p, C_2, S_p, E_p, R_2, n_2, E_p, R_2, n_2 \]
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.    }`
10. `Is a*b available?`
11. `a = a*b;`
12. `}`

- Interprocedurally valid IFP

\[ S_{main}, n_1, C_1, S_p, C_2, S_p, C_2, S_p, E_p, R_2, \text{Kill } n_2, E_p, R_2, n_2 \]
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

\[ S_m, n_1, C_1, S_p, C_2, S_p, C_2, S_p, E_p, R_2, \text{Kill}_{n_2}, E_p, R_2, n_2 \]
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

\[ S_m, n_1, C_1, S_p, C_2, S_p, C_2, S_p, E_p, R_2, \text{Kill}_{n_2}, E_p, R_2, n_2 \]

- 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
  \[ S_m, n_1, C_1, S_p, C_2, S_p, C_2, S_p, E_p, R_2, \text{Kill } n_2, E_p, R_2, n_2 \]

- 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 up to 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 up to 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 up to the following length must be constructed
    - $K \cdot (|L| + 1)^2$ for general bounded frameworks
      ($L$ is the overall lattice of data flow values)
    - $K \cdot (|\hat{L}| + 1)^2$ for separable bounded frameworks
      ($\hat{L}$ 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 up to the following length must be constructed
  - $K \cdot (|L| + 1)^2$ for general bounded frameworks
    ($L$ is the overall lattice of data flow values)
  - $K \cdot (|\hat{L}| + 1)^2$ for separable bounded frameworks
    ($\hat{L}$ 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 up to the following length must be constructed
      - \( K \cdot (|L| + 1)^2 \) for general bounded frameworks
        (\( L \) is the overall lattice of data flow values)
      - \( K \cdot (|\hat{L}| + 1)^2 \) for separable bounded frameworks
        (\( \hat{L} \) 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

\( \Rightarrow \) 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 up to the following length must be constructed
    - $K \cdot (|L| + 1)^2$ for general bounded frameworks
      ($L$ is the overall lattice of data flow values)
    - $K \cdot (|\hat{L}| + 1)^2$ for separable bounded frameworks
      ($\hat{L}$ 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

  $\Rightarrow$ Not a bound but prescribed necessary length

  $\Rightarrow$ 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 $\rho$ that do not have the matching return node in $\rho$
    (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
  $$CS(\rho') \leq M, \text{ and } f_\rho(BI) = f_{\rho'}(BI)$$

  $\Rightarrow \rho$, the longer path, is redundant for data flow analysis.
Sharir-Pnueli [1981]

- Consider the smallest prefix \( \rho_0 \) of \( \rho \) such that \( CS(\rho_0) > M \)
- Consider a triple \( \langle c_i, \alpha_i, \beta_i \rangle \) where
  - \( \alpha_i \) is the data flow value reaching call node \( C_i \) along \( \rho \) and
  - \( \beta_i \) is the data flow value reaching the corresponding return node \( R_i \) along \( \rho \)
  - If \( R_i \) is not in \( \rho \), then \( \beta_i = \Omega \) (undefined)
Classical Call String Length

$M$

$\rho_0$

$\rho$
Classical Call String Length

$M$

$\alpha_i$

$\beta_i$

$\langle c_i, \alpha_i, \beta_i \rangle$

$\rho_0$

$\rho$
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.
When $\beta_i$ is not $\Omega$
Classical Call String Length

When $\beta_i$ is not $\Omega$
When $\beta_i$ is not $\Omega$
Classical Call String Length

When $\beta_i$ is $\Omega$

\[ M \]

[Diagram showing a step function increasing from $\rho_0$ to $\rho$]
When $\beta_i$ is $\Omega$
Classical Call String Length

When $\beta_i$ is $\Omega$
Tighter Bound for Bit Vector Frameworks

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

<table>
<thead>
<tr>
<th>Raise</th>
<th>Lower</th>
<th>Propagate</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>
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 Tighter Bound for Bit Vector Frameworks
Relevant Path Segments for Tighter Bound for Bit Vector Frameworks
Relevant Path Segments for Tighter 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 Tighter 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
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:

\textbf{Source} is a call node \( C_S \) and \textbf{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 $R_T$

- $P(I \leadsto C_S \leadsto C_T \leadsto R_T)$

- $P(I \leadsto C_T \leadsto C_S \leadsto R_S \leadsto R_T)$
Tighter Length for Bit Vector Frameworks

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

- $P(I_C \xrightarrow{C_S} C_T \xrightarrow{\circ} R_T)$
  
  Call strings are derived from the paths $P(I_C \xrightarrow{C_S} C_T \xrightarrow{\circ} C_L)$ 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 $R_T$

- $P(I \leadsto C_S \leadsto C_T \leadsto R_T)$

  - 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 $P(C_T \leadsto C_L)$
Tighter Length for Bit Vector Frameworks

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

- $P(I \leadsto C_S \leadsto C_T \leadsto R_T)$

  - 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
Tighter Length for Bit Vector Frameworks

Case B:

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

- $P(I \leadsto C_S \leadsto C_T \leadsto R_T)$
  - 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

- $P(I \leadsto C_T \leadsto C_S \leadsto R_S \leadsto R_T)$
  - $C_T$ 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 $R_T$

- $P(I \rightsquigarrow C_S \rightsquigarrow C_T \rightsquigarrow R_T)$
  - 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
- $P(I \rightsquigarrow C_T \rightsquigarrow C_S \rightsquigarrow R_S \rightsquigarrow R_T)$
  - $C_T$ 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 $R_T$

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

- $P(I \mapsto (C_T) \mapsto C_S \mapsto R_S \mapsto R_T)$
  - $C_T$ is required because of validity constraints
  - Call strings are derived from the paths $P(I \mapsto C_T \mapsto C_S \mapsto C_L)$ where $C_L$ is the last call node
  - Again, there are three acyclic segments and at most 3 occurrences of a call site
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 up to a given length $m$

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

$C_a$

$R_a$
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

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

$\downarrow$

$C_a$

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

$\downarrow$

$R_a$
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

Call string of length $m - 1$

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

$$\downarrow$$

$C_a$

Call string of length $m$

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

$$\downarrow$$

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

$$\downarrow$$

$R_a$
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

$$\langle C_i \cdot C_i \ldots C_{i_{m-1}} | x \rangle$$

$$\langle C_i \cdot C_i \ldots C_{i_{m-1}} \cdot C_a | x \rangle$$

$$\langle C_i \cdot C_i \ldots C_{i_{m-1}} \cdot C_a | y \rangle$$

$$\langle C_i \cdot C_i \ldots C_{i_{m-1}} | y \rangle$$
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

Call string of length $m$  \[ \langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x \rangle \]

\[ C_a \]

\[ R_a \]
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given 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 $c_{i_1}$ removed from incoming call string and call site $c_a$ attached)

$R_a$
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

\[
\langle C_i \cdot C_2 \ldots C_m \mid x \rangle
\]

\[
\frac{\downarrow}{C_a}
\]

\[
\langle C_i \ldots C_m \cdot C_a \mid x \rangle
\]

\[
\frac{\downarrow}{C_i \ldots C_m \cdot C_a \mid y}
\]

\[
\frac{\downarrow}{R_a}
\]
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

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

\[ C_a \]

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

\[ R_a \]

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

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

(First call site $c_{i_1}$ removed from incoming call string and call site $c_a$ attached)
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 up to a given length $m$

\[
\langle C_{i_1} \cdot C_{i_2} \cdots C_{i_m} \mid x_1 \rangle \quad \langle C_{j_1} \cdot C_{i_2} \cdots C_{i_m} \mid x_2 \rangle
\]

\[C_a\]

\[R_a\]
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

$$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_1 \rangle \quad \langle C_{j_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_2 \rangle$$

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

$C_a$ 

$R_a$
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

\[
\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_1 \rangle, \quad \langle C_{j_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_2 \rangle
\]

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

\[
\langle C_{i_2} \cdot C_{i_3} \ldots C_{i_m} \cdot C_a \mid y \rangle
\]
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

\[
\begin{align*}
\langle C_{i_1} \cdot C_{i_2} \cdots C_{i_m} | x_1 \rangle & \quad \langle C_{j_1} \cdot C_{i_2} \cdots C_{i_m} | x_2 \rangle \\
\langle C_{i_2} \cdot C_{i_3} \cdots C_{i_m} \cdot C_a | x_1 \cap x_2 \rangle & \\
\langle C_{i_2} \cdot C_{i_3} \cdots C_{i_m} \cdot C_a | y \rangle & \quad \langle C_{j_1} \cdot C_{i_2} \cdots C_{i_m} | y \rangle
\end{align*}
\]
Classical Approximate Call Strings Approach

- Maintain call string suffixes of up to a given length $m$

$$\langle C_{i_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_1 \rangle \quad \langle C_{j_1} \cdot C_{i_2} \ldots C_{i_m} \mid x_2 \rangle$$

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

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

- Practical choices of $m$ have been 1 or 2
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$

$$\langle C_b \mid x_1 \rangle$$

Diagram:

- $C_a$
- $R_a$
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$

$\langle C_b \mid x_1 \rangle$

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

$R_a$
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$

![Diagram showing call strings and operations](attachment:image.png)
Approximate Call Strings in Presence of Recursion

• For simplicity, assume $m = 2$

\[
\langle C_b \mid x_1 \rangle \quad \langle C_b \cdot C_a \mid x_2 \rangle
\]

\[
\langle C_b \cdot C_a \mid x_1 \rangle, \quad \langle C_a \cdot C_a \mid x_2 \rangle
\]
Approximate Call Strings in Presence of Recursion

• For simplicity, assume $m = 2$

\[
\langle C_b \mid x_1 \rangle, \langle C_b \cdot C_a \mid x_2 \rangle, \langle C_a \cdot C_a \mid x_3 \rangle
\]

\[
\langle C_b \cdot C_a \mid x_1 \rangle, \langle C_a \cdot C_a \mid x_2 \rangle
\]

\[
R_a
\]
Approximate Call Strings in Presence of Recursion

- For simplicity, assume \( m = 2 \)

\[
\langle C_b \mid x_1 \rangle, \quad \langle C_b \cdot C_a \mid x_2 \rangle, \quad \langle C_a \cdot C_a \mid x_3 \rangle
\]

\[
\langle C_b \cdot C_a \mid x_1 \rangle, \quad \langle C_a \cdot C_a \mid x_2 \cap x_3 \rangle
\]

\[
R_a
\]
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$

\[
\langle C_b \mid x_1 \rangle \quad \langle C_b \cdot C_a \mid x_2 \rangle, \quad \langle C_a \cdot C_a \mid x_4 \rangle
\]

\[
\langle C_b \cdot C_a \mid x_1 \rangle, \quad \langle C_a \cdot C_a \mid x_2 \cap x_3 \rangle
\]

\[\boxed{C_a}\]

\[\boxed{R_a}\]
Approximate Call Strings in Presence of Recursion

• For simplicity, assume $m = 2$

\[
\langle C_b \mid x_1 \rangle, \quad \langle C_b \cdot C_a \mid x_2 \rangle, \quad \langle C_a \cdot C_a \mid x_4 \rangle
\]

\[
\langle C_b \cdot C_a \mid x_1 \rangle, \quad \langle C_a \cdot C_a \mid x_5 \rangle
\]

\[
R_a
\]
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$

\[
\langle C_b \mid x_1 \rangle \quad \langle C_b \cdot C_a \mid x_2 \rangle, \quad \langle C_a \cdot C_a \mid x_4 \rangle
\]

\[
\langle C_b \cdot C_a \mid x_1 \rangle, \quad \langle C_a \cdot C_a \mid x_5 \rangle
\]

\[
\langle C_b \cdot C_a \mid y_1 \rangle, \quad \langle C_a \cdot C_a \mid y_2 \rangle
\]
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$

\[ \langle C_b \mid x_1 \rangle, \langle C_b \cdot C_a \mid x_2 \rangle, \langle C_a \cdot C_a \mid x_4 \rangle \]

\[ C_a \]

\[ \langle C_b \cdot C_a \mid x_1 \rangle, \langle C_a \cdot C_a \mid x_5 \rangle \]

\[ \langle C_b \cdot C_a \mid y_1 \rangle, \langle C_a \cdot C_a \mid y_2 \rangle \]

\[ R_a \]

\[ \langle C_b \mid y_1 \rangle, \langle C_b \cdot C_a \mid y_2 \rangle, \langle C_a \cdot C_a \mid y_2 \rangle \]
Approximate Call Strings in Presence of Recursion

- For simplicity, assume $m = 2$

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

$$\langle C_a \rangle$$

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

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

$$\langle C_a \rangle$$

$$\langle C_b \mid y_1 \rangle, \langle C_b \cdot C_a \mid y_2 \rangle, \langle C_a \cdot C_a \mid y_2 \rangle$$
Part 5

IPDFA Using Value Contexts
Value Contexts: Key Ideas

Consider call chains $\sigma_1$ and $\sigma_2$ reaching $S_p$

- Data flow value invariant:
  If the data flow reaching $S_p$ along $\sigma_1$ and $\sigma_2$ are identical, then
Value Contexts: Key Ideas

Consider call chains $\sigma_1$ and $\sigma_2$ reaching $S_p$

- Data flow value invariant:
  
  If the data flow reaching $S_p$ along $\sigma_1$ and $\sigma_2$ 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 $\sigma_1$ and $\sigma_2$ reaching $S_p$

- Data flow value invariant:
  If the data flow reaching $S_p$ along $\sigma_1$ and $\sigma_2$ 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 \((\text{input } \mapsto \text{output})\) 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 encountered, the pairs are consulted to 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

\[
\sigma_0 / x_0 \quad \sigma_1 / x_1 \quad \sigma_2 / x_1 \quad \sigma_3 / x_2 \quad \sigma_4 / x_3
\]

- \( S_p \)
- \( C_i \)

Call q

- \( R_i \)
- \( E_p \)

- \( S_q \)
- \( E_q \)
Understanding Value Contexts

Separate contexts are created for each unique data flow value
Understanding Value Contexts

Call q

\[ \sigma_0 X_0 \quad \sigma_1 X_1 \quad \sigma_2 X_1 \quad \sigma_3 X_2 \quad \sigma_4 X_3 \]

\[ S_p \quad C_i \quad R_i \quad E_p \quad S_q \quad E_q \]
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

\[
\begin{align*}
S_0 \xrightarrow{\sigma_0} S_1 \xrightarrow{\sigma_1} S_2 \xrightarrow{\sigma_3} S_3 \xrightarrow{\sigma_4} S_4 \xrightarrow{S_0 C_i} S_5 \xrightarrow{S_1 C_i} S_6 \xrightarrow{S_2 C_i} S_7 \xrightarrow{S_3 C_i} S_8 \\
\end{align*}
\]
Understanding Value Contexts

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

Oct 2017 IIT Bombay
Understanding Value Contexts

Context transitions are consulted to transfer data flow values to calling contexts.
Understanding Value Contexts

- $S_0$ and $S_1$ connected by $C_i$.
- $S_2$ and $S_3$ connected by $C_i$.
- $S_4$, $S_5$, and $S_6$ connected by $C_i$.
- $S_p$, $S_q$, and $E_p$ connected by $C_i$.
- $S_0$ and $S_1$ connected by $R_i$.
- $S_2$ and $S_3$ connected by $R_i$.
- $S_4$, $S_5$, and $S_6$ connected by $R_i$.
- $S_0$ and $S_1$ connected by $E_p$.
- $S_2$ and $S_3$ connected by $E_p$.
- $S_4$, $S_5$, and $S_6$ connected by $E_p$.
Understanding Value Contexts

\begin{align*}
S_0 & \xrightarrow{\sigma_0} x_0 \\
S_1 & \xrightarrow{\sigma_1} x_1 \\
S_2 & \xrightarrow{\sigma_2} x_1 \\
S_3 & \xrightarrow{\sigma_3} x_2 \\
S_4 & \xrightarrow{S_0 C_i} x_0 \\
S_5 & \xrightarrow{S_1 C_i} x_1 \\
S_6 & \xrightarrow{S_3 C_i} x_3 \\
\end{align*}

\begin{align*}
S_0 & \xrightarrow{\sigma_0} y_0 \\
S_1 & \xrightarrow{\sigma_1} y_1 \\
S_2 & \xrightarrow{\sigma_2} y_1 \\
S_3 & \xrightarrow{\sigma_3} y_2 \\
S_4 & \xrightarrow{S_4} y_0 \\
S_5 & \xrightarrow{S_5} y_1 \\
S_6 & \xrightarrow{S_6} y_3 \\
\end{align*}

Oct 2017 IIT Bombay
Understanding Value Contexts

\[ s_0 \xrightarrow{x_0} \sigma_0 \]
\[ s_1 \xrightarrow{x_1} \sigma_1 \]
\[ s_2 \xrightarrow{x_2} \sigma_2 \]
\[ s_3 \xrightarrow{x_3} \sigma_3 \]

\[ s_4 \xrightarrow{x_0} s_0 \]
\[ s_5 \xrightarrow{x_1} s_1 \]
\[ s_6 \xrightarrow{x_3} s_3 \]

\[ s_0 \xrightarrow{c_i} s_4 \]
\[ s_1 \xrightarrow{c_i} s_5 \]
\[ s_2 \xrightarrow{c_i} s_6 \]
\[ s_3 \xrightarrow{c_i} s_6 \]

\[ s_0 \xrightarrow{y_0} s_4 \]
\[ s_1 \xrightarrow{y_1} s_5 \]
\[ s_2 \xrightarrow{y_1} s_6 \]
\[ s_3 \xrightarrow{y_3} s_6 \]

\[ \sigma_0 \xrightarrow{y_0} \]
\[ \sigma_1 \xrightarrow{y_1} \]
\[ \sigma_2 \xrightarrow{y_1} \]
\[ \sigma_3 \xrightarrow{y_1} \]
\[ \sigma_4 \xrightarrow{y_3} \]
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 $\mathit{VC} = \mathit{Procs} \times \mathit{L}$

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

• Supporting functions ($\mathit{CS}$ is the set of call sites)
  
  ▶ $\mathit{exitValue} : \mathit{VC} \mapsto \mathit{L}$
  eg. $\mathit{exitValue}(X) = v$

  ▶ $\mathit{transitions} : (\mathit{VC} \times \mathit{CS}) \mapsto \mathit{VC}$
  eg. $X \xrightarrow{\mathit{C}_i} Y$
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:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle p, v \rangle$</td>
<td>Context for procedure $p$ with data flow value $v$</td>
</tr>
<tr>
<td>$X</td>
<td>m$</td>
</tr>
<tr>
<td>$X.v$</td>
<td>Data flow value in context $X$ is $v$</td>
</tr>
<tr>
<td>$Out_m[X]$</td>
<td>Data flow value of context $X$ in $Out_m$</td>
</tr>
<tr>
<td>$X \xleftarrow{C_i} Y$</td>
<td>Transition from context $X$ to context $Y$ at call site $C_i$</td>
</tr>
</tbody>
</table>
Interprocedural Data Flow Analysis Using Value Contexts: An Overview

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

- If $n = C_i$ calling procedure $p$
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 $⟨p,v⟩$ exists (say $Y$) /* $p$ is the callee */
  - If it does not exist
Select $X \mid n$ 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 $X \xrightarrow{C_i} Y$
    - $Out_{C_i}[X] = Out_{C_i}[X] \cap exitValue(Y)$
    - if there is a change, add $X \mid m$, $\forall m \in succ(C_i)$ 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 $X \xrightarrow{C_i} Y$
  - 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 $X \xrightarrow{C_i} Y$
    - $Out_{C_i}[X] = Out_{C_i}[X] \sqcap \text{exitValue}(Y)$
    - if there is a change, add $X|m$, $\forall m \in \text{succ}(C_i)$ to $WL$
  - If it does not exist
    - create a new context $Y = \langle p,v \rangle$ /* $p$ is the callee */
    - initialize $\text{exitValue}(Y) = \top$
    - record the transition $X \xrightarrow{C_i} Y$
    - 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 $In_n$
Select $X|n$ from $WL$. Compute $In_n$. Let $X.v$ be in $In_n$

- If $n = E_p$

- For all other nodes
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 $ln_n$. Let $X.v$ be in $ln_n$

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

- For all other nodes
Select $X|n$ from $WL$. Compute $ln_n$. Let $X.v$ be in $ln_n$

- If $n = E_p$

- For all other nodes
  - Set $Out_n[X] = f_n(v)$
Select $X|n$ from $WL$. Compute $ln_n$. Let $X.v$ be in $ln_n$

- If $n = E_p$
  - Set $exitValue(X) = v$  
    /* $E_p$ is an empty block */
  - Find out all transitions $Z \xrightarrow{C_i} X$
    - Set $Out_{C_j}[Z] = Out_{C_j}[Z] \cap v$
    - 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

$S_{main}$

read $a, b$

$t := a \times b$

$C_1$

call $p$

$n_1$

print $a \times b$

$E_{main}$

$S_p$

if $a == 0$

$n_2$

$a = a - 1$

$C_2$

call $p$

$n_3$

$t = a \times b$

$E_p$
Available Expressions Analysis Using Value Contexts

\[ S_{main} \]
- read \( a, b \)
- \( t := a \times b \)

\[ C_1 \]
- call \( p \)

\[ n_1 \]
- print \( a \times b \)

\[ E_{main} \]

\[ S_p \]
- if \( a == 0 \)

\[ n_2 \]
- \( a = a - 1 \)

\[ C_2 \]
- call \( p \)

\[ n_3 \]
- \( t = a \times b \)

\[ E_p \]

**Is \( a \times b \) available?**
Available Expressions Analysis Using Value Contexts

\[ S_{main} \]
read \( a, b \)
\[ t := a \times b \]

\[ C_1 \]
call \( p \)

\[ n_1 \]
print \( a \times b \)

\[ E_{main} \]

Is \( a \times b \) available?

\[ S_p \]
if \( a == 0 \)

\[ n_2 \]
\( a = a - 1 \)

\[ C_2 \]
call \( p \)

\[ n_3 \]
\( t = a \times b \)

\[ E_p \]

int \( a, b, t; \)
void \( p() \)
{  if \( (a == 0) \)
  {  \( a = a-1; \)
      \( p(); \)
      \( t = a*b; \)
    }  
}
Available Expressions Analysis Using Value Contexts

\[ S_{main} \]
\[
\begin{align*}
\text{read } a, b \\
t &:= a \times b
\end{align*}
\]

\[ C_1 \]
\[
\text{call } p
\]

\[ n_1 \]
\[
\text{print } a \times b
\]

Is \( a \times b \) available?

\[ S_p \]
\[
\begin{align*}
\text{if } a &= 0 \\
a &= a - 1
\end{align*}
\]

\[ C_2 \]
\[
\text{call } p
\]

\[ n_3 \]
\[
t = a \times b
\]

\[ E_{main} \]

\[ E_p \]

\[
\text{int } a, b, t; \\
\text{void } p() \\
\{ 
\quad \text{if } (a == 0) \\
\quad \{ 
\quad \quad a = a - 1; \\
\quad \quad p(); \\
\quad \quad t = a \times b; \\
\quad \} \\
\}
\]
Available Expressions Analysis Using Value Contexts

\[ S_{\text{main}} \]
\[ \text{read } a, b \]
\[ t := a \times b \]
\[ C_1 \]
\[ \text{call } p \]
\[ n_1 \]
\[ \text{print } a \times b \]
\[ E_{\text{main}} \]

\[ S_p \]
\[ \text{if } a == 0 \]
\[ n_2 \]
\[ a = a - 1 \]
\[ C_2 \]
\[ \text{call } p \]
\[ n_3 \]
\[ t = a \times b \]
\[ E_p \]
Available Expressions Analysis Using Value Contexts

Create a new context $X_0$ with $Bl$ which is 0 for available expressions analysis.
Available Expressions Analysis Using Value Contexts

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

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

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

Compute the data flow values for \( S_m \) for context \( X_0 \)

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main},0 \rangle )</td>
<td>1</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

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

**Compute the data flow values for \( S_m \) for context \( X_0 \)**

It does not change
Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main}, 0 \rangle )</td>
<td>1</td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_0 = \langle \text{main,0} \rangle$</td>
<td>1</td>
</tr>
<tr>
<td>$X_1 = \langle p,1 \rangle$</td>
<td>1</td>
</tr>
</tbody>
</table>

Create a new context $X_1$ with entry value $1$

Record the transition to $X_1$

Initialize $\text{exitValue}(X_1)$ 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

\[ WL = [X_1 \mid S_p, X_1 \mid n_2, X_1 \mid C_2, X_1 \mid n_3, X_1 \mid E_p, X_0 \mid n_1, X_0 \mid E_m] \]

\[
\begin{align*}
X_0 & \xrightarrow{C_1} X_1 \\
X_{0.0} & \xrightarrow{S_{main}} \text{read } a, b \\
 & \quad t := a \ast b \\
X_{0.1} & \xrightarrow{C_1} \text{call } p \\
& \quad \text{print } a \ast b \\
\end{align*}
\]

\[
\begin{align*}
X_{1.1} & \xrightarrow{S_p} \text{if } a \equiv 0 \\
& \quad a = a - 1 \\
& \quad \text{call } p \\
\end{align*}
\]

\[
\begin{align*}
n_2 & \quad a = a - 1 \\
& \quad \text{call } p \\
n_3 & \quad t = a \ast b \\
\end{align*}
\]

\[
\begin{array}{c|c}
\text{Context} & \text{exitValue} \\
\hline
X_0 = \langle \text{main}, 0 \rangle & 1 \\
X_1 = \langle p, 1 \rangle & 1 \\
\end{array}
\]
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

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

\[ X_0 \xrightarrow{C_1} X_1 \]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main}, 0 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle p, 1 \rangle )</td>
<td>1</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

\[ WL = [X_1 \mid n_2, X_1 \mid C_2, X_1 \mid n_3, X_1 \mid E_p, X_0 \mid n_1, X_0 \mid E_m] \]

Context | exitValue
--- | ---
\( X_0 = \langle \text{main,0} \rangle \) | 1
\( X_1 = \langle \text{p,1} \rangle \) | 1
Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_0 = \langle \text{main},0 \rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_1 = \langle p,1 \rangle)</td>
<td>1</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_0 = \langle \text{main}, 0 \rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_1 = \langle p, 1 \rangle)</td>
<td>1</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

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

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

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main,0} \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle \text{p,1} \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 = \langle \text{p,0} \rangle )</td>
<td>1</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_0 = \langle \text{main},0 \rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_1 = \langle p,1 \rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_2 = \langle p,0 \rangle)</td>
<td>1</td>
</tr>
</tbody>
</table>
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
--- | ---
$X_0 = \langle\text{main}, 0\rangle$ | 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**
---|---
\( X_0 = \langle \text{main}, 0 \rangle \) | 1
\( X_1 = \langle p, 1 \rangle \) | 1
\( X_2 = \langle p, 0 \rangle \) | 1

**Diagrams:**
- **Diagram 1:**
  - **Start:** \( S_{\text{main}} \)
  - **Steps:**
    - \( X_0 \) : read \( a, b \), \( t := a \times b \)
    - \( X_0 \rightarrow C_1 \)
    - \( C_1 \) : call \( p \)
    - \( n_1 \) : print \( a \times b \)

- **Diagram 2:**
  - **Start:** \( S_p \)
  - **Steps:**
    - \( X_1 \) : if \( a == 0 \)
    - \( X_1 \rightarrow X_2 \)
    - \( X_1 \rightarrow X_2 \)
    - \( n_2 \) : \( a = a - 1 \)
    - \( X_1 \rightarrow X_2 \)
    - \( C_2 \) : call \( p \)
    - \( n_3 \) : \( t = a \times b \)
Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_0)</td>
<td>1</td>
</tr>
<tr>
<td>(X_1)</td>
<td>1</td>
</tr>
<tr>
<td>(X_2)</td>
<td>1</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

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

- \( S_{main} \):
  - \( X_0.0 \): read \( a, b \)
  - \( X_0.1 \): \( t := a \times b \)
  - \( C_1 \):
    - call \( p \)
  - \( n_1 \): print \( a \times b \)

- \( S_p \):
  - \( X_1.1 \): if \( a == 0 \)
    - \( n_2 \): \( a = a - 1 \)
    - \( C_2 \):
      - call \( p \)
  - \( n_3 \): \( t = a \times b \)

- Context:
  - \( X_0 = \langle \text{main}, 0 \rangle \) exitValue 1
  - \( X_1 = \langle p, 1 \rangle \) exitValue 1
  - \( X_2 = \langle p, 0 \rangle \) exitValue 1

- \( p \) has context \( X_2 \) with value 0 so no need to create a new context.
Available Expressions Analysis Using Value Contexts

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

\[
\begin{array}{cccc}
  X_0 & C_1 & X_1 & C_2 \\
\end{array}
\]

\[
C_2
\]

\[
X_0.0
\]

\[
S_{main}
\]

\[
read\ a,\ b \\
t := a \ast b
\]

\[
X_0.1
\]

\[
C_1
\]

\[
call\ p
\]

\[
n_1
\]

\[
print\ a \ast b
\]

\[
E_{main}
\]

\[
X_1.0
\]

\[
X_1.1
\]

\[
S_p
\]

\[
if\ a == 0
\]

\[
X_2.0
\]

\[
n_2
\]

\[
a = a - 1
\]

\[
X_1.0
\]

\[
X_1.1
\]

\[
C_2
\]

\[
call\ p
\]

\[
n_3
\]

\[
t = a \ast b
\]

\[
E_p
\]

\[
X_0 = \langle main, 0 \rangle
\]

\[
X_1 = \langle p, 1 \rangle
\]

\[
X_2 = \langle p, 0 \rangle
\]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_0</td>
<td>1</td>
</tr>
<tr>
<td>X_1</td>
<td>1</td>
</tr>
<tr>
<td>X_2</td>
<td>1</td>
</tr>
</tbody>
</table>

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

Record the transition from context \(X_2\) to itself.
Available Expressions Analysis Using Value Contexts

\[ \text{WL} = [X_2|C_2, X_2|n_3, X_2|E_p, X_1|n_3, X_1|E_p, X_0|n_1, X_0|E_m] \]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_0 = \langle \text{main}, 0 \rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_1 = \langle p, 1 \rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_2 = \langle p, 0 \rangle)</td>
<td>1</td>
</tr>
</tbody>
</table>

\(p\) 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 \(\text{exitValue}(X_2)\) to compute \(\text{Out}_{C_2}[X_2]\).
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
---|---
\( X_0 = \langle \text{main},0 \rangle \) | 1
\( X_1 = \langle p,1 \rangle \) | 1
\( X_2 = \langle p,0 \rangle \) | 1
Available Expressions Analysis Using Value Contexts

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

\[ X_0 = \langle \text{main}, 0 \rangle \]

\[ X_1 = \langle p, 1 \rangle \]

\[ X_2 = \langle p, 0 \rangle \]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_0</td>
<td>1</td>
</tr>
<tr>
<td>X_1</td>
<td>1</td>
</tr>
<tr>
<td>X_2</td>
<td>1</td>
</tr>
</tbody>
</table>

Oct 2017 IIT Bombay
Available Expressions Analysis Using Value Contexts

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

![Diagram of Available Expressions Analysis Using Value Contexts]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_0 = \langle \text{main}, 0 \rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_1 = \langle p, 1 \rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_2 = \langle p, 0 \rangle)</td>
<td>1</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

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

\begin{itemize}
  \item \( X_0 = \langle \text{main}, 0 \rangle \)
  \item \( X_1 = \langle p, 1 \rangle \)
  \item \( X_2 = \langle p, 0 \rangle \)
\end{itemize}

At \( E_p \) the values from \( S_p \) and \( n_3 \) are merged for context \( X_2 \).
Available Expressions Analysis Using Value Contexts

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

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

\( \text{exitValue}(X_2) \) is set to 0

---

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main}, 0 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle p, 1 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 = \langle p, 0 \rangle )</td>
<td>0</td>
</tr>
</tbody>
</table>
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
\( X_1 = \langle p,1 \rangle \) | 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

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

At \( E_p \) the values from \( S_p \) and \( n_3 \) are merged for context \( X_2 \)
\( \text{exitValue}(X_2) \) 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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main},0 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle p,1 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 = \langle p,0 \rangle )</td>
<td>0</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

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

There is no change in \( \text{Out}_{n_3}[X_2] \) (because it was initialized to \( \top \))
Available Expressions Analysis Using Value Contexts

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

**Contexts and Exit Values**

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main}, 0 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle \text{p}, 1 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 = \langle \text{p}, 0 \rangle )</td>
<td>0</td>
</tr>
</tbody>
</table>

There is no change in \( \text{Out}_{n_3}[X_2] \) (because it was initialized to \( \top \))
Available Expressions Analysis Using Value Contexts

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

\[
\begin{array}{ccc}
X_0 & \xrightarrow{C_1} & X_1 \\
& & \xrightarrow{C_2} & X_2
\end{array}
\]

**Context**

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main},0 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle p,1 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 = \langle p,0 \rangle )</td>
<td>0</td>
</tr>
</tbody>
</table>

There is no change in \( \text{Out}_{n_3}[X_1] \) either.
Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main}, 0 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle p, 1 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 = \langle p, 0 \rangle )</td>
<td>0</td>
</tr>
</tbody>
</table>

There is no change in \( Out_{n_3}[X_1] \) either.
Available Expressions Analysis Using Value Contexts

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

At \( E_p \) the values from \( S_p \) and \( n_3 \) are merged for context \( X_1 \)
Available Expressions Analysis Using Value Contexts

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

\[ \begin{align*}
X_0 & \xrightarrow{C_1} X_1 \\
X_1 & \xrightarrow{C_2} X_2 \\
X_2 & \xrightarrow{C_2} X_1
\end{align*} \]

Context | exitValue
--- | ---
\(X_0 = \langle \text{main}, 0 \rangle\) | 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

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

\[ \begin{align*}
X_0 \xrightarrow{C_1} X_1 & \xrightarrow{C_2} X_2 \\
X_0.0 & \quad X_1.1 \quad X_2.0 \\
S_{\text{main}} & \text{read } a, b \\
t & := a \ast b
\end{align*} \]

\[ \begin{align*}
S_p & \text{if } a == 0 \\
X_1.1 & \quad X_2.0
\end{align*} \]

\[ \begin{align*}
n_2 & \text{a = } a - 1 \\
X_1.0 & \quad X_2.0
\end{align*} \]

\[ \begin{align*}
n_3 & \text{t = a } \ast b \\
X_1.0 & \quad X_2.0
\end{align*} \]

At \( E_p \) the values from \( S_p \) and \( n_3 \) are merged for context \( X_1 \)
\( \text{exitValue}(X_1) \) remains 1
Since \( X_1 \) has transition \( X_0 \xrightarrow{C_1} X_1 \), \( \text{Out}_{C_1}[X_0] \) becomes 1

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main}, 0 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle p, 1 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 = \langle p, 0 \rangle )</td>
<td>0</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

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

\[
\begin{align*}
X_0.0 & \quad \text{read } a, b \\
X_0.1 & \quad t := a \times b \\
C_1 & \quad \text{call } p \\
X_0.1 & \quad \text{print } a \times b \\
E_{main} & \quad \text{ } \\
X_1.0 & \quad S_p \quad \text{if } a == 0 \\
X_1.1 & \quad a = a - 1 \\
X_2.0 & \quad \text{call } p \\
X_2.0 & \quad n_2 \\
X_2.0 & \quad X_2.0 \\
X_1.1 & \quad t = a \times b \\
X_2.1 & \quad \text{call } p \\
X_2.0 & \quad n_3 \\
X_2.0 & \quad X_2.0 \\
X_1.1 & \quad X_1.1 \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_0 = \langle \text{main},0\rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_1 = \langle \text{p},1\rangle)</td>
<td>1</td>
</tr>
<tr>
<td>(X_2 = \langle \text{p},0\rangle)</td>
<td>0</td>
</tr>
</tbody>
</table>

Oct 2017 IIT Bombay
Available Expressions Analysis Using Value Contexts

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

**Contexts and Exit Values**

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 = \langle \text{main}, 0 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_1 = \langle p, 1 \rangle )</td>
<td>1</td>
</tr>
<tr>
<td>( X_2 = \langle p, 0 \rangle )</td>
<td>0</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

\[ WL = [X_0|E_m] \]

\[ \begin{array}{c|c}
\text{Context} & \text{exitValue} \\
\hline
X_0 = \langle \text{main}, 0 \rangle & 1 \\
X_1 = \langle p, 1 \rangle & 1 \\
X_2 = \langle p, 0 \rangle & 0 \\
\end{array} \]
Available Expressions Analysis Using Value Contexts

\[
WL = [X_0|E_m]
\]

- **Contexts**:
  - \(X_0 = \langle \text{main,0} \rangle\)
  - \(X_1 = \langle p,1 \rangle\)
  - \(X_2 = \langle p,0 \rangle\)

- **Exit Values**:
  - \(1\)
  - \(0\)

**Code Snippet**:

```plaintext
main:
    read a, b
    t := a * b
    print a * b

sp:
    if a == 0
        a = a - 1
    n2

C1:
    call p
    n1

C2:
    call p
    n3
    t = a * b
```

**Diagrams**:

- A graph showing the flow of contexts and value expressions.
- Nodes represent states, and arrows represent transitions.
- Contexts are labeled with state transitions, and value expressions are shown within nodes.
Available Expressions Analysis Using Value Contexts

\[ WL = [X_0 | E_m] \]

\[
\begin{align*}
X_0.0 & \quad S_{main} \\
X_1.1 & \quad n_2 \quad a = a - 1 \\
X_2.0 & \\
E_{main} & \quad X_0.1 \\
X_2.0 & \quad n_3 \quad t = a \times b \\
X_1.1 & \quad C_2 \\
X_2.1 & \\
E_p & \quad X_1.1 \\
X_0.1 & \quad X_2.0 \\
X_2.0 & \quad X_1.1
\end{align*}
\]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_0) = \langle main, 0 \rangle</td>
<td>1</td>
</tr>
<tr>
<td>(X_1) = \langle p, 1 \rangle</td>
<td>1</td>
</tr>
<tr>
<td>(X_2) = \langle p, 0 \rangle</td>
<td>0</td>
</tr>
</tbody>
</table>
Available Expressions Analysis Using Value Contexts

\[ WL = [ ] \]

\[ X_0 \xrightarrow{C_1} X_1 \xrightarrow{C_2} X_2 \]

\[ S_{\text{main}} \]

read \( a, b \)
\[ t := a \times b \]

\[ X_0.0 \]

\[ X_0.1 \]

\[ C_1 \]

\[ \text{call } p \]

\[ X_0.1 \]

\[ n_1 \]

\[ \text{print } a \times b \]

\[ E_{\text{main}} \]

\[ X_0.1 \]

\[ X_{1.0} \]

\[ C_2 \]

\[ \text{call } p \]

\[ X_{1.0} \]

\[ X_{2.0} \]

\[ n_2 \]

\[ a = a - 1 \]

\[ X_{1.1} \]

\[ X_{2.0} \]

\[ X_{1.0} \]

\[ n_3 \]

\[ t = a \times b \]

\[ X_{2.0} \]

\[ X_{1.1} \]

\[ X_{2.1} \]

Context | exitValue
---|---
\( X_0 = \langle \text{main}, 0 \rangle \) | 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)

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Work List</th>
<th>Sel. node</th>
<th>Data flow value</th>
<th>New context</th>
<th>New trans.</th>
<th>exit value</th>
<th>Addition to the work list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>$X_0 = \langle m,0 \rangle$</td>
<td></td>
<td>$X_0.1$</td>
<td>$X_0</td>
</tr>
<tr>
<td>2</td>
<td>$X_0</td>
<td>S_m, X_0</td>
<td>C_1, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
<td>$S_m$</td>
<td>$Out_{S_m}[X_0] = 1$</td>
</tr>
<tr>
<td>3</td>
<td>$X_0</td>
<td>C_1, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
<td>$C_1$</td>
<td></td>
<td>$X_1 = \langle p,1 \rangle$</td>
</tr>
<tr>
<td>4</td>
<td>$X_1</td>
<td>S_p, X_1</td>
<td>n_2, X_1</td>
<td>C_2, X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
</tr>
<tr>
<td>5</td>
<td>$X_1</td>
<td>n_2, X_1</td>
<td>C_2, X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
</tr>
<tr>
<td>6</td>
<td>$X_1</td>
<td>C_2, X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>7</td>
<td>$X_2</td>
<td>S_p, X_2</td>
<td>n_2, X_2</td>
<td>C_2, X_2</td>
<td>n_3, X_2</td>
<td>E_p, X_1</td>
<td>n_3, X_1</td>
</tr>
</tbody>
</table>
## A Trace of Value Context Based Analysis (2)

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Work List</th>
<th>Sel. node</th>
<th>Data flow value</th>
<th>New context</th>
<th>New trans.</th>
<th>exit value</th>
<th>Addition to the work list</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$X_2</td>
<td>n_2, X_2</td>
<td>C_2, X_2</td>
<td>n_3, X_2</td>
<td>E_p, X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
</tr>
<tr>
<td>9</td>
<td>$X_2</td>
<td>C_2, X_2</td>
<td>n_3, X_2</td>
<td>E_p, X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
</tr>
<tr>
<td>10</td>
<td>$X_2</td>
<td>n_3, X_2</td>
<td>E_p, X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
</tr>
<tr>
<td>11</td>
<td>$X_2</td>
<td>E_p, X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
<td>$E_p$</td>
</tr>
<tr>
<td>12</td>
<td>$X_2</td>
<td>n_3, X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
<td>$n_3$</td>
</tr>
<tr>
<td>13</td>
<td>$X_1</td>
<td>n_3, X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
<td>$n_3$</td>
<td>$Out_{n_3}[X_1]=1$</td>
</tr>
<tr>
<td>14</td>
<td>$X_1</td>
<td>E_p, X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
<td>$E_p$</td>
<td>$Out_{E_p}[X_1]=1$</td>
<td>$Out_{C_1}[X_0]=1$</td>
</tr>
<tr>
<td>15</td>
<td>$X_0</td>
<td>n_1, X_0</td>
<td>E_m$</td>
<td>$n_1$</td>
<td>$Out_{n_1}[X_0]=1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>$X_0</td>
<td>E_m$</td>
<td>$E_m$</td>
<td>$Out_{E_m}[X_0]=1$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Merging ExitValue with Previous Out Value at the Call Site

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

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

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)
This example illustrates non-termination of analysis if the \textit{exitValue} is not merged with the previous \textit{Out} value.

We assume that procedure main calls procedure \textit{p} (and not \textit{q}) and the expression \(a \times b\) is partially available on entry to \textit{p}.
Partially Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p, 1 \rangle$</td>
<td>0</td>
</tr>
</tbody>
</table>

Oct 2017 IIT Bombay
Partially Available Expressions Analysis Using Value Contexts

\[
X_1 = \langle p, 1 \rangle
\]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_1)</td>
<td>0</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

Value 1 reaches \(q\) and a new context must be created for it.
Partially Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q,1 \rangle$</td>
<td>0</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

$X_1 \xrightarrow{C_2} X_2$

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q, 1 \rangle$</td>
<td>0</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

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

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p, 0 \rangle$</td>
<td>0</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

X₁ \xrightarrow{C_2} X₂ \xrightarrow{C_3} X₃

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁ = ⟨p,1⟩</td>
<td>0</td>
</tr>
<tr>
<td>X₂ = ⟨q,1⟩</td>
<td>0</td>
</tr>
<tr>
<td>X₃ = ⟨p,0⟩</td>
<td>0</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

And now the value 0 reaches q at call site $C_2$.
Partially Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p,0 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_4 = \langle q,0 \rangle$</td>
<td>0</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p, 0 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_4 = \langle q, 0 \rangle$</td>
<td>0</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

Context | exitValue
--- | ---
X₁ = ⟨p,1⟩ | 0
X₂ = ⟨q,1⟩ | 0
X₃ = ⟨p,0⟩ | 0
X₄ = ⟨q,0⟩ | 0

And now the value 0 reaches p at call site C₃
Partially Available Expressions Analysis Using Value Contexts

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

\[
\begin{align*}
X_1 &\xrightarrow{C_2} X_2 \\
X_2 &\xrightarrow{C_3} X_3 \\
X_3 &\xrightarrow{C_2} X_4 \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_1 = \langle p, 1 \rangle)</td>
<td>0</td>
</tr>
<tr>
<td>(X_2 = \langle q, 1 \rangle)</td>
<td>0</td>
</tr>
<tr>
<td>(X_3 = \langle p, 0 \rangle)</td>
<td>0</td>
</tr>
<tr>
<td>(X_4 = \langle q, 0 \rangle)</td>
<td>0</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

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 $X_3 \stackrel{C_2}{\rightarrow} X_4$).
Partially Available Expressions Analysis Using Value Contexts

The analysis of $q$ for $X_4$ is now and the \textit{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 $X_3 \overset{C_2}{\rightarrow} X_4$).

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p,0 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_4 = \langle q,0 \rangle$</td>
<td>1</td>
</tr>
</tbody>
</table>
**Partially Available Expressions Analysis Using Value Contexts**

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

- **Diagram**
  - $X_1 \xrightarrow{C_2} X_2 \xrightarrow{C_3} X_3 \xrightarrow{C_2} X_4$
  - $S_p$  
  - $S_q$

- **Expressions**
  - $E_p$  
  - $E_q$

- **Value Contexts**
  - $X_{1.1}$  
  - $X_{1.1}$  
  - $X_{2.1}$  
  - $X_{2.1}$

- **Operations**
  - $a = 5$
  - $a \ast b$

- **Outcomes**
  - $Out_{C_2}$ 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

\[
\begin{array}{c}
\text{Context} & \text{exitValue} \\
X_1 = \langle p, 1 \rangle & 0 \\
X_2 = \langle q, 1 \rangle & 0 \\
X_3 = \langle p, 0 \rangle & 0 \\
X_4 = \langle q, 0 \rangle & 1 \\
\end{array}
\]

Out_{C_2} becomes 1 for \( X_3 \) which changes the value at \( \text{In}_{n_2} \) for \( X_3 \) to 1.
Partially Available Expressions Analysis Using Value Contexts

CS 618 Interprocedural DFA: IPDFA Using Value Contexts

Oct 2017

IIT Bombay
Partially Available Expressions Analysis Using Value Contexts

\[
\begin{array}{c}
\text{Context} & \text{exitValue} \\
X_1 = \langle p, 1 \rangle & 0 \\
X_2 = \langle q, 1 \rangle & 0 \\
X_3 = \langle p, 0 \rangle & 0 \\
X_4 = \langle q, 0 \rangle & 1 \\
\end{array}
\]

\text{In } C_2 \text{ becomes 1 for } X_3

Since we have a context for } q \text{ with entry value 1 (} X_2 \text{), we remove the transition } X_3 \xrightarrow{C_2} X_4 \text{ and add the transition } X_3 \xrightarrow{C_2} X_2
Partially Available Expressions Analysis Using Value Contexts

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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p, 0 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_4 = \langle q, 0 \rangle$</td>
<td>1</td>
</tr>
</tbody>
</table>
### Partially Available Expressions Analysis Using Value Contexts

#### Context Table

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p, 0 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_4 = \langle q, 0 \rangle$</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Example

The value of $X_3$ in $In_{n_2}$ once again becomes 0.
Partially Available Expressions Analysis Using Value Contexts

The value of $X_3$ in $\text{In}_{C_2}$ once again becomes 0.
The transition from $X_3$ needs to be restored to $X_3 \xrightarrow{C_2} X_4$ removing the transition $X_3 \xrightarrow{C_2} X_2$.

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p,0 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_4 = \langle q,0 \rangle$</td>
<td>1</td>
</tr>
</tbody>
</table>
Partially Available Expressions Analysis Using Value Contexts

**Context Table:**

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q, 1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p, 0 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_4 = \langle q, 0 \rangle$</td>
<td>1</td>
</tr>
</tbody>
</table>

The value of $X_3$ in $\text{In}_{C_2}$ once again becomes 0.
The transition from $X_3$ needs to be restored to $X_3 \xrightarrow{C_2} X_4$ removing the transition $X_3 \xrightarrow{C_2} X_2$. 

The value of $X_3$ in $\text{In}_{C_2}$ once again becomes 0.
The transition from $X_3$ needs to be restored to $X_3 \xrightarrow{C_2} X_4$ removing the transition $X_3 \xrightarrow{C_2} X_2$. 

**Diagram:**

- **$X_1$** connected to $X_2$ via $C_2$.
- **$X_2$** connected to $X_3$ via $C_3$.
- **$X_3$** connected to $X_4$ via $C_2$.
- **$X_1$** connected to $X_2$ via $C_2$.
- **$X_2$** connected to $X_3$ via $C_3$.
- **$X_3$** connected to $X_4$ via $C_2$.

**Context Nodes:**

- $S_p$: $X_1.1$
- $S_q$: $X_2.1$, $X_4.0$
- $E_p$: $X_1.1$
- $E_q$: $X_4.1$
- $C_2$: Call $q$
- $C_3$: Call $p$
- $n_2$: $X_2.0$, $X_4.0$
- $n_3$: $a \ast b$
- $a = 5$
- $a \ast b$

**Transitions:**

- $X_1 \xrightarrow{C_2} X_2$
- $X_2 \xrightarrow{C_3} X_3$
- $X_3 \xrightarrow{C_2} X_4$
- $X_1 \xrightarrow{C_2} X_2$
- $X_2 \xrightarrow{C_3} X_3$
- $X_3 \xrightarrow{C_2} X_4$
- $X_1 \xrightarrow{C_2} X_2$
- $X_2 \xrightarrow{C_3} X_3$
- $X_3 \xrightarrow{C_2} X_4$
Partially Available Expressions Analysis Using Value Contexts

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.

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1 = \langle p,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_2 = \langle q,1 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_3 = \langle p,0 \rangle$</td>
<td>0</td>
</tr>
<tr>
<td>$X_4 = \langle q,0 \rangle$</td>
<td>1</td>
</tr>
</tbody>
</table>
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.
Defining Value Context Method Using Data Flow Equations

- The overall data flow values $\Gamma$ 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

$$\Gamma_1 \uplus \Gamma_2 = \{ X.w \mid X.u \in \Gamma_1 \land X.v \in \Gamma_2 \Rightarrow w = u \sqcap v, X.u \in \Gamma_1 \land X.v \notin \Gamma_2 \Rightarrow w = u, X.u \notin \Gamma_1 \land X.v \in \Gamma_2 \Rightarrow w = v \}$$

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

- Data flow variables for node $n$ in procedure $p$ are $ln(p, n)$ 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
  
  $context(p, v)$ 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 $exitValue(X)$ returns the exit value of context $X$

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

- Function $gpred$ extends the predecessor relation $\text{pred}$ (which is local to a procedure) to a global level across procedures

  $$gpred(p, n) = \begin{cases} 
  \{(q, m) \mid \text{call site } m \text{ in } q \text{ calls } p\} & n \text{ is } S_p \\
  \{(p, m) \mid m \in \text{pred}(n)\} & \text{otherwise}
  \end{cases}$$
Defining Value Context Method Using Data Flow Equations

We define data flow equations for a forward data flow analysis

\[
\text{In}(p, n) = \begin{cases} 
\{ X.v \mid X = \text{context}(p, v), Y.v \in \text{In}(q, m), \quad n \text{ is } S_p \\
(q, m) \in \text{gpred}(p, n) \} \\
\cup \quad \text{Out}(p, m) \\n(p,m)\in\text{gpred}(p,n)
\end{cases}
\]

\[
\text{Out}(p, n) = \begin{cases} 
\text{Out}(p, n) \cup \{ X.v \mid X.v' \in \text{In}(p, m), \quad n \text{ calls } q \\
Y = \text{context}(q, v'), \\
v = \text{exitValue}(Y) \} \\
\{ X.v \mid X.v' \in \text{In}(p, m), v = f(p, n)(v') \} \\n\end{cases}
\]

Otherwise
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

<table>
<thead>
<tr>
<th>Context</th>
<th>exitValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₀ : ⟨main, 0⟩</td>
<td>1</td>
</tr>
<tr>
<td>X₁ : ⟨p, 0⟩</td>
<td>1</td>
</tr>
<tr>
<td>X₂ : ⟨p, 1⟩</td>
<td>1</td>
</tr>
</tbody>
</table>
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 $C_2$

$S_m \rightarrow C_1 \rightarrow E_m$
Innermost Recursion Along the Call at $C_2$
Innermost Recursion Along the Call at $C_2$

![Diagram showing the innermost recursion process.]

- $S_p \rightarrow C_2 \rightarrow n_2 \rightarrow E_p$
- $S_p \rightarrow C_3$
- $S_m \rightarrow C_1 \rightarrow E_m$
- $n_1 \rightarrow E_p$

Oct 2017
Innermost Recursion Along the Call at $C_2$
Innermost Recursion Along the Call at $C_2$
Innermost Recursion Along the Call at $C_2$

$E_p$ is 0
Innermost Recursion Along the Call at \( C_2 \)

\[
\begin{align*}
S_p &\rightarrow C_2 \rightarrow n_1 \rightarrow E_p \\
S_p &\rightarrow C_3 \rightarrow n_2 \rightarrow E_p \\
S_p &\rightarrow C_2 \rightarrow X_1 \rightarrow n_1 \rightarrow E_p(X_1.0) \\
S_m &\rightarrow C_1 \rightarrow X_0 \rightarrow E_m(X_0.0)
\end{align*}
\]

\( Bl\is 0 \)
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
```

```
(X_1.0) \rightarrow S_p \rightarrow X_1 \rightarrow E_p(X_1.0)

S_p \rightarrow C_2 \rightarrow X_2 \rightarrow n_1 \rightarrow E_p(X_2.1)

S_p \rightarrow C_3 \rightarrow X_1 \rightarrow n_2 \rightarrow E_p(X_1.0)

S_p \rightarrow C_2 \rightarrow X_1 \rightarrow n_1 \rightarrow E_p(X_1.0)

S_m \rightarrow C_1 \rightarrow X_0 \rightarrow E_m(X_0.0)
```
Innermost Recursion Along the Call at $C_2$

```
\begin{align*}
(X_{1.0}) \, S_p \rightarrow& \quad X_1 \rightarrow E_p(X_{1.0}) \\
(X_{2.0}) \, S_p \rightarrow& \quad C_2 \rightarrow X_2 \rightarrow n_1 \rightarrow E_p(X_{2.1}) \\
S_p \rightarrow& \quad C_3 \rightarrow X_1 \rightarrow n_2 \rightarrow E_p(X_{1.0}) \\
S_p \rightarrow& \quad C_2 \rightarrow X_1 \rightarrow n_1 \rightarrow E_p(X_{1.0}) \\
S_m \rightarrow& \quad C_1 \rightarrow X_0 \rightarrow E_m(X_{0.0})
\end{align*}
```
Innermost Recursion Along the Call at $C_2$

- $(X_1.0) S_p \xrightarrow{} X_1 \xrightarrow{} E_p (X_1.0)$
- $(X_2.0) S_p \xrightarrow{} C_2 \xrightarrow{} X_2 \xrightarrow{} n_1 \xrightarrow{} E_p (X_2.1)$
- $(X_1.0) S_p \xrightarrow{} C_3 \xrightarrow{} X_1 \xrightarrow{} n_2 \xrightarrow{} E_p (X_1.0)$
- $S_p \xrightarrow{} C_2 \xrightarrow{} X_1 \xrightarrow{} n_1 \xrightarrow{} E_p (X_1.0)$
- $S_m \xrightarrow{} C_1 \xrightarrow{} X_0 \xrightarrow{} E_m (X_0.0)$

exitValue of $X_1$ remains 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

Diagram:

- $(X_1.0) \xrightarrow{S} X_1 \xrightarrow{} E_p(X_1.0)$
- $(X_2.0) \xrightarrow{S} C_2 \xrightarrow{} X_2 \xrightarrow{n_1} E_p(X_2.1)$
- $(X_1.0) \xrightarrow{S} C_3 \xrightarrow{} X_1 \xrightarrow{n_2} E_p(X_1.0)$
- $(X_1.0) \xrightarrow{S} C_2 \xrightarrow{} X_1 \xrightarrow{n_1} E_p(X_1.0)$
- $(X_0.0) \xrightarrow{S} C_1 \xrightarrow{} X_0 \xrightarrow{} E_m(X_0.0)$
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 $C_3$

$S_m \rightarrow C_1 \rightarrow \rightarrow \rightarrow E_m$
Innermost Recursion Along the Call at $C_3$
Innermost Recursion Along the Call at $C_3$
Innermost Recursion Along the Call at $C_3$
Innermost Recursion Along the Call at $C_3$
Innermost Recursion Along the Call at $C_3$
Innermost Recursion Along the Call at $C_3$

$B$ is 0
Innermost Recursion Along the Call at $C_3$

$n_1$ kills the liveness of $a$
New context is not required

Diagram:

- $S_p \rightarrow C_3 \rightarrow n_2 \rightarrow E_p$
- $S_p \rightarrow C_2 \rightarrow n_1 \rightarrow E_p$
- $S_p \rightarrow C_3 \rightarrow n_2 \rightarrow E_p$
- $S_p \rightarrow C_2 \rightarrow X_1 \rightarrow n_1 \rightarrow E_p(X_1.0)$
- $S_m \rightarrow C_1 \rightarrow X_0 \rightarrow E_m(X_0.0)$
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 $C_3$

$n_2$ 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

\[ S_p \rightarrow C_3 \quad \ldots \quad X_1 \quad \ldots \quad n_2 \rightarrow E_p(X_1.0) \]

\[ S_p \rightarrow C_2 \quad \ldots \quad X_2 \quad \ldots \quad n_1 \rightarrow E_p(X_2.1) \]

\[ S_p \rightarrow C_3 \quad \ldots \quad X_1 \quad \ldots \quad n_2 \rightarrow E_p(X_1.0) \]

\[ S_p \rightarrow C_2 \quad \ldots \quad X_1 \quad \ldots \quad n_1 \rightarrow E_p(X_1.0) \]

\[ S_m \rightarrow C_1 \quad \ldots \quad X_0 \quad \ldots \quad E_m(X_0.0) \]
Innermost Recursion Along the Call at $C_3$

$n_2$ generates the liveness of a New context is not required
Innermost Recursion Along the Call at $C_3$

exitValue of $X_2$ is 1 (after merging with previous value 0)
Innermost Recursion Along the Call at $C_3$

$exitValue$ of $X_1$ is 1 (after merging with previous value 0)
Innermost Recursion Along the Call at $C_3$

exitValue of $X_2$ remains 1
Innermost Recursion Along the Call at $C_3$

exitValue of $X_1$ remains 1
Innermost Recursion Along the Call at $C_3$

$(X_{2.1}) \quad S_p \rightarrow X_2 \rightarrow E_p (X_{2.1})$

$(X_{1.1}) \quad S_p \rightarrow C_3 \rightarrow X_1 \rightarrow n_2 \rightarrow E_p (X_{1.0})$

$(X_{2.1}) \quad S_p \rightarrow C_2 \rightarrow X_2 \rightarrow n_1 \rightarrow E_p (X_{2.1})$

$(X_{1.1}) \quad S_p \rightarrow C_3 \rightarrow X_1 \rightarrow n_2 \rightarrow E_p (X_{1.0})$

$(X_{1.1}) \quad S_p \rightarrow C_2 \rightarrow X_1 \rightarrow n_1 \rightarrow E_p (X_{1.0})$

$S_m \rightarrow C_1 \rightarrow X_0 \rightarrow E_m (X_{0.0})$

exitValue of $X_1$ remains 1
Innermost Recursion Along the Call at $C_3$

exitValue of $X_0$ is 1 (after merging with previous value 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).
Tutorial Problem #1 for Value Contexts

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

- `S_{main}`
- `n_1` (c = a * b)
- `C_1` (Call p)
- `E_{main}`

- `S_p`
- `n_2` (a = a * b)
- `C_2` (Call p)
- `E_p`
### Tutorial Problem #2 for Value Contexts

Perform interprocedural live variables analysis using value contexts

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

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

Observe the change in edges in the transition diagram
Tutorial Problem #3 for Value Contexts

Perform interprocedural available expressions analysis using value contexts

```c
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

```plaintext
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. }
```
Tutorial Problem #5 for Value Contexts

Perform interprocedural live variables analysis using value contexts

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

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

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

Context sensitivity: e is live on entry to p but not before its call in main
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;
    }
}
## Reaching Definitions Analysis in GCC 4.0

<table>
<thead>
<tr>
<th>Program</th>
<th>LoC</th>
<th>#F</th>
<th>#C</th>
<th>3K length bound</th>
<th>Proposed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>K</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#CS</td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#CS</td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>hanoi</td>
<td>33</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>100000+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99922</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>3973 × 10³</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Proposed Approach</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>bit_gray</td>
<td>53</td>
<td>5</td>
<td>11</td>
<td>7</td>
<td>100000+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31374</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>2705 × 10³</td>
</tr>
<tr>
<td>analyzer</td>
<td>288</td>
<td>14</td>
<td>20</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>20.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Proposed Approach</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>distray</td>
<td>331</td>
<td>9</td>
<td>21</td>
<td>6</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>322.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Proposed Approach</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>mason</td>
<td>350</td>
<td>9</td>
<td>13</td>
<td>8</td>
<td>100000+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22143</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>432 × 10³</td>
</tr>
<tr>
<td>fourinarow</td>
<td>676</td>
<td>17</td>
<td>45</td>
<td>5</td>
<td>510</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>158</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>397.76</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Proposed Approach</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>sim</td>
<td>1146</td>
<td>13</td>
<td>45</td>
<td>8</td>
<td>100000+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>33546</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>1427 × 10³</td>
</tr>
<tr>
<td>181_mcf</td>
<td>1299</td>
<td>17</td>
<td>24</td>
<td>6</td>
<td>32789</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32767</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>484 × 10³</td>
</tr>
<tr>
<td>256_bzip2</td>
<td>3320</td>
<td>63</td>
<td>198</td>
<td>7</td>
<td>492</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>63</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3K length bound</td>
<td>258.33</td>
</tr>
</tbody>
</table>

- 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.

\[
\# \text{ distinct tagged values} = \min \left( \# \text{ actual contexts}, \# \text{ actual data flow values} \right)
\]