Precise and Scalable Program Analysis

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

Feb 2022

Outline

- Some meanderings in precise and scalable analysis
- Intraprocedural analysis
- Interprocedural analysis
- Conclusions

Acknowledgements

The following people have contributed to the work presented in these slides

Alan Mycroft,	Prashant Singh Rawat,
Amey Karkare,	Pritam Gharat,
Amitabha Sanyal,	Rasesh Tongia,
Anshuman Dhuliya,	Rohan Padhye,
Bageshri Sathe,	Supratik Chakraborty
Komal Pathade,	Swati Jaiswal,
Mehul Jain,	Vini Kanvar,

. . . and many more have contributed indirectly

Some Meanderings

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

Interprocedural Analysis

Conclusions

References

Pointer Analysis Musings

- A keynote address:

  "The worst thing that has happened to Computer Science is C, because it brought pointers with it ..."

  - Frances Allen, IITK Workshop (2007)

- A couple of influential papers
  - Which Pointer Analysis should I Use?

    Michael Hind and Anthony Pioli. ISTAA 2000
  - Pointer Analysis: Haven't we solved this problem yet ?

    Michael Hind PASTE 2001

Pointer Analysis Musings

- A keynote address:

  > "The worst thing that has happened to Computer Science is C, because it brought pointers with it . . ."
  >
  > - Frances Allen, IITK Workshop (2007)

- A couple of influential papers
  - Which Pointer Analysis should I Use?

    Michael Hind and Anthony Pioli. ISTAA 2000
  - Pointer Analysis: Haven't we solved this problem yet ?

    Michael Hind PASTE 2001

Pointer Analysis Musings

- A keynote address:

  "The worst thing that has happened to Computer Science is C, because it brought pointers with it ..."

  - Frances Allen, IITK Workshop (2007)

- A couple of influential papers
  - Which Pointer Analysis should I Use?

    Michael Hind and Anthony Pioli. ISTAA 2000
  - Pointer Analysis: Haven't we solved this problem yet ?

    Michael Hind PASTE 2001
  - 2022 ..

The Mathematics of Pointer Analysis

In the most general situation

- Alias analysis is undecidable.

  Landi-Ryder [POPL 1991], Landi [LOPLAS 1992],
  Ramalingam [TOPLAS 1994]

- Flow insensitive alias analysis is NP-hard

  Horwitz [TOPLAS 1997]

- Points-to analysis is undecidable

  Chakravarty [POPL 2003]

Adjust your expectations suitably to avoid disappointments!

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

Interprocedural Analysis

Conclusions

References

So what should we expect?

To quote Hind [PASTE 2001]

So what should we expect?

To quote Hind [PASTE 2001]

- "Fortunately many approximations exist"

So what should we expect?

To quote Hind [PASTE 2001]

- "Fortunately many approximations exist"
- "Unfortunately too many approximations exist!"

So what should we expect?

To quote Hind [PASTE 2001]

- "Fortunately many approximations exist"
- "Unfortunately too many approximations exist!"

Engineering of pointer analysis is much more dominant than its science

Pointer Analysis: Engineering or Science?

- Engineering view
  - Build quick approximations
  - The tyranny of (exclusive) OR
    Precision OR Efficiency?
- Science view
  - Build clean abstractions
  - Can we harness the Genius of AND?
    Precision AND Efficiency?

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

Interprocedural Analysis

Conclusions

References

Pointer Analysis: Engineering or Science?

- Engineering view
  - Build quick approximations
  - The tyranny of (exclusive) OR
    Precision OR Efficiency?
- Science view
  - Build clean abstractions
  - Can we harness the Genius of AND?
    Precision AND Efficiency?

- Most common trend as evidenced by publications
  - Build acceptable approximations guided by empirical observations
  - The notion of acceptability is often constrained by beliefs rather than possibilities

Abstraction Vs. Approximation in Static Analysis

- Static analysis needs to create abstract values that represent many concrete values
- Mapping concrete values to abstract values
  - *Abstraction*.
    Deciding which properties of the concrete values are essential *What*
    Ease of understanding, reasoning, modelling etc. *Why*
  - *Approximation*.
    Deciding which properties of the concrete values cannot be represented accurately and should be summarized *What*
    Decidability, tractability, or efficiency and scalability *Why*

Abstraction Vs Approximation

Capture the clouds in this picture for study

Need to meet some resource constraints

Cannot represent the entire picture accurately

Abstraction Vs Approximation

Capture the clouds in this picture for study

Need to meet some resource constraints

Cannot represent the entire picture accurately

Use approximation and meet resource constraints

Usually easy and scalable, but imprecise

Some times, too imprecise to be of any use

Abstraction Vs Approximation

Capture the clouds in this picture for study

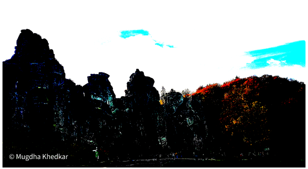
Need to meet some resource constraints

Cannot represent the entire picture accurately

Use approximation and meet resource constraints

Usually easy and scalable, but imprecise

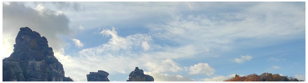
Some times, too imprecise to be of any use



Use abstraction and meet resource constraints

Usually difficult, need to dig deeper to define exactly what is needed and what can be thrown away

However, it can be precise and scalable



Abstraction Vs. Approximation in Static Analysis

- Abstractions
  - focus on precision and conciseness of modelling
  - tell us what we can ignore without being imprecise
- Approximations
  - focus on efficiency and scalability
  - tell us the imprecision that we have to tolerate

Abstraction Vs. Approximation in Static Analysis

- Abstractions
  - focus on precision and conciseness of modelling
  - tell us what we can ignore without being imprecise
- Approximations
  - focus on efficiency and scalability
  - tell us the imprecision that we have to tolerate
- Our Holy Grail

  Build clean abstractions before surrendering to the approximations

Program Analysis: Precision versus Scalability

- Ideally, an analysis should be
  - Sound
  - Precise
  - Scalable

Program Analysis: Precision versus Scalability

- Ideally, an analysis should be
  - Sound
  - Precise
  - Scalable

Common belief

- Precision and scalability cannot be achieved together for exhaustive analysis

Common Practice

- Trade off precision using approximations

Program Analysis: Precision versus Scalability

- Ideally, an analysis should be
  - Sound
  - Precise
  - Scalable

- The main factors enhancing the precision of an exhaustive (as against a demand-driven) analysis are
  - Flow sensitivity
  - Context sensitivity
  - Field sensitivity
  - Precise heap abstraction
  - Precise call graph

Program Analysis: Precision versus Scalability

- Ideally, an analysis should be
  - Sound
  - Precise
  - Scalable

- The main factors enhancing the precision of an exhaustive (as against a demand-driven) analysis are
  - Flow sensitivity
  - Context sensitivity
  - Field sensitivity
  - Precise heap abstraction
  - Precise call graph

Compromises on these lead to approximations

Demand-Driven Analysis Vs. Exhaustive Analysis

- Exhaustive. Compute all possible information
- Demand-Driven. Compute only the requested information (by a client)

  Different from incremental analysis which also computes only some information but it updates the earlier computed solution

The Classical Precision-Efficiency Dilemma

Abstraction	Role in precision Distinguishes between	Cause of non-scalability Needs to consider
Flow sensitivity		
Context sensitivity		
Field sensitivity		
Precise heap abstraction		
Precise call structure		

The Classical Precision-Efficiency Dilemma

Abstraction	Role in precision Distinguishes between	Cause of non-scalability Needs to consider
Flow sensitivity	Information at different program points	
Context sensitivity	Information in different contexts	
Field sensitivity	Different fields of an object	
Precise heap abstraction	Different heap locations	
Precise call structure	Indirect calls made to different callees from the same program point	

The Classical Precision-Efficiency Dilemma

Abstraction	Role in precision Distinguishes between	Cause of non-scalability Needs to consider
Flow sensitivity	Information at different program points	A large number of program points
Context sensitivity	Information in different contexts	Exponentially large number of contexts
Field sensitivity	Different fields of an object	Pointees along different fields
Precise heap abstraction	Different heap locations	Unbounded number of heap locations
Precise call structure	Indirect calls made to different callees from the same program point	Precise points-to information

The First Order Effects of Approximations

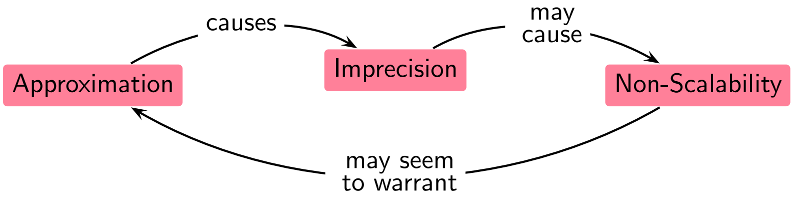
Approximation	Admits
Flow insensitivity	
Context insensitivity (or partial context sensitivity)	
Field insensitivity	
Imprecision in call graphs	
Allocation site based heap abstraction	

The First Order Effects of Approximations

Approximation	Admits
Flow insensitivity	Spurious intraprocedural paths
Context insensitivity (or partial context sensitivity)	Spurious interprocedural paths
Field insensitivity	Spurious paths in the memory graph
Imprecision in call graphs	Spurious call sequences
Allocation site based heap abstraction	Spurious paths in the memory graph

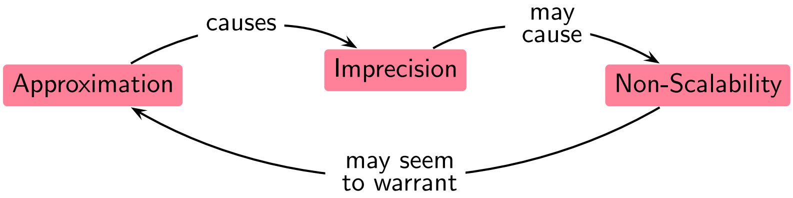
The Second Order Effect of Approximations

- Approximations may create a vicious cycle



The Second Order Effect of Approximations

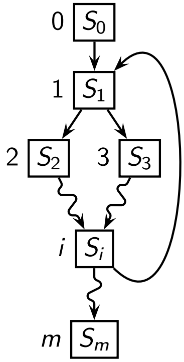
- Approximations may create a vicious cycle



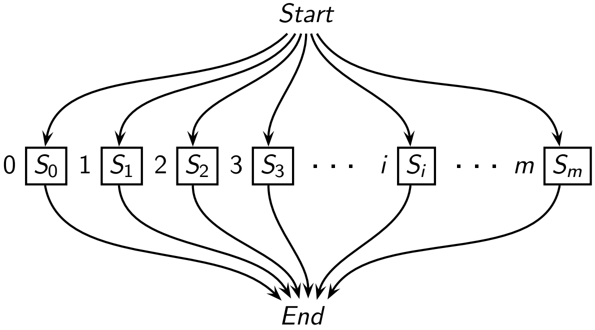
- Two examples of non-scalability cause by approximations
  - *k*-limited call strings may create "butterfly cycles" causing spurious fixed point computations [Hakjoo, 2010]
  - Imprecision in function pointer analysis overapproximates calls may create spurious recursion in call graphs

Flow Sensitivity Vs. Flow Insensitivity

Flow Sensitive

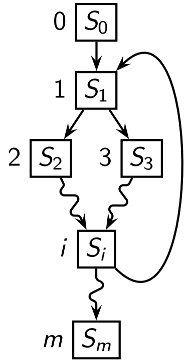


Flow Insensitive

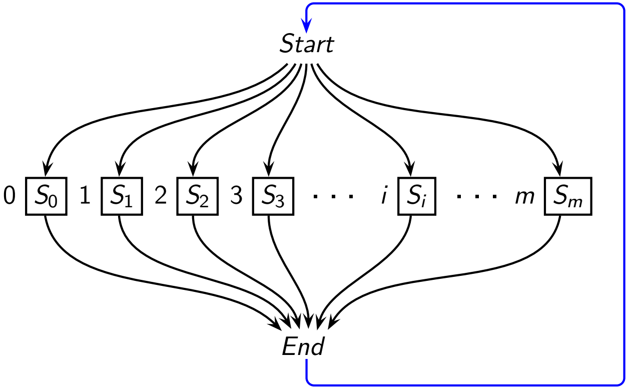


Flow Sensitivity Vs. Flow Insensitivity

Flow Sensitive



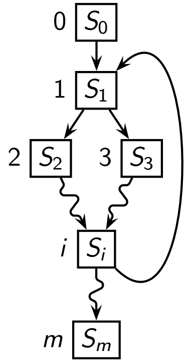
Flow Insensitive



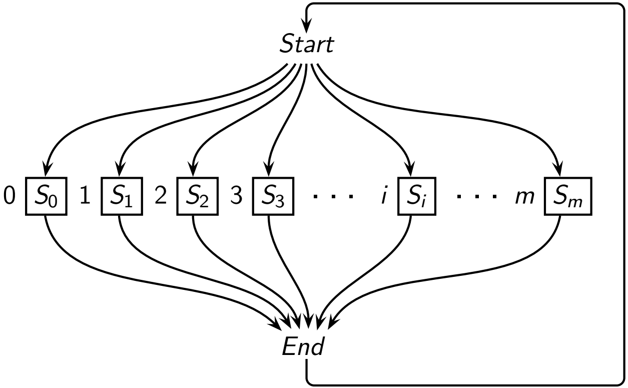
Assumption: Statements can be executed in any order

Flow Sensitivity Vs. Flow Insensitivity

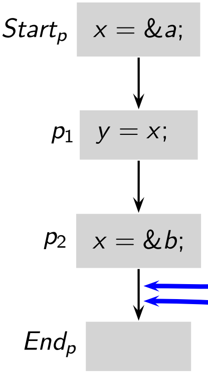
Flow Sensitive



Flow Insensitive



Flow Sensitivity Vs. Flow Insensitivity



Flow-insensitive analysis is less precise than a flow-sensitive analysis

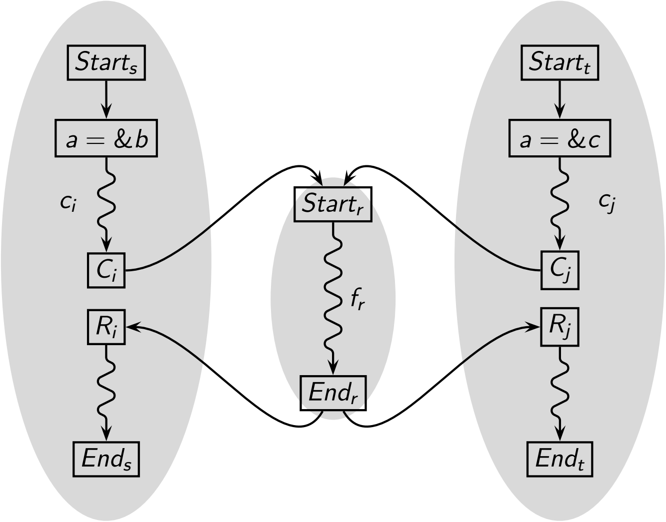
Flow-insensitive points-to information

$$\{x \rightarrow a, x \rightarrow b, y \rightarrow a, y \rightarrow b\}$$

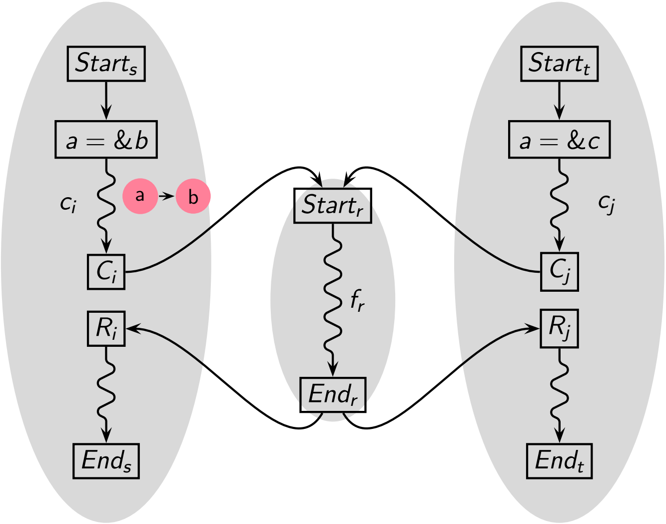
Flow-sensitive points-to information

$$\{x \rightarrow b, y \rightarrow a\}$$

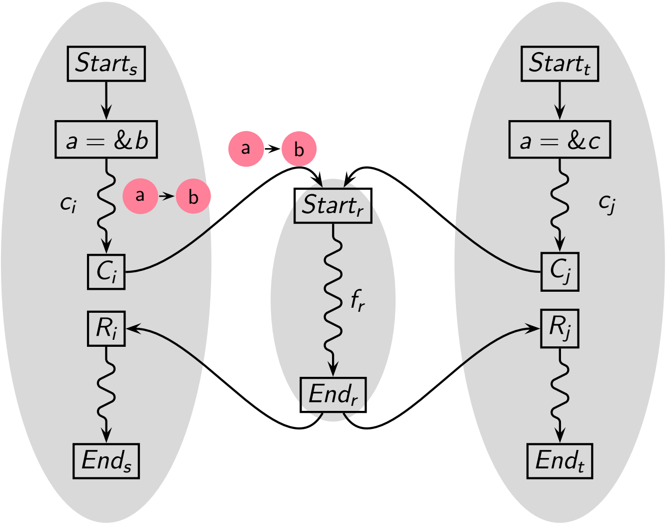
Context Sensitivity Vs. Context Insensitivity



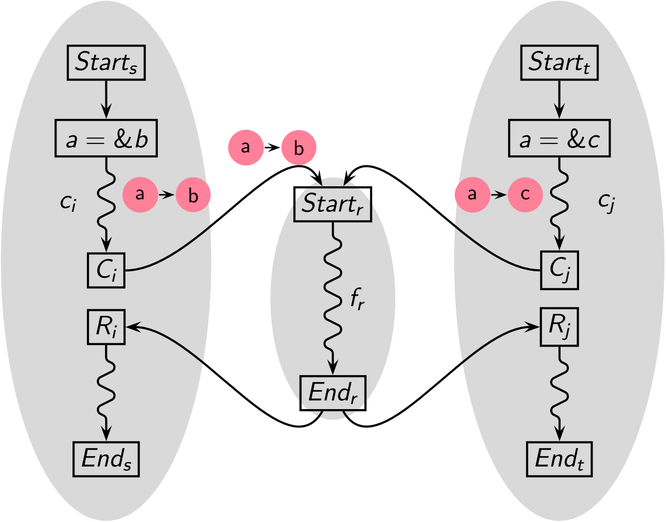
Context Sensitivity Vs. Context Insensitivity



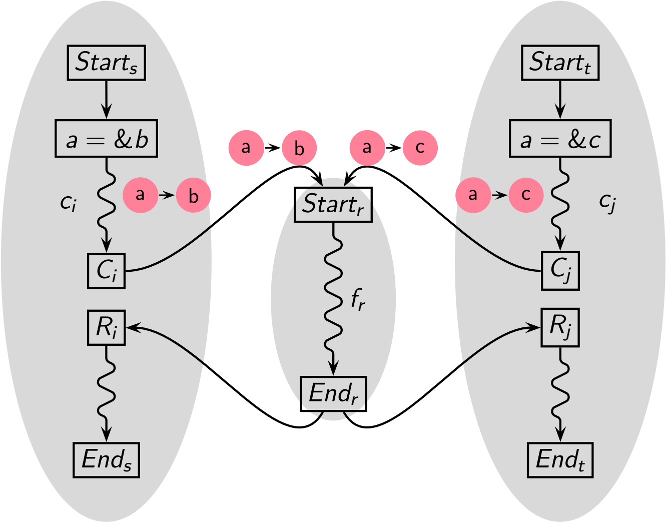
Context Sensitivity Vs. Context Insensitivity



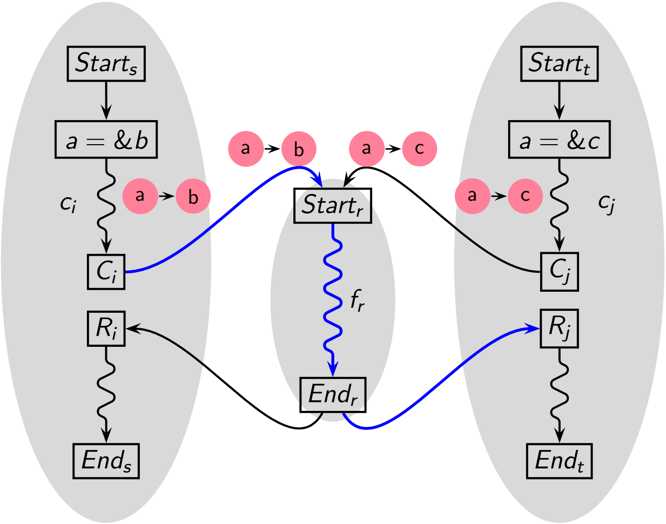
Context Sensitivity Vs. Context Insensitivity



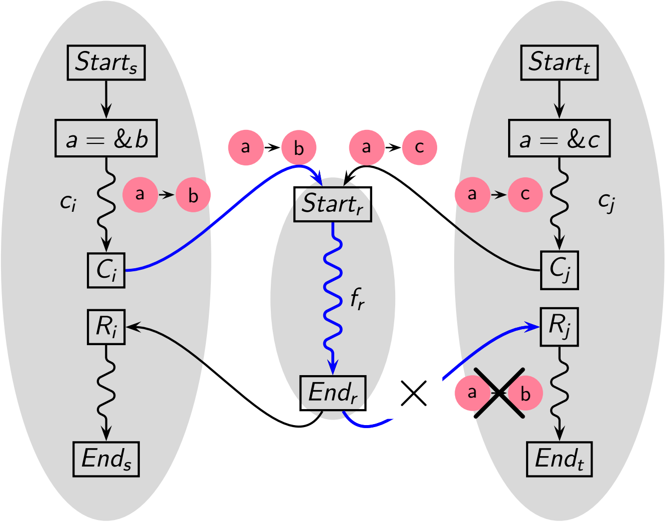
Context Sensitivity Vs. Context Insensitivity



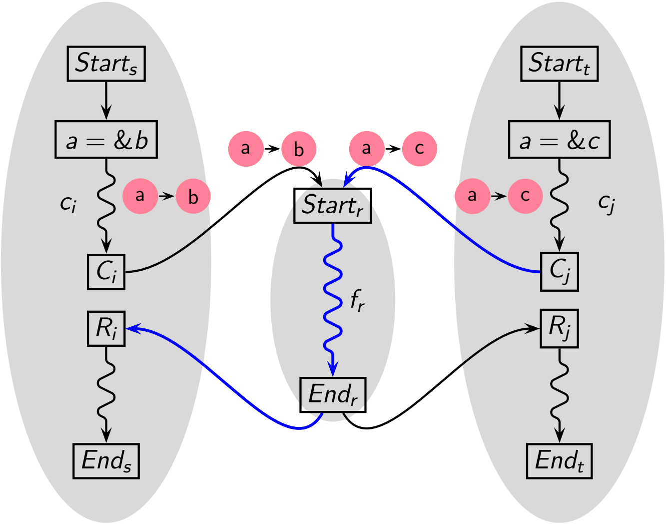
Context Sensitivity Vs. Context Insensitivity



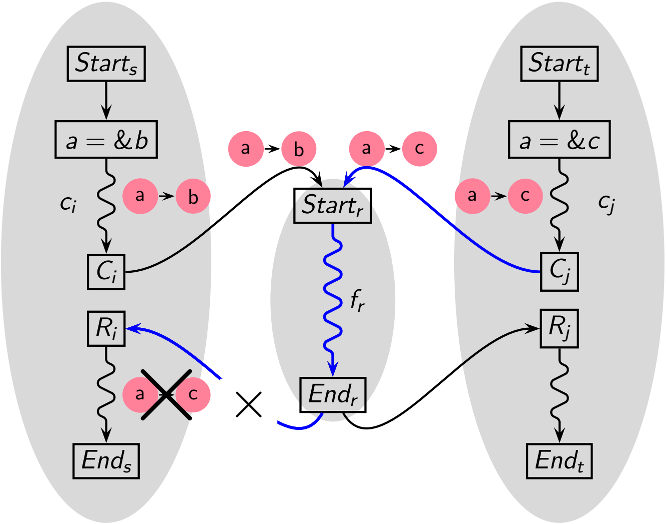
Context Sensitivity Vs. Context Insensitivity



Context Sensitivity Vs. Context Insensitivity



Context Sensitivity Vs. Context Insensitivity

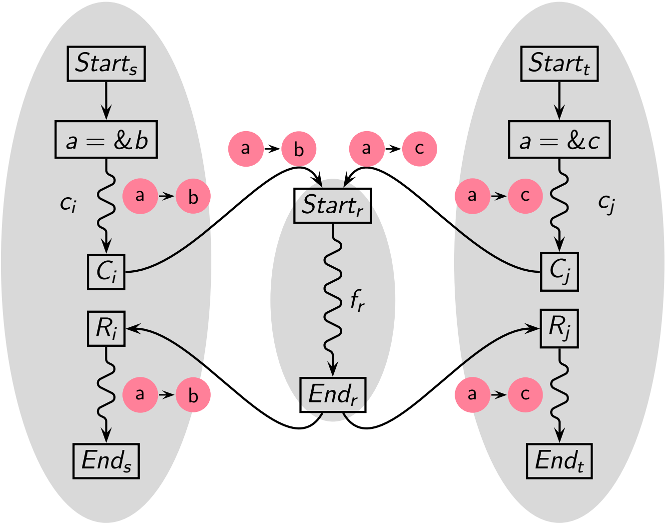


Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

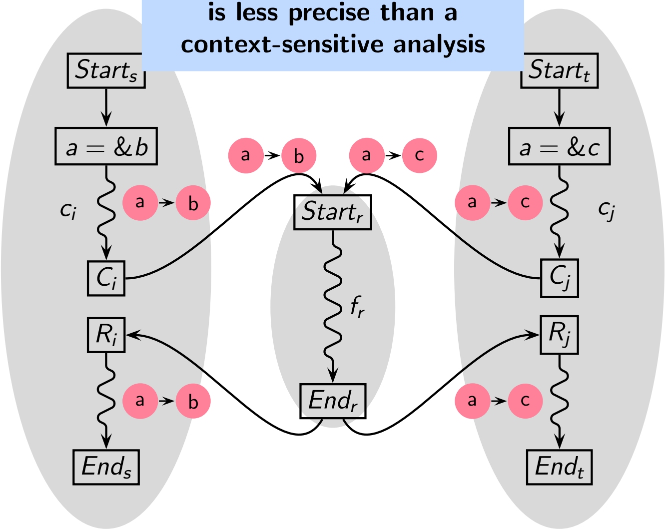
Topic:
Some Meanderings

Context Sensitivity Vs. Context Insensitivity

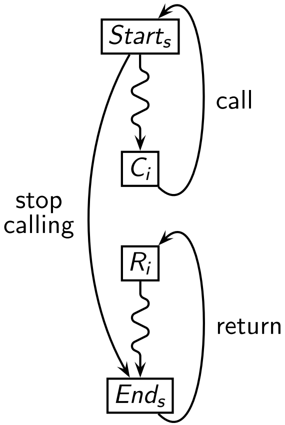


Context Sensitivity Vs. Context Insensitivity

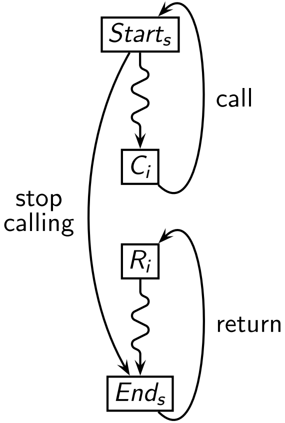
Context-insensitive analysis is less precise than a context-sensitive analysis



Context Sensitivity in the Presence of Recursion



Context Sensitivity in the Presence of Recursion



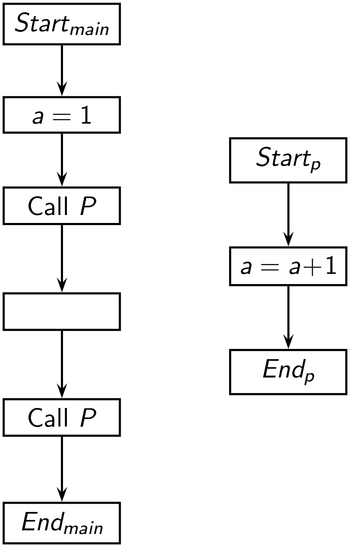
- Paths from $Start_s$ to End_s should constitute a context free language

$$\text{call}^n \cdot \text{stop} \cdot \text{return}^n$$

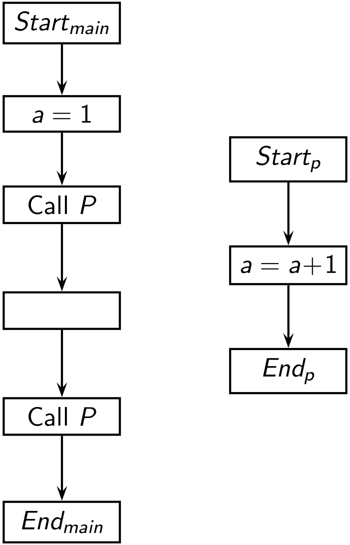
- If we treat cycle of recursion as an SCC
  - Calls and returns become jumps, and
  - paths are approximated by a regular language

$$\text{call}^* \cdot \text{stop} \cdot \text{return}^*$$

Context Insensitivity = Imprecision + Potential Non-Scalability

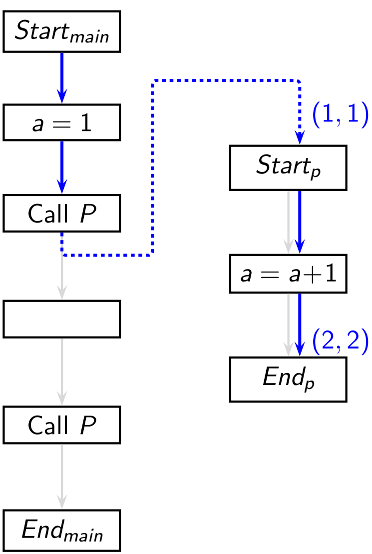


Context Insensitivity = Imprecision + Potential Non-Scalability



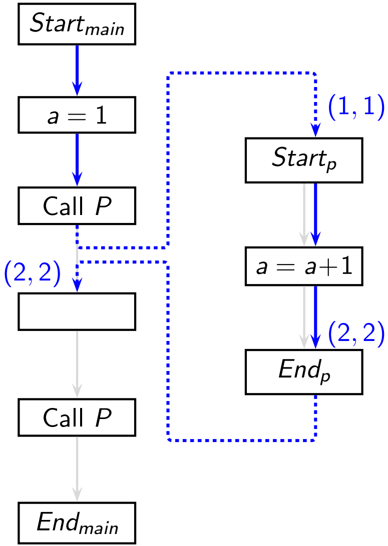
- What is the value range of a?

Context Insensitivity = Imprecision + Potential Non-Scalability



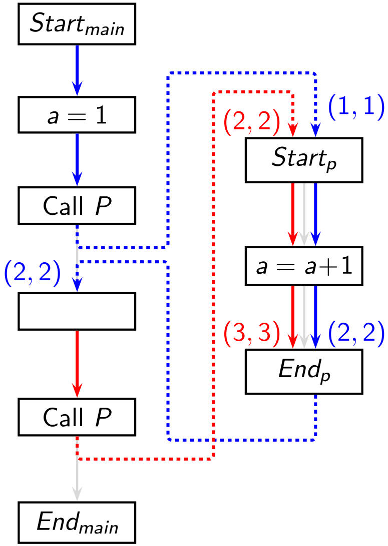
- What is the value range of a?

Context Insensitivity = Imprecision + Potential Non-Scalability



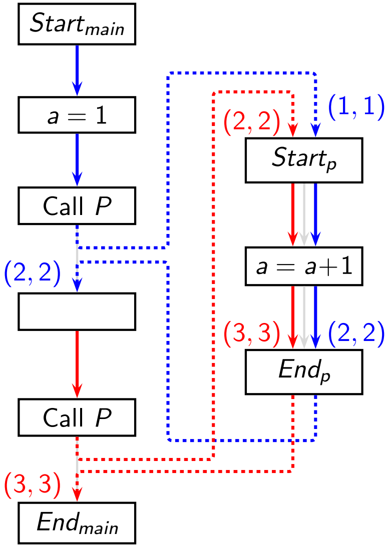
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P

Context Insensitivity = Imprecision + Potential Non-Scalability



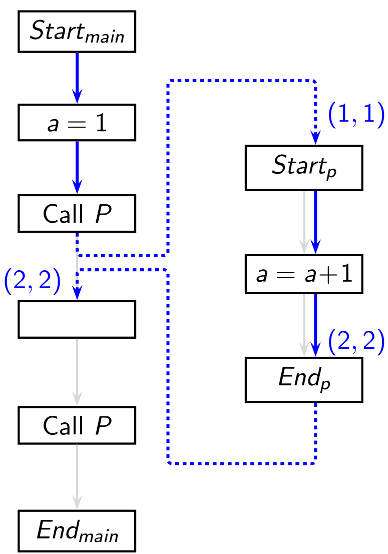
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P

Context Insensitivity = Imprecision + Potential Non-Scalability



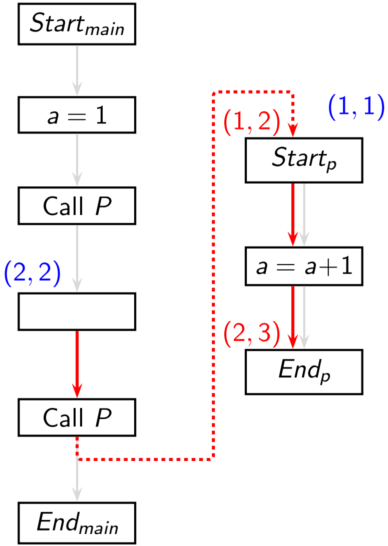
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3, 3)$

Context Insensitivity = Imprecision + Potential Non-Scalability



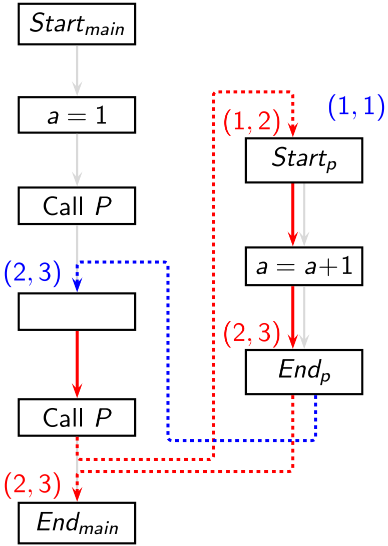
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3,3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller

Context Insensitivity = Imprecision + Potential Non-Scalability



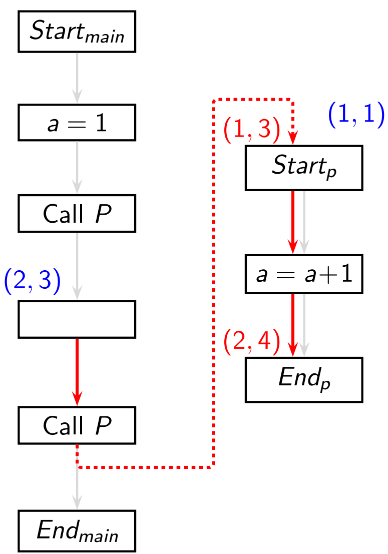
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3,3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller

Context Insensitivity = Imprecision + Potential Non-Scalability



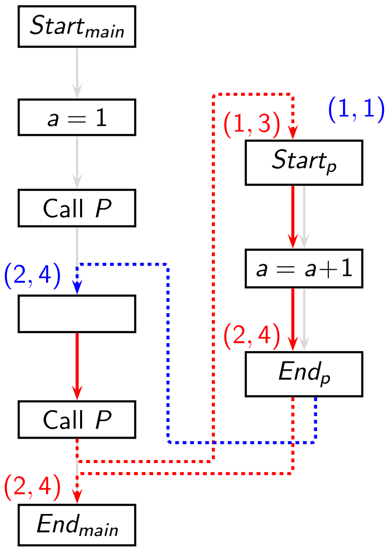
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3,3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller

Context Insensitivity = Imprecision + Potential Non-Scalability



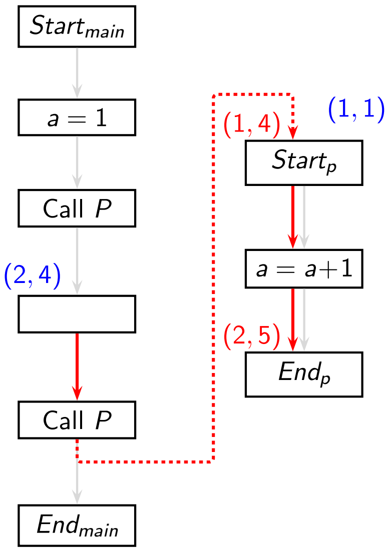
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3,3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller

Context Insensitivity = Imprecision + Potential Non-Scalability



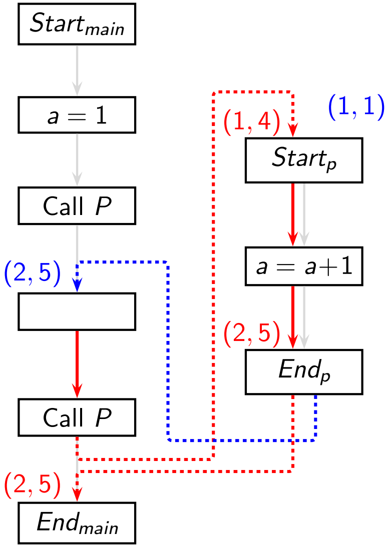
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3,3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller

Context Insensitivity = Imprecision + Potential Non-Scalability



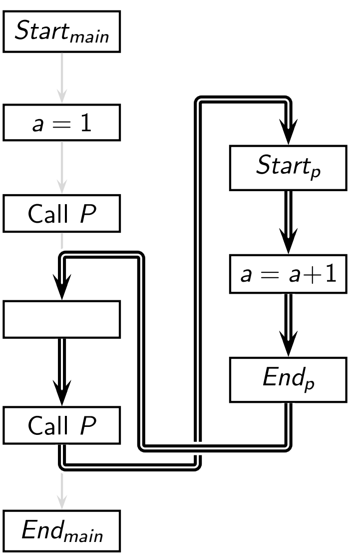
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3, 3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller

Context Insensitivity = Imprecision + Potential Non-Scalability



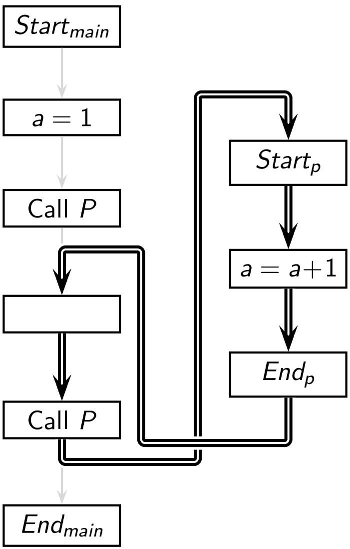
- What is the value range of a?
- Context sensitive analysis
    - Data flow value propagated back to the current caller of P
    - Range of a at End_{main} is $(3, 3)$
- Context insensitive analysis
    - Data flow value propagated back to every caller
    - Range of a at End_{main} is $(2, \ldots)$

Context Insensitivity = Imprecision + Potential Non-Scalability



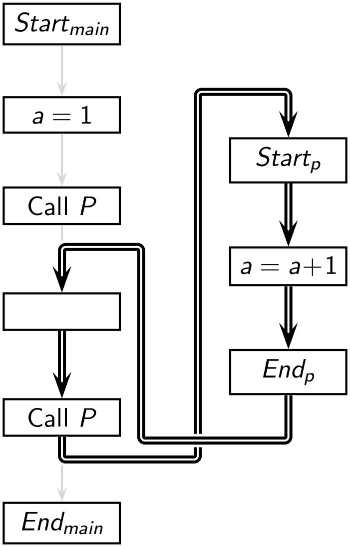
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3, 3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller
  - Range of a at End_{main} is $(2, \ldots)$

Context Insensitivity = Imprecision + Potential Non-Scalability



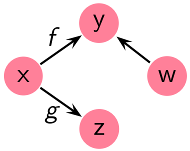
- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3, 3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller
  - Range of a at End_{main} is $(2, \ldots)$
- *Spurious interprocedural loops*

Context Insensitivity = Imprecision + Potential Non-Scalability

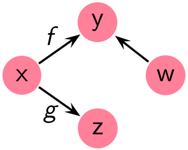
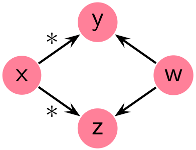


- What is the value range of a?
- Context sensitive analysis
  - Data flow value propagated back to the current caller of P
  - Range of a at End_{main} is $(3, 3)$
- Context insensitive analysis
  - Data flow value propagated back to every caller
  - Range of a at End_{main} is $(2, \ldots)$
- *Spurious interprocedural loops*
- *Spurious fixed point computations*

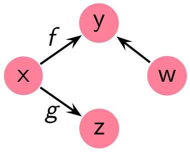
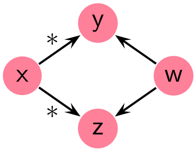
Field Sensitivity Vs. Field Insensitivity

Program	Field-sensitive points-to graph	Field-insensitive points-to graph
$x \rightarrow f = \&y$ $x \rightarrow g = \&z$ $w = x \rightarrow f$	 	

Field Sensitivity Vs. Field Insensitivity

Program	Field-sensitive points-to graph	Field-insensitive points-to graph
$x \to f = \&y$ $x \to g = \&z$ $w = x \to f$		

Field Sensitivity Vs. Field Insensitivity

Program	Field-sensitive points-to graph	Field-insensitive points-to graph
$x \rightarrow f = \&y$ $x \rightarrow g = \&z$ $w = x \rightarrow f$		

Field-insensitive analysis is less precise than a field-sensitive analysis

If I am Allowed to Nitpick ...

- Context sensitivity should involve all of the following

  [A] Full context sensitivity regardless of the call depth even in recursion
  [B] Ability to store data flow information parameterized by contexts at each program point
  [C] Flow sensitivity at the intraprocedural level (otherwise distinct calls to the same procedure within a procedure cannot be distinguished)

If I am Allowed to Nitpick . . .

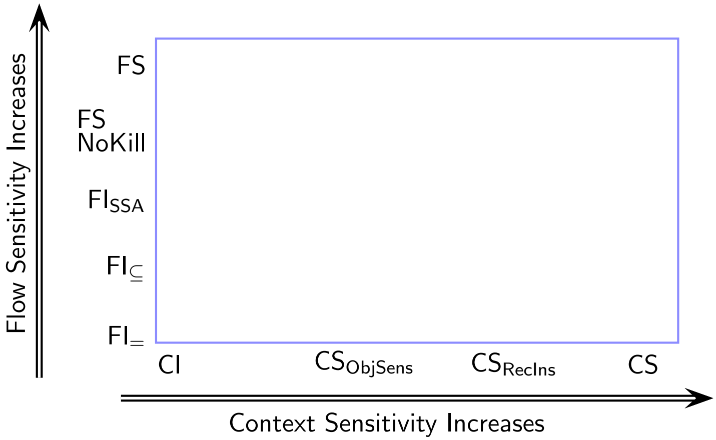
- Context sensitivity should involve all of the following
  - [A] Full context sensitivity regardless of the call depth even in recursion
  - [B] Ability to store data flow information parameterized by contexts at each program point
  - [C] Flow sensitivity at the intraprocedural level (otherwise distinct calls to the same procedure within a procedure cannot be distinguished)
- In particular
  - k-limiting violates [A]
  - Treating recursion as an SCC violates [A]
  - Functional approaches violate [B]
  - Object sensitivity violates [C]

If I am Allowed to Nitpick ...

- Context sensitivity should involve all of the following
  [A] Full context sensitivity regardless of the call depth even in recursion
  [B] Ability to store data flow information parameterized by contexts at each program point
  [C] Flow sensitivity at the intraprocedural level (otherwise distinct calls to the same procedure within a procedure cannot be distinguished)
- In particular
  - *k*-limiting violates [A]
  - Treating recursion as an SCC violates [A]
  - Functional approaches violate [B]
  - Object sensitivity violates [C]
- Object sensitivity (without the creation sites) can be modelled by call context sensitivity
  - by a flow sensitive propagation of values representing objects, and
  - identifying a procedure by an (object, procedure) pair, and
  - identifying a context by a call site and the pairs defined as above

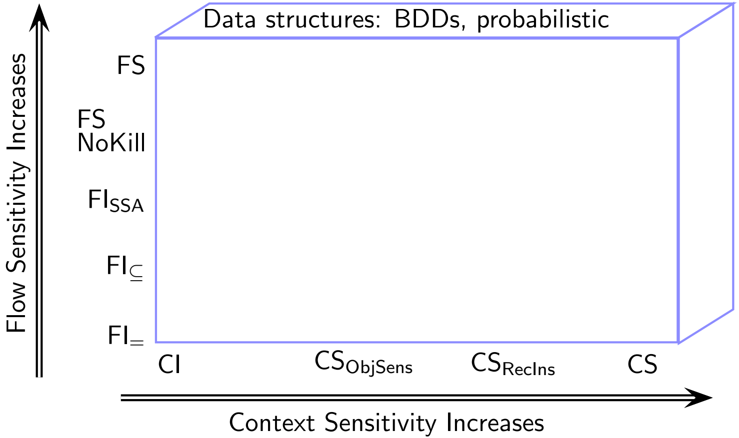
Pointer Analysis: An Engineer's Landscape

Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability



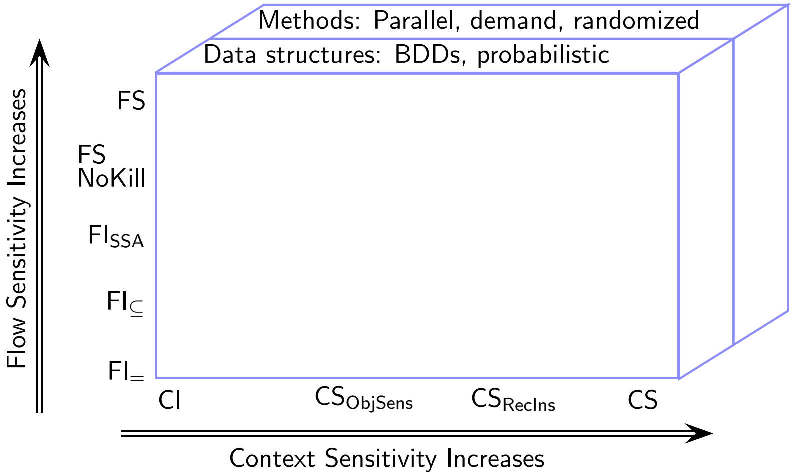
Pointer Analysis: An Engineer's Landscape

Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability



Pointer Analysis: An Engineer's Landscape

Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability



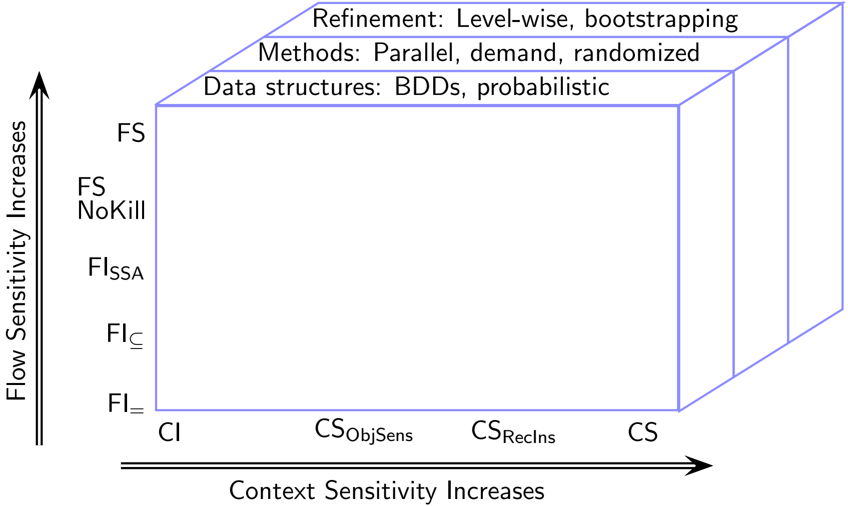
Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

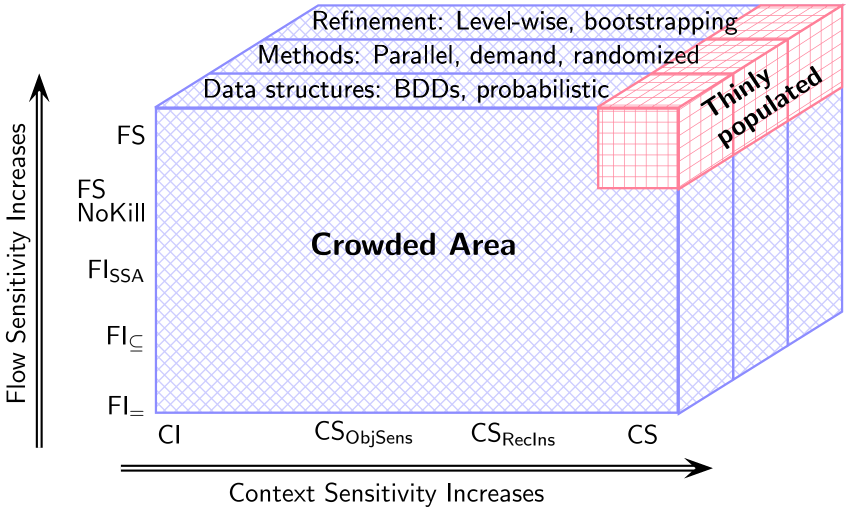
Pointer Analysis: An Engineer's Landscape

Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability



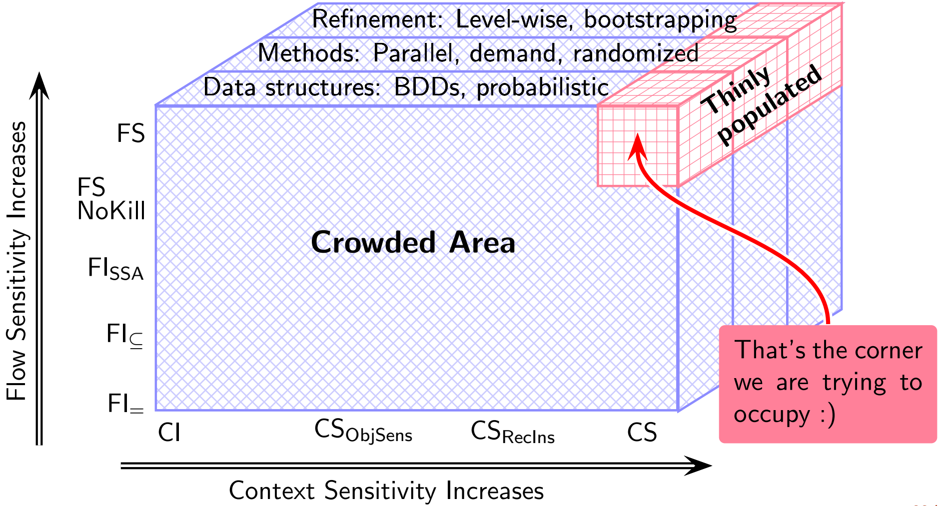
Pointer Analysis: An Engineer's Landscape

Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability



Pointer Analysis: An Engineer's Landscape

Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability



In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity		
Context-sensitivity (actually caller-sensitivity)		
Precise heap abstraction		
Precise call structure		

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
Context-sensitivity (actually caller-sensitivity)		
Precise heap abstraction		
Precise call structure		

Restrict the computation only to the usable data.
Weave liveness discovery into the analysis

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
Context-sensitivity (actually caller-sensitivity)		
Precise heap abstraction		
Precise call structure		

Avoid propagation of irrelevant information about pointers to callees

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
Context-sensitivity (actually caller-sensitivity)		
Precise heap abstraction		
Precise call structure		

Automatic flexible need-based collaboration between analyses

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

Interprocedural Analysis

Conclusions

References

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensitive analysis	Mature accomplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)		
Precise heap abstraction		
Precise call structure		

Lifting any given analyses to a level where known infeasible paths are excluded

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensitive analysis	Mature accomplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	Mature accomplishment (CC08,SOAP13)
Precise heap abstraction		
Precise call structure		

Distinguish between contexts by their data flow values and not their call chains

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	W
	Partially path-sensitive analysis	M CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	M SOAP13)
	GPG based bottom-up summary flow functions	Ma , TOPLAS20)
Precise heap abstraction		
Precise call structure		

Avoid recomputations for each context.
Use a higher level abstraction of memory.

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	W
	Partially path-sensitive analysis	N CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	M SOAP13)
	GPG based bottom-up summary flow functions	Mature accomplishment (SAS16, TOPLAS20)
	A Unified model of context-sensitive methods	Mature accomplishment (CSUR21)
Precise heap abstraction		
Precise call structure		

Model the essential requirements of context-sensitivity

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensitive analysis	Mature accomplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	SOAP13)
	GPG based bottom-up summ flow functions	,
	A Unified mod of context-sensitive methods	Mature accomplishment (CSUR21)
	Interprocedural SSA form	Work in progress
Precise heap abstraction		
Precise call structure		

Eliminate pointer dereferences and construct context-sensitive SSA

Uday Khedker
IIT Bombay

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensitive analysis	Mature accomplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	SOAP13)
	GPG based bottom-up summary flow functions	,
	A Unified model of context-sensitive methods	21)
	Interprocedural SSA form	Work in progress
Precise heap abstraction	Liveness analysis of heap	Partial accomplishment (TOPLAS07)
Precise call structure		

Identify the part of heap actually accessed in terms of patterns of accesses

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensitive analysis	Mature accomplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	Mature accomplishment (CC08,SOAP13)
	GPG based bottom-up summary flow functions	,
	A Unified model of context-sensitive methods	21)
	Interprocedural SSA form	
Precise heap abstraction	Liveness analysis of heap	Partial accomplishment (TOPLAS07)
	Combined allocation site and access path abstraction	Mature accomplishment (ISMM17)
Precise call structure		

Distinguish between heap locations based on how they are accessed apart from how they are allocated

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

Interprocedural Analysis

Conclusions

References

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensitive analysis	Mature accomplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	Mature accomplishment (CC08,SOAP13)
	GPG based bottom-up summary flow functions	Mature accomplishment (SAS16, TOPLAS20)
	A Unified model of context-sensitive methods	Mature accomplishment (CSUR21)
	Interprocedural SSA form	Wo
Precise heap abstraction	Liveness analysis of heap	Par S07)
	Combined allocation site and access path abstraction	Ma 17)
Precise call structure	Callee sensitivity	Work in progress

Call strings record call *history*. We need to record call *future* also.

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensitive analysis	Mature accomplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	Mature accomplishment (CC08,SOAP13)
	GPG based bottom-up summary flow functions	Mature accomplishment (SAS16, TOPLAS20)
	A Unified model of context-sensitive methods	Mature accomplishment (CSUR21)
	Interprocedural SSA form	
Precise heap abstraction	Liveness analysis of heap	S07)
	Combined allocation site and access path abstraction	17)
Precise call structure	Callee sensitivity	Work in progress
	Precise virtual call resolution	Mature accomplishment (SCP20)

Make the call graph more precise by computing a more precise set of callees

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensitive analysis	Mature accomplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	Mature accomplishment (CC08,SOAP13)
	GPG based bottom-up summary flow functions	Mature accomplishment (SAS16, TOPLAS20)
	A Unified model of context-sensitive methods	Mature accomplishment (CSUR21)
	Interprocedural SSA form	Work in progress
Precise heap abstraction	Liveness analysis of heap	Partial accomplishment (TOPLAS07)
	Combined allocation site and access path abstraction	Mature accomplishment (ISMM17)
Precise call structure	Callee sensitivity	Work in progress
	Precise virtual call resolution	Mature accomplishment (SCP20)

In Search of Abstractions for Precision Without Non-Scalability

Desired Abstraction	Enabling Abstraction	Status of our work
Flow- and Field-sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	Bypassing irrelevant calls for liveness and points-to analysis	Work in progress
	Synergistic program analyses	Work in progress
	Partially path-sensiti	omplishment (CC18,CC19)
Context-sensitivity (actually caller-sensitivity)	Value contexts	hment (CC08,SOAP13)
	GPG based flow functio	ent (SAS16,
	A Unified m methods	ent (CSUR21)
	Interprocedural S	s
Precise heap abstraction	Liveness analysis of hea	accomplishment (TOPLAS07)
	Combined allocation site and access path abstraction	Mature accomplishment (ISMM17)
Precise call structure	Callee sensitivity	Work in progress
	Precise virtual call resolution	Mature accomplishment (SCP20)

We are destined to a long haul with no guarantees :-)

Some Short Trips

Examples of some research explorations in

- Intraprocedural Analysis
  - Combined allocation site and access path abstraction for heap
  - Liveness analysis of heap data
  - Liveness-based points-to analysis
  - Synergistic program analysis
  - Partially path-sensitive analysis
- Interprocedural analysis
  - Broad categories of interprocedural analysis
  - Scaling top-down analysis using value contexts and bypassing
  - Improving bottom-up analysis by eliminating control flow
  - Precise virtual call resolution with demand-driven analysis
  - Improving call graphs using callee contexts

Intraprocedural Analysis

An Outline of Research Explorations in Intraprocedural Analysis

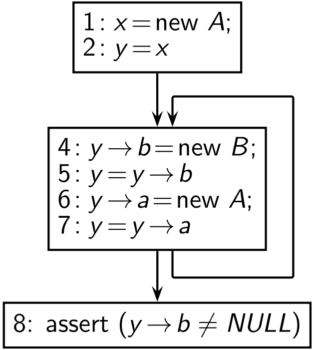
- Combined allocation site and access path abstraction for heap
  Next Topic
- Liveness analysis of heap data
- Liveness-based points-to analysis
- Synergistic program analysis
- Partially path-sensitive analysis

Towards A More Precise Heap Abstraction

- Challenges in static analysis of heap pointer variables
  - Unpredictable lifetime
  - Unbounded number of allocations
  - Unnamed locations
- Unbounded heap memory can be summarized by creating a finite number of abstract nodes where each abstract node represents multiple concrete nodes
- Three options to create a finite number of abstract locations with compile-time names
  1. Represent all heap locations by a single abstract heap location
  2. Represent all heap locations of a particular type by a single abstract heap location
  3. Represent all heap locations allocated at a given memory allocation site by a single abstract heap location

Allocation Site Based Abstraction of Heap Memory

Program



Allocation Site Based Abstraction of Heap Memory

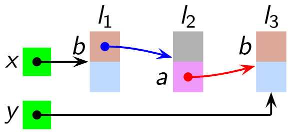
Program

$1: x = \text{new } A;$
$2: y = x$

$4: y \rightarrow b = \text{new } B;$
$5: y = y \rightarrow b$
$6: y \rightarrow a = \text{new } A;$
$7: y = y \rightarrow a$

$8: \text{ assert } (y \rightarrow b \neq NULL)$

Memory graphs at statement 8 in different executions

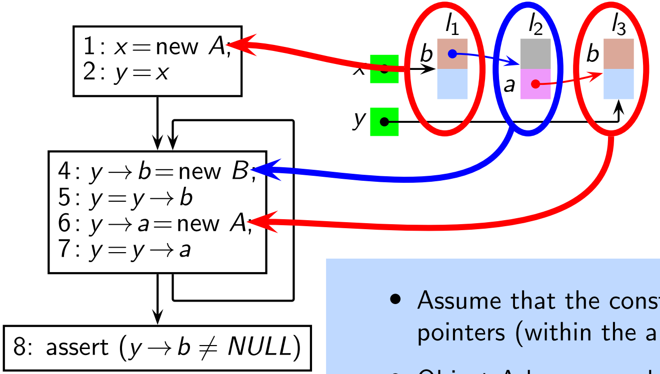


- Assume that the constructors initialize all pointers (within the allocated object) to NULL

Allocation Site Based Abstraction of Heap Memory

Program

Memory graphs at statement 8 in different executions



- Assume that the constructors initialize all pointers (within the allocated object) to NULL
- Object A has a member field *b* that contains a pointer to object B
- Object B has a member field *a* that contains a pointer to object A

Allocation Site Based Abstraction of Heap Memory

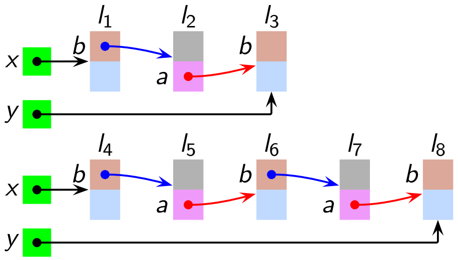
Program

$1: x = \text{new } A;$
$2: y = x$

$4: y \rightarrow b = \text{new } B;$
$5: y = y \rightarrow b$
$6: y \rightarrow a = \text{new } A;$
$7: y = y \rightarrow a$

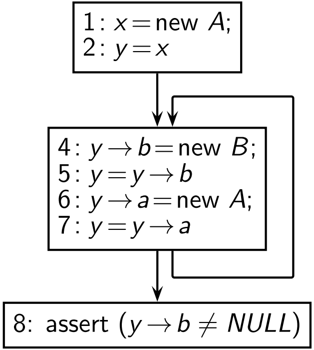
$8:\ \text{assert } (y \rightarrow b \neq NULL)$

Memory graphs at statement 8 in different executions

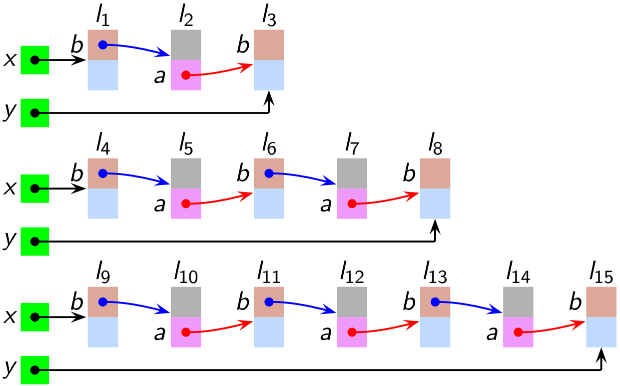


Allocation Site Based Abstraction of Heap Memory

Program

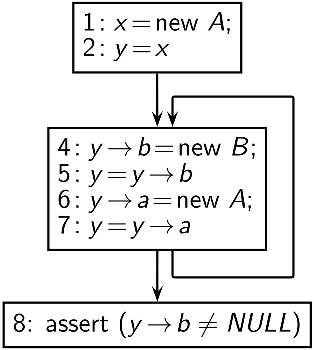


Memory graphs at statement 8 in different executions

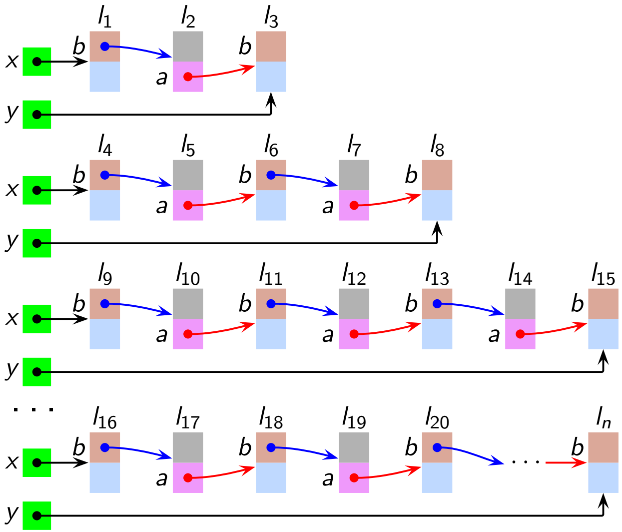


Allocation Site Based Abstraction of Heap Memory

Program



Memory graphs at statement 8 in different executions



Allocation Site Based Abstraction of Heap Memory

Program

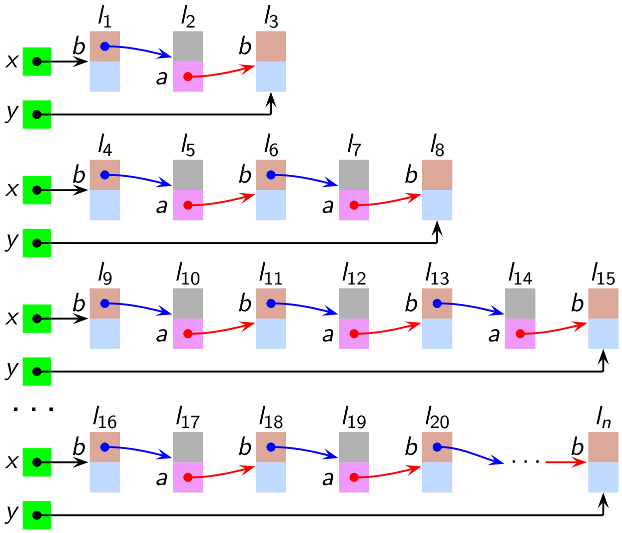
$1: x = \text{new } A;$
$2: y = x$

$4: y \rightarrow b = \text{new } B;$
$5: y = y \rightarrow b$
$6: y \rightarrow a = \text{new } A;$
$7: y = y \rightarrow a$

$8:$ assert $(y \rightarrow b \neq NULL)$

The assertion is false because $y \rightarrow b$ is always NULL in statement 8

Memory graphs at statement 8 in different executions

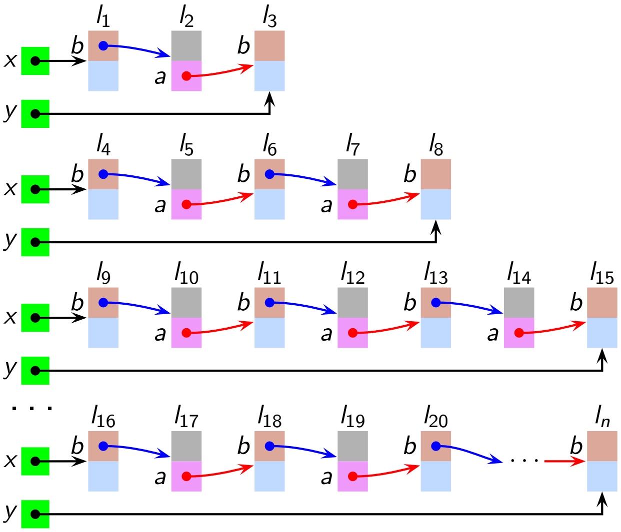


Summarizing the Unbounded Heap

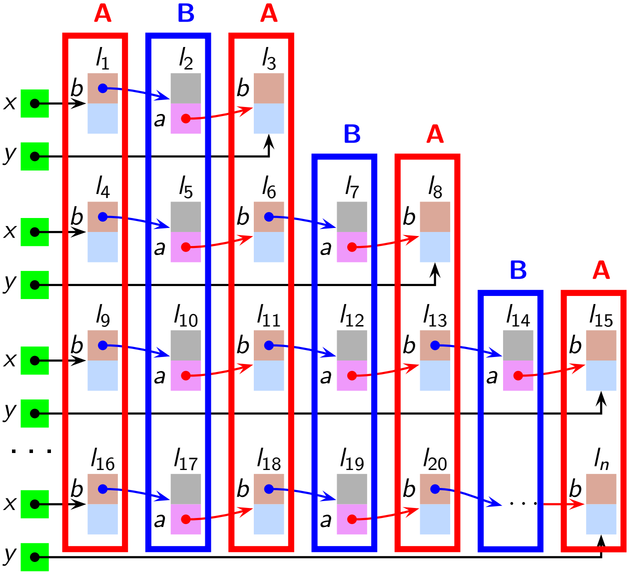
We discuss the following strategies for summarizing the unbounded heap

1. Type Abstraction (TA). Summarization using types by creating a single abstract node per type representing all locations of the same type
2. Allocation Site Abstraction (ASA). Summarization using allocation sites by creating a single abstract node per allocation site representing all locations allocated at the same site
3. Access Path Abstraction (APA). Summarization using access paths (which also need summarization) where each summarized access paths denotes a single abstract node representing all locations reached by the access path
   - 3.1 Summarization of access paths using finite automata
   - 3.2 Summarization of access paths using k-limiting
   - 3.3 Summarization of access paths using at most k repetitions of a field
4. Combined Allocation Site and Access Path Abstraction (CASAPA). Summarization using allocation sites and access paths together

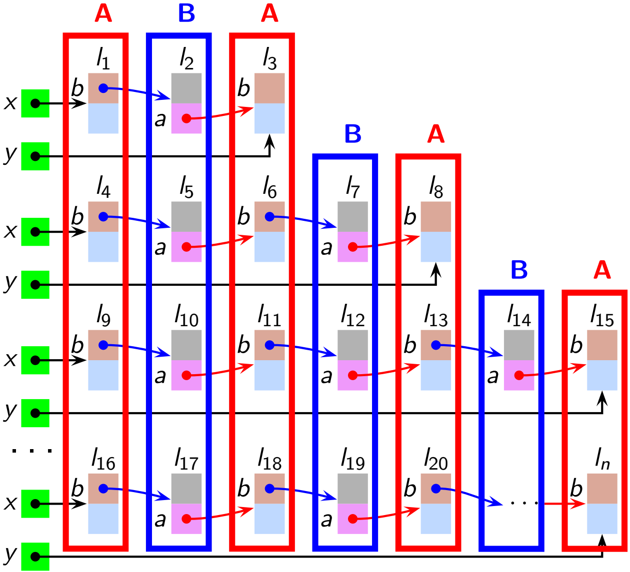
Summarizing Heap at Statement 8 Using Type Abstraction

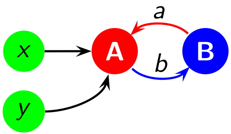


Summarizing Heap at Statement 8 Using Type Abstraction

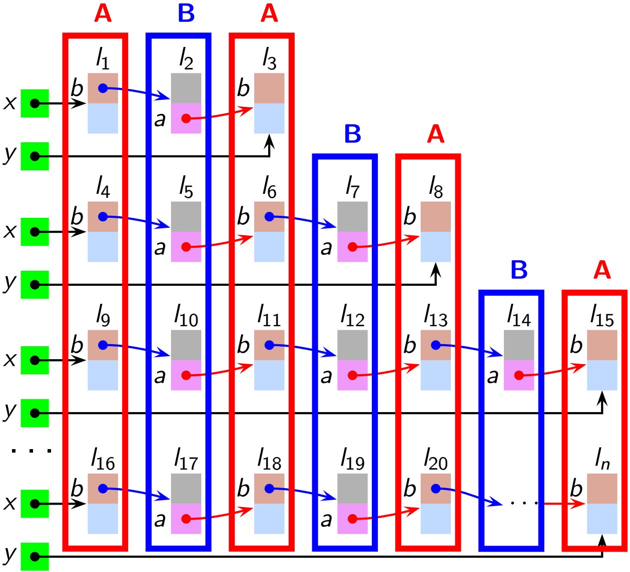


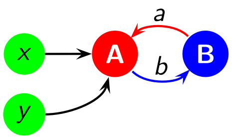
Summarizing Heap at Statement 8 Using Type Abstraction





Summarizing Heap at Statement 8 Using Type Abstraction



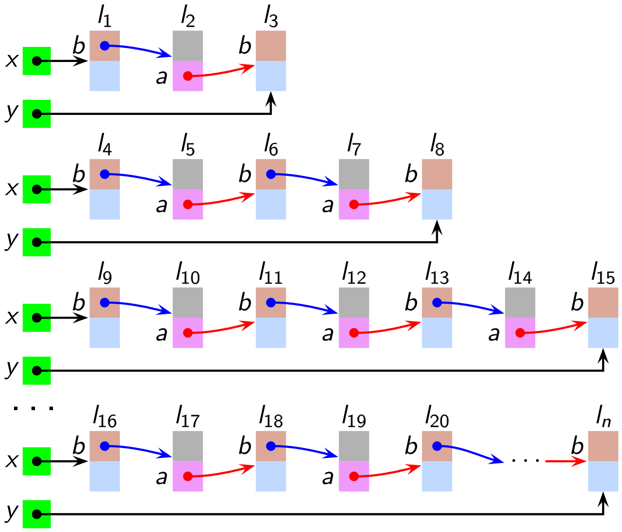


Since node A has an out edge labelled b, we cannot guarantee that $y \rightarrow b$ is NULL

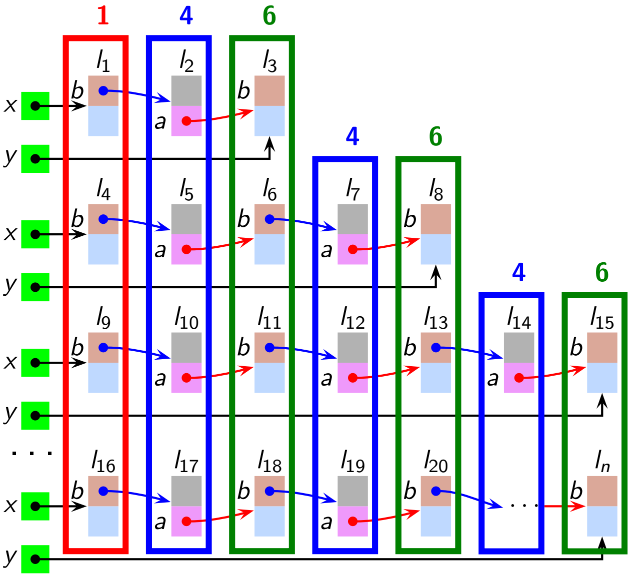
Hence we cannot prove the assertion

Besides, x and y are not aliased at statement 8

Summarizing Heap at Statement 8 Using Allocation Sites

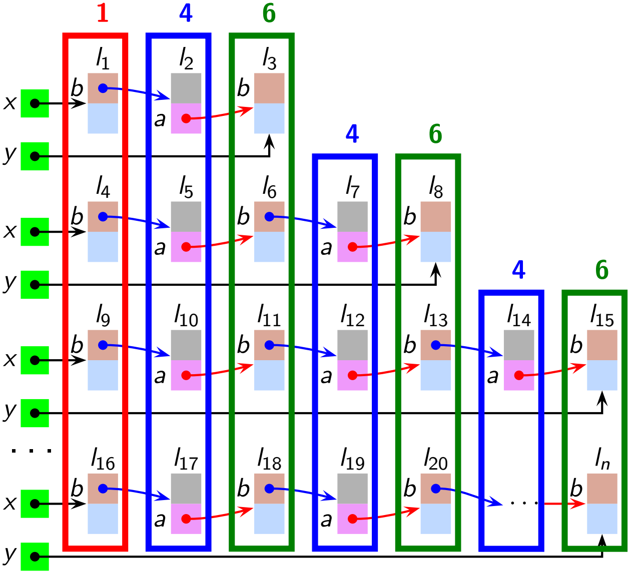


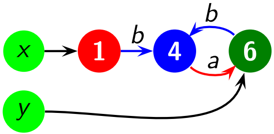
Summarizing Heap at Statement 8 Using Allocation Sites



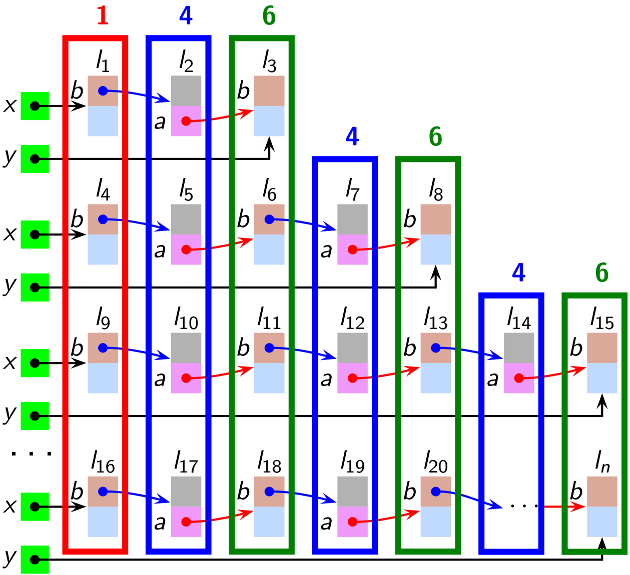
Uday Khedker
IIT Bombay

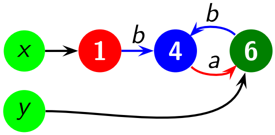
Summarizing Heap at Statement 8 Using Allocation Sites





Summarizing Heap at Statement 8 Using Allocation Sites





Since node 6 has an out edge labelled b, we cannot guarantee that $y \rightarrow b$ is NULL

Hence we cannot prove the assertion

However, this is more precise than type abstraction because x and y are not aliased

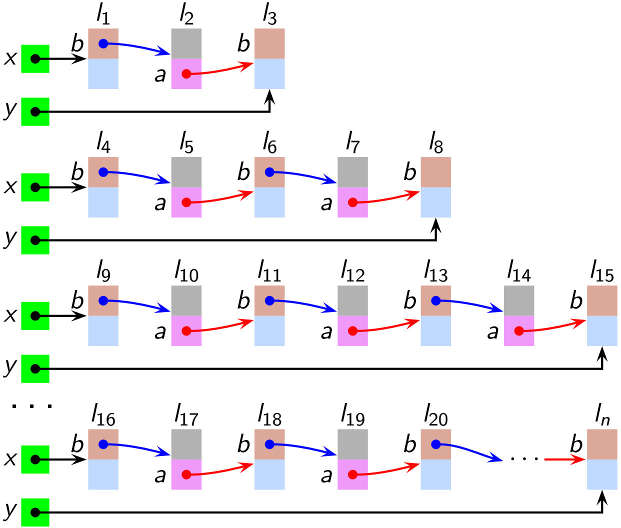
Summarizing Heap at Statement 8 Using Access Paths

- An access path is sequence of field names starting with a root variable
  - Some examples are: x , $x \cdot a \cdot a$, $x \cdot a \cdot b \cdot a \cdots b \cdots a \cdot b$
  - The last access path is unbounded
  - It represents multiple access paths that have the same pattern but different lengths
- We use three methods of summarizing unbounded access paths
  - Summarization of access paths using finite automata (or regular expressions)
  - Summarization of access paths using k-limiting
  - Summarization of access paths using at most k repetitions of a field

Uday Khedker
IIT Bombay

Summarizing Heap at Statement 8 Using Access Paths

Paths in the Memory



Corresponding Access Paths

$x \cdot b \cdot a$

y

$x \cdot b \cdot a \cdot b \cdot a$

y

$x \cdot b \cdot a \cdot b \cdot a \cdot b \cdot a$

y

$x \cdot b \cdot a \cdot b \cdot a \cdots b \cdot a$

y

Summarizing Heap at Statement 8 Using Access Paths

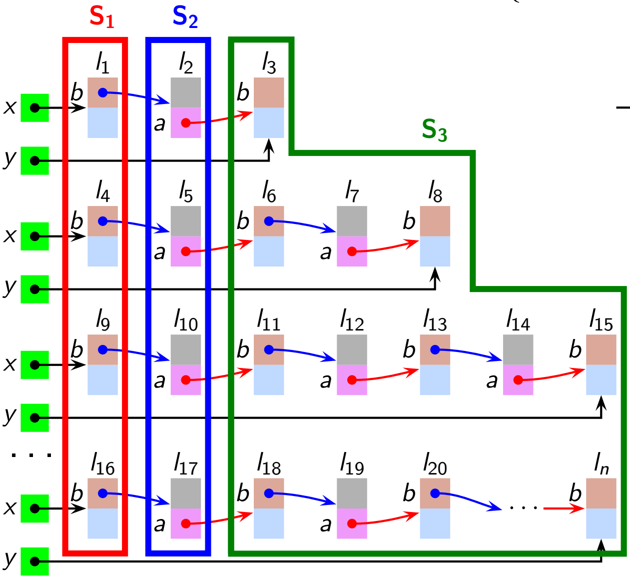
- The set of out access paths is
  $\{y,\ x \cdot b \cdot a \cdot b \cdot a,\ x \cdot b \cdot a \cdot b \cdot a \cdot b \cdot a,\ x \cdot b \cdot a \cdot b \cdot a \cdots b\}$
- Since each prefix represents a subpath reaching a memory node appearing on the path, a set of access paths is considered prefix-closed and thus our set is $\{y,\ x \cdot b \cdot a \cdot b \cdot a \cdots b\}$

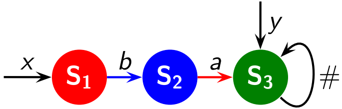
Summarizing Heap at Statement 8 Using Access Paths

- The set $\{y,\ x\cdot b\cdot a\cdot b\cdot a\cdots b\}$ can be summarized in three ways
  - With k-limiting, for $k = 2$, it becomes
    $L_2 = \{y,\ x\cdot b\cdot a\cdot \#^*\}$ where "#" is the wild card symbol representing any field and $*$ is the kleene closure operator
  - With at most k repetitions of a field, for $k = 2$, it becomes
    $R_2 = \{y,\ x\cdot b\cdot a\cdot b\cdot \#^*\}$ where "#" is the wild card symbol and $*$ is the kleene closure operator
  - With finite automata represented by regular expressions, it becomes
    $RE = \{y,\ x\cdot b,\ x\cdot(b\cdot a)^+\}$ where $+$ is the positive closure operator ($x\cdot b$ is separated from $x\cdot(b\cdot a)^+$ because y is not aliased to $x\cdot b$)
- In each case, we create appropriate abstract nodes depending upon the sets of access paths reaching a memory location represented by the abstract node

Summarizing Heap at Statement 8 Using Access Paths

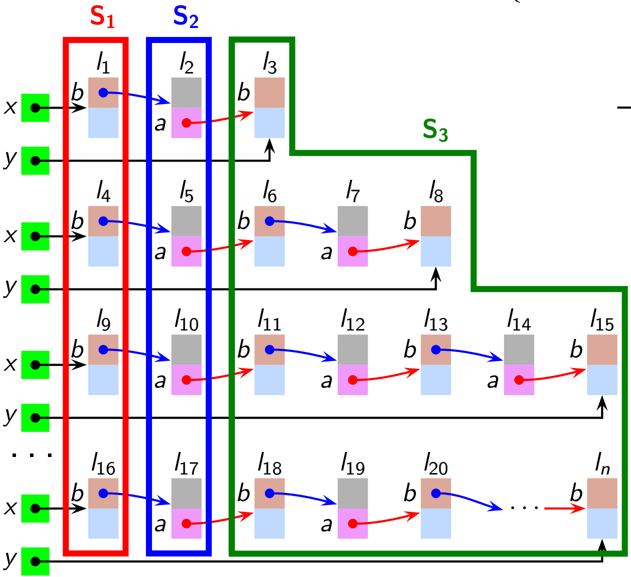
Summarization using 2-limiting with $L_2 = \{y,\ x \cdot b \cdot a \cdot \#^*\}$

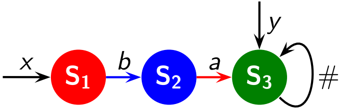




Summarizing Heap at Statement 8 Using Access Paths

Summarization using 2-limiting with $L_2 = \{y,\ x \cdot b \cdot a \cdot \#^*\}$

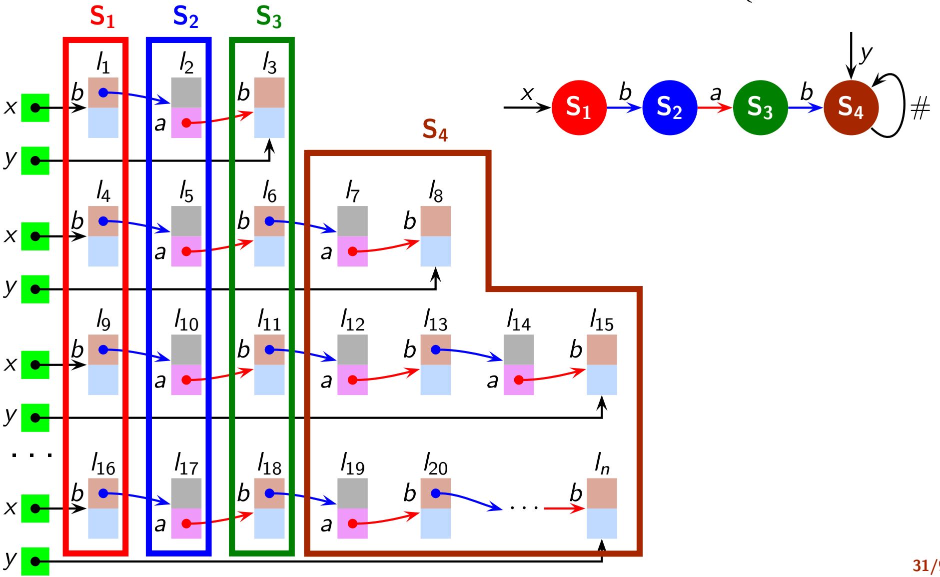




Since node S_3 has an out edge labelled $\#$, we cannot guarantee that $y \rightarrow b$ is NULL and hence we cannot prove the assertion

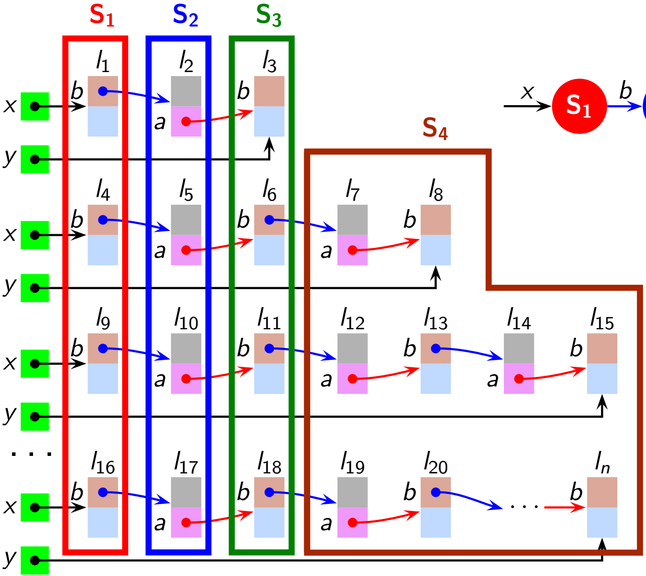
Summarizing Heap at Statement 8 Using Access Paths

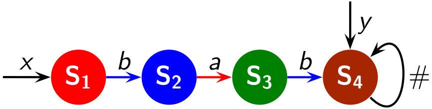
Summarization using at most 2 repetitions of any field with $R_2 = \{y,\ x \cdot b \cdot a \cdot b \cdot \#^*\}$



Summarizing Heap at Statement 8 Using Access Paths

Summarization using at most 2 repetitions of any field with $R_2 = \{y,\ x \cdot b \cdot a \cdot b \cdot \#^*\}$

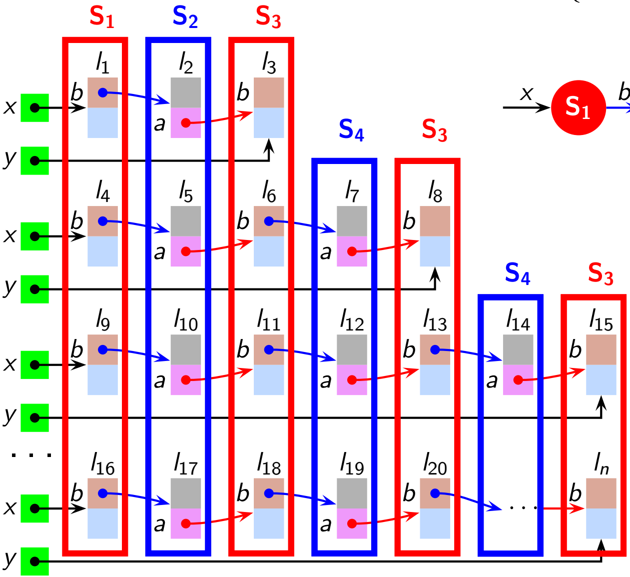


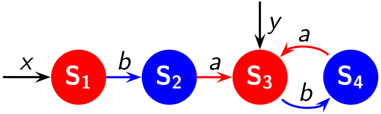


Since node S_4 has an out edge labelled $\#$, we cannot guarantee that $y \rightarrow b$ is NULL and hence we cannot prove the assertion

Summarizing Heap at Statement 8 Using Access Paths

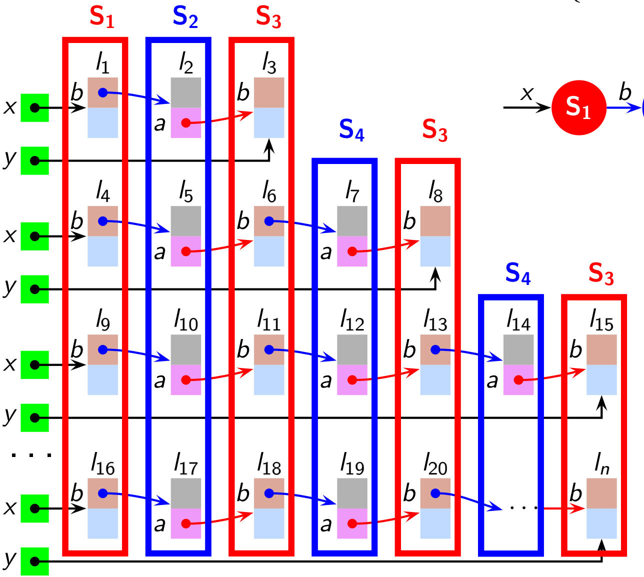
Summarization using finite automata with $RE = \{y,\ x\cdot b,\ x\cdot(b\cdot a)^+\}$

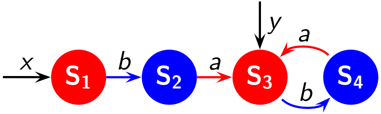




Summarizing Heap at Statement 8 Using Access Paths

Summarization using finite automata with $RE = \{y,\ x{\cdot}b,\ x{\cdot}(b\cdot a)^{+}\}$





Since node S_3 has an out edge labelled b, we cannot guarantee that $y \rightarrow b$ is NULL and hence we cannot prove the assertion

Summarization using regular expressions gives the most precise summarized access paths

Summarizing Heap at Statement 8 Using Combined Allocation Sites and Access Paths Abstraction (CASAPA) [ISMM17]

- Allocation site abstraction (ASA) groups locations that are allocated at the same site in the hope that they will be used in a similar manner
- Access path abstraction (APA) groups locations that are accessed similarly and the program text cannot distinguish between them statically
- Some times both these are too coarse so we can combine the two to get the best of both the worlds

  The idea is to make further subdivisions of the sets of locations for an allocation site using access paths

Summarizing Heap at Statement 8 Using Combined Allocation Sites and Access Paths Abstraction (CASAPA) [ISMM17]

The key idea behind CASAPA

- Use allocation sites to partition all memory locations

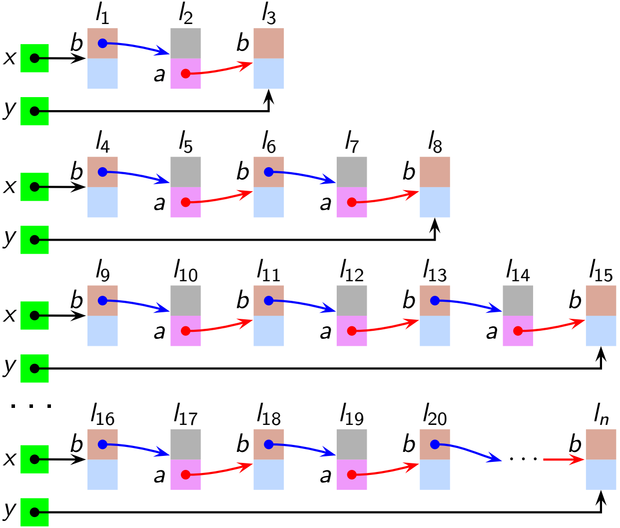
  Let L_n be the set of locations allocated at site n

  In general, L_n may be unbounded

- Partition S_n on the basis of sets of access paths reaching locations in S_n
  - We can distinguish between two nodes statically if different sets of access paths reach them
  - Otherwise, we cannot distinguish between them

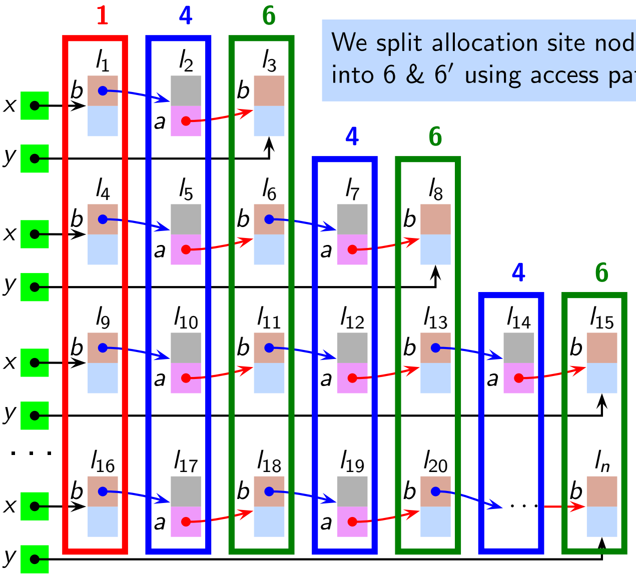
  Make the distinctions that we can, merge other locations into a single abstract location

Summarizing Heap at Statement 8 Using Combined Allocation Sites and Access Paths Abstraction (CASAPA) [ISMM17]

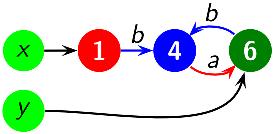


Summarizing Heap at Statement 8 Using Combined Allocation Sites and Access Paths Abstraction (CASAPA) [ISMM17]

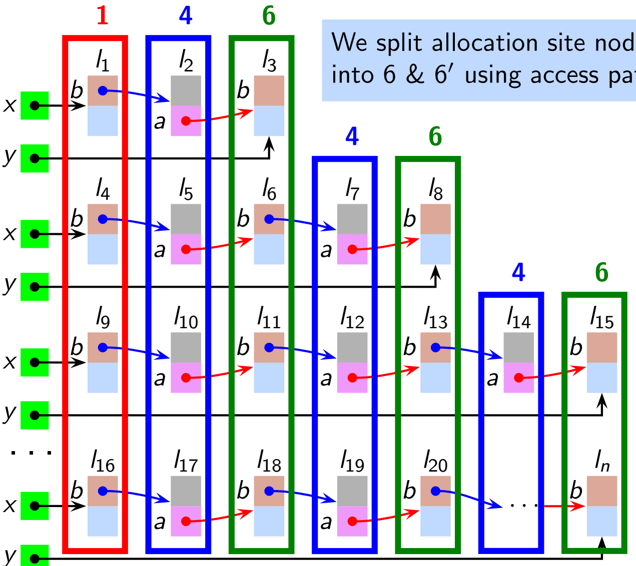
Uday Khedker
IIT Bombay



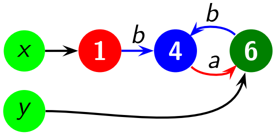
We split allocation site node 6 into 6 & 6′ using access paths



Summarizing Heap at Statement 8 Using Combined Allocation Sites and Access Paths Abstraction (CASAPA) [ISMM17]



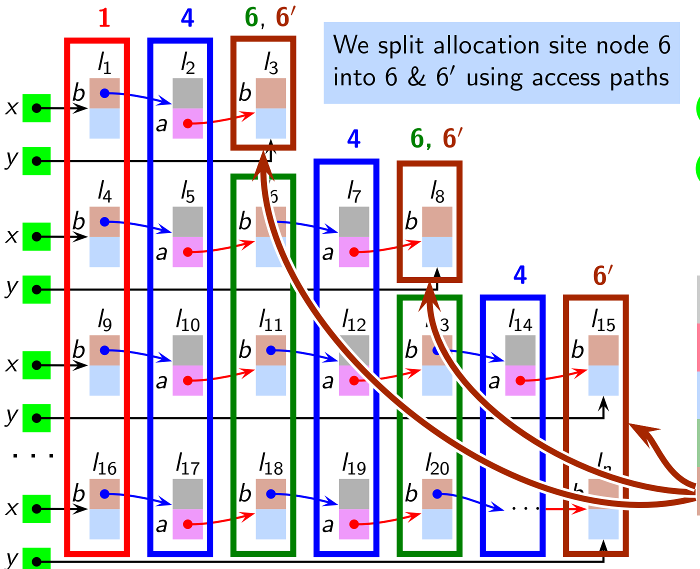
We split allocation site node 6 into 6 & 6′ using access paths



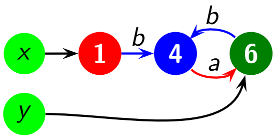
AS	AP Set
1	$\{x\}$
4	$\{x \cdot b \cdot (a \cdot b)^*\}$
6	$\{x \cdot (b \cdot a)^+\}$
6	$\{y, x \cdot (b \cdot a)^+\}$

Summarizing Heap at Statement 8 Using Combined Allocation Sites and Access Paths Abstraction (CASAPA) [ISMM17]

Uday Khedker
IIT Bombay

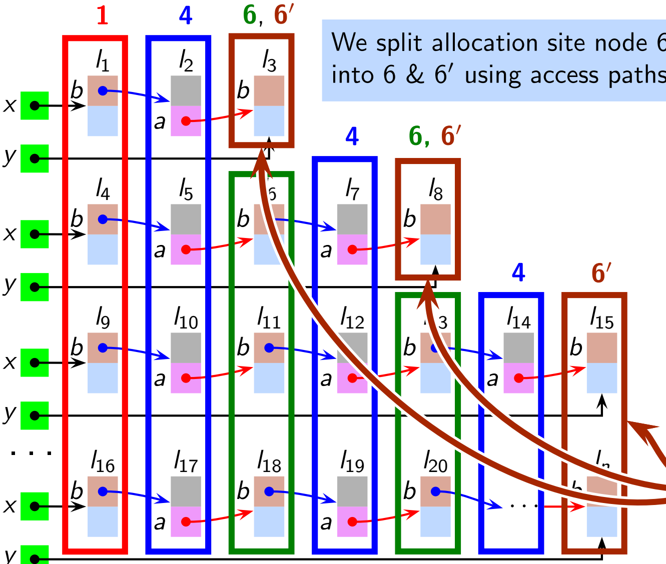


We split allocation site node 6 into 6 & 6′ using access paths

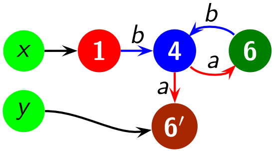


AS	AP Set
1	$\{x\}$
4	$\{x \cdot b \cdot (a \cdot b)^*\}$
6	$\{x \cdot (b \cdot a)^+\}$
6	$\{y, x \cdot (b \cdot a)^+\}$

Summarizing Heap at Statement 8 Using Combined Allocation Sites and Access Paths Abstraction (CASAPA) [ISMM17]

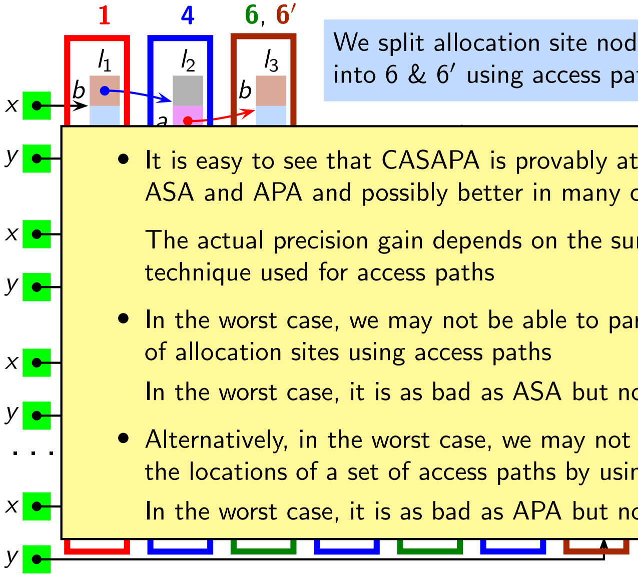


We split allocation site node 6 into 6 & 6′ using access paths

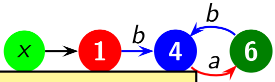


AS	AP Set
1	$\{x\}$
4	$\{x \cdot b \cdot (a \cdot b)^*\}$
6	$\{x \cdot (b \cdot a)^+\}$
6	$\{y, x \cdot (b \cdot a)^+\}$

Summarizing Heap at Statement 8 Using Combined Allocation Sites and Access Paths Abstraction (CASAPA) [ISMM17]



We split allocation site node 6 into 6 & 6′ using access paths



- It is easy to see that CASAPA is provably at least as good as ASA and APA and possibly better in many cases

  The actual precision gain depends on the summarization technique used for access paths

- In the worst case, we may not be able to partition the locations of allocation sites using access paths

  In the worst case, it is as bad as ASA but no worse

- Alternatively, in the worst case, we may not be able to partition the locations of a set of access paths by using allocation sites

  In the worst case, it is as bad as APA but no worse

An Outline of Research Explorations in Intraprocedural Analysis

- Combined allocation site and access path abstraction for heap
- Liveness analysis of heap data — Next Topic
- Liveness-based points-to analysis
- Synergistic program analysis
- Partially path-sensitive analysis

Liveness Analysis of Heap Data

- Problem.
- Our Objectives.
- Main Challenge.
- Our Key Idea.
- Current status.
- Further Work.

Liveness Analysis of Heap Data

- Problem. A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem
- Our Objectives.
- Main Challenge.
- Our Key Idea.
- Current status.
- Further Work.

Liveness Analysis of Heap Data

- Problem. A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem
- Our Objectives. Static analysis of heap data to improve garbage collection and plug memory leaks
- Main Challenge.
- Our Key Idea.
- Current status.
- Further Work.

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

Interprocedural Analysis

Conclusions

References

Liveness Analysis of Heap Data

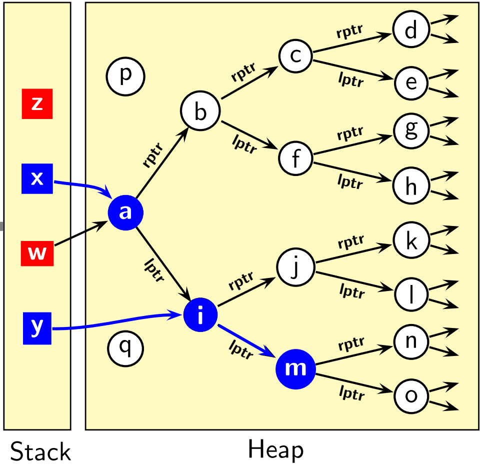
- Problem. A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem
- Our Objectives. Static analysis of heap data to improve garbage collection and plug memory leaks
- Main Challenge. Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime
- Our Key Idea.
- Current status.
- Further Work.

Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is not executed even once.

```
w = x          // x points to m_a
while (x.data < max)
      x = x.rptr
y = x.lptr

z = New class_of_z
y = y.lptr
z.sum = x.data + y.data
return z.sum
```

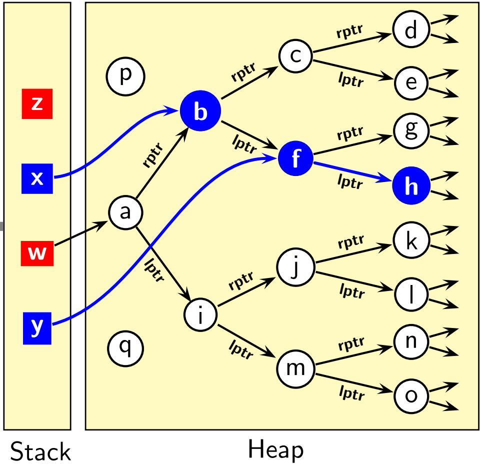


Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is executed once.

```
1    w = x          // x points to m_a
2    while  (x.data < max)
3           x = x.rptr
4    y = x.lptr

5    z = New  class_of_z
6    y = y.lptr
7    z.sum = x.data + y.data
8    return z.sum
```

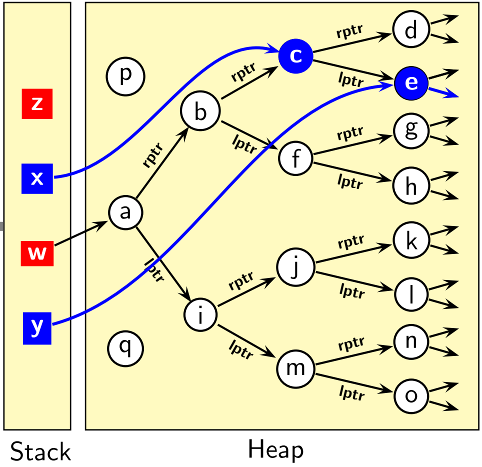


Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is executed twice.

```
w = x          // x points to m_a
while  (x.data < max)
        x = x.rptr
y = x.lptr

z = New  class_of_z
y = y.lptr
z.sum = x.data + y.data
return z.sum
```



Liveness Analysis of Heap Data [TOPLAS07]

- Problem. A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem
- Our Objectives. Static analysis of heap data to improve garbage collection and plug memory leaks
- Main Challenge. Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime
- Our Key Idea.
- Current status.
- Further Work.

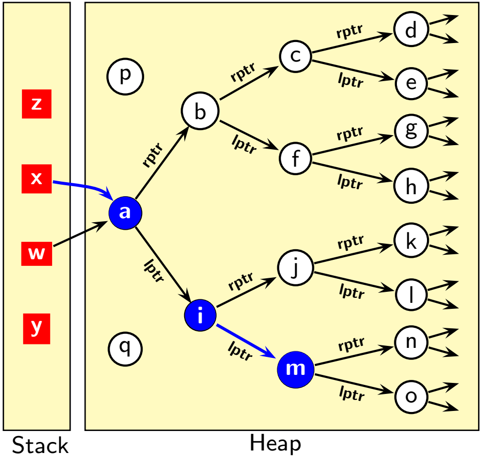
Liveness Analysis of Heap Data [TOPLAS07]

- Problem. A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem
- Our Objectives. Static analysis of heap data to improve garbage collection and plug memory leaks
- Main Challenge. Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime
- Our Key Idea. Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values
- Current status.
- Further Work.

Liveness Analysis of Heap Data [TOPLAS07]


```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {      x.lptr = null
3          x = x.rptr       }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = null
8   return z.sum
    z = null
```

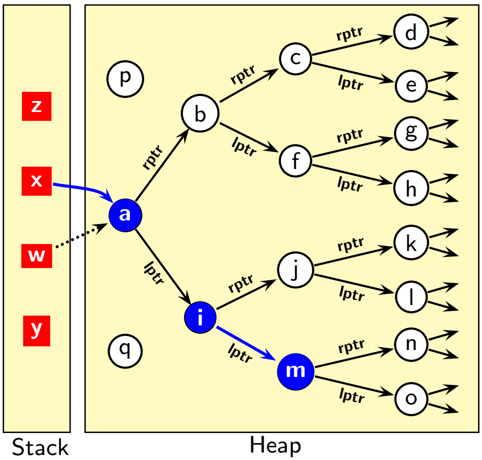
While loop is not executed even once



Liveness Analysis of Heap Data [TOPLAS07]

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr       }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = null
8   return z.sum
    z = null
```

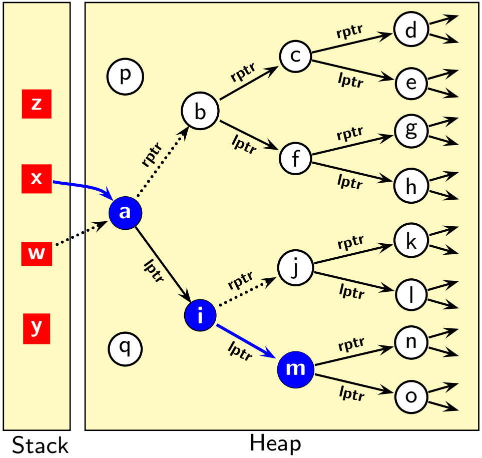
While loop is not executed even once



Liveness Analysis of Heap Data [TOPLAS07]

	y = z = null
1	w = x
	w = null
2	while (x.data < max)
	{ x.lptr = null
3	x = x.rptr }
	x.rptr = x.lptr.rptr = null
	x.lptr.lptr.lptr = null
	x.lptr.lptr.rptr = null
4	y = x.lptr
	x.lptr = y.rptr = null
	y.lptr.lptr = y.lptr.rptr = null
5	z = *New class_of_z*
	z.lptr = z.rptr = null
6	y = y.lptr
	y.lptr = y.rptr = null
7	z.sum = x.data + y.data
	x = y = null
8	return z.sum
	z = null

While loop is not executed even once



Liveness Analysis of Heap Data [TOPLAS07]

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

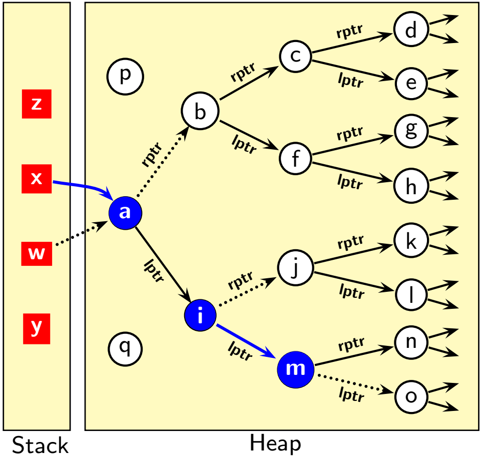
Interprocedural Analysis

Conclusions

References

	y = z = null
1	w = x
	w = null
2	while (x.data < max)
	{ x.lptr = null
3	x = x.rptr }
	x.rptr = x.lptr.rptr = null
	x.lptr.lptr.lptr = null
	x.lptr.lptr.rptr = null
4	y = x.lptr
	x.lptr = y.rptr = null
	y.lptr.lptr = y.lptr.rptr = null
5	z = *New class_of_z*
	z.lptr = z.rptr = null
6	y = y.lptr
	y.lptr = y.rptr = null
7	z.sum = x.data + y.data
	x = y = null
8	return z.sum
	z = null

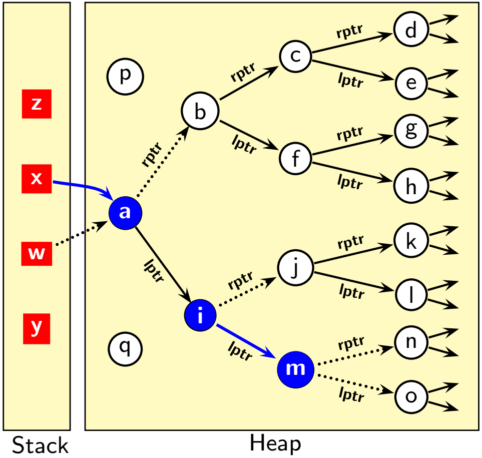
While loop is not executed even once



Liveness Analysis of Heap Data [TOPLAS07]

```
     y = z = null
1    w = x
     w = null
2    while (x.data < max)
     {      x.lptr = null
3           x = x.rptr       }
     x.rptr = x.lptr.rptr = null
     x.lptr.lptr.lptr = null
     x.lptr.lptr.rptr = null
4    y = x.lptr
     x.lptr = y.rptr = null
     y.lptr.lptr = y.lptr.rptr = null
5    z = New  class_of_z
     z.lptr = z.rptr = null
6    y = y.lptr
     y.lptr = y.rptr = null
7    z.sum = x.data + y.data
     x = y = null
8    return z.sum
     z = null
```

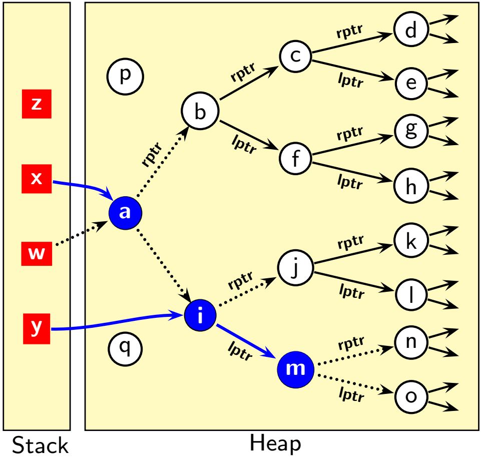
While loop is not executed even once



Liveness Analysis of Heap Data [TOPLAS07]

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {      x.lptr = null
3          x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = null
8   return z.sum
    z = null
```

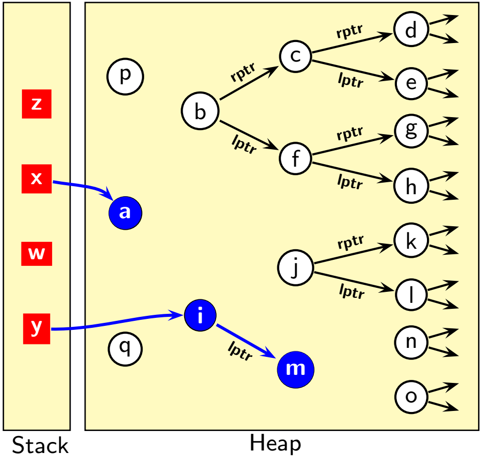
While loop is not executed even once



Liveness Analysis of Heap Data [TOPLAS07]

```
     y = z = null
1    w = x
     w = null
2    while (x.data < max)
     {       x.lptr = null
3            x = x.rptr        }
     x.rptr = x.lptr.rptr = null
     x.lptr.lptr.lptr = null
     x.lptr.lptr.rptr = null
4    y = x.lptr
     x.lptr = y.rptr = null
     y.lptr.lptr = y.lptr.rptr = null
5    z = New  class_of_z
     z.lptr = z.rptr = null
6    y = y.lptr
     y.lptr = y.rptr = null
7    z.sum = x.data + y.data
     x = y = null
8    return z.sum
     z = null
```

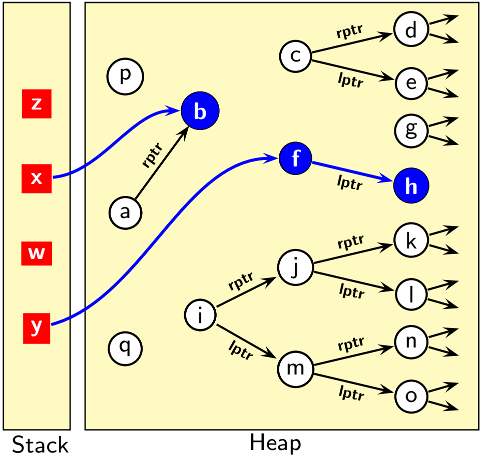
While loop is not executed even once



Liveness Analysis of Heap Data [TOPLAS07]


```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr        }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = null
8   return z.sum
    z = null
```

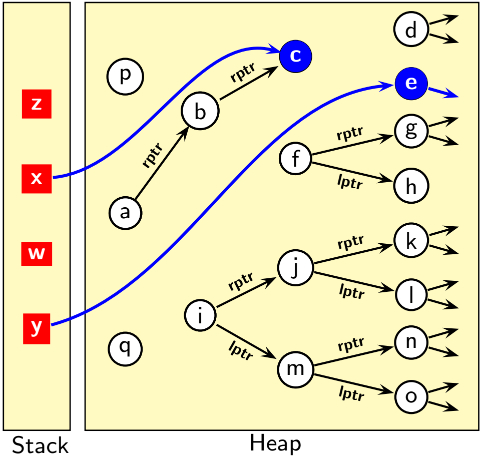
While loop is executed once



Liveness Analysis of Heap Data [TOPLAS07]

	y = z = null
1	w = x
	w = null
2	while (x.data < max)
	{ x.lptr = null
3	x = x.rptr }
	x.rptr = x.lptr.rptr = null
	x.lptr.lptr.lptr = null
	x.lptr.lptr.rptr = null
4	y = x.lptr
	x.lptr = y.rptr = null
	y.lptr.lptr = y.lptr.rptr = null
5	z = *New class_of_z*
	z.lptr = z.rptr = null
6	y = y.lptr
	y.lptr = y.rptr = null
7	z.sum = x.data + y.data
	x = y = null
8	return z.sum
	z = null

While loop is executed twice



Liveness Analysis of Heap Data [TOPLAS07]

- Problem. A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem
- Our Objectives. Static analysis of heap allocated data to improve garbage collection and plug memory leaks
- Main Challenge. Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime
- Our Key Idea. Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values
- Current status.
- Further Work.

Liveness Analysis of Heap Data [TOPLAS07]

- Problem. A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem
- Our Objectives. Static analysis of heap allocated data to improve garbage collection and plug memory leaks
- Main Challenge. Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime
- Our Key Idea. Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values
- Current status. Theory and prototype implementation (at the intraprocedural level) ready for Java
- Further Work.

Liveness Analysis of Heap Data [TOPLAS07]

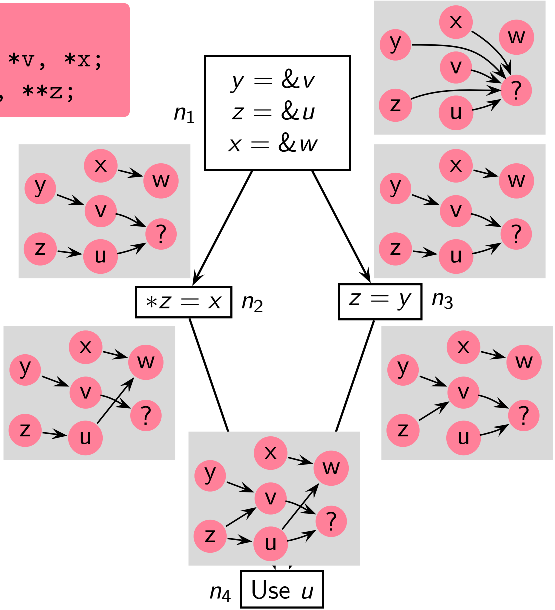
- Problem. A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem
- Our Objectives. Static analysis of heap allocated data to improve garbage collection and plug memory leaks
- Main Challenge. Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime
- Our Key Idea. Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values
- Current status. Theory and prototype implementation (at the intraprocedural level) ready for Java
- Further Work. Liveness based interprocedural alias analysis

Research Explorations in Intraprocedural Analysis

- Combined allocation site and access path abstraction for heap
- Liveness analysis of heap data
- Liveness-based points-to analysis Next Topic
- Synergistic program analysis
- Partially path-sensitive analysis

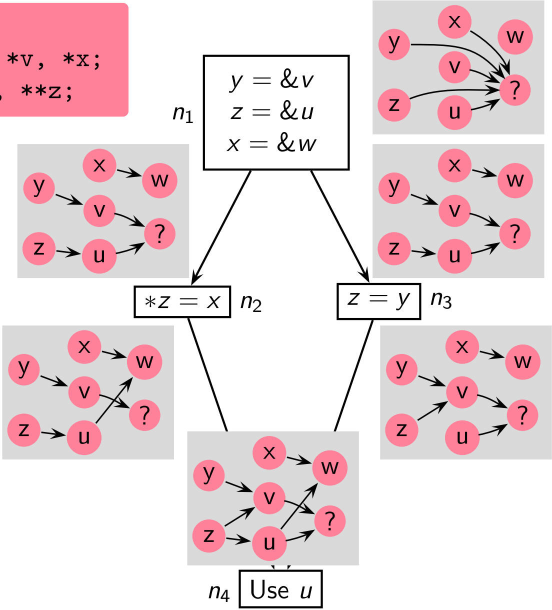
Our Motivating Example for FCPA

```
int w;
int *u, *v, *x;
int **y, **z;
```



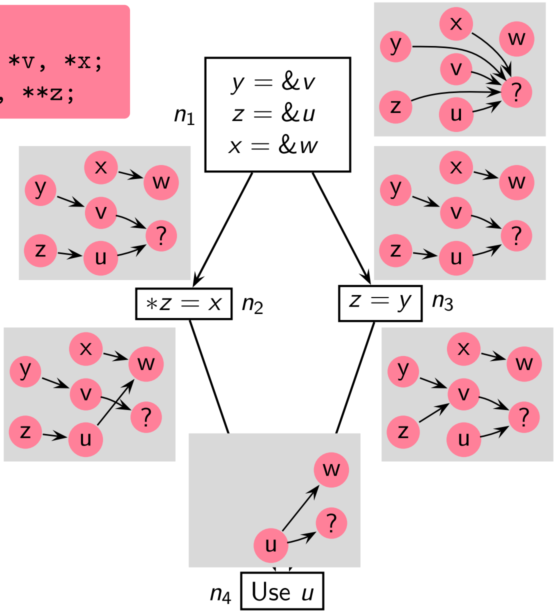
Is All This Information Useful

```
int w;
int *u, *v, *x;
int **y, **z;
```



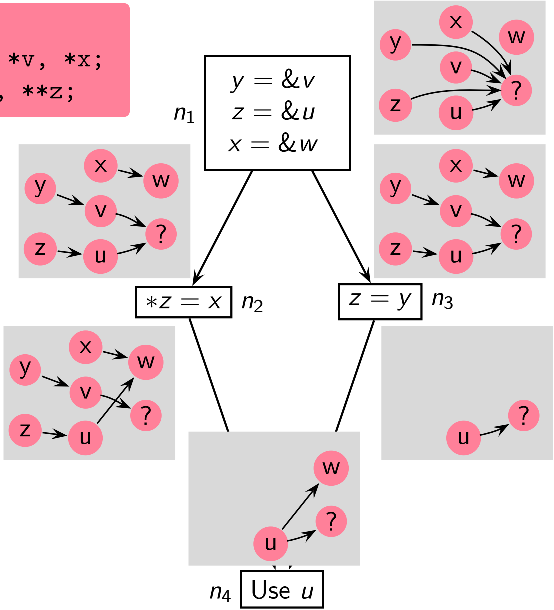
Is All This Information Useful

```
int w;
int *u, *v, *x;
int **y, **z;
```



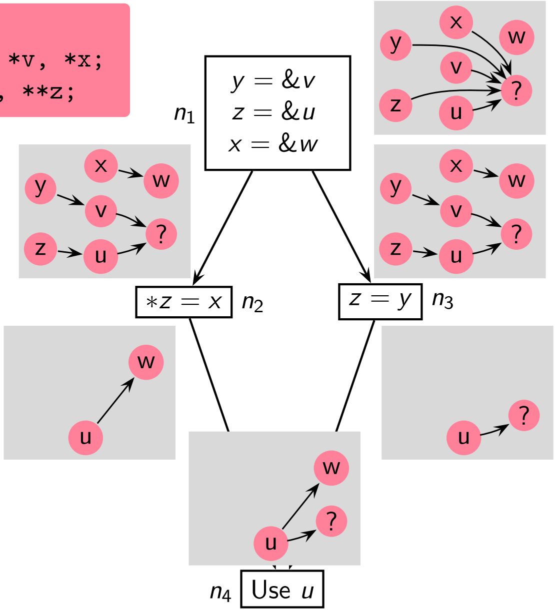
Is All This Information Useful

```
int w;
int *u, *v, *x;
int **y, **z;
```



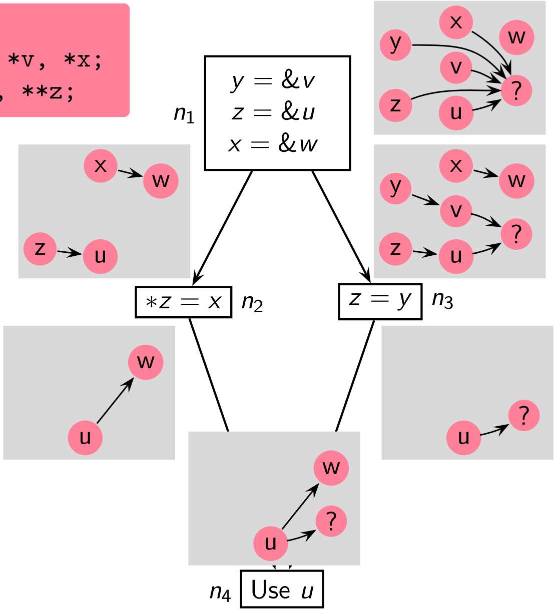
Is All This Information Useful

```
int w;
int *u, *v, *x;
int **y, **z;
```



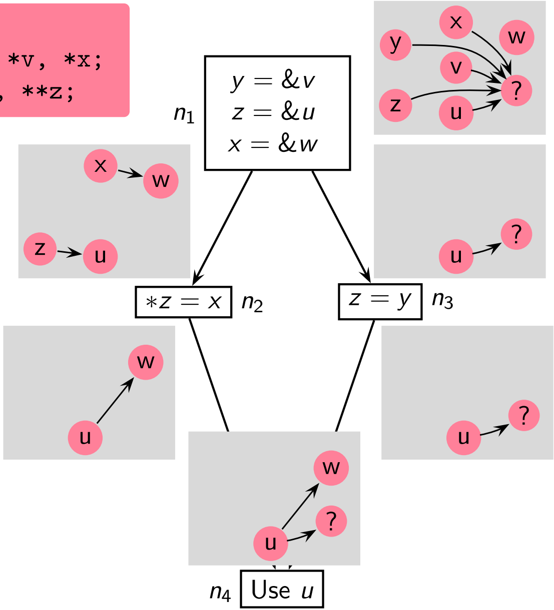
Is All This Information Useful

```
int w;
int *u, *v, *x;
int **y, **z;
```



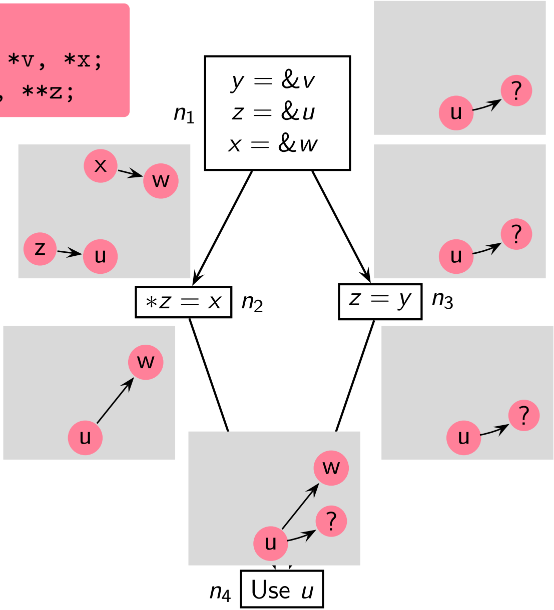
Is All This Information Useful

```
int w;
int *u, *v, *x;
int **y, **z;
```



Is All This Information Useful

```
int w;
int *u, *v, *x;
int **y, **z;
```



Liveness-Based Points-to Analysis (LFCPA) [SAS12]

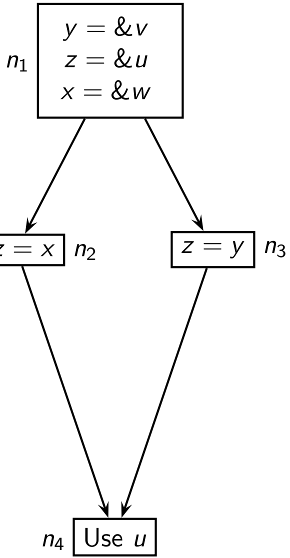
- Mutual dependence of liveness and points-to information
  - Define points-to information only for live pointers
  - For pointer indirections, define liveness information using points-to information
- Use strong liveness
  - Use of a pointer in a non-assignment statement
  - Indirect pointer assignment statement

Motivating Example Revisited [SAS12]

- For convenience, we show complete sweeps of liveness and points-to analysis repeatedly
- This is not required by the computation
- The data flow equations define a single fixed point computation

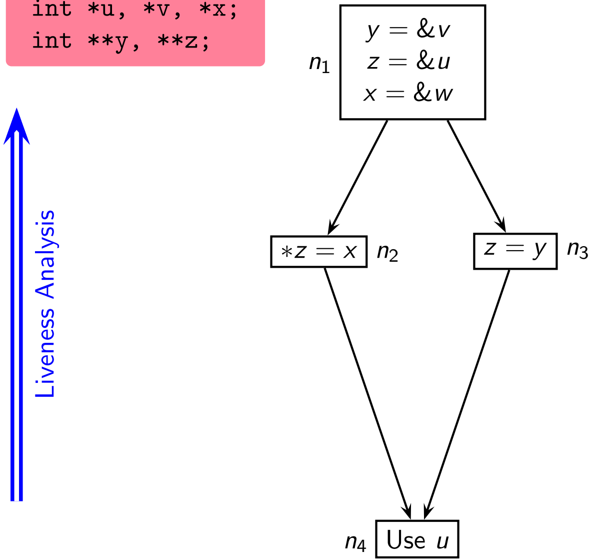
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



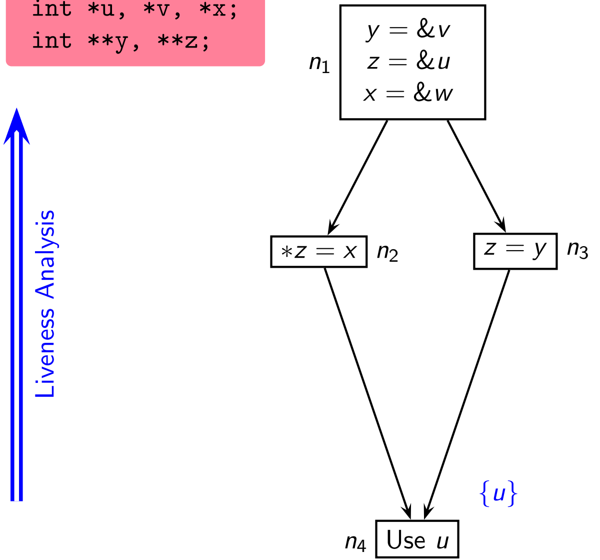
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

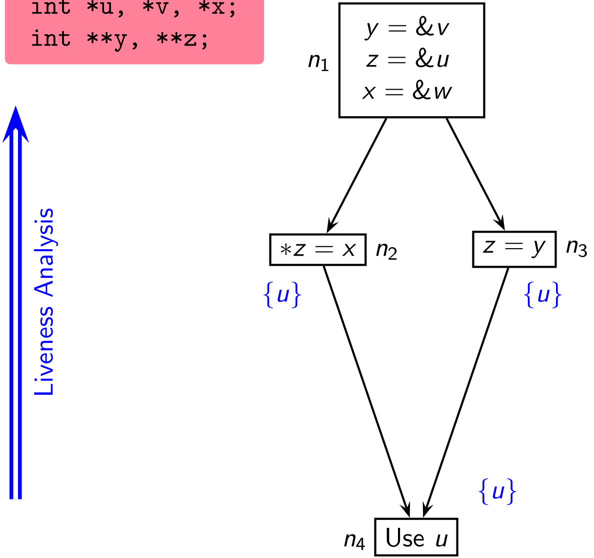
Interprocedural Analysis

Conclusions

References

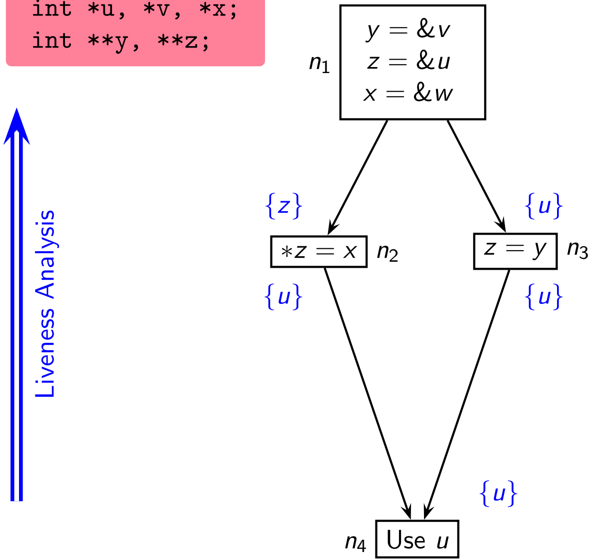
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



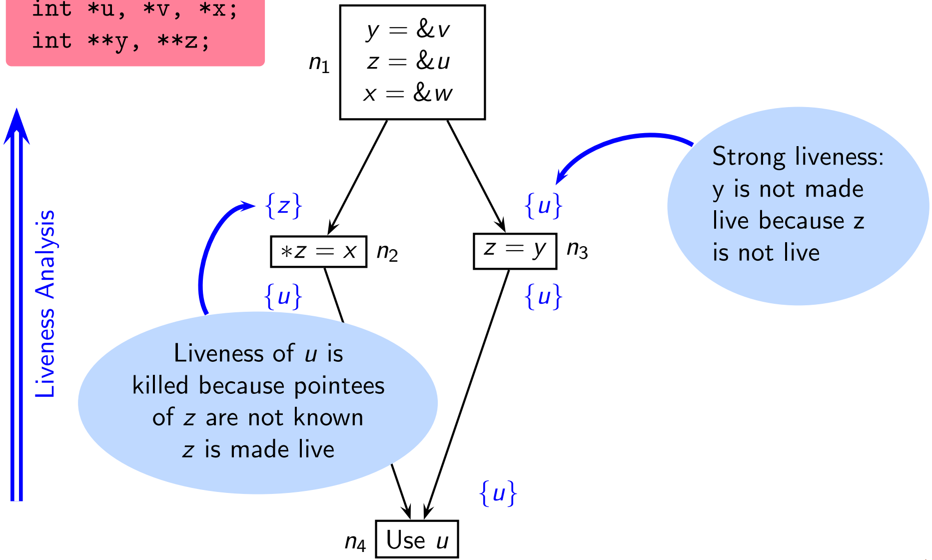
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



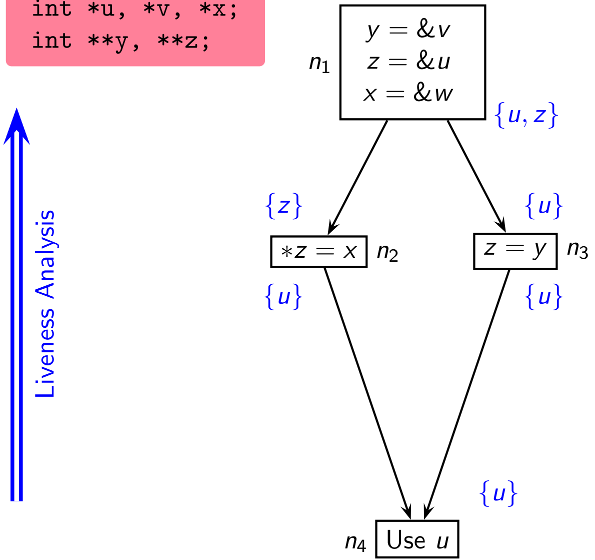
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



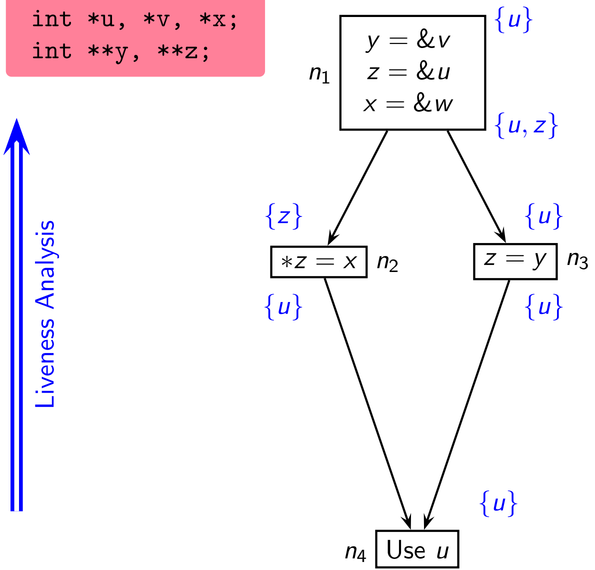
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



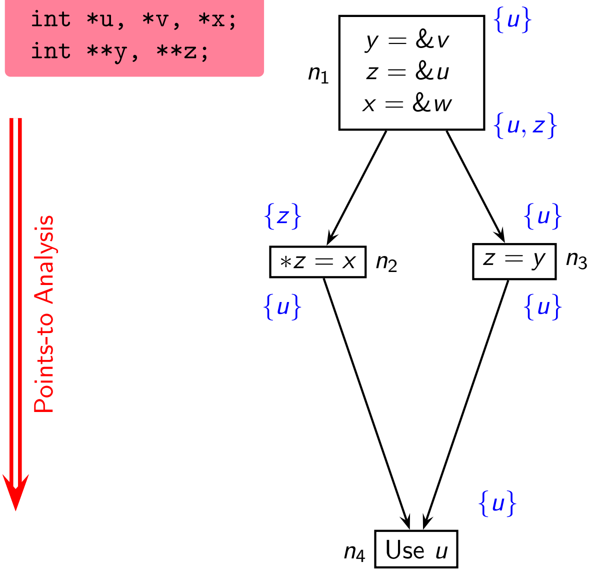
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



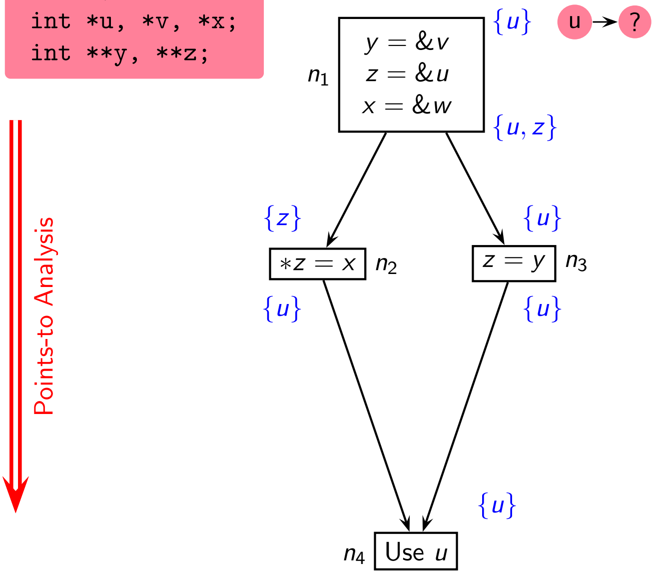
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



First Round of Liveness Analysis and Points-to Analysis

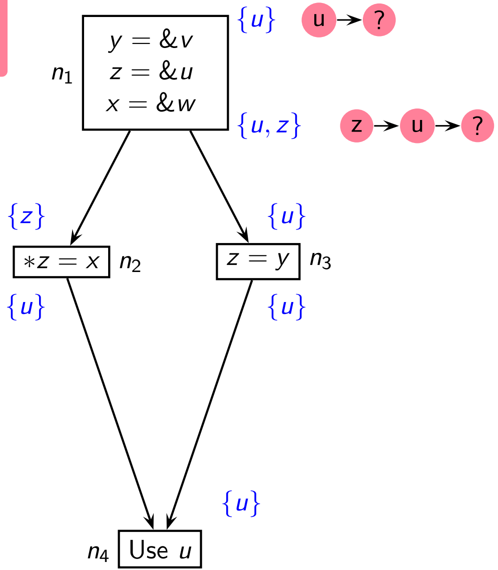
```
int w;
int *u, *v, *x;
int **y, **z;
```



Uday Khedker
IIT Bombay

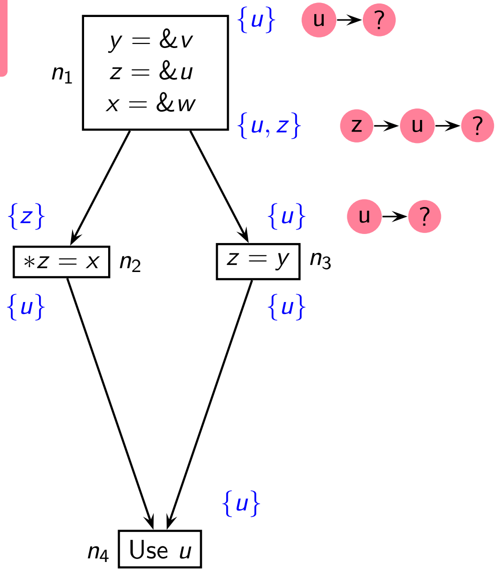
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



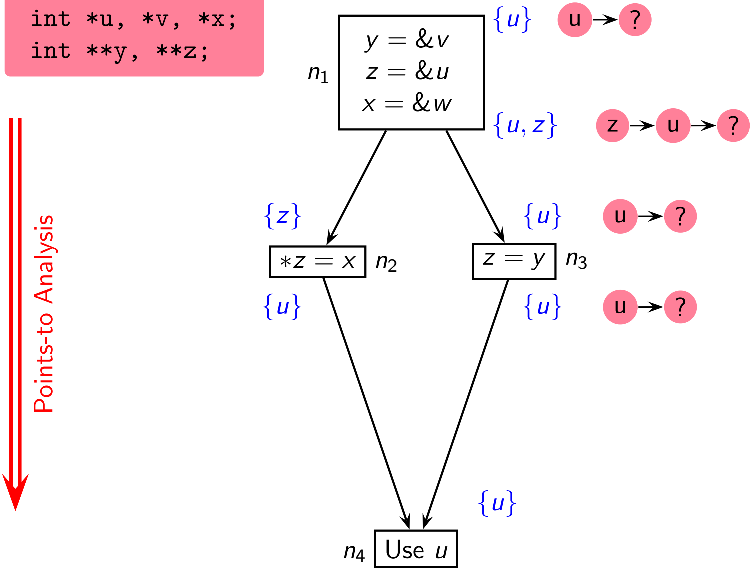
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

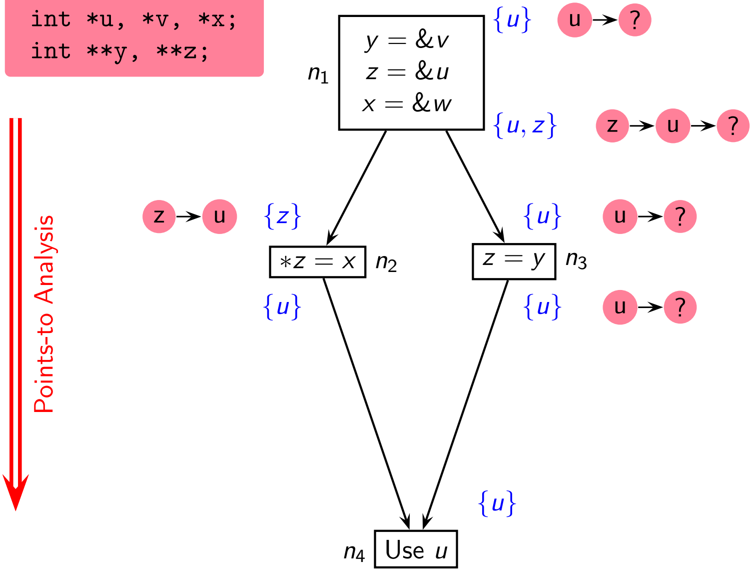
Interprocedural Analysis

Conclusions

References

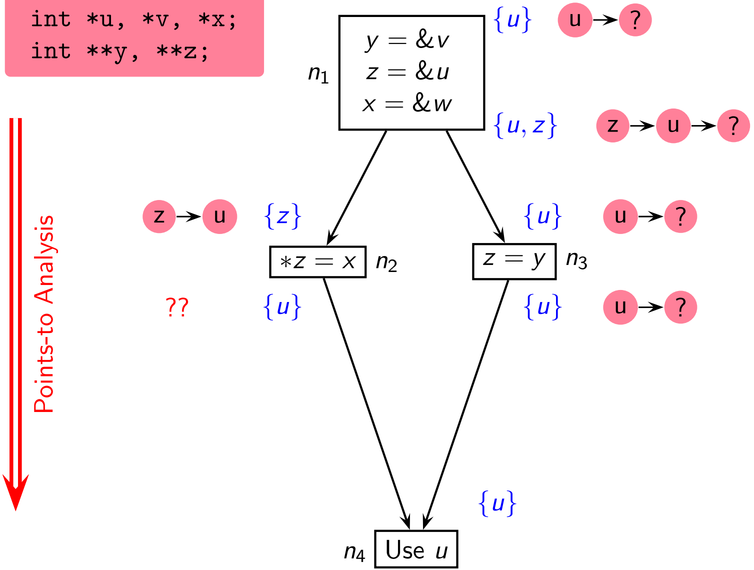
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



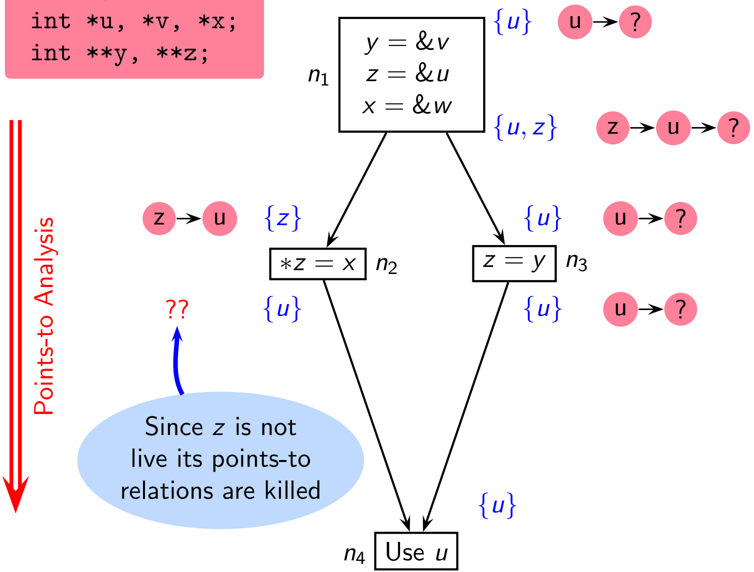
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

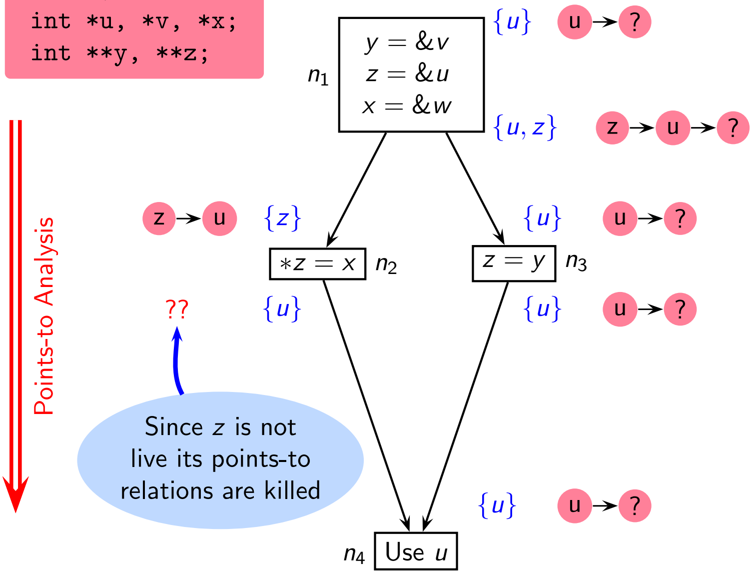
Interprocedural Analysis

Conclusions

References

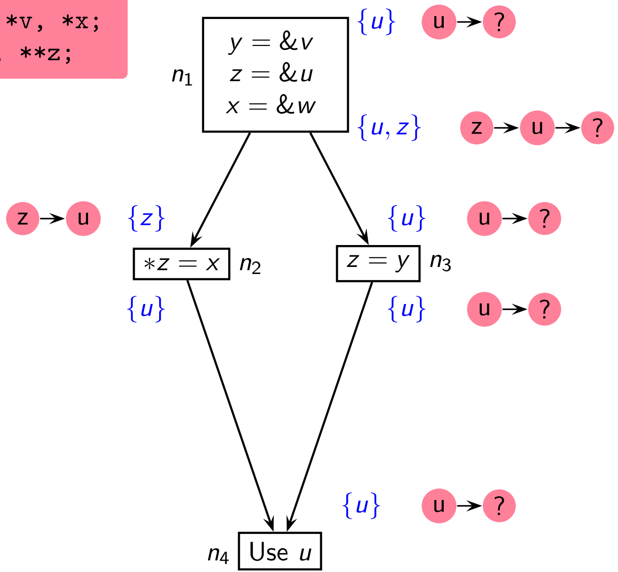
First Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



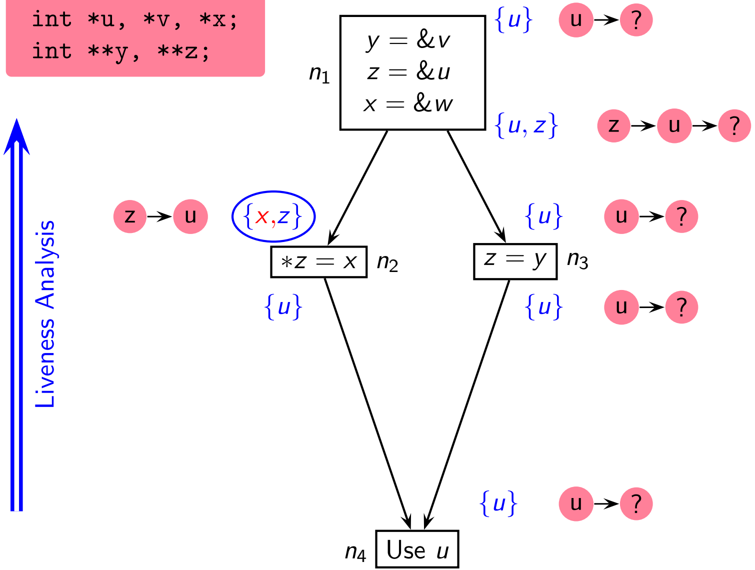
Second Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



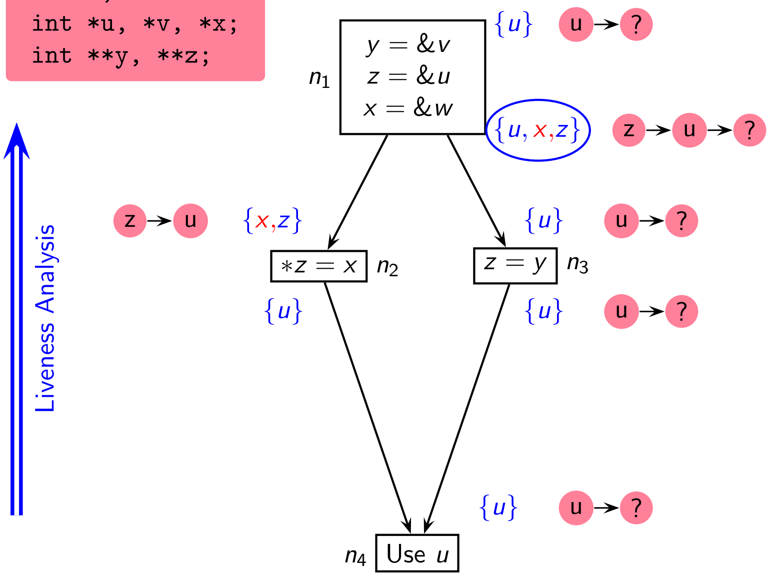
Second Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



Second Round of Liveness Analysis and Points-to Analysis

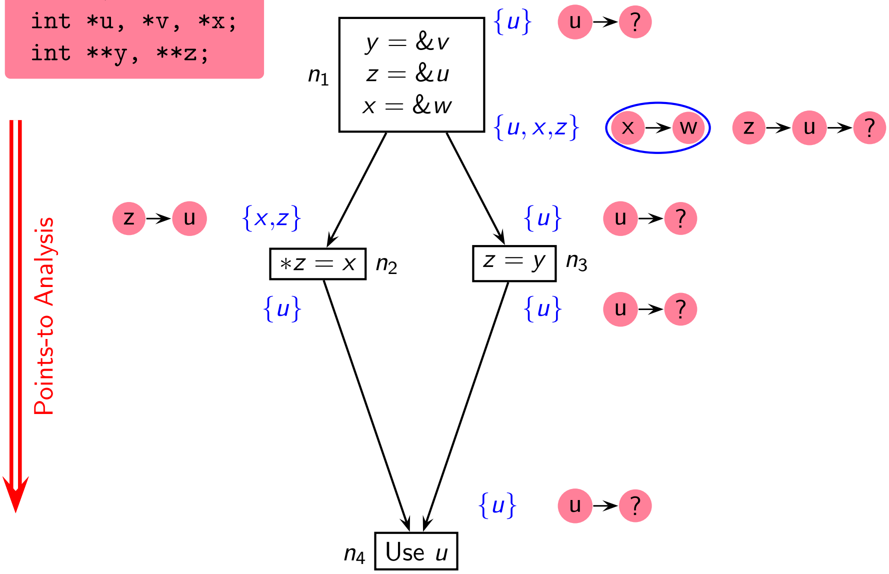
```
int w;
int *u, *v, *x;
int **y, **z;
```



Uday Khedker
IIT Bombay

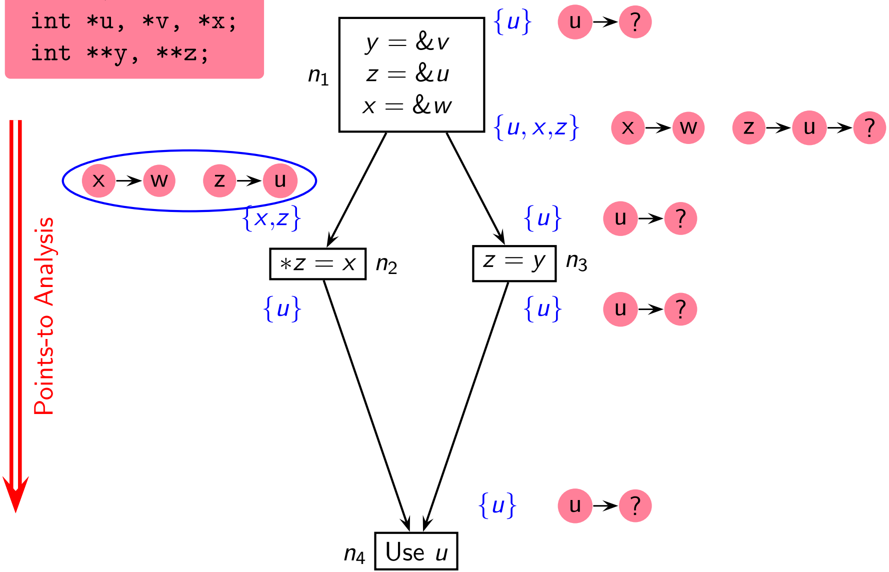
Second Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



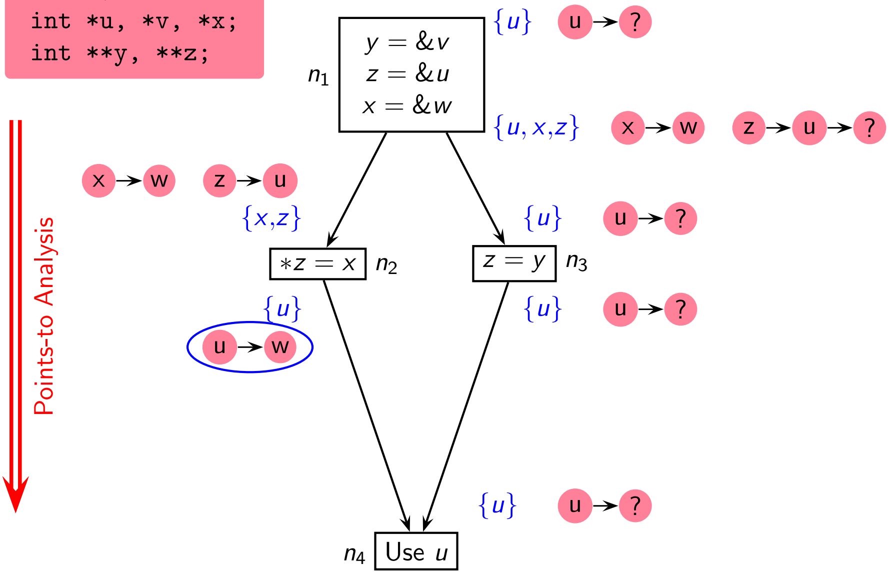
Second Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



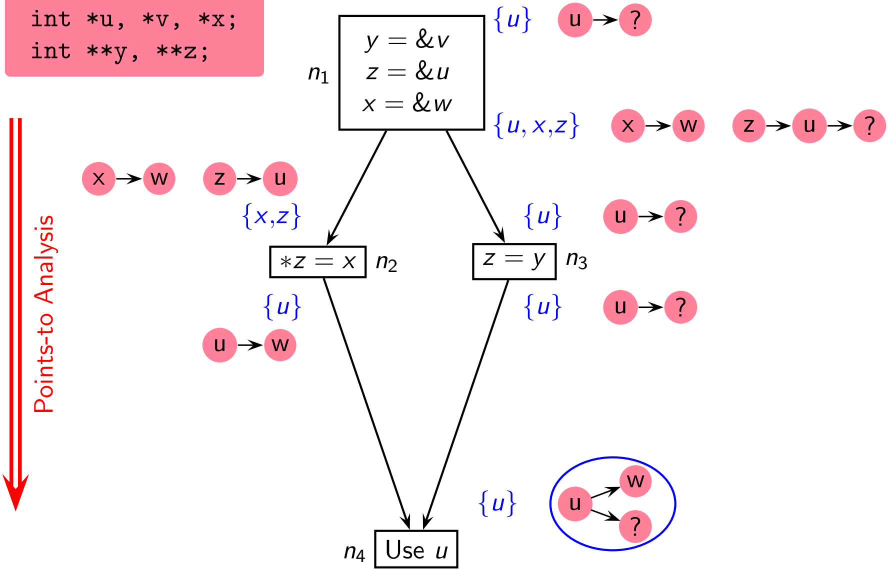
Second Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



Second Round of Liveness Analysis and Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```



LFCPA Observations [SAS12]

- Usable pointer information is very small and sparse
- Data flow propagation in real programs seems to involve only a small subset of all possible data flow values
- Earlier approaches reported inefficiency and non-scalability because they computed far more information than the actual usable information

LFCPA Measurements

- Observations on SPEC CPU 2006 benchmarks in GCC 4.6.0 (Prashant Singh Rawat, IITB 2012)

  Usable pointer information is small and sparse

No of Points-to pairs	Percentage of basic blocks
0	64-96%
1-4	9-25%
5-8	0-10%
8+	0-4%

LFCPA Measurements

- Observations on SPEC CPU 2006 benchmarks in GCC 4.6.0 (Prashant Singh Rawat, IITB 2012)

Usable pointer information is small and sparse

No of Points-to pairs	Percentage of basic blocks
0	64-96%
1-4	9-25%
5-8	0-10%
8+	0-4%

LFCPA Measurements

- Observations on SPEC CPU 2006 benchmarks in GCC 4.6.0 (Prashant Singh Rawat, IITB 2012)

  Usable pointer information is small and sparse

No of Points-to pairs	Percentage of basic blocks
0	64-96%
1-4	9-25%
5-8	0-10%
8+	0-4%

- Independently implemented and verified in
  - LLVM (Dylan McDermott, Cambridge, 2016) and
  - GCC 4.7.2 (Priyanka Sawant, IITB, 2016)

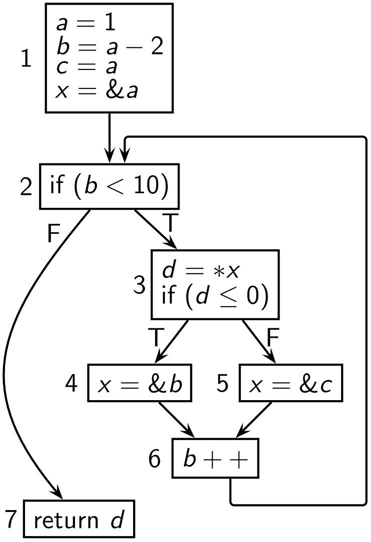
Research Explorations in Intraprocedural Analysis

- Combined allocation site and access path abstraction for heap
- Liveness analysis of heap data
- Liveness-based points-to analysis
- Synergistic program analysis Next Topic
- Partially path-sensitive analysis

Collaboration Between Constant Propagation and Points-to Analysis [WiP]

```
int main()
{  int a, b, c, d, *x;
    a=1; b=a-2; c=a;
    x=&a;
    while (b<10)
    {  d=*x;
        if (d<=0) x=&b;
        else x=&c;
        b++;
    }
    return d;
}
```

The value of d in the loop is 1, the condition fails, and x does not point to b at any time



Collaboration Between Constant Propagation and Points-to Analysis [WiP]

We have three options to enable interaction (illustrated next)

- Conventional Cascading. Perform analyses in a fixed sequence
    - CP $\rightarrow$ transform the program $\rightarrow$ PTA
    - PTA $\rightarrow$ transform the program $\rightarrow$ CP

  This method fails on our example

- Simultaneous Analyses (Lerner's method)

  Perform CP and PTA in locked steps and transform the program whenever possible, repeat the analyses as long as transformations are possible

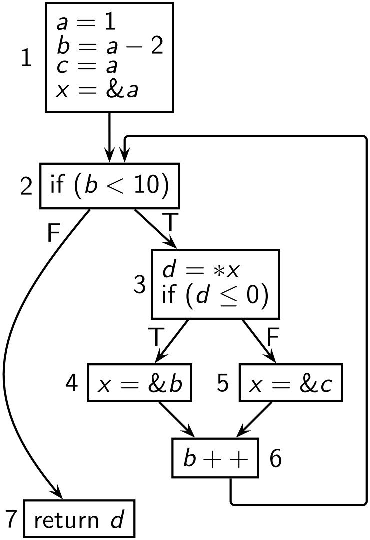
  This method also fails on our example

- Interleaved Synergistic Program Analysis (SPAN)

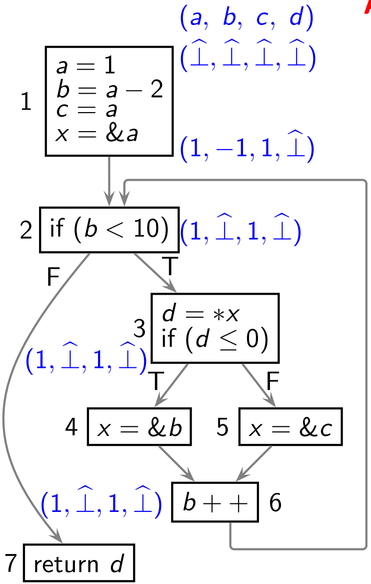
  Interleave the analyses on a need basis, use data flow values to achieve the effect of transforming the program (without actually transforming it)

  This method succeeds on our example

Collaboration Between Constant Propagation and Points-to Analysis [WiP]



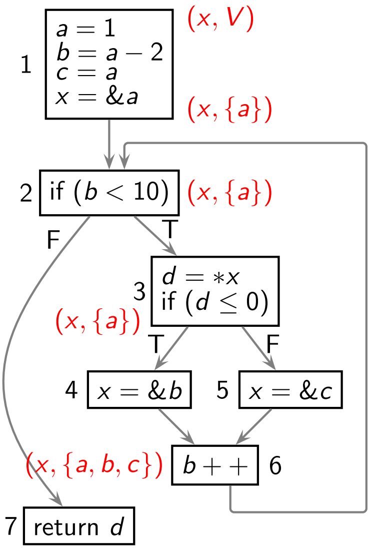
Collaboration Between Constant Propagation and Points-to Analysis [WiP]



If we perform constant propagation first,

- We do not know the pointees of x in node 3, hence we assume all variables as possible pointees
- Thus the value of d is $\perp$ and the branch outcome is uncertain and no path is ruled out

Collaboration Between Constant Propagation and Points-to Analysis [WiP]



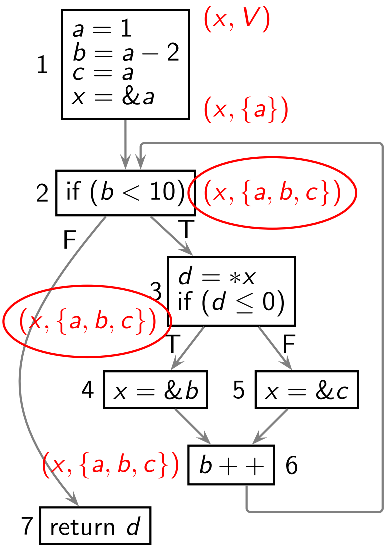
If we perform constant propagation first,

- We do not know the pointees of x in node 3, hence we assume all variables as possible pointees
- Thus the value of d is $\perp$ and the branch outcome is uncertain and no path is ruled out

Then, when we perform points-to analysis,

- The pointees of x are found to be a, b, and c
- $d = *x$ cannot be simplified
- A subsequent round of constant propagation will find d to be $\perp$

Collaboration Between Constant Propagation and Points-to Analysis [WiP]



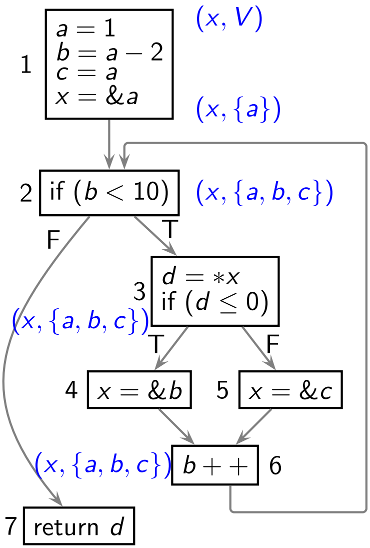
If we perform constant propagation first,

- We do not know the pointees of x in node 3, hence we assume all variables as possible pointees
- Thus the value of d is $\perp$ and the branch outcome is uncertain and no path is ruled out

Then, when we perform points-to analysis,

- The pointees of x are found to be a, b, and c
- $d = *x$ cannot be simplified
- A subsequent round of constant propagation will find d to be $\perp$

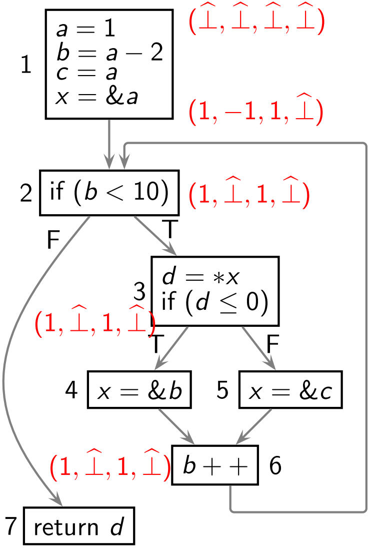
Collaboration Between Constant Propagation and Points-to Analysis [WiP]



If we perform points-to analysis first,

- The pointees of x are found to be a, b, and c because both branch outcomes are possible
- $d = *x$ cannot be simplified

Collaboration Between Constant Propagation and Points-to Analysis [WiP]



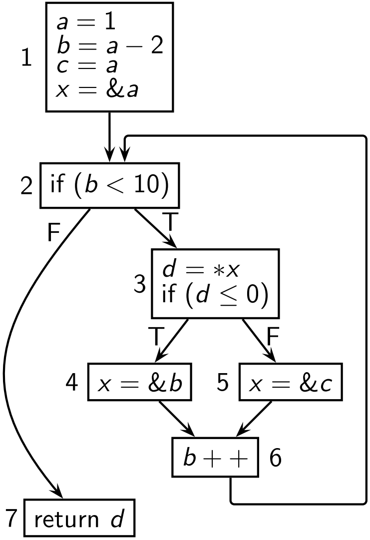
If we perform points-to analysis first,

- The pointees of x are found to be a, b, and c because both branch outcomes are possible
- $d = *x$ cannot be simplified

Then, when we perform constant propagation

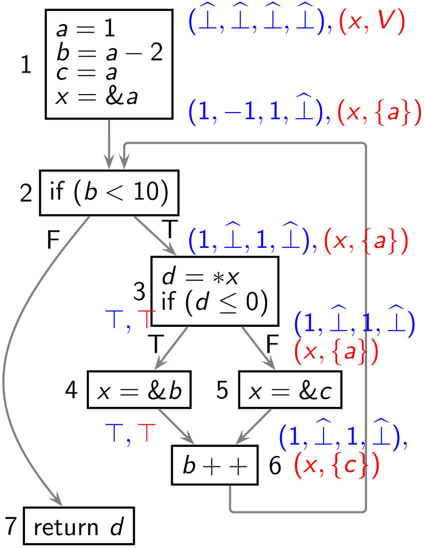
- The value of d is $\bot$ and the branch outcome is uncertain and no path is ruled out
- A subsequent round of points-to analysis will find the pointees of x as a, b and c

Collaboration Between Constant Propagation and Points-to Analysis [WiP]



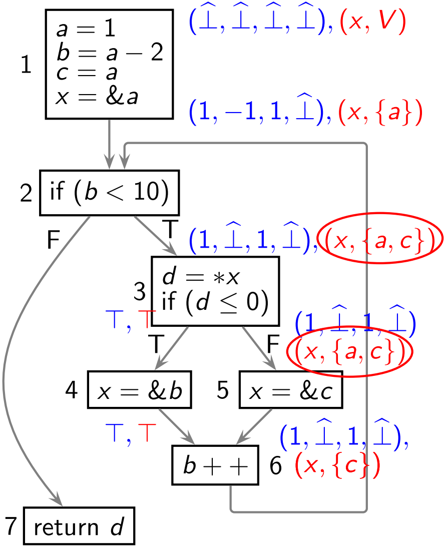
- The precision of the two analyses depends on each other's results
- If we perform them together, we can rule out the T branch out of node 3, x points to a and c, and both are 1

Collaboration Between Constant Propagation and Points-to Analysis [WiP]



- The precision of the two analyses depends on each other's results
- If we perform them together, we can rule out the T branch out of node 3, x points to a and c, and both are 1
- SPAN achieves this

Collaboration Between Constant Propagation and Points-to Analysis [WiP]



- The precision of the two analyses depends on each other's results
- If we perform them together, we can rule out the T branch out of node 3, x points to a and c, and both are 1
- SPAN achieves this

Collaboration Between Constant Propagation and Points-to Analysis [WiP]

SPAN is more general than Lerner's method because

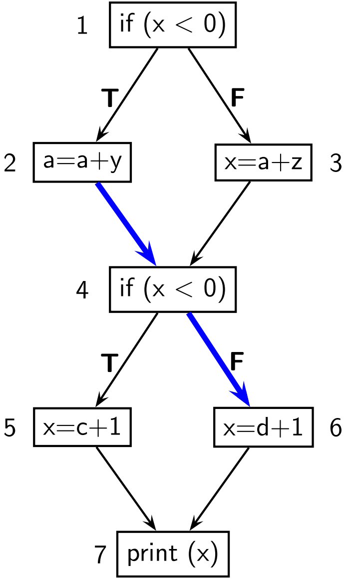
- SPAN does not transform the program but uses data flow values (Lerner's method tries to transform $d = *x$ and fails)
- The analyses need not be performed in locked steps and hence forward and backward analyses can be combined
- The need of interaction is inferred automatically and the user does not need to specify it
- Arbitrary data flow analyses can be added to the system at will

  Each analysis must specify the statements that it can (conceptually) simplify and the statements that it cannot simplify

Research Explorations in Intraprocedural Analysis

- Combined allocation site and access path abstraction for heap
- Liveness analysis of heap data
- Liveness-based points-to analysis
- Synergistic program analysis
- Partially path-sensitive analysis Next Topic

Excluding the Known Infeasible Paths [CC18, CC19]



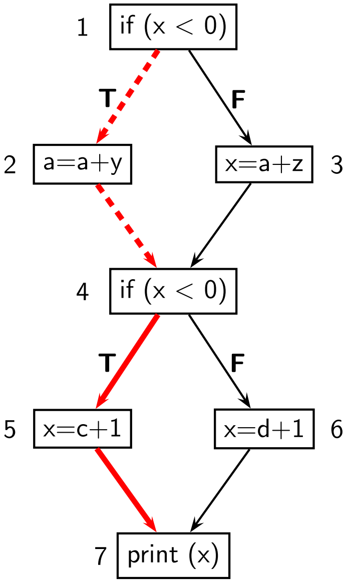
- Every path containing $\rho : (2, 4, 6)$ is infeasible

  It could lead to imprecision (e.g. *d* is spuriously marked live at the exit of node 2)

- We cannot delete any edge to exclude this path

Such deletion could lead to unsoundness

Excluding the Known Infeasible Paths [CC18, CC19]



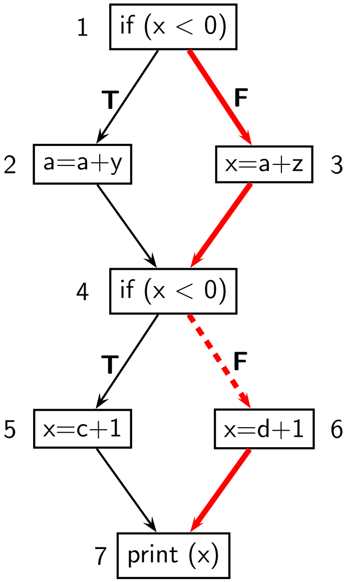
- Every path containing $\rho: (2,4,6)$ is infeasible

  It could lead to imprecision (e.g. *d* is spuriously marked live at the exit of node 2)

- We cannot delete any edge to exclude this path
  - If we delete edge $(2,4)$, it excludes a feasible path $(1,2,4,5,7)$

Such deletion could lead to unsoundness

Excluding the Known Infeasible Paths [CC18, CC19]



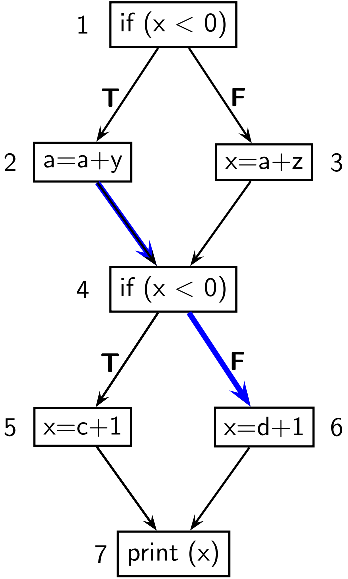
- Every path containing $\rho:(2,4,6)$ is infeasible

  It could lead to imprecision (e.g. *d* is spuriously marked live at the exit of node 2)

- We cannot delete any edge to exclude this path
  - If we delete edge $(2,4)$, it excludes a feasible path $(1,2,4,5,7)$
  - If we delete edge $(4,6)$, it excludes a feasible path $(1,3,4,6,7)$

  Such deletion could lead to unsoundness

Excluding the Known Infeasible Paths [CC18, CC19]



- Every path containing $\rho:(2,4,6)$ is infeasible

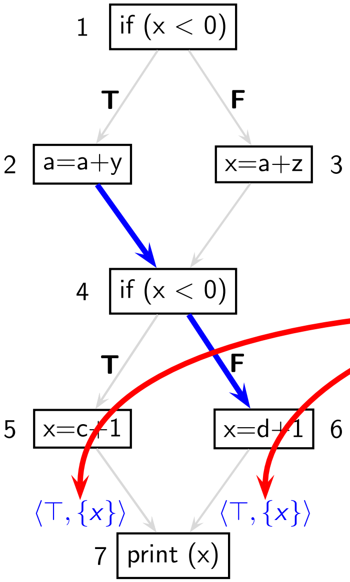
  It could lead to imprecision (e.g. *d* is spuriously marked live at the exit of node 2)

- We cannot delete any edge to exclude this path
  - If we delete edge $(2,4)$, it excludes a feasible path $(1,2,4,5,7)$
  - If we delete edge $(4,6)$, it excludes a feasible path $(1,3,4,6,7)$

  Such deletion could lead to unsoundness

- Our solution: At each edge, distinguish the data flow value of ρ from other values so that it is not allowed to go out of ρ on an infeasible path

Excluding the Known Infeasible Paths [CC18, CC19]

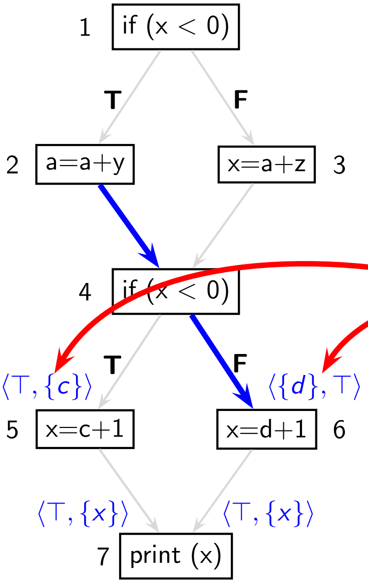


Our Notation: $\langle$dfv ofρ, other dfv$\rangle$

Edges 5 $\rightarrow$ 7 and 6 $\rightarrow$ 7 are not a part of ρ

Hence the data flow value in the first component is $\top$

Excluding the Known Infeasible Paths [CC18, CC19]

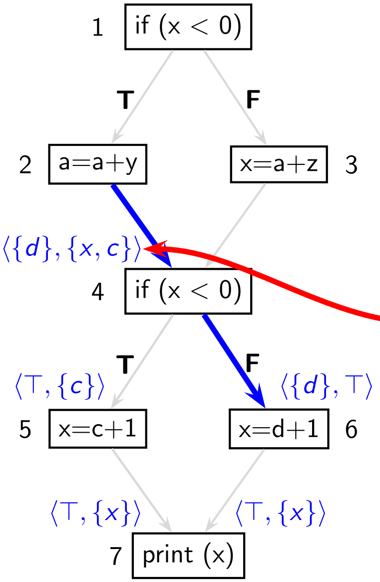


Our Notation: $\langle$dfv ofρ, other dfv$\rangle$

Edge $4 \rightarrow 5$ is not a part of ρ and the first component is $\top$

Edge $4 \rightarrow 6$ is a part of ρ but edge $6 \rightarrow 7$ is not a part of ρ (i.e. the effect of ρ begins here) so the data flow value shifts from the second component to the first component

Excluding the Known Infeasible Paths [CC18, CC19]

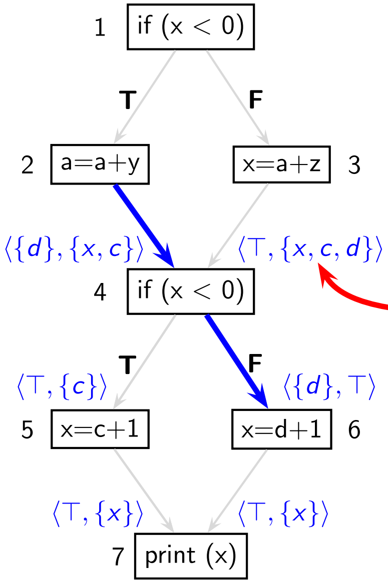


Our Notation: $\langle$dfv ofρ, other dfv$\rangle$

Edge $2 \rightarrow 4$ is a part of ρ hence it will continue to hold the data flow value of ρ coming from edge $4 \rightarrow 6$ which is also a part of ρ

The data flow value generated in node 4 or the data flow value coming from edge $4 \rightarrow 5$ go to the second component because a path that does not include ρ completely, is not infeasible

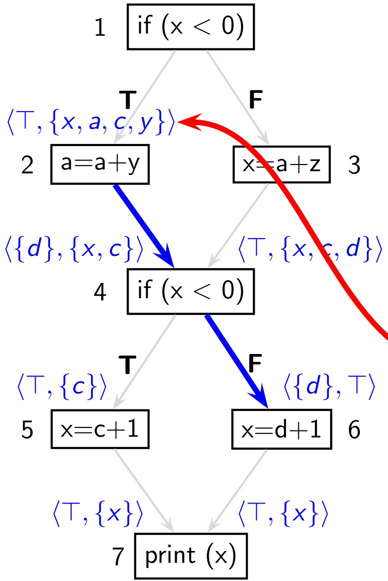
Excluding the Known Infeasible Paths [CC18, CC19]



Our Notation: $\langle \text{dfv of} \rho, \text{other dfv} \rangle$

Edge $3 \rightarrow 4$ is not a part of ρ hence the first component is $\top$ and all data flow values move to the second component

Excluding the Known Infeasible Paths [CC18, CC19]



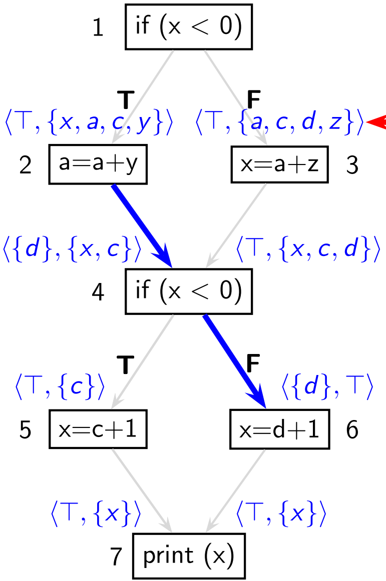
Our Notation: $\langle$dfv ofρ, other dfv$\rangle$

Edge $1 \rightarrow 2$ is not a part of ρ hence the first component is $\top$ and all data flow values move to the second component

Since d belongs to ρ, it is blocked and is not propagated further because the path $(1, 2, 4, 6, 7)$ is infeasible

This separation and blocking of values gives a more precise solution than the usual MFP solution

Excluding the Known Infeasible Paths [CC18, CC19]



Our Notation: $\langle$dfv ofρ, other dfv$\rangle$

Edge 1 $\rightarrow$ 3 is not a part of ρ hence the first component is $\top$ and all data flow values move to the second component

Excluding the Known Infeasible Paths [CC18, CC19]

- Infeasibility is a property of the control flow graph and not that of an analysis
- Our method takes as input the information about the minimal infeasible path segments in program
- Our method is very general
  - It handles multiple minimal infeasible path segments that may overlap with each other
  - It lifts any data flow analysis to an analysis that excludes the effect of known infeasible paths
- Existing approaches to remove the effect of infeasible paths are either analysis specific or involve CFG restructuring

  Our approach avoids CFG restructuring and still achieves a generic solution

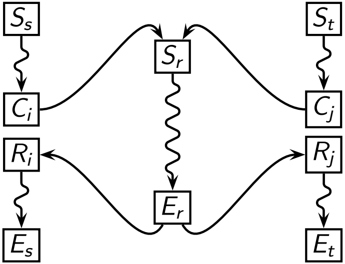
Interprocedural Analysis

Research Explorations in Interprocedural Analysis

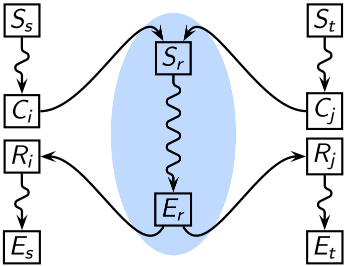
- Broad categories of interprocedural analysis Next Topic
- Scaling top-down analysis using value contexts and bypassing
- A unified model of context-sensitive methods
- Improving bottom-up analysis by eliminating control flow
- Precise virtual call resolution with demand-driven analysis
- Improving call graphs using callee contexts

Skip Section

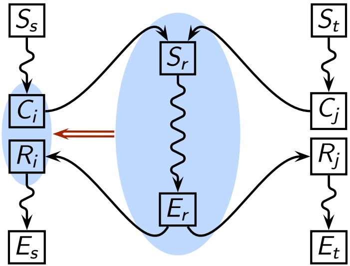
Understanding Context Sensitivity



Understanding Context Sensitivity

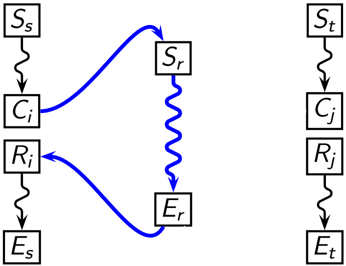


Understanding Context Sensitivity



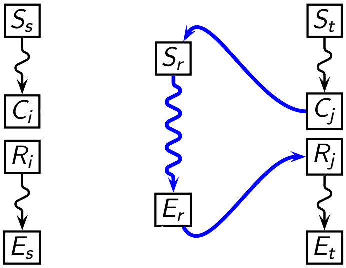
Precise interprocedural analysis aims to achieve the effect of inlining

Understanding Context Sensitivity



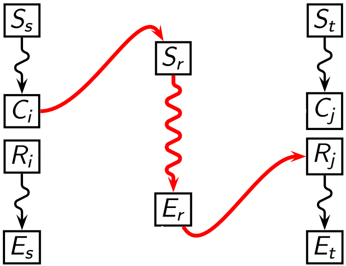
Interprocedurally valid path

Understanding Context Sensitivity



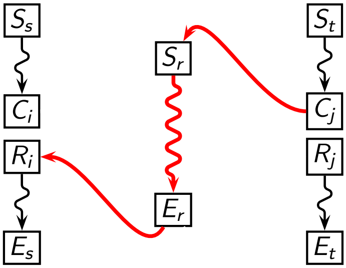
Interprocedurally valid path

Understanding Context Sensitivity



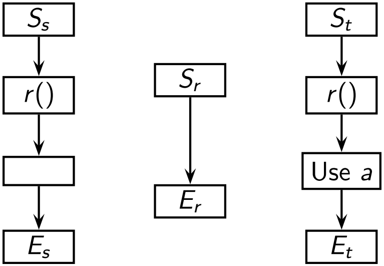
Interprocedurally invalid path

Understanding Context Sensitivity

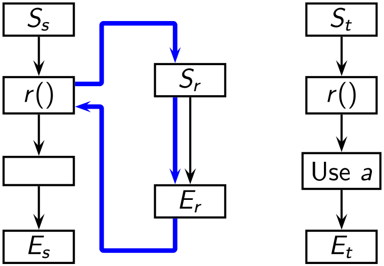


Interprocedurally invalid path

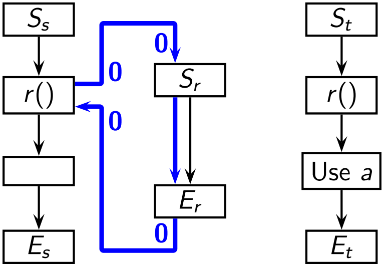
Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

Context-sensitive Analysis	Context-insensitive Analysis
	
Data flow values of distinct contexts are kept as separate values	Data flow values of all contexts are merged into a single value

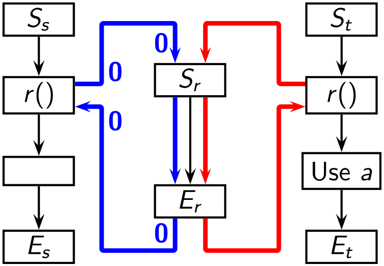
Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

Context-sensitive Analysis	Context-insensitive Analysis
	
Data flow values of distinct contexts are kept as separate values	Data flow values of all contexts are merged into a single value

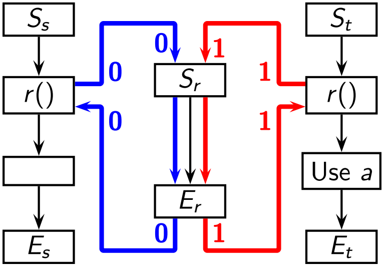
Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

Context-sensitive Analysis	Context-insensitive Analysis
	
Data flow values of distinct contexts are kept as separate values	Data flow values of all contexts are merged into a single value

Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

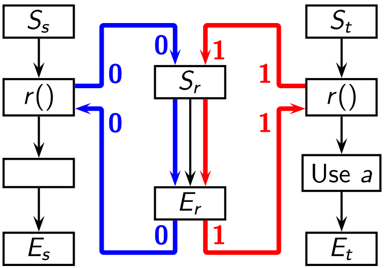
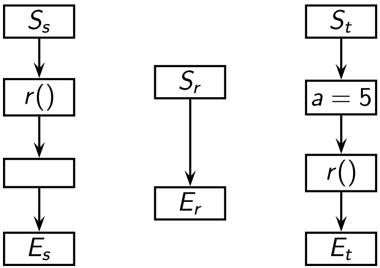
Context-sensitive Analysis	Context-insensitive Analysis
	
Data flow values of distinct contexts are kept as separate values	Data flow values of all contexts are merged into a single value

Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

Context-sensitive Analysis	Context-insensitive Analysis
	
Data flow values of distinct contexts are kept as separate values	Data flow values of all contexts are merged into a single value

Uday Khedker
IIT Bombay

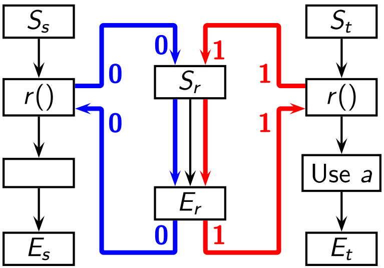
Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

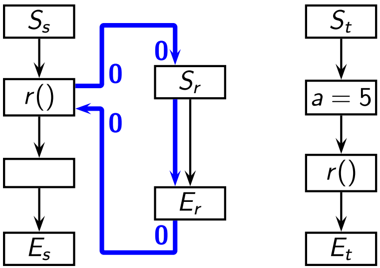
Context-sensitive Analysis	Context-insensitive Analysis
	
Data flow values of distinct contexts are kept as separate values	Data flow values of all contexts are merged into a single value

Uday Khedker
IIT Bombay

Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

Context-sensitive Analysis	Context-insensitive Analysis

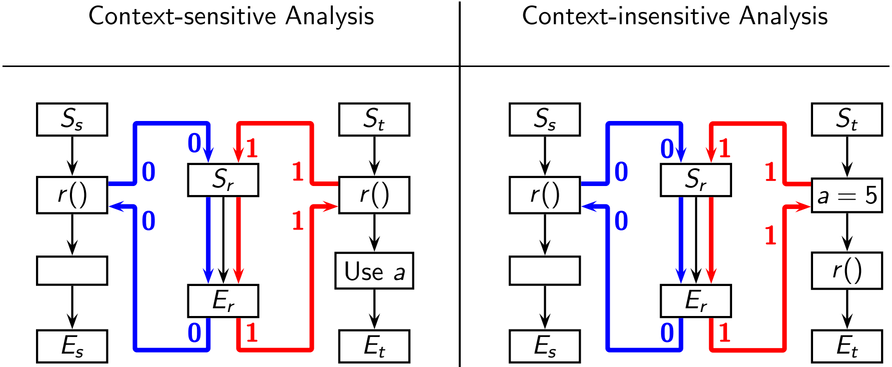




Data flow values of distinct contexts are kept as separate values

Data flow values of all contexts are merged into a single value

Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)



Data flow values of distinct contexts are kept as separate values

Data flow values of all contexts are merged into a single value

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

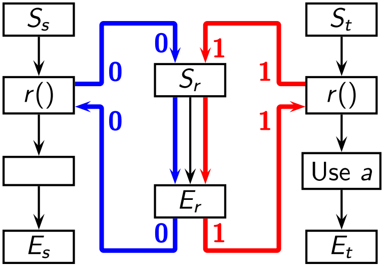
Interprocedural Analysis

Conclusions

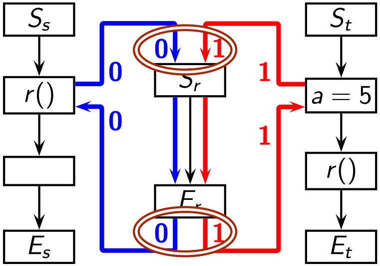
References

Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

Context-sensitive Analysis	Context-insensitive Analysis

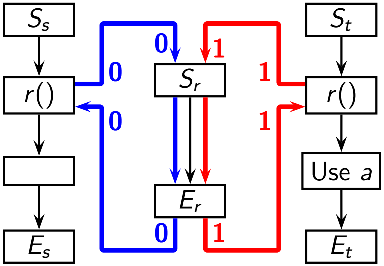
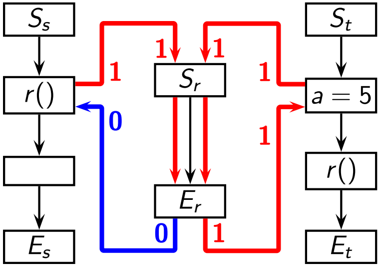


Data flow values of distinct contexts are kept as separate values



Data flow values of all contexts are merged into a single value

Context Sensitivity Vs. Context Insensitivity (Example for Live Variables Analysis)

Context-sensitive Analysis	Context-insensitive Analysis
	
Data flow values of distinct contexts are kept as separate values	Data flow values of all contexts are merged into a single value

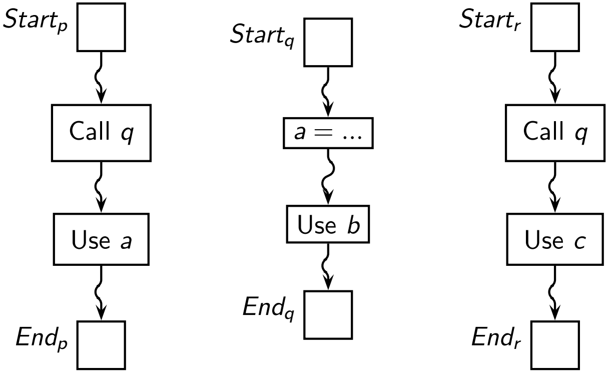
Understanding Context Sensitivity

The effect of inlining is achieved by

- call-return matching (call strings method),
- computing the summary of a procedure and incorporating it at the call point (functional method), or
- analyzing a procedure for a particular data flow value and using the analysed result at the call point (graph reachability, value context method)

Top-down Vs. Bottom-up (Context-Sensitive) Procedure Summaries

Top-down Live Variables Analysis

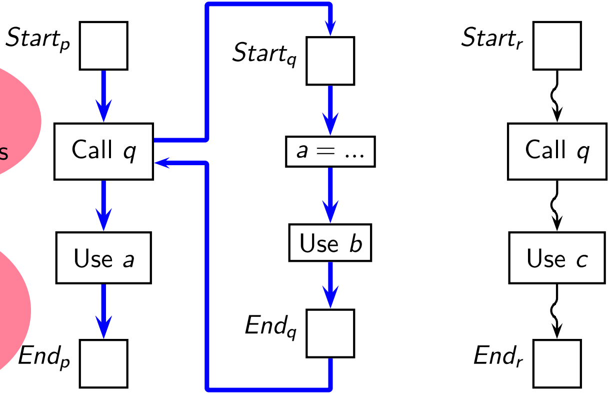


Top-down Vs. Bottom-up (Context-Sensitive) Procedure Summaries

Top-down Live Variables Analysis

Procedure q is analysed multiple times

Contexts are created explicitly



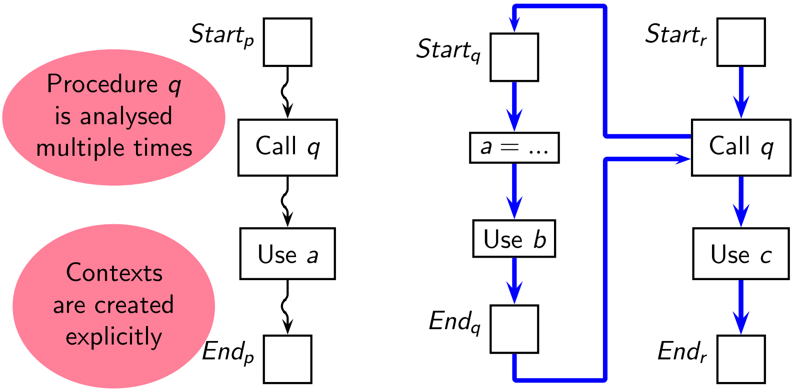
Context σ_1

Variable b is live at S_q and hence at S_p

Variables a and c are not live at S_q and hence not live at S_p

Top-down Vs. Bottom-up (Context-Sensitive) Procedure Summaries

Top-down Live Variables Analysis

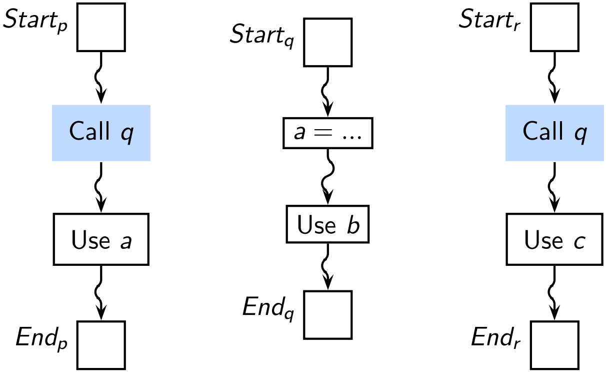


Context σ_2 Variables b and c are live at S_q and hence at S_r

Variable a is not live at S_q and hence not live at S_r

Top-down Vs. Bottom-up (Context-Sensitive) Procedure Summaries

Bottom-Up Live Variables Analysis

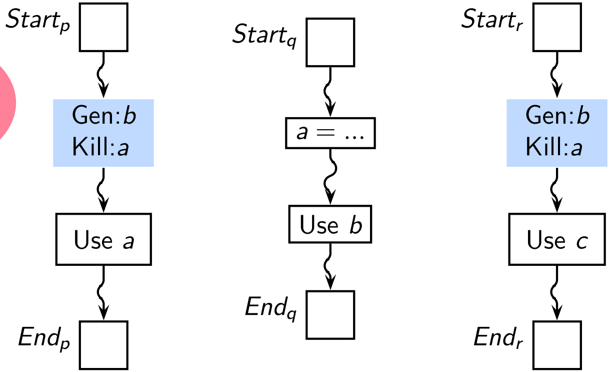


Top-down Vs. Bottom-up (Context-Sensitive) Procedure Summaries

Bottom-Up Live Variables Analysis

Procedure q is analysed once

Contexts are left implicit



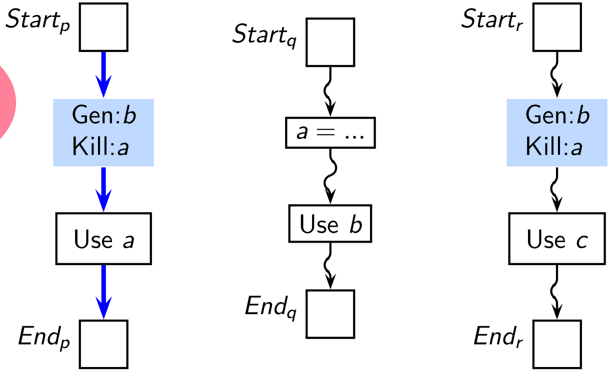
Using procedure summary of q at call sites

Top-down Vs. Bottom-up (Context-Sensitive) Procedure Summaries

Bottom-Up Live Variables Analysis

Procedure q is analysed once

Contexts are left implicit



Variable b is live at S_p

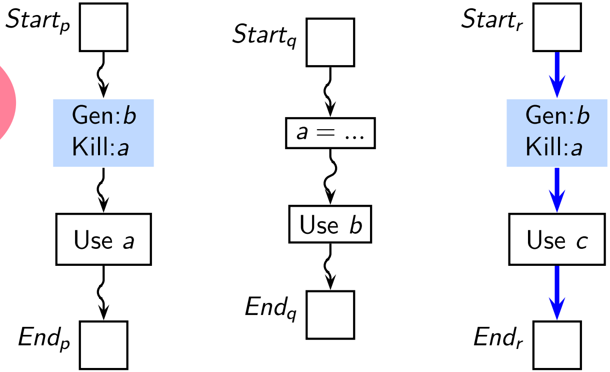
Variables a and c are not live at S_p

Top-down Vs. Bottom-up (Context-Sensitive) Procedure Summaries

Bottom-Up Live Variables Analysis

Procedure q is analysed once

Contexts are left implicit



Variables b and c are live at S_r

Variable a is not live at S_r

Research Explorations in Interprocedural Analysis

- Broad categories of interprocedural analysis
- Scaling top-down analysis using value contexts and bypassing
  Next Topic
- A unified model of context-sensitive methods
- Improving bottom-up analysis by eliminating control flow
- Precise virtual call resolution with demand-driven analysis
- Improving call graphs using callee contexts

Uday Khedker
IIT Bombay

Value Contexts [CC08, SOAP13]

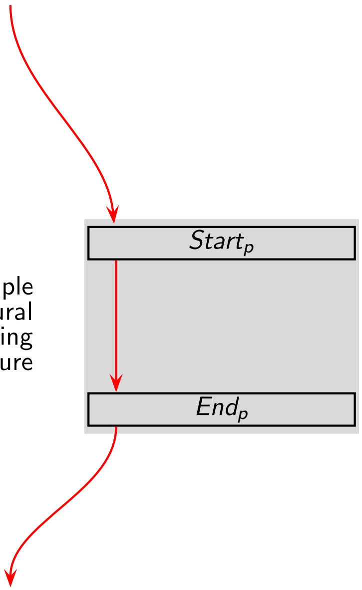
$Start_p$

Procedure
Body

End_p

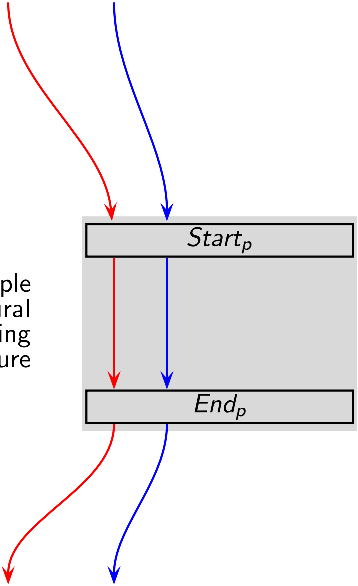
Value Contexts [CC08, SOAP13]

Multiple interprocedural paths reaching the procedure



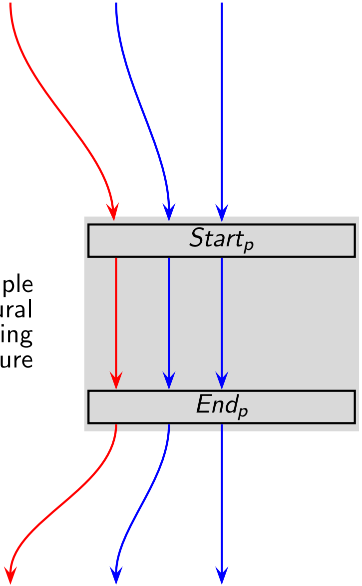
Value Contexts [CC08, SOAP13]

Multiple interprocedural paths reaching the procedure



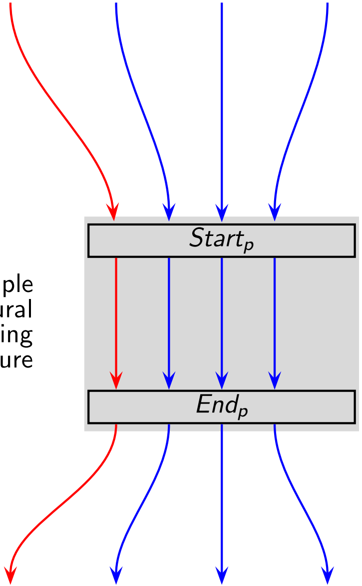
Value Contexts [CC08, SOAP13]

Multiple interprocedural paths reaching the procedure

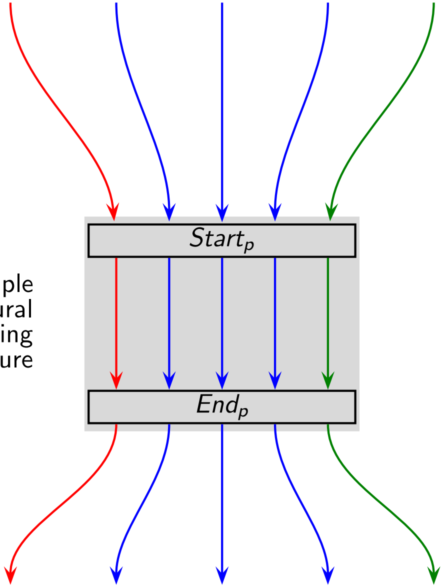


Value Contexts [CC08, SOAP13]

Multiple interprocedural paths reaching the procedure

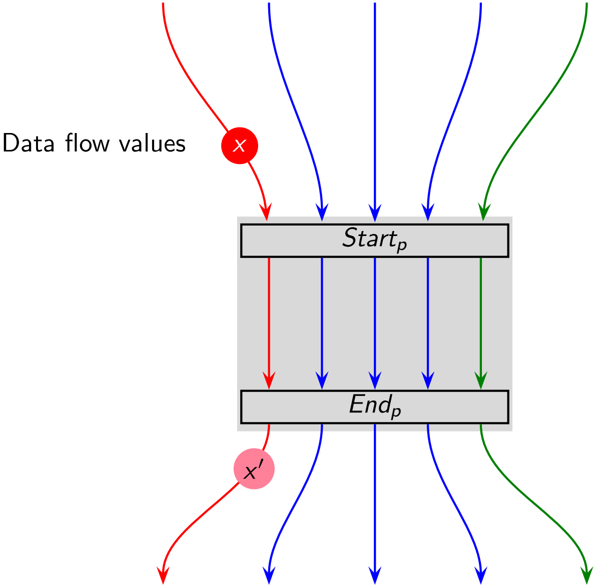


Value Contexts [CC08, SOAP13]

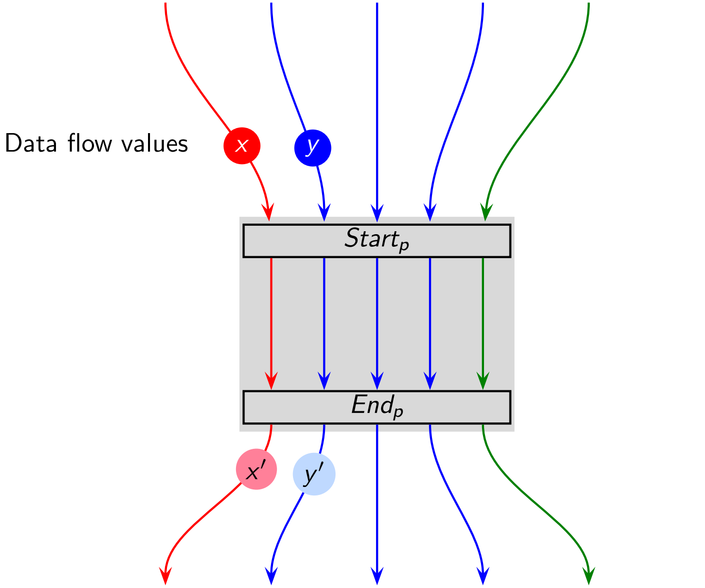


Multiple interprocedural paths reaching the procedure

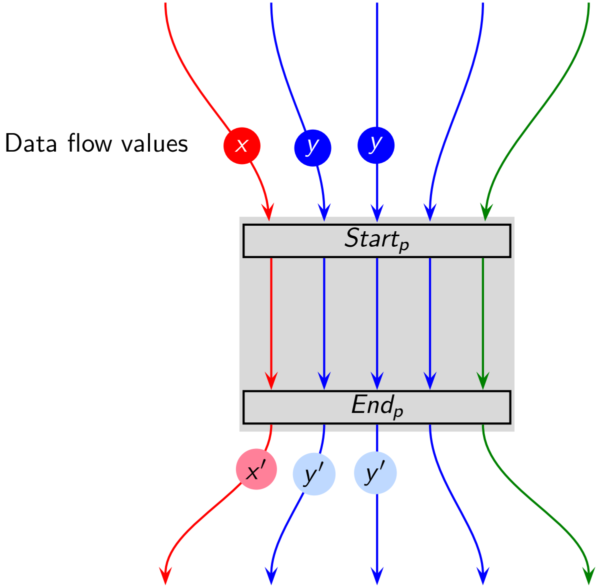
Value Contexts [CC08, SOAP13]



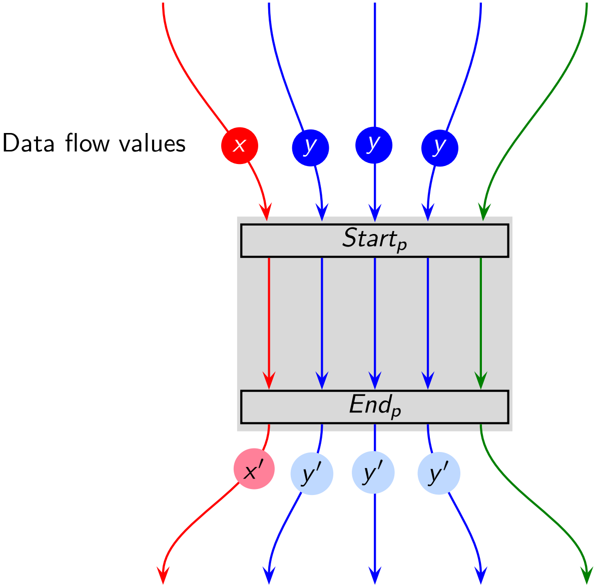
Value Contexts [CC08, SOAP13]



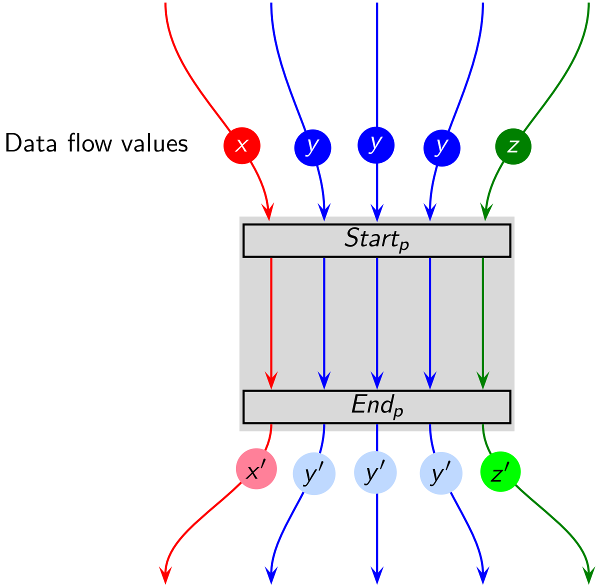
Value Contexts [CC08, SOAP13]



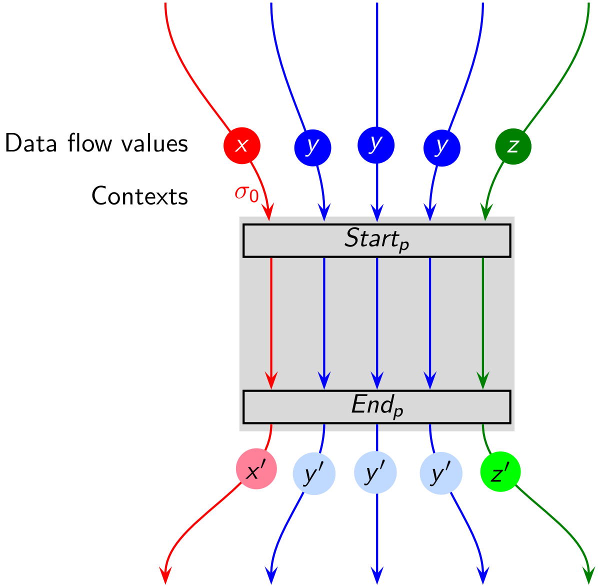
Value Contexts [CC08, SOAP13]



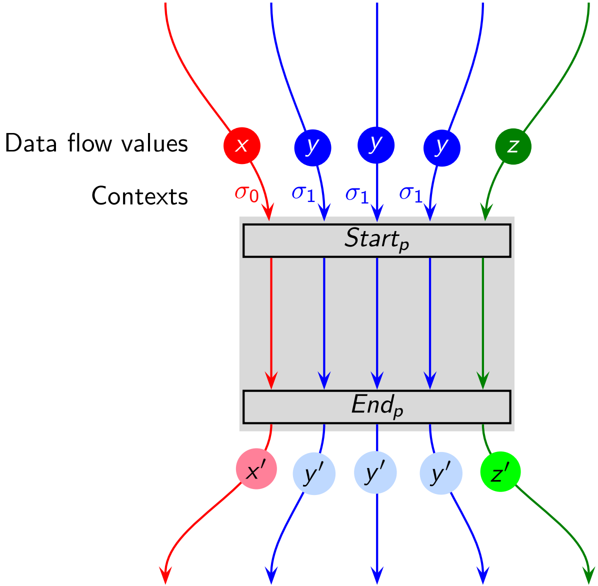
Value Contexts [CC08, SOAP13]



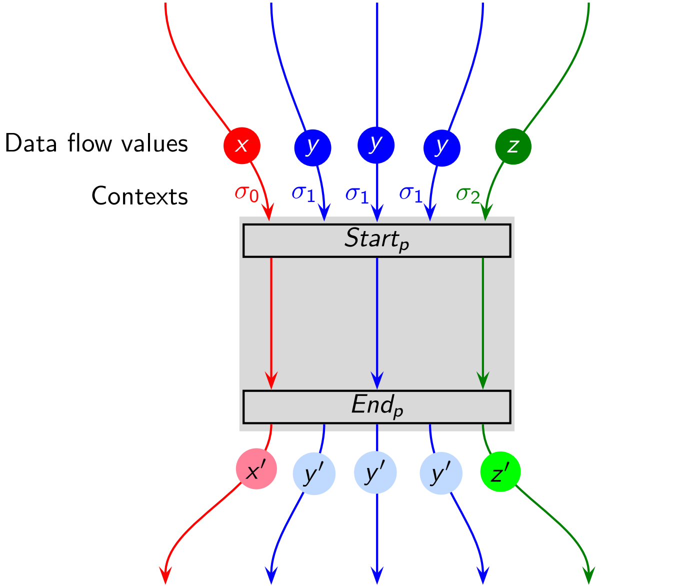
Value Contexts [CC08, SOAP13]



Value Contexts [CC08, SOAP13]



Value Contexts [CC08, SOAP13]



Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

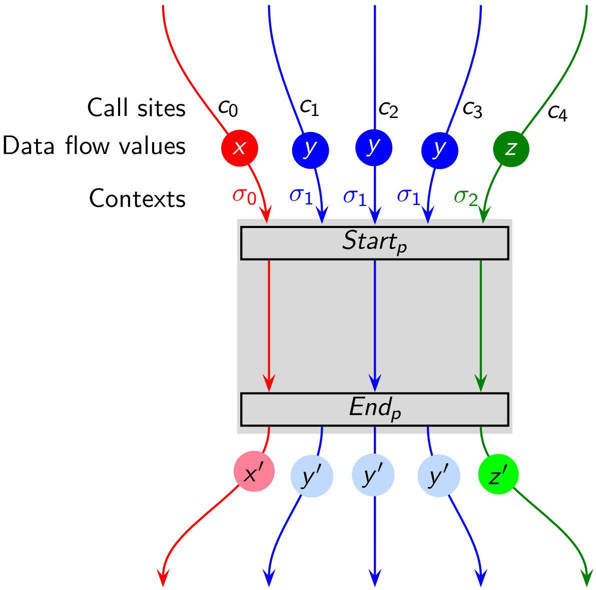
Intraprocedural Analysis

Interprocedural Analysis

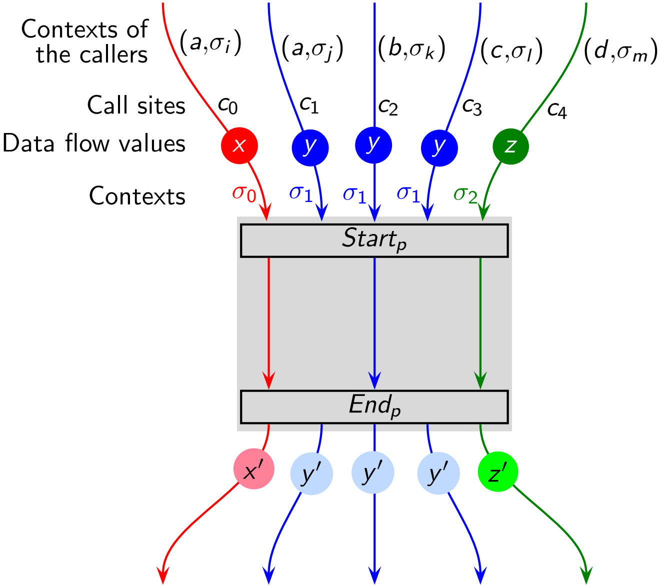
Conclusions

References

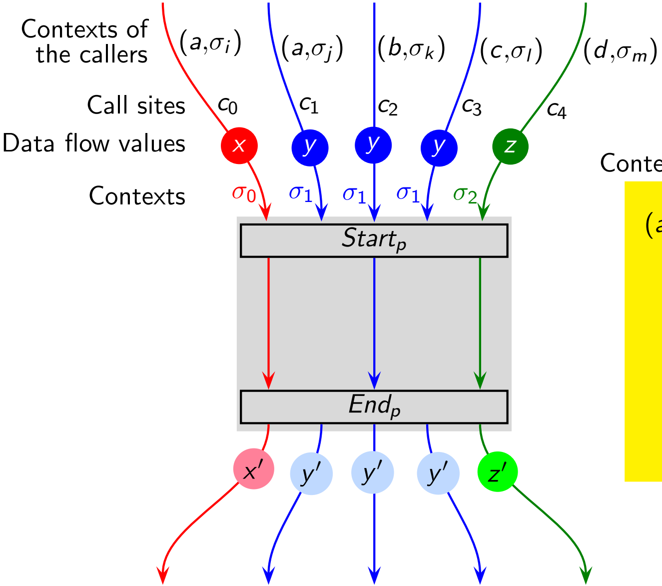
Value Contexts [CC08, SOAP13]



Value Contexts [CC08, SOAP13]



Value Contexts [CC08, SOAP13]



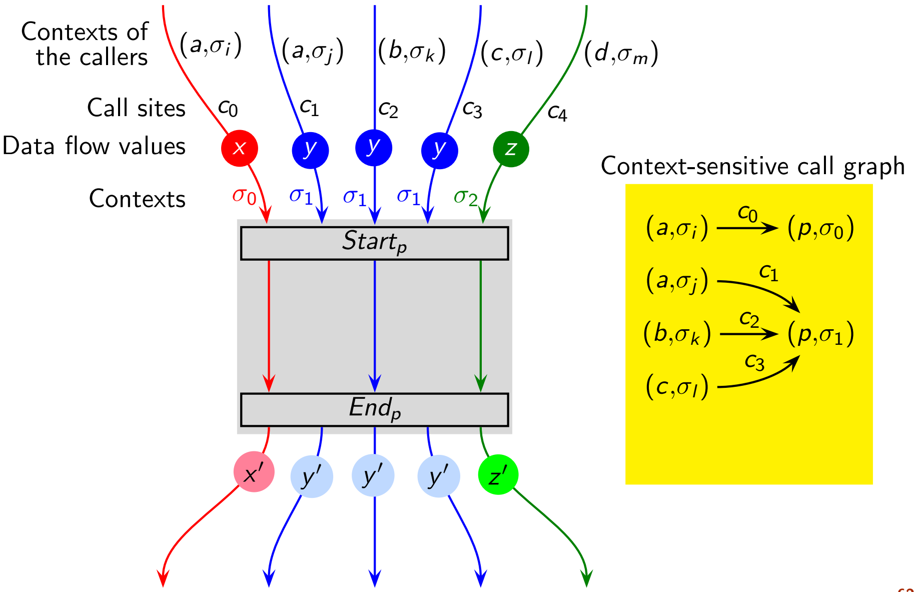
Context-sensitive call graph

$$(a,\sigma_i) \xrightarrow{c_0} (p,\sigma_0)$$

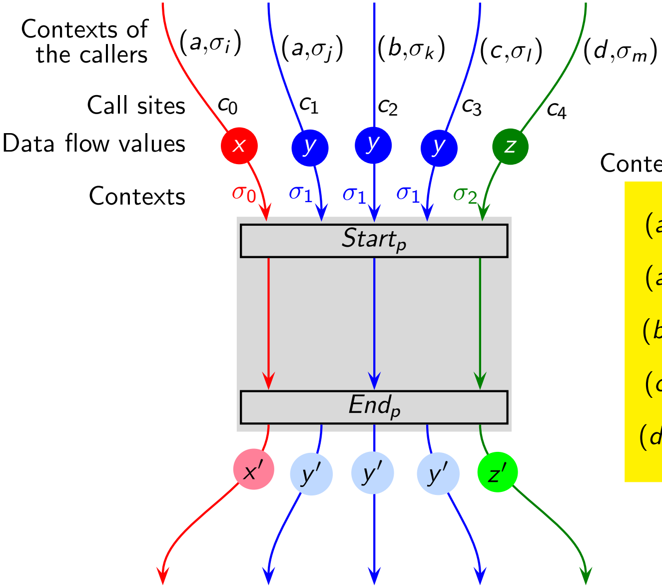
Uday Khedker
IIT Bombay

Uday Khedker
IIT Bombay

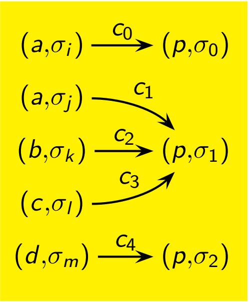
Value Contexts [CC08, SOAP13]



Value Contexts [CC08, SOAP13]



Context-sensitive call graph



Value Contexts [CC08, SOAP13]

Analyze a procedure once for an input data flow value

- The number of times a procedure is analyzed reduces dramatically
- Similar to the tabulation based method of functional approach [Sharir-Pnueli, 1981]

  However,

  - Value contexts record calling contexts too
    Useful for context matching across program analyses
  - Can avoid some reprocessing even when a new input value is found

Empirical Observations About Value Contexts

- Reaching definitions analysis in GCC 4.2.0 (CC-2008)
  Analysis of Towers of Hanoi
  - Time brought down from 3.973×10^6 ms to 2.37 ms
  - No of call strings brought down from 10^6+ to 8

Empirical Observations About Value Contexts

- Reaching definitions analysis in GCC 4.2.0 (CC-2008)

  Analysis of Towers of Hanoi

  - Time brought down from 3.973×10^6 ms to 2.37 ms
  - No of call strings brought down from 10^6+ to 8

- Generic Interprocedural Analysis Framework in SOOT (SOAP-2013)

  Empirical observations on SPECJVM98 and DaCapo 2006 benchmarks for on-the-fly call graph construction

  - Average number of contexts per procedure lies in the range 4-25
  - Much fewer long call chains than in the default call graph constructed using SPARK

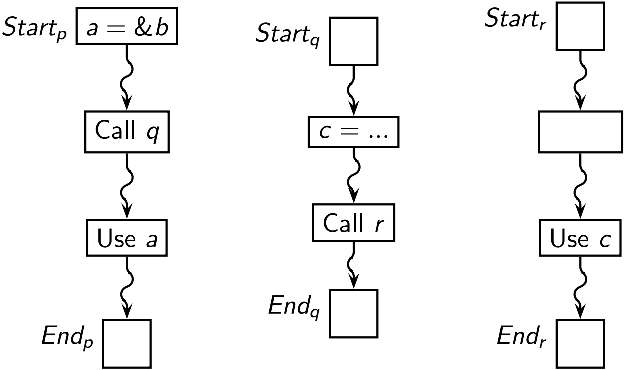
    For legnth 7, less than 50%

    For length 10, less than 5%

Empirical Observations About Value Contexts

- Reaching definitions analysis in GCC 4.2.0 (CC-2008)
  Analysis of Towers of Hanoi
  - Time brought down from 3.973×10^6 ms to 2.37 ms
  - No of call strings brought down from 10^6+ to 8
- Generic Interprocedural Analysis Framework in SOOT (SOAP-2013)
  Empirical observations on SPECJVM98 and DaCapo 2006 benchmarks for on-the-fly call graph construction
  - Average number of contexts per procedure lies in the range 4-25
  - Much fewer long call chains than in the default call graph constructed using SPARK
    For legnth 7, less than 50%
    For length 10, less than 5%
- And yet, it is insufficient for scaling flow- and context-sensitive points-to analysis to more than 35 kLoC

Top-down Analysis With Bypassing



- Procedures q and r do not access a
- Can we avoid propagating the points-to pair (a, b) *through* procedure q (and hence through r)?
- How do we know which pairs should *bypass* a call?

  Compute the bypassing set for each procedure during the analysis

Research Explorations in Interprocedural Analysis

- Broad categories of interprocedural analysis
- Scaling top-down analysis using value contexts and bypassing
- A unified model of context-sensitive methods Next Topic
- Improving bottom-up analysis by eliminating control flow
- Precise virtual call resolution with demand-driven analysis
- Improving call graphs using callee contexts

Context-Sensitive Methods

Method	Notion of context	Definition of context	Inlining strategy: Call graph traversal	Inlining strategy: Call-return matching	Restrictions on frameworks
Call strings	Explicit	Call strings	Context-dep. top-down	Context	Finite L
VBTCS	Explicit	Call strings and data flow values	Context-dep. top-down	Context	Finite lattice
Value contexts	Explicit	Data flow values	Context-dep. top-down	Context transition diagram	Finite L
IFDS	Implicit	Not specified	Context-dep. top-down	By the algorithm	Distributive ffs L with $\subseteq$
IDE	Implicit	Not specified	Context-dep. top-down	By the algorithm	L of distributive maps
Object sensitivity	Explicit	Strings of object allocation sites	Context-dep. top-down	Context	Finite lattice
Functional	None	None	Context-ind. bottom-up	Inlining summaries	Reducing $\sqcap$ and $\circ$ of ffs

Formalizing Context-Sensitive Analyses [CSUR21]

- Context-sensitive information at a program point is a pair (σ, x) where
  σ is the context and x is the data flow value
  - A separate value is computed for each context reaching a procedure
  - We leave the context σ and the data flow value x undefined
    σ is defined by the method of performing analysis
    x is defined by the analysis to be performed
- Examples of contexts
  - A call chain (i.e. a sequence of unfinished calls) reaching a procedure
  - A data flow reaching the start of a procedure

Formalizing Context-Sensitive Analyses [CSUR21]

- Context-sensitive information at a program point is a pair (σ, x)
- We model context-sensitive methods by defining two structures
  - An *abstract value structure* models ways of computing x

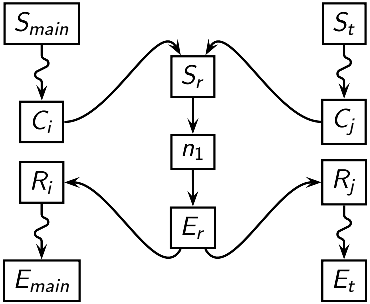
$$\boxed{\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)}$$

  - An *abstract context structure* models ways of computing σ

$$\boxed{\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)}$$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

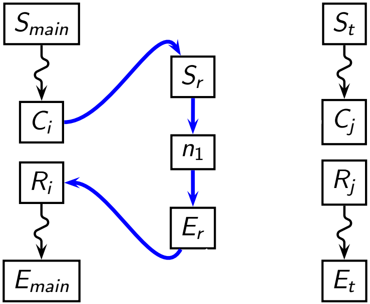
Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$

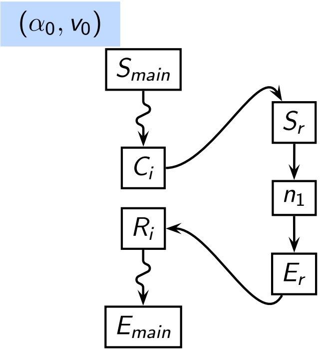


Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

Uday Khedker
IIT Bombay

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

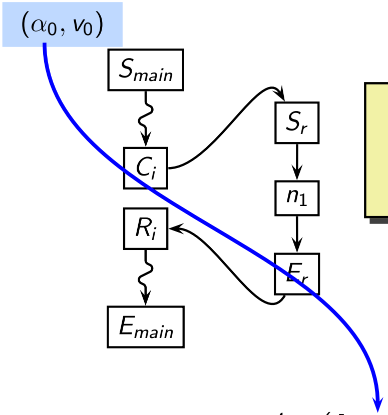
Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$

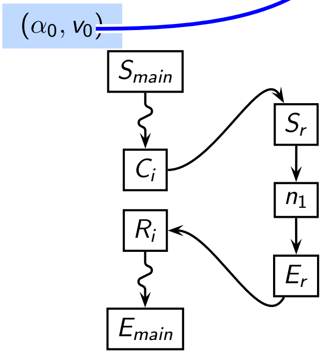


- Set of *abstract contexts* $\mathbb{A}$
- Initial abstract context $\alpha_0 \in \mathbb{A}$

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



- Set of *abstract values* $\mathbb{M}$

  May be different from the underlying data flow values in $\mathbb{L}$

  Function $\text{Project}_{Q \in \mathbb{P}\text{roc}} : \mathbb{M} \rightarrow \mathbb{L}$ gives values in $\mathbb{L}$ from those in $\mathbb{M}$

- Initial abstract value $v_0 \in \mathbb{M}$

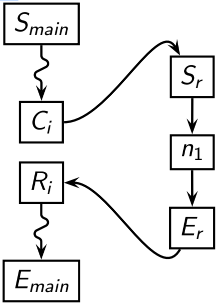
  (holds at $n : Start_{main}$)

Abstract context structure: $\mathcal{A} = \left(\mathbb{A}, \alpha_0, \text{Ncontext}_n\right)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = (\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q)$

(α_0, v_0)



- Set of *abstract values* $\mathbb{M}$

  May be different from the underlying data flow values in $\mathbb{L}$

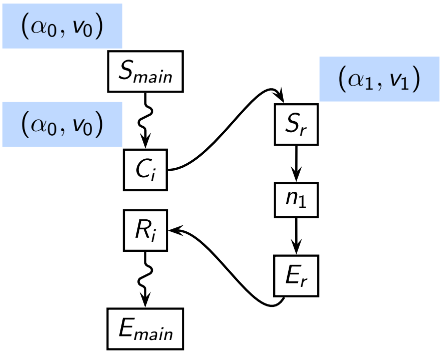
  Function $\text{Project}_{Q \in \mathbb{P}\text{roc}} : \mathbb{M} \to \mathbb{L}$ gives values in $\mathbb{L}$ from those in $\mathbb{M}$

- Initial abstract value $v_0 \in \mathbb{M}$ (holds at $n : Start_{main}$)

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

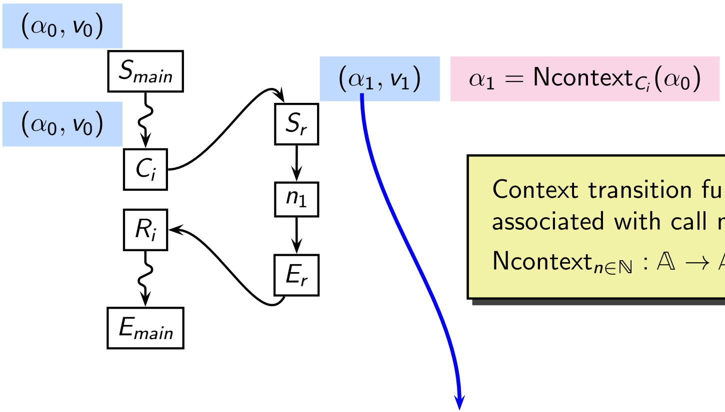
Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



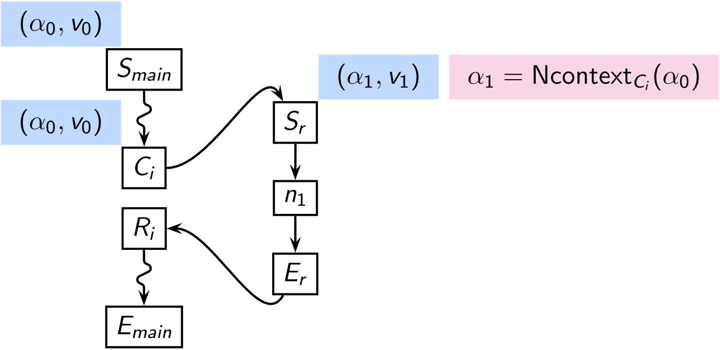
Context transition function associated with call node n

$\text{Ncontext}_{n \in \mathbb{N}} : \mathbb{A} \to \mathbb{A}$

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

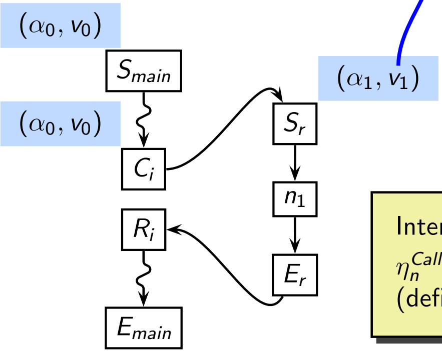
Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



$\alpha_1 = \text{Ncontext}_{C_i}(\alpha_0)$

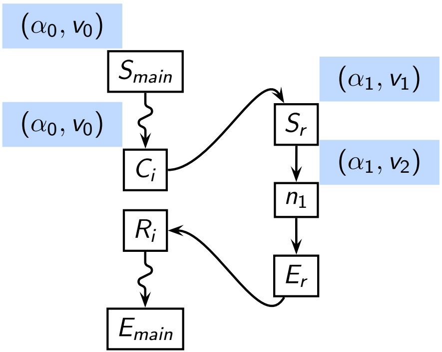
$v_1 = \eta_{C_i}^{Call}(v_0)$

Interprocedural abstract flow function
$\eta_n^{Call} : \mathbb{M} \rightarrow \mathbb{M}$
(defined only for call nodes)

Abstract context structure: $\mathcal{A} = \left(\mathbb{A}, \alpha_0, \text{Ncontext}_n\right)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

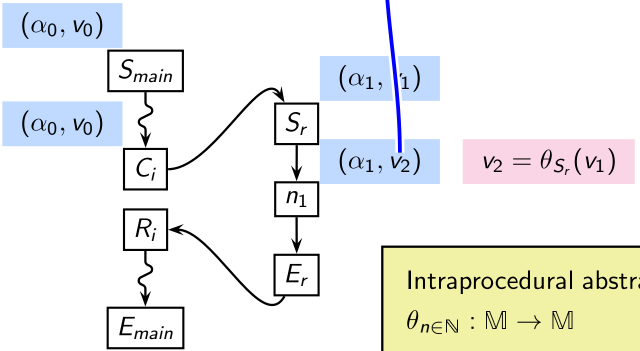
Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



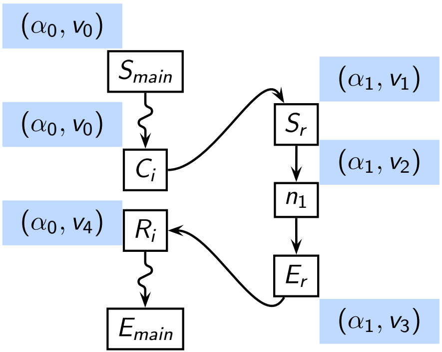
Intraprocedural abstract flow function

$\theta_{n \in \mathbb{N}} : \mathbb{M} \to \mathbb{M}$

Abstract context structure: $\mathcal{A} = \left(\mathbb{A}, \alpha_0, \text{Ncontext}_n\right)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$

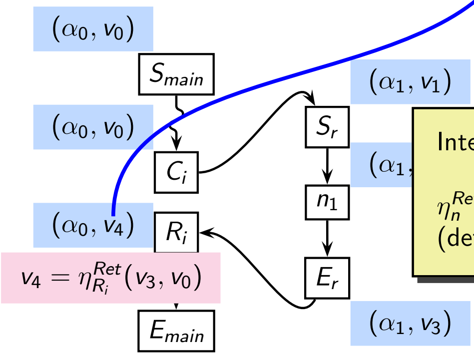


Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

Uday Khedker
IIT Bombay

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



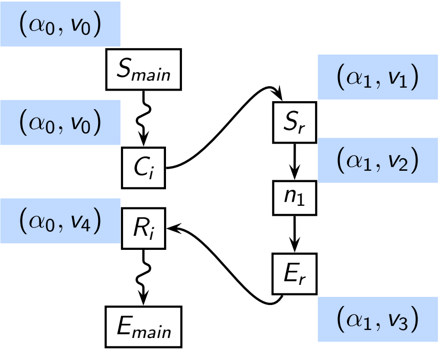
Interprocedural abstract flow function

$\eta_n^{Ret} : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{M}$
(defined only for return nodes)

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$

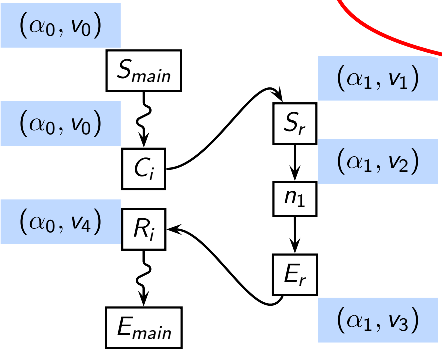


- Abstract values
- Projection function for underlying data flow values
- Abstract flow functions
- Abstract contexts
- Context transition function

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = (\underbrace{\mathbb{M}, v_0}, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q)$

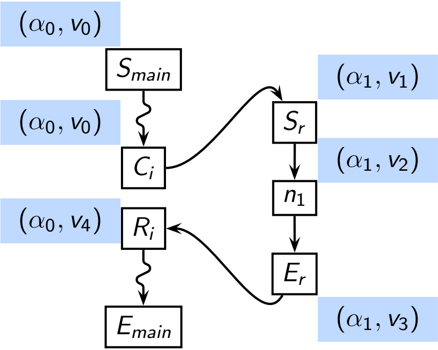


- Abstract values
- Projection function for underlying data flow values
- Abstract flow functions
- Abstract contexts
- Context transition function

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$

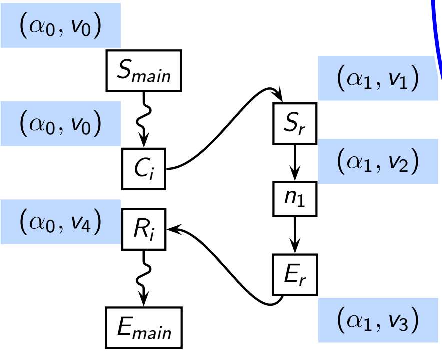


- Abstract values
- Projection function for underlying data flow values
- Abstract flow functions
- Abstract contexts
- Context transition function

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$

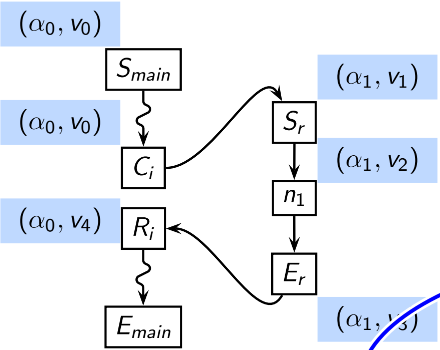


- Abstract values
- Projection function for underlying data flow values
- Abstract flow functions
- Abstract contexts
- Context transition function

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$

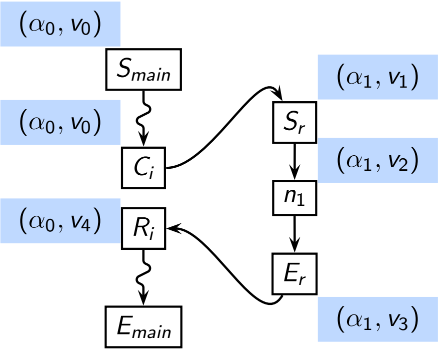


- Abstract values
- Projection function for underlying data flow values
- Abstract flow functions
- Abstract contexts
- Context transition function

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = \left(\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q\right)$



- Abstract values
- Projection function for underlying data flow values
- Abstract flow functions
- Abstract contexts
- Context transition function

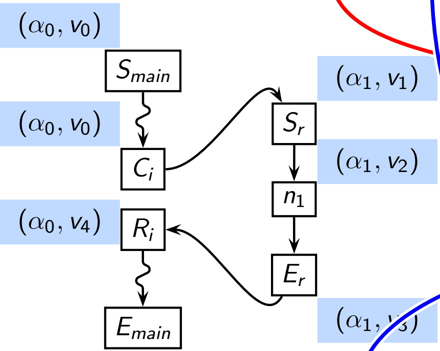
Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = (\mathbb{M}, v_0, \theta_n, \eta_n^{Call}, \eta_n^{Ret}, \text{Project}_Q)$



- Abstract values
- Projection function for underlying data flow values
- Abstract flow functions
- Abstract contexts
- Context transition function

Abstract context structure: $\mathcal{A} = (\mathbb{A}, \alpha_0, \text{Ncontext}_n)$

A Unified Model of Context-Sensitive Data Flow Analysis [CSUR21]

Abstract value structure: $\mathcal{V} = (\underbrace{\mathbb{M}, v_0}, \underbrace{\theta_n, \eta_n^{Call}, \eta_n^{Ret}}, \text{Project}_Q)$

(α_0, v_0)

E_{main}

$(\alpha_1, \ldots)$

- This model can be instantiated to a large class of methods by defining $\mathcal{V}$ and $\mathcal{A}$ for a method
- Soundness and precision of the given method can be argued using the model

- Abstract values
- Projection function for underlying data flow values
- Abstract flow functions
- Abstract contexts
- Context transition function

Abstract context structure: $\mathcal{A} = (\underbrace{\mathbb{A}, \alpha_0}, \text{Ncontext}_n)$

Uday Khedker
IIT Bombay

Abstract Context Structure for Context-Sensitive Methods [CSUR21]

Method	$\mathbb{A}$	α_0	$\mathsf{Ncontext}_n(\alpha)$

Abstract Context Structure for Context-Sensitive Methods [CSUR21]

Method	$\mathbb{A}$	α_0	$\text{Ncontext}_n(\alpha)$
Full call strings	Σ	ϵ	$\alpha \cdot n$
VBTCS	Σ	ϵ	$R(\alpha \cdot n, \text{InA}_n(\alpha))$
VASCO	$\mathbb{L}$	BI	$\text{InA}_n(\alpha)$
Restricted contexts	$\{\epsilon\}$	ϵ	ϵ
IFDS method	$\mathbb{L}$	BI	$\text{InA}_n(\alpha)$
IDE method	$\{\epsilon\}$	ϵ	ϵ
Functional method	$\{\epsilon\}$	ϵ	ϵ
k-limited call strings	Σ_k	ϵ	$suffix_k(\alpha \cdot n)$
Context-insensitive	$\{\epsilon\}$	ϵ	ϵ

Abstract Value Structure for Context-Sensitive Methods [CSUR21]

Method	$\mathbb{M}$	v_0	$\theta_n(v)$	$\eta_n^{Call}(v)$	$\eta_n^{Ret}(v, w)$	$\text{Project}_Q(v)$
Full call strings	$\mathbb{L}$	BI	$f_n(v)$	v	v	v
VBTCS	$\mathbb{L}$	BI	$f_n(v)$	v	v	v
VASCO	$\mathbb{L}$	BI	$f_n(v)$	v	v	v
IFDS method	$\mathbb{L}$	BI	$f_n(v)$	v	v	v
IDE method	$\mathbb{L} \rightarrow \mathbb{L}$	id	$f_n \circ v$	id	$v \circ w$	$v\,(BI_Q)$
Functional method	$\mathbb{L} \rightarrow \mathbb{L}$	id	$f_n \circ v$	id	$v \circ w$	$v\,(BI_Q)$
k-limited call strings	$\mathbb{L}$	BI	$f_n(v)$	v	v	v
Context-insensitive	$\mathbb{L}$	BI	$f_n(v)$	v	v	v

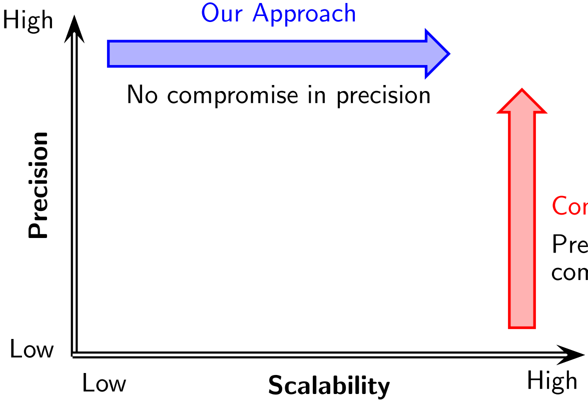
Research Explorations in Interprocedural Analysis

- Broad categories of interprocedural analysis
- Scaling top-down analysis using value contexts and bypassing
- A unified model of context-sensitive methods
- Improving bottom-up analysis by eliminating control flow — Next Topic
- Precise virtual call resolution with demand-driven analysis
- Improving call graphs using callee contexts

Our Approach [SAS16, TOPLAS20]

Improve the scalability of *exhaustive* pointer analysis without losing precision

- Construct sound and precise but compact statement level summaries
- Combine them naively and optimize for scalability without compromising soundness or precision



Summarizing a Procedure for Points-to Analysis [SAS16, TOPLAS20]

A flow-sensitive analysis requires control flow to be recorded between memory updates that share data dependence

Data dependence exists $\Rightarrow$
- Can be eliminated and the
- Control flow between the updates becomes redundant

1. $x = \&a;$
2. $y = x;$

$\Downarrow$

$x = \&a \;||\; y = \&a$

Data dependence does not exist $\Rightarrow$
- Redundant memory updates can be eliminated
- Control flow between the updates is redundant

1. $x = \&a;$
2. $y = \&b;$
3. $x = \&b;$

$\Downarrow$

$y = \&b \;||\; x = \&b$

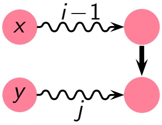
Data dependence is unknown $\Rightarrow$
- More information is required (available in callers)
- Control flow between the updates is required

1. $y = \&b;$
2. $*x = \&a;$
3. $z = y;$

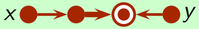
Generalized Points-to Updates (GPUs) [SAS16, TOPLAS20]

General Case

GPU $x \xrightarrow[s]{i|j} y$



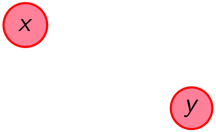
Specific Examples

Pointer assignment	GPU	Relevant memory graph after the assignment
$s\colon x = \&y$	$x \xrightarrow[s]{1\|0} y$	
$s\colon x = y$	$x \xrightarrow[s]{1\|1} y$	
$s\colon x = *y$	$x \xrightarrow[s]{1\|2} y$	
$s\colon *x = y$	$x \xrightarrow[s]{2\|1} y$	

- The direction in a GPU is to distinguish between what is being defined to what is being read
- For pointer analysis, case $i = 0$ does not exist
- Classical points-to update is a special case of generalized points-to update with $i = 1$ and $j = 0$

Classical Points-to Updates: A Low Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```

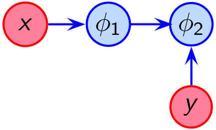


All variables are global

Red nodes are known named locations

Classical Points-to Updates: A Low Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```



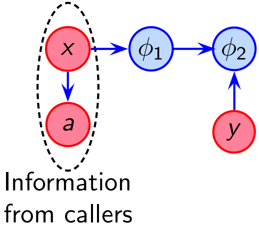
All variables are global

Red nodes are known named locations

Blue nodes are placeholders denoting unknown locations

Classical Points-to Updates: A Low Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```



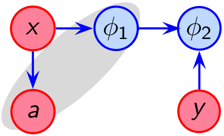
All variables are global

Red nodes are known named locations

Blue nodes are placeholders denoting unknown locations

Classical Points-to Updates: A Low Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```



All variables are global

Red nodes are known named locations

Blue nodes are placeholders denoting unknown locations

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

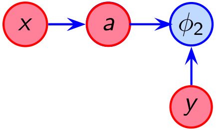
Interprocedural Analysis

Conclusions

References

Classical Points-to Updates: A Low Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```



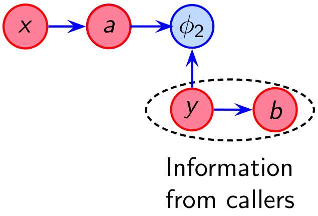
All variables are global

Red nodes are known named locations

Blue nodes are placeholders denoting unknown locations

Classical Points-to Updates: A Low Level Abstraction of Memory for Points-to Analysis

```
f()
{
  *x = y
}
```



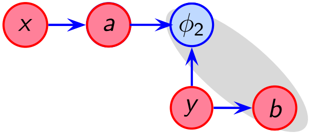
All variables are global

Red nodes are known named locations

Blue nodes are placeholders denoting unknown locations

Classical Points-to Updates: A Low Level Abstraction of Memory for Points-to Analysis

```
f()
{
  *x = y
}
```



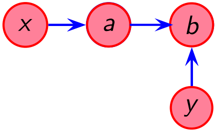
All variables are global

Red nodes are known named locations

Blue nodes are placeholders denoting unknown locations

Classical Points-to Updates: A Low Level Abstraction of Memory for Points-to Analysis

```
f()
{
  *x = y
}
```



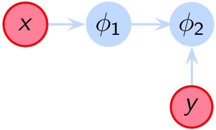
All variables are global

Red nodes are known named locations

Blue nodes are placeholders denoting unknown locations

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

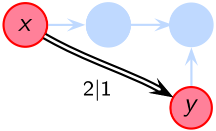
```
f()
{
   *x = y
}
```



Blue arrows are low level view of memory in terms of classical points-to facts

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```

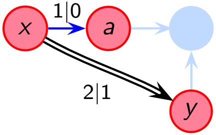


Blue arrows are low level view of memory in terms of classical points-to facts

Black arrows are high level view of memory in terms of generalized points-to facts

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

```
f()
{
  *x = y
}
```

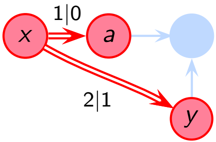


Blue arrows are low level view of memory in terms of classical points-to facts

Black arrows are high level view of memory in terms of generalized points-to facts

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

```
f()
{
  *x = y
}
```

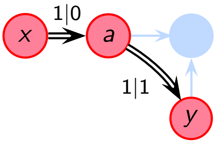


Blue arrows are low level view of memory in terms of classical points-to facts

Black arrows are high level view of memory in terms of generalized points-to facts

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```

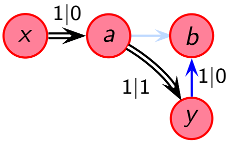


Blue arrows are low level view of memory in terms of classical points-to facts

Black arrows are high level view of memory in terms of generalized points-to facts

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

```
f()
{
  *x = y
}
```

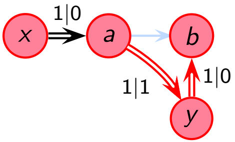


Blue arrows are low level view of memory in terms of classical points-to facts

Black arrows are high level view of memory in terms of generalized points-to facts

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

```
f()
{
  *x = y
}
```

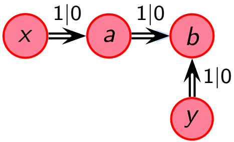


Blue arrows are low level view of memory in terms of classical points-to facts

Black arrows are high level view of memory in terms of generalized points-to facts

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```

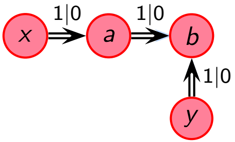


Blue arrows are low level view of memory in terms of classical points-to facts

Black arrows are high level view of memory in terms of generalized points-to facts

Generalized Points-to Updates: A High Level Abstraction of Memory for Points-to Analysis

```
f()
{
   *x = y
}
```



Blue arrows are low level view of memory in terms of classical points-to facts

Black arrows are high level view of memory in terms of generalized points-to facts

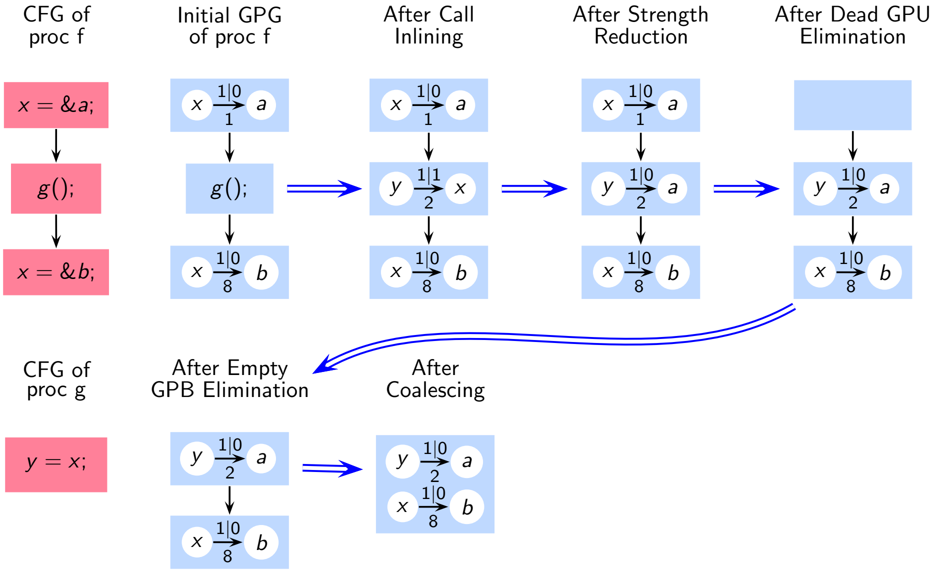
This abstraction does not introduce any imprecision over the classical points-to graph

How GPGs Handle the Factors Affecting Scalability

Three issues that cause non-scalability

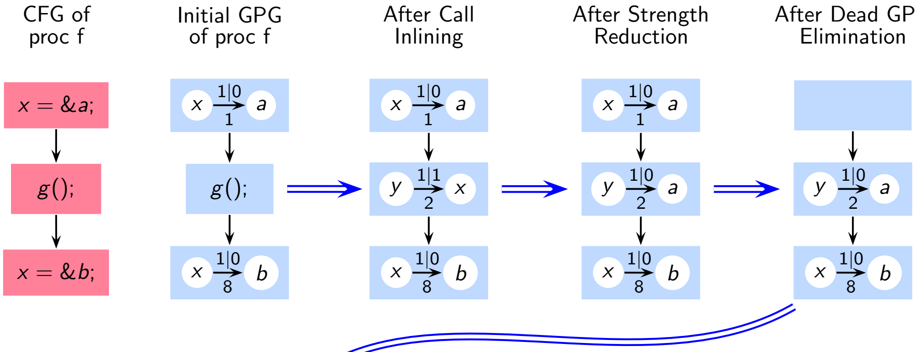
- Modelling indirect accesses of pointees that are defined in callers without examining their code
  - GPUs track indirection levels that relate (transitively indirect) pointees of a variable with those of other variables
- Preserving data dependence between memory updates
  - Maintain minimal control flow between memory updates ensuring soundness and precision
- Incorporating the effect of summaries of the callee procedures transitively
  - Series of GPG optimizations gives compactness that mitigate the impact of transitive inlining
- Scale to 158 kLoC when implemented using LTO framework in GCC 4.7.2

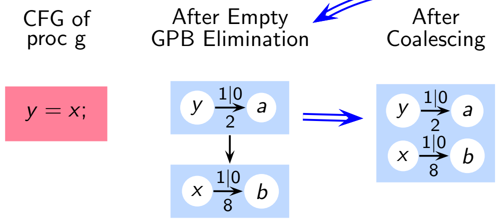
GPGs Across Optimizations [SAS16, TOPLAS20]



Uday Khedker
IIT Bombay

GPGs Across Optimizations [SAS16, TOPLAS20]





All of the above GPGs represent sound, precise, and semantically equivalent summaries of procedure f for points-to analysis

Research Explorations in Interprocedural Analysis

- Broad categories of interprocedural analysis
- Scaling top-down analysis using value contexts and bypassing
- A unified model of context-sensitive methods
- Improving bottom-up analysis by eliminating control flow
- Precise virtual call resolution with demand-driven analysis Next Topic
- Improving call graphs using callee contexts

Precise Virtual Call Resolution With Demand-Driven Analysis [SCP20]

- Virtual calls are made through pointers to objects
- The resolution of virtual calls needs the class of the objects rather than the objects thus a full blown points-to analysis is an overkill hampering scalability

  We need a points-to-class analysis that uses type based abstraction
- Any data abstraction (such as type based abstraction) conflates many objects together and introduces imprecision
- We show how this imprecision can be mitigated in a demand-driven method

Uday Khedker
IIT Bombay

Exhaustive Analysis and Data Abstraction

$$p = \&z$$

$$\downarrow$$

$$y = \text{new}A$$

$$\downarrow$$

$$*p = \text{new}A$$

$$\downarrow$$

$$z \rightarrow f = \text{new}B$$

$$\downarrow$$

$$y \rightarrow f = \text{new}C$$

$$\downarrow$$

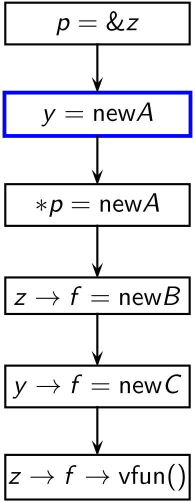
$$z \rightarrow f \rightarrow \text{vfun}()$$

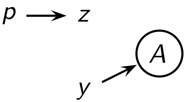
Exhaustive Analysis and Data Abstraction

$$\boxed{p = \&z} \rightarrow \boxed{y = \text{new}A} \rightarrow \boxed{*p = \text{new}A} \rightarrow \boxed{z \rightarrow f = \text{new}B} \rightarrow \boxed{y \rightarrow f = \text{new}C} \rightarrow \boxed{z \rightarrow f \rightarrow \text{vfun}()}$$

$p \longrightarrow z$

Exhaustive Analysis and Data Abstraction





Exhaustive Analysis and Data Abstraction

$p = \&z$

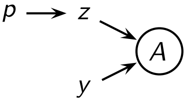
$y = \text{new}A$

$\boxed{*p = \text{new}A}$

$z \rightarrow f = \text{new}B$

$y \rightarrow f = \text{new}C$

$z \rightarrow f \rightarrow \text{vfun}()$



Exhaustive Analysis and Data Abstraction

$p = \&z$

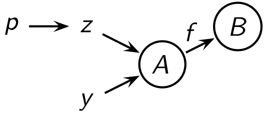
$y = \text{new} A$

$*p = \text{new} A$

$\boxed{z \rightarrow f = \text{new} B}$

$y \rightarrow f = \text{new} C$

$z \rightarrow f \rightarrow \text{vfun}()$



Exhaustive Analysis and Data Abstraction

$p = \&z$

↓

$y = \text{new}A$

↓

$*p = \text{new}A$

↓

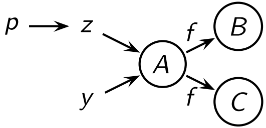
$z \rightarrow f = \text{new}B$

↓

$\boxed{y \rightarrow f = \text{new}C}$

↓

$z \rightarrow f \rightarrow \text{vfun}()$



Exhaustive Analysis and Data Abstraction

$p = \&z$

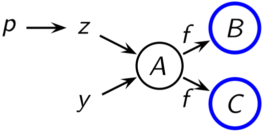
$y = \text{new}A$

$*p = \text{new}A$

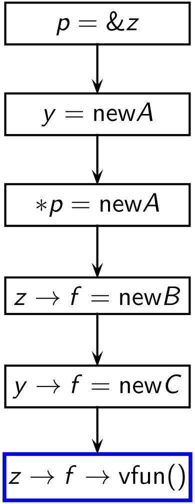
$z \rightarrow f = \text{new}B$

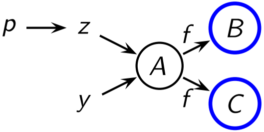
$y \rightarrow f = \text{new}C$

$\boxed{z \rightarrow f \rightarrow \text{vfun}()}$



Exhaustive Analysis and Data Abstraction





- Type abstraction introduces spurious alias (y, z)
- We spuriously conclude that vfun could be called for class C too

Need For Speculation in Demand-Driven Method

$$p = \&z$$

$$\downarrow$$

$$*p = \text{new}A$$

$$\downarrow$$

$$z \rightarrow \text{vfun}()$$

Need For Speculation in Demand-Driven Method

$p = \&z$

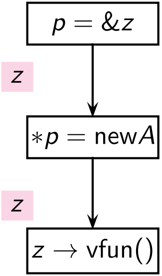
$\downarrow$

$*p = \text{new}A$

$\downarrow$ z

$z \rightarrow \text{vfun}()$

Need For Speculation in Demand-Driven Method



Uday Khedker
IIT Bombay

Need For Speculation in Demand-Driven Method

$p = \&z$

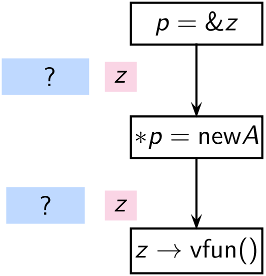
? z

$*p = \text{new}A$

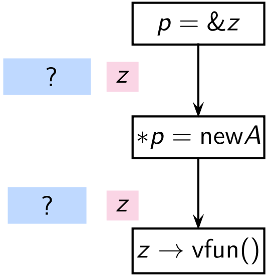
z

$z \rightarrow \text{vfun}()$

Need For Speculation in Demand-Driven Method

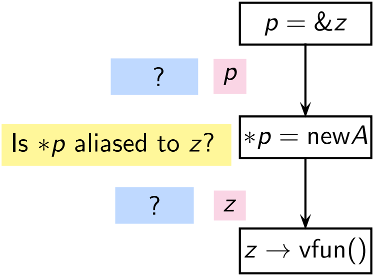


Need For Speculation in Demand-Driven Method



To ensure soundness, we need to speculate demands at indirect assignment statements

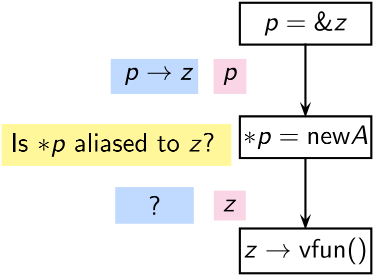
Need For Speculation in Demand-Driven Method



Alias speculation

To ensure soundness, we need to speculate demands at indirect assignment statements

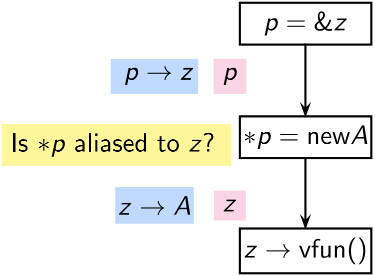
Need For Speculation in Demand-Driven Method



Alias speculation

To ensure soundness, we need to speculate demands at indirect assignment statements

Need For Speculation in Demand-Driven Method



Alias speculation

To ensure soundness, we need to speculate demands at indirect assignment statements

Need For Speculation in Demand-Driven Method

$p = \&z$

$p \rightarrow z$ p

Is $*p$ aliased to z?

$*p = \text{new}A$

Alias pair (p, &z) can be found by raising demand for &z

$z \rightarrow A$ z

$z, \&z$

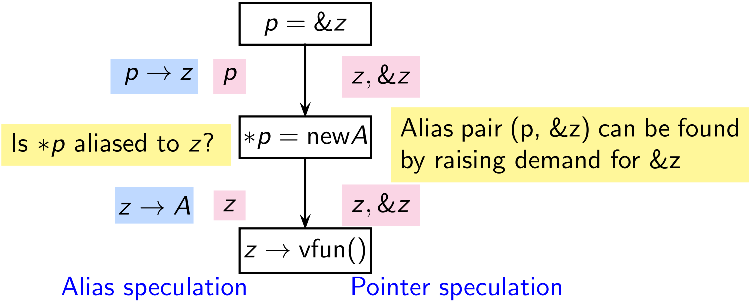
$z \rightarrow \text{vfun}()$

Alias speculation

Pointer speculation

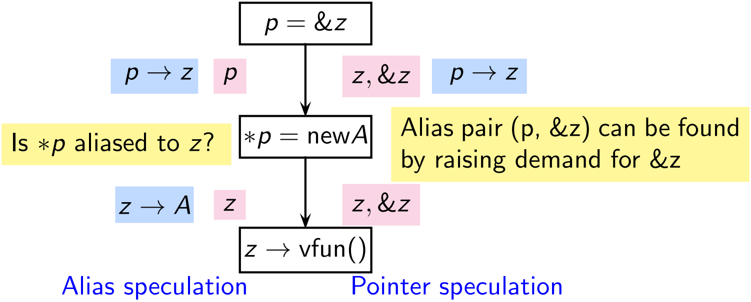
To ensure soundness, we need to speculate demands at indirect assignment statements

Need For Speculation in Demand-Driven Method



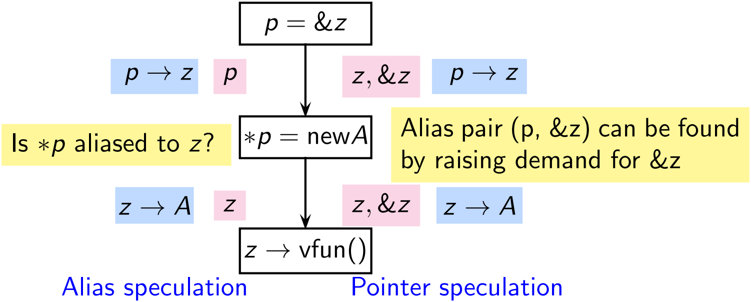
To ensure soundness, we need to speculate demands at indirect assignment statements

Need For Speculation in Demand-Driven Method



To ensure soundness, we need to speculate demands at indirect assignment statements

Need For Speculation in Demand-Driven Method



To ensure soundness, we need to speculate demands at indirect assignment statements

Need For Speculation in Demand-Driven Method

$p = \&z$

EFSA	Exhaustive flow-sensitive points-to analysis
DFSA	Demand-driven flow-sensitive points-to analysis
ADFSA	Demand-driven flow-sensitive points-to analysis using alias speculation
PDFSA	Demand-driven flow-sensitive points-to analysis using pointer speculation

Alias speculation　　Pointer speculation

To ensure soundness, we need to speculate demands at indirect assignment statements

Uday Khedker
IIT Bombay

ADFSA **and** PDFSA **[SCP20]**

$$p = \&z$$
$$\downarrow$$
$$y = \text{new} A$$
$$\downarrow$$
$$*p = \text{new} A$$
$$\downarrow$$
$$z \rightarrow f = \text{new} B$$
$$\downarrow$$
$$y \rightarrow f = \text{new} C$$
$$\downarrow$$
$$z \rightarrow f \rightarrow \text{vfun}()$$

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)

Demand Propagation

$p = \&z$

$\downarrow$

$y = \text{new}A$

$\downarrow$

$*p = \text{new}A$

$\downarrow$

$z \rightarrow f = \text{new}B$

$\downarrow$

$y \rightarrow f = \text{new}C$

$\downarrow$ $z, z \rightarrow f$

$z \rightarrow f \rightarrow \text{vfun}()$

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)

Demand Propagation

$p = \&z$

$y = \text{new} A$

$*p = \text{new} A$

$z \rightarrow f = \text{new} B$

z, y

$y \rightarrow f = \text{new} C$

$z, z \rightarrow f$

$z \rightarrow f \rightarrow \text{vfun}()$

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)

Demand Propagation

$p = \&z$

$y = \text{new}A$

$*p = \text{new}A$

z, y

$z \rightarrow f = \text{new}B$

z, y

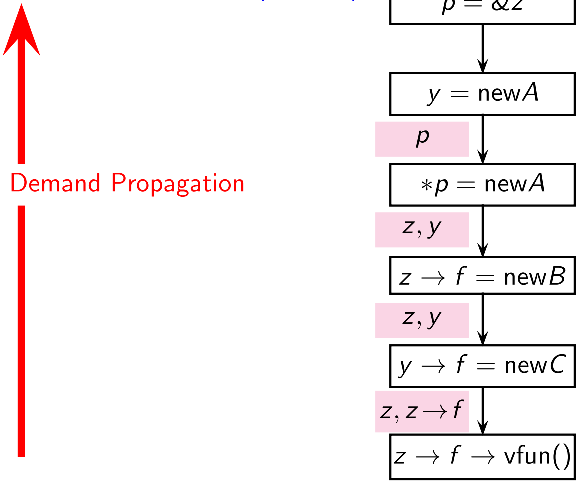
$y \rightarrow f = \text{new}C$

$z, z \rightarrow f$

$z \rightarrow f \rightarrow \text{vfun}()$

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)



ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)

Demand Propagation

$p = \&z$

p

$y = \text{new}A$

p

$*p = \text{new}A$

z, y

$z \rightarrow f = \text{new}B$

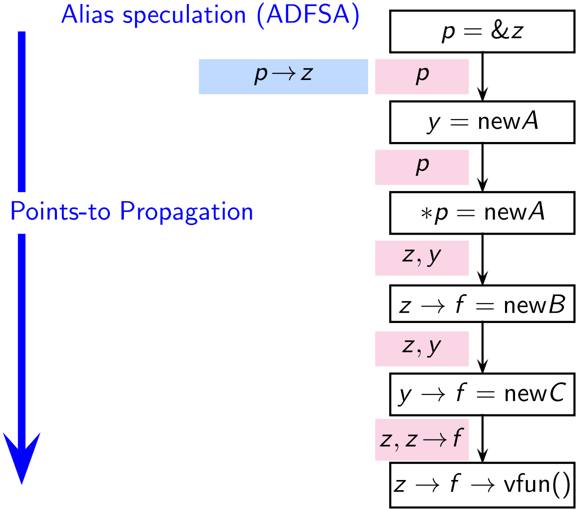
z, y

$y \rightarrow f = \text{new}C$

$z, z \rightarrow f$

$z \rightarrow f \rightarrow \text{vfun}()$

ADFSA and PDFSA [SCP20]



ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA)

Points-to Propagation

$p = \&z$

$p \rightarrow z$ | p

$y = \text{new}A$

$p \rightarrow z$ | p

$*p = \text{new}A$

z, y

$z \rightarrow f = \text{new}B$

z, y

$y \rightarrow f = \text{new}C$

$z, z \rightarrow f$

$z \rightarrow f \rightarrow \text{vfun}()$

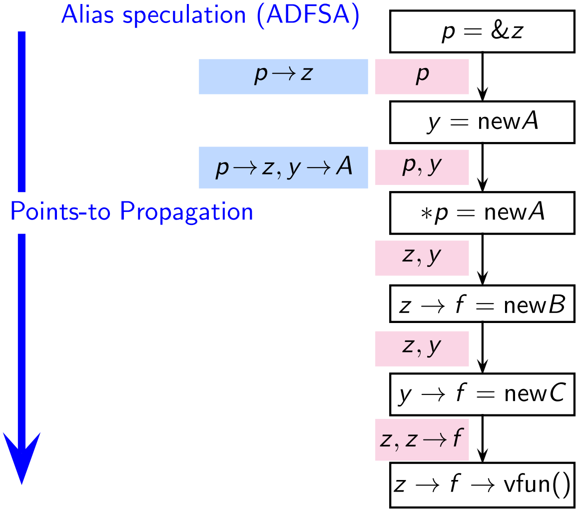
ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA)

Demand Propagation

Alias speculation	Demand	Statement
		$p = \&z$
$p \rightarrow z$	p	
		$y = \text{new}A$
$p \rightarrow z$	p, y	
		$*p = \text{new}A$
	z, y	
		$z \rightarrow f = \text{new}B$
	z, y	
		$y \rightarrow f = \text{new}C$
	$z, z \rightarrow f$	
		$z \rightarrow f \rightarrow \text{vfun}()$

ADFSA and PDFSA [SCP20]



Uday Khedker
IIT Bombay

ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA)

Points-to Propagation

$p = \&z$

$p \rightarrow z$ | p

$y = \text{new}A$

$p \rightarrow z, y \rightarrow A$ | p, y

$*p = \text{new}A$

$p \rightarrow z \rightarrow A, y \rightarrow A$ | z, y

$z \rightarrow f = \text{new}B$

z, y

$y \rightarrow f = \text{new}C$

$z, z \rightarrow f$

$z \rightarrow f \rightarrow \text{vfun}()$

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)

Points-to Propagation

$p = \&z$

$p \rightarrow z$ | p

$y = \text{new}A$

$p \rightarrow z, y \rightarrow A$ | p, y

$*p = \text{new}A$

$p \rightarrow z \rightarrow A, y \rightarrow A$ | z, y

$z \rightarrow f = \text{new}B$

$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$ | z, y

$y \rightarrow f = \text{new}C$

$z, z \rightarrow f$

$z \rightarrow f \rightarrow \text{vfun}()$

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)

Points-to Propagation

Points-to	Alias speculation	Statement
		$p = \&z$
$p \rightarrow z$	p	
		$y = \text{new}A$
$p \rightarrow z, y \rightarrow A$	p, y	
		$*p = \text{new}A$
$p \rightarrow z \rightarrow A, y \rightarrow A$	z, y	
		$z \rightarrow f = \text{new}B$
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y	
		$y \rightarrow f = \text{new}C$
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$	
		$z \rightarrow f \rightarrow \text{vfun}()$

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:

ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA)

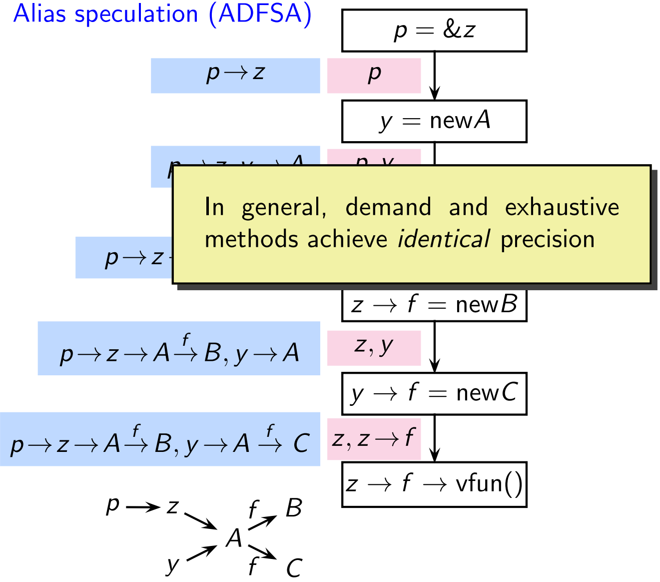
Points-to	Live	Statement
		$p = \&z$
$p \rightarrow z$	p	
		$y = \text{new} A$
$p \rightarrow z, y \rightarrow A$	p, y	
		$*p = \text{new} A$
$p \rightarrow z \rightarrow A, y \rightarrow A$	z, y	
		$z \rightarrow f = \text{new} B$
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y	
		$y \rightarrow f = \text{new} C$
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$	
		$z \rightarrow f \rightarrow \text{vfun}()$

$$\begin{tikzcd} p \arrow[r] & z \arrow[dr] & & B \\ & & A \arrow[ur, "f"] \arrow[dr, "f"'] & \\ & y \arrow[ur] & & C \end{tikzcd}$$

Uday Khedker
IIT Bombay

ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA)



In general, demand and exhaustive methods achieve *identical* precision

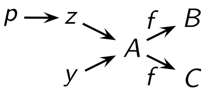
ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)

Pointer speculation (PDFSA)

Alias speculation (ADFSA)	Demand	Statement	Demand
		$p = \&z$	
$p \rightarrow z$	p		
		$y = \text{new}A$	
$p \rightarrow z, y \rightarrow A$	p, y		
		$*p = \text{new}A$	
$p \rightarrow z \rightarrow A, y \rightarrow A$	z, y		
		$z \rightarrow f = \text{new}B$	
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y		
		$y \rightarrow f = \text{new}C$	
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$		$z, \&z$
		$z \rightarrow f \rightarrow \text{vfun}()$	

Demand Propagation

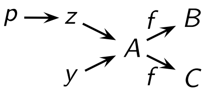


ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA) | Pointer speculation (PDFSA)

Alias speculation (ADFSA)		Statement	Pointer speculation (PDFSA)
		$p = \&z$	
$p \rightarrow z$	p		
		$y = \text{new}A$	
$p \rightarrow z, y \rightarrow A$	p, y		
		$*p = \text{new}A$	
$p \rightarrow z \rightarrow A, y \rightarrow A$	z, y		
		$z \rightarrow f = \text{new}B$	
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y		$z, \&z$
		$y \rightarrow f = \text{new}C$	
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$		$z, \&z$
		$z \rightarrow f \rightarrow \text{vfun}()$	

Demand Propagation



ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA) | Pointer speculation (PDFSA)

Alias speculation (ADFSA)		Statement	Pointer speculation (PDFSA)
		$p = \&z$	
$p \to z$	p		
		$y = \text{new} A$	
$p \to z, y \to A$	p, y		
		$*p = \text{new} A$	
$p \to z \to A, y \to A$	z, y		$z, \&z$
		$z \to f = \text{new} B$	
$p \to z \to A \xrightarrow{f} B, y \to A$	z, y		$z, \&z$
		$y \to f = \text{new} C$	
$p \to z \to A \xrightarrow{f} B, y \to A \xrightarrow{f} C$	$z, z \to f$		$z, \&z$
		$z \to f \to \text{vfun}()$	

Demand Propagation

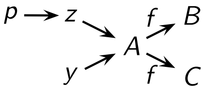
$$p \to z \to A \xrightarrow{f} B, \quad y \to A \xrightarrow{f} C$$

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA) | **Pointer speculation (PDFSA)**

Alias speculation (ADFSA)	Demand (before)	Statement	Demand (after)
		$p = \&z$	
$p \rightarrow z$	p		
		$y = \text{new} A$	
$p \rightarrow z, y \rightarrow A$	p, y		$z, \&z$
		$*p = \text{new} A$	
$p \rightarrow z \rightarrow A, y \rightarrow A$	z, y		$z, \&z$
		$z \rightarrow f = \text{new} B$	
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y		$z, \&z$
		$y \rightarrow f = \text{new} C$	
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$		$z, \&z$
		$z \rightarrow f \rightarrow \text{vfun}()$	

Demand Propagation

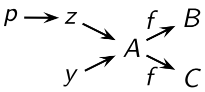


ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA) | Pointer speculation (PDFSA)

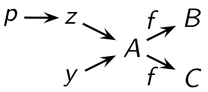
Alias speculation (ADFSA)		Statement		Pointer speculation (PDFSA)
		$p = \&z$		
$p \rightarrow z$	p		$z, \&z$	
		$y = \text{new} A$		
$p \rightarrow z, y \rightarrow A$	p, y		$z, \&z$	
		$*p = \text{new} A$		
$p \rightarrow z \rightarrow A, y \rightarrow A$	z, y		$z, \&z$	
		$z \rightarrow f = \text{new} B$		
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y		$z, \&z$	
		$y \rightarrow f = \text{new} C$		
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$		$z, \&z$	
		$z \rightarrow f \rightarrow \text{vfun}()$		

Demand Propagation



ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)				Pointer speculation (PDFSA)
		$p = \&z$		
$p \to z$	p		$z, \&z$	$p \to z$
		$y = \text{new}A$		
$p \to z, y \to A$	p, y		$z, \&z$	
		$*p = \text{new}A$		
$p \to z \to A, y \to A$	z, y		$z, \&z$	
		$z \to f = \text{new}B$		
$p \to z \to A \xrightarrow{f} B, y \to A$	z, y		$z, \&z$	
		$y \to f = \text{new}C$		
$p \to z \to A \xrightarrow{f} B, y \to A \xrightarrow{f} C$	$z, z \to f$		$z, \&z$	
		$z \to f \to \text{vfun}()$		



Points-to Propagation

ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA) | Pointer speculation (PDFSA)

ADFSA			PDFSA
	$p = \&z$		
$p \rightarrow z$	p	$z, \&z$	$p \rightarrow z$
	$y = \text{new}A$		
$p \rightarrow z, y \rightarrow A$	p, y	$z, \&z$	$p \rightarrow z$
	$*p = \text{new}A$		
$p \rightarrow z \rightarrow A, y \rightarrow A$	z, y	$z, \&z$	
	$z \rightarrow f = \text{new}B$		
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y	$z, \&z$	
	$y \rightarrow f = \text{new}C$		
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$	$z, \&z$	
	$z \rightarrow f \rightarrow \text{vfun}()$		

Points-to Propagation

$$\begin{tikzcd} p \arrow[r] & z \arrow[dr] & & B \\ & & A \arrow[ur, "f"] \arrow[dr, "f"'] & \\ & y \arrow[ur] & & C \end{tikzcd}$$

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

Interprocedural Analysis

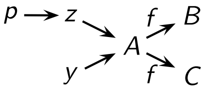
Conclusions

References

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)				Pointer speculation (PDFSA)
		$p = \&z$		
$p \to z$	p	$\downarrow$	$z, \&z$	$p \to z$
		$y = \text{new} A$		
$p \to z, y \to A$	p, y	$\downarrow$	$z, \&z$	$p \to z$
		$*p = \text{new} A$		
$p \to z \to A, y \to A$	z, y	$\downarrow$	$z, \&z$	$p \to z, z \to A$
		$z \to f = \text{new} B$		
$p \to z \to A \xrightarrow{f} B, y \to A$	z, y	$\downarrow$	$z, \&z$	
		$y \to f = \text{new} C$		
$p \to z \to A \xrightarrow{f} B, y \to A \xrightarrow{f} C$	$z, z \to f$	$\downarrow$	$z, \&z$	
		$z \to f \to \text{vfun()}$		

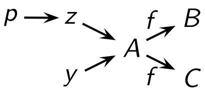
Points-to Propagation



ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA)		Statement		Pointer speculation (PDFSA)
		$p = \&z$		
$p \rightarrow z$	p		$z, \&z$	$p \rightarrow z$
		$y = \text{new}A$		
$p \rightarrow z, y \rightarrow A$	p, y		$z, \&z$	$p \rightarrow z$
		$*p = \text{new}A$		
$p \rightarrow z \rightarrow A, y \rightarrow A$	z, y		$z, \&z$	$p \rightarrow z, z \rightarrow A$
		$z \rightarrow f = \text{new}B$		
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y		$z, \&z$	$p \rightarrow z, z \rightarrow A \xrightarrow{f} B$
		$y \rightarrow f = \text{new}C$		
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$		$z, \&z$	
		$z \rightarrow f \rightarrow \text{vfun}()$		

Points-to Propagation



ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA) | Pointer speculation (PDFSA)

Alias speculation (ADFSA)			Pointer speculation (PDFSA)	Statement
				$p = \&z$
$p \to z$	p	$z, \&z$	$p \to z$	
				$y = \text{new}A$
$p \to z, y \to A$	p, y	$z, \&z$	$p \to z$	
				$*p = \text{new}A$
$p \to z \to A, y \to A$	z, y	$z, \&z$	$p \to z, z \to A$	
				$z \to f = \text{new}B$
$p \to z \to A \xrightarrow{f} B, y \to A$	z, y	$z, \&z$	$p \to z, z \to A \xrightarrow{f} B$	
				$y \to f = \text{new}C$
$p \to z \to A \xrightarrow{f} B, y \to A \xrightarrow{f} C$	$z, z \to f$	$z, \&z$	$p \to z, z \to A \xrightarrow{f} B$	
				$z \to f \to \text{vfun}()$

Points-to Propagation

$$\begin{tikzcd} p \arrow[r] & z \arrow[dr] & & B \\ & & A \arrow[ur, "f"] \arrow[dr, "f"'] & \\ & y \arrow[ur] & & C \end{tikzcd}$$

ADFSA **and** PDFSA **[SCP20]**

Alias speculation (ADFSA)		Statement		Pointer speculation (PDFSA)
		$p = \&z$		
$p \to z$	p	$\downarrow$	$z, \&z$	$p \to z$
		$y = \text{new}A$		
$p \to z, y \to A$	p, y	$\downarrow$	$z, \&z$	$p \to z$
		$*p = \text{new}A$		
$p \to z \to A, y \to A$	z, y	$\downarrow$	$z, \&z$	$p \to z, z \to A$
		$z \to f = \text{new}B$		
$p \to z \to A \xrightarrow{f} B, y \to A$	z, y	$\downarrow$	$z, \&z$	$p \to z, z \to A \xrightarrow{f} B$
		$y \to f = \text{new}C$		
$p \to z \to A \xrightarrow{f} B, y \to A \xrightarrow{f} C$	$z, z \to f$	$\downarrow$	$z, \&z$	$p \to z, z \to A \xrightarrow{f} B$
		$z \to f \to \text{vfun}()$		

$$\begin{tikzcd} p \arrow[r] & z \arrow[dr] & & B \\ & & A \arrow[ur, "f"] \arrow[dr, "f"'] & \\ & y \arrow[ur] & & C \end{tikzcd}$$

$$p \to z \to A \xrightarrow{f} B$$

ADFSA and PDFSA [SCP20]

Alias speculation (ADFSA)				Pointer speculation (PDFSA)
		$p = \&z$		
$p \rightarrow z$	p		$z, \&z$	$p \rightarrow z$
		$y = \text{new}A$		
$p \rightarrow z, y \rightarrow A$	p, y		$z, \&z$	$p \rightarrow z$
		$*p = \text{new}A$		
$p \rightarrow z \rightarrow A$				$\rightarrow A$
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A$	z, y		$z, \&z$	$p \rightarrow z, z \rightarrow A \xrightarrow{f} B$
		$y \rightarrow f = \text{new}C$		
$p \rightarrow z \rightarrow A \xrightarrow{f} B, y \rightarrow A \xrightarrow{f} C$	$z, z \rightarrow f$		$z, \&z$	$p \rightarrow z, z \rightarrow A \xrightarrow{f} B$
		$z \rightarrow f \rightarrow \text{vfun}()$		

Demand-driven analysis with pointer speculation is more precise than with alias speculation in this case

$$\begin{tikzcd} p \arrow[r] & z \arrow[dr] & & B \\ & & A \arrow[ur, "f"] \arrow[dr, "f"'] & \\ & y \arrow[ur] & & C \end{tikzcd}$$

$$p \rightarrow z \rightarrow A \xrightarrow{f} B$$

Uday Khedker
IIT Bombay

Comparing Precision of ADFSA and PDFSA [SCP20]

- ADFSA seeks aliases of the demand raised whereas PDFSA seeks aliases and pointers of the demand raised

Comparing Precision of ADFSA and PDFSA [SCP20]

- ADFSA seeks aliases of the demand raised whereas PDFSA seeks aliases and pointers of the demand raised
- Since a pointer cannot be an object whereas pointee can be, PDFSA can avoid some imprecision that ADFSA cannot

Comparing Precision of ADFSA and PDFSA [SCP20]

- ADFSA seeks aliases of the demand raised whereas PDFSA seeks aliases and pointers of the demand raised
- Since a pointer cannot be an object whereas pointee can be, PDFSA can avoid some imprecision that ADFSA cannot
- PDFSA is at least as precise as ADFSA in each case and more precise in some cases

Comparing Precision of PDFSA and EFSA [SCP20]

- Conventional wisdom
  - Demand-driven methods are more efficient versions of exhaustive methods because they compute only the required information
  - Precision of demand-driven method and exhaustive method is identical

Comparing Precision of PDFSA and EFSA [SCP20]

- Conventional wisdom
  - Demand-driven methods are more efficient versions of exhaustive methods because they compute only the required information
  - Precision of demand-driven method and exhaustive method is identical
- Known Fact

  Demand-driven methods compute less information

Comparing Precision of PDFSA and EFSA [SCP20]

- Conventional wisdom
  - Demand-driven methods are more efficient versions of exhaustive methods because they compute only the required information
  - Precision of demand-driven method and exhaustive method is identical
- Known Fact

  Demand-driven methods compute less information
- Self-evident intuition

  Imprecision caused by abstraction should increase with the amount of data abstracted

Comparing Precision of PDFSA and EFSA [SCP20]

- Conventional wisdom
  - Demand-driven methods are more efficient versions of exhaustive methods because they compute only the required information
  - Precision of demand-driven method and exhaustive method is identical
- Known Fact

  Demand-driven methods compute less information
- Self-evident intuition

  Imprecision caused by abstraction should increase with the amount of data abstracted
- Our work shows how a demand-driven method can be made more precise than the corresponding exhaustive method in some cases

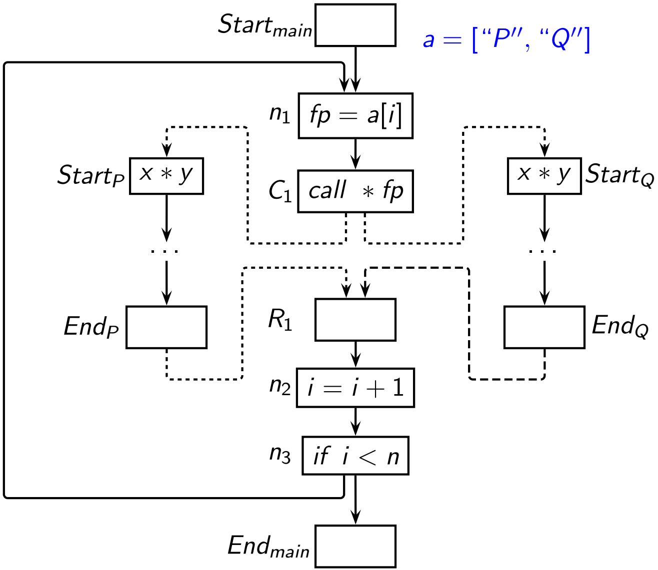
Research Explorations in Interprocedural Analysis

- Broad categories of interprocedural analysis
- Scaling top-down analysis using value contexts and bypassing
- A unified model of context-sensitive methods
- Improving bottom-up analysis by eliminating control flow
- Precise virtual call resolution with demand-driven analysis
- Improving call graphs using callee contexts Next Topic

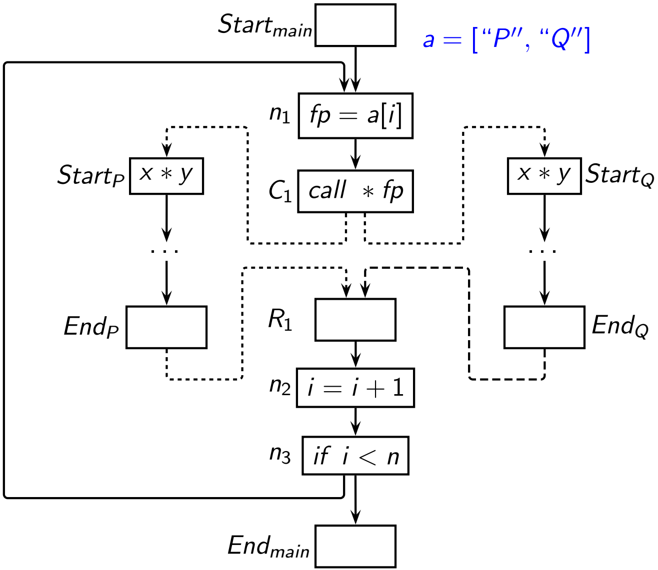
Precise Construction of Call Graphs (or Constructing Callee Contexts) [WiP]

- Problem. Presence of function pointers obscures the caller-callee relationship between procedures.
  - Significant imprecision in the result of any analysis
  - Efficiency and scalability is adversely affected
- Main Challenges.
- Research Goals.
- Additional Benefits.

What Does A Callee Context Mean? [WiP]

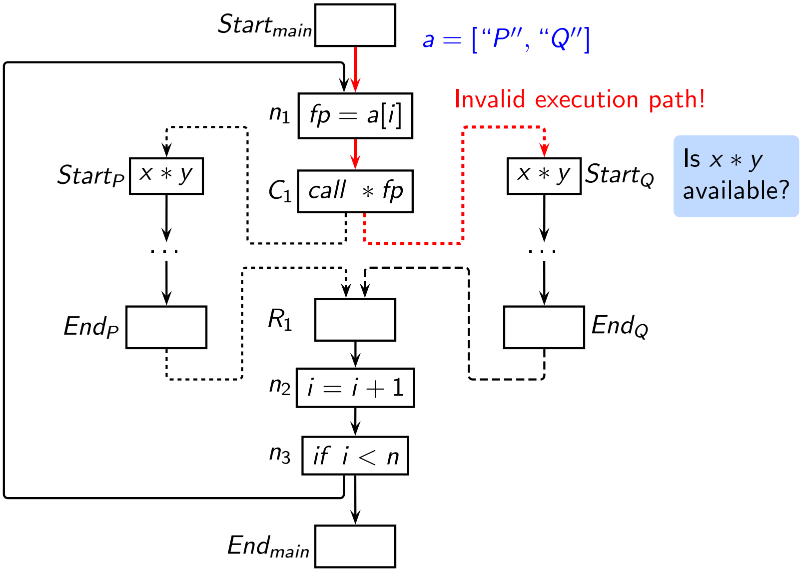


What Does A Callee Context Mean? [WiP]

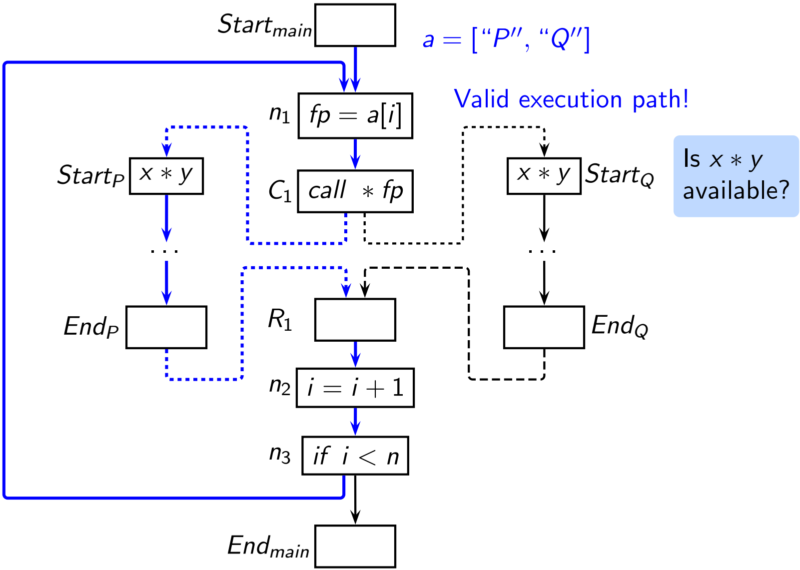


Is $x * y$ available?

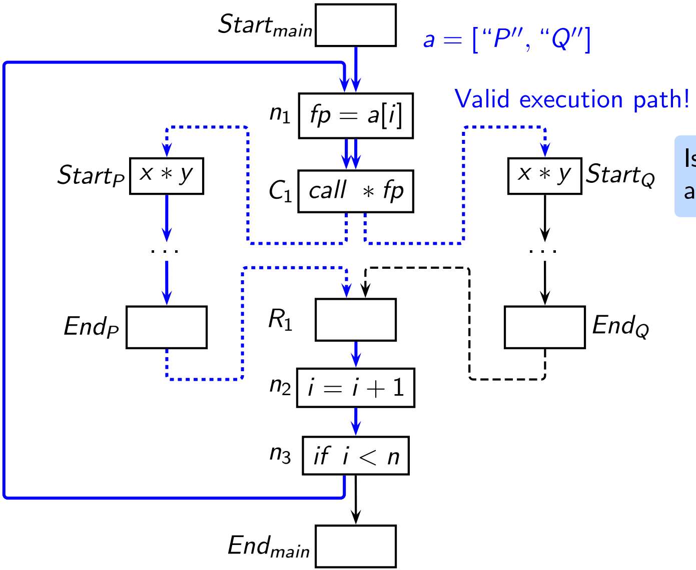
What Does A Callee Context Mean? [WiP]



What Does A Callee Context Mean? [WiP]



What Does A Callee Context Mean? [WiP]



Is $x * y$ available?

Precise Construction of Call Graphs (or Constructing Callee Contexts) [WiP]

- Problem. Presence of function pointers obscures the caller-callee relationship between procedures.
    - Significant imprecision in the result of any analysis
    - Efficiency and scalability is adversely affected
- Main Challenges.
- Research Goals.
- Additional Benefits.

Precise Construction of Call Graphs (or Constructing Callee Contexts) [WiP]

- Problem. Presence of function pointers obscures the caller-callee relationship between procedures.
    - Significant imprecision in the result of any analysis
    - Efficiency and scalability is adversely affected
- Main Challenges. Precise and efficient interprocedural analysis of
    - pointers, and
    - data structure hierarchy declaration and usage
- Research Goals.
- Additional Benefits.

Precise Construction of Call Graphs (or Constructing Callee Contexts) [WiP]

- Problem. Presence of function pointers obscures the caller-callee relationship between procedures.
    - Significant imprecision in the result of any analysis
    - Efficiency and scalability is adversely affected
- Main Challenges. Precise and efficient interprocedural analysis of
    - pointers, and
    - data structure hierarchy declaration and usage
- Research Goals. Order sensitive call disambiguation analysis
    - Flow and context sensitive data structure analysis
    - Creating a mechanism to identify the exact caller to which information should be propagated
- Additional Benefits.

Precise Construction of Call Graphs (or Constructing Callee Contexts) [WiP]

- **Problem.** Presence of function pointers obscures the caller-callee relationship between procedures.
  - Significant imprecision in the result of any analysis
  - Efficiency and scalability is adversely affected
- **Main Challenges.** Precise and efficient interprocedural analysis of
  - pointers, and
  - data structure hierarchy declaration and usage
- **Research Goals.** Order sensitive call disambiguation analysis
  - Flow and context sensitive data structure analysis
  - Creating a mechanism to identify the exact caller to which information should be propagated
- **Additional Benefits.** Precise analysis of programs in object oriented languages

Conclusions

Observations

- Relevant pointer information in a program is very small and sparse
- Data flow propagation in real programs seems to involve a much smaller subset of all possible data flow values

  In large programs that work properly, pointer usage is very disciplined and the core information is very small!

- Precision of analysis can be improved by
  - Excluding infeasible control flow paths
  - Interleaving program analyses

Observations

- Our explorations in both top-down and bottom-up approaches of interprocedural analysis lead us to observe that

The real killer of scalability in program analysis is not
the *data that needs to be computed* but
the *control flow that it is subjected to* in search of precision

Observations

- Our explorations in both top-down and bottom-up approaches of interprocedural analysis lead us to observe that

> The real killer of scalability in program analysis is not
> the *data that needs to be computed* but
> the *control flow that it is subjected to* in search of precision

- For scaling program analysis, we need to optimize away the part of the control flow that does not contribute to data flow
- We achieve this without compromising soundness or precision

The Next Holy Grail in Search of Scalability?

Under-approximated Control Flow ⟵ Original Control Flow ⟶ Over-approximated Control Flow

The Next Holy Grail in Search of Scalability?

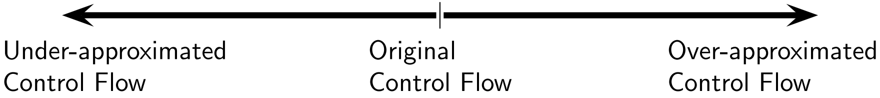
Under-approximated Control Flow	Original Control Flow	Over-approximated Control Flow
May under-approximate data dependences		May over-approximate data dependences

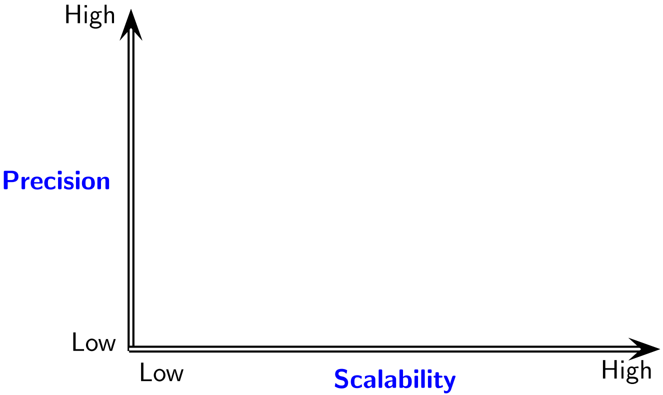
The Next Holy Grail in Search of Scalability?

Under-approximated Control Flow	Original Control Flow	Over-approximated Control Flow
May under-approximate data dependences		May over-approximate data dependences
Unsound	Sound and Precise	Sound and Imprecise

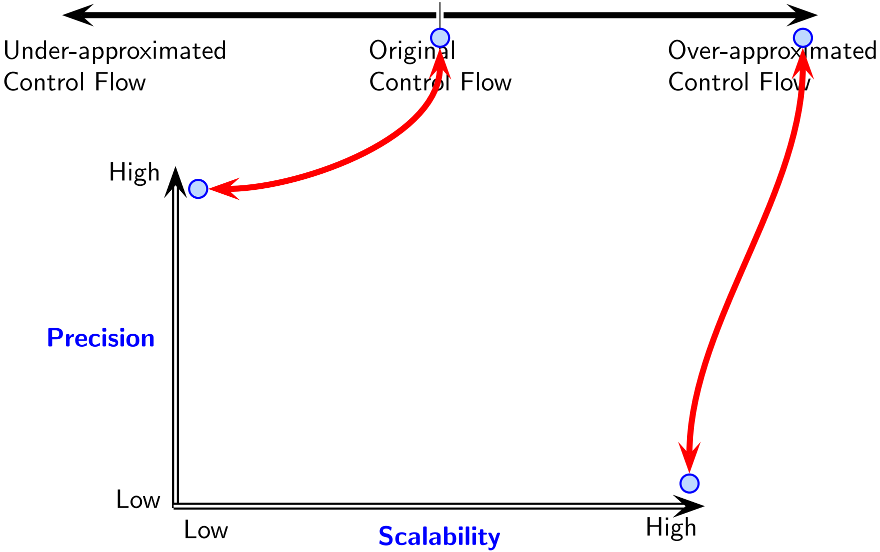
Uday Khedker
IIT Bombay

The Next Holy Grail in Search of Scalability?



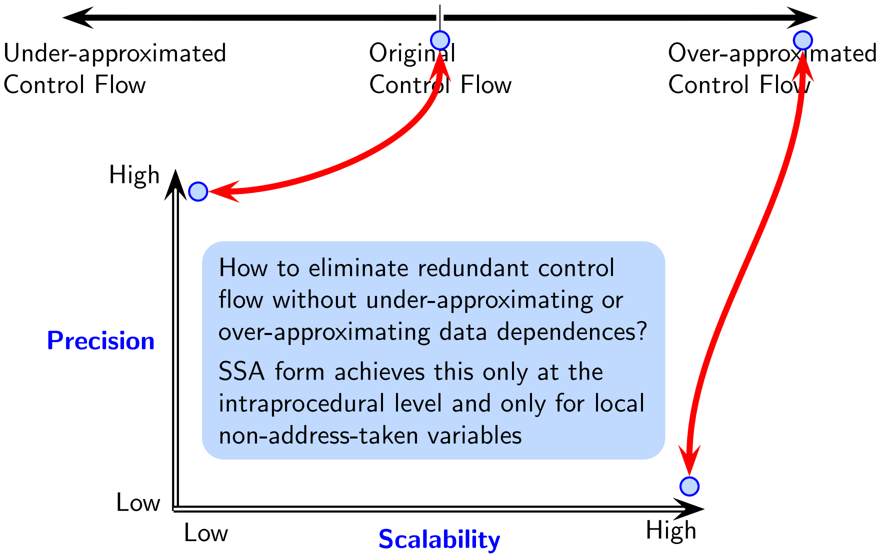


The Next Holy Grail in Search of Scalability?



Uday Khedker
IIT Bombay

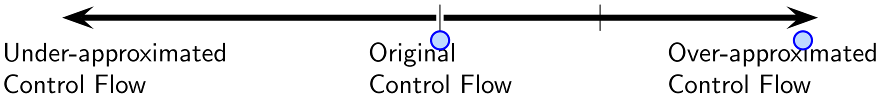
The Next Holy Grail in Search of Scalability?

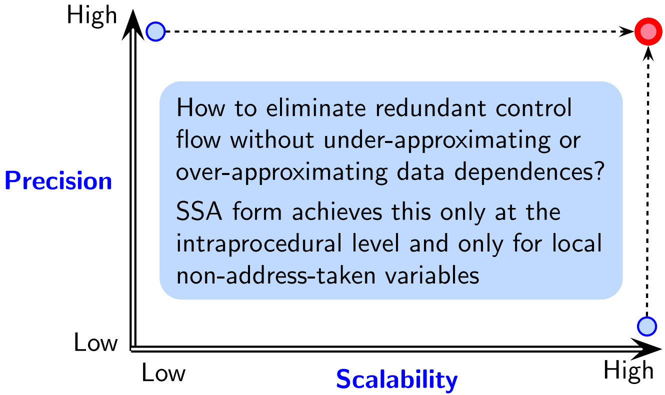


How to eliminate redundant control flow without under-approximating or over-approximating data dependences?

SSA form achieves this only at the intraprocedural level and only for local non-address-taken variables

The Next Holy Grail in Search of Scalability?





A Spectrum of Possible Ways of Performing Computation

exhaustive computation	computation restricted to usable information	avoiding redundant computation	demand driven computation

← What should be computed? →

Maximum Computation ... Minimum Computation

← When should it be computed? →

Early Computation ... Late Computation

A Spectrum of Possible Ways of Performing Computation

exhaustive computation	computation restricted to usable information	avoiding redundant computation	demand driven computation

← What should be computed? →

Maximum Computation ... Minimum Computation

← When should it be computed? →

Early Computation ... Late Computation

Do not compute what you don't need!

Who defines what is needed?

A Spectrum of Possible Ways of Performing Computation

exhaustive computation | computation restricted to usable information | avoiding redundant computation | demand driven computation

What should be computed?

Maximum Computation ⟵ ⟶ Minimum Computation

When should it be computed?

Early Computation ⟵ ⟶ Late Computation

Do not compute what you don't need!

Who defines what is needed? Client

A Spectrum of Possible Ways of Performing Computation

incremental computation

splitting into pre and post computation

exhaustive computation

computation restricted to usable information

avoiding redundant computation

demand driven computation

← What should be computed? →

Maximum Computation — Minimum Computation

← When should it be computed? →

Early Computation — Late Computation

Do not compute what you don't need!

Who defines what is needed? Algorithm, Data Structure

Uday Khedker
IIT Bombay

Talk Title:
PSPA Research

Topic:
Some Meanderings

Intraprocedural Analysis

Interprocedural Analysis

Conclusions

References

A Spectrum of Possible Ways of Performing Computation

incremental computation

splitting into pre and post computation

exhaustive computation

computation restricted to usable information

avoiding redundant computation

demand driven computation

What should

Maximum Computation

When should

Early Computation

Other examples:

- Bottom up summary flow functions
- Value contexts
- Work list based methods
- BDDs

Do not compute what you don't need

Who defines what is needed?

Algorithm, Data Structure

A Spectrum of Possible Ways of Performing Computation

incremental computation | splitting into pre and post computation

exhaustive computation | computation restricted to usable information | avoiding redundant computation | demand driven computation

Maximum Computation ← What should be computed? → Minimum Computation

Early Computation ← When should it be computed? → Late Computation

Do not compute what you don't need!

Who defines what is needed? Definition of Analysis

A Spectrum of Possible Ways of Performing Computation

incremental computation

splitting into pre and post computation

exhaustive computation

computation restricted to usable information

avoiding redundant computation

demand driven computation

← What should be computed? →

Maximum Computation — Minimum Computation

← When should it be computed? →

Early Computation — Late Computation

Do not compute what you don't need!

Who defines what is needed? No One!

A Spectrum of Possible Ways of Performing Computation

incremental computation

splitting into pre and post computation

exhaustive computation

computation restricted to usable information

avoiding redundant computation

demand driven computation

What should be computed?

Maximum Computation

Minimum Computation

When should it be computed?

Early Computation

Late Computation

Do not compute what you don't need!

Who defines what is needed?

These seem orthogonal and may be used together

References

References

- [CSUR21] Swati Jaiswal, Uday P. Khedker, and Alan Mycroft. *A Unified Model for Context-Sensitive Program Analysis: The Blind Men and the Elephant*. ACM Computing Surveys. 54 (6), July 2021.
- [TOPLAS20] Pritam Gharat, Uday P. Khedker, and Alan Mycroft. *Generalized Points-to Graphs: A Precise and Scalable Abstraction for Points-to Analysis.* ACM Transactions on Programming Languages and Systems. 42(2), May 2020.
- [SCP20] Swati Jaiswal, Uday P. Khedker, and Supratik Chakraborty. *Bidirectionality in Flow-Sensitive Demand-Driven Analysis.* Science of Computer Programming Volume 190, Pages 1-49, Jan 2020..
- [CC19] Komal Pathade, Uday P. Khedker. *Path-sensitive MFP solutions in presence of intersecting infeasible control flow path segments.* International Conference on Compiler Construction (CC 2019). USA 2019.
- [CC18] Komal Pathade, Uday P. Khedker. *Computing partially path-sensitive MFP solutions in data flow analyses.* International Conference on Compiler Construction (CC 2018). Austria 2018.

References

- [ISMM17] Vini Kanvar, Uday P. Khedker. *"What's in a name?" Going beyond allocation site names in heap analysis.* International Symposium on Memory Management (ISMM 2017). Spain 2017.
- [SAS16] Pritam M. Gharat, Uday P. Khedker, Alan Mycroft. *Flow and Context Sensitive Points-to Analysis using Generalized Points-to Graphs.* International Static Analysis Symposium (SAS 2016). UK 2016.
- [CSUR16] Vini Kanvar, Uday P. Khedker. *Heap Abstractions for Static Analysis*. ACM Computing Surveys. 49(2): 29:1-29:47, 2016.
- [SOAP13] Rohan Padhye and Uday P. Khedker. *Interprocedural Data Flow Analysis in Soot using Value Contexts.* ACM SIGPLAN International Workshop on the State Of the Art in Java Program Analysis (SOAP 2013). Seattle, USA.
- [SAS12] Uday P. Khedker, Alan Mycroft, and Prashant Singh Rawat. *Liveness Based Pointer Analysis.* International Static Analysis Symposium (SAS 2012). France 2012.

References

- [CC08] Uday Khedker and Bageshri Sathe. *Efficiency, precision, simplicity, and generality in interprocedural data flow analysis: resurrecting the classical call strings method.* International Conference on Compiler Construction (CC 2008), Budapest, Hungary.
- [TOPLAS07] Uday P. Khedker, Amitabha Sanyal, and Amey Karkare. *Heap reference analysis using access graphs.* ACM Transactions on Programming Languages & Systems. Vol. 30, Issue 1, Nov. 2007.

Thank You!