A Tutorial on Program Analysis

Markus Müller-Olm
Dortmund University

Thanks !

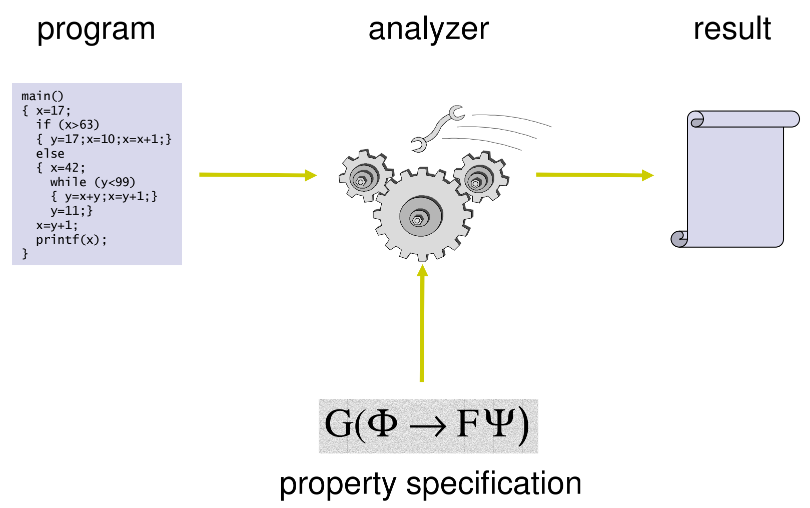
Helmut Seidl
(TU München)

and

Bernhard Steffen
(Universität Dortmund)

for discussions, inspiration, joint work, ...

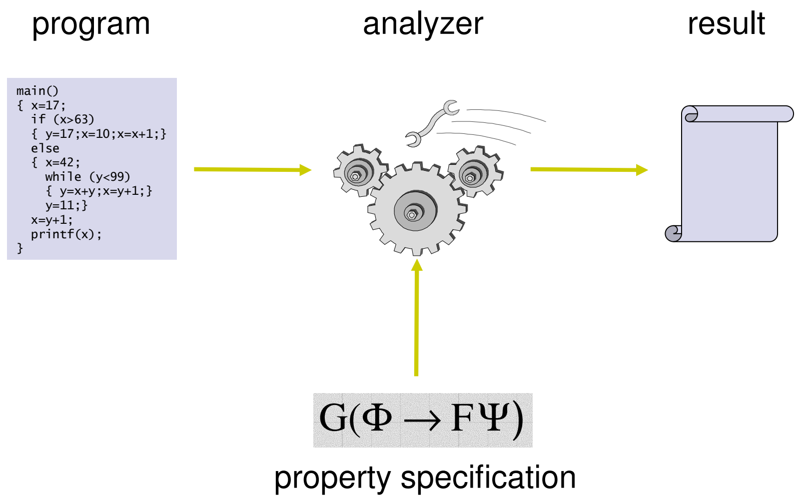
Dream of Program Analysis



Purposes of Automatic Analysis

- Optimizing compilation
- Validation/Verification
  - Type checking
  - Functional correctness
  - Security properties ☺
  - . . .
- Debugging

Dream of Program Analysis



Fundamental Limit

Rice's Theorem [Rice,1953]:

All non-trivial semantic questions about programs from a universal programming language are undecidable.

Two Solutions

Weaker formalisms

- analyze abstract models of systems
- e.g.: automata, labelled transition systems,...

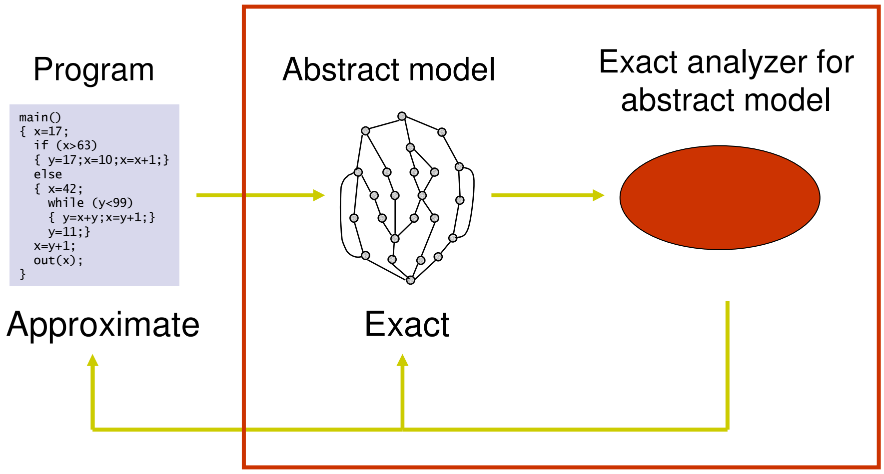
Approximate analyses

- yield sound but, in general, incomplete results
- e.g.: detects some instead of all constants

Model checking

Flow analysis
Abstract interpretation
Type checking

Weaker Formalisms



Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion

Apology for not giving detailed credit !

Credits

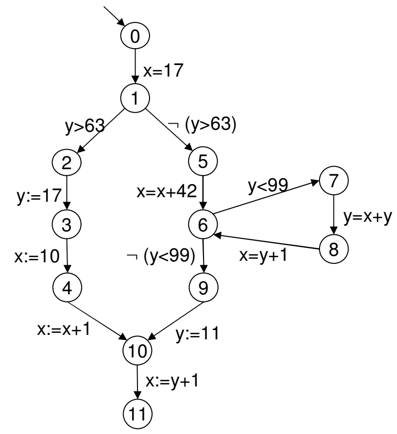
- Pioneers of Iterative Program Analysis:
  - Kildall, Wegbreit, Kam & Ullman, Karr, ...
- Abstract Interpretation:
  - Cousot/Cousot, Halbwachs, ...
- Interprocedural Analysis:
  - Sharir & Pnueli, Knoop, Steffen, Rüthing, Sagiv, Reps, Wilhelm, Seidl, ...
- Analysis of Parallel Programs:
  - Knoop, Steffen, Vollmer, Seidl, ...
- And many more:
  - Apology ...

Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion

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

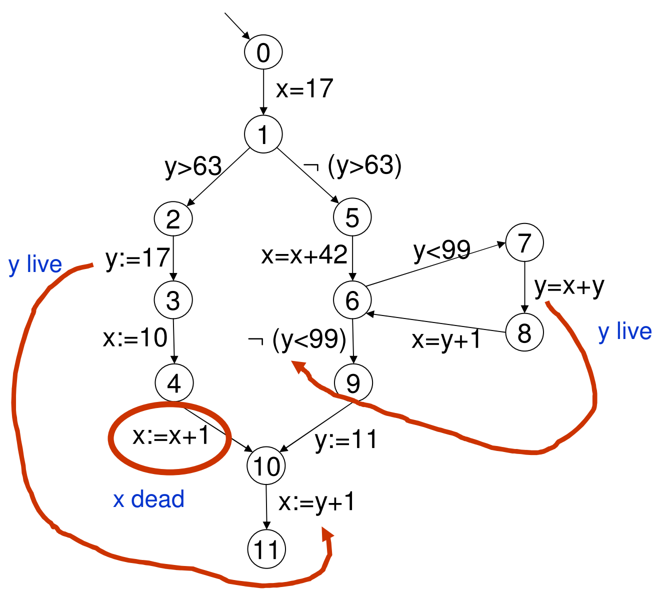
→ use approximate algorithm:
e.g.: ignore that guards prohibit certain execution paths

Technique:

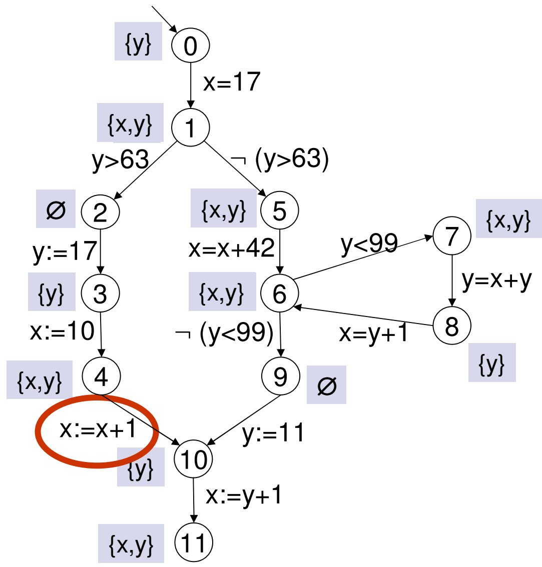
1) perform *live variables* analyses:
variable *x* is live at program point *u* iff
there is a path from *u* on which *x* is used before it is modified

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

Live Variables



Live Variables Analysis



Remarks

- Forward vs. backward analyses
- (Separable) bitvector analyses
  - forward: reaching definitions, available expressions, ...
  - backward: live/dead variables, very busy expressions, ...

Partial Order

Partial order $(L,\sqsubseteq)$:

set L with binary relation $\sqsubseteq\ \subseteq L\times L$ s.t.

- $\sqsubseteq$ is reflexive:
  $\forall x\in L:\quad x\sqsubseteq x$
- $\sqsubseteq$ is antisymmetric:
  $\forall x,y\in L:\quad x\sqsubseteq y\ \Rightarrow\ \neg(y\sqsubseteq x)$
- $\sqsubseteq$ is transitive
  $\forall x,y,z\in L:\quad (x\sqsubseteq y\wedge y\sqsubseteq z)\ \Rightarrow\ x\sqsubseteq z$

For a subset $X\subseteq L$:

$\sqcup X$: least upper bound (*join*), if it exists

$\sqcap X$: greatest lower bound (*meet*), if it exists

Complete Lattice

- Complete lattice $(L,\sqsubseteq)$:
  - a partial order $(L,\sqsubseteq)$ for which $\sqcup X$ 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 $\bot$ exists: $\quad \bot = \sqcup L = \sqcap\emptyset$
  - greatest element $\top$ exists: $\quad \top = \sqcup\emptyset = \sqcap L$
- Example:
  - for any set A let $P(A) = \{X \mid X\subseteq A\}$.
  - $(P(A),\subseteq)$ is a complete lattice.
  - $(P(A),\supseteq)$ is a complete lattice.

Interpretation in Approximate Program Analysis

$x \sqsubseteq y$:

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

$\sqcup X$ for $X \subseteq L$:

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

Remark:

often dual interpretation in the literature !

Example:

lattice for live variables analysis:

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

Specifying Live Variables Analysis by a Constraint System

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

$V^\#[fin] \sqsupseteq init$, for fin, the termination node

$V^\#[u] \sqsupseteq f_e(V^\#[v])$, for each edge $e = (u,s,v)$

where $init = \text{Var}$,

f_e:P(Var) $\to$ P(Var), $f_e(x) = x \setminus \text{kill}_e \cup \text{gen}_e$, 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“.
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.

Data-Flow Frameworks

- Correctness
  - generic properties of frameworks can be studied and proved
- Implementation
  - efficient, generic implementations can be constructed

Questions

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

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:

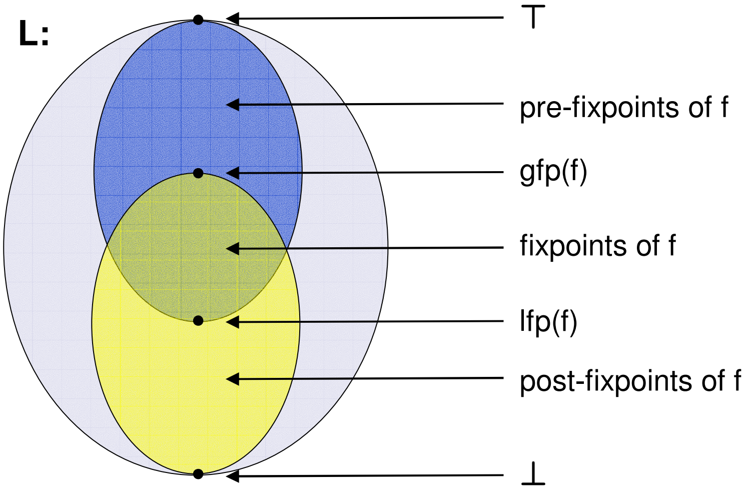
Every monotonic function f on a complete lattice L has a least fixpoint lfp(f) and a greatest fixpoint gfp(f).

More precisely,

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

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

Knaster-Tarski Fixpoint Theorem



Source: Nielson/Nielson/Hankin, *Principles of Program Analysis*

Smallest Solutions Exist Always

- 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. ☺

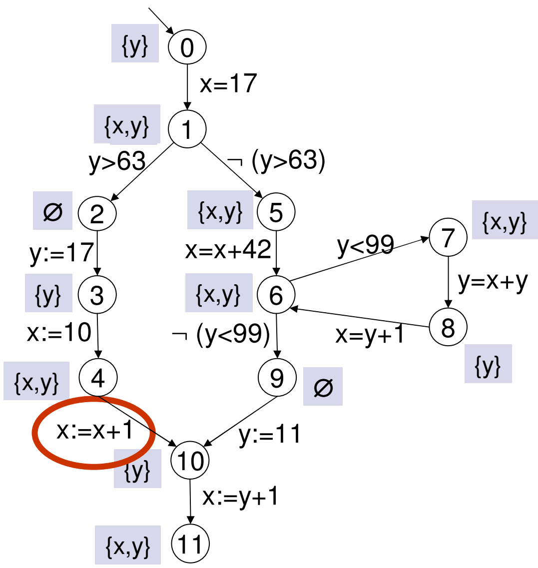
Questions

- 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 $W \neq \emptyset$ {
    $v = Extract(W);$
    forall (u, s **with** $e = (u,s,v)$ edge) {
        $t = f_e(A[v]);$
        if $\neg(t \sqsubseteq A[u])$ {
            $A[u] = A[u] \sqcup t;$
            $W = W \cup \{u\};$
        }
    }
}

Live Variables Analysis



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 $e = (u,s,v)$

If and when worklist algorithm terminates:

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

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

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

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

2) for all ascending chains $(l_n)_n$, the ascending chain $(w_n)_n$ defined by
$w_0 = l_0, \quad w_{i+1} = w_i \nabla l_i$ for $i>0$
stabilizes eventually.

Workset-Algorithm with Widening

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

Invariants of the Main Loop

a) ~~$A[u] \sqsubseteq MFP[u]$~~ 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)

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

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

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

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

A widening operator:

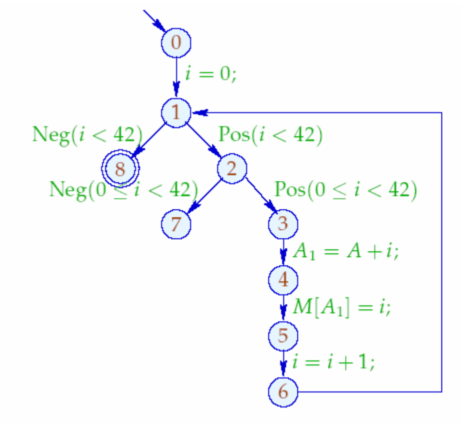
$$[l_0, u_0] \triangledown [l_1, u_1] = [l_2, u_2], \text{ where}$$

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

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



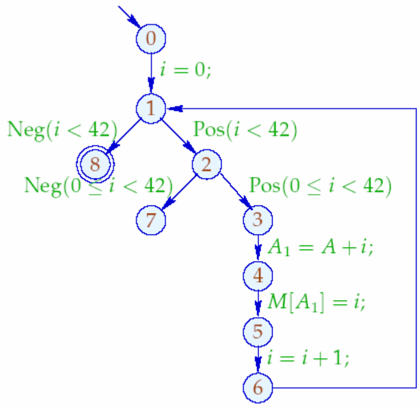
	1		2		3	
	l	u	l	u	l	u
0	$-\infty$	$+\infty$	$-\infty$	$+\infty$		
1	0	0	0	$+\infty$		
2	0	0	0	$+\infty$		
3	0	0	0	$+\infty$		
4	0	0	0	$+\infty$	dito	
5	0	0	0	$+\infty$		
6	1	1	1	$+\infty$		
7	$\bot$		42	$+\infty$		
8	$\bot$		42	$+\infty$		

$\Rightarrow$ Result is far too imprecise ! ☹

Example taken from: H. Seidl, Vorlesung „Programmoptimierung“, WS 04/05

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.

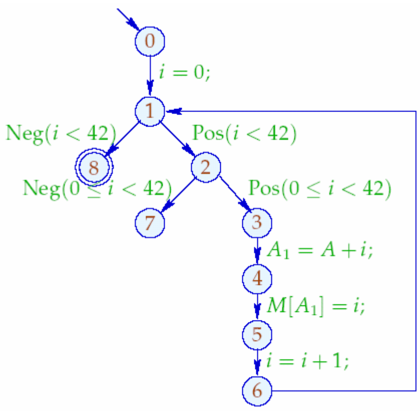


	1		2		3	
	l	u	l	u	l	u
0	$-\infty$	$+\infty$	$-\infty$	$+\infty$	dito	
1	0	0	0	$+\infty$		
2	0	0	0	41		
3	0	0	0	41		
4	0	0	0	41		
5	0	0	0	41		
6	1	1	1	42		
7	$\perp$		$\perp$			
8	$\perp$		42	$+\infty$		

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

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 ! ☺

Remarks

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

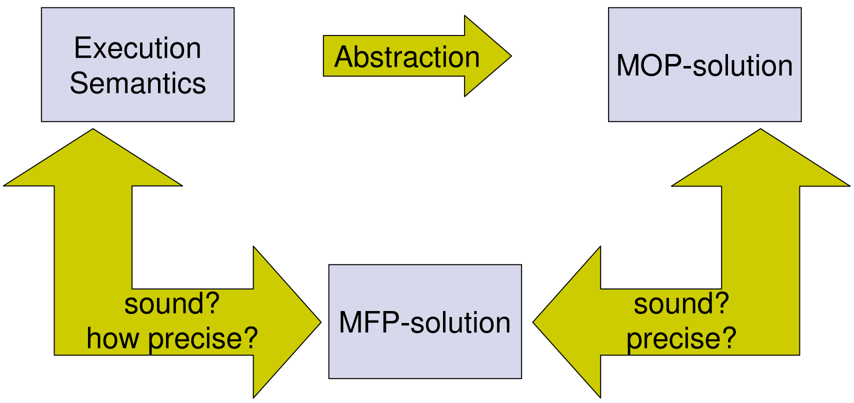
Questions

- 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

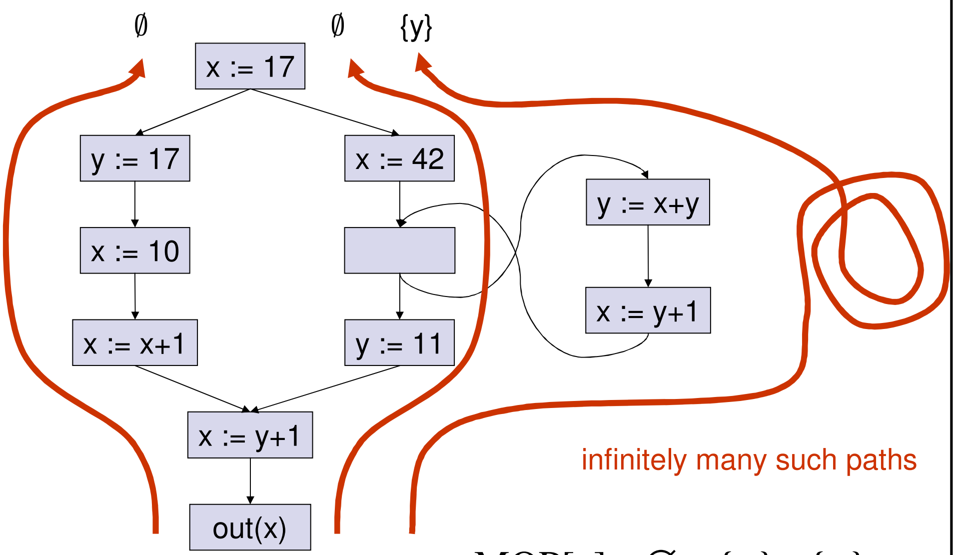
Questions

- 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



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

Meet-Over-All-Paths Solution

- 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.

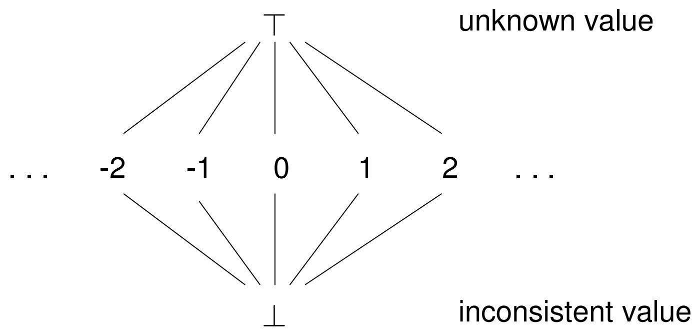
Remark:

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.

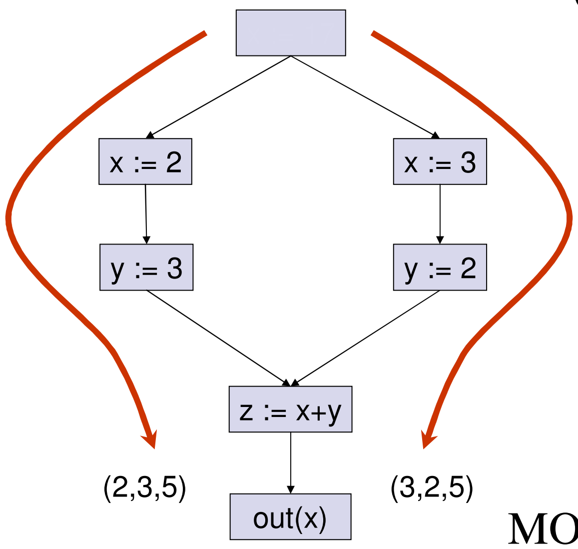
Lattice for Constant Propagation



Constant Propagation Framework & Instance

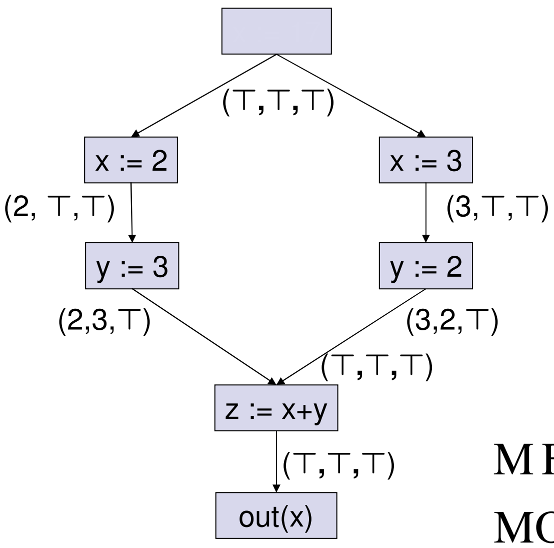
lattice L	$\mathrm{Var} \to (\mathbb{Z} \cup \{\bot, \top\}), \; = \; \mathrm{Var} \to \mathrm{ConstVal}$
$\sqsubseteq$	$\rho \sqsubseteq \rho' :\Leftrightarrow \forall x : \rho(x) \sqsubseteq \rho'(x)$
$\sqcup$	pointwise join
$\top$	$\top(x) = \top$ f.a. $x \in \mathrm{Var}$
control flow	program graph
initial value	$\top$
function space	$\{ f : D \to D \mid f \text{ monotone} \}$
f_i	$f_i(d) = \begin{cases} d[x \mapsto [\![e]\!]^{CP}(d)] & \text{if } i \text{ annotated with } x := e \\ d & \text{otherwise} \end{cases}$

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



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

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



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

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

Correctness Theorem

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:

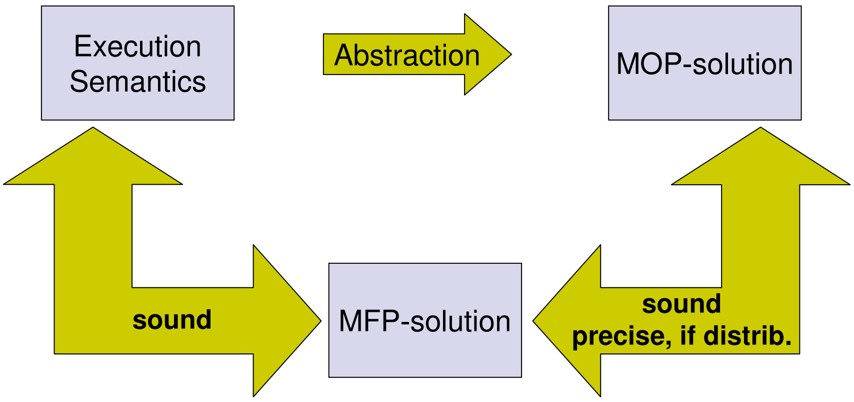
In any monotone framework:

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

Remark:

Any "reasonable" framework is monotone. ☺

Assessing Data Flow Frameworks



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 $o^\#$

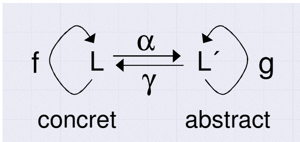
constraint system for
Analysis
on abstract lattice $(D^\#,\sqsubseteq^\#)$

$MFP^\#$

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 $\Sigma = Var \to Val$

Transfer Lemma



Situation:

complete lattices $(L,\sqsubseteq)$, $(L',\sqsubseteq')$
montonic functions $f{:}L\to L$, $g{:}\ L'\to L'$, $\alpha{:}L\to L'$

Definition:

Let $(L,\sqsubseteq)$ be a complete lattice.
$\alpha : L\to L$ is called *universally-disjunctive* iff $\forall\, X\subseteq L{:}\ \alpha(\sqcup X) = \sqcup\{\,\alpha(x) \mid x\in X\}$.

Remark:

- (α,γ) is called *Galois connection* iff $\forall\, x\in L,\ x'\in L'{:}\ \alpha(x) \sqsubseteq' y \Leftrightarrow x \sqsubseteq \gamma(y)$.
- α is universally-disjunctive iff $\exists\, \gamma{:}L'\to L : (\alpha,\gamma)$ is Galois connection.

Transfer Lemma:

Suppose α is universally-disjunctive. Then:
(a) $\alpha\circ f \sqsubseteq' g\circ\alpha \Rightarrow \alpha(lfp(f)) \sqsubseteq' lfp(g)$.
(b) $\alpha\circ f = g\circ\alpha \Rightarrow \alpha(lfp(f)) = lfp(g)$.

Abstract Interpretation

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

Correct abstract interpretation:

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

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

Correct and precise abstract interpretation:

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

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

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

Abstract Interpretation

Constraint system for reaching runs:

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

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

Operational justification:

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

$$\underline{R}[u] = R^{op}[u] =_{def} \{r \in Edges^* \mid st \xrightarrow{r} u\} \quad \text{f.a. } u$$

Prove:

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

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

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

$\Rightarrow$ $R^{op}[u] \subseteq \underline{R}[u]$ 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} : Edges* $\to$ L by

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

Remark:

For all monotone frameworks the abstraction is correct:

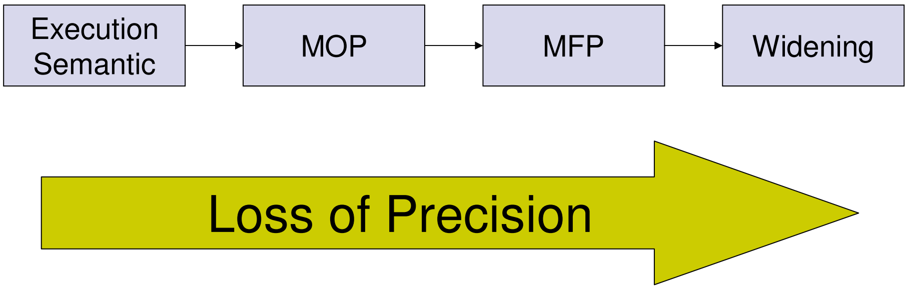
$\alpha_{MOP}(\underline{R}[u]) \sqsubseteq \underline{R}^\#[u]$ f.a. prg. points u

For all universally-distributive frameworks 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*). ☺

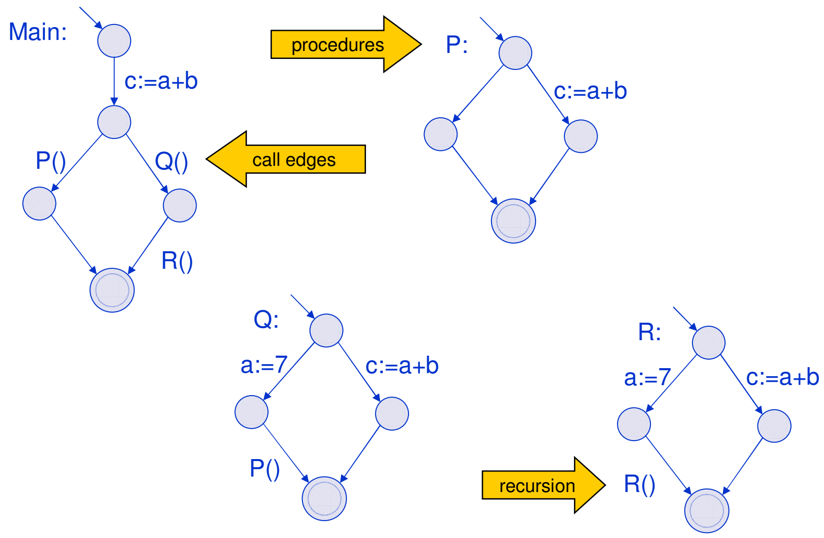
Where Flow Analysis Looses Precision



Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion

Interprocedural Analysis



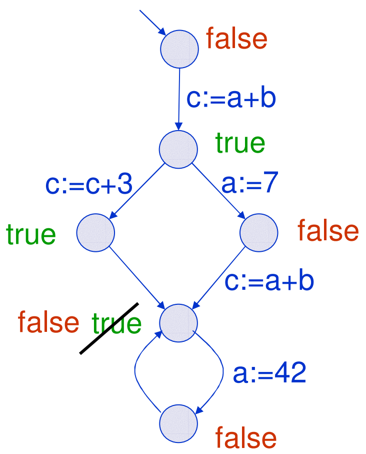
Running Example:
Availability of the single expression a+b

The lattice:

false a+b not available

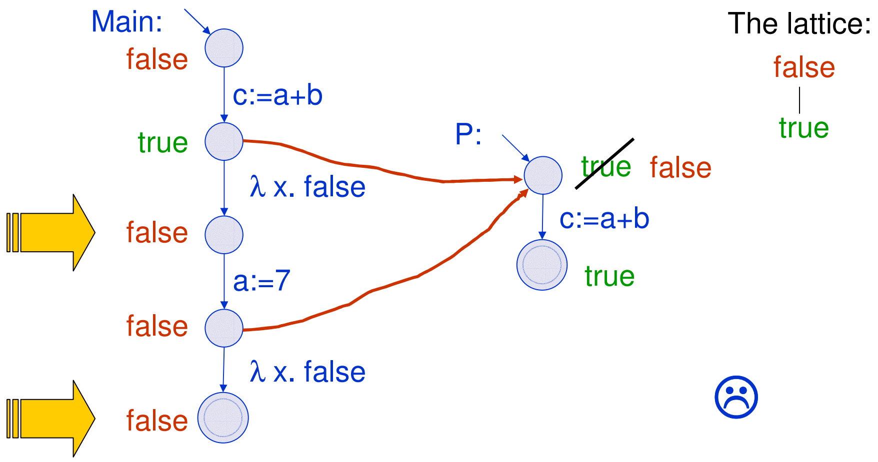
true a+b available

Initial value: false



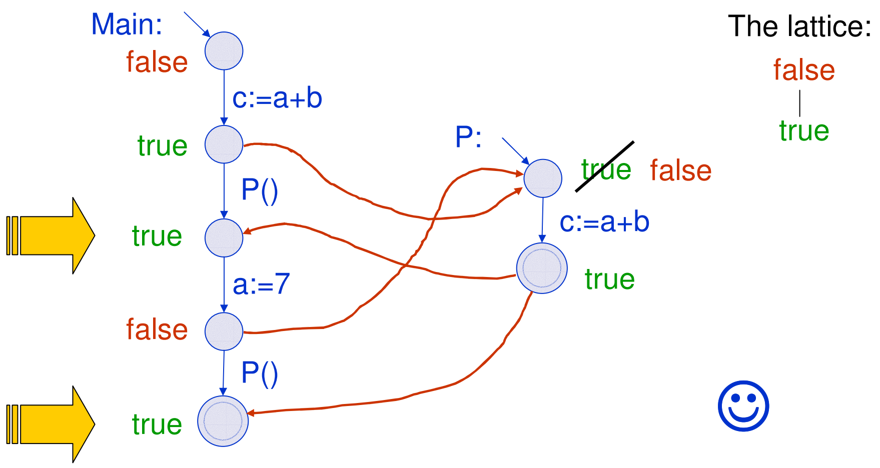
Intra-Procedural-Like Analysis

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



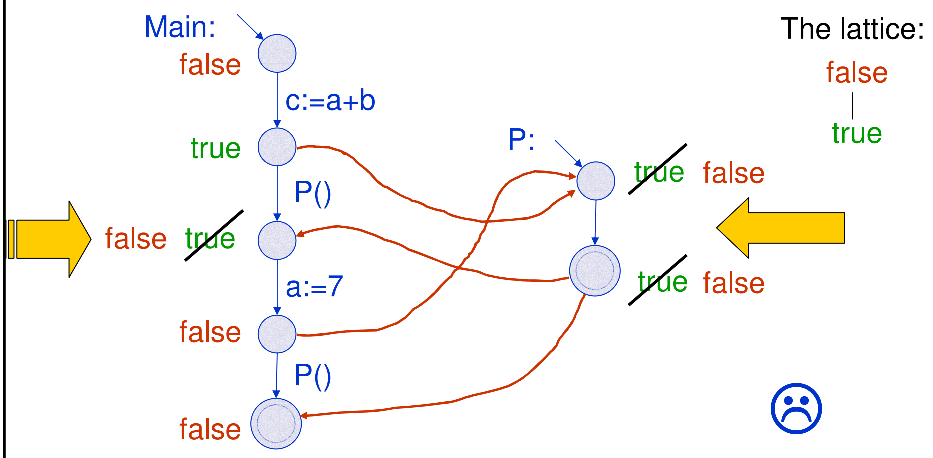
Context-Insensitive Analysis

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



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

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

Summary information:

1) Functional approach:
Use (monotonic) functions on data flow informations !

2) Relational approach:
Use relations (of a representable class) on data flow informations !

3) etc...

Functional Approach for Availability of Single Expression Problem

Observations:

Just three montone functions on lattice L:

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

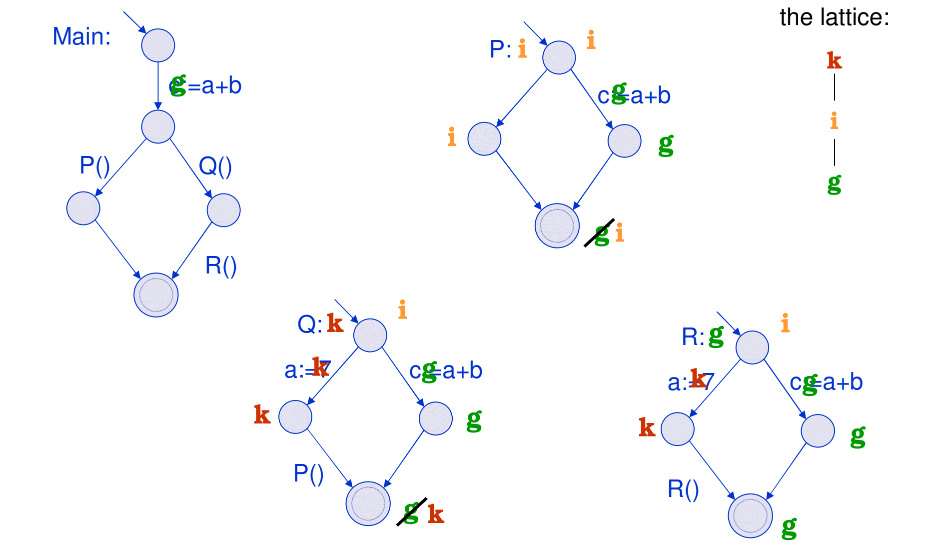
$$\begin{array}{ccc} \lambda\, x\, .\, \textbf{false} & & \textbf{k (ill)} \\ | & & | \\ \lambda\, x\, .\, x & & \textbf{i (gnore)} \\ | & & | \\ \lambda\, x\, .\, \textbf{true} & & \textbf{g (enerate)} \end{array}$$

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

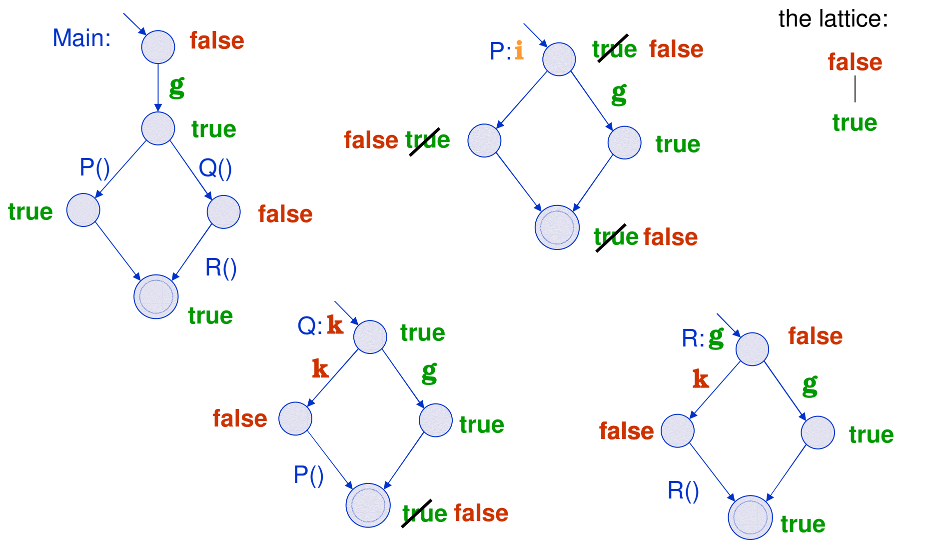
$$h \circ f = \begin{cases} f & \text{if } h = \mathrm{i} \\ h & \text{if } h \in \{\mathrm{g},\mathrm{k}\} \end{cases}$$

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



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.

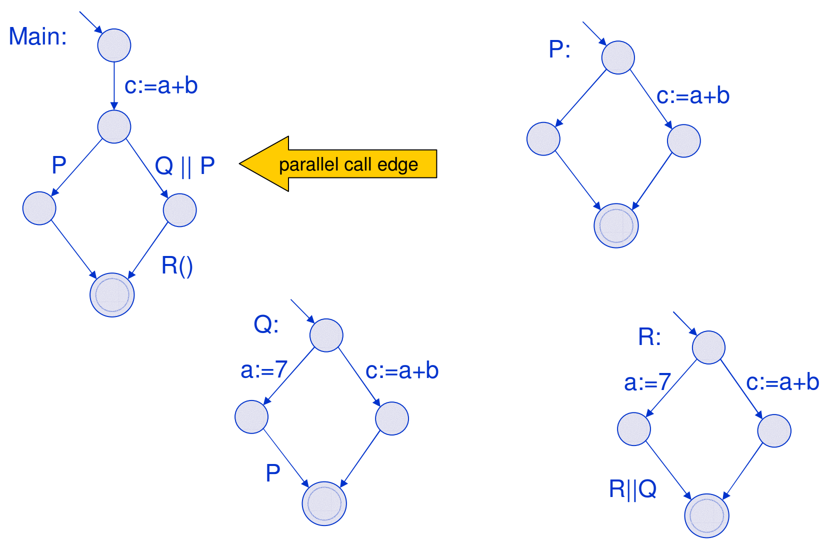
Extensions

- Parameters, return values, local variables can be handled also

Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- 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:

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

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
$S(v) \supseteq S(u) \cdot (S(p_0) \otimes S(p_1))$	$e = (u, p_0 \mid\mid p_1, v)$ parallel call edge

Constraint System for 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:

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

Interleaving potential:

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

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

Example:

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

Only new ingredient:

interleaving operator $\otimes$ must be abstracted ! ☺

Case: Availability of Single Expression

Abstract shuffle operator:

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

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 ! ☺

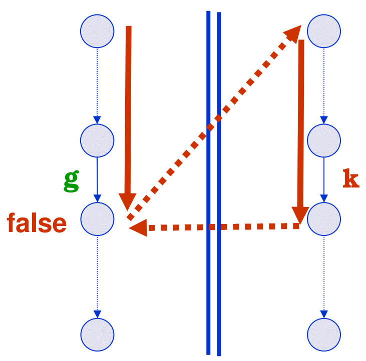
Bitvector Problems

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



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

Beyond Bitvector-Analysis: Analysis of Transitive Dependences

- Analysis problem:
  - Is there an execution from *u* to *v* mediating a dependence from *x* to *y* ?
    - a:=x … b:=a … c:=b … y:=c
- Anwendungen:
  - *program slicing*
  - *faint-code*-elimination
  - copy constants
  - information flow ☺

Complexity Results

In parallel programs:

[MO/Seidl, STOC 2001]

analysis of transitive dependences is …

- undecidable, interprocedurally
- PSPACE-complete, intraprocedurally
- already NP-complete for programs without loop

under assumption

„Basic statements are executed atomically“

Analysis of Transitive Dependences in Parallel Programs

```
         ||  x := 1;
a := x   ||  x := 0;
         ||  a := 0;
         ||  write(a)
```

Nevertheless: a is constantly 0 !

Spring School on Security, Marseille, April 25-29, 2005

Algorithmic Potential

In parallel programs: [MO, TCS 2004]

- transitive dependences are computable (in exponential time), even interprocedurally, if (unrealistic) assumption

  „Basic statements are executed atomically“

  is abandoned !

Technique:

- a (complex) domain of „dependence traces“
- abstract operators $;^\#$ and $\otimes^\#$ which are precise and correct abstractions of ; and $\otimes$ relative to a non-atomic semantics.

Analysis of Transitive Dependences in Parallel Programs

atomic execution

```
         ||  x := 1;
a := x   ||  x := 0;
         ||  a := 0;
         ||  write(a)
```

non-atomic execution

```
         ||  x := 1;
p := x   ||  x := 0;
a := p   ||  a := 0;
         ||  write(a)
```

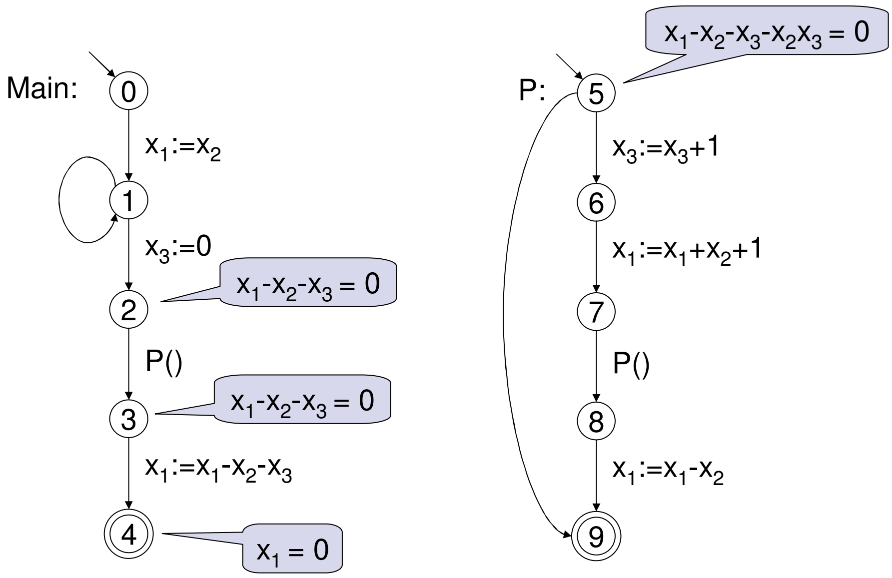
a ist constantly 0 !

a is not constantly 0 !

Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion

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,...)

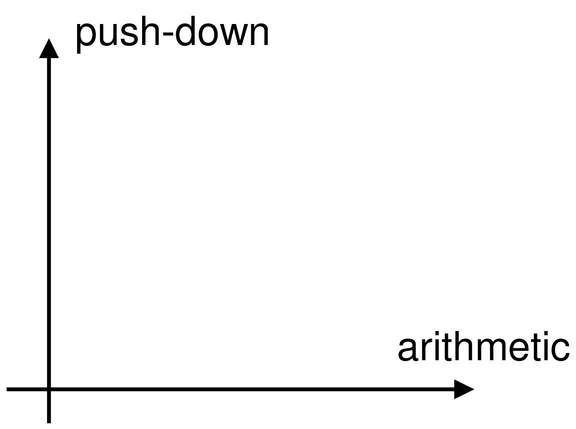
- determine all valid affine relations:
  $a_0 + \sum a_i x_i = 0$ $\quad a_i \in R$ $\quad 5x+7y-42=0$
- determine all valid polynomial relations (of degree $\leq$ d):
  $p(x_1,\ldots,x_k) = 0$ $\quad p \in R[x_1,\ldots,x_n]$ $\quad 5xy^2+7z^3-42=0$

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

Finding Invariants in Affine Programs

- Intraprocedural:
  - [Karr 76]: affine relations over fields
  - [Granger 91]: affine congruence relations over $\mathbb{Z}$
  - [Gulwani/Necula 03]: affine relations over random $\mathbb{Z}_p$, p prime
  - [MO/Seidl 04]: polynomial relations over fields
- Interprocedural:
  - [Horwitz/Reps/Sagiv 96]: linear constants
  - [MO/Seidl 04]: polynomial relations over fields
  - [Gulwani/Necula 05]: affine relations over random $\mathbb{Z}_p$, p prime
  - [MO/Seidl 05]: polynomial relations over modular rings $\mathbb{Z}_m$, $m \in \mathbb{Z}$ and PIRs

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

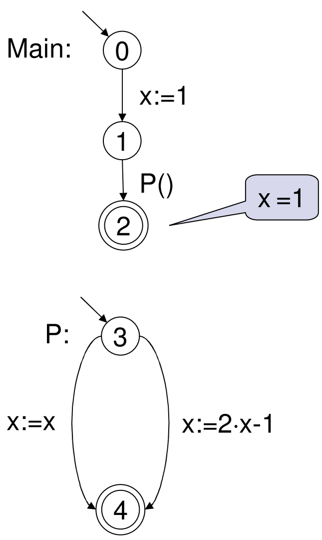
Call-string approach [Sharir/Pnueli, 1981]

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

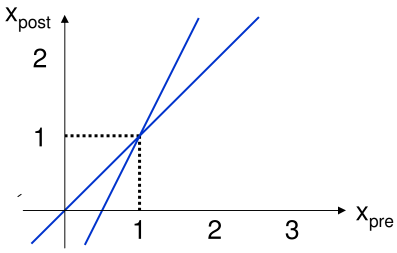
Relational analysis [Cousot², 1977]

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

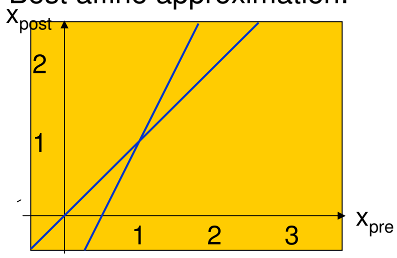
Relational Analysis is Not Strong Enough



True relational semantics of P:



Best affine approximation:



Towards the Algorithm ...

Concrete Semantics of an Execution Path

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

$$
\begin{aligned}
& [\![x_1 := x_1 + x_2 + 1;\ x_3 := x_3 + 1]\!](v) \\
= \ & [\![x_3 := x_3 + 1]\!]\left([\![x_1 := x_1 + x_2 + 1]\!](v) \right) \\
= \ & [\![x_3 := x_3 + 1]\!]\left(\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right) \\
= \ & \left(\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \right)
\end{aligned}
$$

Affine Relations

- An affine relation can be viewed as a vector:

$$5 + x_1 - 2x_2 - x_3 \ = \ 0 \quad \text{corresponds to} \quad a = \begin{pmatrix} 5 \\ 1 \\ -2 \\ -1 \end{pmatrix}$$

WP of Affine Relations

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

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

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

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

- Matrices M form the R-module $R^{(k+1)\times(k+1)}$.

- Sub-modules form a complete lattice of height $O(w{\cdot}k^2)$.

Algorithm for Computing Affine Relations

1) Compute a generating system G with:

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

   by a precise abstract interpretation.

2) Solve the linear equation system:

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

$\Rightarrow$ Need algorithms for:

1) Keeping generating systems in echelon form.
2) Solving (homogeneous) linear equation systems.

.

Theorem

1) The R-modules of matrices
   Span { $[\![\pi]\!]^T$ | π reaches v }
   can be computed using arithmetic in R.

2) The R-modules
   { $a \in R^{k+1}$ | affine relation a is valid at v }
   can be computed using arithmetic in R.

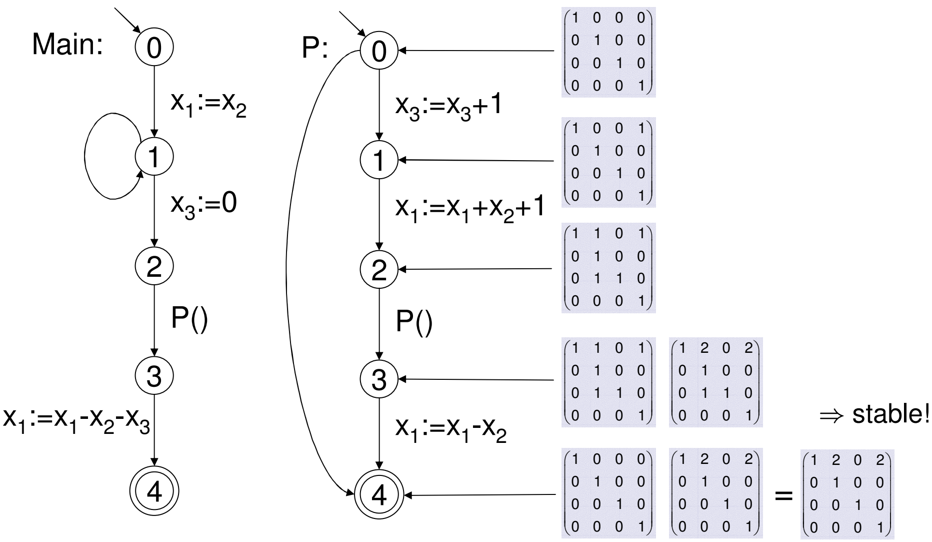
3) The time complexity is linear in the program size and polynomial in the number of variables (unit cost measure!):

   e.g. $\mathcal{O}(n \cdot k^8)$ for $R = \mathbb{Q}$

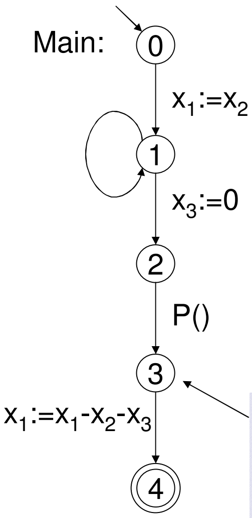
   (n size of the program, k number of variables)

4) We do not know how to avoid exponential growth of number sizes in interprocedural analysis for $R \in \{\mathbb{Q}, \mathbb{Z}\}$.
   However: we can avoid exponential growth in intra-procedural algorithms !

An Example



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

- Local variables, value parameters, return values
- Computing polynomial relations of degree $\leq d$
- Affine pre-conditions

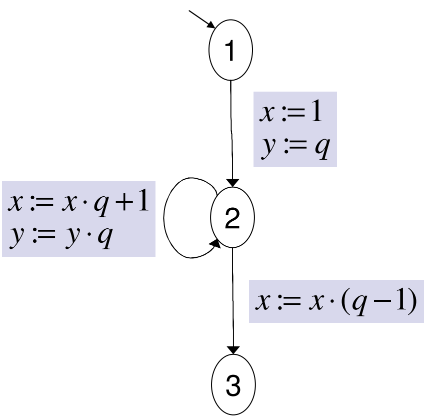
Precise Analysis through Algebra

- Algebra
  - Polynomial rings, ideals, Gröbner bases, …
  - Hilbert´s Basis Theorem ensures termination.

- Polynomial programs (over $\mathbb{Q}$):
  - Polynomial assignments: x := xy – 5z
  - Negated polynomial guards: ¬ (xy – 3z = 0)
  - The rest as for affine programs !

- Intraprocedural computation of „polynomial constants“ [MO/Seidl 2002]
- Intraprocedural derivation of all valid polynomial relations of degree $\leq d$ [MO/Seidl 2003]

A Polynomial Program



After *n* iterations at 2:

$$x = \sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1} \quad \text{(Horner´s method)}$$

$$y = q^{n+1}$$

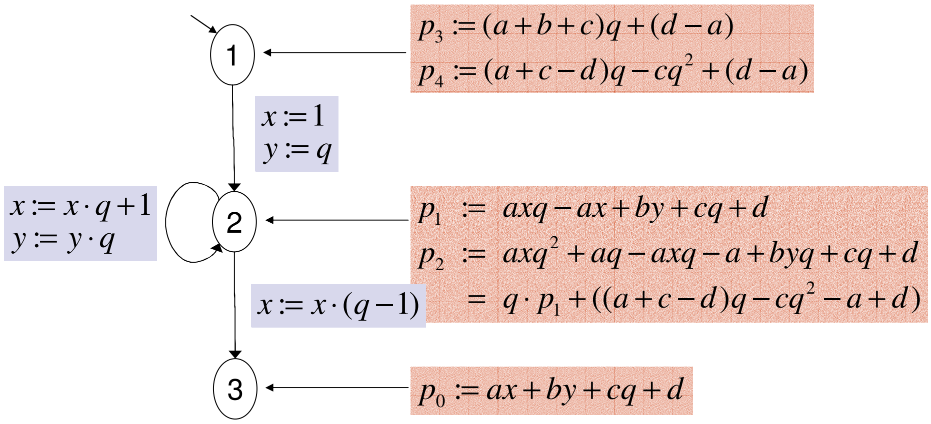
$$\Rightarrow \quad x \cdot (q-1) = y - 1$$

$$\Rightarrow \quad x \cdot q - x - y + 1 = 0$$

At 3:

$$x - y + 1 = 0$$

Computing Polynomial Relations



$$\begin{array}{lll} a+b+c=0 & d-a=0 & \\ a+c-d=0 & c=0 & d-a=0 \end{array} \quad \Leftrightarrow \quad a=d=-b \qquad c=0$$

$\Rightarrow$ All identities of the form $ax - ay + a = 0$ are valid. ☺

Conclusion

- Program analysis very broad topic
- Provides generic analysis techniques
- Some topics not covered:
  - Analyzing pointers and heap structures
  - Automata-theoretic methods
  - (Software) model checking
  - ...