Fundamentals of Program Analysis
+
Generation of Linear Prg. Invariants

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster, Germany

2nd Tutorial of
SPP RS3: Reliably Secure Software Systems

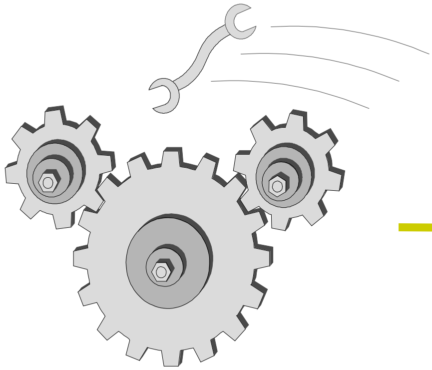
Schloss Buchenau, September 3-6, 2012

Dream of Automatic Analysis

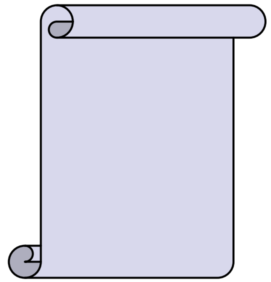
program

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

analyzer



result



$$G(\Phi \rightarrow F\Psi)$$

specification of property

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

- Conclusion

Apology for not giving proper credit in these lectures !

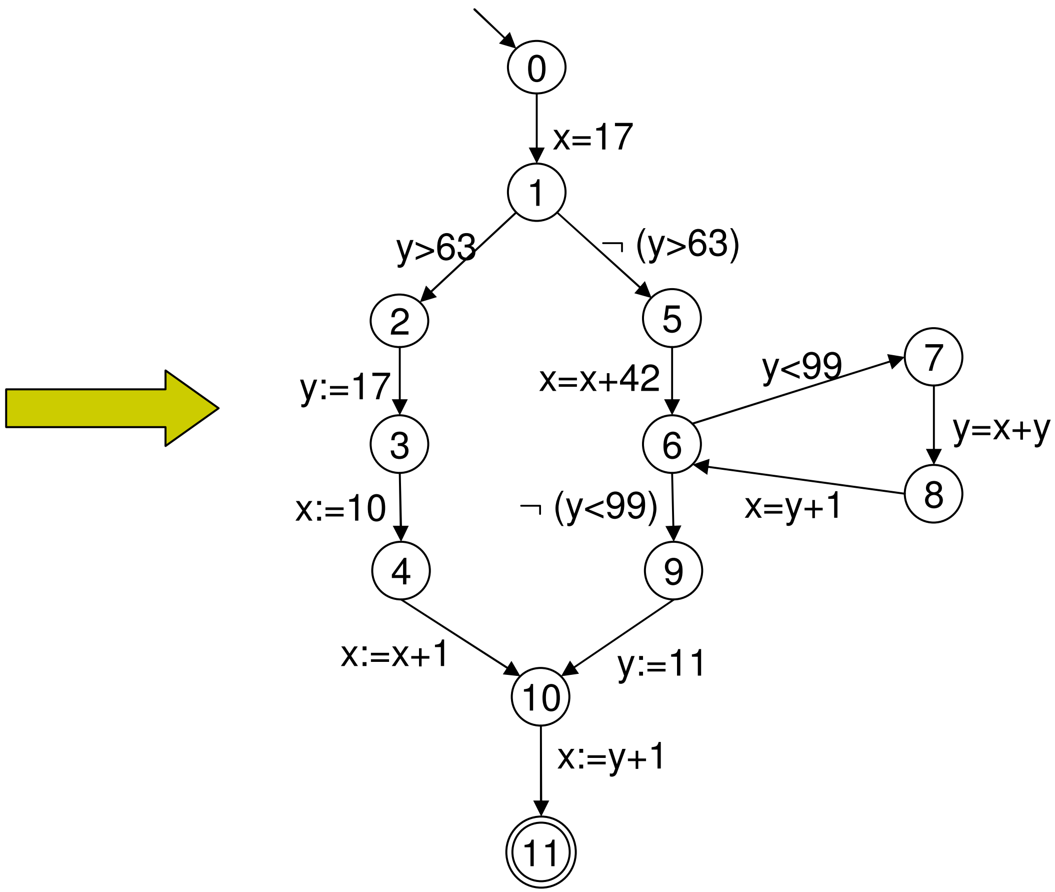
Overview

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

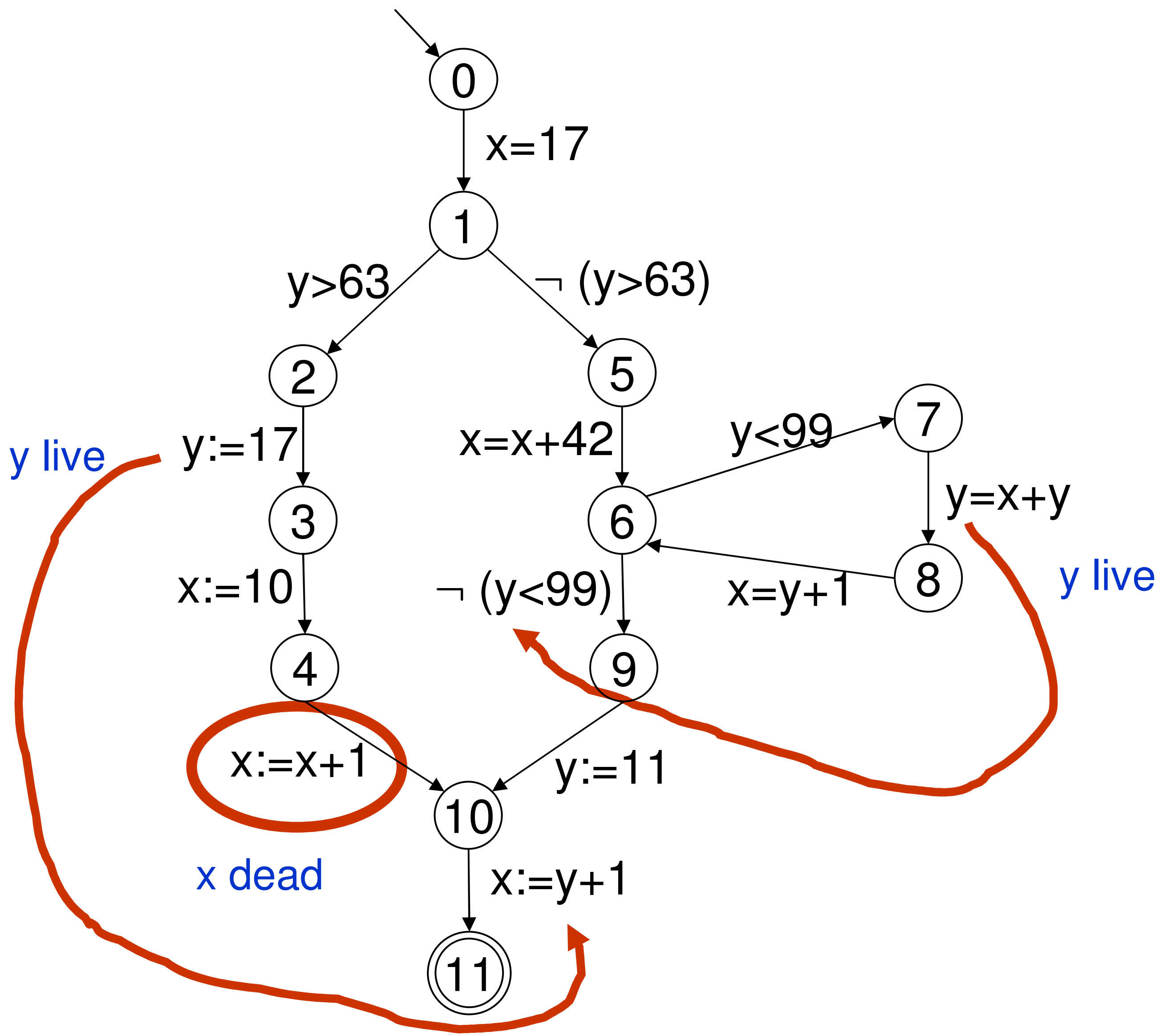
1) perform *live variables* analyses:

variable x is live at program point u iff

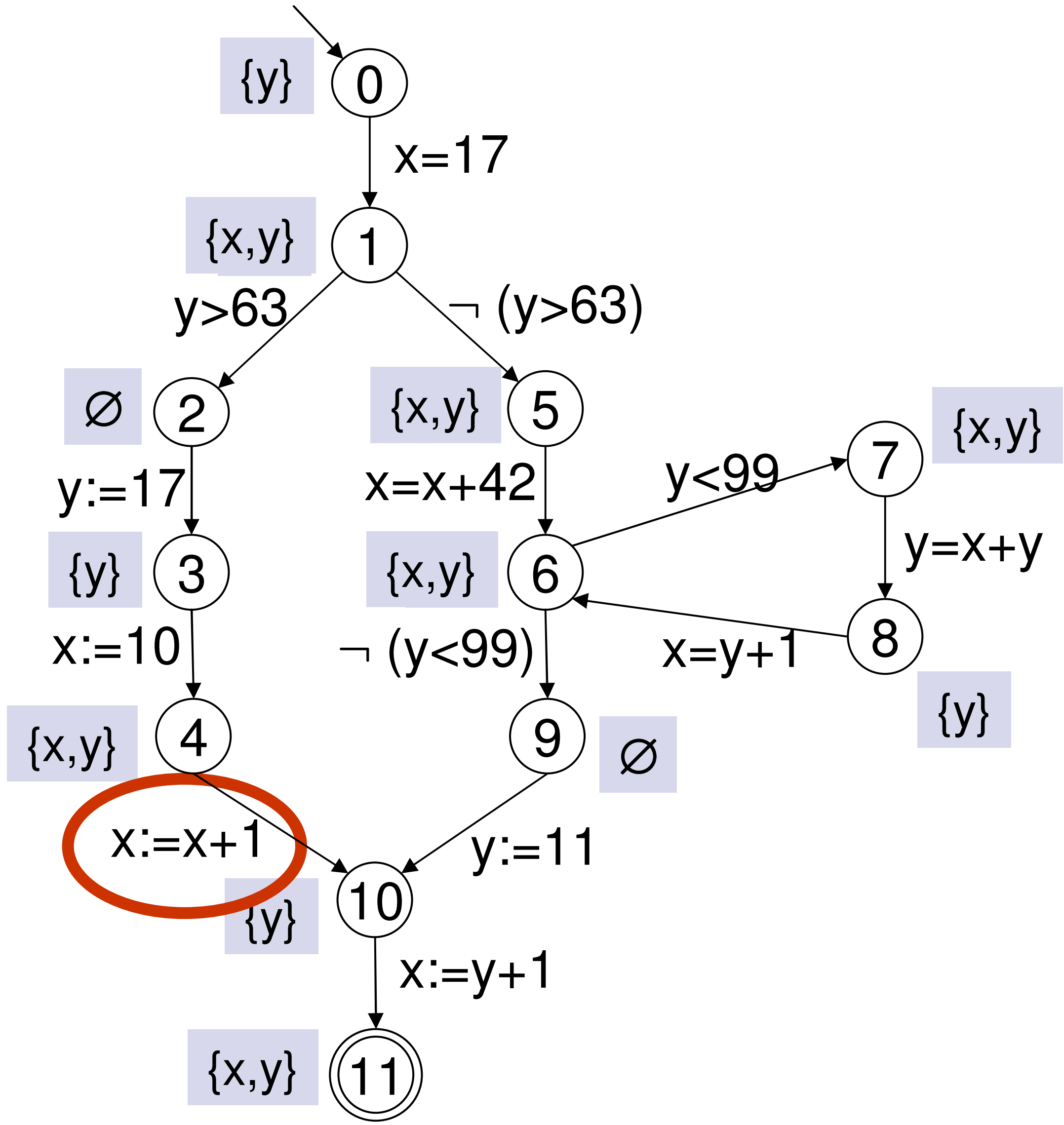
there is a path from u on which x is used before it is modified

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:

- $(P(\mathrm{Var}),\subseteq)$ 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 $(L, \sqsubseteq) = (\mathcal{P}(\mathrm{Var}), \subseteq)$ of:

$$
\begin{array}{lll}
A[\mathit{fin}] & \sqsupseteq & \mathit{init}, \qquad \text{for } \mathit{fin}\text{, the termination node} \\
A[u] & \sqsupseteq & f_e(A[v]), \qquad \text{for each edge } e = (u, s, v)
\end{array}
$$

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

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

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

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.

Constant Propagation

The Goal:

Find for each program point expressions or variables that have a constant value at this program point

Enabled Optimizations:

- Replace constant variables or expressions at compile time by their value
  - $\rightarrow$ smaller and faster code
- Eliminate unreachable branches in the program
  - $\rightarrow$ smaller code

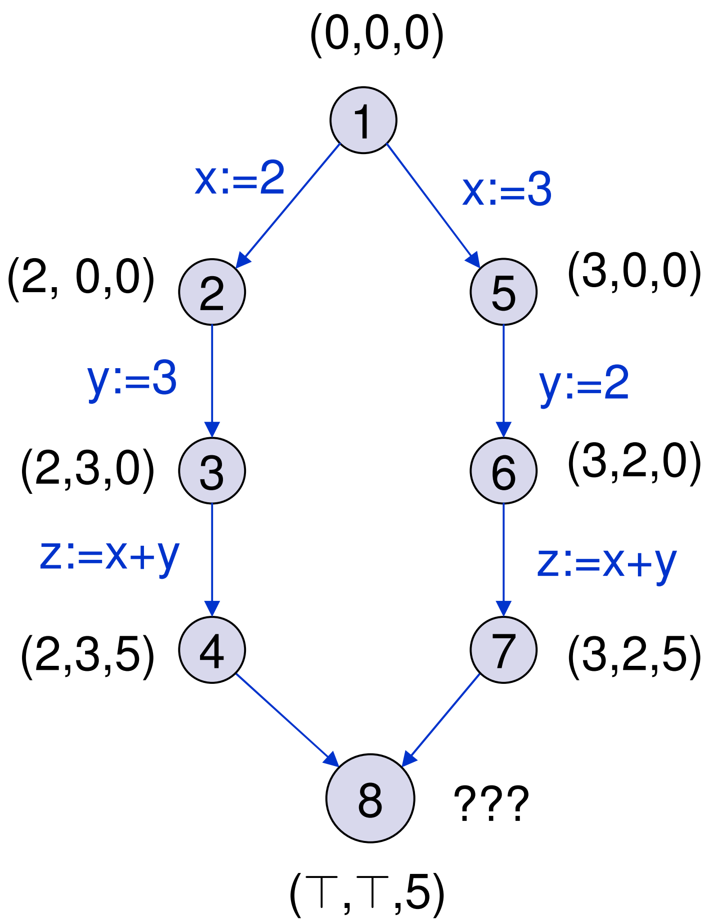
Remarks:

Constant expressions and variables appear often, e.g.;

- const-declarations in PASCAL
- final attributes in JAVA-interfaces, ...
- values computed out of declared constants

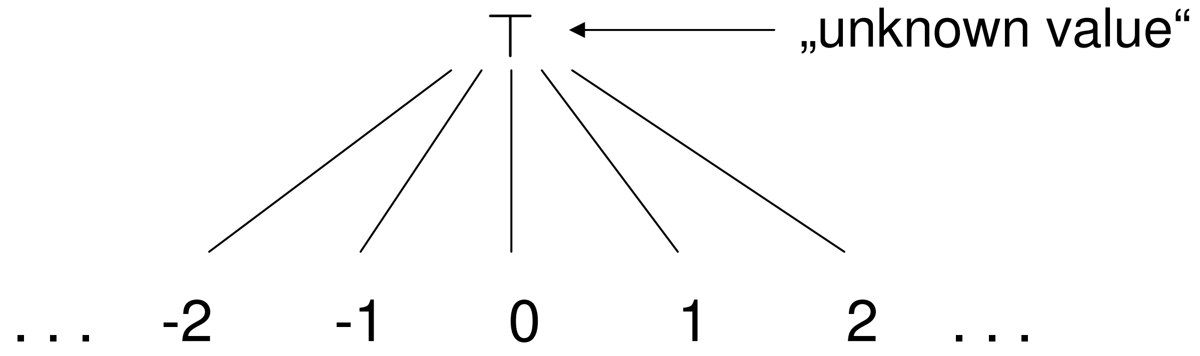
Constant Propagation

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



A Lattice for Constant Propagation

An order $\sqsubseteq$ on $\mathbb{Z} \cup \{\top\}$:



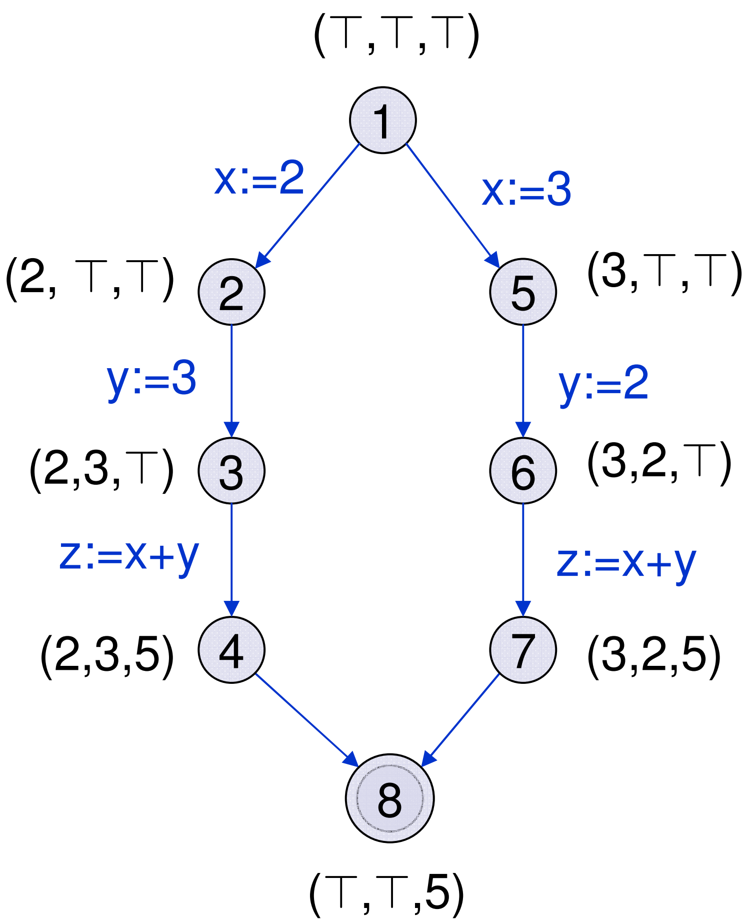
$$L_{\mathsf{CP}} = \{\rho \mid \rho : \mathsf{Var}_G \to (\mathbb{Z} \cup \{\top\})\} \cup \{\bot\}$$

$$\rho \sqsubseteq_{\mathsf{CP}} \rho' \;:\Leftrightarrow\; \rho = \bot \vee \left(\rho \neq \bot \wedge \rho' \neq \bot \wedge \forall x \in \mathsf{Var}_G : \; \rho(x) \sqsubseteq \rho'(x)\right)$$

Remark: $(L_{\mathsf{CP}}, \sqsubseteq_{\mathsf{CP}})$ is a complete lattice.

Constant Propagation

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



Specifying Constant Propagation by a Constraint System

Sei $G = (N,E,st,te)$ ein Flussgraph über BA_{std}.

Compute (smallest) solution over $(L,\sqsubseteq) = (L_{CP},\sqsubseteq_{CP})$ of:

$$V[st] \sqsupseteq init, \qquad \text{for } st\text{, the start note}$$

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

where:

$init = \top_{CP} \in L_{CP}$ is the mapping $\top_{CP}(x) = \top$ and

$f_e: L_{CP} \to L_{CP}$ is defined by

$$f_e(\rho) =_{df} \begin{cases} \rho\{[\![t]\!]_{CP}(\rho) / x\}, & \text{if } e = (u,x:=t,v) \text{ und } \rho \neq \bot \\ \rho, & \text{otherwise} \end{cases}$$

Specifying Constant Propagation by a Constraint System

Remarks:

1. Again, every solution is „correct“ (whatever this means).
2. Again, the smallest solution is called **MFP-solution**;
   it comprises a value MFP[u] $\in$ L for each program point u.
3. 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, constant propagation is a ***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

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

Other examples: reaching definitions, available expressions, ...

Monotone Data-Flow Problems

Goal:

- A generic notion that captures what is common for different analyses

Advantages:

- Study general properties of data flow problems independently of concrete analysis questions
- Build efficient, generic implementations

Monotone Data-Flow Problems

Definition:

A monotone data-flow problem is a tuple $P = ((L,\sqsubseteq),F,(N,E),st,init)$ consisting of:

- a complete lattice $(L,\sqsubseteq)$.
  The elements of L are called (data-flow facts).
- a set F of transfer functions $f: L \to L$, such that:
  - each $f \in F$ is monotone: $\forall\, x,y \in L :\ x \sqsubseteq y \ \Rightarrow f(x) \sqsubseteq f(y)$
  - $id \in F$
  - F is closed under composition: $\forall\, f,g \in F : f \circ g \in F$.
- A graph (N, E) with a finite set of edges $N;$
  each node of the graph is annotated with a transfer function $f \in F$:
  $$E \subseteq N \times F \times N \,.$$
- $st \in N$ is a designated initial node.
- $init \in L$ is a designated initial information.

Constraint System for a Data-Flow Problem

Let $P = ((L,\sqsubseteq),F,(N,E),u_0,\text{init})$ be a data-flow problem

Compute (smallest) solution over $(L,\sqsubseteq)$ of the followi constraint system:

$$
\begin{array}{lll}
A[st] & \sqsupseteq \text{init}, & \text{for st, the start node} \\
A[v] & \sqsupseteq f(A[u]), & \text{for each node } e = (u,f,v) \in E
\end{array}
$$

Note:

Here, information flows from nodes to their successor nodes only.

Hence, for backwards analyses the direction of the edges must be reversed when mapping it to the corresponding data-flow problem.

Constraint System for a Data-Flow Problem

Remarks:

1. Again, every solution is „correct“ (whatever this means).
2. Again, the smallest solution is called **MFP-solution**;
   it comprises a value MFP[u] $\in$ L for each program point u.
3. The MFP-solution is the **most precise** one.

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:

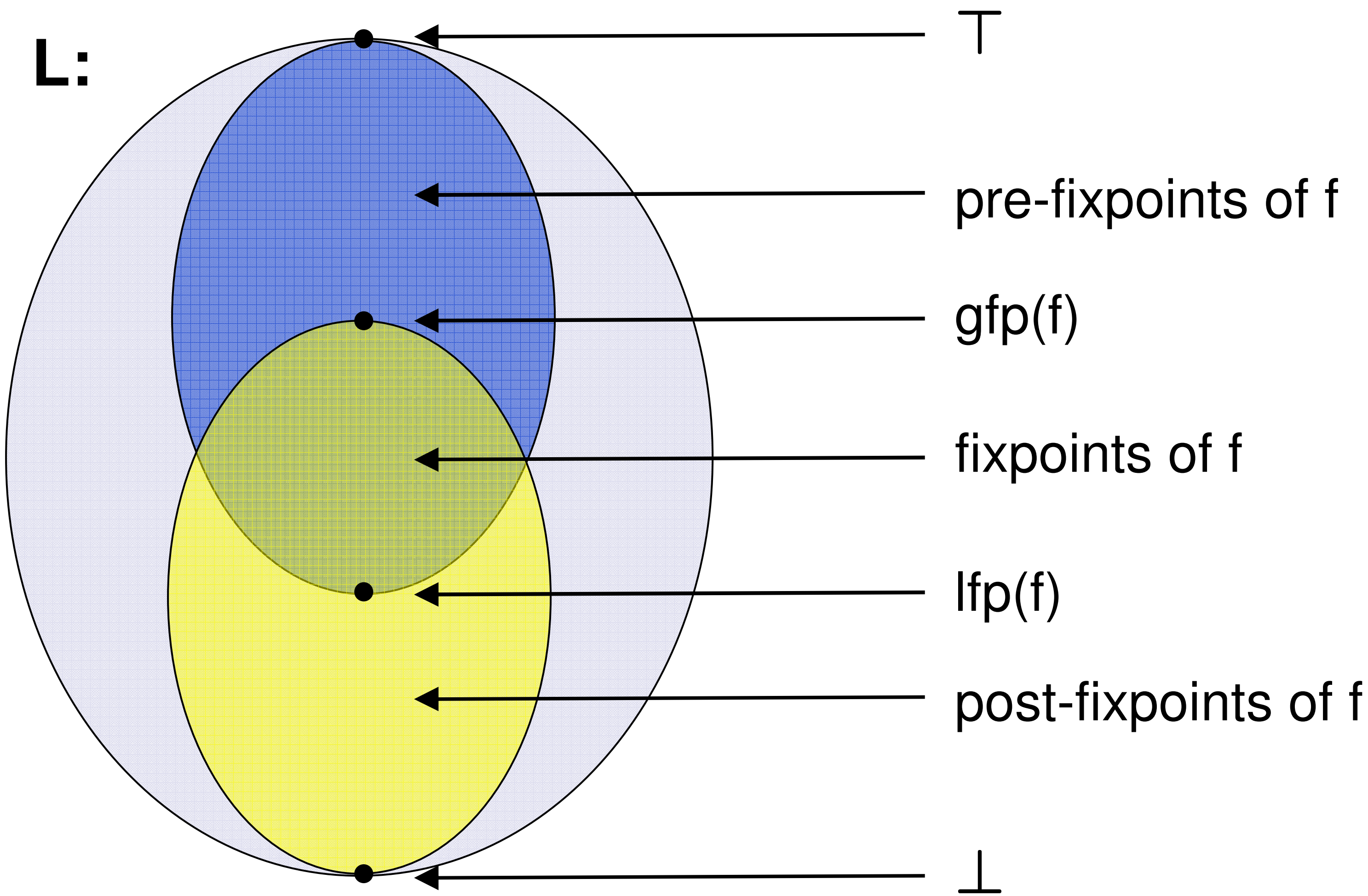
Every monotonic function f on a complete lattice L has a least fixpoint lfp(f) and a greatest fixpoint gfp(f).

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

$A[st] = init;$

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

$\quad u = Extract(W);$

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

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

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

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

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

$\quad\quad \}$

$\quad \}$

$\}$

Invariants of the Main Loop

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

b1) $A[st] \sqsupseteq init$

b2) $u \notin W \Rightarrow A[v] \sqsupseteq f_e(A[u])$ 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/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) $\{\ A[v] = \bot;\ W = W \cup \{v\};\ \}$

$A[st] = init;$

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

$\quad u = Extract(W);$

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

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

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

$\quad\quad\quad A[v] = A[v] \triangledown t;$

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

$\quad\quad \}$

$\quad \}$

$\}$

Invariants of the Main Loop

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

b1) $A[st] \sqsupseteq init$

b2) $u \notin W \Rightarrow A[v] \sqsupseteq f_e(A[u])$ 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)

$\Rightarrow A[u] \sqsupseteq MFP[u]$ 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...

$$\left(L,\sqsubseteq\right) \;=\; \left(\left\{\, [l,u] \;\middle|\; l\in\mathbb{Z}\cup\{-\infty\}, u\in\mathbb{Z}\cup\{+\infty\}, l\le u\right\}\cup\{\varnothing\},\ \subseteq\right)$$

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

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

A widening operator:

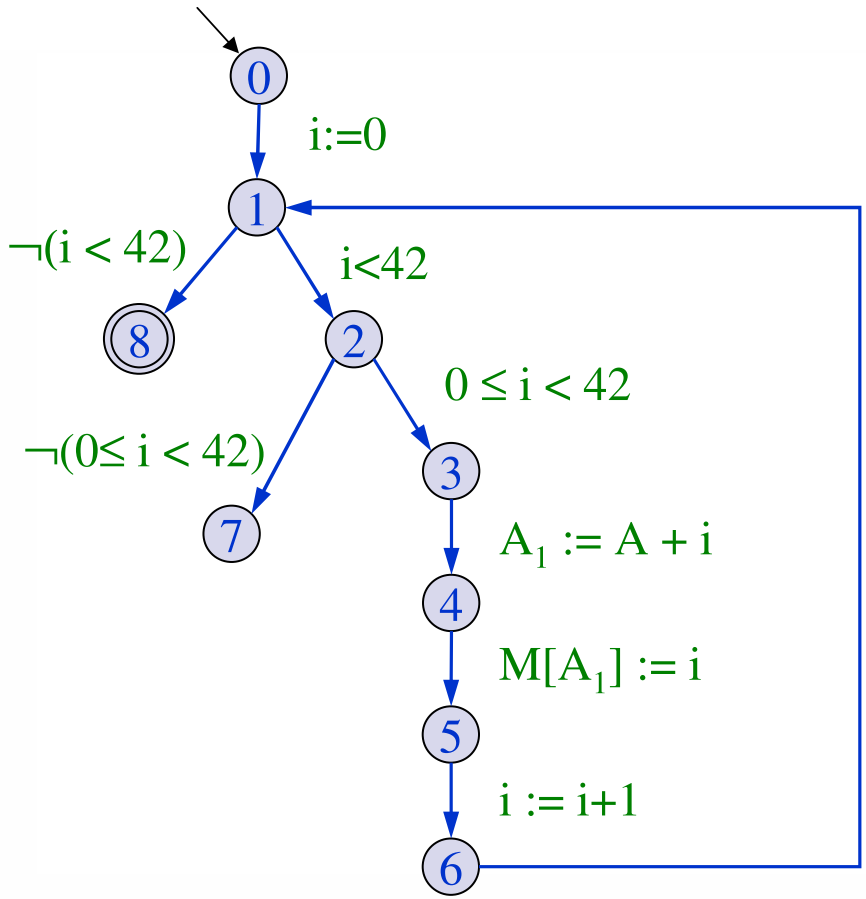
$[l_0,u_0]\ \triangledown\ [l_1,u_1]=[l_2,u_2]$, where

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

A chain of maximal length arising with this widening operator:

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

Analyzing the Program with the Widening Operator



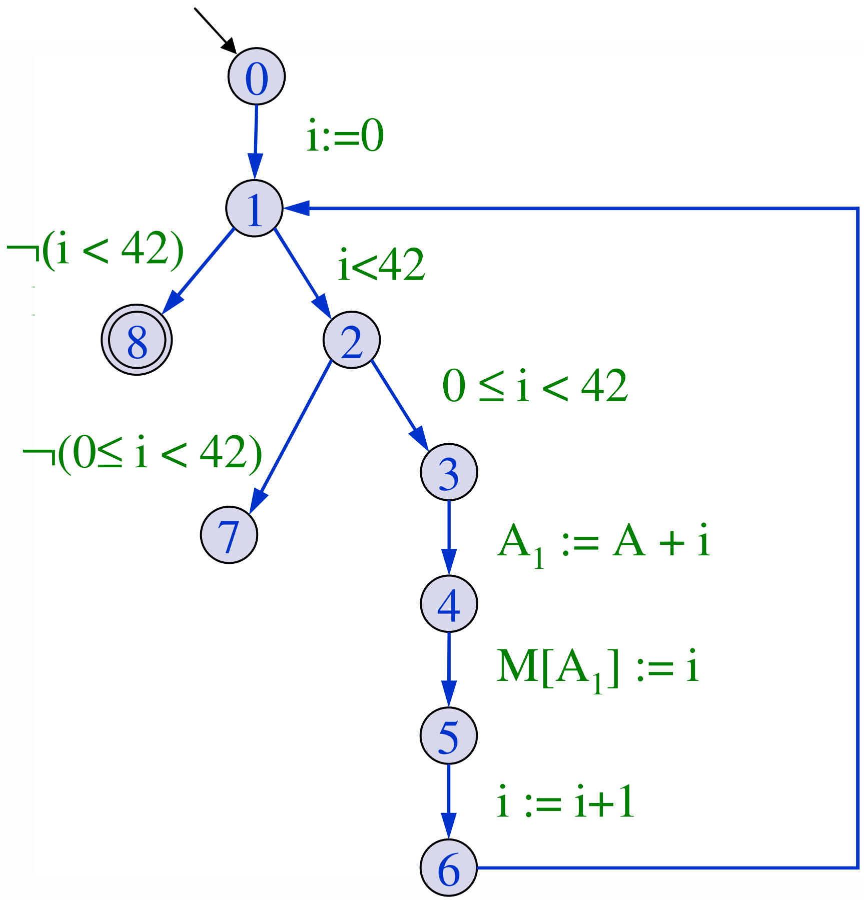
	1		2		3	
	l	u	l	u	l	u
0	$-\infty$	$+\infty$	$-\infty$	$+\infty$	dito	
1	0	0	0	$+\infty$		
2	0	0	0	$+\infty$		
3	0	0	0	$+\infty$		
4	0	0	0	$+\infty$		
5	0	0	0	$+\infty$		
6	1	1	1	$+\infty$		
7	$\perp$		42	$+\infty$		
8	$\perp$		42	$+\infty$		

$\Rightarrow$ 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.

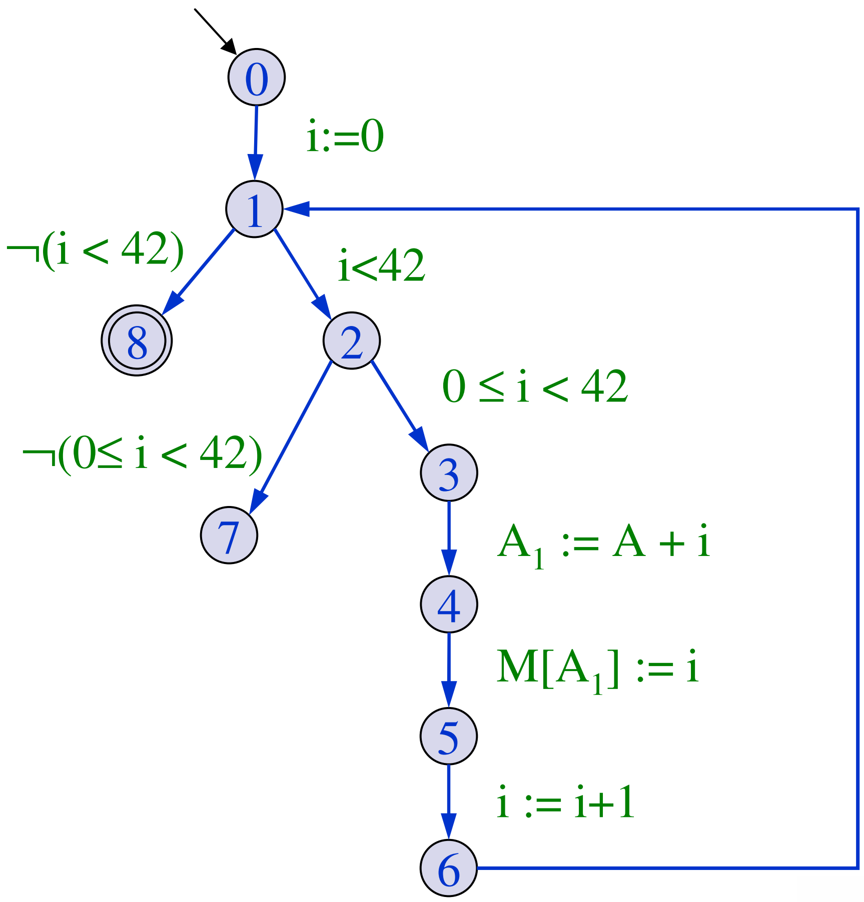


	1		2		3	
	l	u	l	u	l	u
0	$-\infty$	$+\infty$	$-\infty$	$+\infty$	dito	
1	0	0	0	$+\infty$		
2	0	0	0	41		
3	0	0	0	41		
4	0	0	0	41		
5	0	0	0	41		
6	1	1	1	42		
7	$\perp$		$\perp$			
8	$\perp$		42	$+\infty$		

$\Rightarrow$ Identify condition at edge from 2 to 3 as redundant !
Find out, prg. point 7 is unreachable ! ☺

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 (later!)
- Narrowing operators

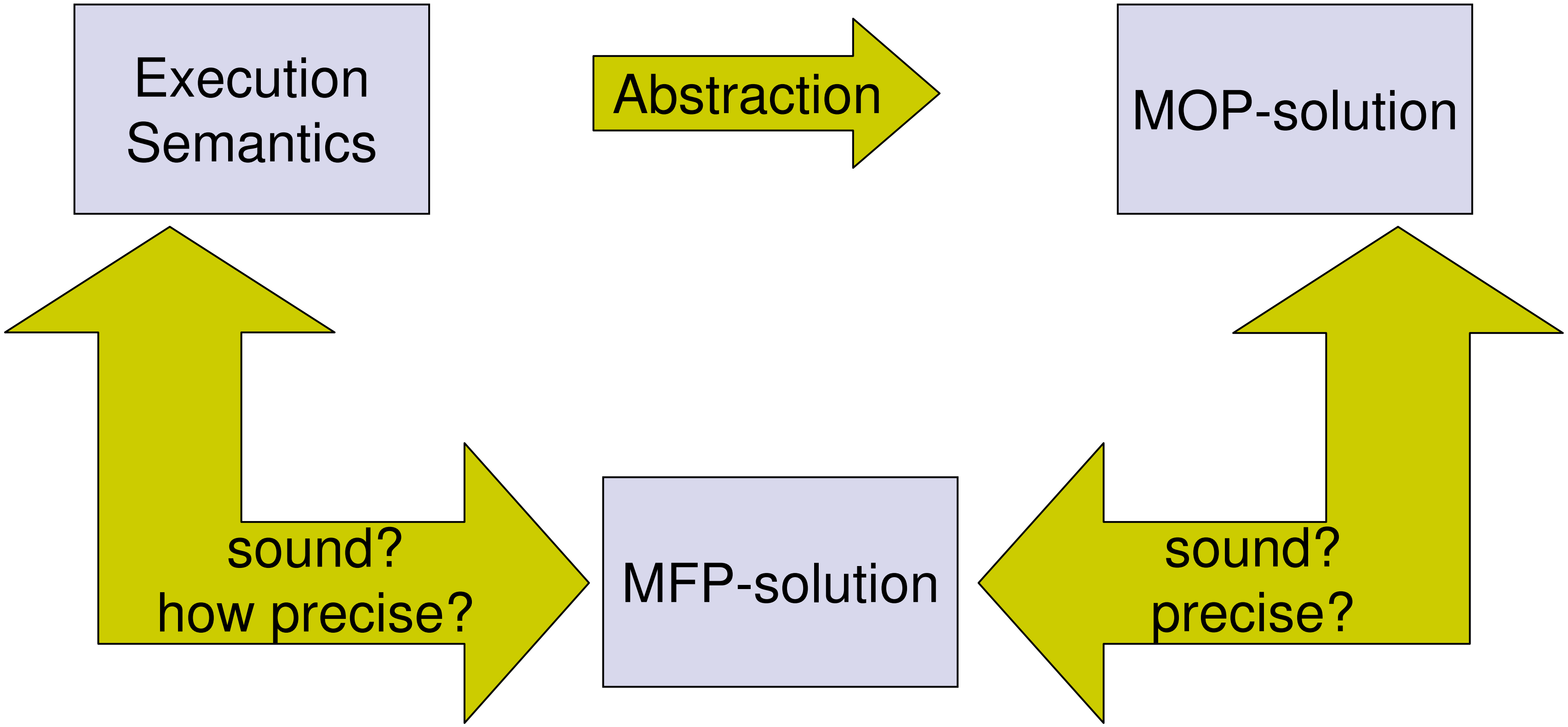
Three Questions

- 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

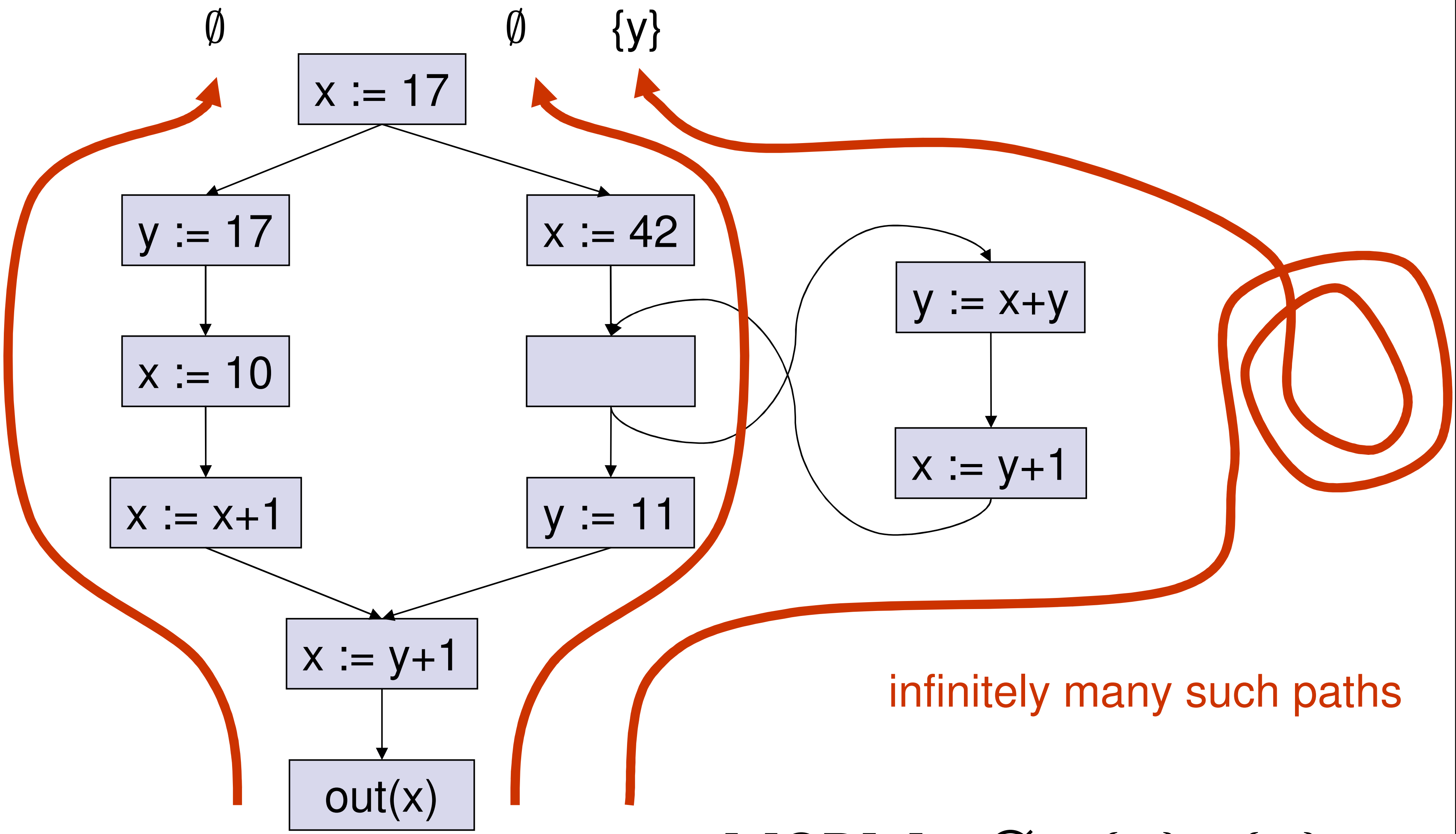
Three Questions

- 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



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

Meet-Over-All-Paths Solution (MOP)

Definition:

The ***transfer function $f_\pi : L \to L$ of a path*** π: $v_0 f_0 \ldots f_{k-1} v_k$, $k \geq 0$, is:

$$f_\pi = f_{k-1} \circ \ldots \circ f_0 \,.$$

The ***MOP-solution*** is:

$$\mathrm{MOP}[v] = \bigsqcup \; \{ f_\pi(\mathrm{init}) \mid \pi \in \mathrm{Paths}[st,v] \} \;\; \text{für alle } v \in N.$$

Coincidence Theorem

Definition:

A data-flow problem is positively-distributive if

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

Theorem:

For any instance of a positively-distributive data-flow problem:

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

(if all program points reachable).

Remark:

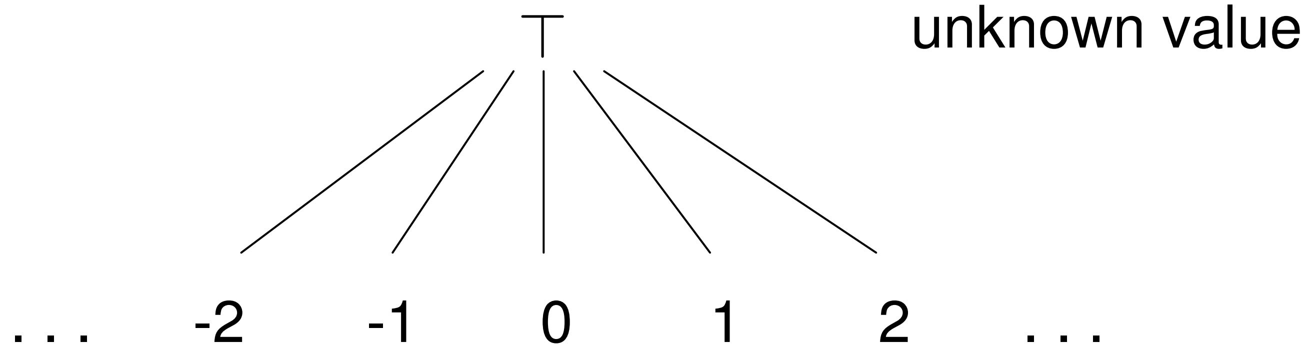
A data-flow problem is positively-distributive if a) and b) hold:

(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: the lattice L does not have infinite ascending chains.

Remark: All bitvector frameworks are distributive and effective.

Recall: Lattice for Constant Propagation

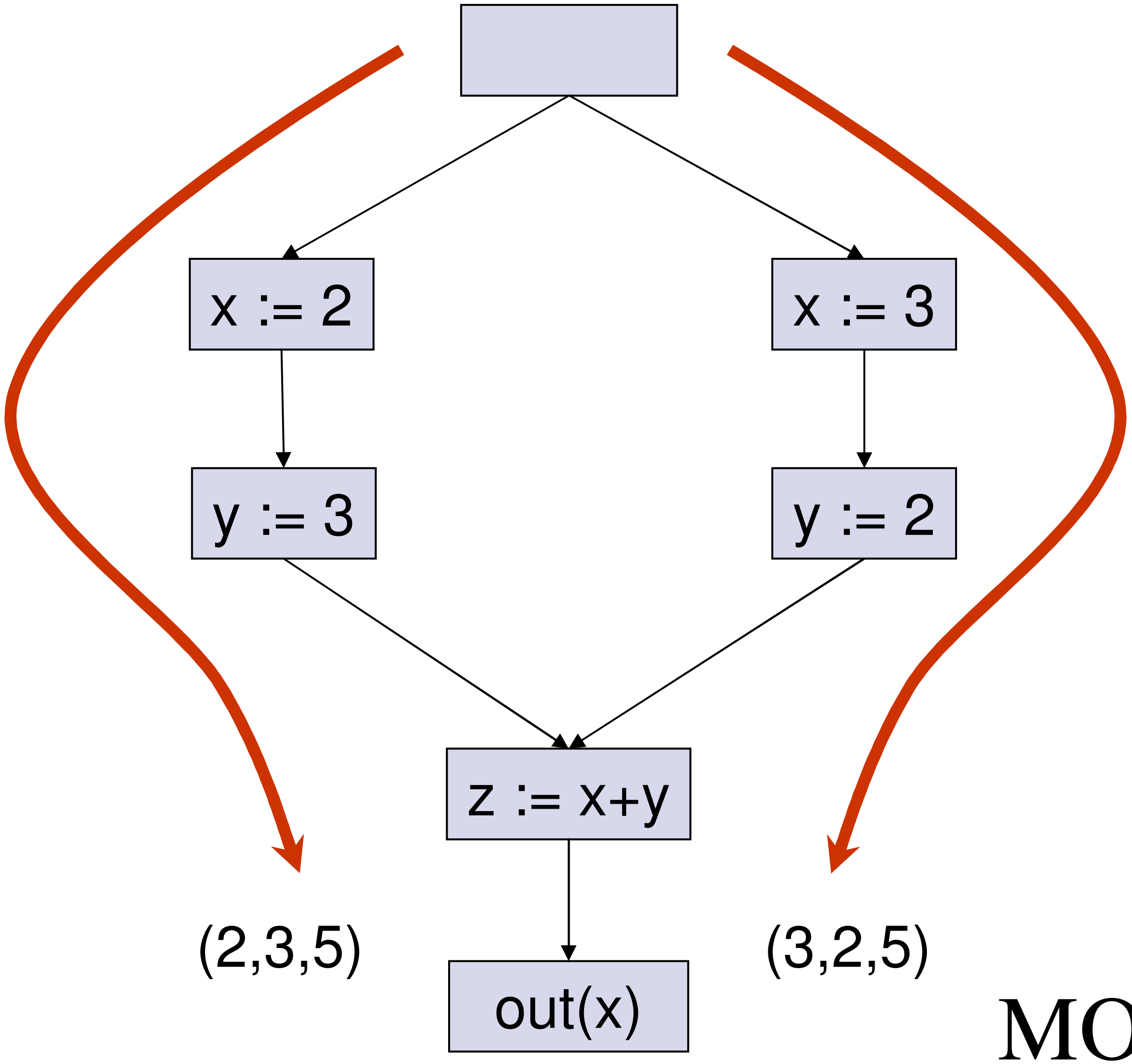


$$\text{lattice } L: \quad \{\rho \mid \rho : \mathrm{Var} \to (\mathbb{Z} \cup \{\top\})\} \cup \{\bot\}$$

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

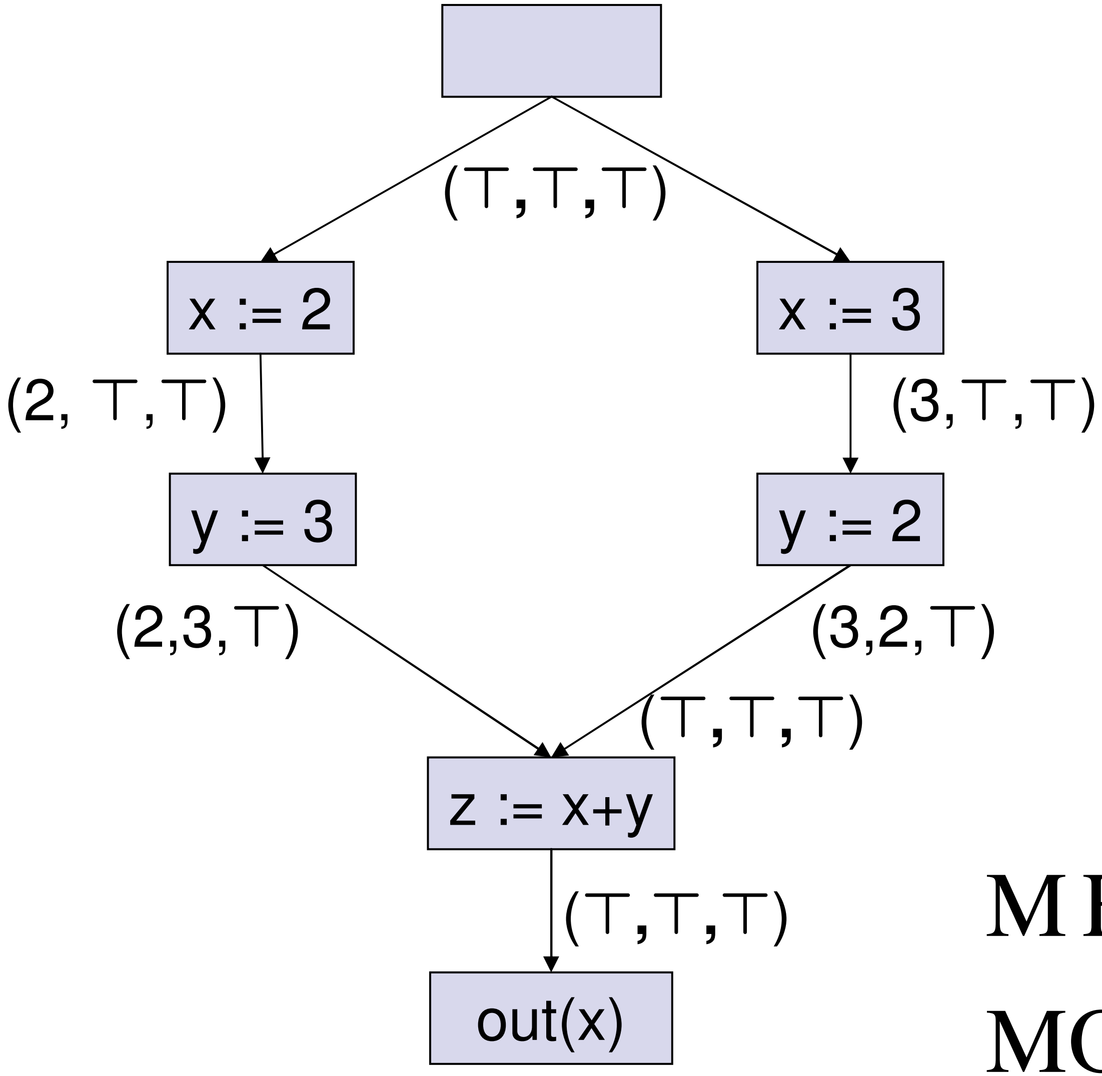
$$(\rho, \rho' \neq \bot \wedge \forall x : \rho(x) \sqsubseteq \rho'(x))$$

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



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

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



$$MFP[v] = (\top, \top, \top)$$

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

Correctness Theorem

Recall:

We assume transfer functions in a data-flow problem to be monotone i.e.:

$$x \sqsubseteq y \quad \Rightarrow \quad f(x) \sqsubseteq f(y)$$ for all $f \in F$, $x,y \in L$.

☺

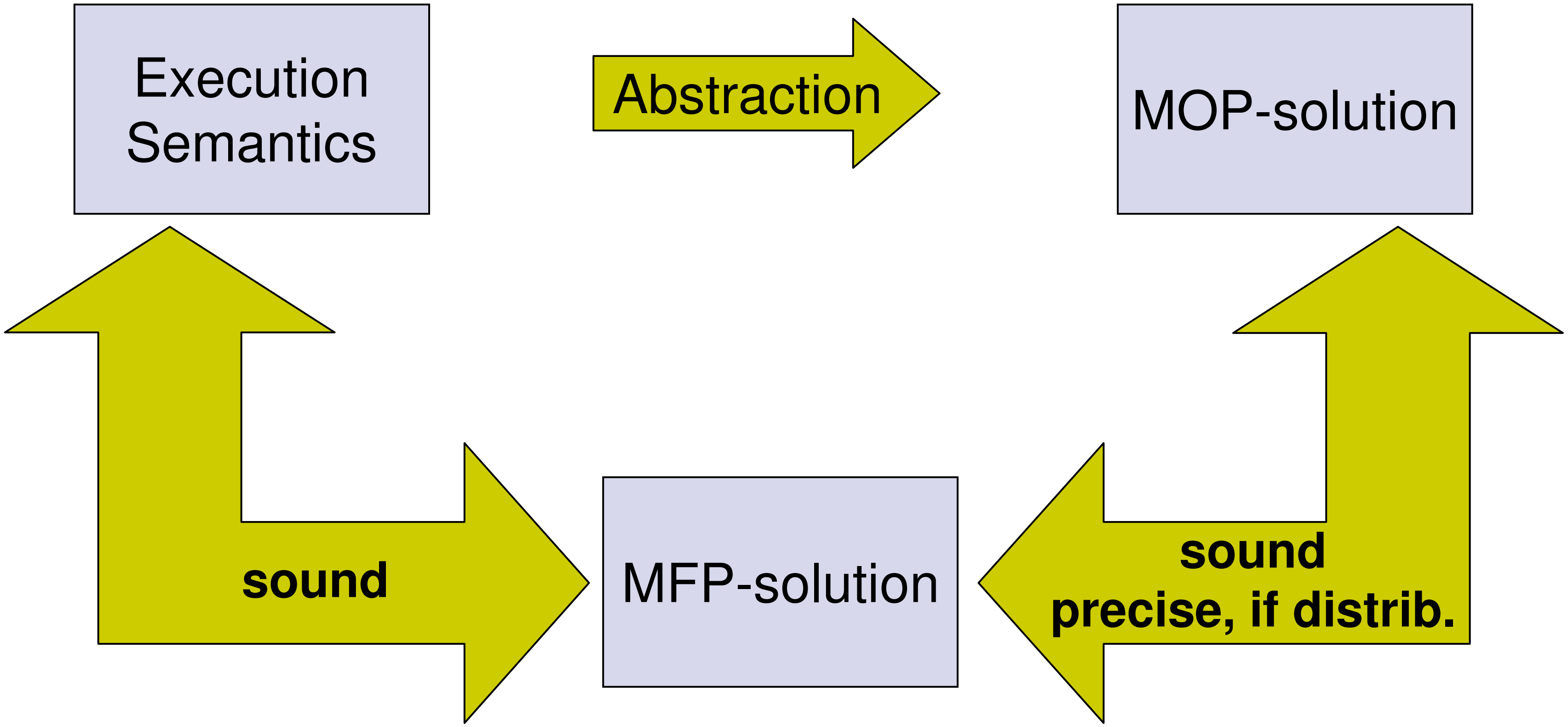
.

Theorem:

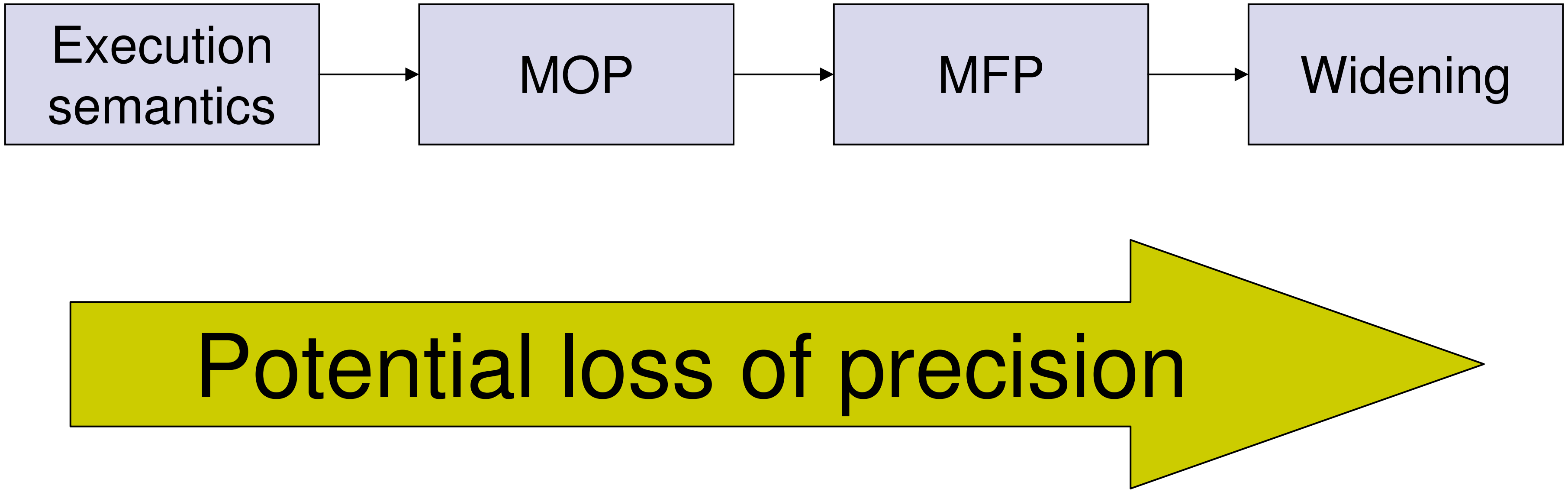
For any data-flow problem:

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

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 $(D,\sqsubseteq)$

MFP

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

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

$MFP^{\#}$

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 $\alpha : D \rightarrow D^\#$.

Correct abstract interpretation:

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

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

Correct and precise abstract interpretation:

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

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

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

If all transfer function f_e are universally-distributive, i.e.,

$f(\sqcup X) = \sqcup\{ f(x) \mid x \in X\}$ for all sets $X \subseteq L$

the abstraction is correct and precise, hence:

$\alpha_{MOP}(\underline{R}[u]) = \underline{R}^\#[u]$ 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
- Conclusion

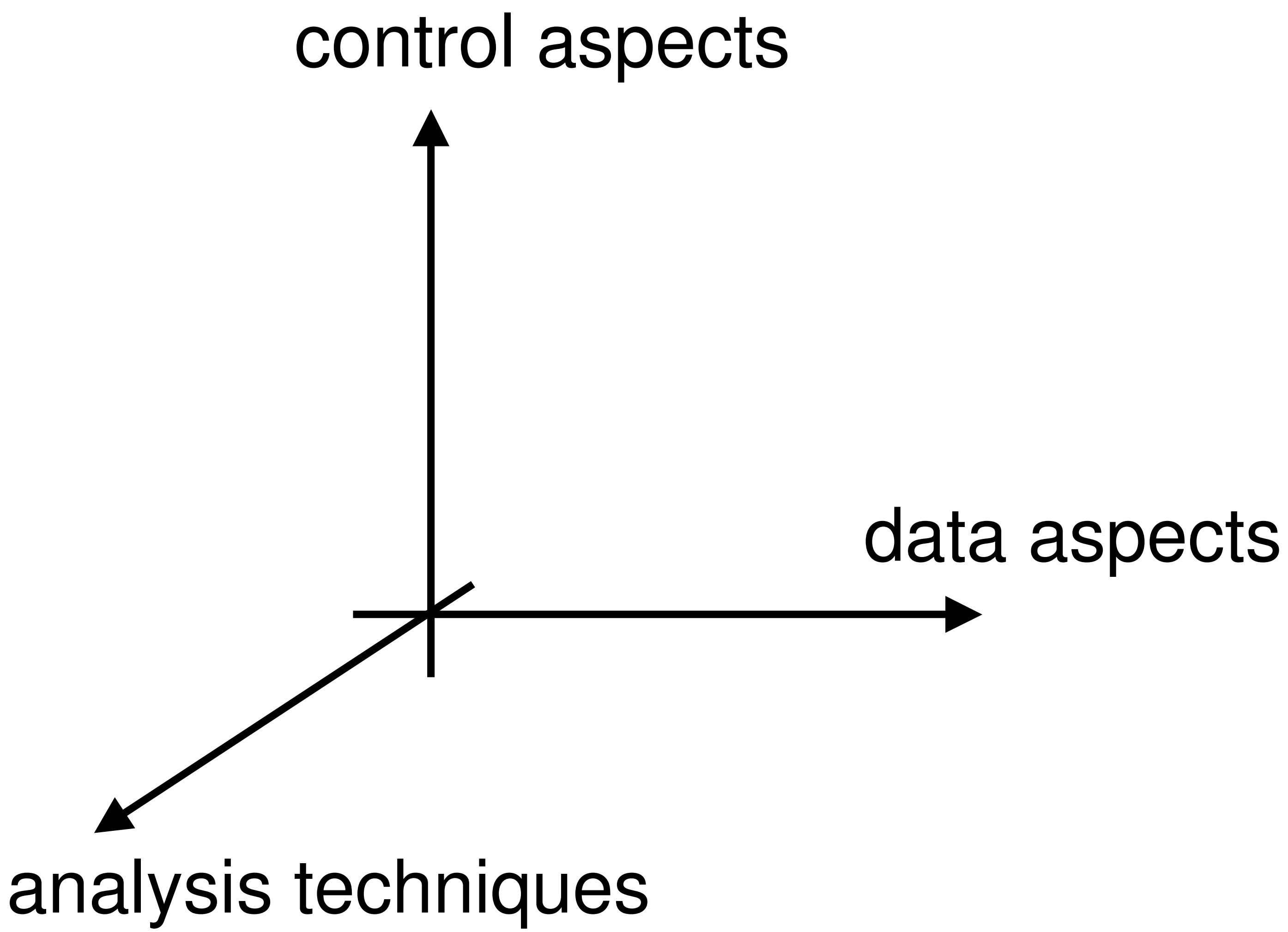
Challenges for Automatic Analysis

- 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
- Conclusion

A Note on Karr´s Algorithm

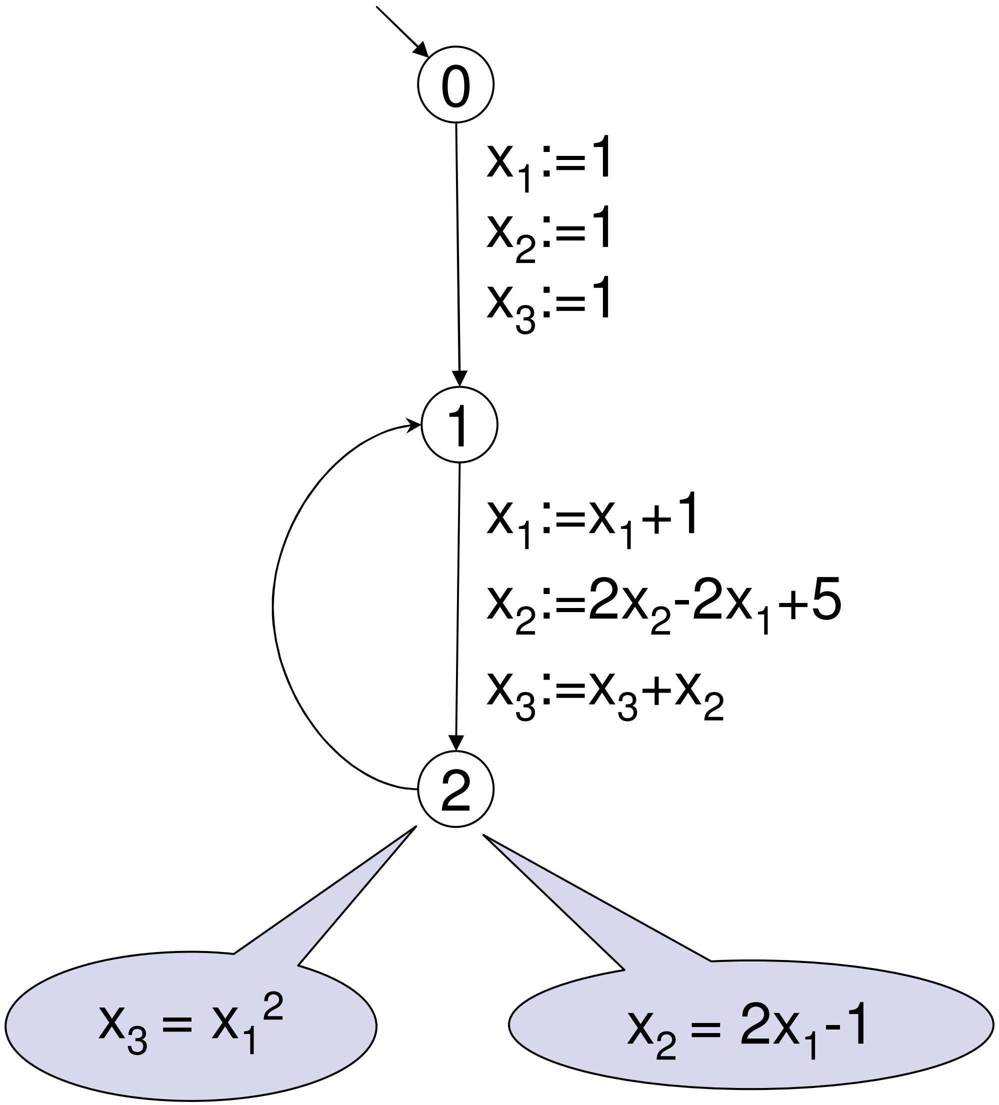
Markus Müller-Olm

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
    - $E \subseteq N \times Stmt \times N$ 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:

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

$5x_1+7x_2-42=0$

More ambitious goal:

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

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

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

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)

- RS3:
  - Improve precision of PDG-based IFC analysis
    (with Gregor Snelting (KIT, Karlsruhe) and his group)

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

- Number may grow to exponential size

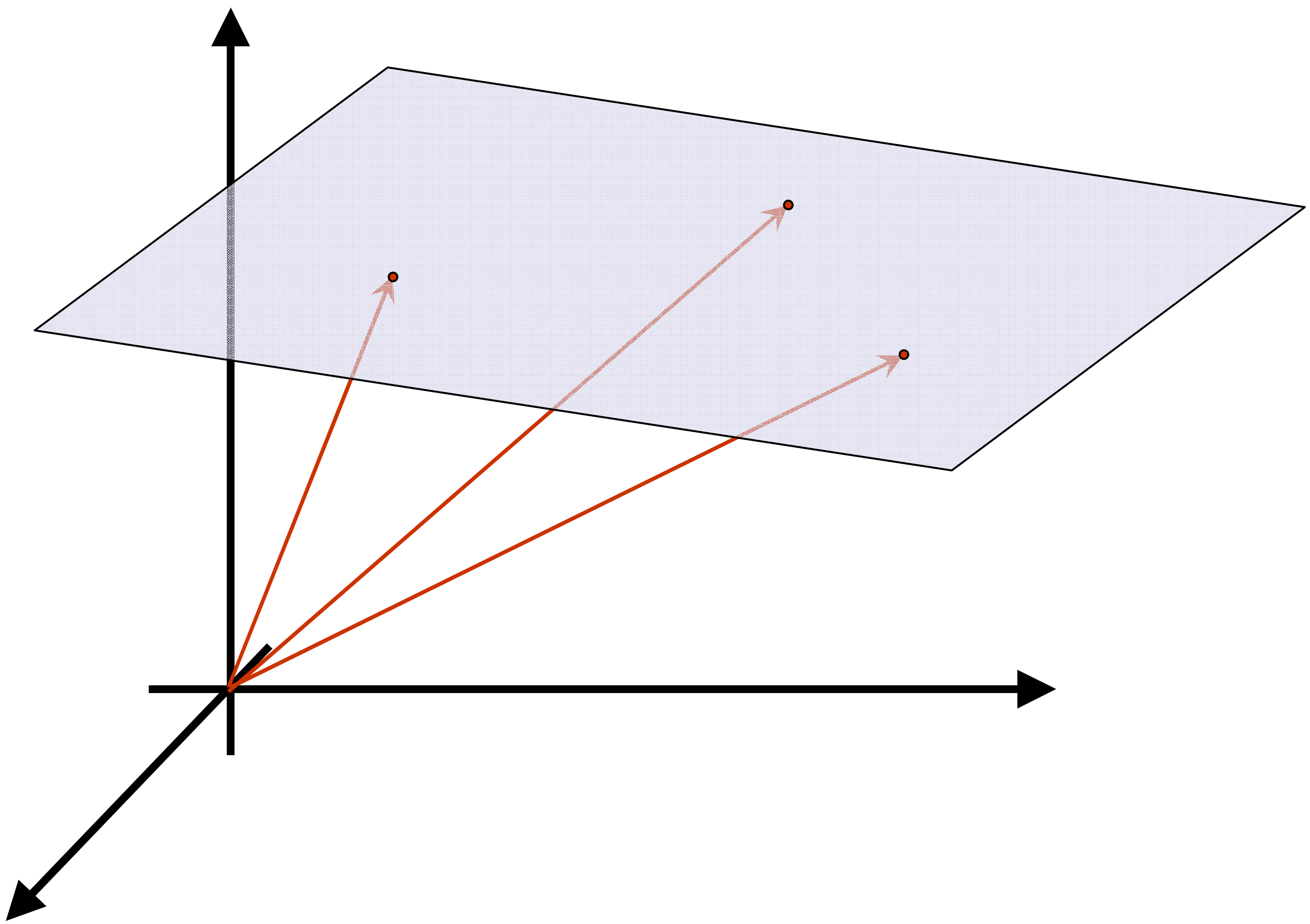
Our Contribution

- 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:

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

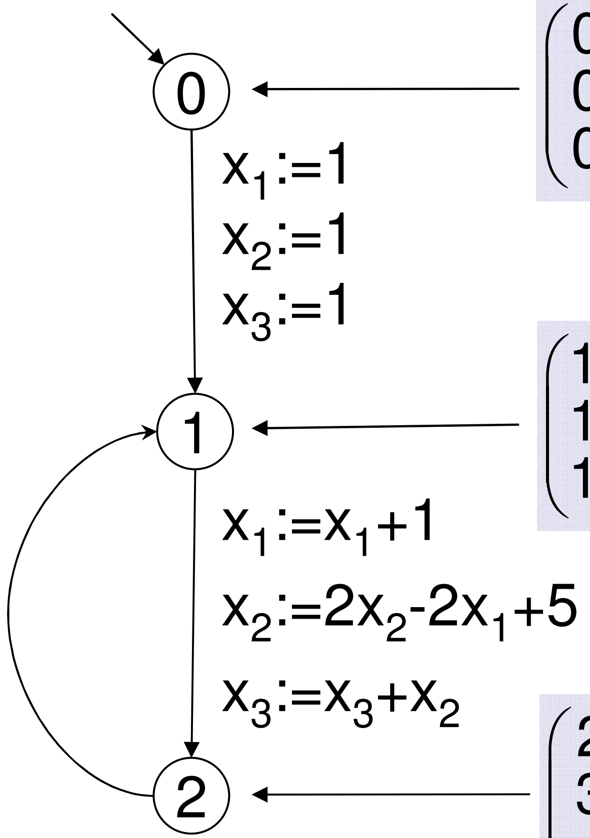
$$V^{\#}[v] \sqsupseteq f_s(V^{\#}[u]) \,, \quad \text{for each edge } (u,s,v)$$

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



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

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

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

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:

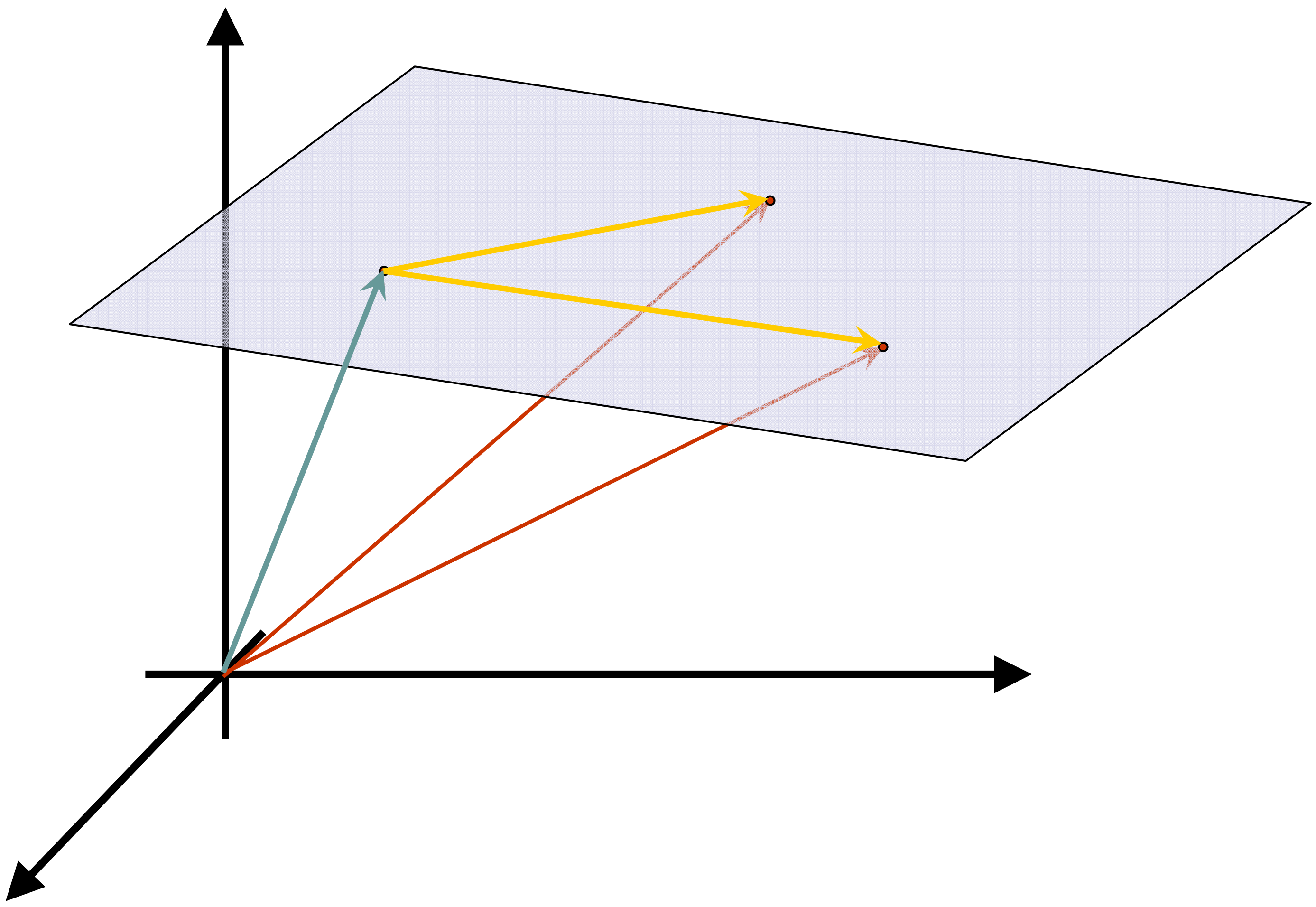
a) The affine hulls $V^{\#}[u] = \mathit{aff}(V[u])$ can be computed in time $O(n \cdot k^3)$, where $n = |N| + |E|$.

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

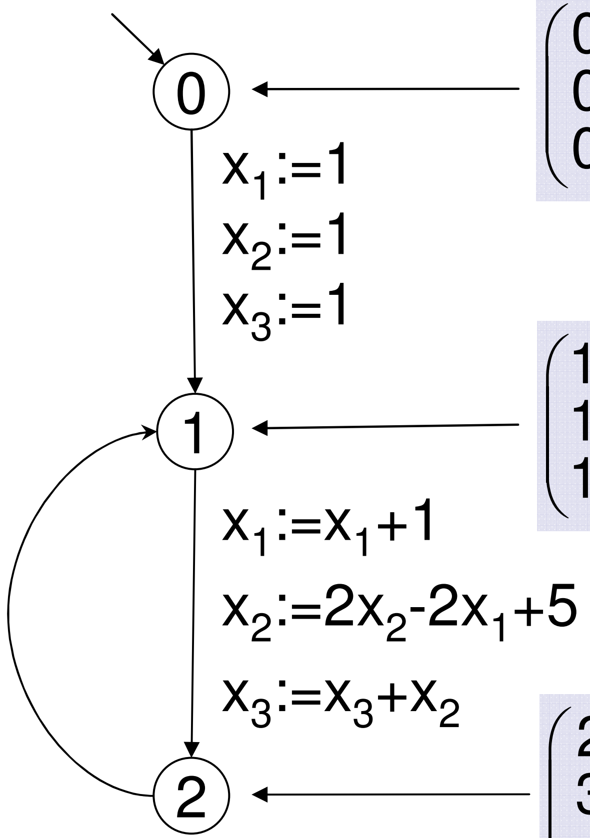
Store diagonal basis for membership tests.

Propagate original vectors.

Point + Linear Basis



Example



State 0:

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

State 1:

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

State 2:

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

Determining Affine Relations

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

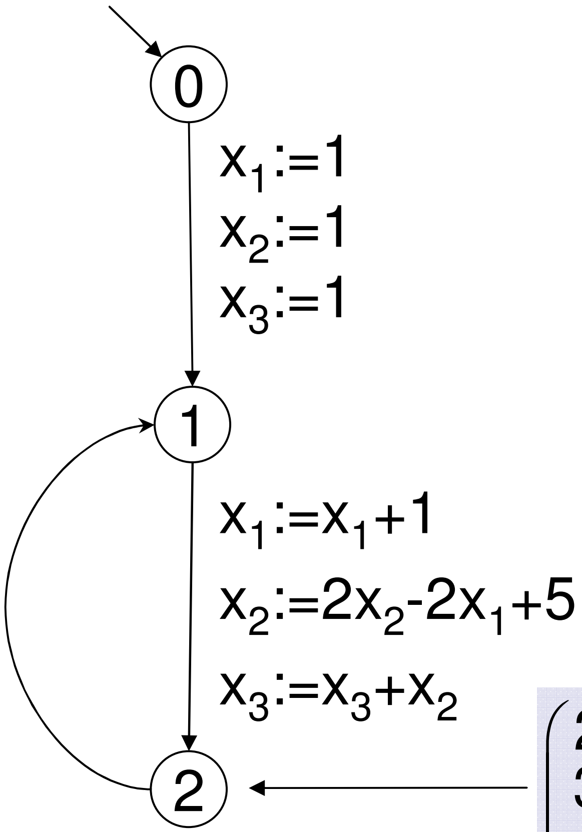
$\Rightarrow$ suffices to determine the affine relations valid for affine bases; can be done with a linear equation system!

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

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

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

$\Rightarrow$ $2x_1 - x_2 - 1$ is valid at 2

Also in the Paper

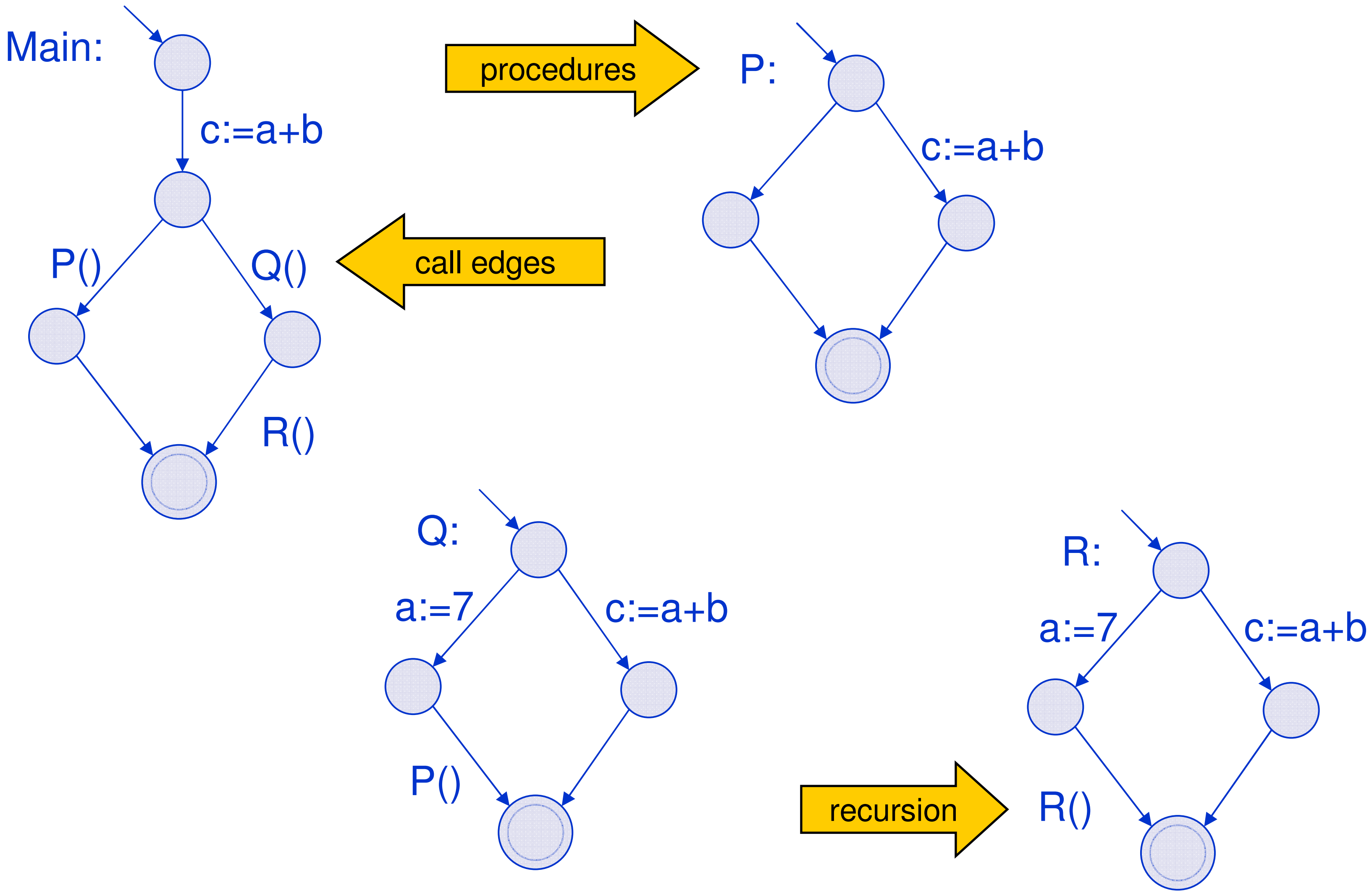
- Non-deterministic assignments
- Bit length estimation
- Polynomial relations
- Affine programs + affine equality guards
  - validity of affine relations undecidable

End of Excursion 1

Overview

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

Interprocedural Analysis



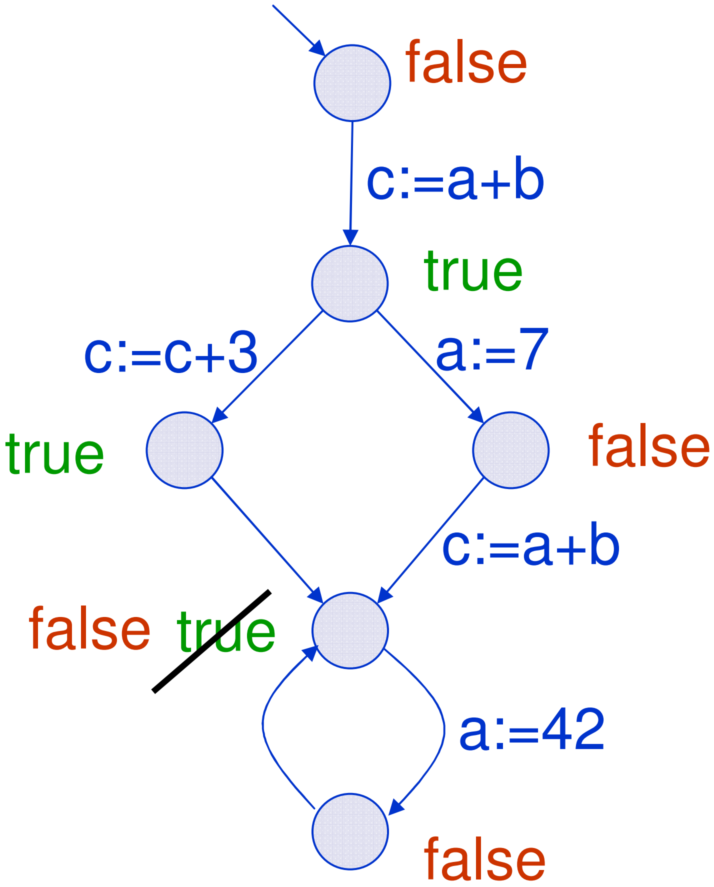
Running Example:
(Definite) Availability of the single expression a+b

The lattice:

false — a+b not available

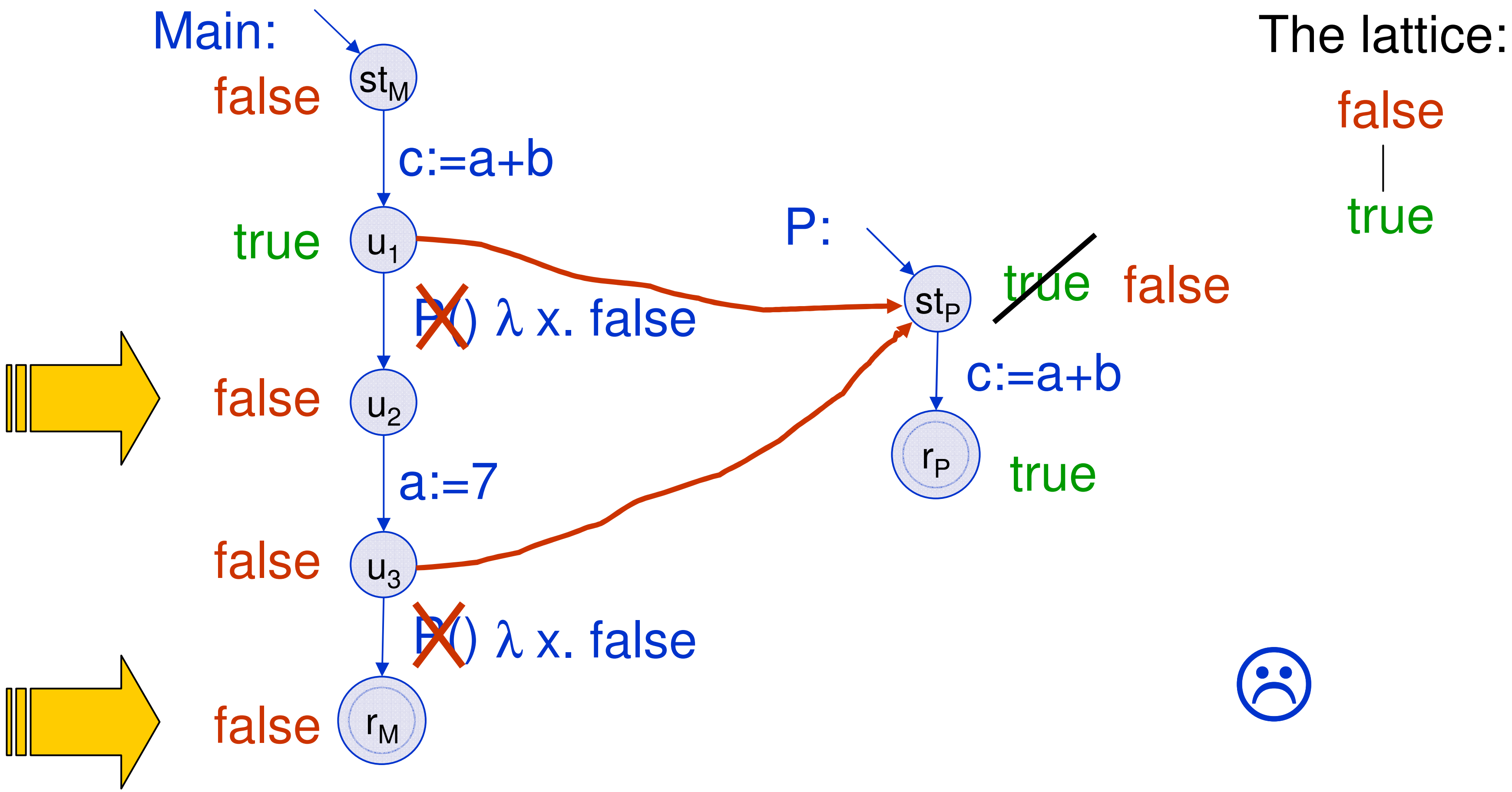
true — a+b available

Initial value: false



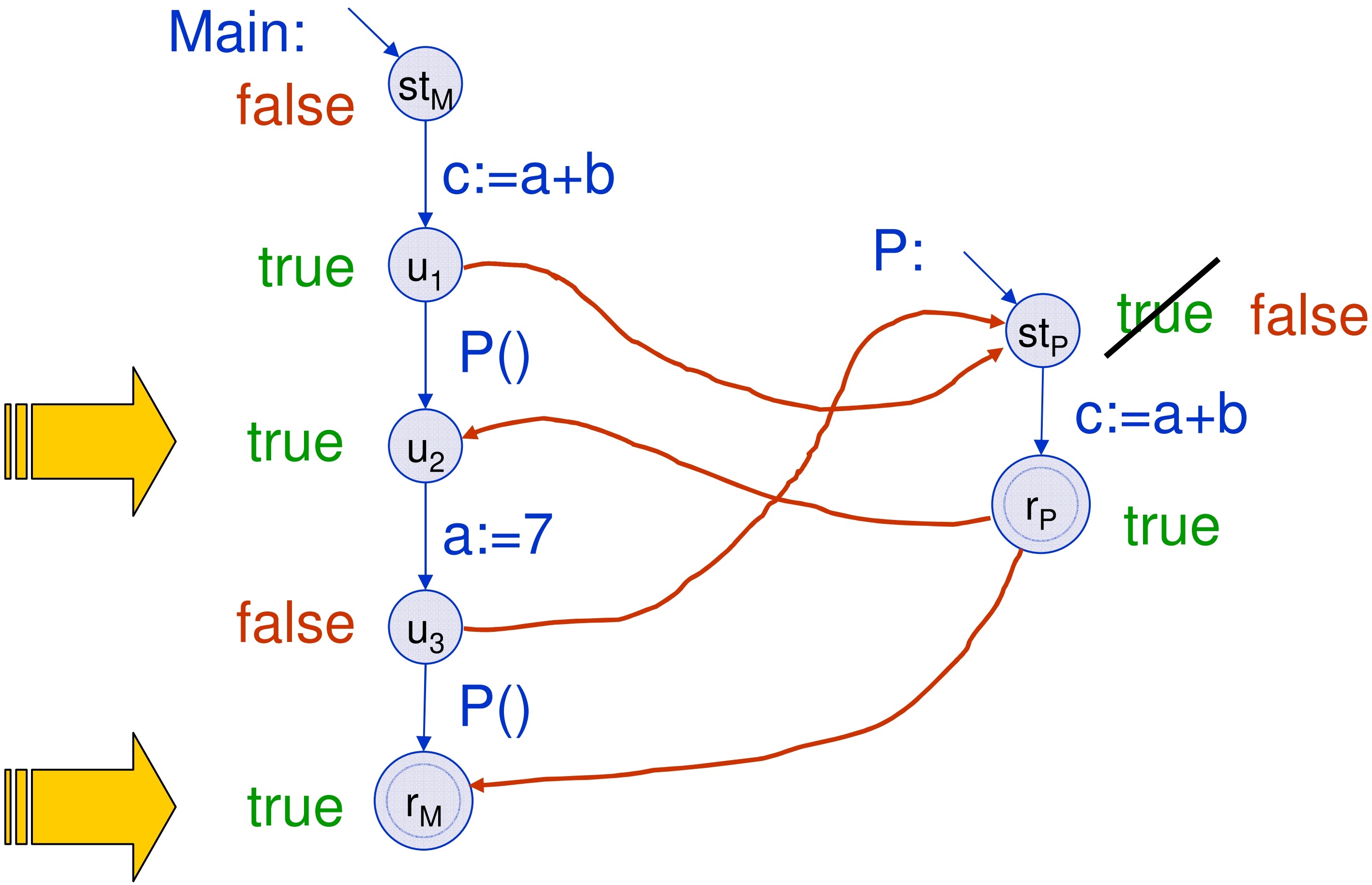
Intra-Procedural-Like Analysis

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



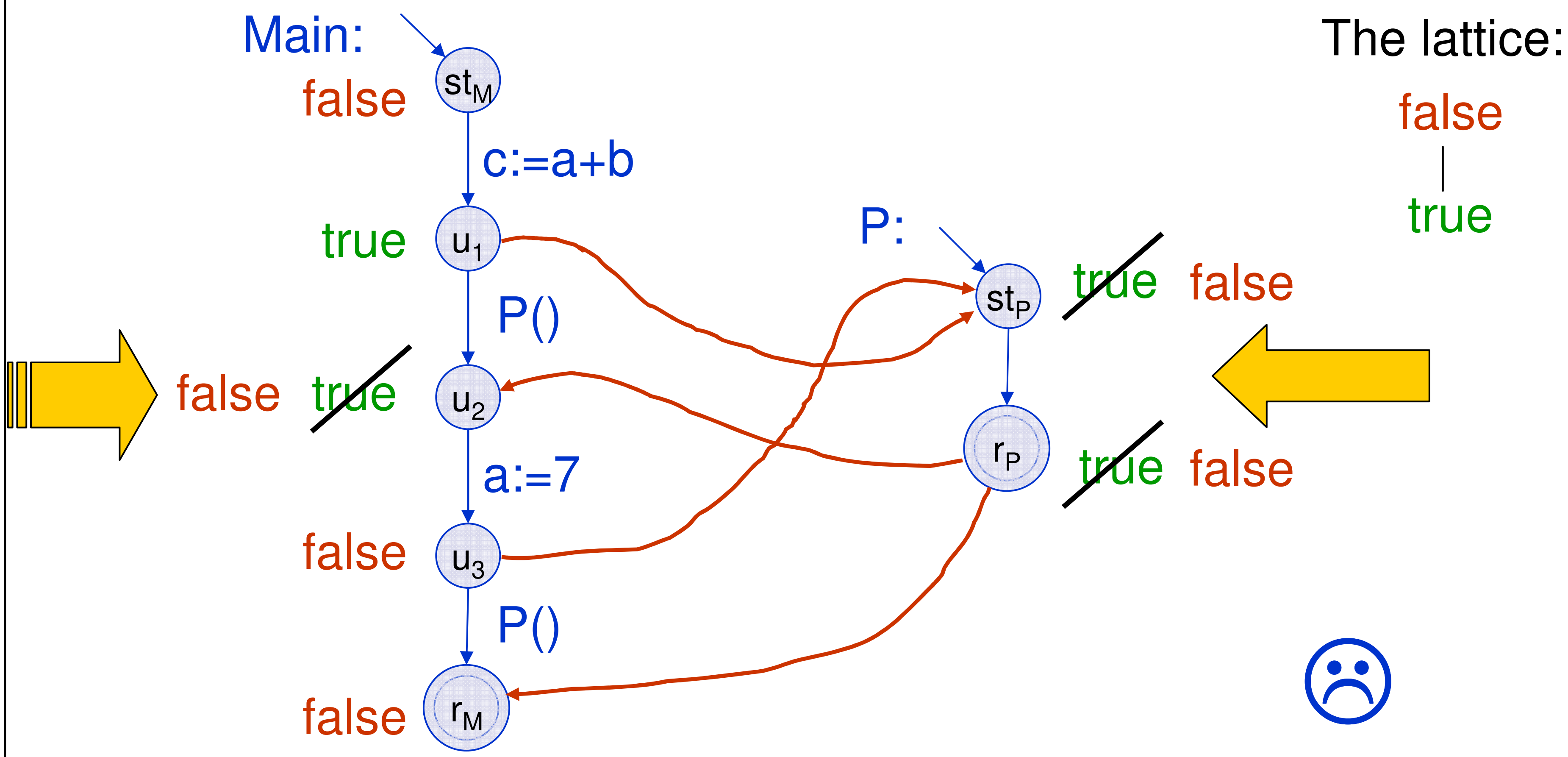
The lattice:

false
|
true

☺

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 $(D,\sqsubseteq)$

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

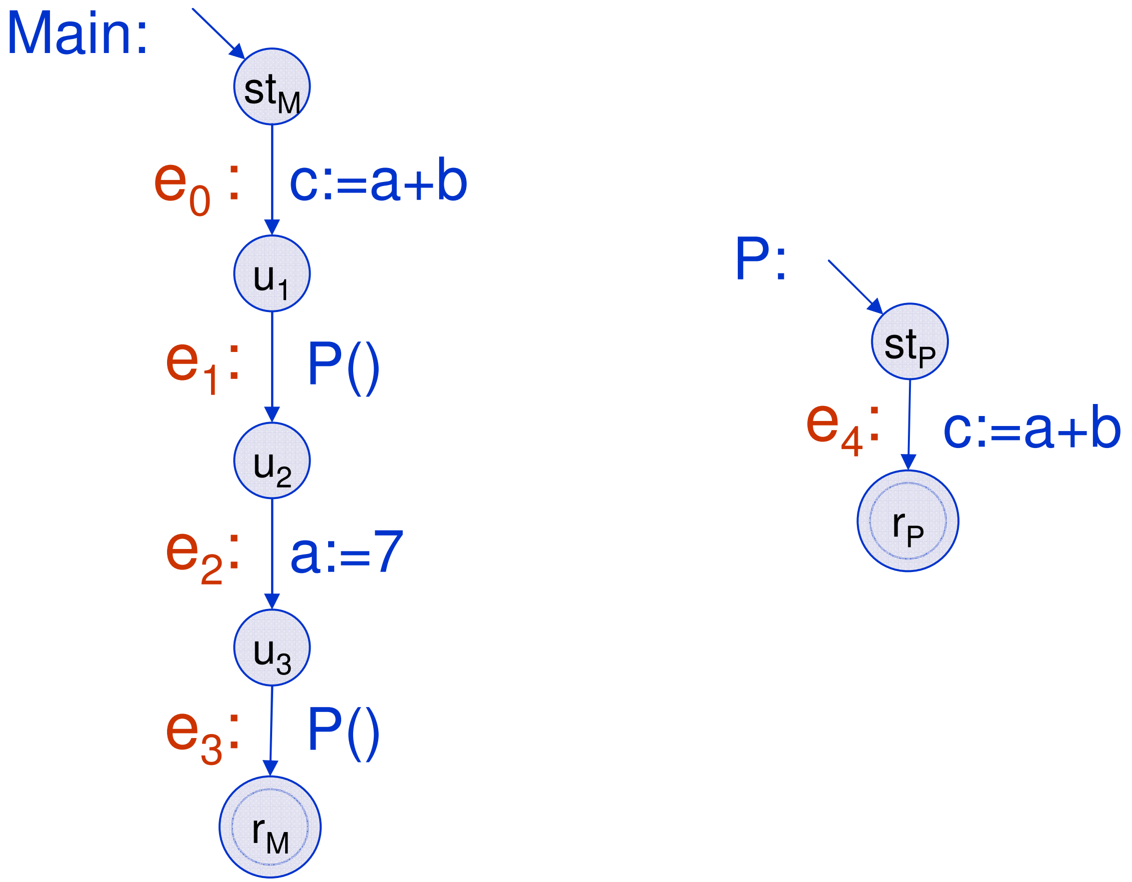
Correct and precise abstract interpretation:

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 !

Example Flow Graph



The lattice:

false

|

true

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

Context-Sensitive Analysis

Summary-based approaches:

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 types of summary information:

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

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

Call-string-based approaches:

e.g [Sharir/Pnueli 81], [Khedker/Karkare: CC´08]

- Analysis relative to finite portion of call stack
- Applicable to arbitrary lattices
- Sometimes less precise than summary-based approaches

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 for Availability of Single Expression Problem

Observations:

Just three montone functions on lattice L:

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

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

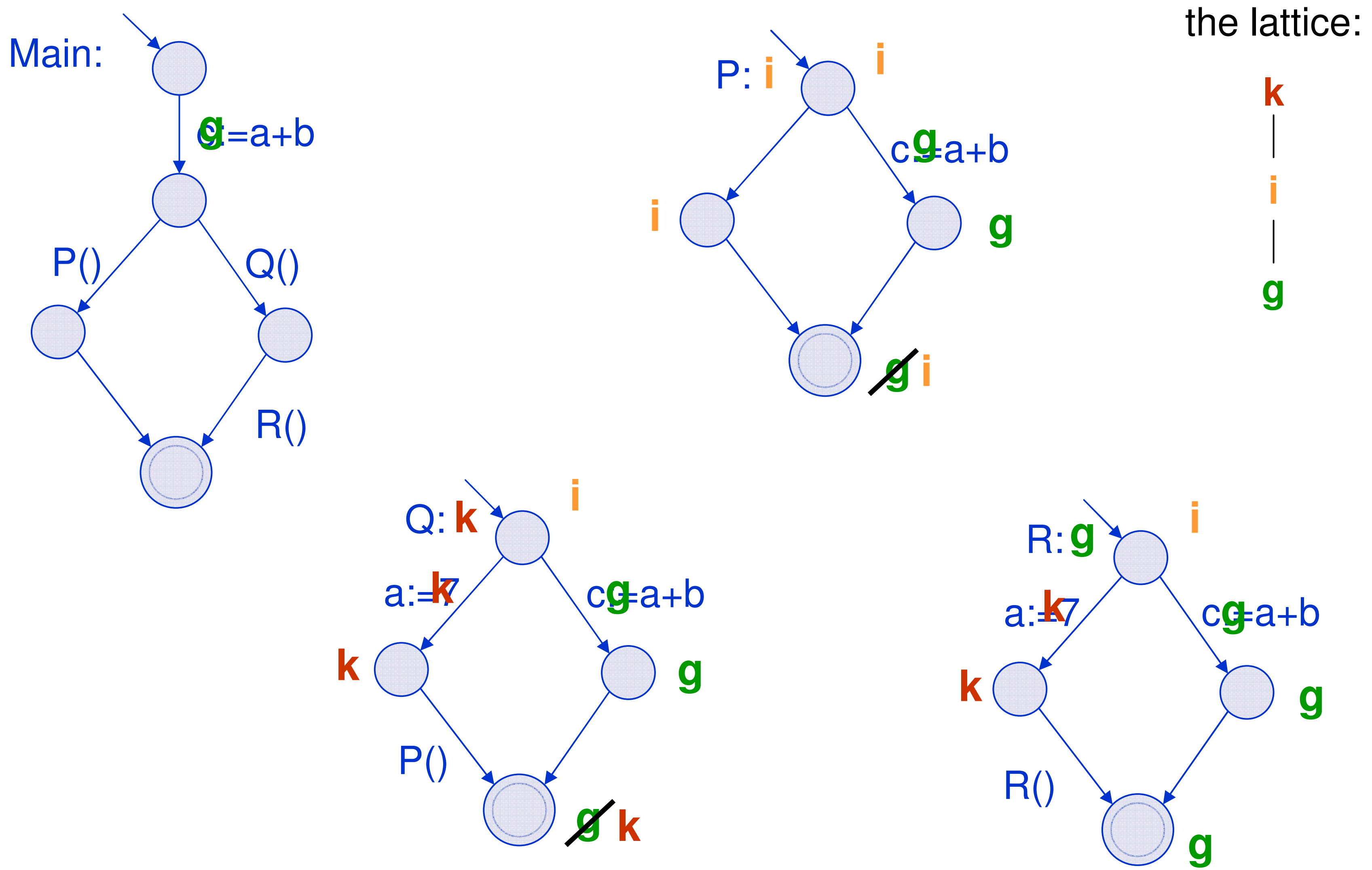
Functional composition of two such functions $f,g : L \to L$:

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

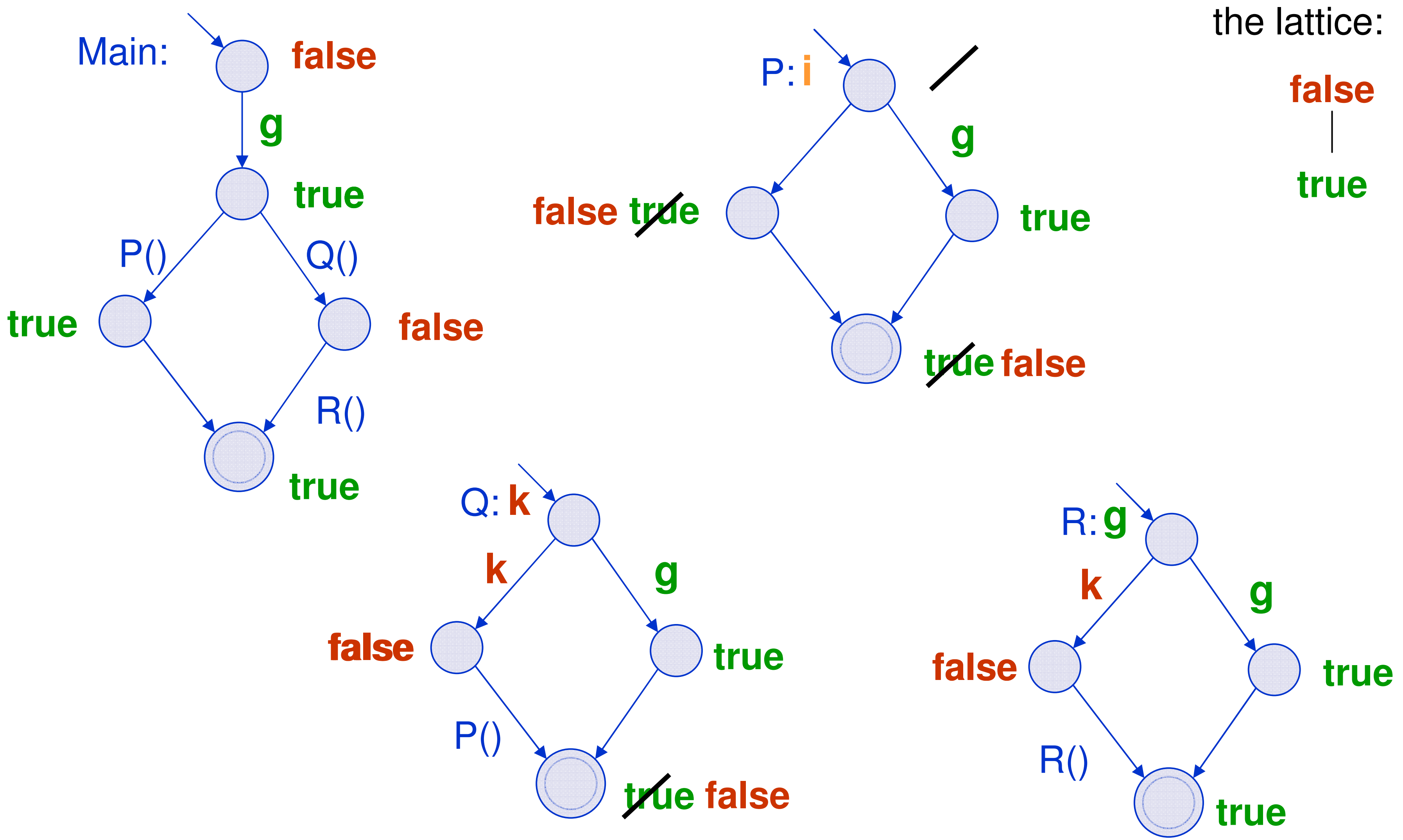
Analogous: precise interprocedural analysis for
all (separable) bitvector problems
in time linear in program size.

☺

Context-Sensitive Analysis, 1. Phase



Context-Sensitive Analysis, 2. Phase



Functional Approach

Theorem:

Correctness: For any monotone framework:

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

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 $|L| \cdot$ height(L) at each program point.

Overview

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

Excursion Mainly Based On ...

MMO + Helmut Seidl (TU München)

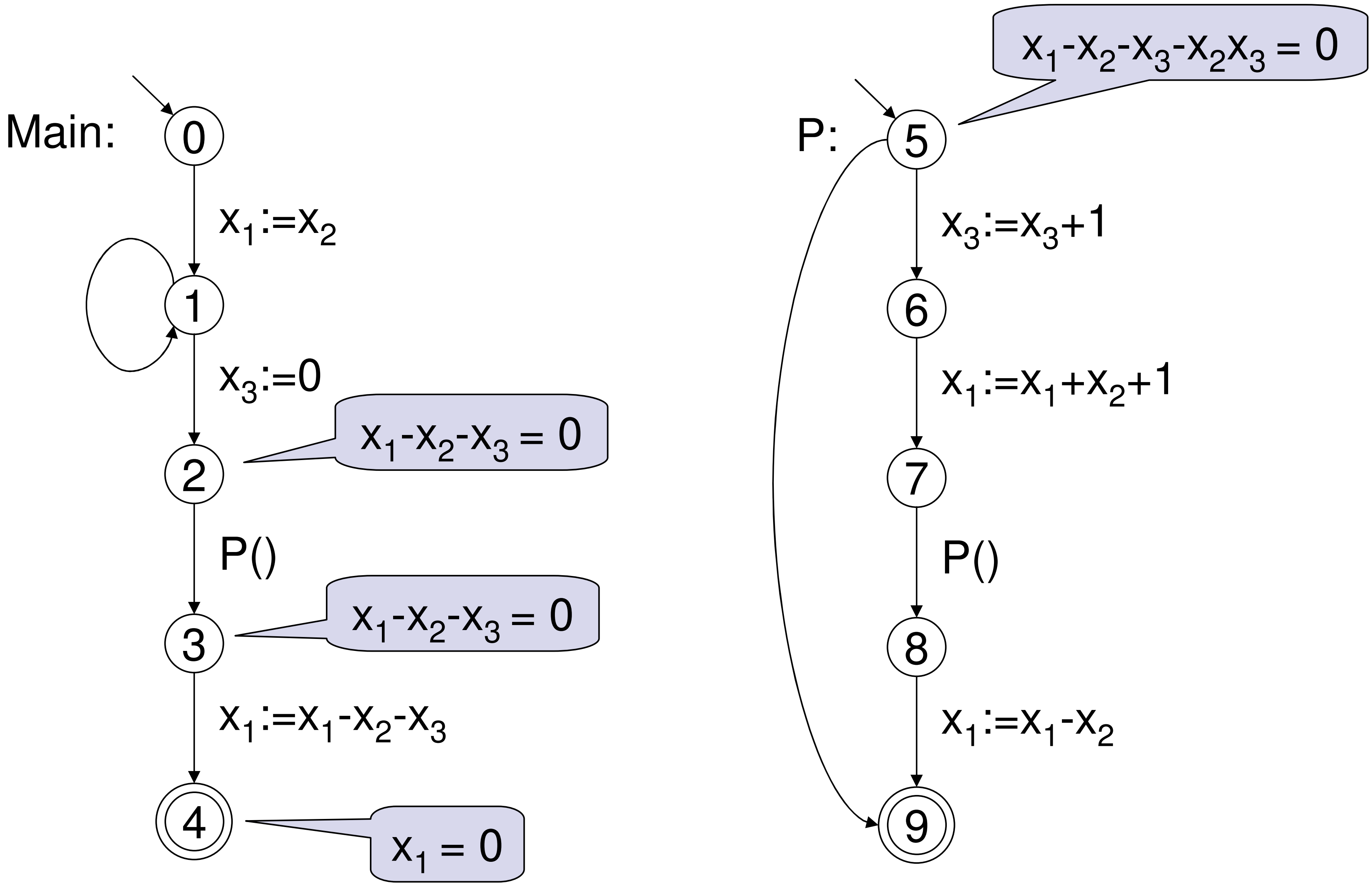
Precise Interprocedural Analysis through Linear Algebra, POPL 2004

MMO + Helmut Seidl (TU München)

Analysis of Modular Arithmetic

ESOP 2005 + TOPLAS 2007

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
- improving PDG-based IFC analysis (with G. Snelting´s group, KIT)
- ...

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:

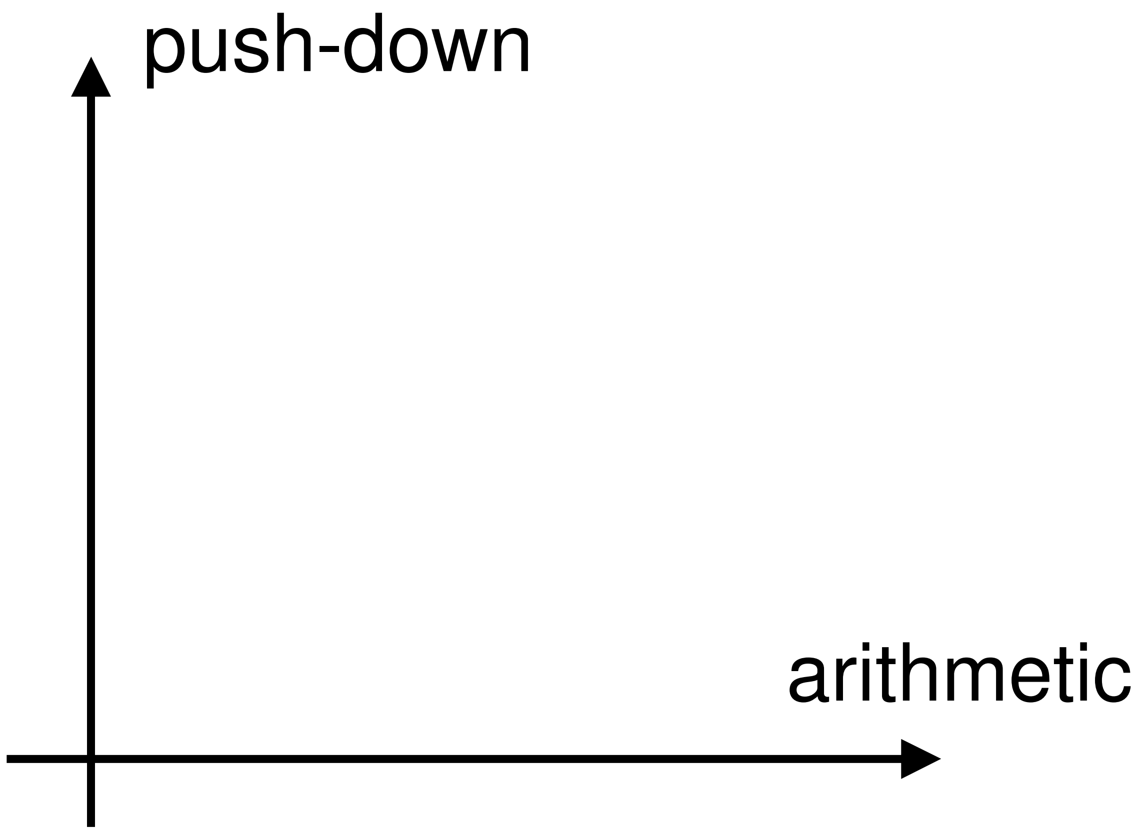
  $a_0 + \sum a_i x_i = 0$ $\quad$ $a_i \in R$ $\quad$ $5x+7y-42=0$

- 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‘s Algo ?

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

- Idea: summarize each procedure by function on data flow facts
- Problem: not applicable, lattice is infinite

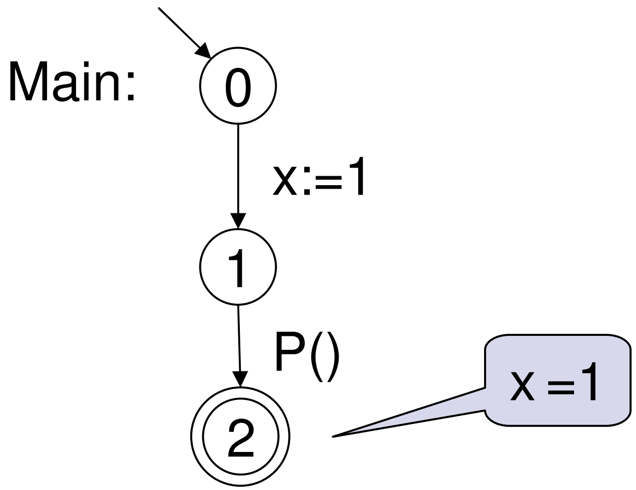
Relational approach [Cousot/Cousot, 1977]

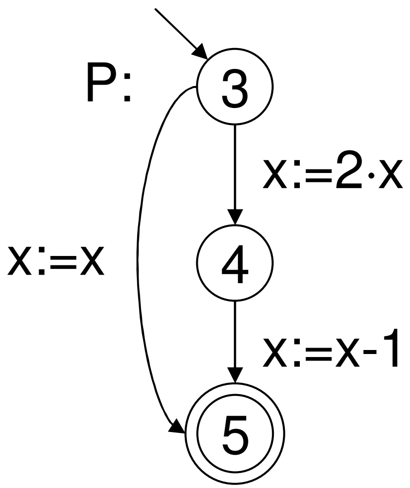
- Idea: summarize each procedure by approximation of I/O relation
- Problem: not exact

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

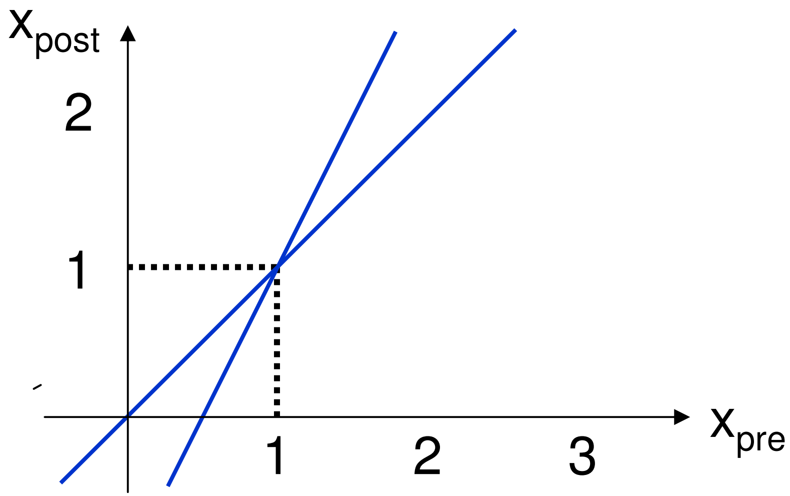
- Idea: take a finite piece of the run-time stack into account
- Problem: not exact

Relational Analysis is Not Strong Enough

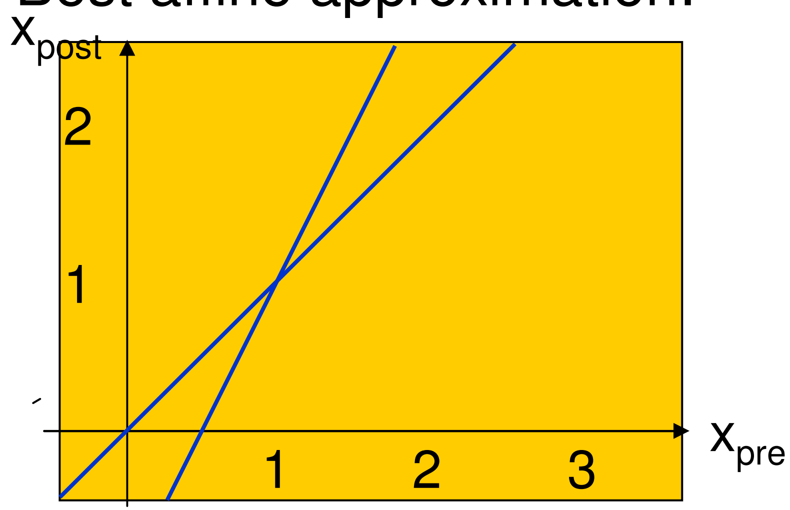




True relational semantics of P:



Best affine approximation:



Towards the Algorithm ...

Concrete Semantics of an Execution Path

- Every execution path π induces a
linear transformation on **extended program states**:

$$\llbracket \overbrace{x_1 := 2x_1 + 3x_2 + 4,\ \ x_2 := 5x_1 + 6}^{\pi} \rrbracket (v)$$

$$= \llbracket x_2 := 5x_1 + 6 \rrbracket \left(\llbracket x_1 := 2x_1 + 3x_2 + 4 \rrbracket (v) \right)$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 6 & 5 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 4 & 2 & 3 \\ 0 & 0 & 1 \end{pmatrix} (v)$$

$$= \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 4 & 2 & 3 \\ 26 & 10 & 15 \end{pmatrix}}_{\llbracket \pi \rrbracket \text{, the transformation matrix for path } \pi} (v) \qquad \text{with } v = \begin{pmatrix} 1 \\ v_1 \\ v_2 \end{pmatrix}$$

Observation 1

Extended states ...

- ... form a vector space of dimension k+1 = O(k).
- Subspaces form a lattice of height k+1 = O(k)

Transformation matrices ...

- ... form a vector space of dimension $k\cdot(k+1)+1 = O(k^2)$.
- Subspaces from a complete lattice of height $k\cdot(k+1)+1 = O(k^2)$.

Observation 2

Affine relation is valid for a set M of extended states

iff

it is valid for span M, i.e.:

$$a_0 + a_1 v_1 + \ldots + a_k v_k \qquad \text{for all} \begin{pmatrix} 1 \\ v_1 \\ v_2 \end{pmatrix} \in V$$

$$\Leftrightarrow \quad a_0 v_0 + a_1 v_1 + \ldots + a_k v_k \qquad \text{for all} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} \in span\ V$$

$\Rightarrow$ Suffices to compute **span of reaching states** !! (cf. Excursion 1)

Observation 3

$$
\begin{aligned}
& span\,\{Mv \mid M \in P,\ v \in V\} \\
= \quad & span\,\{Mv \mid M \in span\,P,\ v \in span\,V\}
\end{aligned}
$$

$\Rightarrow$ **Span of transformation matrices** of paths through procedure can be used as precise summary !!

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 (for each prg. point and procedure u) a basis *B* with:

   Span *B* = Span $\{[\![\pi]\!] \mid \pi \in S(u)\}$

   by a precise abstract interpretation.

   $\alpha_S(S(u))$

2) Compute (for each prg. point u) a basis *B* with:

   Span *B* = Span $\{[\![\pi]\!](v) \mid \pi \in R(u), v \in \mathbb{F}^{k+1}\}$

   by a precise abstract interpretation.

   $\alpha_R(R(u))$

3) Solve the linear equation system:

   $a_0\, v_0 + a_1\, v_1 + \ldots + a_k\, v_k = \mathbf{0}$ for all $(v_0,\ldots,v_k)^T \in R(u)$

Constraint Systems obtained by Abstract Interpretation

Same-level runs:

$S(p)$	$\sqsupseteq$	$S(r_p)$	r_p return point of p
$S(st_p)$	$\sqsupseteq$	$span\,\{\,I\,\}$	st_p entry point of p, I identity matrix
$S(v)$	$\sqsupseteq$	$\{[\![e]\!]\} \cdot S(u)$	$e=(u,s,v)$ base edge, $\cdot$ matrix product lifted to sets
$\mathrm{S(v)}$	$\sqsupseteq$	$S(u) \cdot S(p)$	$e=(u,p,v)$ call edge

Reaching runs:

$R(st_{Main})$	$\sqsupseteq$	$\mathbb{F}^{k+1}$	st_{Main} entry point of *Main*
$R(v)$	$\sqsupseteq$	$\{[\![e]\!]\,v \mid v \in R(u)\}$	$e=(u,s,v)$ basic edge
$R(v)$	$\sqsupseteq$	$\{Mv \mid M \in S(p),\ v \in R(u)\}$	$e=(u,p,v)$ call edge
$R(st_p)$	$\sqsupseteq$	$R(u)$	$e=(u,p,v)$ call edge, st_p entry point of p

All computations can be and are performed on bases !

Theorem

In an affine program:

- We can compute bases for the following vector spaces:

  $\alpha_S(S(u)) = \mathrm{Span}\ \{ [\![\pi]\!] \mid \pi \in S(u) \}$ for all u.

  $\alpha_R(R(u)) = \mathrm{Span}\ \{ [\![\pi]\!]\, v \mid \pi \in R(u),\ v \in \mathbb{F}^{k+1} \}$ for all u.

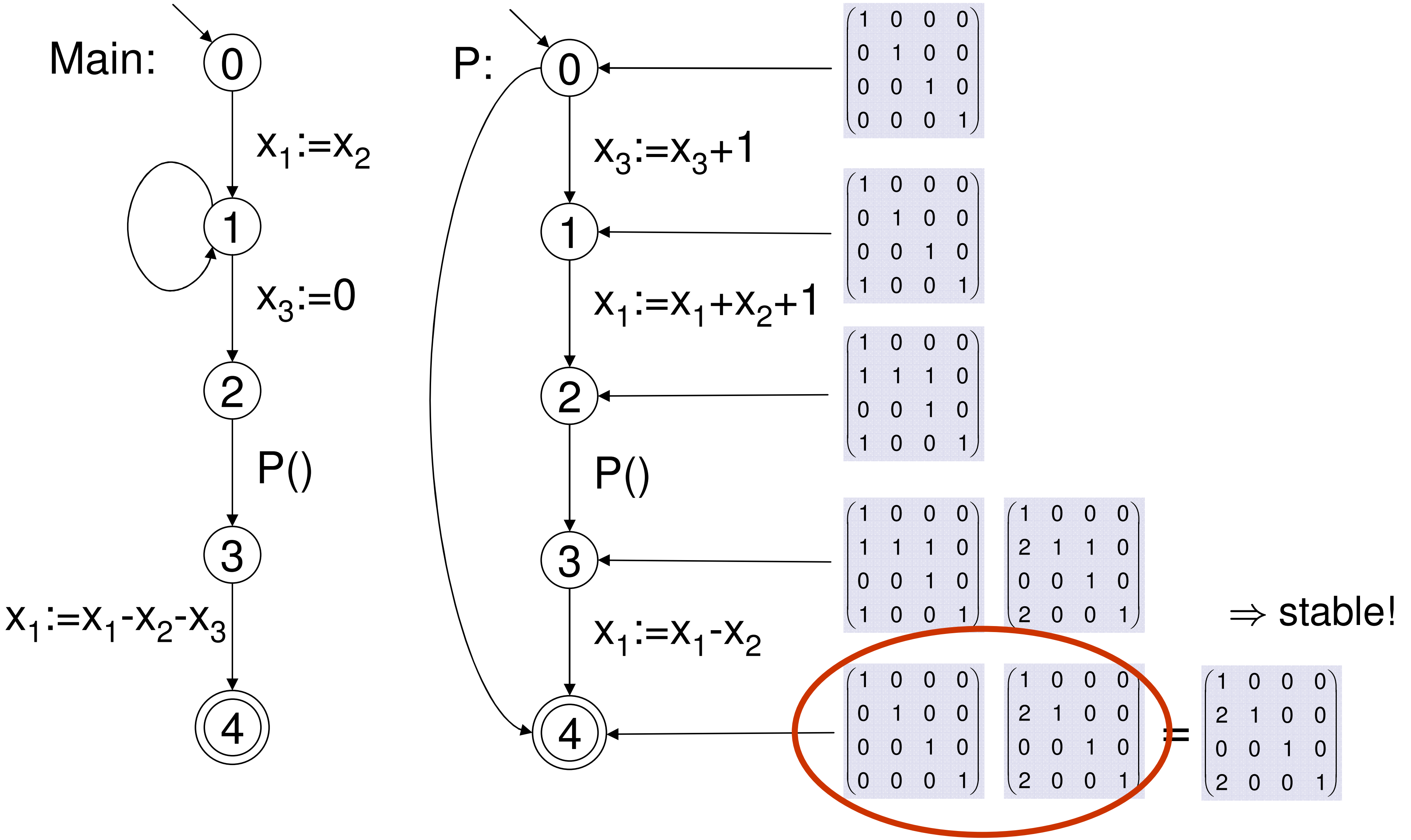
- The vector spaces

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

  can be computed precisely for all prg. points *u*.

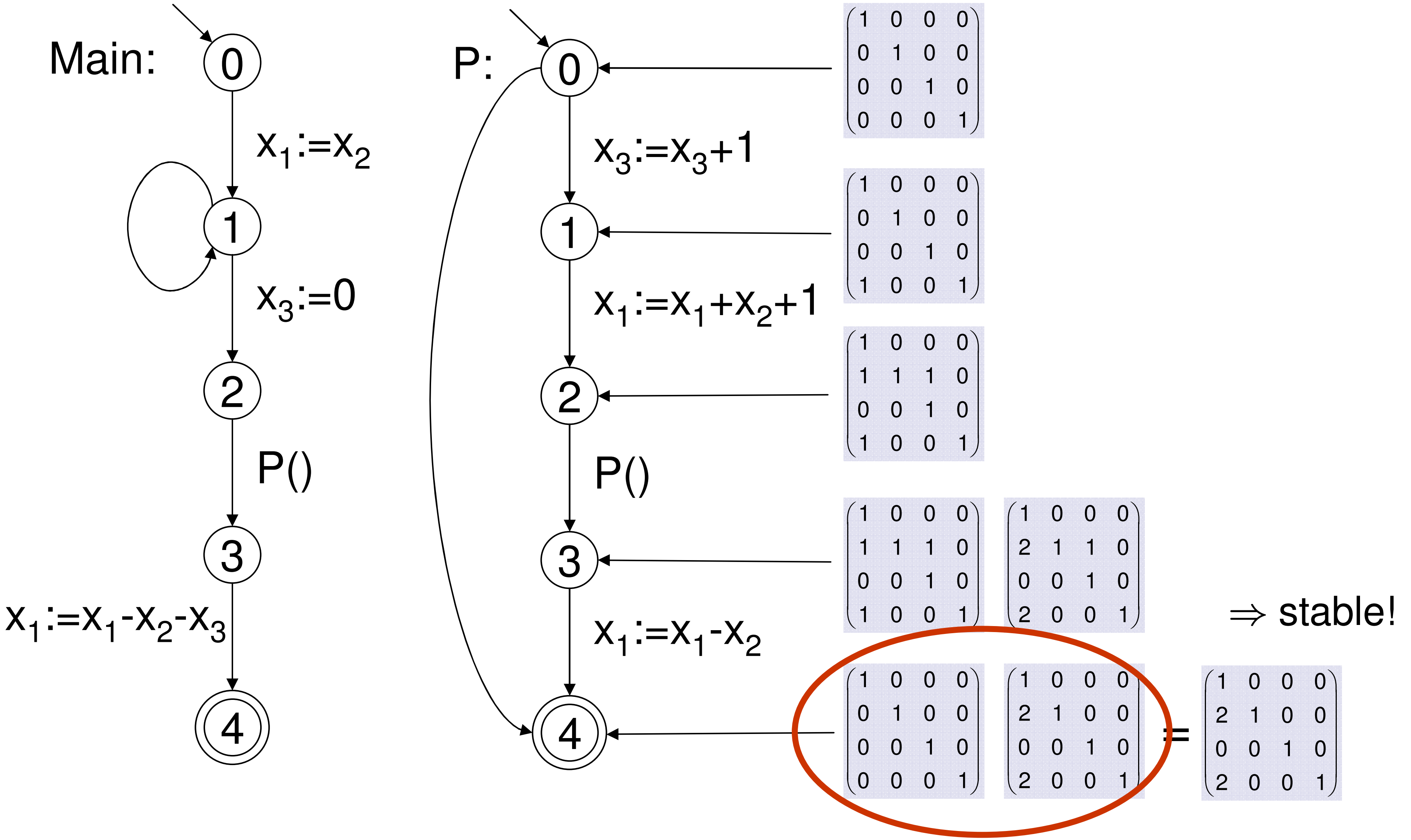
- The time complexity is linear in the program size and polynomial in the number of variables
  More specifically, for fields, it is $\mathcal{O}(n \cdot k^8)$
  (*n* size of the program, *k* number of variables)

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
$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$
$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\1&0&0&1\end{pmatrix}$
$\begin{pmatrix}1&0&0&0\\1&1&1&0\\0&0&1&0\\1&0&0&1\end{pmatrix}$
$\begin{pmatrix}1&0&0&0\\1&1&1&0\\0&0&1&0\\1&0&0&1\end{pmatrix}$ $\begin{pmatrix}1&0&0&0\\2&1&1&0\\0&0&1&0\\2&0&0&1\end{pmatrix}$
$\Rightarrow$ stable!
$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$ $\begin{pmatrix}1&0&0&0\\2&1&0&0\\0&0&1&0\\2&0&0&1\end{pmatrix}$ = $\begin{pmatrix}1&0&0&0\\2&1&0&0\\0&0&1&0\\2&0&0&1\end{pmatrix}$

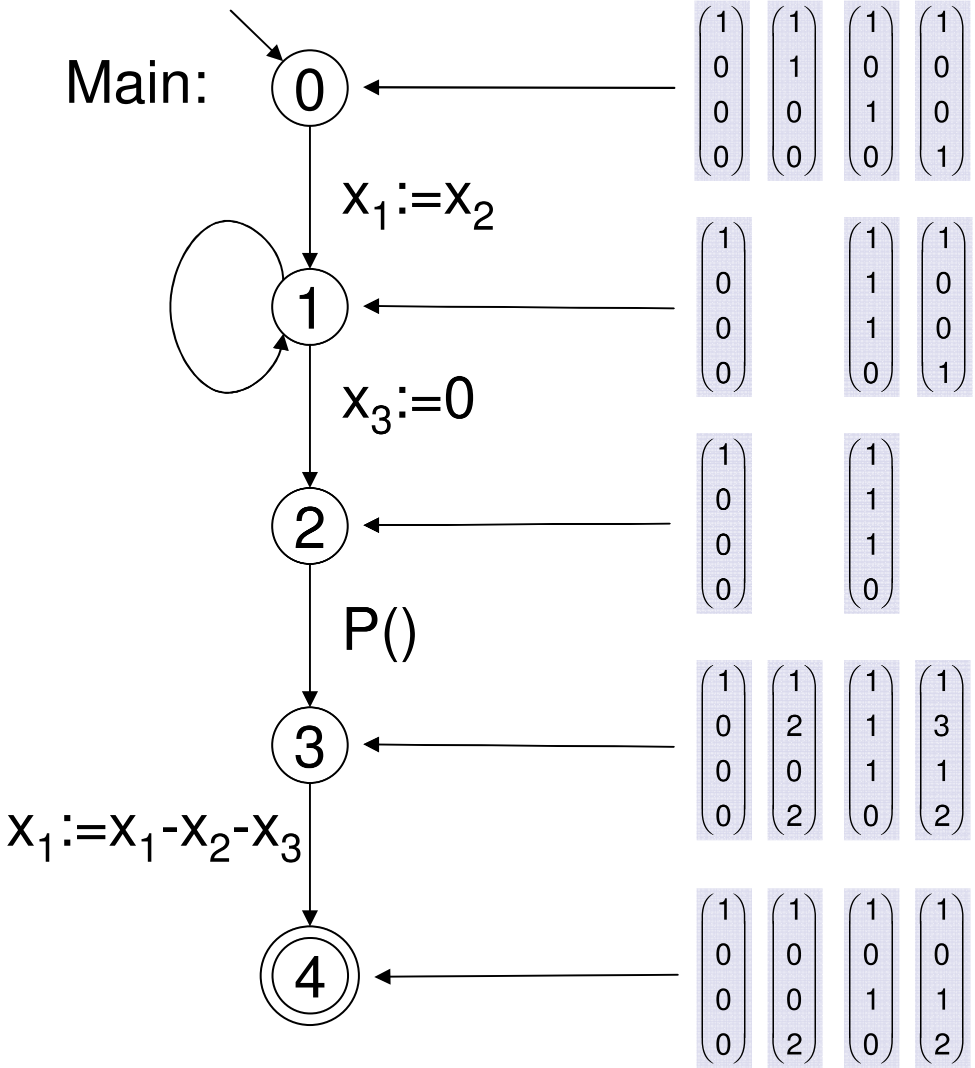
An Example



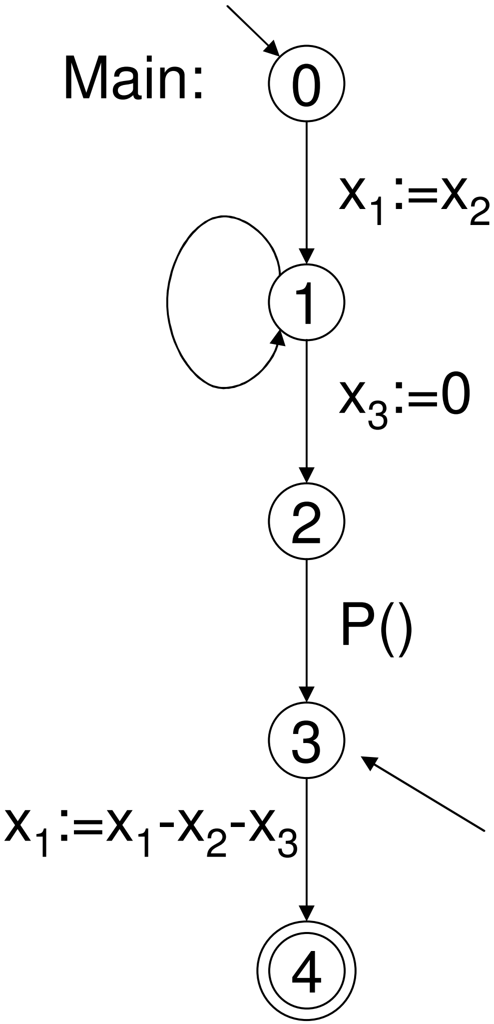
Main:
0
$x_1 := x_2$
1
$x_3 := 0$
2
P()
3
$x_1 := x_1 - x_2 - x_3$
4
P:
0
$x_3 := x_3 + 1$
1
$x_1 := x_1 + x_2 + 1$
2
P()
3
$x_1 := x_1 - x_2$
4
$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$
$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\1&0&0&1\end{pmatrix}$
$\begin{pmatrix}1&0&0&0\\1&1&1&0\\0&0&1&0\\1&0&0&1\end{pmatrix}$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

An Example

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix}$$



An Example



$a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 = 0$ is valid at 3

$$\Leftrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 2 \\ 1 & 1 & 1 & 0 \\ 1 & 3 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = 0$$

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

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 0 \\ 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 1 \\ 2 \end{pmatrix}$$

$\Rightarrow$ Just the affine relations of the form

$$ax_1 - ax_2 - ax_3 = 0 \qquad (\text{for } a \in \mathbb{F})$$

are valid at 3, in particular:

$$x_1 - x_2 - x_3 = 0$$

☺

Extensions

Also in the papers:

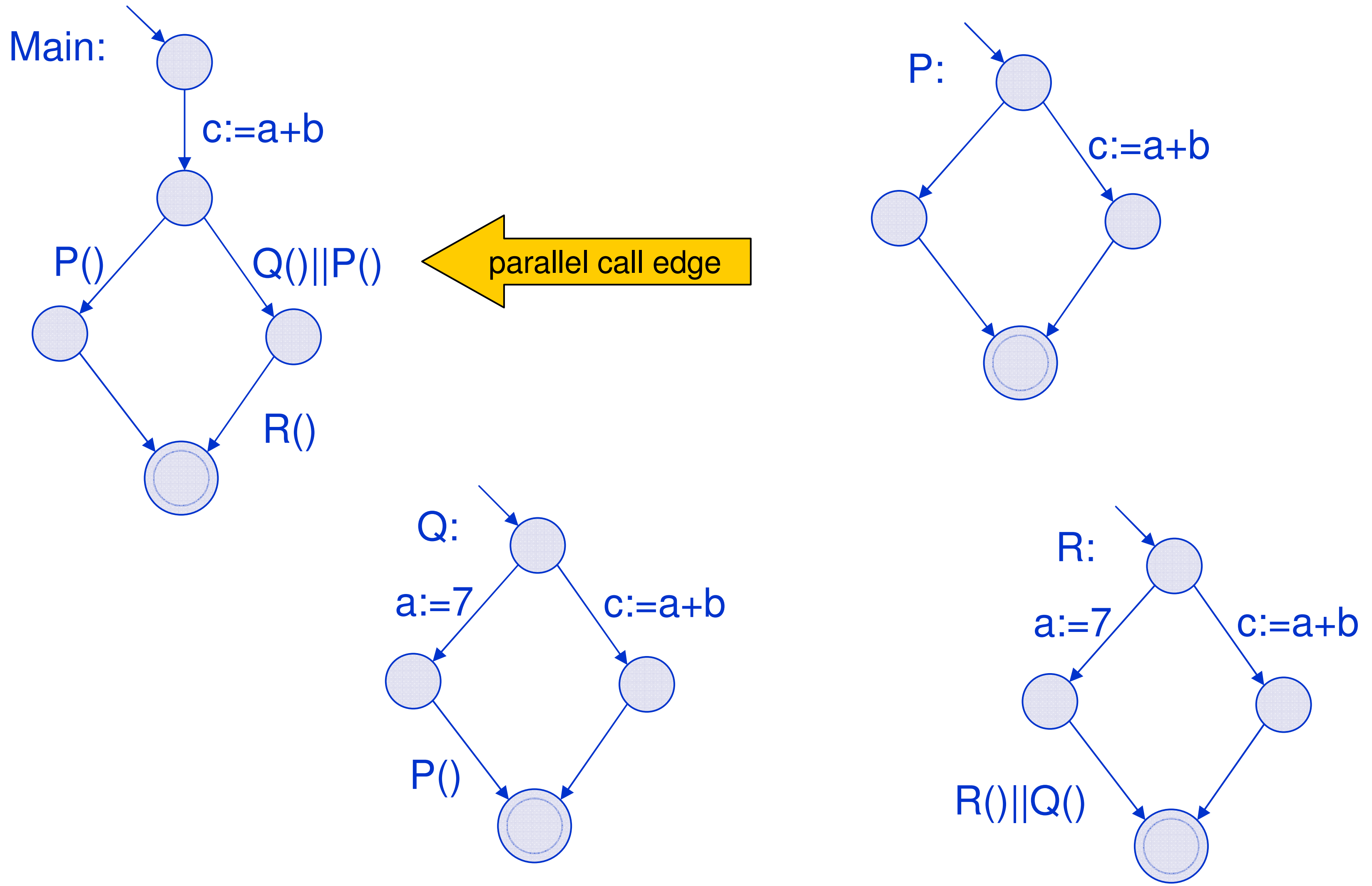
- Local variables, value parameters, return values
- Computing polynomial relations of bounded degree
- Affine pre-conditions
- Computing over modular rings (e.g. modulo 2^w) or PIR

End of Excursion 2

Overview

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

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

[Seidl/Steffen: ESOP 2000]

Reaching runs:

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

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

$R(u,q) \supseteq S(v) \cdot (R(u,p_i) \otimes P(p_{1-i}))$ — $e = (v, p_0 \,||\, p_1, \_)$ 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:

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

Treat other (separable) bitvector problems analogously...

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

Bitvector Problems

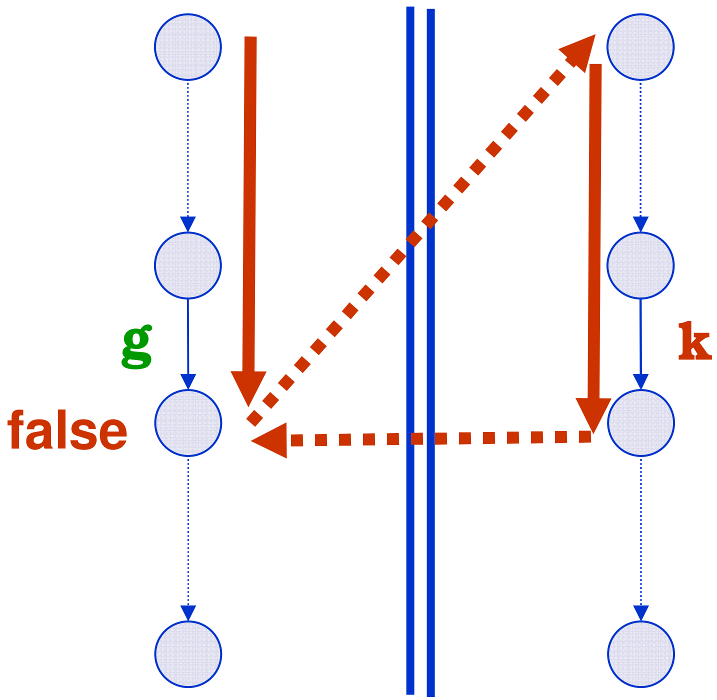
Problem of this algorithm:

Complexity: quadratic in program size:
quadratically many constraints for reaching runs !

Solution: linear-time „search for killers“-algorithm.

Idea of „Search for Killers“-Algorithm

[Knoop/Steffen/Vollmer: TACAS 1995]
[Seidl/Steffen: ESOP 2000]



the basic lattice:

false
|
true

the function lattice:

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

⇒ perform, „normal“ analysis but weaken information if a „killer“ can run in parallel !

Formalization of „Search for Killers"-Algorithm

Kill Potential:

$KP(p)$	$\sqsupseteq \top$	if p contains reachable edge e with $f_e = k$
$KP(p)$	$\sqsupseteq KP(q)$	if p calls q, $q \| \_$, or $\_ \| q$ at some reachable edge

Possible Interference:

$PI(p)$	$\sqsupseteq PI(q)$	if q contains reachable call to p
$PI(p_i)$	$\sqsupseteq PI(q) \sqcup KP(p_{1-i})$	if q contains reachable parallel call $p_0 \| p_1$, $i = 0,1$

Weaken data flow information in 2nd phase if killer can run in ||:

$R^\#(st_{Main})$	$\sqsupseteq init$	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
$R^\#(v)$	$\sqsupseteq PI(p)$	v reachable prg. point in p

Overview

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

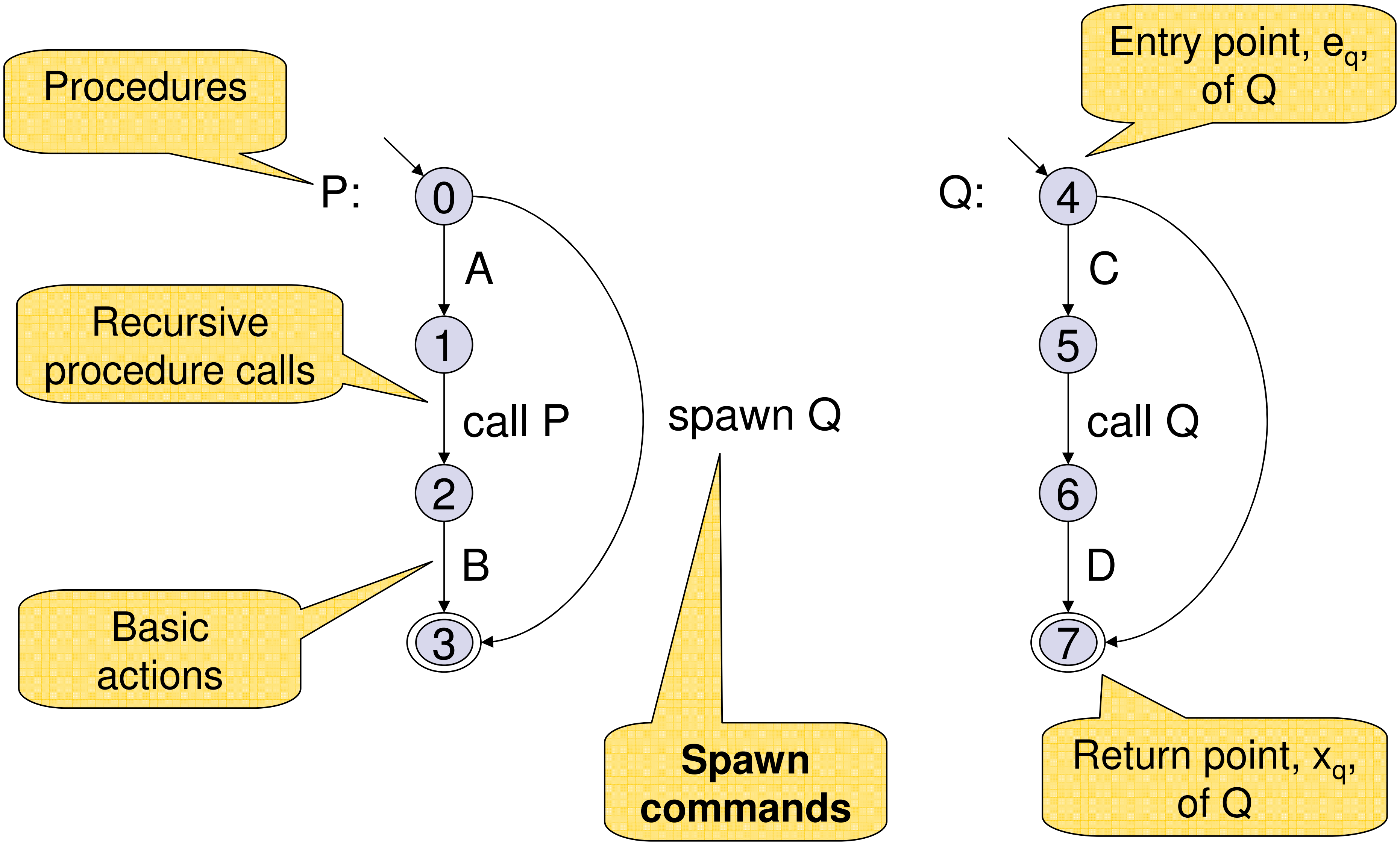
Control aspects

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

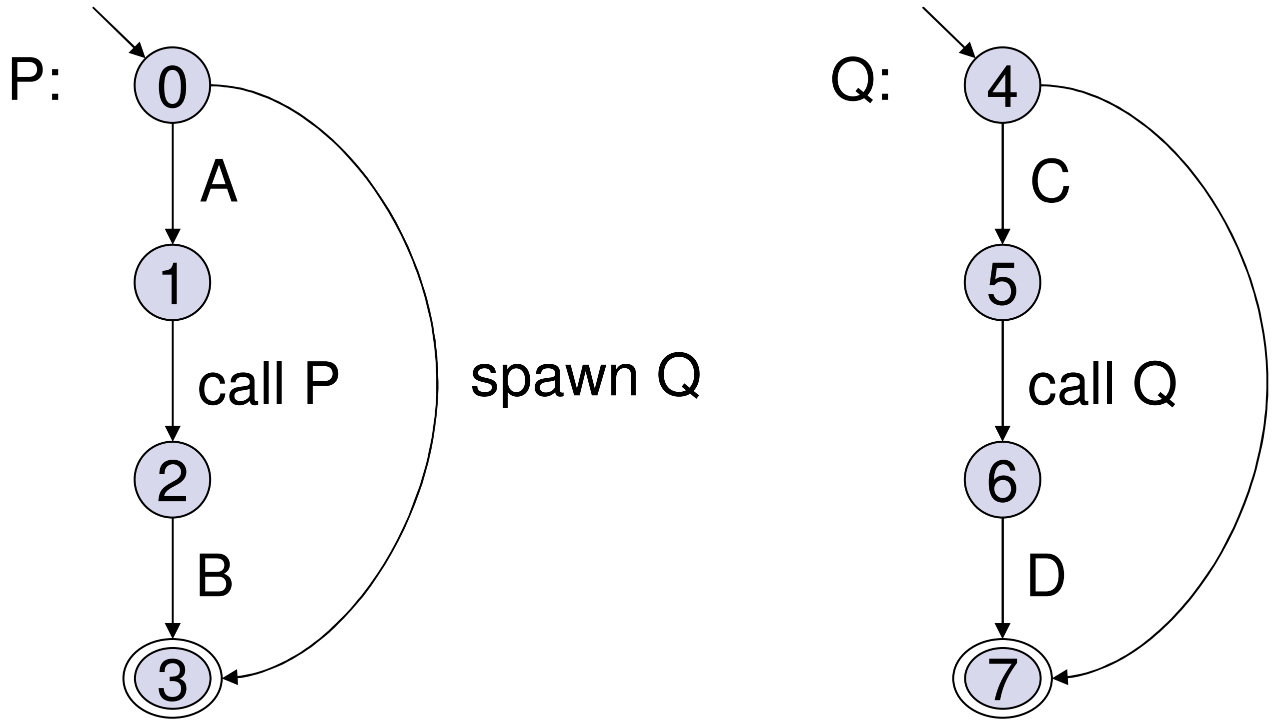
Technics used

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

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

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

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

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

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“.

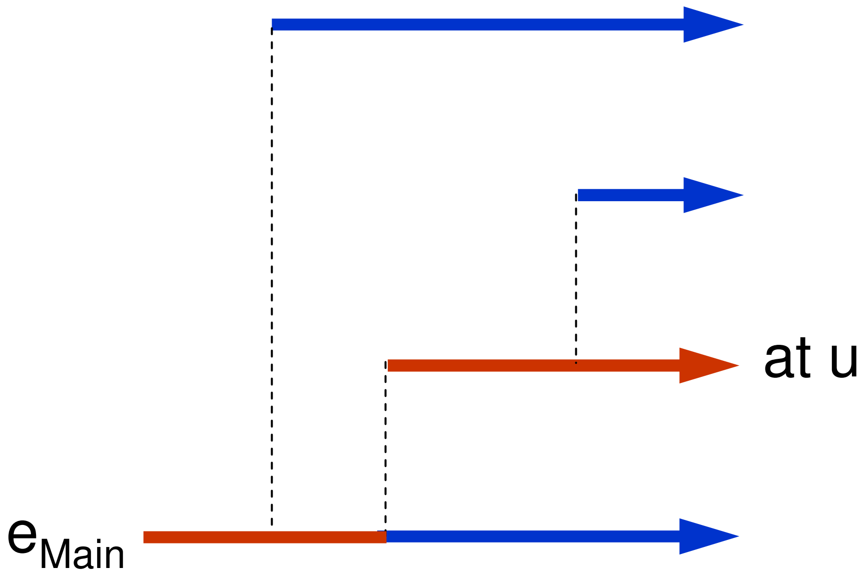
One Way Out

[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:

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

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

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:

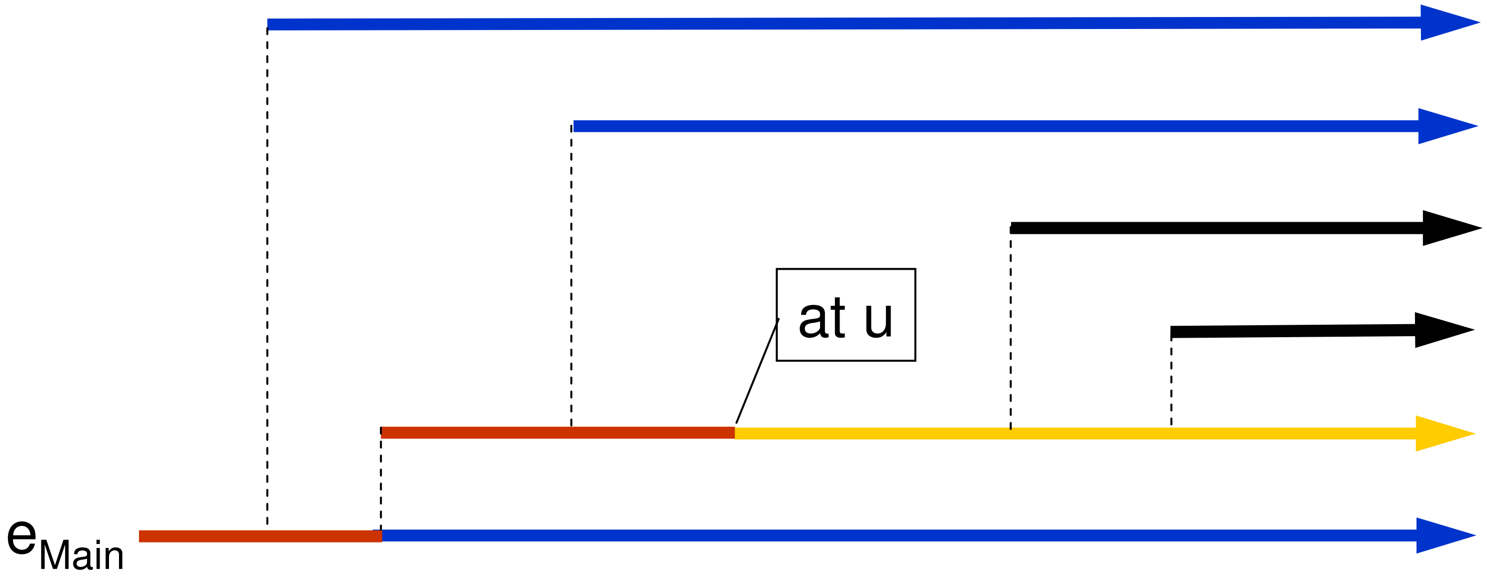
$$
\begin{array}{lll}
[\mathrm{init}] & \mathsf{S}[\mathsf{e}_q] \supseteq \{\varepsilon\} & \text{for } q \in P \\
[\mathrm{base}] & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e & \text{for } e = (u, \mathsf{base}\ \_, v) \in E \\
[\mathrm{call}] & \mathsf{S}[v] \supseteq \mathsf{S}[u]; e; \mathsf{S}[\mathsf{r}_q];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

Directly reaching paths:

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

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:

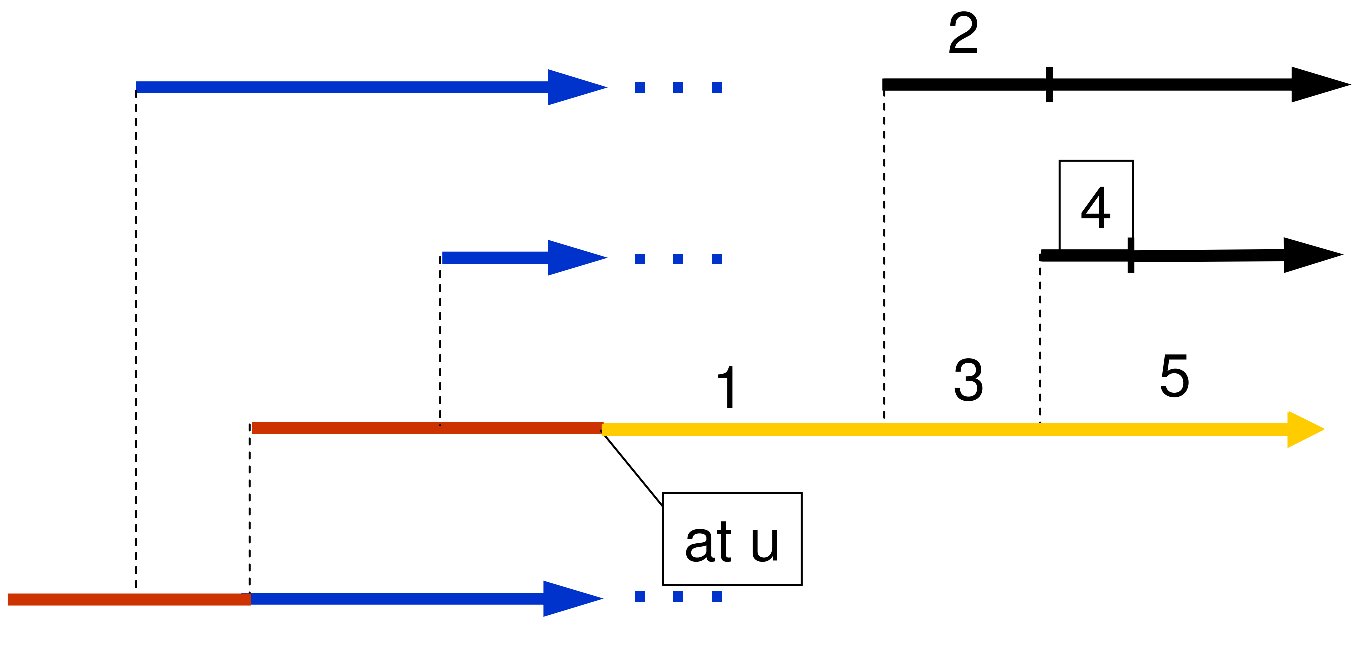
$$MOP^B[u] = \alpha^B(DLeave[u]) \sqcup \alpha^{PI}(PI[u]),$$

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

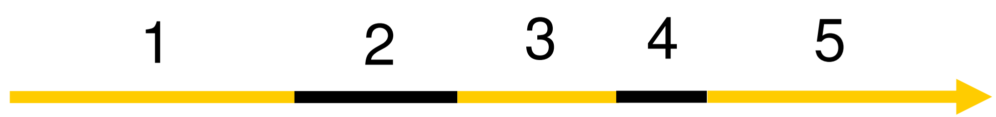
Remark

- 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

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

Corollary

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

Remark

- RDLeave[u] and PI[u] can be characterized by constraint systems (see paper)
- $\alpha^B(\mathrm{RDLeave}[u])$ and $\alpha^{PI}(\mathrm{PI}[u])$ 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
- Recent work: Precise analysis in presence of locks/monitors (see papers at SAS 2008, CAV 2009, VMCAI 2011)

End of Excursion 3

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 !