Frameworks for

Interprocedural Analysis

Helmut Seidl
Vesal Vojdani
Kalmer Apinis

MOD 2013

Frameworks for

Interprocedural Analysis

The Goblint Approach

Helmut Seidl

Vesal Vojdani

Kalmer Apinis

MOD 2013

Analyzer Goblint

Certify absence of concurrency bugs in C !

Automotive

Avionics

Analyzer Goblint

Certify absence of concurrency bugs in C !

Automotive

Avionics

Challenges

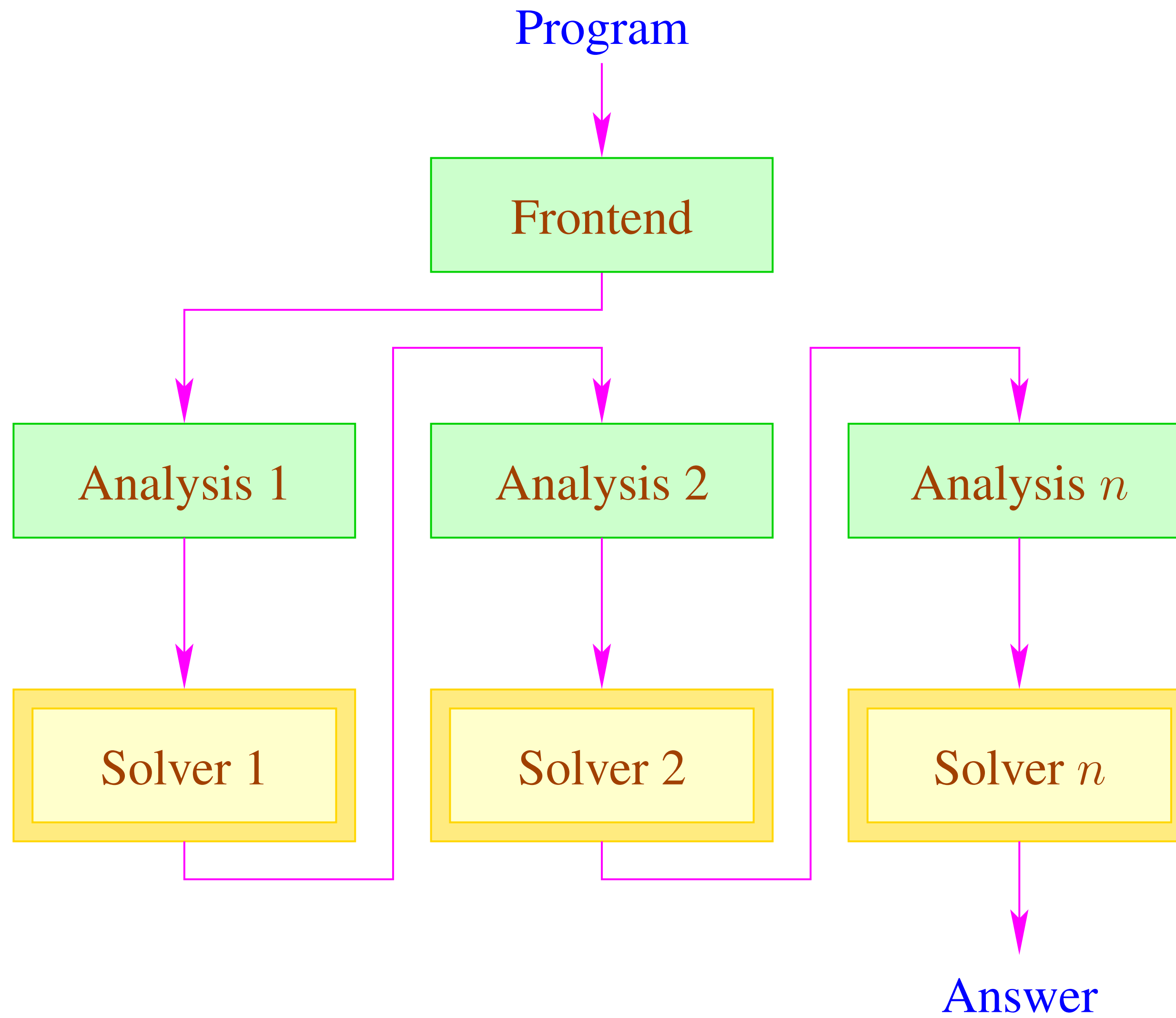
$\Longrightarrow$ Control-flow may depend on data.

$\Longrightarrow$ Call-graph may depend on pointer analysis.

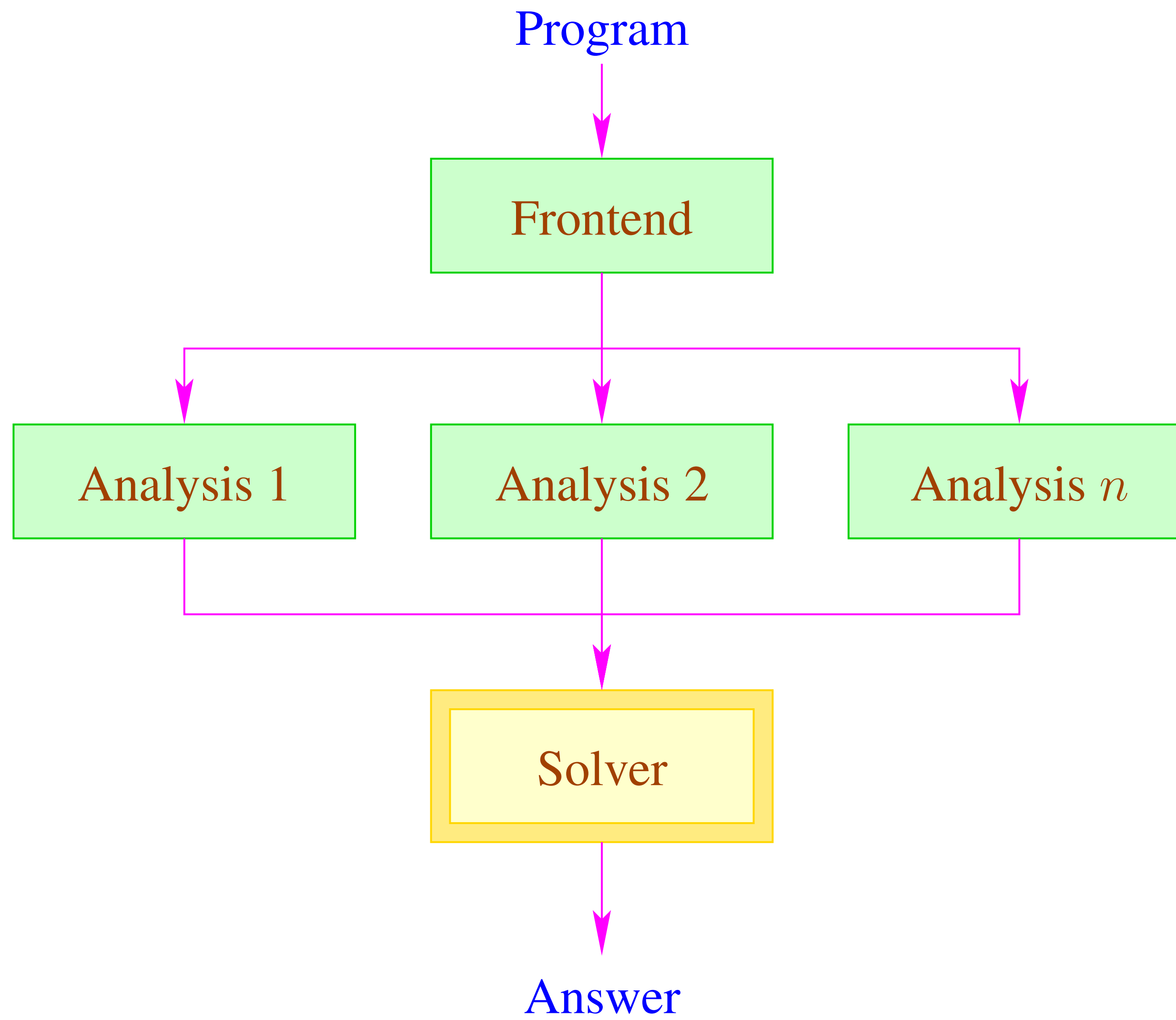
$\Longrightarrow$ Multi-threading.

$\Longrightarrow$ Decent scalability.

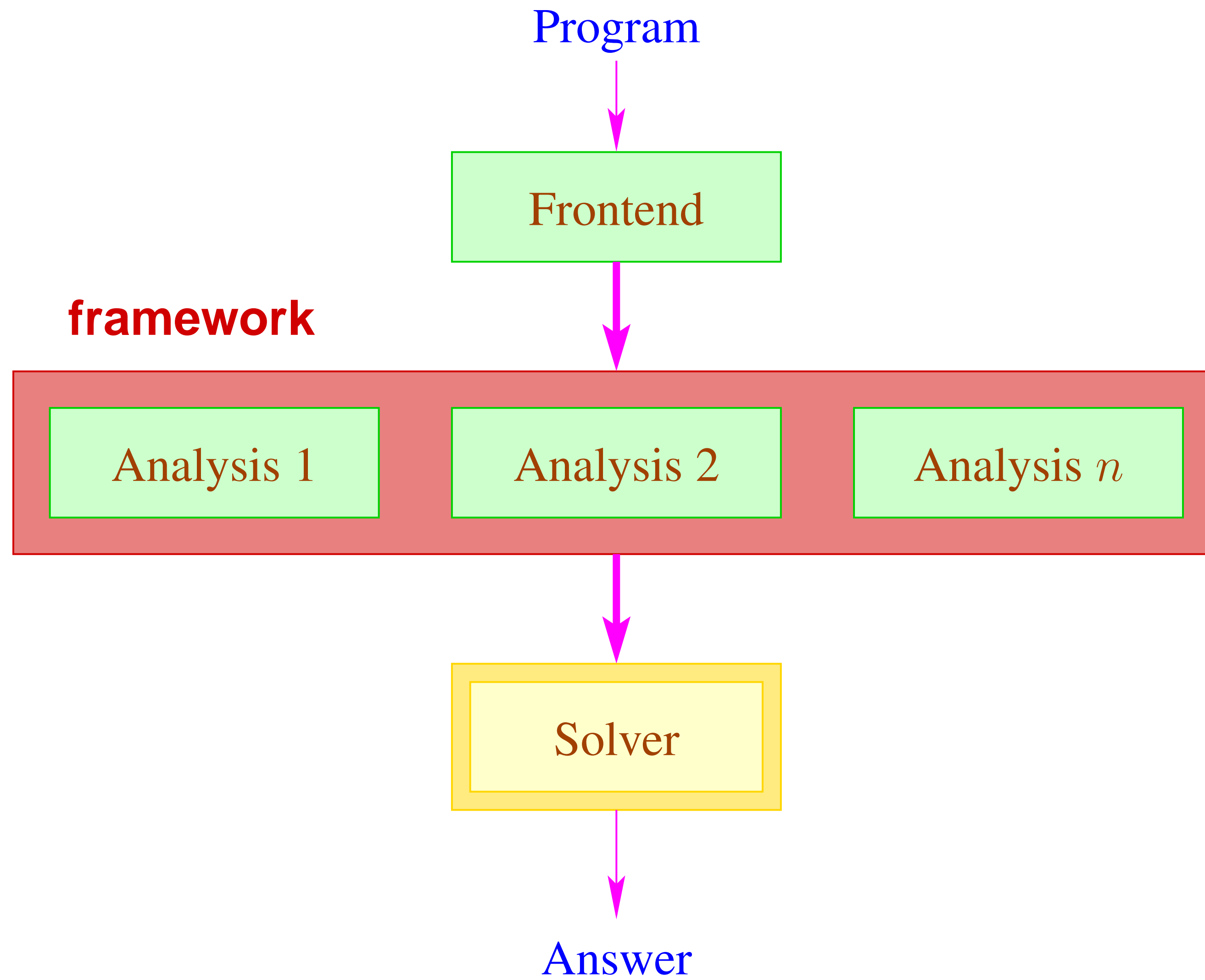
Architecture:



Dream:



Goblint:



Organization

Part 1: Constraint Systems

Part 2: Widening and Narrowing

Organization

Part 1: Constraint Systems

Part 2: Widening and Narrowing

Constraint Systems

Overview — The Problem

- Context-insensitive Analysis

- Context-sensitive Analysis

- Partially Context-sensitive Analysis

...

Overview — The Problem

- Context-insensitive Analysis

  finite constraint systems

- Context-sensitive Analysis

- Partially Context-sensitive Analysis

  ...

Overview — The Problem

- Context-insensitive Analysis

  finite constraint systems

- Context-sensitive Analysis

  infinite constraint systems

- Partially Context-sensitive Analysis

...

Overview — The Problem

- Context-insensitive Analysis

  finite constraint systems

- Context-sensitive Analysis

  infinite constraint systems

- Partially Context-sensitive Analysis

  infinite constraint systems, infinite dependences

...

Overview — Our Solution

- Side-effecting Constraint Systems
- Sharir-Pnueli
- Global Invariants
- Some Experiments
- Lessons learnt

Context-insensitive Analysis

- Represent program by control-flow graph
- Compute one abstract value per program point

$$
\begin{array}{lcll}
a_0 & : & \mathbb{D} & \text{value at program start} \\
[\![s]\!]^\sharp & : & \mathbb{D} \to \mathbb{D} & \text{abstract semantics}
\end{array}
$$

$$
\mathcal{D}[v] \quad == \quad \text{value at program point } v
$$

The Corresponding Constraint System:

$$\begin{array}{lllll} \mathcal{D}[v] & \sqsupseteq & a_0 & v & \text{entry point} \\ \mathcal{D}[v] & \sqsupseteq & [\![s]\!]^\sharp\,(\mathcal{D}[u]) & (u, s, v) & \text{edge} \end{array}$$

Discussion

- The constraint system is finite.
- Right-hand sides are monotonic.
- Dependences between unknowns are explicitly given by the control-flow graph.

Discussion

- The constraint system is finite.
- Right-hand sides are monotonic.
- Dependences between unknowns are explicitly given by the control-flow graph

$\Longrightarrow$ any fixpoint algo will do !

Context-sensitive Analysis

For each procedure, determine a summary:

$$\mathbb{D} \to \mathbb{D}$$

that describes the abstract effect.

Context-sensitive Analysis

For each procedure, determine a summary:

$$\mathbb{D} \to \mathbb{D}$$

that describes the abstract effect.

Problem

- Practical summaries are extremely complicated.
- No succinct representations.
- There are infinite ascending chains.

Idea

Cousot '77, Sharir/Pnueli '81

Analyze each procedure f for each abstract context a separately !

$\mathcal{D}[v, a] \in \mathbb{D}$ $\quad = \quad$ value at program point v within a call for context a .

Idea

Cousot '77, Sharir/Pnueli '81

Analyze each procedure f for each abstract context a separately !

$$\mathcal{D}[v, a] \in \mathbb{D} \qquad == \qquad \text{value at program point } v \text{ within a call for context } a \,.$$

Auxiliary Functions:

$\mathsf{enter}^\sharp$	:	$\mathbb{D} \to \mathbb{D}$	passing of parameters
$\mathsf{combine}^\sharp$	:	$\mathbb{D} \to \mathbb{D} \to \mathbb{D}$	returning of results

The Constraint System:

$$\mathcal{D}[st_f, a] \sqsupseteq a \qquad st_f \quad \text{entry point of } f$$

$$\mathcal{D}[v, a] \sqsupseteq [\![s]\!]^\sharp (\mathcal{D}[u, a]) \qquad (u, s, v) \quad \text{edge}$$

The Constraint System:

$$
\begin{array}{lll}
\mathcal{D}[st_f, a] & \sqsupseteq & a \\
\mathcal{D}[v, a] & \sqsupseteq & [\![s]\!]^\sharp\,(\mathcal{D}[u, a]) \\
\mathcal{D}[v, a] & \sqsupseteq & \mathsf{combine}^\sharp\,(\mathcal{D}[u, a])(\mathcal{D}[rt_f, \mathsf{enter}^\sharp\,(\mathcal{D}[u, a])])
\end{array}
$$

st_f entry point of f

(u, s, v) edge

$(u, f\,(), v)$ call

rt_f return point of f

Problem

- This constraint system is generally infinite ?
- The dependences between unknowns change ?
- Right-hand sides are no longer monotonic ?

Problem

- This constraint system is generally infinite ?
- The dependences between unknowns change ?
- Right-hand sides are no longer monotonic ?

Idea

- Do not solve it completely !
- Analyse only calls that are queried when determining the value of

$$\mathcal{D}[rt_{\mathsf{main}}, \mathsf{enter}^\sharp\, a_0]$$

Problem

- This constraint system is generally infinite ?
- The dependences between unknowns change ?
- Right-hand sides are no longer monotonic ?

Idea

- Do not solve it completely !
- Analyse only calls that are queried when determining the value of

$$\mathcal{D}[rt_{\mathsf{main}}, \mathsf{enter}^\sharp\, a_0]$$

$\Longrightarrow$ local solving

Generic Local Solving — Summary

- Start with the exploration of some interesting unknown.
- Identify dependences on the fly !
- Explore values of unknowns on demand !
- Use accumulation to deal with non-monotonicity !

Generic Local Solving — Summary

- Start with the exploration of some interesting unknown.
- Identify dependences on the fly !
- Explore values of unknowns on demand !
- Use accumulation to deal with non-monotonicity !

Simplification

$$x \sqsupseteq f_x \qquad (x \in \mathsf{Unknowns})$$

The Generic Local Solver LR

- Start with the exploration of some interesting unknown.
- If during evaluation of the right-hand side f_x of x, an unknown y is accessed, y is first solved recursively. Then x is added to $\mathsf{infl}[y]$ ...

$$\begin{array}{lll} \mathsf{eval}\ x\ y & = & \mathsf{solve}\, y; \\ & & \mathsf{infl}[y] = \mathsf{infl}[y] \cup \{x\}; \\ & & D[y]; \end{array}$$

$\Longrightarrow$ detection of dependences on the fly !!

The Function solve :

$$
\begin{array}{l}
\textsf{solve}\ x \;=\; \\
\qquad t = f_x\,(\textsf{eval}\ x); \quad t = D[x] \sqcup t; \\
\qquad \textsf{if}\ (t \neq D[x])\ \{ \\
\qquad\qquad W = \textsf{infl}[x]; \\
\qquad\qquad D[x] = t; \\
\\
\qquad\qquad \textsf{iter}\ \textsf{solve}\ W; \\
\qquad \} \\
\}
\end{array}
$$

Preventing Non-termination

- Recursion will cause infinite descent into unknowns and thus result in non-termination ??
- In order to prevent that, a set Stable of unknown is maintained for which solve just looks up their values

  ...
- Initially, $\mathsf{Stable} = \emptyset$.

The Function solve :

$$
\begin{array}{lll}
\mathsf{solve}\ x & = & \mathsf{if}\ (x \notin \mathsf{Stable})\ \{ \\
& & \quad \mathsf{Stable} = \mathsf{Stable} \cup \{x\}; \\
& & \quad t = f_x\,(\mathsf{eval}\ \ x); \quad t = D[x] \sqcup t; \\
& & \quad \mathsf{if}\ (t \neq D[x])\ \{ \\
& & \qquad W \ = \mathsf{infl}[x]; \\
& & \qquad D[x] = t; \\
& & \qquad \mathsf{Stable} = \mathsf{Stable} \backslash W; \\
& & \qquad \mathsf{iter}\ \ \mathsf{solve}\ \ W; \\
& & \quad \} \\
& & \}
\end{array}
$$

The Function solve :

$$
\begin{array}{lll}
\mathsf{solve}\ x & = & \mathsf{if}\ (x \notin \mathsf{Stable})\ \{ \\
& & \quad \mathsf{Stable} = \mathsf{Stable} \cup \{x\}; \\
& & \quad t = f_x\,(\mathsf{eval}\ \ x); \quad t = D[x] \sqcup t; \\
& & \quad \mathsf{if}\ (t \neq D[x])\ \{ \\
& & \qquad W = \mathsf{infl}[x]; \quad \mathsf{infl}[x] = \emptyset; \\
& & \qquad D[x] = t; \\
& & \qquad \mathsf{Stable} = \mathsf{Stable} \backslash W; \\
& & \qquad \mathsf{iter}\ \ \mathsf{solve}\ \ W; \\
& & \quad \} \\
& & \}
\end{array}
$$

Example:

Consider the system:

$$\begin{array}{lll} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a, b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

A trace of the fixpoint algorithm then looks as follows:

solve x_2 eval x_2 x_3 solve x_3 eval x_3 x_1 solve x_1 eval x_1 x_3 solve x_3

stable!

infl$[x_3] = \{x_1\}$

$\Rightarrow \quad \emptyset$

$D[x_1] = \{a\}$

infl$[x_1] = \{x_3\}$

$\Rightarrow \quad \{a\}$

$D[x_3] = \{a, c\}$

infl$[x_3] = \emptyset$

solve x_1 eval x_1 x_3 solve x_3

stable!

infl$[x_3] = \{x_1\}$

$\Rightarrow \quad \{a, c\}$

$D[x_1] = \{a, c\}$

infl$[x_1] = \emptyset$

solve x_3 eval x_3 x_1 solve x_1

stable!

infl$[x_1] = \{x_3\}$

$\Rightarrow \quad \{a, c\}$

ok

infl$[x_3] = \{x_1, x_2\}$

$\Rightarrow \quad \{a, c\}$

$D[x_2] = \{a\}$

Generic Local Solving — Discussion

- It is reasonably efficient !
- It terminates if $\mathbb{D}$ has only finite ascending chains and only finitely many contexts are encountered.

Generic Local Solving — Discussion

- It is reasonably efficient !
- It terminates if $\mathbb{D}$ has only finite ascending chains and only finitely many contexts are encountered.
- It does not require pre-processing of constraint system !

Generic Local Solving — Discussion

- It is reasonably efficient !
- It terminates if $\mathbb{D}$ has only finite ascending chains and only finitely many contexts are encountered.
- It does not require pre-processing of constraint system !
- Solver LR is small, but intricate $\Longrightarrow$ verification

Hofmann, Karbyshev, S. 2010

So far:

- Context-insensitive analysis
- Context-sensitive analysis

Specification

complete lattice

unknowns

constraints

Solving

Local fixpoint algorithm LR

Incredients of Spec

$\mathbb{D}$ —— complete lattice of invariants

- $\bot$ least element – unreachability
- $\top$ greatest element – don't know
- $\sqsubseteq$ ordering
- $\sqcup$ least upper bound

$[\![i = j - 1;]\!]^\sharp$ —— abstract semantics

transfer function $\mathbb{D} \to \mathbb{D}$

should preserve $\bot$

Incredients of Spec (cont.)

Unkowns

$\mathcal{D}[v]$ — invariant at program point v

// context-insensitive analysis

$\mathcal{D}[v, a]$ — invariant at program point v

for context a

// context-sensitive analysis

Constraints $x \sqsupseteq f$

f : (Unknowns $\to \mathbb{D}$) $\to \mathbb{D}$

Disadvantage of LR

Reachability information only implicitly given ...

Disadvantage

Reachability information only implicitly given by:

$$\mathsf{Reach}[v] = \bigsqcup \{\mathcal{D}[v, a] \mid a \quad \text{encountered context}\}$$

Disadvantage

Reachability information only implicitly given by:

$$\mathsf{Reach}[v] = \bigsqcup \{\mathcal{D}[v, a] \mid a \quad \text{encountered context}\}$$

Explicit Reachability

$$\mathcal{D}[st_{\mathsf{main}}, a] \quad \sqsupseteq \quad a \qquad\qquad a = \mathsf{enter}^\sharp\, a_0$$

$$\mathcal{D}[st_f, a] \quad \sqsupseteq \quad \bigsqcup \{a \mid a = \mathsf{enter}^\sharp\, (\mathcal{D}[u, a'])\}$$

$$f \quad \text{procedure}$$

Disadvantage

Reachability information only implicitly given by:

$$\mathsf{Reach}[v] = \bigsqcup\{\mathcal{D}[v, a] \mid a \quad \text{encountered context}\}$$

Explicit Reachability

$$\begin{array}{lll} \mathcal{D}[st_{\mathsf{main}}, a] & \sqsupseteq & a \qquad\qquad a = \mathsf{enter}^\sharp\ a_0 \\ \mathcal{D}[st_f, a] & \sqsupseteq & \bigsqcup\{a \mid a = \mathsf{enter}^\sharp\ (\mathcal{D}[u, a'])\} \\ & & \qquad\qquad f \quad \text{procedure} \end{array}$$

$\Longrightarrow$ Dependences have become infinite !?

Partially Context-sensitive Analysis

Assume that

$$\mathbb{D} = \mathbb{A} \times \mathbb{B}$$

where only $\mathbb{B}$ should go into the context.

Partially Context-sensitive Analysis

Assume that

$$\mathbb{D} = \mathbb{A} \times \mathbb{B}$$

where only $\mathbb{B}$ should go into the context.

- Analyze each procedure f for each b only !
- Accumulate occurring contributions to the rest !

$\mathcal{D}[v, b]$ $=\!=$ value at program point v within calls for contexts (a, b), $a \in \mathbb{A}$.

Example Contexts

- First parameter,
- Parameters of certain types,
- Extra information such as held locks, opened files ...
- Abstraction of the call-stack

$\Longrightarrow$ call-strings

The Constraint System:

$$\mathcal{D}[st_{\mathsf{main}}, b] \quad \sqsupseteq \quad (a, b) \qquad\qquad (a, b) = \mathsf{enter}^\sharp\ (a_0, b_0)$$

$$\mathcal{D}[st_f, b] \quad \sqsupseteq \quad \bigsqcup\{(a, b) \mid (a, b) = \mathsf{enter}^\sharp\ (\mathcal{D}[u, b'])\}$$

f procedure

The Constraint System:

$$
\begin{array}{lll}
\mathcal{D}[st_{\mathsf{main}}, b] & \sqsupseteq & (a, b) \qquad\qquad (a, b) = \mathsf{enter}^\sharp\,(a_0, b_0) \\
\mathcal{D}[st_f, b] & \sqsupseteq & \bigsqcup\{(a, b) \mid (a, b) = \mathsf{enter}^\sharp\,(\mathcal{D}[u, b'])\} \\
 & & \qquad\qquad f \quad \text{procedure} \\
\mathcal{D}[v, b] & \sqsupseteq & [\![s]\!]^\sharp\,(\mathcal{D}[u, b]) \qquad (u, s, v) \quad \text{edge} \\
\mathcal{D}[v, b] & \sqsupseteq & \mathsf{combine}^\sharp\,(\mathcal{D}[u, b])(\mathcal{D}[rt_f, \pi_2(\mathsf{enter}^\sharp\,(\mathcal{D}[u, b]))]) \\
 & & \qquad\qquad (u, f\,(), v) \quad \text{call}
\end{array}
$$

Discussion

- Explicit tracking of reachability is now mandatory !
- Since the system is infinite, ordinary fixpoint iteration fails.
- Since potential dependences between unknowns are infinite, local solving fails as well ?

Discussion

- Explicit tracking of reachability is now mandatory !
- Since the system is infinite, ordinary fixpoint iteration fails.
- Since potential dependences between unknowns are infinite, local solving fails as well ?

$\Longrightarrow$ The system does not support solving, but may serve as a specification to be proven correct w.r.t. the concrete semantics

Side-effecting Constraint Systems

Re-formulating Partial Context-Sensitivity

- Remove critical constraints for entry points of procedures !
- Generate the contributions to entry points of procedures when they occur ...

Idea

Constraints now take the form $\quad (x, f) \quad$ where:

$$f : (\mathsf{Unknowns} \to \mathbb{D}) \to (\mathsf{Unknowns} \to \mathbb{D} \to \mathbf{void}) \to \mathbb{D}$$

Idea

Constraints now take the form $\quad (x, f) \quad$ where:

$$f : (\mathsf{Unknowns} \to \mathbb{D}) \to (\mathsf{Unknowns} \to \mathbb{D} \to \textbf{void}) \to \mathbb{D}$$

When evaluating $\quad f\ \mathsf{get}\ \mathsf{set}$,

- the argument get returns for every variable x, the current value $\mathsf{get}\, x \in \mathbb{D}$;

Idea

Constraints now take the form (x, f) where:

$$f : (\text{Unknowns} \to \mathbb{D}) \to (\text{Unknowns} \to \mathbb{D} \to \textbf{void}) \to \mathbb{D}$$

When evaluating f get set,

- the argument get returns for every variable x, the current value get $x \in \mathbb{D}$;
- the argument set when called for $x \in$ Unknowns and $d \in \mathbb{D}$ allows to produce as a side effect the contribution d to x.

The Constraint System with Side Effects:

$$\mathcal{D}[st_{\mathsf{main}}, b] \quad \sqsupseteq \quad (a, b) \qquad\qquad (a, b) = \mathsf{enter}^\sharp\,(a_0, b_0)$$

$$\mathcal{D}[v, b] \quad \sqsupseteq \quad [\![s]\!]^\sharp\,(\mathsf{get}\,[u, b]) \qquad (u, s, v) \quad \text{edge}$$

The Constraint System with Side Effects:

$$\mathcal{D}[st_{\mathsf{main}}, b] \sqsupseteq (a, b) \qquad (a, b) = \mathsf{enter}^\sharp\ (a_0, b_0)$$

$$\mathcal{D}[v, b] \sqsupseteq [\![s]\!]^\sharp\,(\mathsf{get}\ [u, b]) \qquad (u, s, v) \quad \text{edge}$$

$$\begin{array}{lll} \mathcal{D}[v, b] & \sqsupseteq & \textbf{let}\ (a', b') = \mathsf{enter}^\sharp\ (\mathsf{get}\ [u, b])\ \textbf{in} \\ & & \textbf{let}\ () = \mathsf{set}\ [st_f, b']\ (a', b') \\ & & \textbf{in} \quad \mathsf{combine}^\sharp\,(\mathsf{get}\ [u, b])(\mathsf{get}\ [rt_f, b']) \end{array}$$

$$(u, f\ (), v) \quad \text{call}$$

Results

- The new constraint system is equivalent to the original one

  $\Longrightarrow$ every solution satisfies the specification !

- Each constraint depends only on few variables !

Results

- The new constraint system is equivalent to the original one

  $\Longrightarrow$ every solution satisfies the specification !

- Each constraint depends only on few variables

  $\Longrightarrow$

  Local solving, e.g., by LR, enhanced with side effects, can be used as solver !

Partial-Context Sharir/Pnueli

- Sharir/Pnueli's algorithm is operationally different from Cousot's.
- SPA propagates changes forward through the control-flow graph.
- At a call, the graph is expanded by a new copy ...

Forward propagation, though, can be simulated by local solving and side effects ...

The SPA Constraint System

$$
\begin{array}{lcll}
\mathcal{D}[st_{\mathsf{main}}, b] & \sqsupseteq & (a, b) & (a, b) = \mathsf{enter}^\sharp\ (a_0, b_0) \\
\mathcal{D}[u, b] & \sqsupseteq & \mathbf{let}\ () = \mathsf{set}\ [v, b]\ ([\![s]\!]^\sharp\ (\mathsf{get}\ [u, b])) & \\
 & & \mathbf{in}\quad \bot & (u, s, v) \quad \text{edge}
\end{array}
$$

The SPA Constraint System

$$\mathcal{D}[st_{\mathsf{main}}, b] \sqsupseteq (a,b) \qquad (a,b) = \mathsf{enter}^\sharp\ (a_0, b_0)$$

$$\begin{array}{lll}\mathcal{D}[u,b] & \sqsupseteq & \textbf{let}\ () = \mathsf{set}\ [v,b]\ (\llbracket s \rrbracket^\sharp\ (\mathsf{get}\ [u,b])) \\ & & \textbf{in}\quad \bot \qquad\qquad (u,s,v) \quad \text{edge}\end{array}$$

$$\begin{array}{lll}\mathcal{D}[u,b] & \sqsupseteq & \textbf{let}\ (a',b') = \mathsf{enter}^\sharp\ (\mathsf{get}\ [u,b])\ \textbf{in} \\ & & \textbf{let}\ () = \mathsf{set}\ [st_f, b']\ (a',b')\ \textbf{in} \\ & & \textbf{let}\ () = \mathsf{set}\ [v,v]\ (\mathsf{combine}^\sharp\ (\mathsf{get}\ [u,b])(\pi_2(\mathsf{get}\ [rt_f, b']))) \\ & & \textbf{in}\quad \bot \\ & & \qquad\qquad (u, f\ (), v) \quad \text{call}\end{array}$$

Discussion

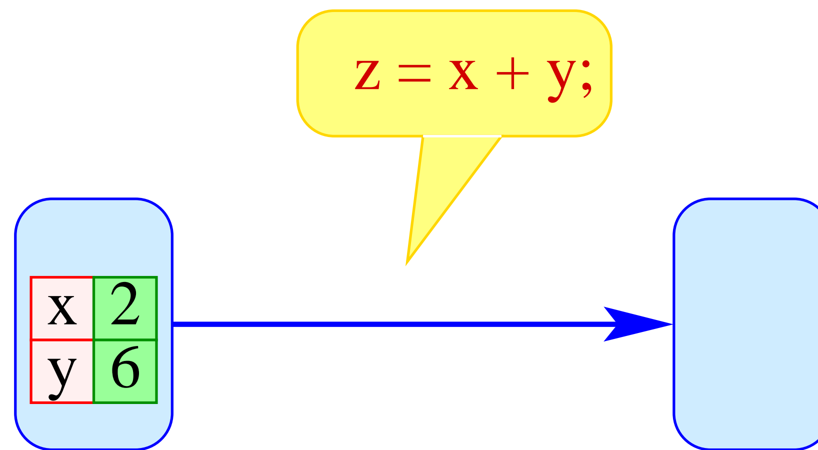
- Local solving now starts with querying $\mathcal{D}[st_{\textsf{main}}, b]$, where $(_, b) = \textsf{enter}^\sharp\ (a_0, b_0)$.
- Besides in the initial constraint for $\mathcal{D}[st_{\textsf{main}}, b]$, each right-hand side just returns $\bot$.

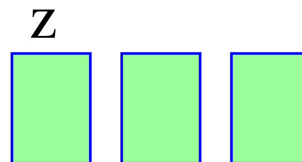
Discussion

- Local solving now starts with querying $\mathcal{D}[st_{\mathsf{main}}, b]$, where $(_, b) = \mathsf{enter}^\sharp\ (a_0, b_0)$.
- Besides in the initial constraint for $\mathcal{D}[st_{\mathsf{main}}, b]$, each right-hand side just returns $\bot$.
- Subsequent program points receive their values through side effects

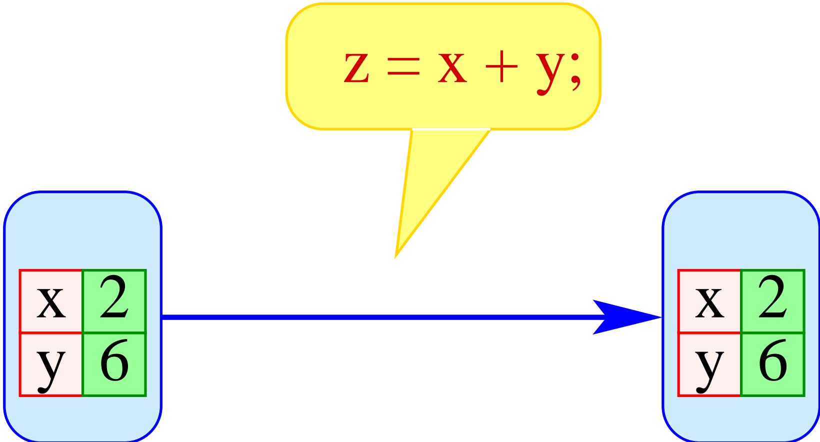
  $\Longrightarrow$ well-suited for the analysis of binary code !
- A quite non-standard formulation: Yet it enables to realize SPA by means of generic local solver :-)

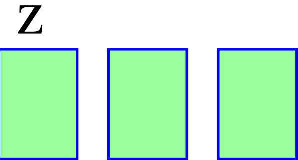
Flow-insensitive Invariants



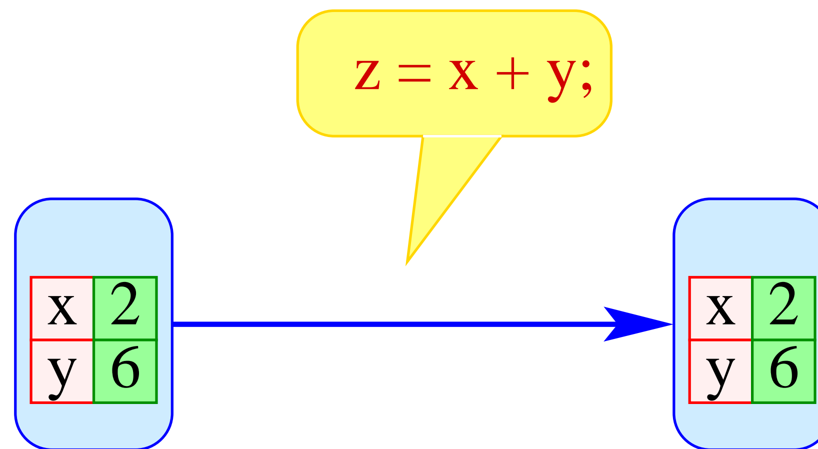


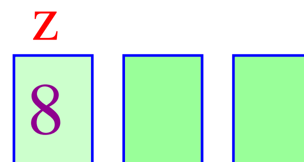
Flow-insensitive Invariants





Flow-insensitive Invariants





Applications

- points-to information
- thread-safe information about globals

Applications

- points-to information
- thread-safe information about globals

$\Longrightarrow$ de-couples threads

$\Longrightarrow$ reduces thread analysis to sequential analysis !

Approach

- Only those contributions should be summarized which occur at reachable configurations.
- These become known only during fixpoint computation.

$\Longrightarrow$ constraints with side effects

Experimental Evaluation

Goal

Accumulate for each global g, a set of definitely held mutexes when accessing g.

Experimental Evaluation

Goal

Accumulate for each global g, a set of definitely held mutexes when accessing g.

Prerequisites

- full constant propagation;
- may/must points-to analysis, ...
- context-sensitivity;
- side-effects to globals.

Benchmarks

aget multithreaded HTTP download accelerator

automount Autofs kernel-based automounter for Linux

ctrace C tracing library sample program

knot Knot web-server

pfscan Parallel file scanner

smtprc Smtp relay checker

ypbind Linux NIS daemon

zfs-fuse ZFS filesystem for FUSE/Linux

Sizes: 1280 – 24097 LoC

Setup

- Cousot-style vs. SPA
- no/relevant/full context

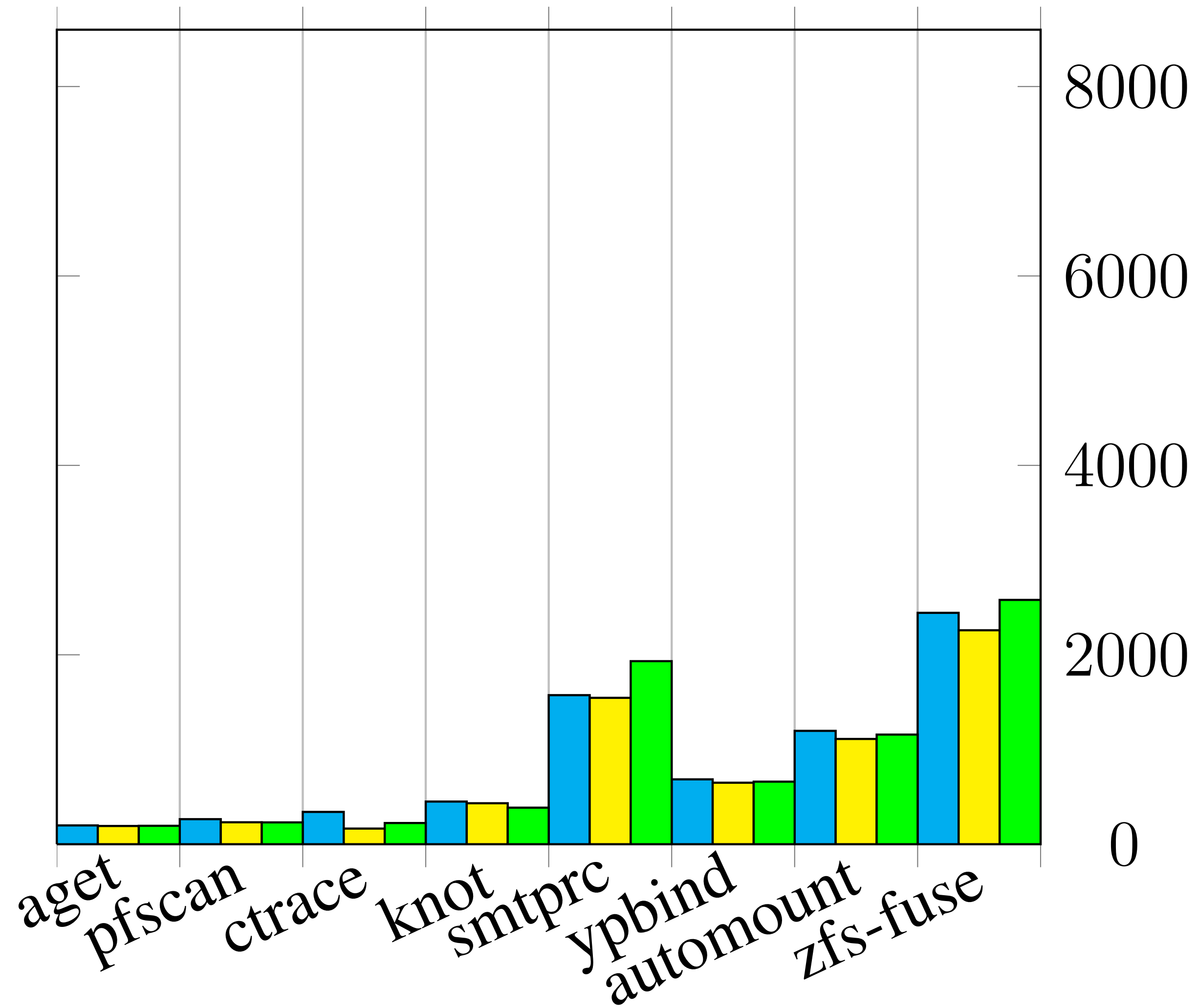
Setup

- Cousot-style vs. SPA
- no/relevant/full context

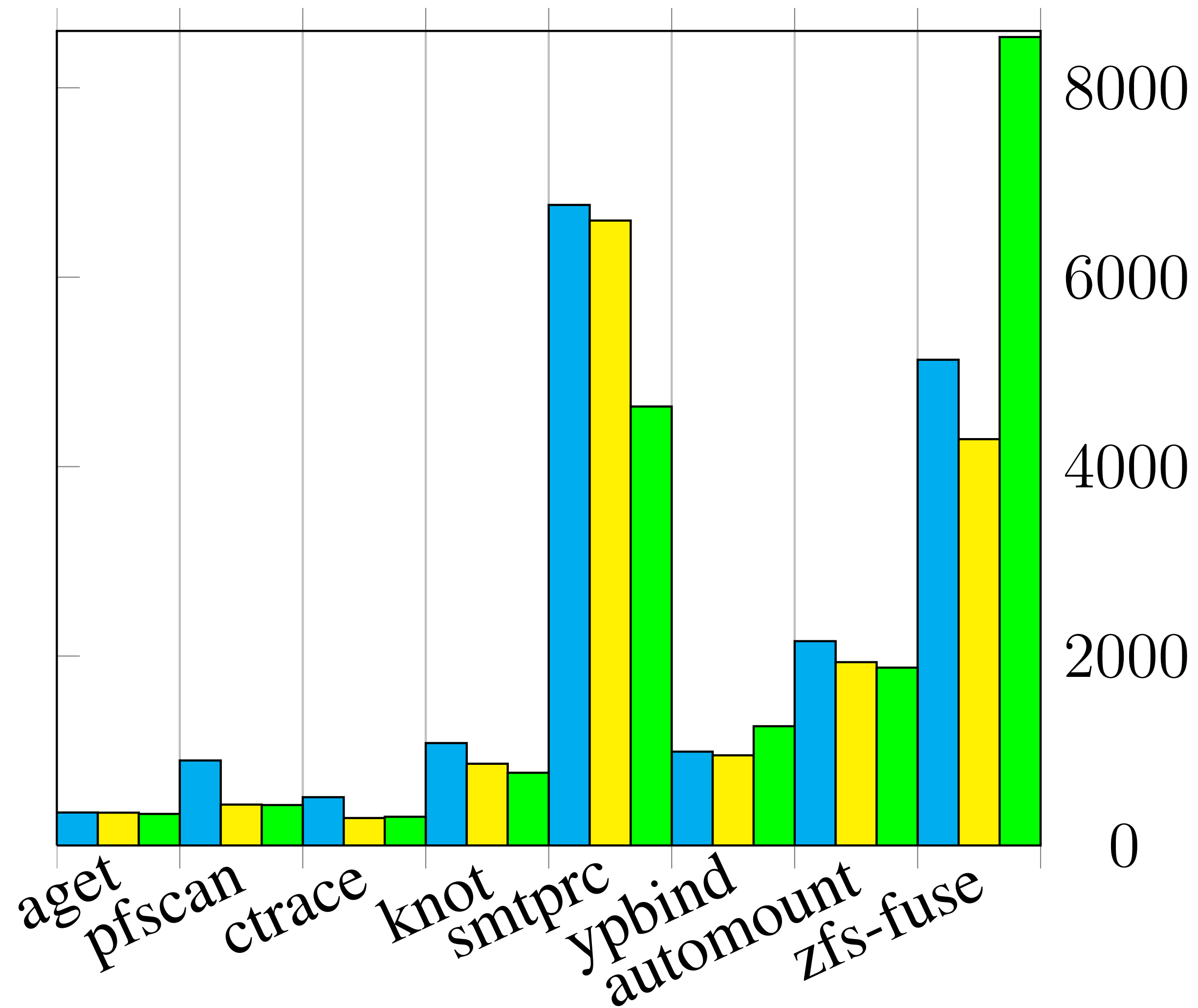
Results

- Overall, run-times are decent (3.5s for 24loc)
- For a fair comparison, we count evaluations of right-hand sides.

Cousot-style — # rhs



SPA — # rhs



Precision Results — Warnings

Name	Size	none	half	full
aget	1280	162	162	162
pfscan	1295	72	72	72
ctrace	1407	87	79	79
knot	2255	140	62	62
smtprc	5787	1068	636	636
ypbind	6596	251	244	244
automount	20624	505	480	480
zfs-fuse	24097	2319	2318	2318

Lessions learnt

- Static analyzers are complex software systems which themselves may contain bugs.
- A modular design allows to separate:
  - specification
  - solving

Lessions learnt

- Static analyzers are complex software systems which themselves may contain bugs.
- A modular design allows to separate:
  - specification
  - solving
- Side-effecting constraint systems are a convenient formalism to freely combine
  - flow-insensitive,
  - flow-sensitive analysis,
  - fine-tune the amount of tracked contexts.

Lessions learnt (cont.)

- Side-effecting constraint systems all can be solved by one dedicated local fixpoint algorithm.

- To ensure reliability of results, we provided:
  - a correctness proof of the algorithm
  - a fixpoint checker for analysis results
  - a performance checker for the overall system

Lessions learnt (cont.)

- The resulting implementation scales reasonably well.
- Some technical experiences:
  - No context is not necessarily faster than some context.
  - More context does not necessarily pay off.
  - Cousot is generally preferable to SPA.

goblint.in.tum.de