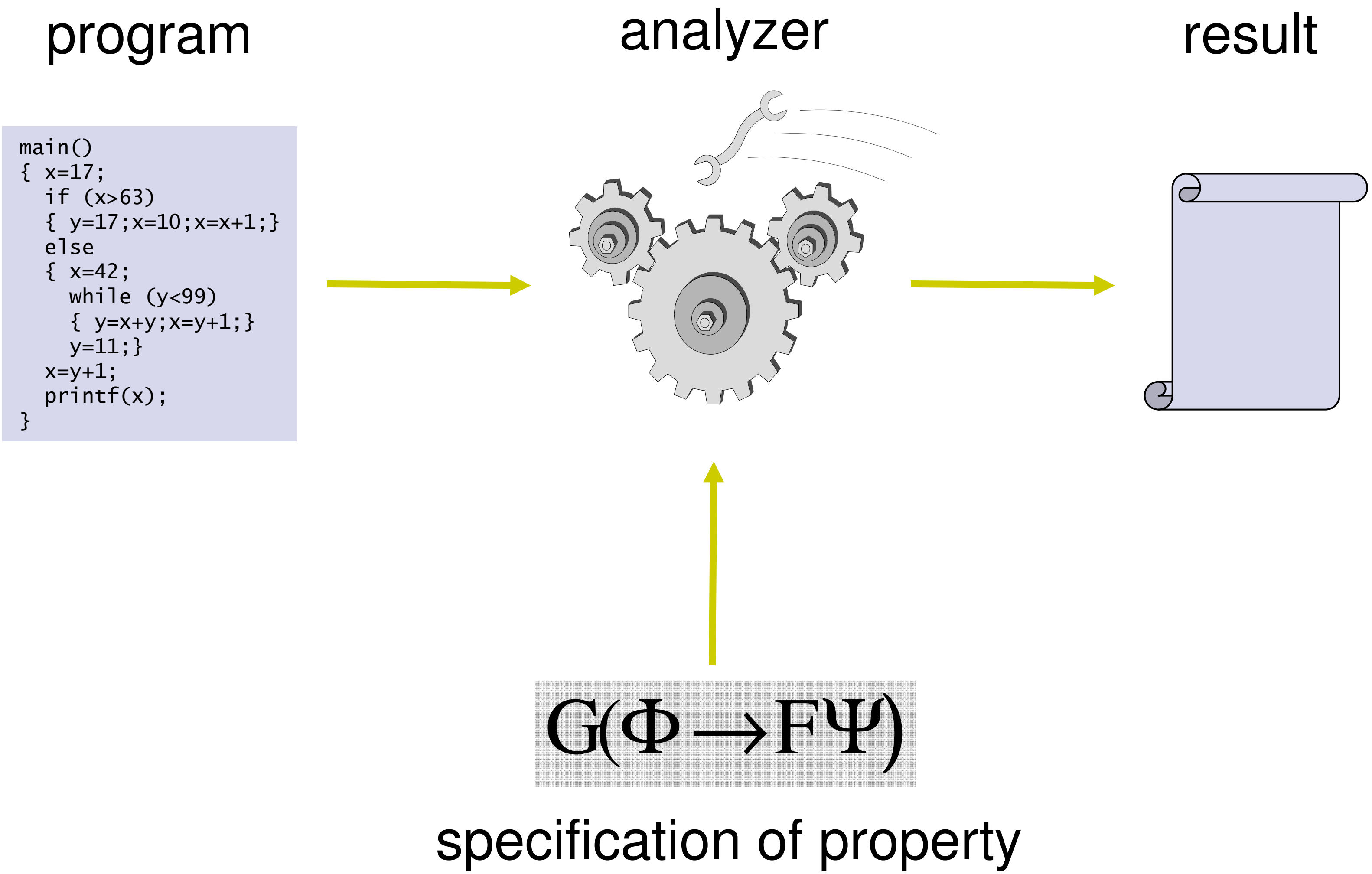
(Optimal) Program Analysis of Sequential and Parallel Programs

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster, Germany

3rd Summer School on
Verification Technology, Systems, and Applications

Luxemburg, September 6-10, 2010

Dream of Automatic Analysis



Fundamental Problem

Rice's Theorem (informal version):

All non-trivial semantic properties of programs from a Turing-complete programming language are undecidable.

Consequence:

For Turing-complete programming languages:

Automatic analyzers of semantic properties, which are both correct and complete are impossible.

☹

What can we do about it?

- Give up „automatic“: interactive approaches:
  - proof calculi, theorem provers, …

- Give up „sound“: ???

- Give up „complete“: approximative approaches:
  - Approximate analyses:
    - data flow analysis, abstract interpretation, type checking, …
  - Analyse weaker formalism:
    - model checking, reachability analysis, equivalence- or preorder-checking, …

What can we do about it?

- Give up „automatic“: interactive approaches:
    - proof calculi, theorem provers, …
- Give up „sound“: ???
- Give up „complete“: approximative approaches:
    - Approximate analyses:
        - data flow analysis, abstract interpretation, type checking, …
    - Analyse weaker formalism:
        - model checking, reachability analysis, equivalence- or preorder-checking, …

Overview

- Introduction
- Fundamentals of Program Analysis
  - Excursion 1
- Interprocedural Analysis
  - Excursion 2
- Analysis of Parallel Programs
  - Excursion 3
  - Appendix
- Conclusion

Apology for not giving proper credit in these lectures !

Overview

- Introduction
- **Fundamentals of Program Analysis**

  Excursion 1

- Interprocedural Analysis

  Excursion 2

- Analysis of Parallel Programs

  Excursion 3

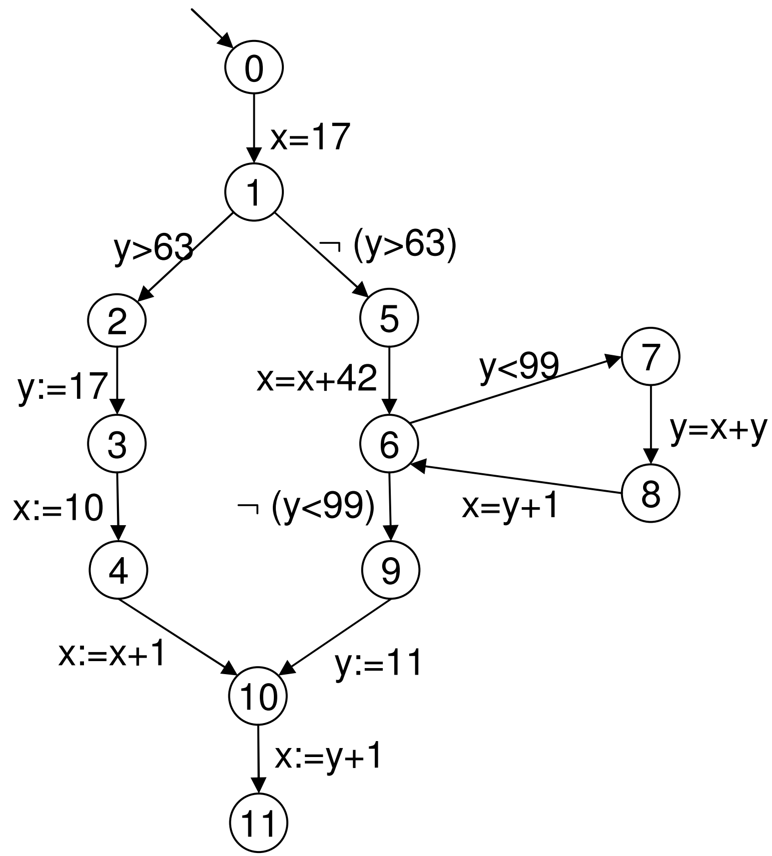
  Appendix

- Conclusion

Apology for not giving proper credit in these lectures !

From Programs to Flow Graphs

```
main()
{ x=17;
  if (x>63)
  { y=17;x=10;x=x+1;}
  else
  { x=x+42;
    while (y<99)
    { y=x+y;x=y+1;}
    y=11;}
  x=y+1;
}
```



Dead Code Elimination

Goal:

find and eliminate assignments that compute values which are never used

Fundamental problem:

undecidability

$\rightarrow$ use approximate algorithm:

e.g.: ignore that guards prohibit certain execution paths

Technique:

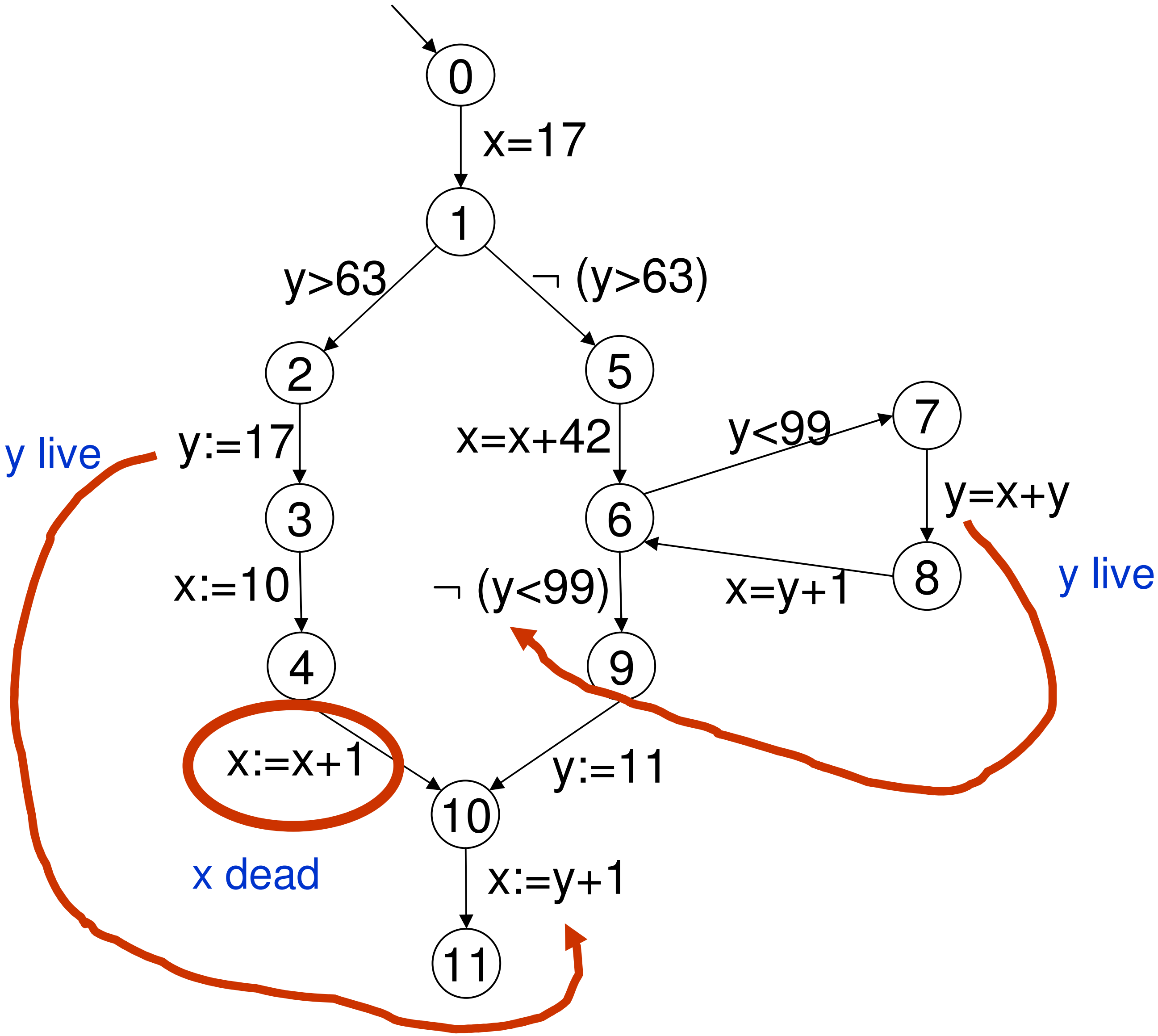
1) perform *live variables* analyses:

variable x is live at program point u iff

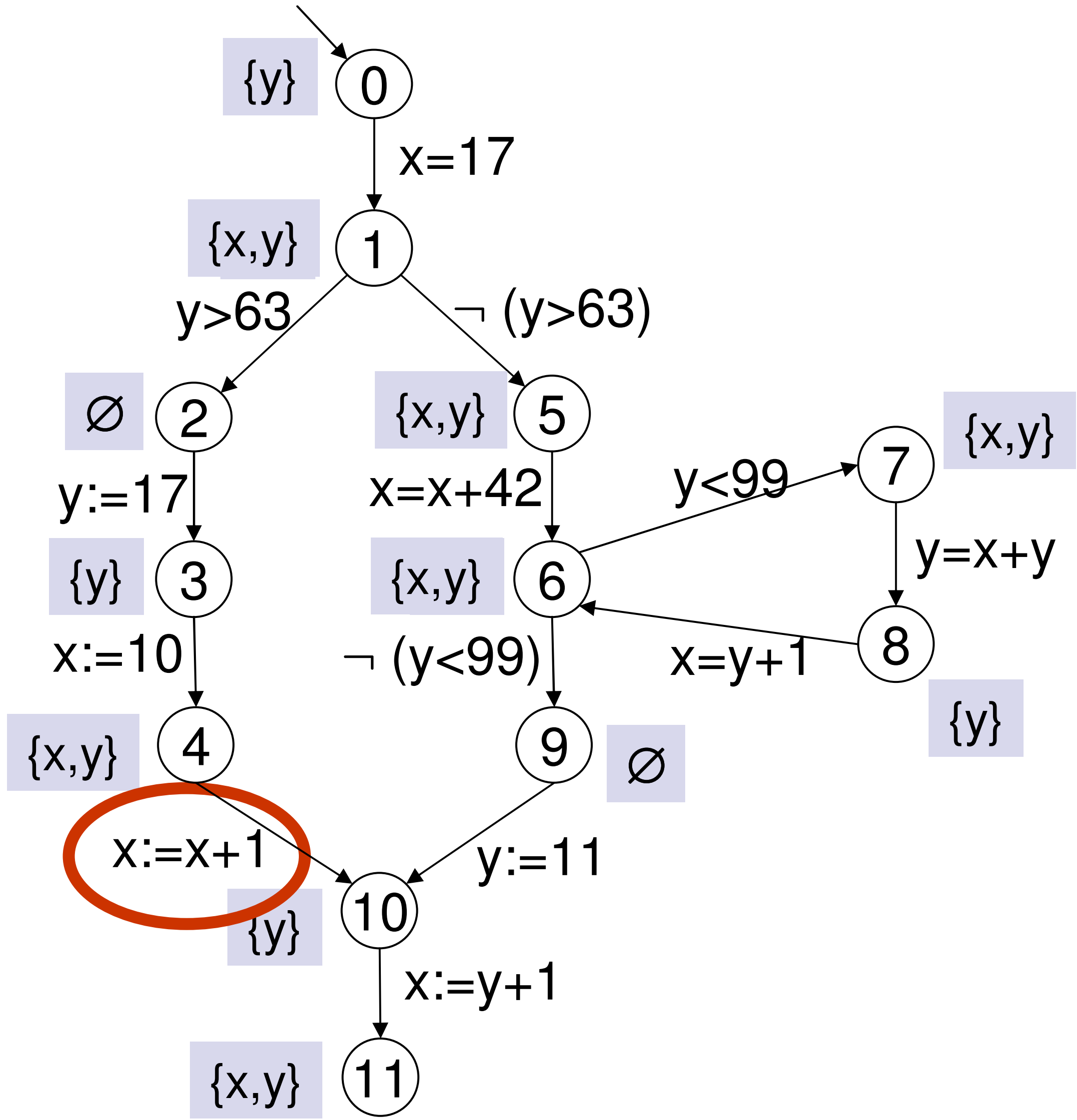
there is a path from u on which x is used before it is modified

2) eliminate assignments to variables that are not live at the target point

Live Variables



Live Variables Analysis



Interpretation of Partial Orders in Approximate Program Analysis

$x \sqsubseteq y$:

- x is more precise information than y.
- y is a correct approximation of x.

$\sqcup X$ for $X \subseteq L$, where $(L,\sqsubseteq)$ is the partial order:

the most precise information consistent with all informations $x \in X$.

Example:

order for live variables analysis:

- $(P(\text{Var}),\subseteq)$ with Var = set of variables in the program

Remark:

often dual interpretation in the literature !

Complete Lattice

Complete lattice $(L,\sqsubseteq)$:

- a partial order $(L,\sqsubseteq)$ for which the least upper bound, $\sqcup X$, exists for all $X \subseteq L$.

In a complete lattice $(L,\sqsubseteq)$:

- $\sqcap X$ exists for all $X \subseteq L$: $\quad \sqcap X = \sqcup \{ x \in L \mid x \sqsubseteq X \}$
- least element $\perp$ exists: $\quad \perp = \sqcup L = \sqcap \emptyset$
- greatest element $\top$ exists: $\quad \top = \sqcup \emptyset = \sqcap L$

Example:

- for any set A let $\mathrm{P}(A) = \{X \mid X \subseteq A\}$ (power set of A).
- $(\mathrm{P}(A),\subseteq)$ is a complete lattice.
- $(\mathrm{P}(A),\supseteq)$ is a complete lattice.

Specifying Live Variables Analysis by a Constraint System

Compute (smallest) solution over $(L,\sqsubseteq) = (\mathrm{P(Var)},\subseteq)$ of:

$$A[fin] \sqsupseteq init, \qquad \text{for } fin\text{, the termination node}$$

$$A[u] \sqsupseteq f_e(A[v]), \qquad \text{for each edge } e = (u,s,v)$$

where $init = \mathrm{Var}$,

$f_e : \mathrm{P(Var)} \to \mathrm{P(Var)}$, $f_e(x) = x \backslash \mathrm{kill}_e \cup \mathrm{gen}_e$, with

- kill_e = variables assigned at e
- gen_e = variables used in an expression evaluated at e

Specifying Live Variables Analysis by a Constraint System

Remarks:

1. Every solution is „correct" (whatever this means).

2. The smallest solution is called **MFP-solution**;
   it comprises a value MFP[u] $\in$ L for each program point u.

3. MFP abbreviates „maximal fixpoint" for traditional reasons.

4. The MFP-solution is the **most precise** one.

Backwards vs. Forward Analyses

Live Variables Analysis is a ***Backwards Analysis,*** i.e.:

- analysis info flows **from target node to source node** of an edge
- the initial inequality is for the termination node of the flow graph

$$A[te] \sqsupseteq init, \qquad \text{for } te, \text{ the termination point}$$

$$A[u] \sqsupseteq f_e(A[v]), \qquad \text{for each edge } e=(u,s,v)\in E$$

Dually, there are ***Forward Analyses*** i.e..:

- analysis info flows **from source node to target node** of an edge.
- the initial inequality is for the start node of the flow graph

$$A[st] \sqsupseteq init, \qquad \text{for } st, \text{the start node}$$

$$A[v] \sqsupseteq f_e(A[u]), \qquad \text{for each edge } e=(u,s,v)\in E$$

Examples: reaching definitions, available expressions, constant propagation, ...

Data-Flow Frameworks

Correctness

- generic properties of frameworks can be studied and proved

Implementation

- efficient, generic implementations can be constructed

Three Questions

- Do (smallest) solutions always exist ?
- How to compute the (smallest) solution ?
- How to justify that a solution is what we want ?

Three Questions

- Do (smallest) solutions always exist ?
- How to compute the (smallest) solution ?
- How to justify that a solution is what we want ?

Knaster-Tarski Fixpoint Theorem

Definitions:

Let $(L,\sqsubseteq)$ be a partial order.

- $f : L \rightarrow L$ is *monotonic* iff $\forall\, x,y \in L :\ x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$.
- $x \in L$ is a *fixpoint* of f iff $f(x)=x$.

Fixpoint Theorem of Knaster-Tarski:

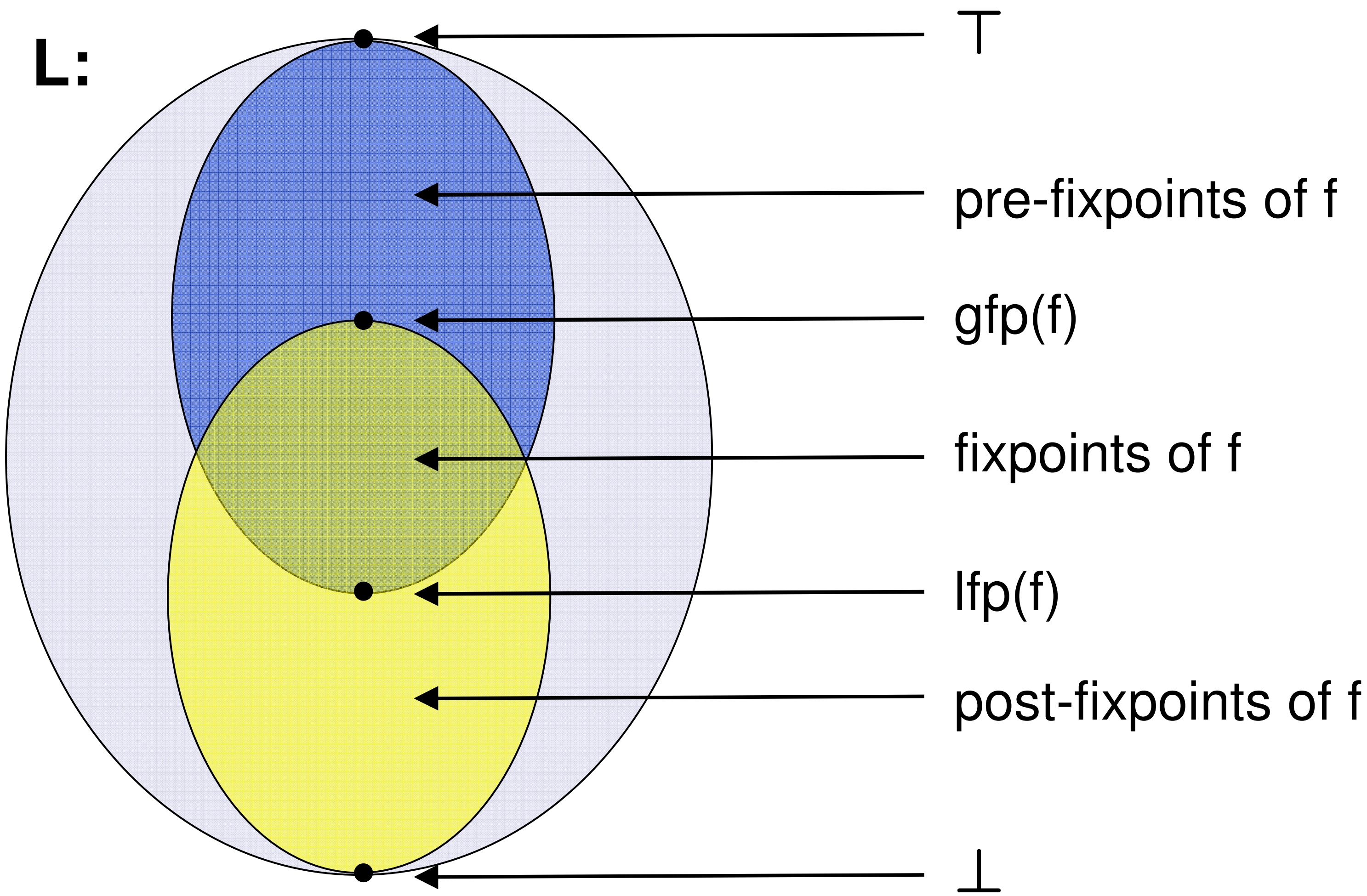
Every monotonic function f on a complete lattice L has a least fixpoint lfp(f) and a greatest fixpoint gfp(f).

More precisely,

$$\mathrm{lfp}(f) = \sqcap \{ x \in L \mid f(x) \sqsubseteq x \} \qquad \textit{least pre-fixpoint}$$

$$\mathrm{gfp}(f) = \sqcup \{ x \in L \mid x \sqsubseteq f(x) \} \qquad \textit{greatest post-fixpoint}$$

Knaster-Tarski Fixpoint Theorem



Picture from: Nielson/Nielson/Hankin, *Principles of Program Analysis*

Smallest Solutions Always Exist

- Define functional $F : L^n \to L^n$ from right hand sides of constraints such that:
  - σ solution of constraint system   iff σ pre-fixpoint of F
- Functional F is monotonic.
- By Knaster-Tarski Fixpoint Theorem:
  - F has a least fixpoint which equals its least pre-fixpoint.

☺

Three Questions

- Do (smallest) solutions always exist ?
- How to compute the (smallest) solution ?
- How to justify that a solution is what we want ?

Workset-Algorithm

$W = \emptyset;$

forall (program points v) $\{\ A[v] = \bot;\ W = W \cup \{v\};\ \}$

$A[fin] = init;$

while $W \neq \emptyset\ \ \{$

$\quad v = Extract(W);$

$\quad$ **forall** $(u, s$ **with** $e = (u, s, v)$ edge$)\ \ \{$

$\quad\quad t = f_e(A[v]);$

$\quad\quad$ **if** $\neg(t \sqsubseteq A[u])\ \{$

$\quad\quad\quad A[u] = A[u] \sqcup t;$

$\quad\quad\quad W = W \cup \{u\};$

$\quad\quad \}$

$\quad \}$

$\}$

Invariants of the Main Loop

a) $A[u] \sqsubseteq \mathrm{MFP}[u]$ f.a. prg. points u

b1) $A[fin] \sqsupseteq init$

b2) $v \notin W \;\Rightarrow\; A[u] \sqsupseteq f_e(A[v])$ f.a. edges $e = (u,s,v)$

If and when workset algorithm terminates:

A is a solution of the constraint system by b1)&b2)

$\Rightarrow \quad A[u] \sqsupseteq MFP[u]$ f.a. u

Hence, with a): $A[u] = MFP[u]$ f.a. u ☺

How to Guarantee Termination

- Lattice $(L,\sqsubseteq)$ has finite heights
  $\Rightarrow$ algorithm terminates after at most
  #prg points · (heights(L)+1)
  iterations of main loop

- Lattice $(L,\sqsubseteq)$ has no infinite ascending chains
  $\Rightarrow$ algorithm terminates

- Lattice $(L,\sqsubseteq)$ has infinite ascending chains:
  $\Rightarrow$ algorithm may not terminate;
  use *widening operators* in order to enforce termination

Widening Operator

[Cousot]

$\nabla: L \times L \to L$ is called a *widening operator* iff

1) $\forall\, x, y \in L: x \sqcup y \sqsubseteq x \nabla y$

2) for all sequences $(l_n)_n$, the (ascending) chain $(w_n)_n$

$$w_0 = l_0, \quad w_{i+1} = w_i \nabla l_{i+1} \text{ for } i > 0$$

stabilizes eventually.

Workset-Algorithm with Widening

$W = \emptyset;$
forall (program points v) $\{\ A[v] = \bot;\ W = W \cup \{v\};\ \}$
$A[\mathit{fin}] = \mathit{init};$
while $W \neq \emptyset$ $\{$
$\quad v = \mathit{Extract}(W);$
$\quad$ **forall** $(u, s$ **with** $e = (u,s,v)$ edge$)$ $\{$
$\qquad t = f_e(A[v]);$
$\qquad$ **if** $\neg(t \sqsubseteq A[u])$ $\{$
$\qquad\quad A[u] = A[u] \triangledown t;$
$\qquad\quad W = W \cup \{u\};$
$\qquad \}$
$\quad \}$
$\}$

Invariants of the Main Loop

a) ~~$A[u] \sqsubseteq \text{MFP}[u]$~~ f.a. prg. points u

b1) $A[fin] \sqsupseteq init$

b2) $v \notin W \;\Rightarrow\; A[u] \sqsupseteq f_e(A[v])$ f.a. edges $e = (u, s, v)$

With a widening operator we enforce termination but we loose invariant a).

Upon termination, we have:

A is a solution of the constraint system by b1)&b2)

$\Rightarrow \quad A[u] \sqsupseteq MFP[u]$ f.a. u

Compute a sound upper approximation (only) !

Example of a Widening Operator: Interval Analysis

The goal

Find save interval for the values of program variables, e.g. of *i* in:

```
for (i=0; i<42; i++)
    if (0<=i and i<42)
   {
        A1 = A+i;
        M[A1] = i;
    }
```

..., e.g., in order to remove the redundant array range check. ☺

Example of a Widening Operator: Interval Analysis

The lattice...

$$(L,\sqsubseteq) = \left(\left\{ [l,u] \mid l \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{+\infty\}, l \leq u \right\} \cup \{\emptyset\},\ \subseteq\right)$$

... has infinite ascending chains, e.g.:

$$[0,0] \subset [0,1] \subset [0,2] \subset \ldots$$

A widening operator:

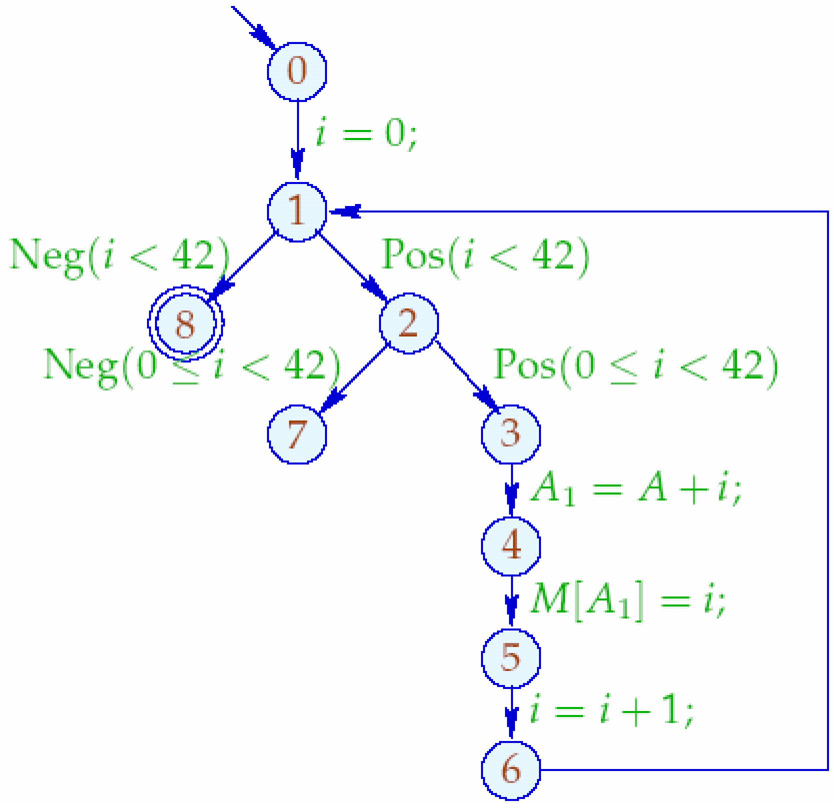
$[l_0,u_0] \bigtriangledown [l_1,u_1] = [l_2,u_2]$, where

$$l_2 = \begin{cases} l_0 & \text{if } l_0 \leq l_1 \\ -\infty & \text{otherwise} \end{cases} \quad \text{and} \quad u_2 = \begin{cases} u_0 & \text{if } u_0 \geq u_1 \\ +\infty & \text{otherwise} \end{cases}$$

A chain of maximal length arising with this widening operator:

$$\emptyset \subset [3,7] \subset [3,+\infty] \subset [-\infty,+\infty]$$

Analyzing the Program with the Widening Operator



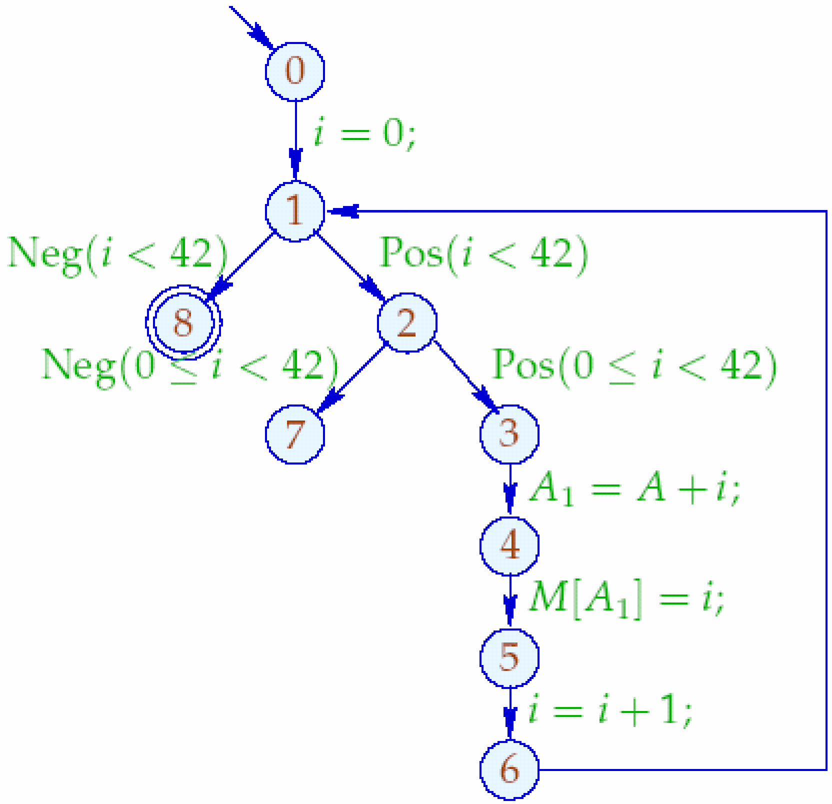
	1		2		3	
	l	u	l	u	l	u
0	$-\infty$	$+\infty$	$-\infty$	$+\infty$		
1	0	0	0	$+\infty$		
2	0	0	0	$+\infty$		
3	0	0	0	$+\infty$		
4	0	0	0	$+\infty$	dito	
5	0	0	0	$+\infty$		
6	1	1	1	$+\infty$		
7	$\perp$		42	$+\infty$		
8	$\perp$		42	$+\infty$		

$\Rightarrow$ Result is far too imprecise ! ☹

Example taken from: H. Seidl, Vorlesung „Programmoptimierung“

Remedy 1: Loop Separators

- Apply the widening operator only at a „*loop separator*“ (a set of program points that cuts each loop).
- We use the loop separator {1} here.

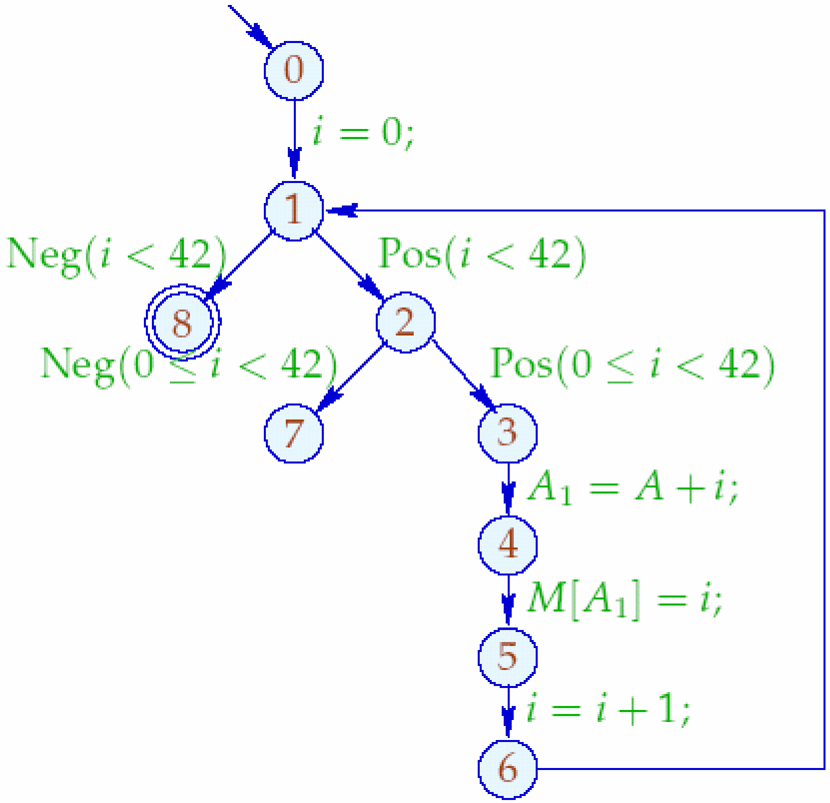


	1		2		3	
	l	u	l	u	l	u
0	$-\infty$	$+\infty$	$-\infty$	$+\infty$		
1	0	0	0	$+\infty$		
2	0	0	0	41		
3	0	0	0	41		
4	0	0	0	41	dito	
5	0	0	0	41		
6	1	1	1	42		
7	$\perp$		$\perp$			
8	$\perp$		42	$+\infty$		

$\Rightarrow$ Identify condition at edge from 2 to 3 as redundant ! ☺

Remedy 2: Narrowing

- Iterate again from the result obtained by widening
  --- Iteration from a prefix-point stays above the least fixpoint ! ---



	0		1		2	
	l	u	l	u	l	u
0	$-\infty$	$+\infty$	$-\infty$	$+\infty$	$-\infty$	$+\infty$
1	0	$+\infty$	0	$+\infty$	0	42
2	0	$+\infty$	0	41	0	41
3	0	$+\infty$	0	41	0	41
4	0	$+\infty$	0	41	0	41
5	0	$+\infty$	0	41	0	41
6	1	$+\infty$	1	42	1	42
7	42	$+\infty$	$\perp$		$\perp$	
8	42	$+\infty$	42	$+\infty$	42	42

$\Rightarrow$ We get the exact result in this example (but not guaranteed) ! ☺

Remarks

- Can use a work-list instead of a work-set
- Special iteration strategies in special situations
- Semi-naive iteration

Recall: Specifying Live Variables Analysis by a Constraint System

Compute (smallest) solution over $(L,\sqsubseteq) = (\mathrm{P}(\mathrm{Var}),\subseteq)$ of:

$$A[\mathit{fin}] \sqsupseteq \mathit{init}, \qquad \text{for } \mathit{fin}\text{, the termination node}$$

$$A[u] \sqsupseteq f_e(A[v]), \qquad \text{for each edge } e = (u,s,v)$$

where $\mathit{init} = \mathrm{Var}$,

$f_e : \mathrm{P}(\mathrm{Var}) \to \mathrm{P}(\mathrm{Var})$, $f_e(x) = x \setminus \mathrm{kill}_e \cup \mathrm{gen}_e$, with

- kill_e = variables assigned at e
- gen_e = variables used in an expression evaluated at e

Recall: Questions

- Do (smallest) solutions always exist ?
- How to compute the (smallest) solution ?
- How to justify that a solution is what we want ?

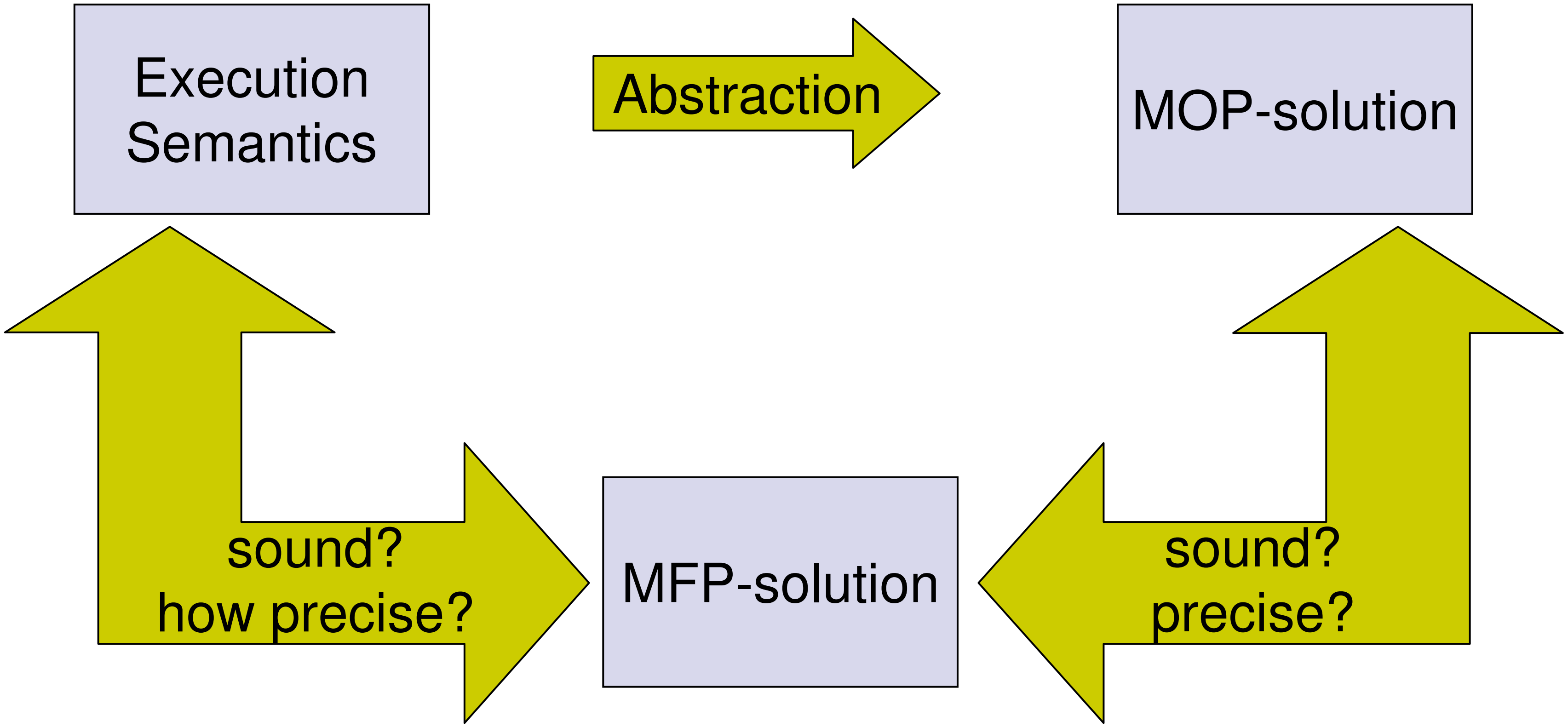
Three Questions

- Do (smallest) solutions always exist ?
- How to compute the (smallest) solution ?
- How to justify that a solution is what we want ?
  - MOP vs MFP-solution
  - Abstract interpretation

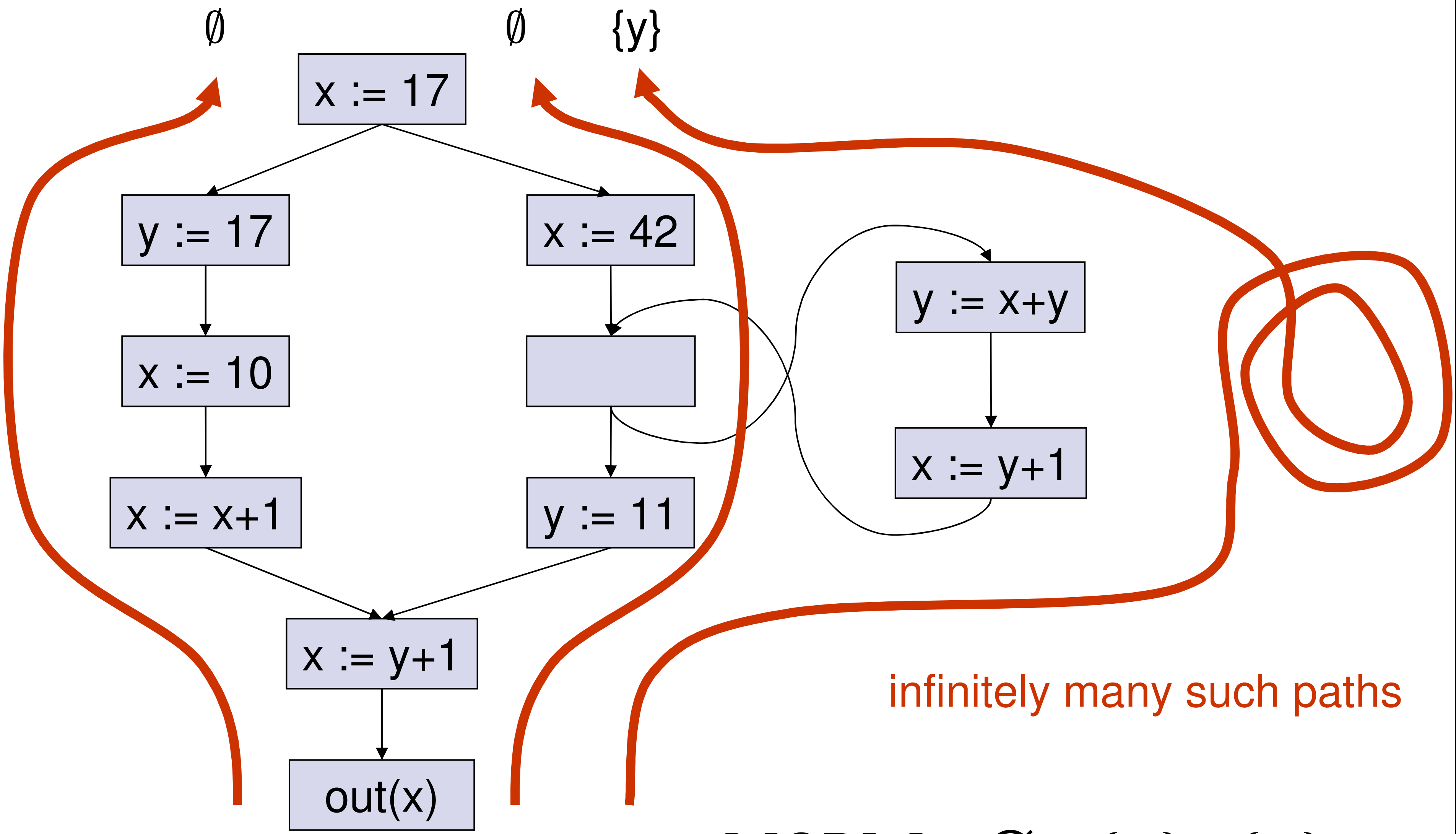
Three Questions

- Do (smallest) solutions always exist ?
- How to compute the (smallest) solution ?
- How to justify that a solution is what we want ?
  - MOP vs MFP-solution
  - Abstract interpretation

Assessing Data Flow Frameworks



Live Variables



infinitely many such paths

$$\mathrm{MOP}[v] = \varnothing \cup \{y\} = \{y\}$$

Meet-Over-All-Paths Solution (MOP)

- Forward Analysis

$$\mathrm{MOP}[u] := \bigsqcup_{p \in \mathrm{Paths}[entry,u]} \mathrm{F}_p(init)$$

- Backward Analysis

$$\mathrm{MOP}[u] := \bigsqcup_{p \in \mathrm{Paths}[u,exit]} \mathrm{F}_p(init)$$

- Here: „Join-over-all-paths“; MOP traditional name

Coincidence Theorem

Definition:

A framework is positively-distributive if

$f(\sqcup X) = \sqcup\{ f(x) \mid x \in X\}$ for all $\emptyset \neq X \subseteq L$, $f \in F$.

Theorem:

For any instance of a positively-distributive framework:

$MOP[u] = MFP[u]$ for all program points u

(if all program points reachable).

Remark:

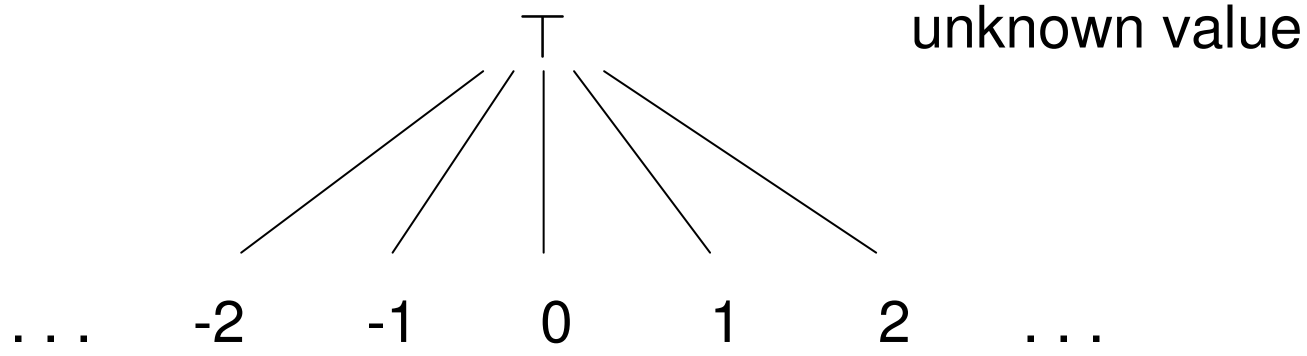
A framework is positively-distributive if a) and b) hold:

(a) it is distributive: $f(x \sqcup y) = f(x) \sqcup f(y)$ f.a. $f \in F$, $x,y \in L$.

(b) it is effective: L does not have infinite ascending chains.

Remark: All bitvector frameworks are distributive and effective.

Lattice for Constant Propagation

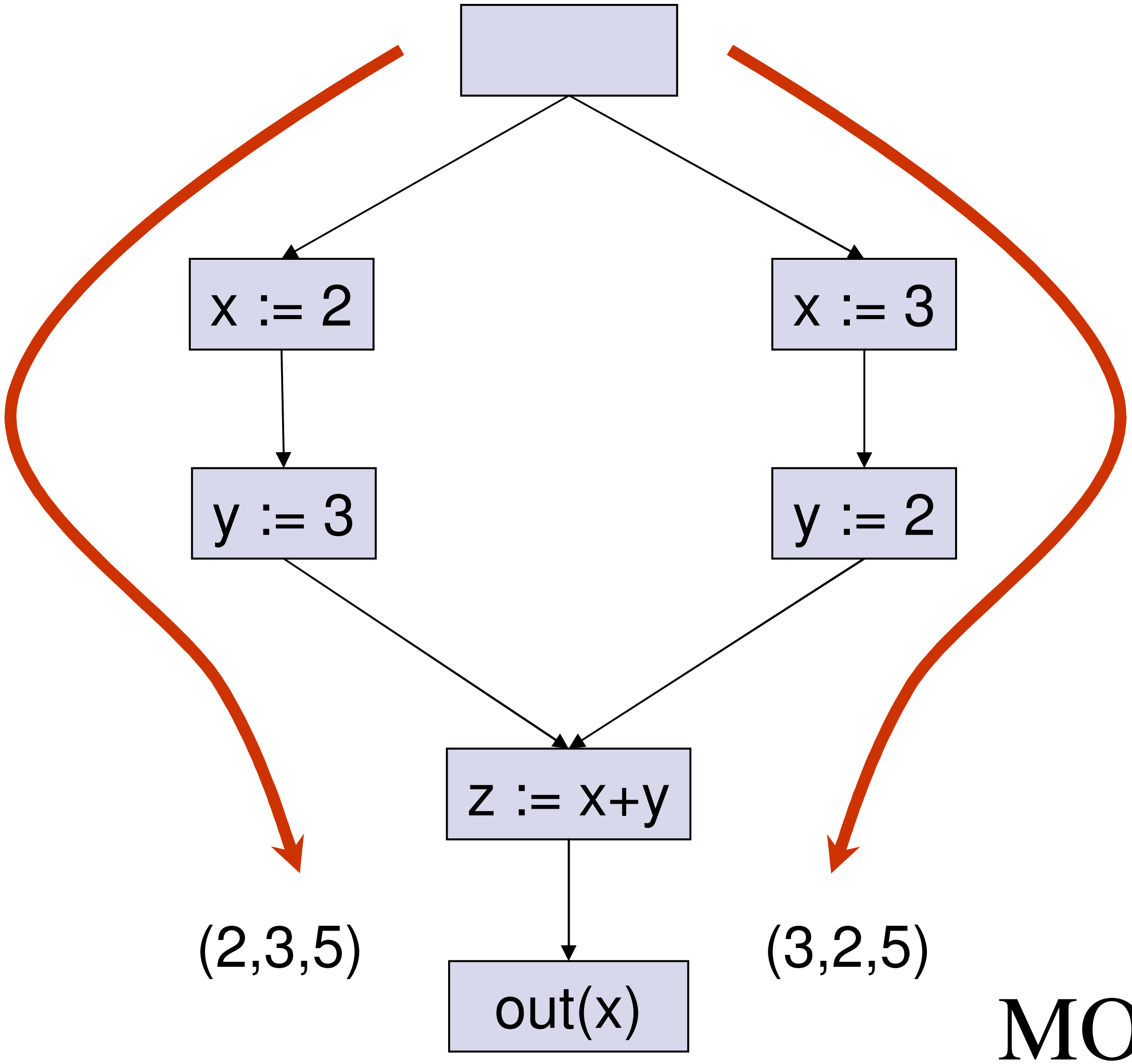


lattice L: $\{\rho \mid \rho : \mathrm{Var} \to (\mathbb{Z} \cup \{\top\})\} \cup \{\bot\}$

$\sqsubseteq$: $\rho \sqsubseteq \rho' \;:\Leftrightarrow\; \rho = \bot \;\vee$

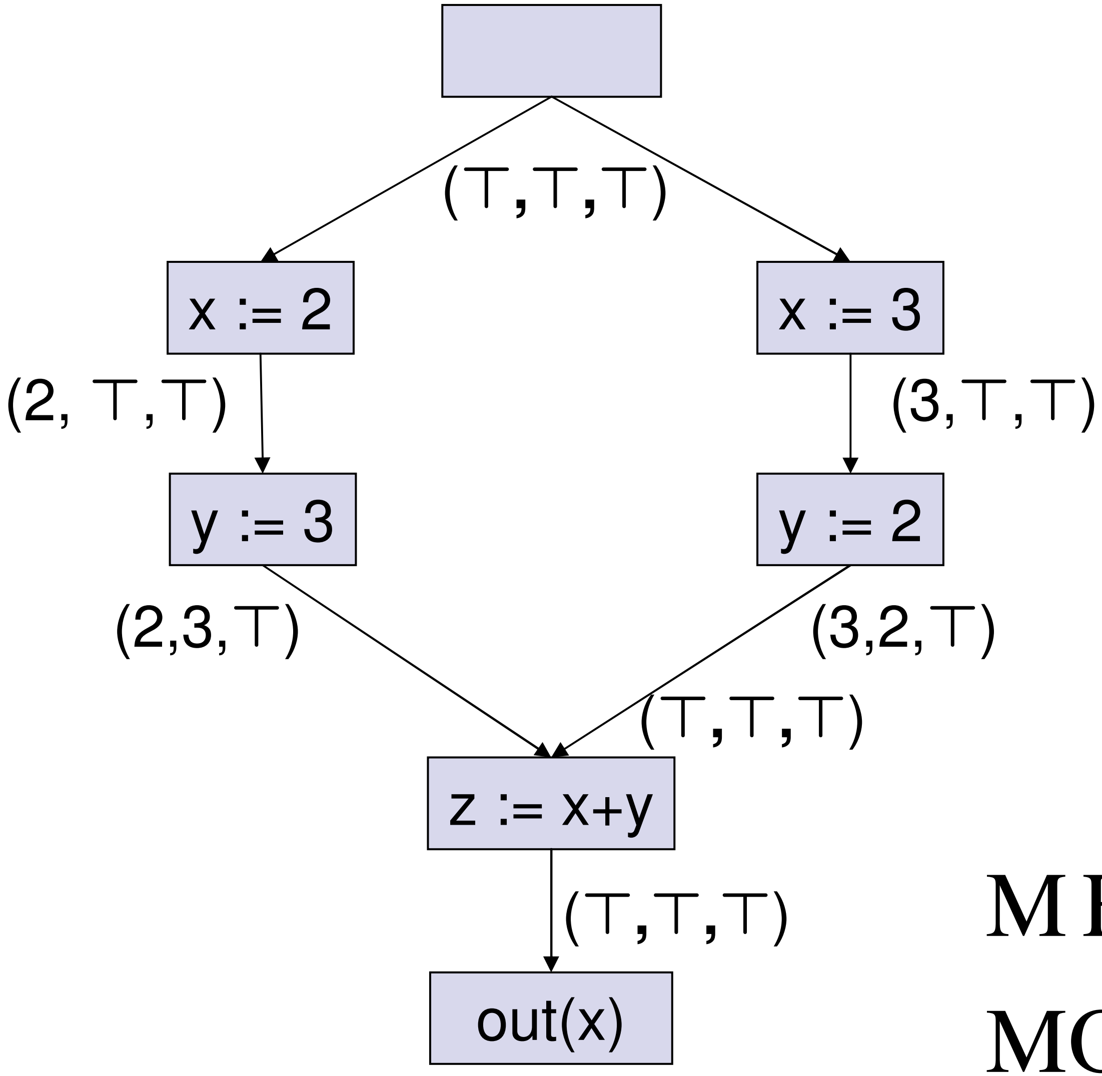
$(\rho, \rho' \neq \bot \wedge \forall x : \rho(x) \sqsubseteq \rho'(x))$

$$(\rho(x), \rho(y), \rho(z))$$



$$\mathrm{MOP}[v] = (\top, \top, 5)$$

$$(\rho(x), \rho(y), \rho(z))$$



$$\mathrm{MFP}[v] = (\top, \top, \top)$$

$$\mathrm{MOP}[v] = (\top, \top, 5)$$

Correctness Theorem

Definition:

A framework is monotone if for all $f \in F$, $x,y \in L$:

$$x \sqsubseteq y \quad \Rightarrow \quad f(x) \sqsubseteq f(y) \ .$$

Theorem:

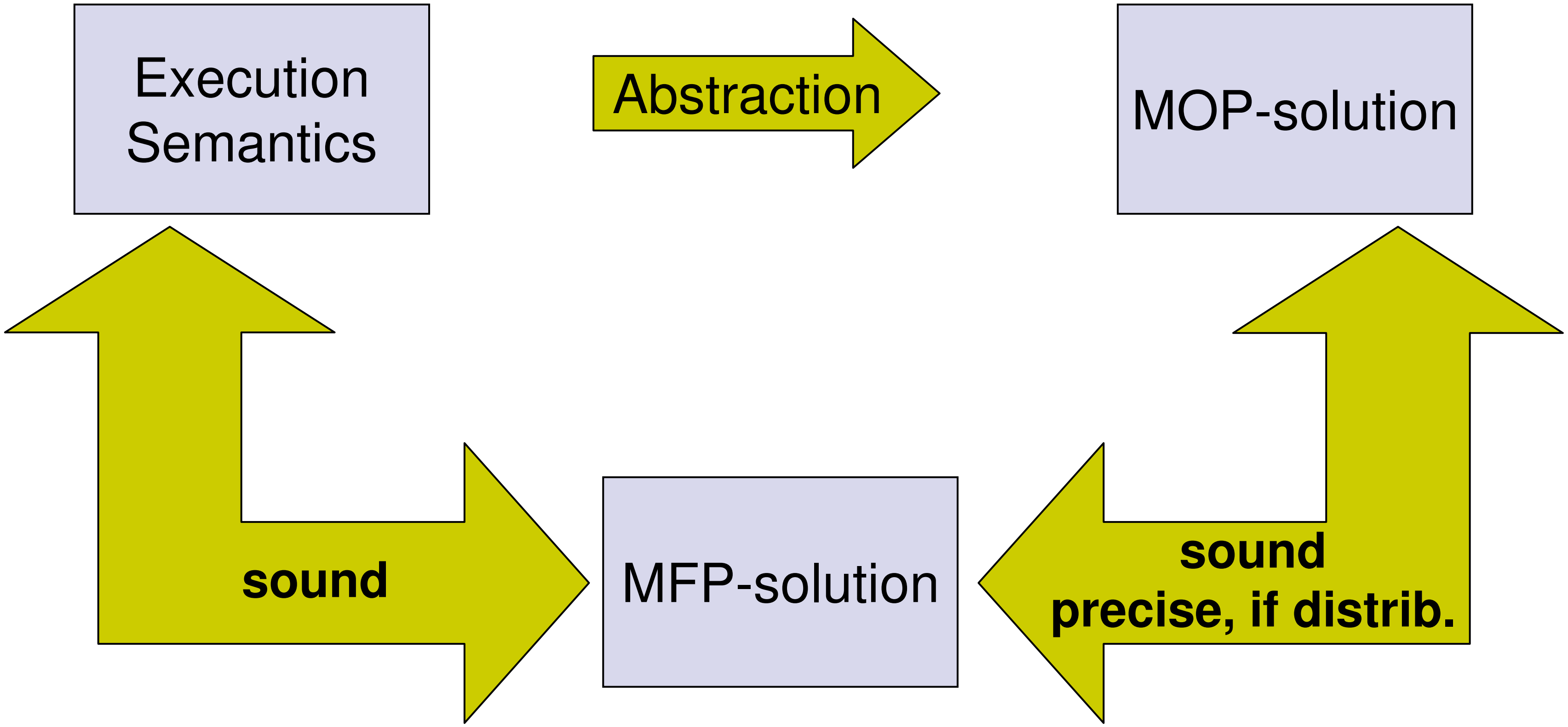
In any monotone framework:

$MOP[u] \sqsubseteq MFP[u]$ for all program points u.

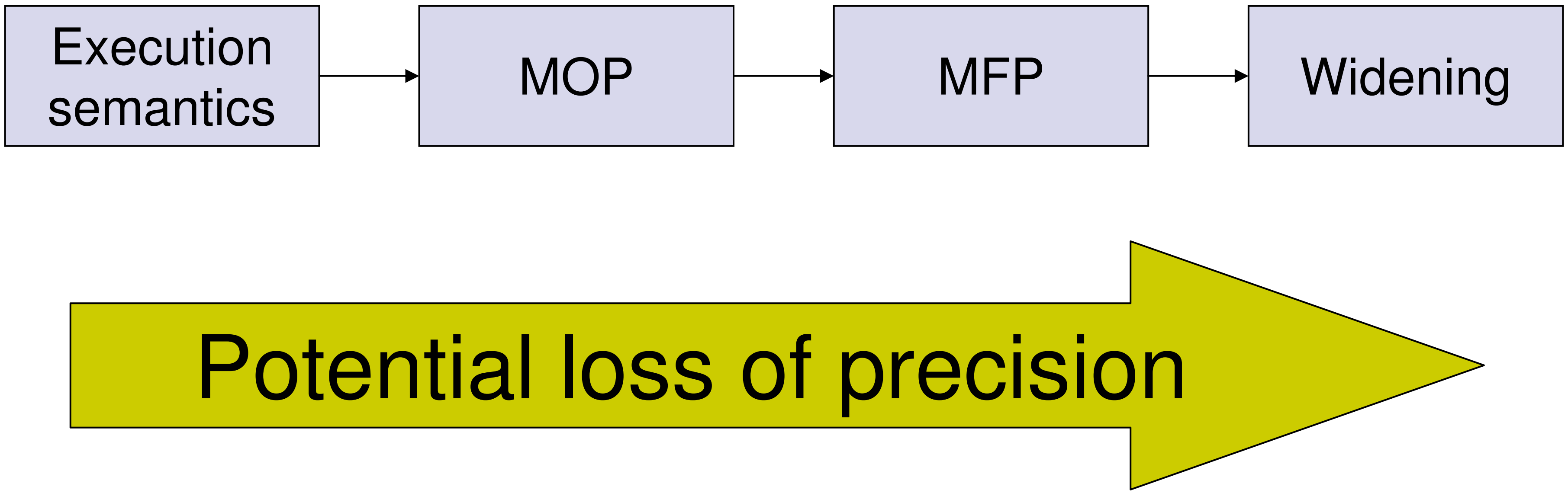
Remark:

Any "reasonable" framework is monotone. ☺

Assessing Data Flow Frameworks



Where Flow Analysis Looses Precision



Three Questions

- Do (smallest) solutions always exist ?
- How to compute the (smallest) solution ?
- How to justify that a solution is what we want ?
  - MOP vs MFP-solution
  - Abstract interpretation

Abstract Interpretation

constraint system for
Reference Semantics
on concrete lattice $(D,\sqsubseteq)$

Replace
concrete operators o
by abstract operators $o^{\#}$

constraint system for
Analysis
on abstract lattice $(D^{\#},\sqsubseteq^{\#})$

MFP

$MFP^{\#}$

Often used as reference semantics:

- sets of reaching runs:
  $(D,\sqsubseteq) = (P(Edges^*),\subseteq)$ or $(D,\sqsubseteq) = (P(Stmt^*),\subseteq)$
- sets of reaching states (*„collecting semantics“*):
  $(D,\sqsubseteq) = (P(\Sigma^*),\subseteq)$ with $\Sigma = Var \to Val$

Abstract Interpretation

constraint system for
Reference Semantics
on concrete lattice $(D, \sqsubseteq)$

MFP

Replace
concrete operators o
by abstract operators $o^\#$

constraint system for
Analysis
on abstract lattice $(D^\#, \sqsubseteq^\#)$

$MFP^\#$

Assume a universally-disjunctive abstraction function $\alpha : D \to D^\#$.

Correct abstract interpretation:

Show $\alpha(o(x_1,\dots,x_k)) \sqsubseteq^\# o^\#(\alpha(x_1),\dots,\alpha(x_k))$ f.a. $x_1,\dots,x_k \in L$, operators o

Then $\alpha(MFP[u]) \sqsubseteq^\# MFP^\#[u]$ f.a. u

Correct and precise abstract interpretation:

Show $\alpha(o(x_1,\dots,x_k)) = o^\#(\alpha(x_1),\dots,\alpha(x_k))$ f.a. $x_1,\dots,x_k \in L$, operators o

Then $\alpha(MFP[u]) = MFP^\#[u]$ f.a. u

Use this as a guideline for designing correct (and precise) analyses !

Abstract Interpretation

Constraint system for reaching runs:

$$R[st] \supseteq \{\varepsilon\}, \qquad \text{for } st\text{, the start node}$$

$$R[v] \supseteq R[u] \cdot \{\langle e \rangle\}, \qquad \text{for each edge } e = (u, s, v)$$

Operational justification:

Let $\underline{R}[u]$ be components of smallest solution over $P(Edges^{*})$. Then

$$\underline{R}[u] = R^{op}[u] =_{def} \{ r \in Edges^{*} \mid st \xrightarrow{r} u \} \qquad \text{for all } u$$

Prove:

a) $R^{op}[u]$ satisfies all constraints (direct)

$\Rightarrow \quad \underline{R}[u] \subseteq R^{op}[u]$ f.a. u

b) $w \in R^{op}[u] \Rightarrow w \in \underline{R}[u]$ (by induction on $|w|$)

$\Rightarrow \quad R^{op}[u] \subseteq \underline{R}[u]$ f.a. u

Abstract Interpretation

Constraint system for reaching runs:

$$R[st] \supseteq \{\varepsilon\}, \qquad \text{for } st\text{, the start node}$$

$$R[v] \supseteq R[u] \cdot \{\langle e \rangle\}, \qquad \text{for each edge } e = (u, s, v)$$

Derive the analysis:

Replace

$\{\varepsilon\}$ by init

$(\bullet) \cdot \{\langle e \rangle\}$ by f_e

Obtain abstracted constraint system:

$$R^{\#}[st] \sqsupseteq init, \qquad \text{for } st\text{, the start node}$$

$$R^{\#}[v] \sqsupseteq f_e(R^{\#}[u]), \qquad \text{for each edge } e = (u, s, v)$$

Abstract Interpretation

MOP-Abstraction:

Define α_{MOP} : P(Edges$^{*)}$ $\rightarrow$ L by

$$\alpha_{MOP}(R) = \sqcup \{ f_r(init) \mid r \in R \} \quad \text{where } f_\varepsilon = Id,\ f_{s\cdot\langle e\rangle} = f_e \circ f_s$$

Remark:

For all transfer functions f_e are monotone, the abstraction is correct:

$\alpha_{MOP}(\underline{R}[u]) \sqsubseteq \underline{R}^{\#}[u]$ f.a. prg. points u

If all transfer function f_e are universally-distributive, the abstraction is correct and precise:

$\alpha_{MOP}(\underline{R}[u]) = \underline{R}^{\#}[u]$ f.a. prg. points u

Justifies MOP vs. MFP theorems (*cum grano salis*). ☺

Overview

- Introduction
- Fundamentals of Program Analysis

Excursion 1

- Interprocedural Analysis

Excursion 2

- Analysis of Parallel Programs

Excursion 3

Appendix

- Conclusion

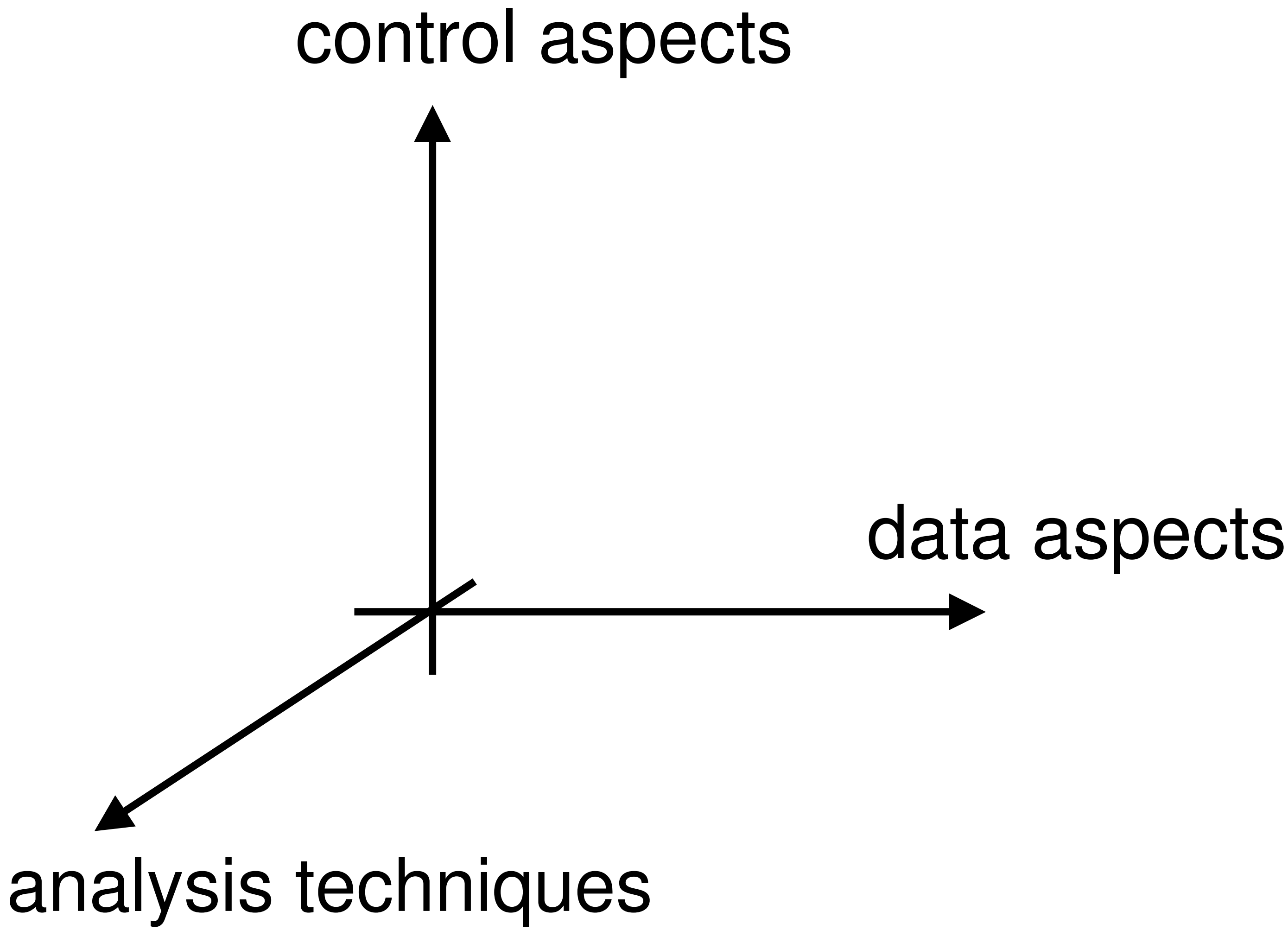
Challenges for Automatic Analysis

- Data aspects:
  - infinite number domains
  - dynamic data structures (e.g. lists of unbounded length)
  - pointers
  - ...

- Control aspects:
  - recursion
  - concurrency
  - creation of processes / threads
  - synchronization primitives (locks, monitors, communication stmts ...)
  - ...

$\Rightarrow$ infinite/unbounded state spaces

Classifying Analysis Approaches



(My) Main Interests of Recent Years

Data aspects:

- algebraic invariants over $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{Z}_m$ ($m = 2^n$) in sequential programs, partly with recursive procedures
- invariant generation relative to Herbrand interpretation

Control aspects:

- recursion
- concurrency with process creation / threads
- synchronization primitives, in particular locks/monitors

Technics:

- fixpoint-based
- automata-based
- (linear) algebra
- syntactic substitution-based techniques
- ...

Overview

- Introduction
- Fundamentals of Program Analysis
    - **Excursion 1**
- Interprocedural Analysis
    - Excursion 2
- Analysis of Parallel Programs
    - Excursion 3
    - Appendix
- Conclusion

A Note on Karr´s Algorithm

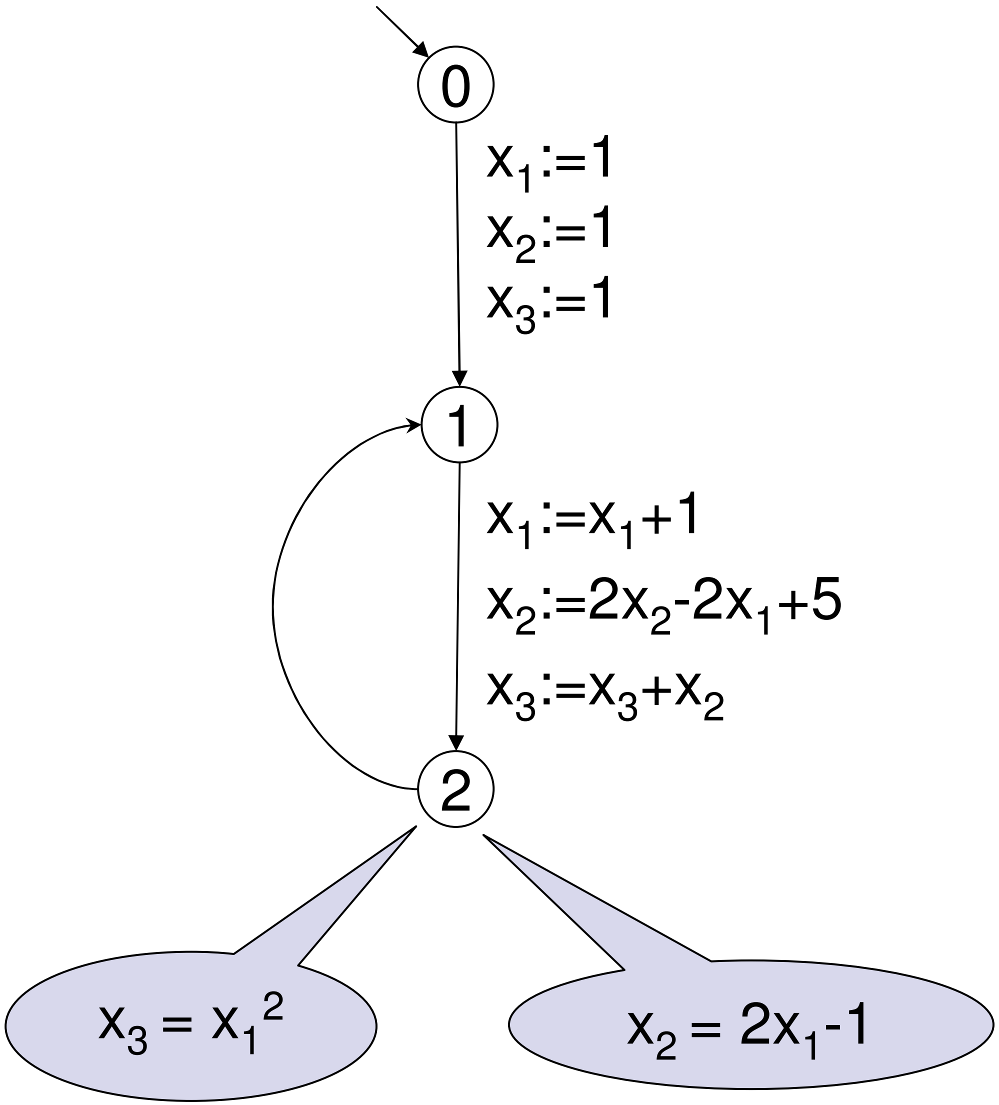
Markus Müller-Olm
FernUniversität Hagen
(on leave from Universität Dortmund)

Joint work with

Helmut Seidl (TU München)

ICALP 2004, Turku, July 12-16, 2004

What this Excursion is About...



Affine Programs

- Basic Statements:
  - affine assignments: $x_1 := x_1 - 2x_3 + 7$
  - unknown assignments: $x_i := ?$

  $\rightarrow$ abstract too complex statements

- Affine Programs:
  - control flow graph $G=(N,E,st)$, where
    - N finite set of program points
    - $E \subseteq N \times \mathit{Stmt} \times N$ set of edges
    - $st \in N$ start node

- Note: non-deterministic instead of guarded branching

The Goal: Precise Analysis

Given an affine program, determine for each program point

- all valid affine relations:

$$a_0 + \sum a_i x_i = 0 \qquad a_i \in \mathbb{Q}$$

$5x_1+7x_2-42=0$

More ambitious goal:

- determine all valid polynomial relations (of degree $\leq$ d):

$$p(x_1,\ldots,x_k) = 0 \qquad p \in \mathbb{Q}[x_1,\ldots,x_n]$$

$5x_1x_2^2+7x_3^3=0$

Applications of Affine (and Polynomial) Relations

- Data-flow analysis:
  - definite equalities: $x = y$
  - constant detection: $x = 42$
  - discovery of symbolic constants: $x = 5yz+17$
  - complex common subexpressions: $xy+42 = y^2+5$
  - loop induction variables

- Program verification
  - strongest valid affine (or polynomial) assertions (cf. Petri Net invariants)

Karr´s Algorithm

[Karr, 1976]

- Determines valid affine relations in programs.

- Idea: Perform a data-flow analysis maintaining for each program point a set of affine relations, i.e., a linear equation system.

- Fact: Set of valid affine relations forms a vector space of dimension at most k+1, where k = #program variables.

  $\Rightarrow$ can be represented by a basis.
  $\Rightarrow$ forms a complete lattice of height k+1.

Deficiencies of Karr´s Algorithm

- Basic operations are complex
  - „non-invertible“ assignments
  - union of affine spaces
- $O(n \cdot k^4)$ arithmetic operations
  - n size of the program
  - k number of variables
- Numbers may have exponential length

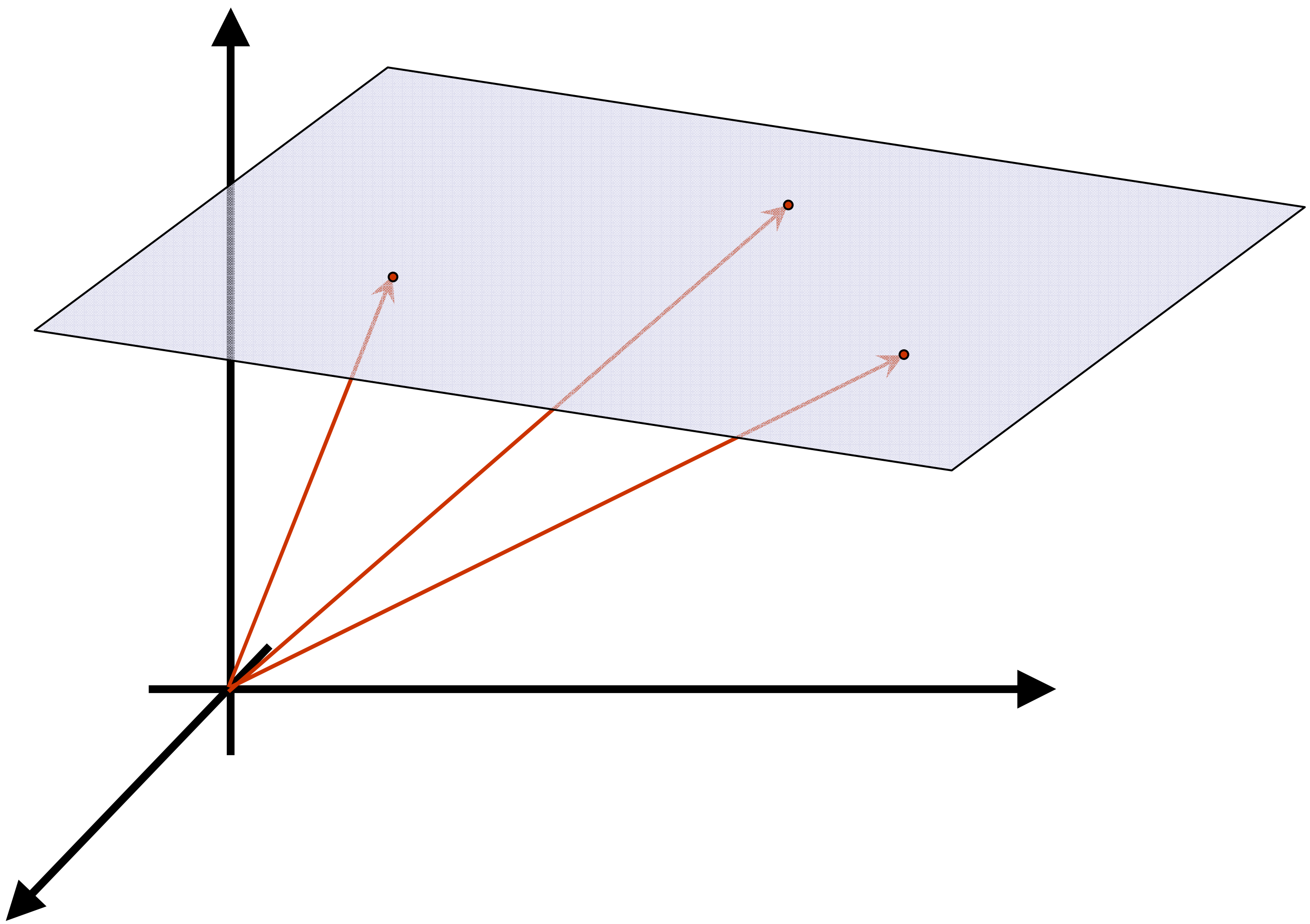
Our Contribution

- Reformulation of Karr´s algorithm:
  - basic operations are simple
  - $O(n \cdot k^3)$ arithmetic operations
  - numbers stay of polynomial length: $O(n \cdot k^2)$

  Moreover:
  - generalization to polynomial relations of bounded degree
  - show, algorithm finds all affine relations in „affine programs“

- Ideas:
  - represent affine spaces by affine bases instead of lin. eq. syst.
  - use semi-naive fixpoint iteration
  - keep a reduced affine basis for each program point during fixpoint iteration

Affine Basis



Concrete Collecting Semantics

Smallest solution over subsets of $\mathbb{Q}^k$ of:

$$V[st] \supseteq \mathbb{Q}^k$$

$$V[v] \supseteq f_s(V[u]) \,, \qquad \text{for each edge } (u,s,v)$$

where

$$f_{x_i:=t}(X) = \{x[x_i \mapsto t(x)] \mid x \in X\}$$

$$f_{x_i:=?}(X) = \{x[x_i \mapsto c] \mid x \in X, c \in \mathbb{Q}\}$$

First goal: compute affine hull of $V[u]$ for each u.

Abstraction

Affine hull:

$$aff(X) = \{\sum \lambda_i x_i \mid x_i \in X, \lambda_i \in \mathbb{Q}, \sum \lambda_i = 1\}$$

The affine hull operator is a closure operator:

$$aff(X) \supseteq X,\ aff(aff(X)) = X,\ X \subseteq Y \Rightarrow aff(X) \subseteq aff(Y)$$

$\Rightarrow$ Affine subspaces of $\mathbb{Q}^k$ ordered by set inclusion form a complete lattice:

$$(D, \sqsubseteq) = \left(\left\{X \subseteq \mathbb{Q}^k \mid aff(X) = X\right\}, \subseteq\right).$$

Affine hull is even a precise abstraction:

Lemma: $f_s(aff(X)) = aff(f_s(X))$.

Abstract Semantics

Smallest solution over (D,$\sqsubseteq$) of:

$$V^{\#}[st] \sqsupseteq \mathbb{Q}^k$$

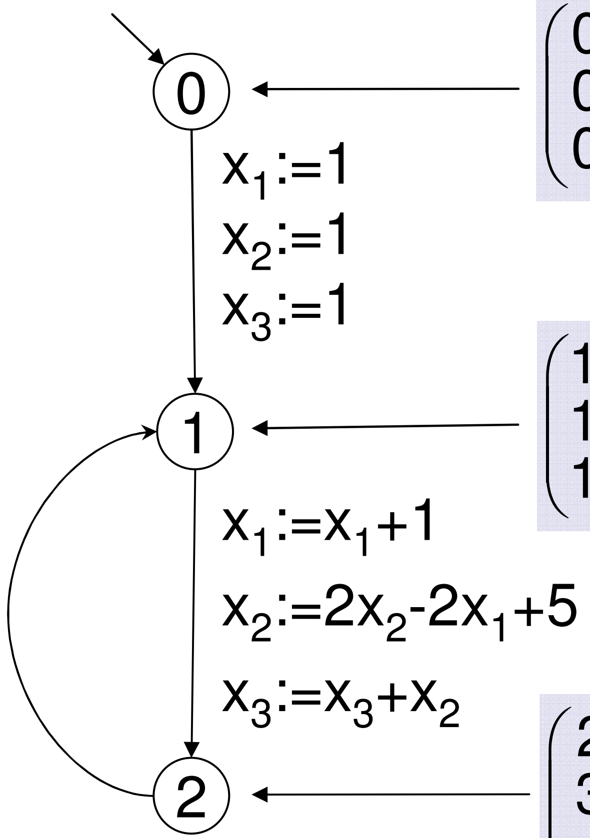
$$V^{\#}[v] \sqsupseteq f_s(V^{\#}[u]) , \quad \text{for each edge } (u,s,v)$$

Lemma: $V^{\#}[u] = aff(V[u])$ for all program points u.

Basic Semi-naive Fixpoint Algorithm

$$
\begin{array}{l}
\textbf{forall}\ (v \in N)\ \ G[v] = \emptyset; \\
G[st] = \{0, e_1, \ldots, e_k\}; \\
W = \{(st, 0), (st, e_1), \ldots, (st, e_k)\}; \\
\textbf{while}\ W \neq \emptyset\ \ \{ \\
\quad (u, x) = Extract(W); \\
\quad \textbf{forall}\ (s, v\ \textbf{with}\ (u, s, v) \in E)\ \ \{ \\
\quad\quad t = [\![s]\!]\, x; \\
\quad\quad \textbf{if}\ (t \notin aff(G[v]))\ \ \{ \\
\quad\quad\quad G[v] = G[v] \cup \{t\}; \\
\quad\quad\quad W = W \cup \{(v, t)\}; \\
\quad\quad \} \\
\quad \} \\
\}
\end{array}
$$

Example



$$\begin{pmatrix}0\\0\\0\end{pmatrix},\begin{pmatrix}1\\0\\0\end{pmatrix},\begin{pmatrix}0\\1\\0\end{pmatrix},\begin{pmatrix}0\\0\\1\end{pmatrix}$$

$$\begin{pmatrix}1\\1\\1\end{pmatrix}\quad\begin{pmatrix}2\\3\\4\end{pmatrix}\quad\begin{pmatrix}3\\5\\9\end{pmatrix}\qquad\begin{pmatrix}4\\7\\16\end{pmatrix}\in \mathit{aff}\left(\left\{\begin{pmatrix}1\\1\\1\end{pmatrix},\begin{pmatrix}2\\3\\4\end{pmatrix},\begin{pmatrix}3\\5\\9\end{pmatrix}\right\}\right)$$

$$\begin{pmatrix}2\\3\\4\end{pmatrix}\quad\begin{pmatrix}3\\5\\9\end{pmatrix}\quad\begin{pmatrix}4\\7\\16\end{pmatrix}$$

Correctness

Theorem:

a) Algorithm terminates after at most $nk + n$ iterations of the loop, where $n = |N|$ and k is the number of variables.

b) For all $v \in N$, we have $aff(G_{fin}[v]) = V^{\#}[v]$.

Invariants for b)

I1: $\forall v \in N : G[v] \subseteq V[v]$ and $\forall (u, x) \in W : x \in V[u]$.

I2: $\forall$(u,s,v)$\in$ E: $aff\left(G[v] \cup \left\{[\![s]\!]x \mid (u, x) \in W\right\}\right) \sqsupseteq f_s(aff(G[u]))$.

Complexity

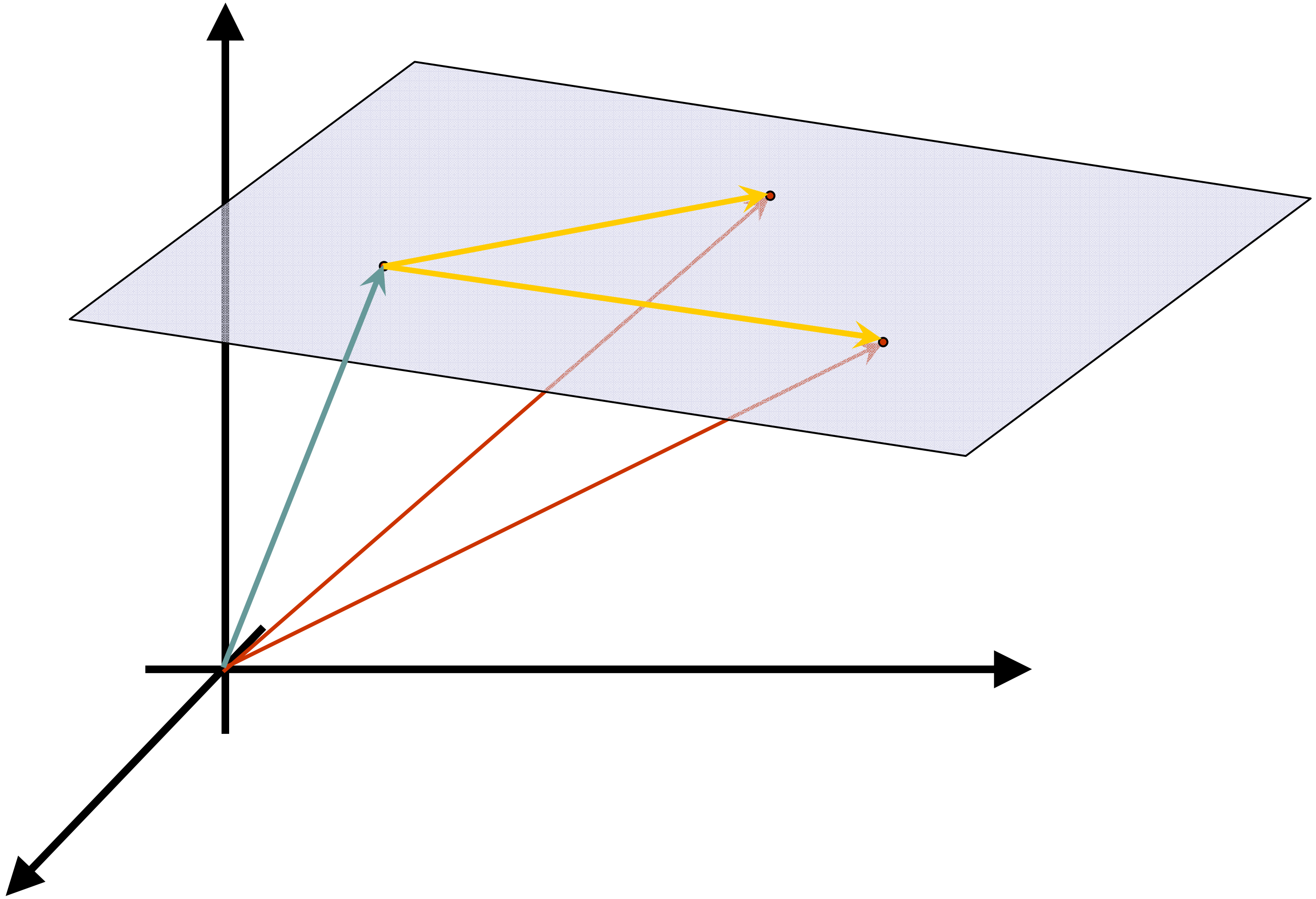
Theorem:

a) The affine hulls $V^{\#}[u] = \mathit{aff}(V[u])$ can be computed in time $O(n \cdot k^3)$, where $n = |N| + |E|$.

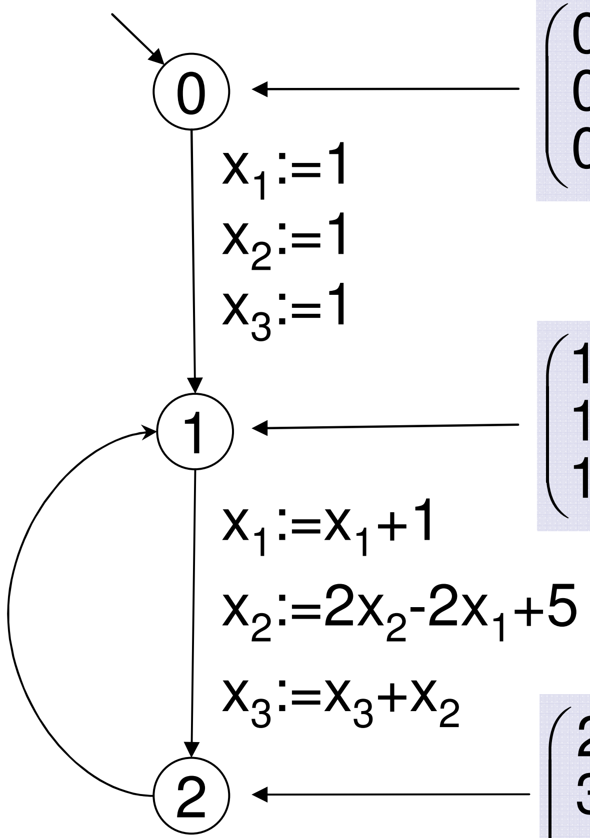
b) In this computation only arithmetic operations on numbers with $O(n \cdot k^2)$ bits are used.

Store diagonal basis for membership tests.
Propagate original vectors.

Point + Linear Basis



Example



$$\begin{pmatrix}0\\0\\0\end{pmatrix},\begin{pmatrix}1\\0\\0\end{pmatrix},\begin{pmatrix}0\\1\\0\end{pmatrix},\begin{pmatrix}0\\0\\1\end{pmatrix}$$

$$\begin{pmatrix}1\\1\\1\end{pmatrix}\quad\begin{pmatrix}2\\3\\4\end{pmatrix}\quad\begin{pmatrix}3\\5\\9\end{pmatrix}\quad\begin{pmatrix}4\\7\\16\end{pmatrix}\qquad\cancel{\begin{pmatrix}1\\2\\3\end{pmatrix}\quad\begin{pmatrix}2\\4\\8\end{pmatrix}}\quad\begin{pmatrix}1\\2\\0\end{pmatrix},\begin{pmatrix}0\\0\\2\end{pmatrix}\quad\begin{pmatrix}0\\0\\0\end{pmatrix}$$

$$\begin{pmatrix}2\\3\\4\end{pmatrix}\quad\begin{pmatrix}3\\5\\9\end{pmatrix}\quad\begin{pmatrix}4\\7\\16\end{pmatrix}\qquad\cancel{\begin{pmatrix}1\\2\\5\end{pmatrix}\quad\begin{pmatrix}2\\4\\12\end{pmatrix}}\quad\begin{pmatrix}1\\2\\0\end{pmatrix},\begin{pmatrix}0\\0\\2\end{pmatrix}$$

Determining Affine Relations

Lemma: a is valid for $X \iff a$ is valid for $\mathit{aff}(X)$.

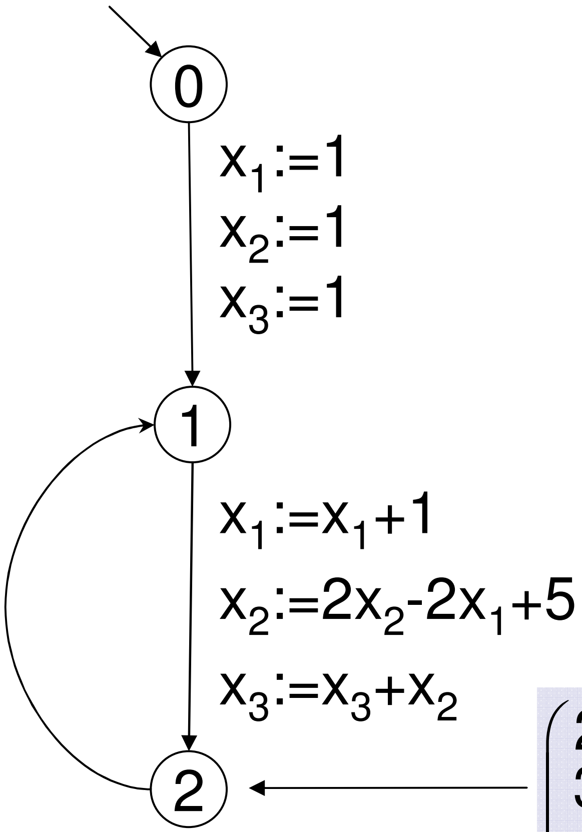
$\Rightarrow$ suffices to determine the affine relations valid for affine bases;
can be done with a linear equation system!

Theorem:

a) The vector spaces of all affine relations valid at the program points of an affine program can be computed in time $O(n \cdot k^3)$.

b) This computation performs arithmetic operations on integers with $O(n \cdot k^2)$ bits only.

Example



$$\begin{pmatrix}2\\3\\4\end{pmatrix}\ \begin{pmatrix}3\\5\\9\end{pmatrix}\ \begin{pmatrix}4\\7\\16\end{pmatrix} \qquad \begin{pmatrix}1\\2\\0\end{pmatrix},\begin{pmatrix}0\\0\\2\end{pmatrix}$$

$a_0 + a_1x_1 + a_2x_2 + a_3x_3 = 0$ is valid at 2

$$\Leftrightarrow \quad \begin{aligned} a_0 + 2a_1 + 3a_2 + 4a_3 &= 0 \\ 1a_1 + 2a_2 &= 0 \\ 2a_3 &= 0 \end{aligned}$$

$$\Leftrightarrow \quad a_0 = a_2,\ a_1 = -2a_2,\ a_3 = 0$$

$$\Rightarrow \quad 2x_1 - x_2 - 1 \text{ is valid at } 2$$

Also in the Paper

- Non-deterministic assignments
- Bit length estimation
- Polynomial relations
- Affine programs + affine equality guards
  - validity of affine relations undecidable

End of Excursion 1

(Optimal) Program Analysis of Sequential and Parallel Programs

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster, Germany

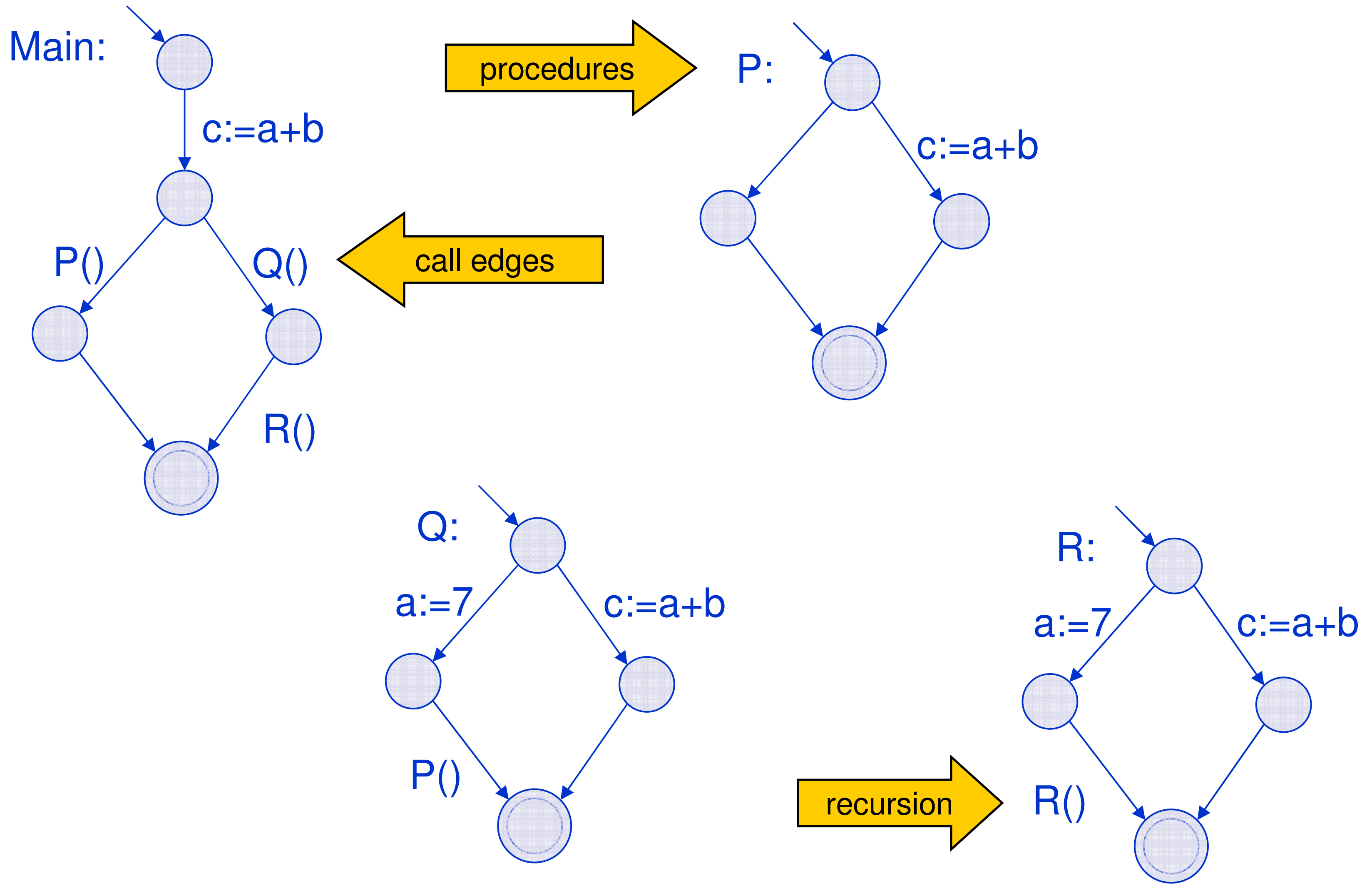
3rd Summer School on
Verification Technology, Systems, and Applications

Luxemburg, September 6-10, 2010

Overview

- Introduction
- Fundamentals of Program Analysis
  - Excursion 1
- **Interprocedural Analysis**
  - Excursion 2
- Analysis of Parallel Programs
  - Excursion 3
  - Appendix
- Conclusion

Interprocedural Analysis

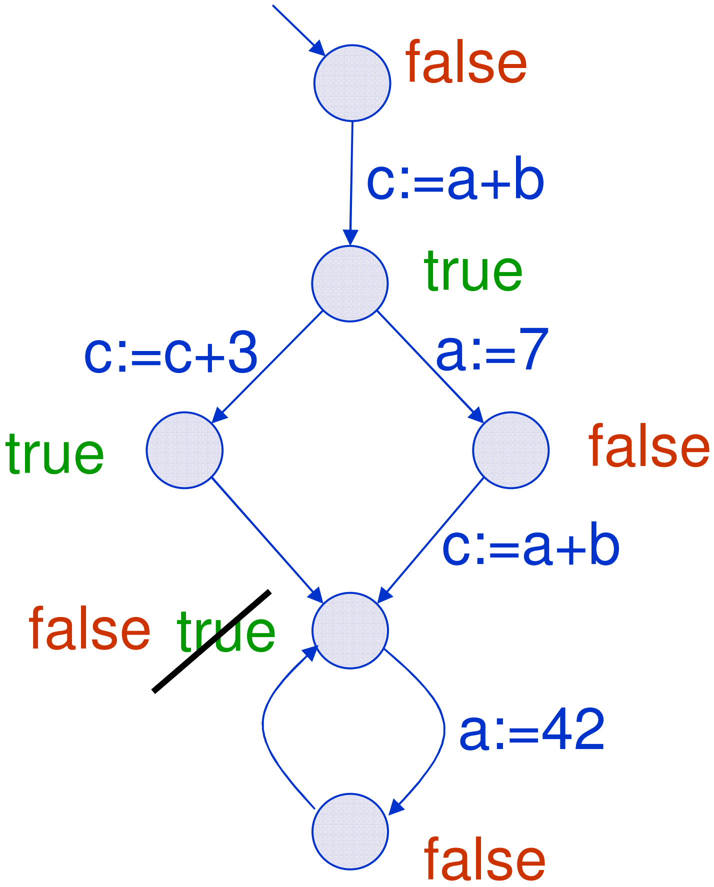


Running Example:
(Definite) Availability of the single expression a+b

The lattice:

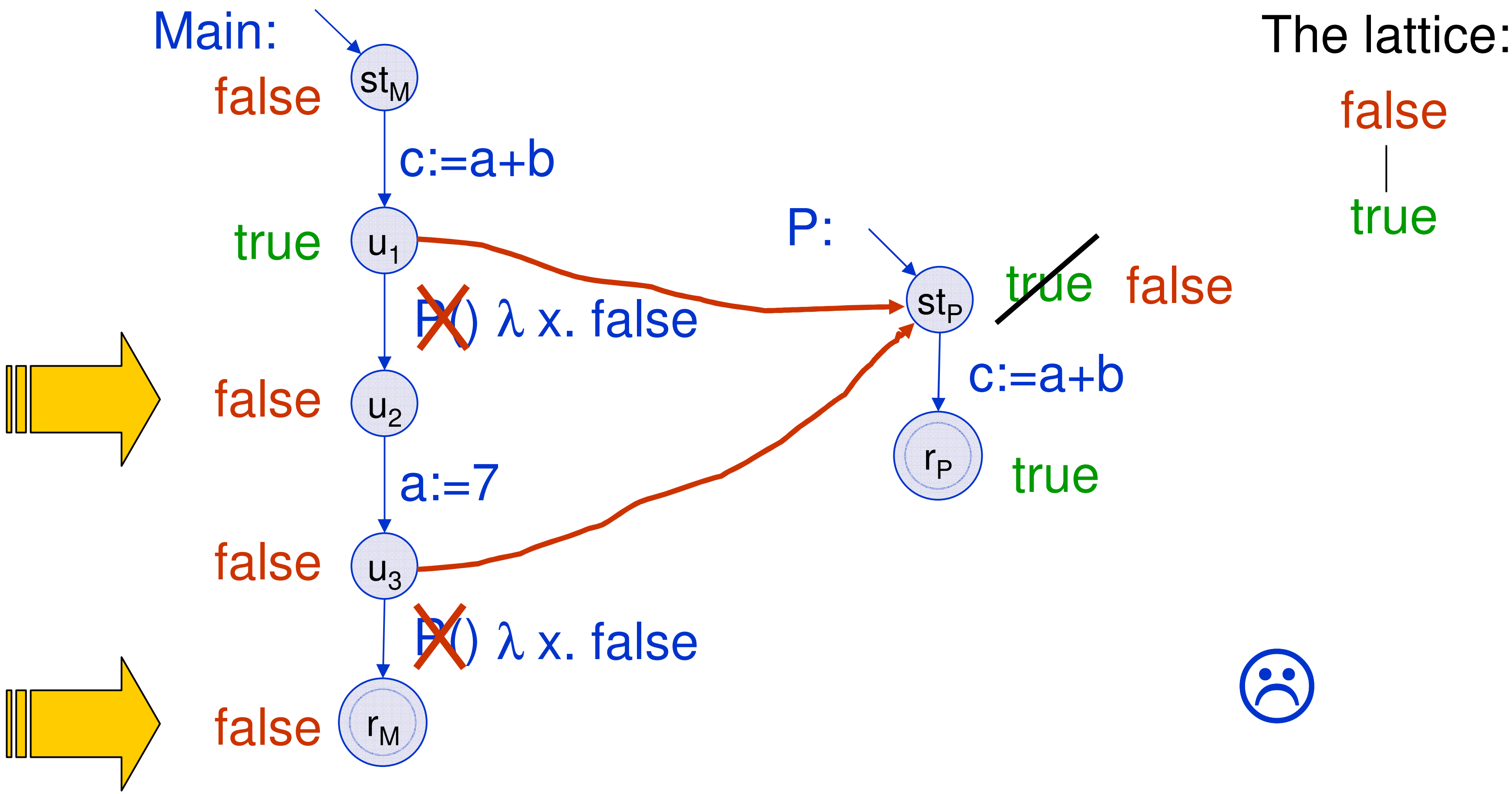
false	a+b not available
true	a+b available

Initial value: false



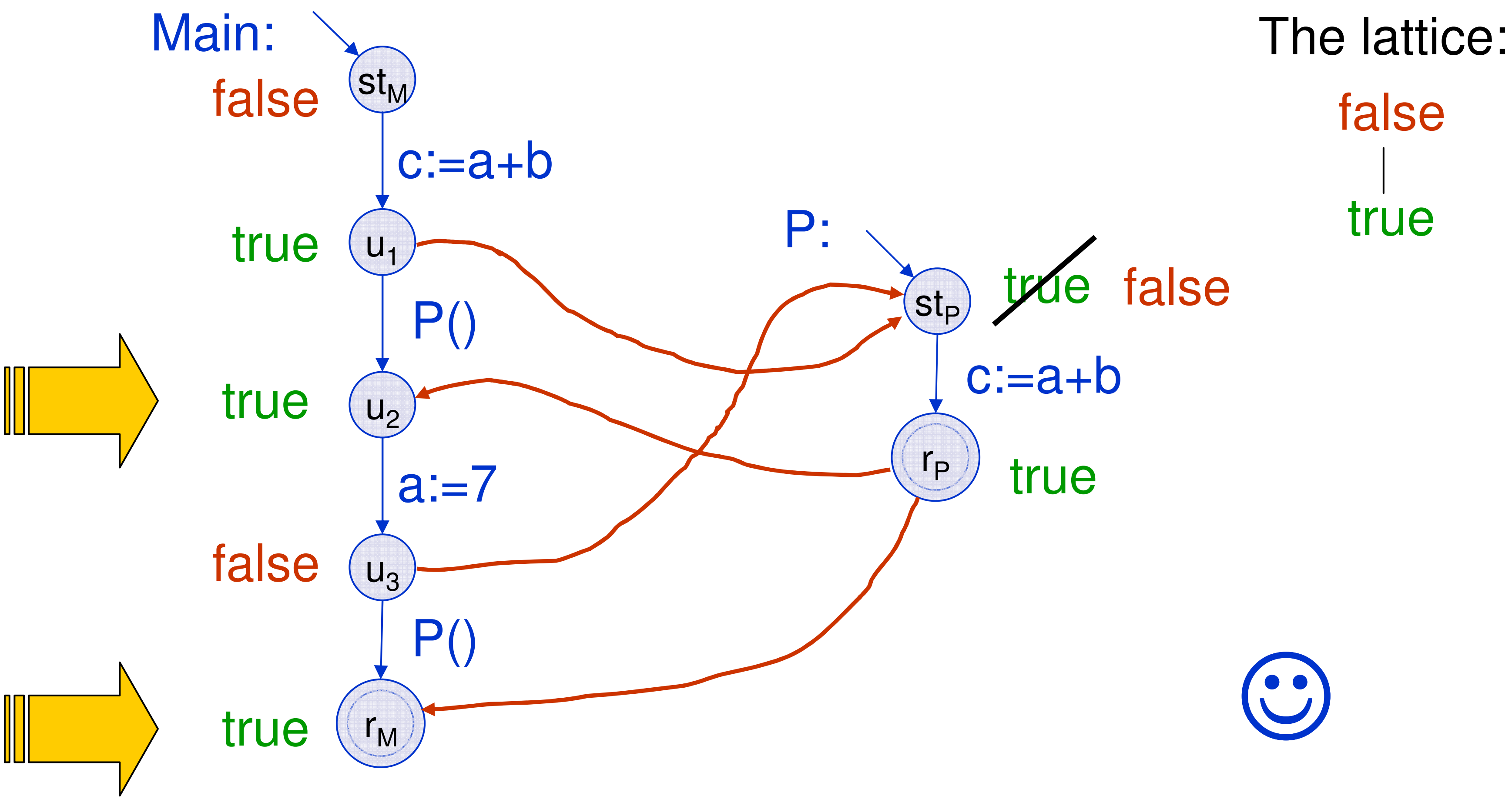
Intra-Procedural-Like Analysis

Conservative assumption: procedure destroys all information; information flows from call node to entry point of procedure



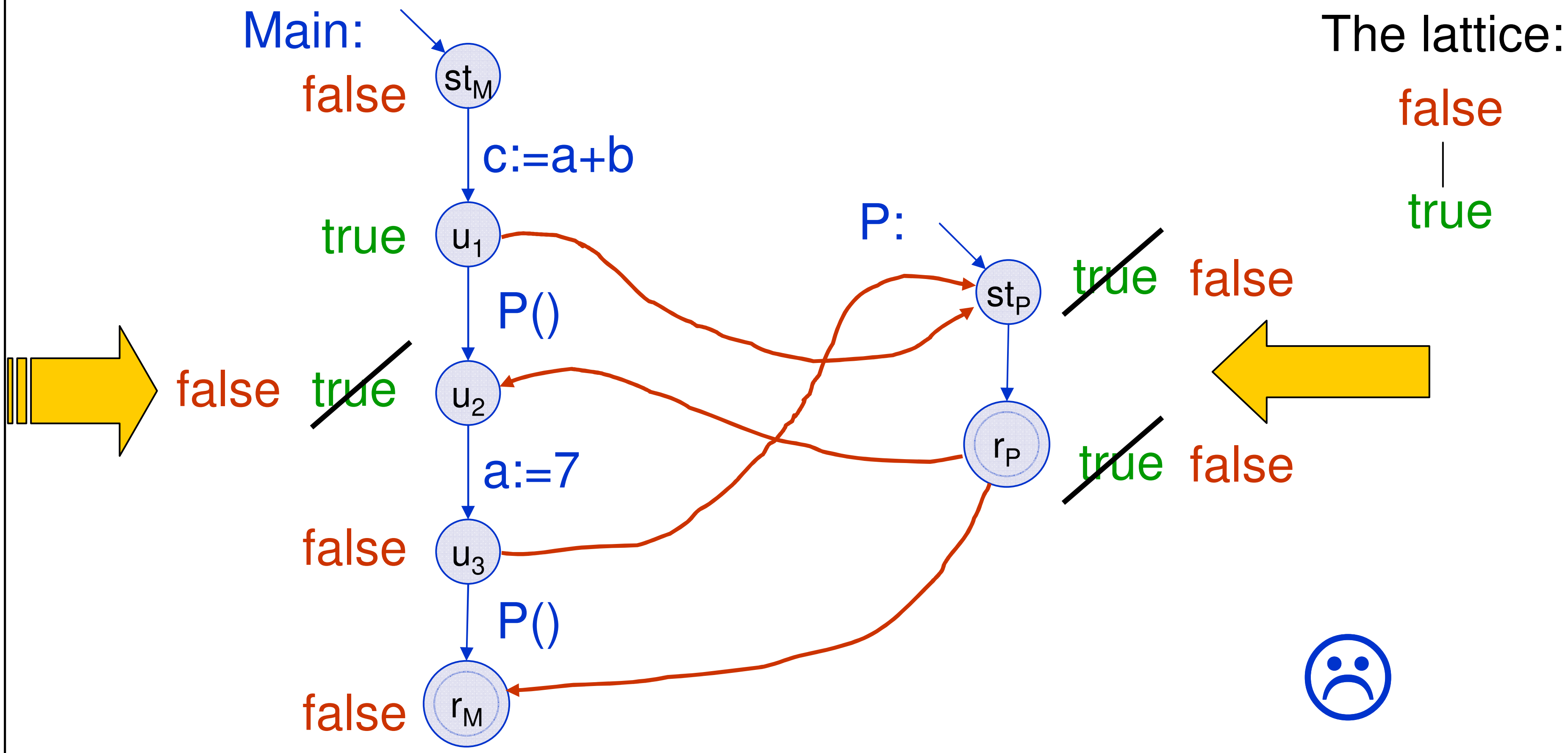
Context-Insensitive Analysis

Conservative assumption: Information flows from each call node to entry of procedure and from exit of procedure back to return point



Context-Insensitive Analysis

Conservative assumption: Information flows from each call node to entry of procedure and from exit of procedure bac to return point



Recall: Abstract Interpretation Recipe

constraint system for Reference Semantics on concrete lattice $(D,\sqsubseteq)$	Replace concrete operators o by abstract operators $o^\#$	constraint system for Analysis on abstract lattice $(D^\#,\sqsubseteq^\#)$
MFP		$MFP^\#$

Assume a universally-disjunctive abstraction function $\alpha : D \to D^\#$.

Correct abstract interpretation:

Show $\alpha(o(x_1,\dots,x_k)) \sqsubseteq^\# o^\#(\alpha(x_1),\dots,\alpha(x_k))$ f.a. $x_1,\dots,x_k \in L$, operators o

Then $\alpha(MFP[u]) \sqsubseteq^\# MFP^\#[u]$ f.a. u

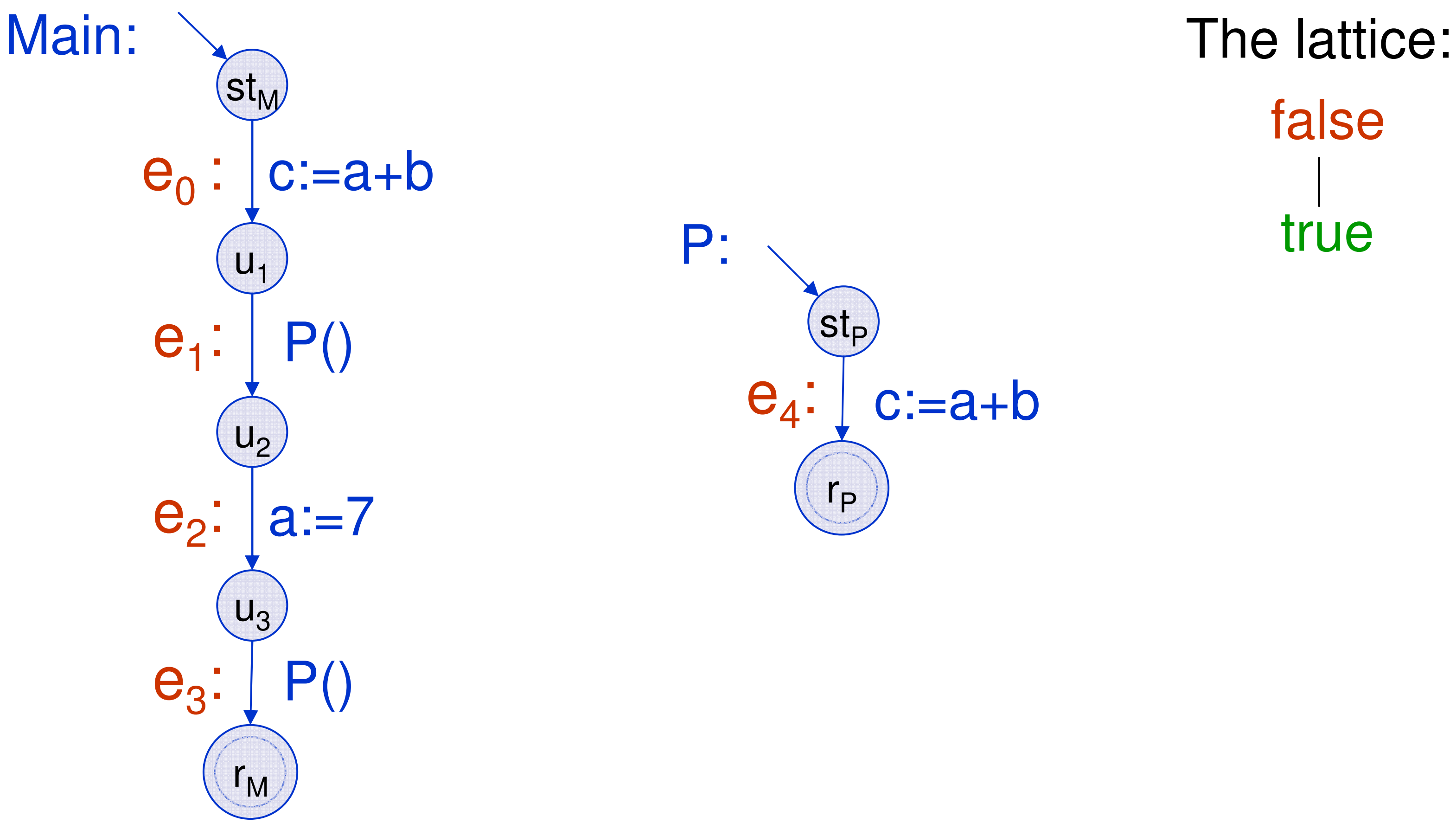
Correct and precise abstract interpretation:

Show $\alpha(o(x_1,\dots,x_k)) = o^\#(\alpha(x_1),\dots,\alpha(x_k))$ f.a. $x_1,\dots,x_k \in L$, operators o

Then $\alpha(MFP[u]) = MFP^\#[u]$ f.a. u

Use this as a guideline for designing correct (and precise) analyses !

Example Flow Graph



Let's Apply Our Abstract Interpretation Recipe: Constraint System for Feasible Paths

Operational justification:

$$\underline{S}(u) \;=\; \{\, r \in \text{Edges}^* \mid st_p \xrightarrow{r} u \,\} \quad \text{for all } u \text{ in procedure } p$$

$$\underline{S}(p) \;=\; \{\, r \in \text{Edges}^* \mid st_p \xrightarrow{r} \varepsilon \,\} \quad \text{for all procedures } p$$

$$\underline{R}(u) \;=\; \{\, r \in \text{Edges}^* \mid \exists w \in \text{Nodes}^* : st_{Main} \xrightarrow{r} uw \,\} \quad \text{for all } u$$

Same-level runs:

$S(p) \supseteq S(r_p)$ — r_p return point of p

$S(st_p) \supseteq \{\varepsilon\}$ — st_p entry point of p

$S(v) \supseteq S(u) \cdot \{\langle e \rangle\}$ — $e = (u,s,v)$ base edge

$\mathrm{S}(\mathrm{v}) \supseteq S(u) \cdot S(p)$ — $e = (u,p,v)$ call edge

Reaching runs:

$R(st_{Main}) \supseteq \{\varepsilon\}$ — st_{Main} entry point of *Main*

$R(v) \supseteq R(u) \cdot \{\langle e \rangle\}$ — $e = (u,s,v)$ basic edge

$R(v) \supseteq R(u) \cdot S(p)$ — $e = (u,p,v)$ call edge

$R(st_p) \supseteq R(u)$ — $e = (u,p,v)$ call edge, st_p entry point of p

Context-Sensitive Analysis

Idea:

Phase 1: Compute summary information for each procedure...
... as an abstraction of same-level runs

Phase 2: Use summary information as transfer functions for procedure calls...
... in an abstraction of reaching runs

Classic approaches for summary informations:

1) Functional approach: [Sharir/Pnueli 81, Knoop/Steffen: CC´92]
Use (monotonic) functions on data flow informations !

2) Relational approach: [Cousot/Cousot: POPL´77]
Use relations (of a representable class) on data flow informations !

3) Call string approach: [Sharir/Pnueli 81], [Khedker/Karkare: CC´08]
Analyse relative to finite portion of call stack !

Formalization of Functional Approach

Abstractions:

Abstract same-level runs with $\alpha_{Funct}: \text{Edges}^* \to (L \to L)$:

$$\alpha_{Funct}(R) \;=\; \bigsqcup \{\, f_r \mid r \in R \,\} \qquad \text{for } R \subseteq \text{Edges}^*$$

Abstract reaching runs with $\alpha_{MOP}: \text{Edges}^* \to L$:

$$\alpha_{MOP}(R) \;=\; \bigsqcup \{\, f_r(init) \mid r \in R \,\} \qquad \text{for } R \subseteq \text{Edges}^*$$

1. Phase: Compute summary informations, i.e., functions:

$S^\#(p)$	$\sqsupseteq$	$S^\#(r_p)$	r_p return point of p
$S^\#(st_p)$	$\sqsupseteq$	id	st_p entry point of p
$S^\#(v)$	$\sqsupseteq$	$f_e^\# \circ S^\#(u)$	$e = (u,s,v)$ base edge
$S^\#(v)$	$\sqsupseteq$	$S^\#(p) \circ S^\#(u)$	$e = (u,p,v)$ call edge

2. Phase: Use summary informations; compute on data flow informations:

$R^\#(st_{Main})$	$\sqsupseteq$	$init$	st_{Main} entry point of $Main$
$R^\#(v)$	$\sqsupseteq$	$f_e^\#(R^\#(u))$	$e = (u,s,v)$ basic edge
$R^\#(v)$	$\sqsupseteq$	$S^\#(p)(R^\#(u))$	$e = (u,p,v)$ call edge
$R^\#(st_p)$	$\sqsupseteq$	$R^\#(u)$	$e = (u,p,v)$ call edge, st_p entry point of p

Functional Approach

Theorem:

Correctness: For any monotone framework:

$$\alpha_{MOP}(\underline{R}[u]) \sqsubseteq \underline{R}^{\#}[u] \quad \text{f.a. } u$$

Completeness: For any universally-distributive framework:

$$\alpha_{MOP}(\underline{R}[u]) = \underline{R}^{\#}[u] \quad \text{f.a. } u$$

Alternative condition:
framework positively-distributive & all prog. point dyn. reachable

Remark:

a) Functional approach is effective, if L is finite...

b) ... but may lead to chains of length up to $|L| \cdot \text{height}(L)$ at each program point (in general).

Functional Approach for Availability of Single Expression Problem

Observations:

Just three montone functions on lattice L:

$$\begin{array}{c} \textbf{false} \\ | \\ \textbf{true} \end{array}$$

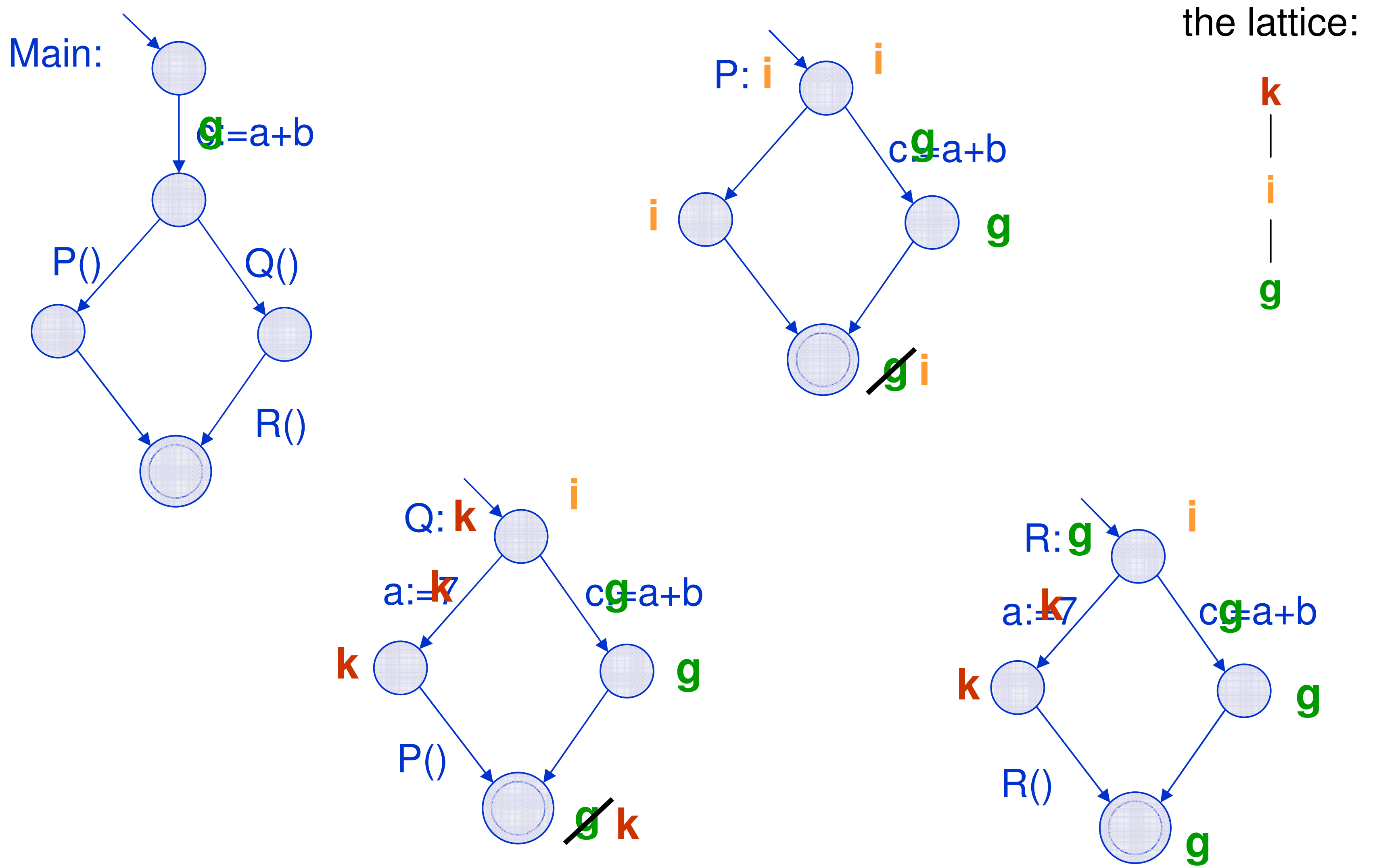
$$\begin{array}{cc} \lambda\, x\, .\, \textbf{false} & \textbf{k (ill)} \\ | & | \\ \lambda\, x\, .\, x & \textbf{i (gnore)} \\ | & | \\ \lambda\, x\, .\, \textbf{true} & \textbf{g (enerate)} \end{array}$$

Functional composition of two such functions $f,g : L \rightarrow L$:

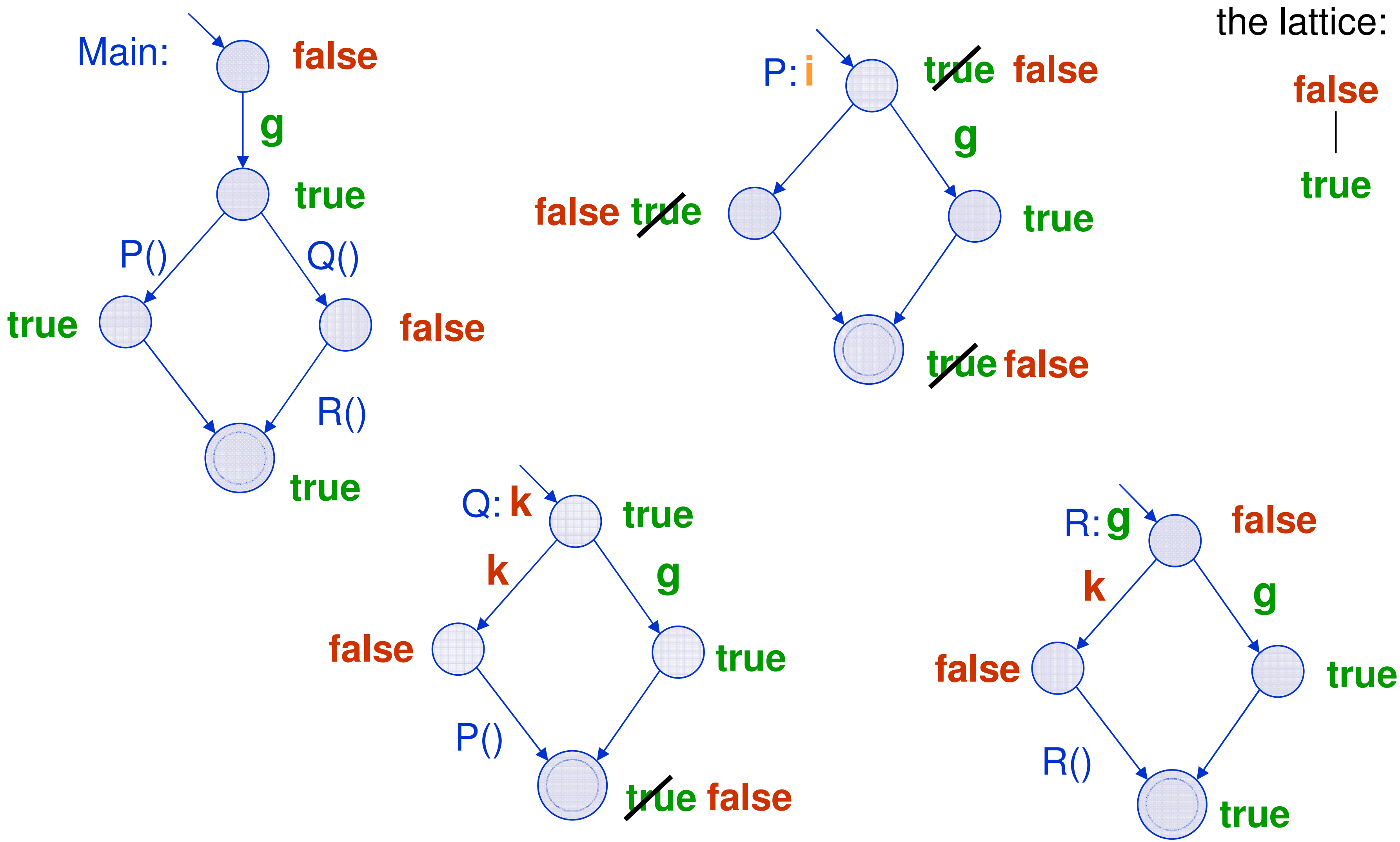
$$h \circ f = \begin{cases} f & \text{if } h = \mathrm{i} \\ h & \text{if } h \in \{\mathrm{g},\mathrm{k}\} \end{cases}$$

Analogous: precise interprocedural analysis for all (separable) bitvector problems in time linear in program size.

Context-Sensitive Analysis, 1. Phase



Context-Sensitive Analysis, 2. Phase



Functional Approach

Theorem:

Correctness: For any monotone framework:

$\alpha_{MOP}(\underline{R}[u]) \sqsubseteq \underline{R}^{\#}[u]$ f.a. u

Completeness: For any universally-distributive framework:

$\alpha_{MOP}(\underline{R}[u]) = \underline{R}^{\#}[u]$ f.a. u

Alternative condition:
framework positively-distributive & all prog. point dyn. reachable

Remark:

a) Functional approach is effective, **if *L* is finite ...**

b) ... but may lead to chains of length up to $|L| \cdot$ height(L) at each program point.

Overview

- Introduction
- Fundamentals of Program Analysis
  - Excursion 1
- Interprocedural Analysis
  - **Excursion 2**
- Analysis of Parallel Programs
  - Excursion 3
  - Appendix
- Conclusion

Precise Interprocedural Analysis through Linear Algebra

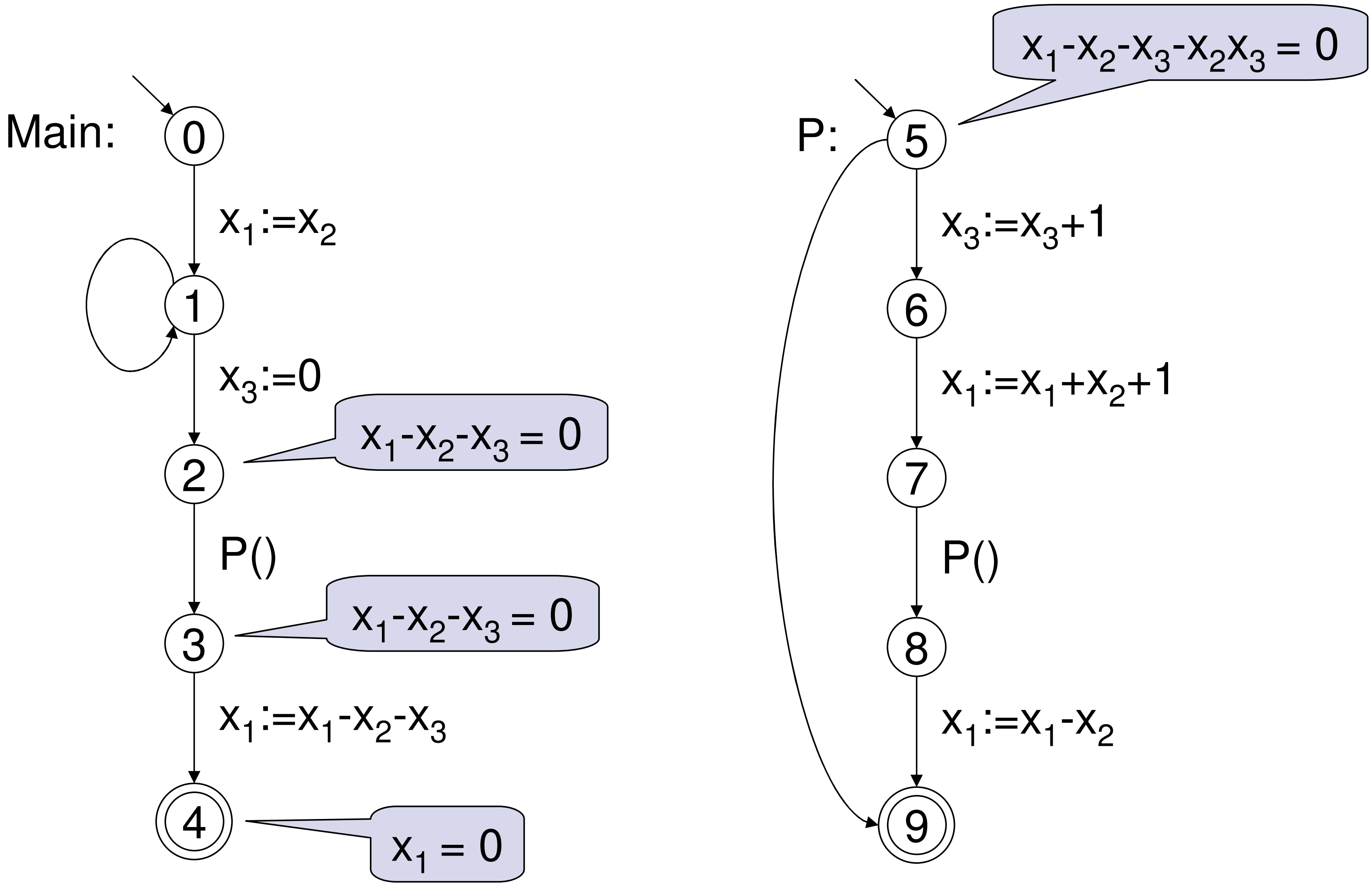
Markus Müller-Olm
FernUniversität Hagen
(on leave from Universität Dortmund)

Joint work with

Helmut Seidl (TU München)

POPL 2004, Venice, January 14-16, 2004

Finding Invariants...



… through Linear Algebra

- Linear Algebra
  - vectors
  - vector spaces, sub-spaces, bases
  - linear maps, matrices
  - vector spaces of matrices
  - Gaussian elimination
  - ...

Applications

- definite equalities: $x = y$
- constant propagation: $x = 42$
- discovery of symbolic constants: $x = 5yz+17$
- complex common subexpressions: $xy+42 = y^2+5$
- loop induction variables
- program verification
- ...

A Program Abstraction

Affine programs:

- affine assignments: $x_1 := x_1 - 2x_3 + 7$
- unknown assignments: $x_i := ?$
  $\rightarrow$ abstract too complex statements!
- non-deterministic instead of guarded branching

The Challenge

Given an affine program

(with procedures, parameters, local and global variables, ...)

over R :

(R the field $\mathbb{Q}$ or $\mathbb{Z}_p$, a modular ring $\mathbb{Z}_{m,}$ the ring of integers $\mathbb{Z}$, an effective PIR,...)

- determine all valid affine relations:

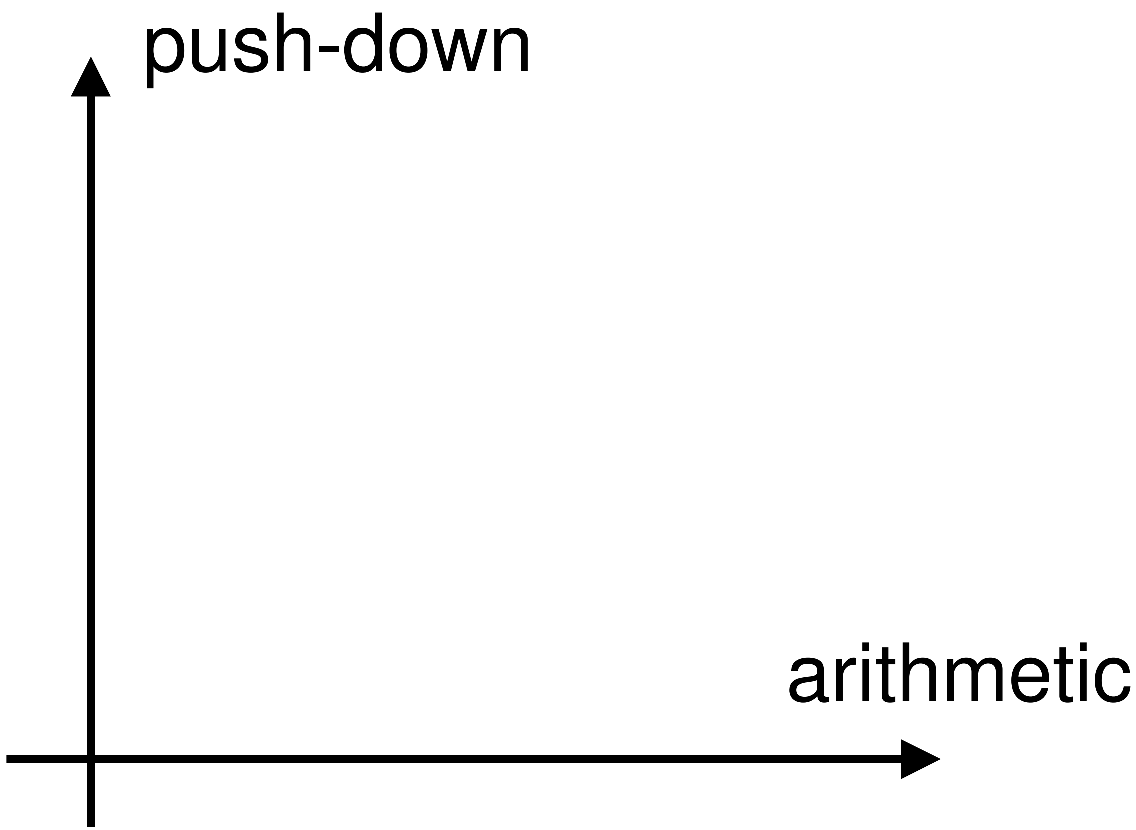
  $a_0 + \sum a_i x_i = 0$ $\quad a_i \in R$ $\quad 5x+7y-42=0$

- determine all valid polynomial relations (of degree $\leq$ d):

  $p(x_1,\ldots,x_k) = 0$ $\quad p \in R[x_1,\ldots,x_n]$ $\quad 5xy^2+7z^3-42=0$

… and all this in polynomial time (unit cost measure) !!!

Infinity Dimensions



Use a Standard Approach for Interprocedural Generalization of Karr ?

Functional approach [Sharir/Pnueli, 1981], [Knoop/Steffen, 1992]

- Idea: summarize each procedure by function on data flow facts
- Problem: not applicable

Call-string approach [Sharir/Pnueli, 1981] , [Khedker/Karkare: CC´08]

- Idea: take just a finite piece of run-time stack into account
- Problem: not exact

Relational approach [Cousot/Cousot, 1977]

- Idea: summarize each procedure by approximation of I/O relation
- Problem: not exact

Towards the Algorithm ...

Concrete Semantics of an Execution Path

- Every execution path π induces an **affine transformation** of the program state:

$$\llbracket x_1 := x_1 + x_2 + 1;\; x_3 := x_3 + 1 \rrbracket (v)$$

$$= \llbracket x_3 := x_3 + 1 \rrbracket \left(\llbracket x_1 := x_1 + x_2 + 1 \rrbracket (v) \right)$$

$$= \llbracket x_3 := x_3 + 1 \rrbracket \left(\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right)$$

$$= \left(\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \right)$$

Affine Relations

- An affine relation can be viewed as a vector:

$$x_1 - 3x_2 + 5 = 0 \quad \text{corresponds to} \quad a = \begin{pmatrix} 5 \\ 1 \\ 3 \\ 0 \end{pmatrix}$$

Affine Assignments induce linear wp- Transformations on Affine Relations

$$\{x_2 + x_3 + 5 = 0\} \quad x_1 := 4x_2 + x_3 + 3 \quad \{x_1 - 3x_2 + 2 = 0\}$$

weakest precondition!

A linear transformation:

$$\begin{pmatrix} 1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 4 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ -3 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

WP of Affine Relations

- Every execution path π induces a linear transformation of affine post-conditions into their weakest pre-conditions:

$$
\begin{aligned}
& [\![x_1 := x_1 + x_2 + 1;\ x_3 := x_3 + 1]\!]^{\mathsf{T}}(a) \\
= \ & [\![x_1 := x_1 + x_2 + 1]\!]^{\mathsf{T}}\left([\![x_3 := x_3 + 1]\!]^{\mathsf{T}}(a) \right) \\
= \ & [\![x_1 := x_1 + x_2 + 1]\!]^{\mathsf{T}}\left(\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \right) \\
= \ & \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}
\end{aligned}
$$

Observations

- Only the zero relation is valid at program start:

$$\mathbf{0} : \; 0+0x_1+\ldots+0x_k = 0$$

- Thus, relation $a_0+a_1x_1+\ldots+a_kx_k=0$ is valid at program point v

iff

$$M\,a = \mathbf{0} \quad \text{for all } M \in \{[\![\pi]\!]^T \mid \pi \text{ reaches } v\}$$

iff

$$M\,a = \mathbf{0} \quad \text{for all } M \in \mathit{Span}\,\{[\![\pi]\!]^T \mid \pi \text{ reaches } v\}$$

iff

$$M\,a = \mathbf{0} \quad \text{for all } M \text{ in a basis of } \mathit{Span}\,\{[\![\pi]\!]^T \mid \pi \text{ reaches } v\}$$

- Matrices M form a vector space of dimension (k+1) x (k+1)

- Sub-spaces form a complete lattice of height $O(k^2)$.

Let's Apply Our Abstract Interpretation Recipe: Constraint System for Feasible Paths

Operational justification:

$$\underline{S}(u) = \{ r \in \text{Edges}^* \mid st_p \xrightarrow{r} u \} \quad \text{for all } u \text{ in procedure } p$$
$$\underline{S}(p) = \{ r \in \text{Edges}^* \mid st_p \xrightarrow{r} \varepsilon \} \quad \text{for all procedures } p$$

$$\underline{R}(u) = \{ r \in \text{Edges}^* \mid \exists \omega \in \text{Nodes}^* : st_{Main} \xrightarrow{r} u\omega \} \quad \text{for all } u$$

Same-level runs:

$S(p) \supseteq S(r_p)$ — r_p return point of p

$S(st_p) \supseteq \{\varepsilon\}$ — st_p entry point of p

$S(v) \supseteq S(u) \cdot \{\langle e \rangle\}$ — $e = (u, s, v)$ base edge

$S(v) \supseteq S(u) \cdot S(p)$ — $e = (u, p, v)$ call edge

Reaching runs:

$R(st_{Main}) \supseteq \{\varepsilon\}$ — st_{Main} entry point of *Main*

$R(v) \supseteq R(u) \cdot \{\langle e \rangle\}$ — $e = (u, s, v)$ basic edge

$R(v) \supseteq R(u) \cdot S(p)$ — $e = (u, p, v)$ call edge

$R(st_p) \supseteq R(u)$ — $e = (u, p, v)$ call edge, st_p entry point of p

Algorithm for Computing Affine Relations

1) Compute a basis B with:

$$\text{Span } B = \text{Span } \{[\![\pi]\!]^T \mid \pi \text{ reaches } v\}$$

for each program point by a precise abstract interpretation:

Lattice: Subspaces of $\mathbb{F}^{(k+1) \times (k+1)}$

Replace:

$\{\varepsilon\}$	*by*	$\{I\}$	(I identity matrix)
concatenation	*by*	matrix product	(lifted to subspaces)
$\{\langle e \rangle\}$	*by*	$\langle A_e \rangle$ for affine assignment edge $e = (u,s,v)$	

2) Solve the linear equation system:

$$M\, a = \mathbf{0} \quad \text{for all } M \in B$$

Theorem

In an affine program:

- The following vector spaces of matrices can be computed precisely:

  $\alpha(R(v)) = \mathrm{Span}\ \{ [\![\pi]\!]^T \mid \pi \in R(v) \}$ for each prg. point v.

- The vector spaces

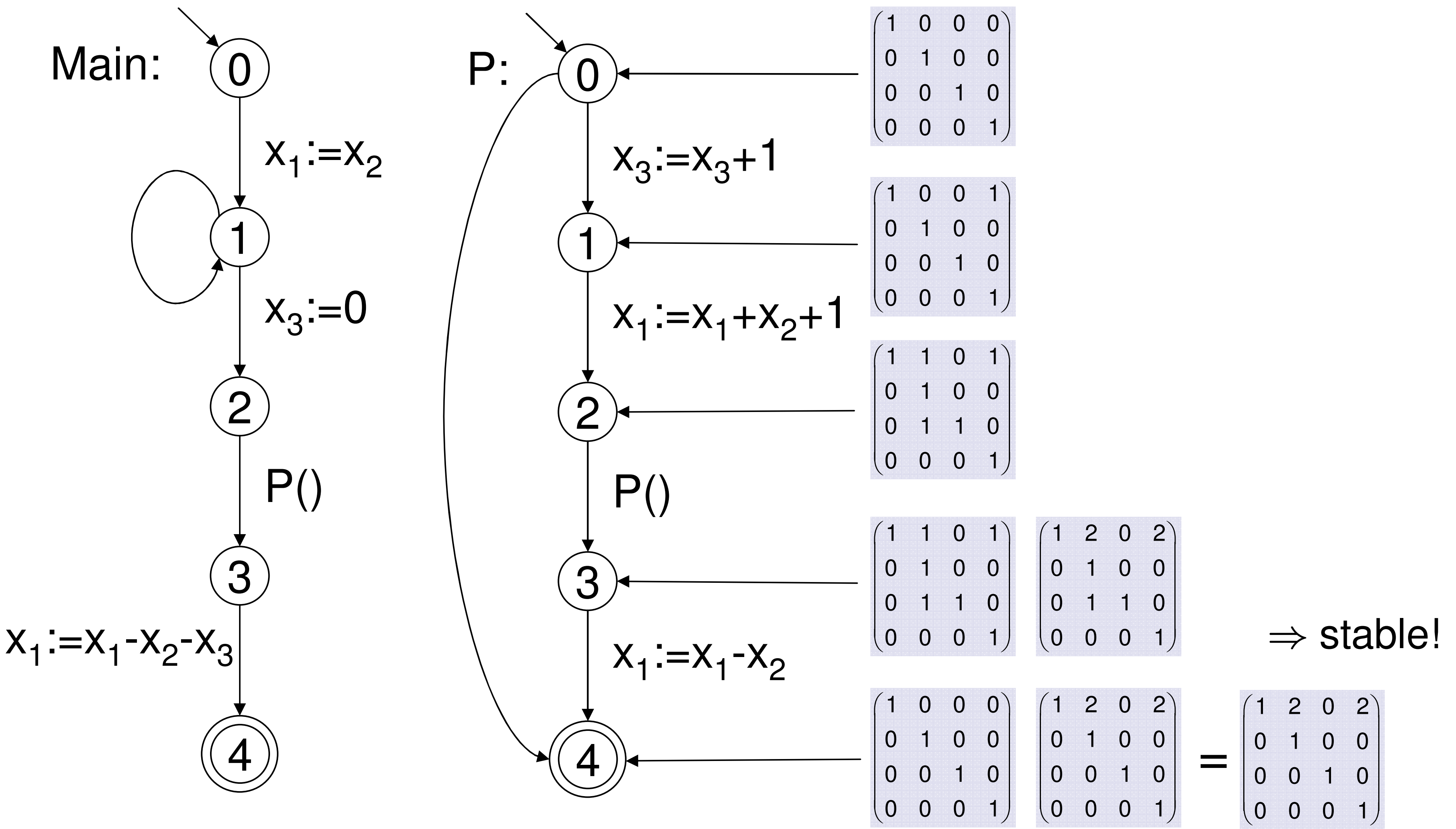
  $\{ a \in \mathbb{F}^{k+1} \mid$ affine relation a is valid at $v \}$

  can be computed precisely for all prg. points v.

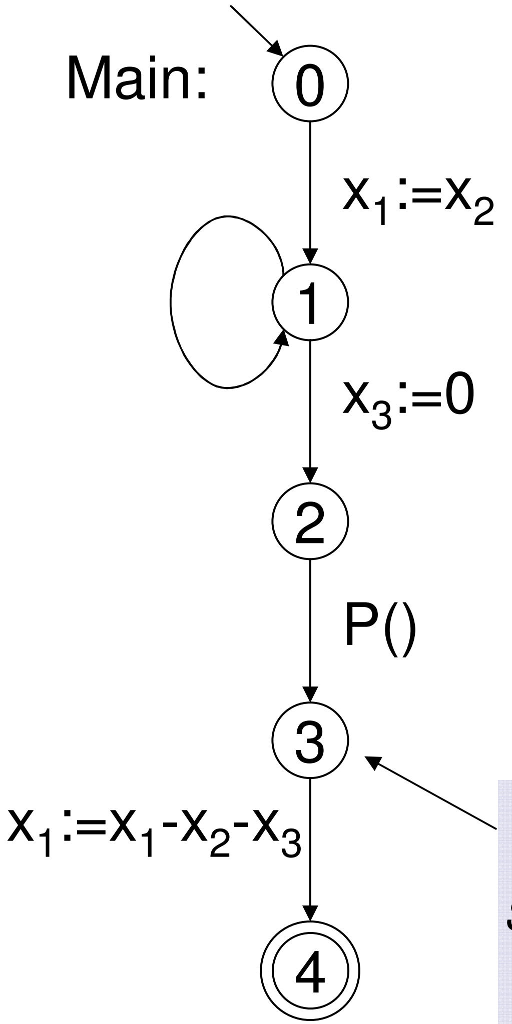
- The time complexity is linear in the program size and polynomial in the number of variables: $\mathcal{O}(n \cdot k^8)$

  (n size of the program, k number of variables)

An Example


Main:
0
$x_1:=x_2$
1
$x_3:=0$
2
P()
3
$x_1:=x_1-x_2-x_3$
4
P:
0
$x_3:=x_3+1$
1
$x_1:=x_1+x_2+1$
2
P()
3
$x_1:=x_1-x_2$
4
$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$
$\begin{pmatrix}1&0&0&1\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$
$\begin{pmatrix}1&1&0&1\\0&1&0&0\\0&1&1&0\\0&0&0&1\end{pmatrix}$
$\begin{pmatrix}1&1&0&1\\0&1&0&0\\0&1&1&0\\0&0&0&1\end{pmatrix}$ $\begin{pmatrix}1&2&0&2\\0&1&0&0\\0&1&1&0\\0&0&0&1\end{pmatrix}$
$\Rightarrow$ stable!
$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$ $\begin{pmatrix}1&2&0&2\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$ = $\begin{pmatrix}1&2&0&2\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$

An Example



$a_0 + a_1x_1 + a_2x_2 + a_3x_3 = 0$ is valid at 3

$$\Leftrightarrow \begin{pmatrix}1&0&0&0\\0&1&1&0\\0&0&0&0\\0&0&0&0\end{pmatrix}\begin{pmatrix}a_0\\a_1\\a_2\\a_3\end{pmatrix} = 0 \text{ and } \begin{pmatrix}0&1&0&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}\begin{pmatrix}a_0\\a_1\\a_2\\a_3\end{pmatrix} = 0$$

$$\Leftrightarrow a_0 = 0 \wedge a_2 = a_3 = -a_1$$

$\Rightarrow$ Just the affine relations of the form

$$a_1x_1 - a_1x_2 - a_1x_3 = 0 \qquad (a_1 \in \mathbb{F})$$

are valid at 3 ☺

$$Span\left\{\begin{pmatrix}1&0&0&0\\0&1&1&0\\0&0&0&0\\0&0&0&0\end{pmatrix}, \begin{pmatrix}0&1&0&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}\right\}$$

Extensions

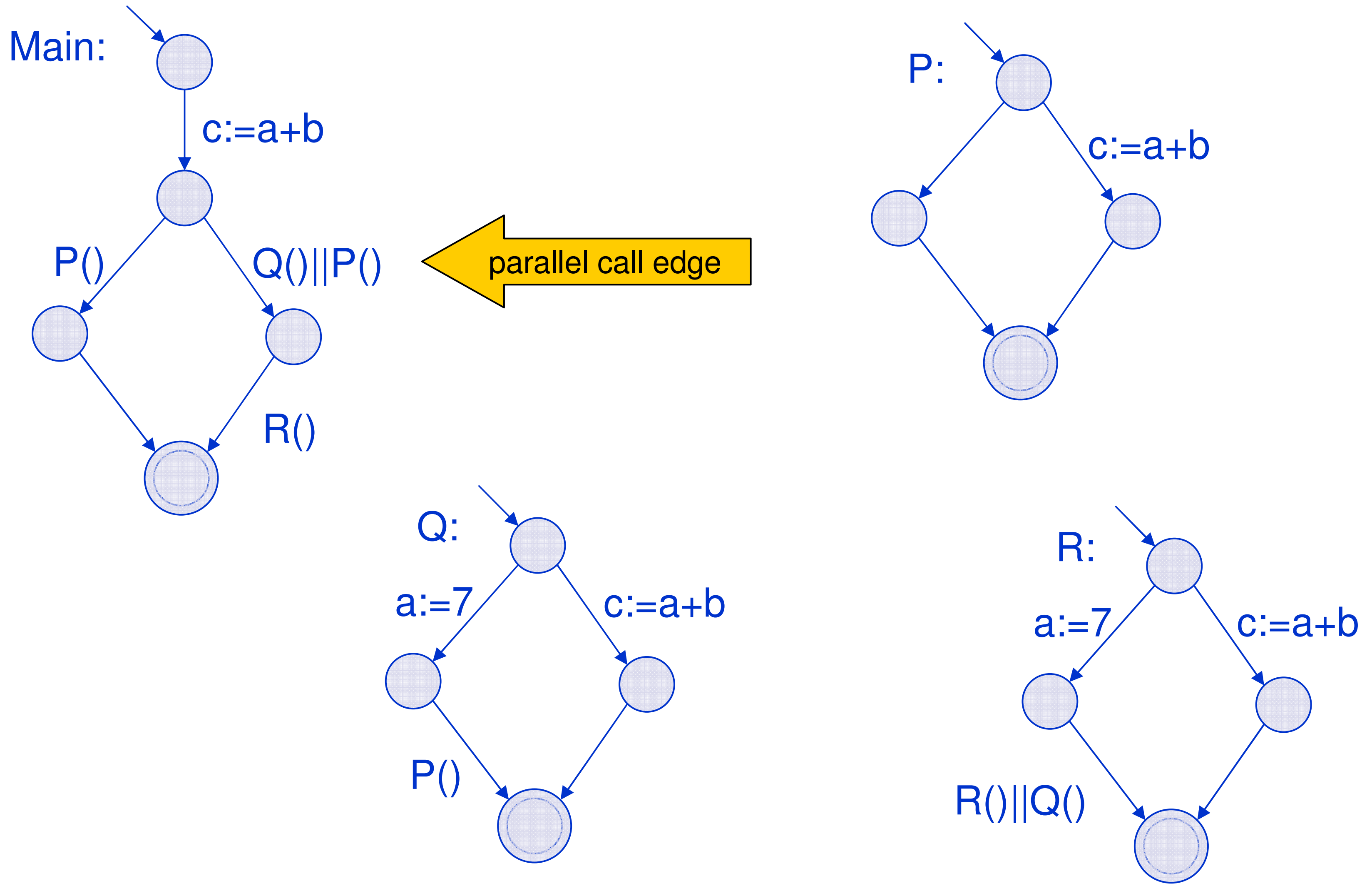
- Also in the paper:
  - Local variables, value parameters, return values
  - Computing polynomial relations of bounded degree
  - Affine pre-conditions
  - Formalization as an abstract interpretation
- In follow-up papers (see webpage):
  - Computing over modular rings (e.g. modulo 2^w) or PIRs
  - Forward algorithm

End of Excursion 2

Overview

- Introduction
- Fundamentals of Program Analysis
  - Excursion 1
- Interprocedural Analysis
  - Excursion 2
- **Analysis of Parallel Programs**
  - Excursion 3
  - Appendix
- Conclusion

Interprocedural Analysis of Parallel Programs



Interleaving- Operator $\otimes$ (Shuffle-Operator)

Example:

$$\langle a,b\rangle \otimes \langle x,y\rangle = \left\{ \begin{array}{l} \langle a,b,x,y\rangle \\ \langle a,x,b,y\rangle, \langle a,x,y,b\rangle \\ \langle x,a,b,y\rangle, \langle x,a,y,b\rangle, \langle x,y,a,b\rangle \end{array} \right\}$$

Constraint System for Same-Level Runs

Operational justification:

$$\underline{S}(u) = \{\, r \in \text{Edges}^* \mid st_p \xrightarrow{r} u \,\} \quad \text{for all } u \text{ in procedure } p$$
$$\underline{S}(p) = \{\, r \in \text{Edges}^* \mid st_p \xrightarrow{r} \varepsilon \,\} \quad \text{for all procedures } p$$

[Seidl/Steffen: ESOP 2000]

Same-level runs:

$S(p)$	$\supseteq$	$S(r_p)$	r_p return point of p
$S(st_p)$	$\supseteq$	$\{\varepsilon\}$	st_p entry point of p
$S(v)$	$\supseteq$	$S(u) \cdot \langle\{e\}\rangle$	$e = (u,s,v)$ base edge
$\mathrm{S(v)}$	$\supseteq$	$S(u) \cdot S(p)$	$e = (u,p,v)$ call edge
$\mathrm{S(v)}$	$\supseteq$	$S(u) \cdot (S(p_0) \otimes S(p_1))$	$e = (u, p_0 \mid\mid p_1, v)$ parallel call edge

Constraint System for a Variant of Reaching Runs

Operational justification:

$$\underline{R}(u,q) \;=\; \{\, r \in \text{Edges}^* \mid \exists c \in \text{Config} : st_q \xrightarrow{r} c,\ \text{At}_u(c) \,\}$$

for progam point u and procedure q

$$\underline{P}(q) \;=\; \{\, r \in \text{Edges}^* \mid \exists c \in \text{Config} : st_q \xrightarrow{r} c \,\}$$

Reaching runs:

[Seidl/Steffen: ESOP 2000]

$R(u,q) \supseteq S(u)$ — u program point in procedure q

$R(u,q) \supseteq S(v) \cdot R(u,p)$ — $e = (v, p, \_)$ call edge in proc. q

$R(u,q) \supseteq S(v) \cdot (R(u,p_i) \otimes P(p_{1-i}))$ — $e = (v, p_0 \mid\mid p_1, \_)$ parallel call edge in proc. q, $i = 0,1$

Interleaving potential:

$P(p) \supseteq R(u,p)$ — u program point and p procedure

Interleaving- Operator $\otimes$ (Shuffle-Operator)

Example:

$$\langle a,b\rangle \otimes \langle x,y\rangle = \left\{ \begin{array}{l} \langle a,b,x,y\rangle \\ \langle a,x,b,y\rangle, \langle a,x,y,b\rangle \\ \langle x,a,b,y\rangle, \langle x,a,y,b\rangle, \langle x,y,a,b\rangle \end{array} \right\}$$

The only new ingredient:

interleaving operator $\otimes$ must be abstracted ! ☺

Case: Availability of Single Expression

[Seidl/Steffen: ESOP 2000]

Abstract shuffle operator:

$\otimes^\#$	i	g	k
i	i	g	k
g	g	g	k
k	k	k	k

$$f_1 \otimes^\# f_2 := f_1 \cdot f_2 \sqcup f_2 \cdot f_1$$

The lattice:

k (ill)
|
i (gnore)
|
g (enerate)

Main lemma:

$$\forall f_j \in \{g, k, i\}: \quad \overbrace{f_n \circ \ldots \circ f_{j+1}}^{\in\{i\}} \circ \underbrace{f_j}_{\in\{g,k\} \,\vee\, j=1} \circ \ldots \circ f_1 = f_j$$

Treat other (separable) bitvector problems analogously...

$\Rightarrow$ precise interprocedural analyses for all bitvector problems ! ☺

Overview

- Introduction
- Fundamentals of Program Analysis
  - Excursion 1
- Interprocedural Analysis
  - Excursion 2
- Analysis of Parallel Programs
  - **Excursion 3**
  - Appendix
- Conclusion

Precise Fixpoint-Based Analysis of Programs with Thread-Creation and Procedures

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster

Joint work with:
Peter Lammich
[same place]

CONCUR 2007

(My) Main Interests of Recent Years

Data aspects

- algebraic invariants over $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{Z}_m$ ($m = 2^n$) in sequential programs, partly with recursive procedures
- invariant generation relative to Herbrand interpretation

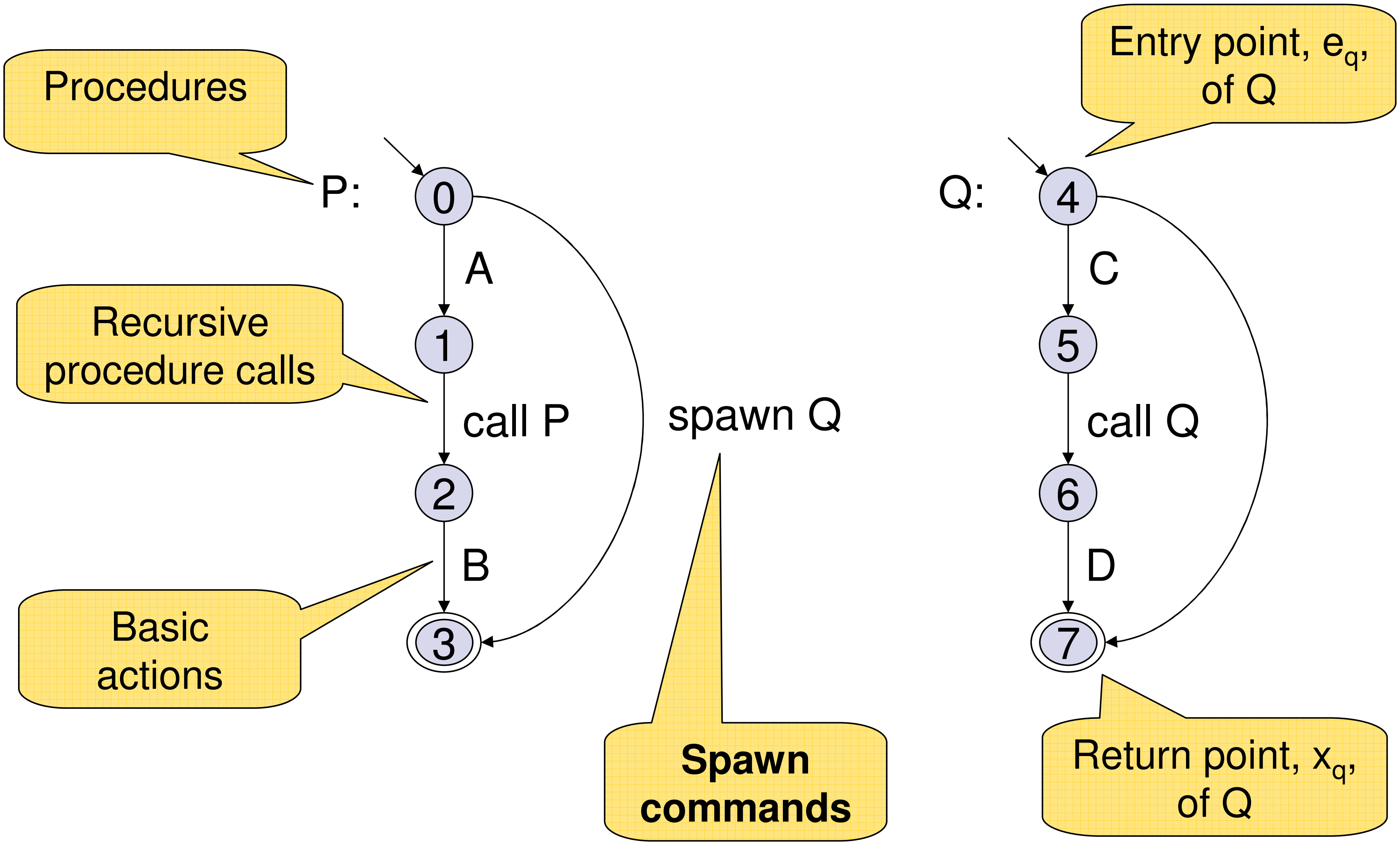
Control aspects

- recursion
- concurrency with **process creation / threads**
- synchronization primitives, in particular locks/monitors

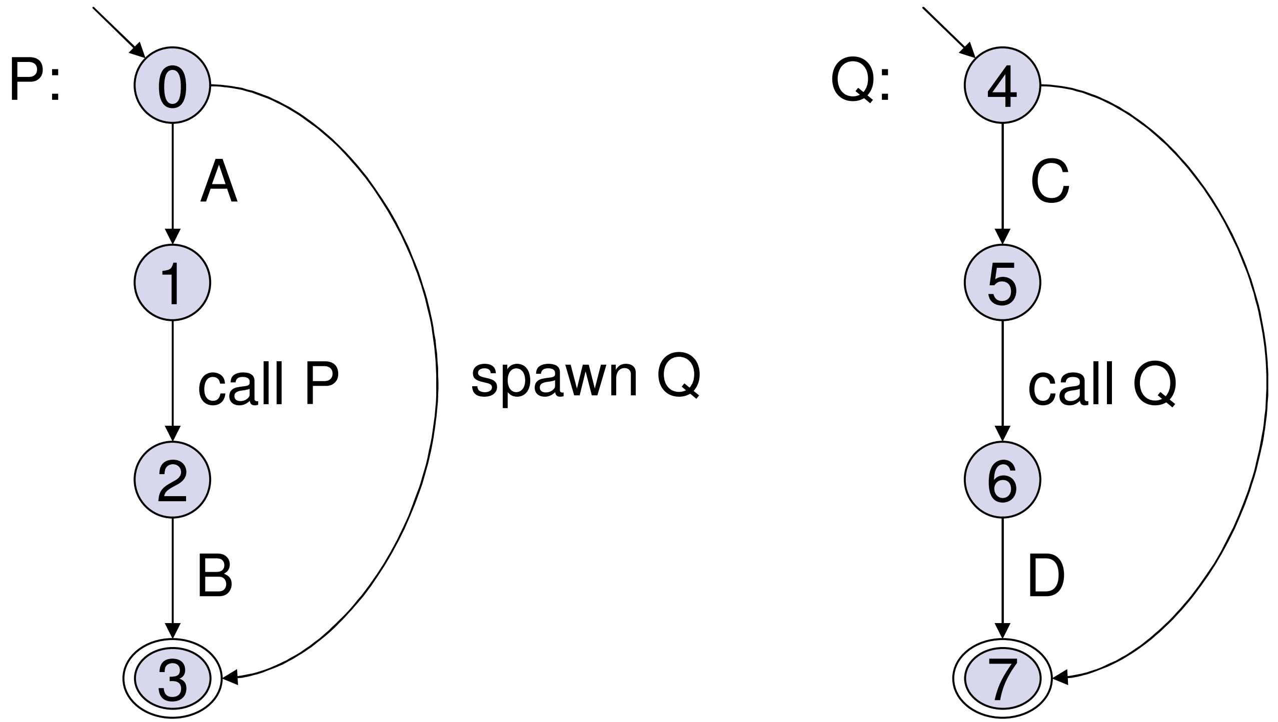
Technics used

- fixpoint-based
- automata-based
- (linear) algebra
- syntactic substitution-based techniques
- ...

Another Program Model



Spawns are Fundamentally Different



P induces trace language: $L = \bigcup \{ A^n \cdot (B^m \otimes (C^i \cdot D^j) \mid n \geq m \geq 0, i \geq j \geq 0 \}$

Cannot characterize L by constraint system with „·“ and „⊗“.

[Bouajjani, MO, Touili: CONCUR 2005] ☹

Gen/Kill-Problems

- Class of simple but important DFA problems
- Assumptions:
  - Lattice $(L,\sqsubseteq)$ is distributive
  - Transfer functions have form $f_e(l) = (l \sqcap kill_e) \sqcup gen_e$ with $kill, gen \in L$
- Examples:
  - bitvector problems, e.g.
  - available expressions, live variables, very busy expressions, ...

Data Flow Analysis

Goal:

Compute, for each program point u:

- Forward analysis: $MOP^F[u] = \alpha^F(Reach[u])$, where $\alpha^F(X) = \sqcup \{ f_w(x_0) \mid w \in X \}$
- Backward analysis: $MOP^B[u] = \alpha^B(Leave[u])$, where $\alpha^B(X) = \sqcup \{ f_w(\perp) \mid w^R \in X \}$

$$\text{Reach[u]} \;=\; \left\{ w \mid \exists c : \{[e_{Main}]\} \xrightarrow{w} c \;\wedge\; at_u(c) \right\}$$

$$\text{Leave[u]} \;=\; \left\{ w \mid \exists c : \{[e_{Main}]\} \xrightarrow{*} c \xrightarrow{w} \_ \;\wedge\; at_u(c) \right\}$$

$$at_u(c) \Leftrightarrow \exists w : (uw) \in c$$

$$f_w \;=\; f_{e_n} \circ \cdots \circ f_{e_1}, \;\; \text{for } w = e_1 \cdots e_n$$

Data Flow Analysis

Goal:

Compute, for each program point u:

- Forward analysis: $MOP^F[u] = \alpha^F(Reach[u])$, where $\alpha^F(X) = \sqcup \{ f_w(x_0) \mid w \in X \}$
- Backward analysis: $MOP^B[u] = \alpha^B(Leave[u])$, where $\alpha^B(X) = \sqcup \{ f_w(\bot) \mid w^R \in X \}$

Problem for programs with threads and procedures:

We cannot characterize Reach[u] and Leave[u] by a constraint system with operators „concatenation“ and „interleaving“.

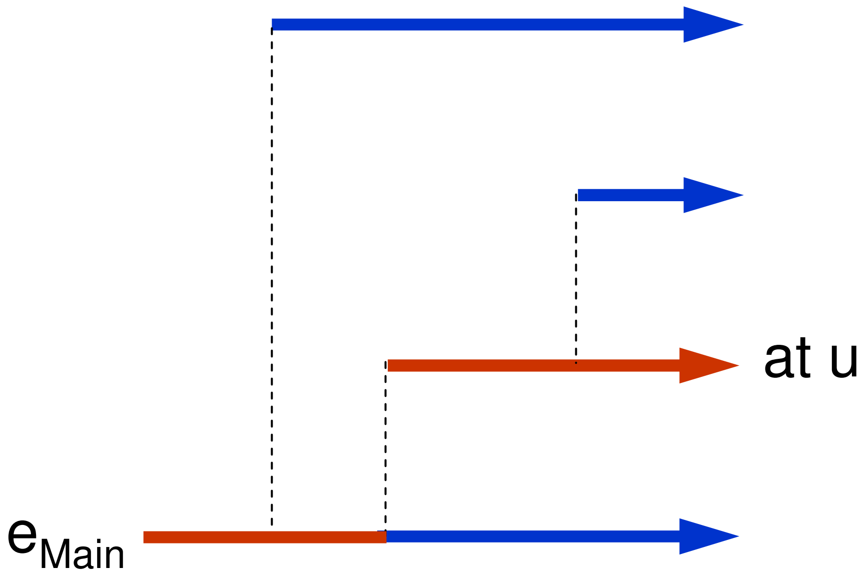
One Way Out

[Lammich/MO: CONCUR 2007]

- Derive alternative characterization of MOP-solution:
  - reason on level of execution paths
  - exploit properties of gen/kill-problems
- Characterize the path sets occuring as least solutions of constraint systems
- Perform analysis by abstract interpretation of these constraint systems

Forward Analysis

Directly Reaching Paths and Potential Interleaving



Reaching path: a suitable interleaving of the red and blue paths

Directly reaching path: the red path

Potential interference: set of edges in the blue paths (note: no order information!)

Formalization by augmented operational semantics with markers (see paper)

Forward MOP-solution

Theorem: For gen/kill problems:

$$\mathrm{MOP}^F[u] = \alpha^F(\mathrm{DReach}[u]) \sqcup \alpha^{PI}(\mathrm{PI}[u]),$$

$$\text{where } \alpha^{PI}(X) = \bigsqcup \{ \mathrm{gen}_e \mid e \in X \}.$$

Remark

- DReach[u] and PI[u] can be characterized by constraint systems (see paper)
- α^F(DReach[u]) and α^{PI}(PI[u]) can be computed by an abstract interpretation of these constraint systems

Characterizing Directly Reaching Paths

Same level paths:

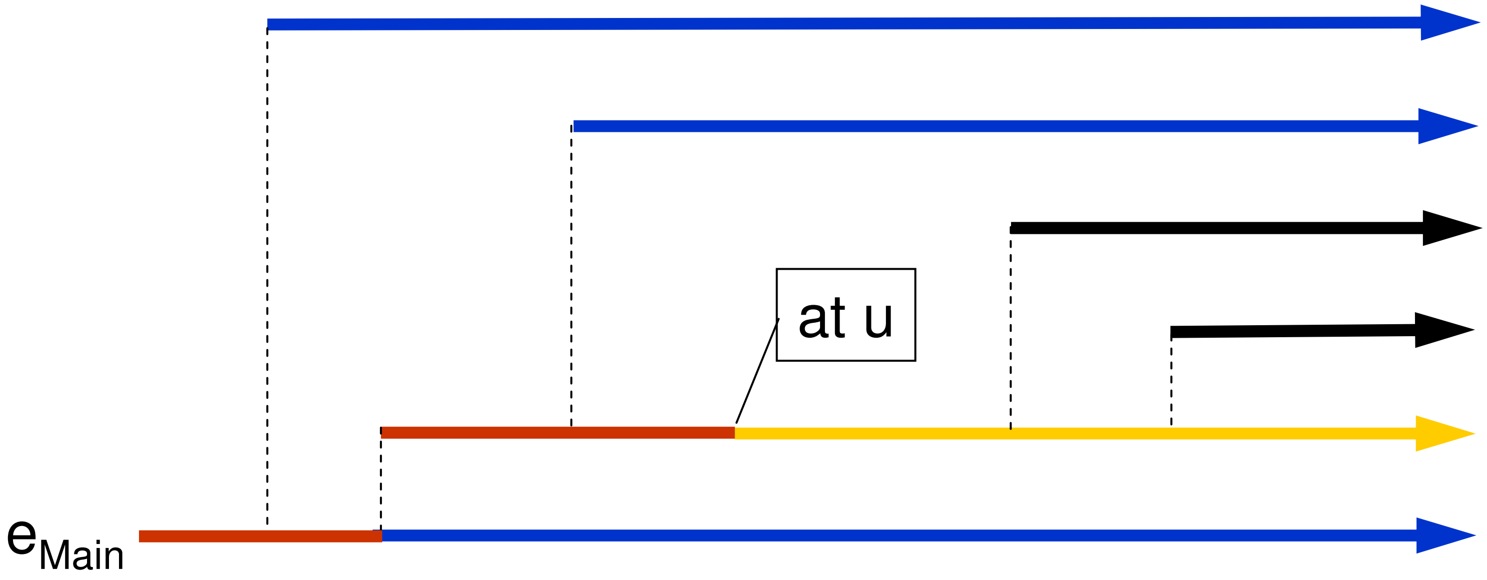
$$
\begin{array}{llll}
[\text{init}] & \mathsf{S}[\mathsf{e}_q] \supseteq \{\varepsilon\} & & \text{for } q \in P \\
[\text{base}] & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e & & \text{for } e = (u, \mathsf{base}\ \_, v) \in E \\
[\text{call}] & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e; \mathsf{S}[\mathsf{r}_q]; \mathsf{ret} & & \text{for } e = (u, \mathsf{call}\ q, v) \in E \\
[\text{spawn}] & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e & & \text{for } e = (u, \mathsf{spawn}\ q, v) \in E
\end{array}
$$

Directly reaching paths:

$$
\begin{array}{llll}
[\text{init}] & \mathsf{R}[\mathsf{e}_{\mathsf{main}}] \supseteq \{\varepsilon\} & & \\
[\text{reach}] & \mathsf{R}[u] \supseteq \mathsf{R}[\mathsf{e}_p]; \mathsf{S}[u] & & \text{for } u \in N_p \\
[\text{callp}] & \mathsf{R}[\mathsf{e}_q] \supseteq \mathsf{R}[u]; e & & \text{for } e = (u, \mathsf{call}\ q, \_) \in E \\
[\text{spawnp}] & \mathsf{R}[\mathsf{e}_q] \supseteq \mathsf{R}[u]; e & & \text{for } e = (u, \mathsf{spawn}\ q, \_) \in E
\end{array}
$$

Backwards Analysis

Directly Leaving Paths and Potential Interleaving



Leaving path: a suitable interleaving of orange, black and parts of blue paths

Directly leaving path: a suitable interleaving of orange and black paths

Potential interference: the edges in the blue paths

Formalization by augmented operational semantics with markers (see paper)

Interleaving from Threads created in the Past

Theorem: For gen/kill problems:

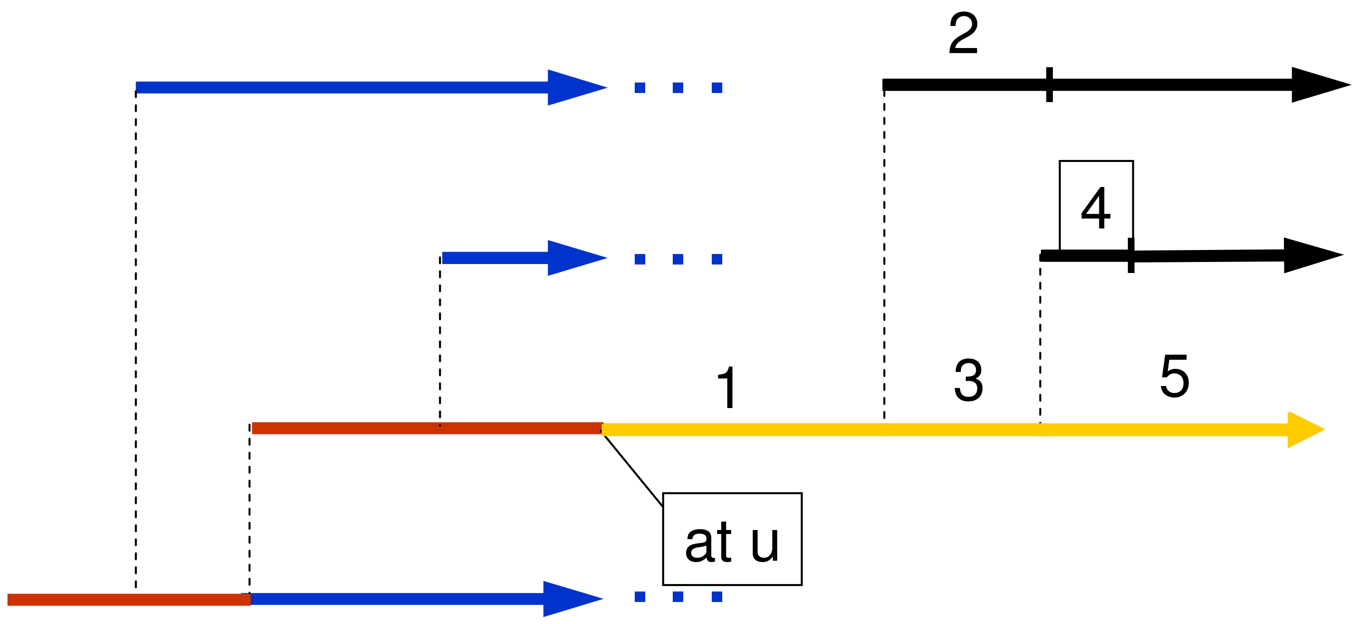
$$MOP^B[u] = \alpha^B(DLeave[u]) \sqcup \alpha^{PI}(PI[u]),$$

$$\text{where } \alpha^{PI}(E) = \bigsqcup \{ gen_e \mid e \in E \}.$$

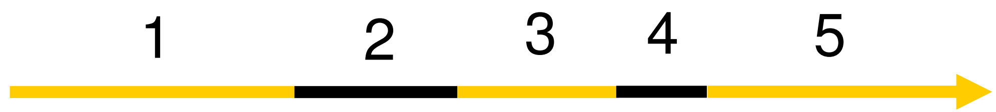
Remark

- We know no simple characterization of DLeave[u] by a constraint system.
- Main problem: Threads generated in a procedure instance survive that instance.

Representative Directly Leaving Paths



A representative directly leaving path:



Interleaving from Threads created in the Future

Lemma

$\alpha^B(\text{DLeave}[u]) = \alpha^B(\text{RDLeave}[u])$ (for gen/kill problems).

Corollary

$\text{MOP}^B[u] = \alpha^B(\text{RDLeave}[u]) \sqcup \alpha^{PI}(\text{PI}[u])$ (for gen/kill problems).

Remark

- RDLeave[u] and PI[u] can be characterized by constraint systems (see paper)
- $\alpha^B(\text{RDLeave}[u])$ and $\alpha^{PI}(\text{PI}[u])$ can be computed by an abstract interpretation of these constraint systems

Also in the Paper

- Formalization of these ideas
  - constraint systems for path sets
  - validation with respect to operational semantics
- Parallel calls in combination with threads
  - threads become trees instead of stacks ...
- Analysis of running time:
  - global information in time linear in the program size

Summary

- Forward- and backward gen/kill-analysis for programs with threads and procedures
- More efficient than automata-based approach
- More general than known fixpoint-based approach
- Current work: Precise analysis in presence of locks/monitors (see papers at SAS 2008, CAV 2009 for first results)

End of Excursion 3

Appendix

Regular Symbolic Analysis of Dynamic Networks of Pushdown Systems

DPNs: Dynamic Pushdown-Networks

A *dynamic pushdown-network* (over a finite set of actions Act) consists of:

- P, a finite set of control symbols
- Γ, a finite set of stack symbols
- Δ, a finite set of rules of the following form

$$p\gamma \xrightarrow{a} p_1 w_1$$

$$p\gamma \xrightarrow{a} p_1 w_1 \rhd p_2 w_2$$

(with $p, p_1, p_2 \in$ P, $\gamma \in \Gamma$, $w_1, w_2 \in \Gamma^*$, $a \in$ Act).

DPNs: Dynamic Pushdown-Networks

A State of a DPN is a word in $(P\Gamma^*)^+$:

$$p_1 w_1 p_2 w_2 \cdots p_k w_k \qquad (\text{with } p_i \in P, w_i \in \Gamma^*, k > 0)$$

... an infinite state space

The transition relation of a DPN:

$$\left(p\gamma \xrightarrow{a} p_1 w_1\right) \in \Delta: \qquad u\,p\,\gamma v \xrightarrow{a} u\,p_1\,w_1\,v$$

$$\left(p\gamma \xrightarrow{a} p_1 w_1 \rhd p_2 w_2\right) \in \Delta: \qquad u\,p\,\gamma v \xrightarrow{a} u\,p_2\,w_2\,p_1\,w_1\,v$$

Example

Consider the following DPN with a single rule

$$p\gamma \xrightarrow{a} p\gamma\gamma \triangleright q\gamma$$

Transitions:

$$p\gamma$$
$$q\gamma p\gamma\gamma$$
$$q\gamma q\gamma p\gamma\gamma\gamma$$
$$q\gamma q\gamma q\gamma p\gamma\gamma\gamma\gamma$$
$$q\gamma q\gamma q\gamma q\gamma p\gamma\gamma\gamma\gamma\gamma$$
$$\vdots$$

Reachability Analysis

Given:

- Model of a system: M
- Set of system states: Bad

Reachability analysis:

- Can a state from Bad be reached from an initial states of the system?

$$\exists \sigma_0, \ldots, \sigma_k : \ \mathrm{Init} \ni \sigma_0 \to \cdots \to \sigma_k \in \mathrm{Bad} \quad ?$$

Applications:

- Check safety properties:

  Bad is a set of states to be avoided

- More applications by iterated computation of reachability sets for sub-models of the system model, e.g. data-flow analysis... ☺

Reachability Analysis

Given:

- Model of a system: M
- Set of system states: Bad

Reachability analysis:

- Can a state from Bad be reached from an initial state of the system?

$$\exists \sigma_0, \ldots, \sigma_k : \ \mathrm{Init} \ni \sigma_0 \to \cdots \to \sigma_k \in \mathrm{Bad} \quad ?$$

Def.: - pre*(X) $=_{df} \{ \sigma \mid \exists \sigma' \in X : \sigma \to^{*} \sigma' \}$

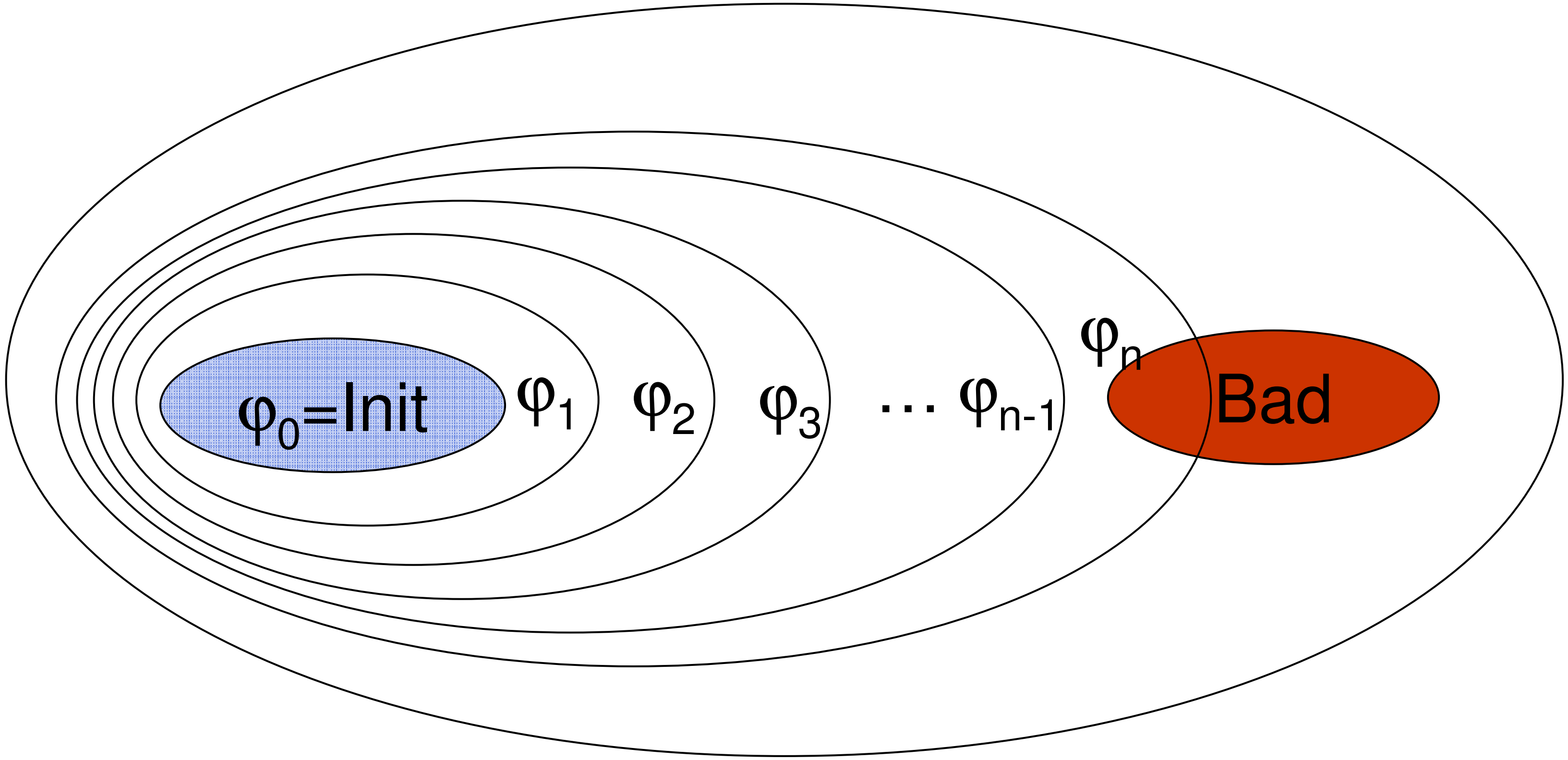
- post*(X) $=_{df} \{ \sigma \mid \exists \sigma' \in X : \sigma' \to^{*} \sigma \}$

Equivalent formulations of reachability analysis:

- pre*(Bad) $\cap$ Init $\neq \emptyset$
- post*(Init) $\cap$ Bad $\neq \emptyset$

$\Rightarrow$ Computation of pre* or post* is key to reachability analysis

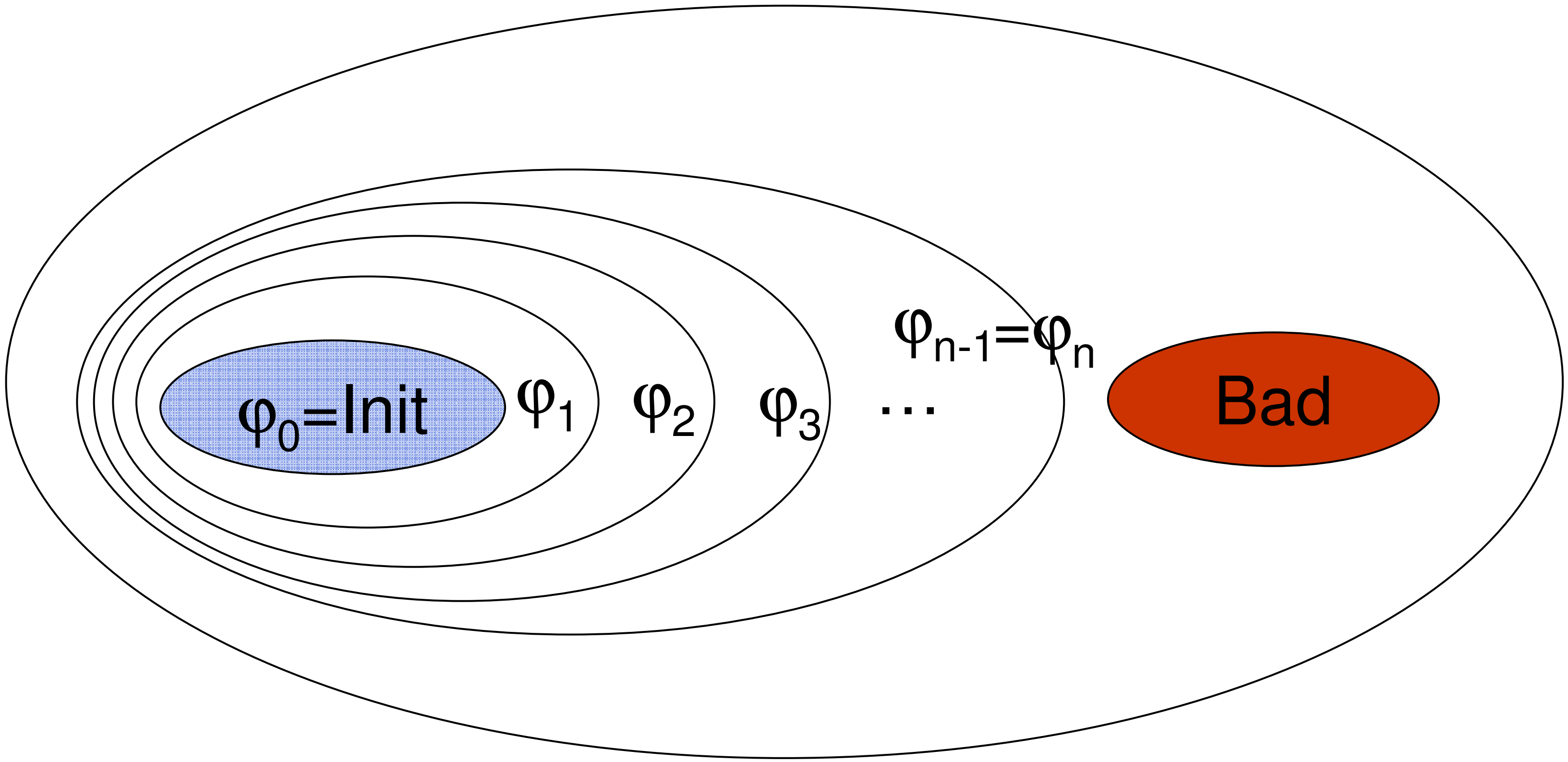
Reachability Analysis of Finite State Systems



$$\varphi_0 =_{df} \text{Init}$$
$$\varphi_{i+1} =_{df} \varphi_i \cup \text{post}(\varphi_i)$$
$$\text{post}(X) =_{df} \{\sigma \mid \exists \sigma' \in X : \sigma \to \sigma'\}$$

$\Rightarrow$ Bad reachable from initial state

Reachability Analysis of Finite State Systems



$$\varphi_0 =_{df} \text{Init}$$
$$\varphi_{i+1} =_{df} \varphi_i \cup \text{post}(\varphi_i)$$
$$\text{post}(X) =_{df} \{\sigma \mid \exists \sigma' \in X : \sigma \to \sigma'\}$$

$\Rightarrow$ Bad **not** reachable from initial state

Problems with Infinite-State Systems

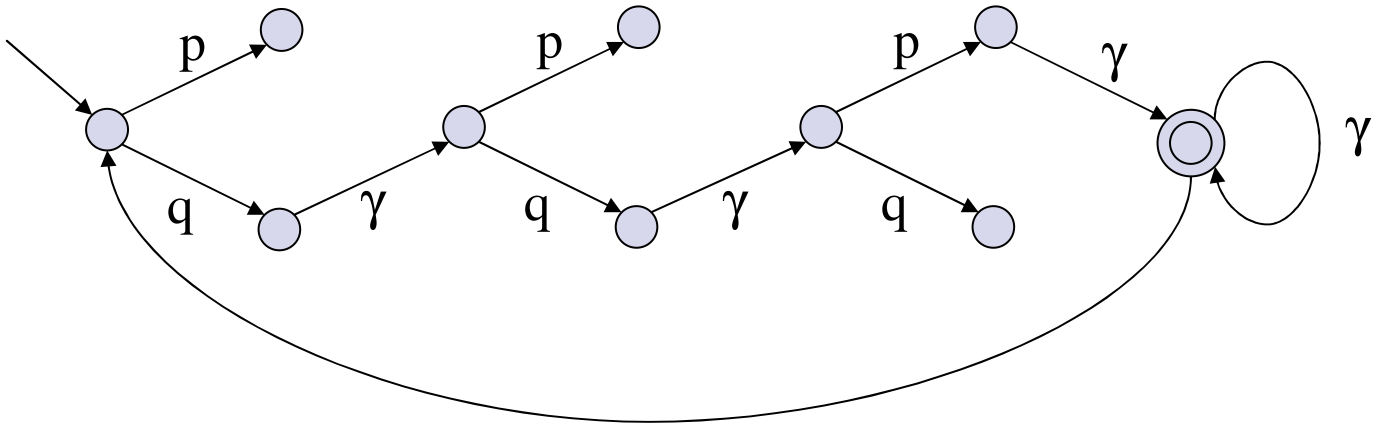
- State sets φ_i can be infinite

  $\Rightarrow$ symbolic representation of (certain) infinite state sets

  Here: by finite automata

Example: Representation of an Infinite State Set of a DPN by a Word Automaton

An automaton A:



The *regular* set of states represented by A:

$$L(A) = \left(q\gamma q\gamma p\gamma^{*}\right)^{*}$$

... an infinite set of states. ☺

Problems with Infinite-State Systems

- State sets φ_i can be infinite

  $\Rightarrow$ symbolic representation of (certain) infinite state sets

  Here: by finite (word) automata

- Iterated computation of reachability sets does not terminate in general

  $\Rightarrow$ Methods for *acceleration* of the computation

  Here: by computing with finite automata

Computing pre* for DPNs with Finite Automata

Theorem [Bouajjani, MO, Touili, 2005]

For every DPN and every regular state set R,
pre*(R) is regular and can be computed in polynomial time.

[Bouajjani/Esparza/Maler, 1997]

Proof:

Generalization of a known technique for single pushdown systems:
saturation of an automaton for R.

$\Rightarrow$ Reachability analysis is effective for regular sets Bad of states !

Example: Reachability Analysis for DPNs

Consider again DPN with the rule

$$p\gamma \xrightarrow{a} p\gamma\gamma \triangleright q\gamma$$

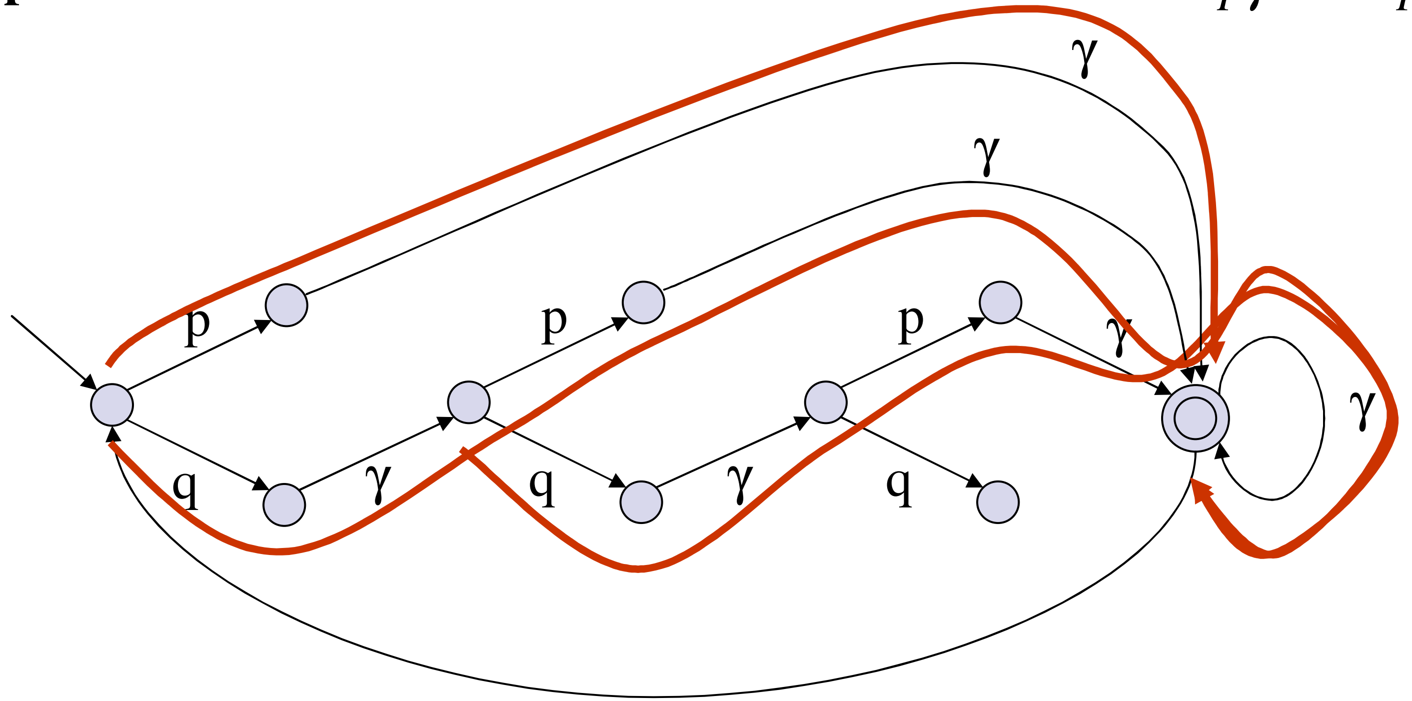
and the infinite set of states

$$\text{Bad} = \left(q\gamma q\gamma p\gamma^{*}\right)^{*} = L(A)$$

Analysis problem: can Bad be reached from pγ ?

Example: Reachability Analysis for DPNs

1. Step: Saturate automaton for Bad with the DPN rule: $p\gamma \xrightarrow{a} p\gamma\gamma \triangleright q\gamma$

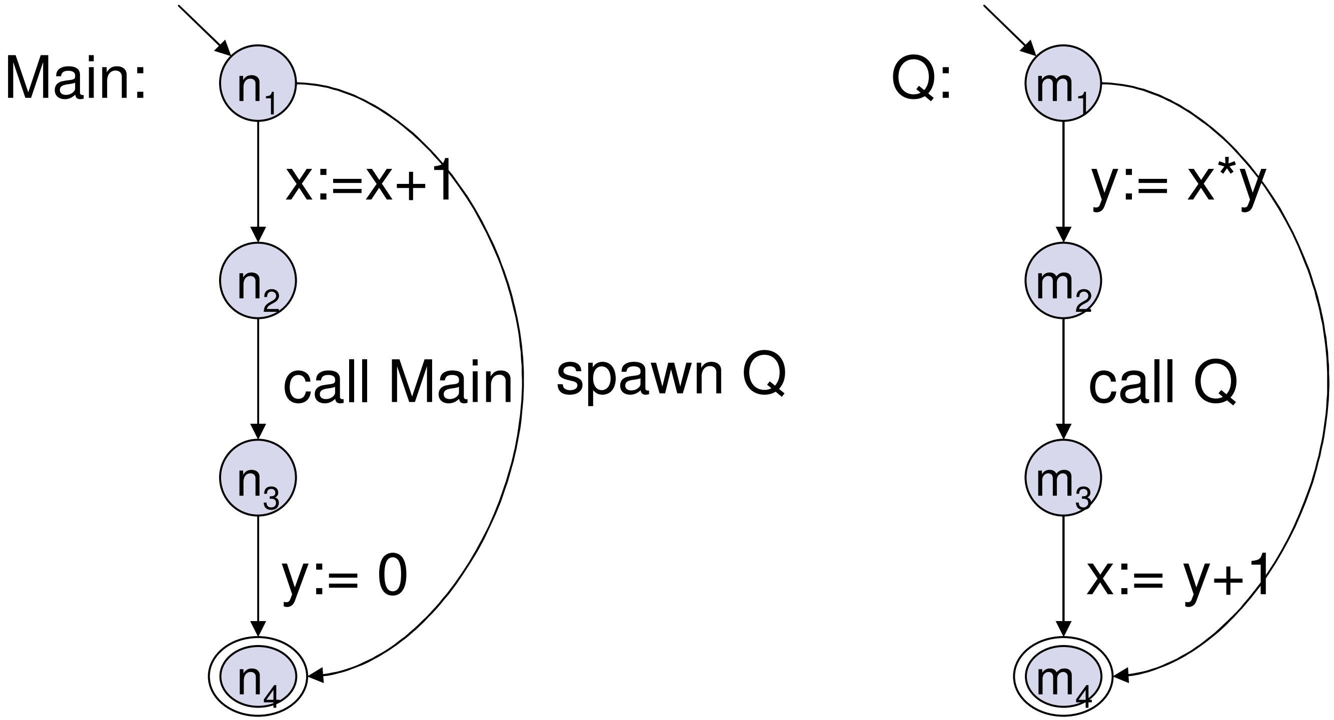


Resulting automaton A_{pre*} represents pre*(Bad) !

2. Step: Check, whether pγ is accepted by A_{pre*} or not

Result: Bad is reachable from pγ, as A_{pre*} accepts pγ.

Modelling Programs with Procedures and Threads by DPNs



$$\#N_1 \xrightarrow{x:=x+1} \#N_2$$

$$\#N_2 \xrightarrow{call_P} \#N_1N_3$$

$$\#N_3 \xrightarrow{y:=0} \#N_4$$

$$\#N_1 \xrightarrow{spawn_Q} \#N_4 \triangleright \#M_1$$

$$\#M_1 \xrightarrow{y:=x*y} \#M_2$$

$$\#M_2 \xrightarrow{call_Q} \#M_1M_3$$

$$\#M_3 \xrightarrow{x:=y+1} \#M_4$$

$$\#N_1 \xrightarrow{skip} \#M_4$$

Live Variables Analysis
via
Iterated pre[*]-computation

Esparza, Knoop

Steffen, Schmidt

Observation

Variable x is *live* at u

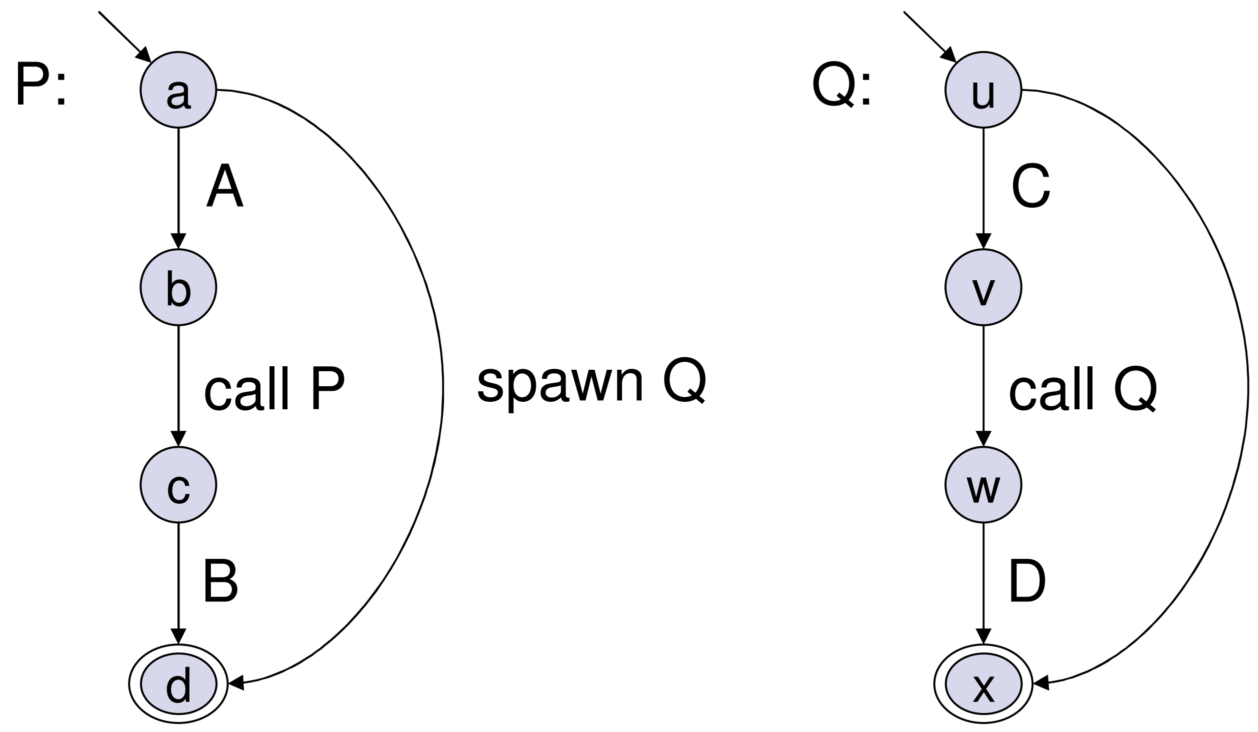
iff

$$e_{Main} \in pre^*(At_u \cap pre^*_{\Delta_{non-def}}(pre_{\Delta_{use}}(Conf)))$$

Remark

This condition can be checked by computing with automata

A Non-Representability Result



- P induces trace language: $L = \bigcup \{ A^n \cdot (B^m \otimes (C^i \cdot D^j)) \mid n \geq m \geq 0, i \geq j \geq 0 \}$
- L cannot be characterized by constraint system with operators „concatenation" and „interleaving"

Forward Reachability Analysis of DPNs

Observation [Bouajjani, MO, Touili, 2005]

In general, post*(R) is not regular, not even if R is finite.

Example:

Consider DPN with the rule $p\gamma \xrightarrow{a} p\gamma\gamma \triangleright q\gamma$

Recall:

$$p\gamma$$
$$q\gamma p\gamma\gamma$$
$$q\gamma q\gamma p\gamma\gamma\gamma$$
$$q\gamma q\gamma q\gamma p\gamma\gamma\gamma\gamma$$
$$q\gamma q\gamma q\gamma q\gamma p\gamma\gamma\gamma\gamma\gamma$$
$$\vdots$$

post*({pγ}) = $\{ (q\gamma)^k p\gamma^{k+1} \mid k \geq 0 \}$ is not regular.

Theorem [Bouajjani, MO, Touili, 2005]

For every DPN, post*(R) is *contextfree* if R is contextfree.

It can be computed in polynomial time.

A Little Bit of Synchronization ...

- CDPNs – Constrained Dynamic Pushdown Networks
- Idea: Threads can observe (stable regular patterns of) their children, but not vice versa
- States are represented by trees in order to mirror father/child relationship
- Use tree automata techniques for
  - representation of state sets and
  - symbolic computation of pre* (under certain conditions)
- See the CONCUR 2005 paper
- More recent papers: lock and monitor-sensitive analysis

Comparison of Fixpoint-based and Automata-based Algorithm

Fixpoint-based algorithm: [Lammich/MO: CONCUR 2007]

- computes information for all program points at once
  in linear time
- can use bitvector operations for computing multiple bits at once

Automata-based algorithm: [Bouajjani/MO/Touili: CONCUR 2005]

- based on pre^*-computations of regular sets of configurations
- needs linear time for each program point:
  thus: overall running time is quadradic
- must be iterated for each bit
- more generic w.r.t. sets of configurations

End of Appendix

Conclusion

- Program analysis very broad topic
- Provides generic analysis techniques for (software) systems
- Here just one path through the forest
- Many interesting topics not covered

Thank you !