Part II: Static Program Analysis

COURSE NOTES

Master Parisien de Recherche en Informatique

Course 2–6: Abstract Interpretation: Application to Verification and Static Analysis

---

Static Inference of Numeric Invariants

by Abstract Interpretation

Antoine MINÉ

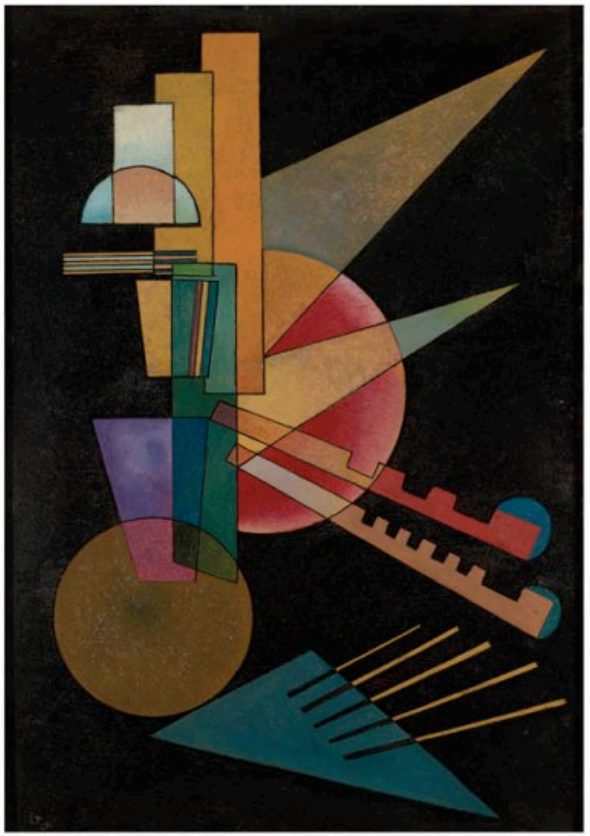
antoine.mine@lip6.fr

Université Pierre et Marie Curie, Paris, France

Draft 2018-08-27

Do not circulate

PRINCIPLES OF
ABSTRACT INTERPRETATION



PATRICK COUSOT

Static program analysis

From Wikipedia, the free encyclopedia

Static program analysis is the analysis of computer software that is performed without actually executing programs (analysis performed on executing programs is known as dynamic analysis).[1] In most cases the analysis is performed on some version of the source code and in the other cases some form of the object code. The term is usually applied to the analysis performed by an automated tool, with human analysis being called program understanding, program comprehension or code review.

Contents [hide]

Rationale [edit]

The sophistication of the analysis performed by tools varies from those that only consider the behavior of individual statements and declarations, to those that include the complete source code of a program in their analysis. Uses of the information obtained from the analysis vary from highlighting possible coding errors (e.g., the lint tool) to formal methods that mathematically prove properties about a given program (e.g., its behavior matches that of its specification).

Static Program Analysis

- Automatic techniques to infer at compile-time (thus statically) approximate information on run-time program behaviors

- Applications
  - Bug detection
  - Program verification
  - Code optimization
  - Program parallelization
  - Type inference
  - …

Program points

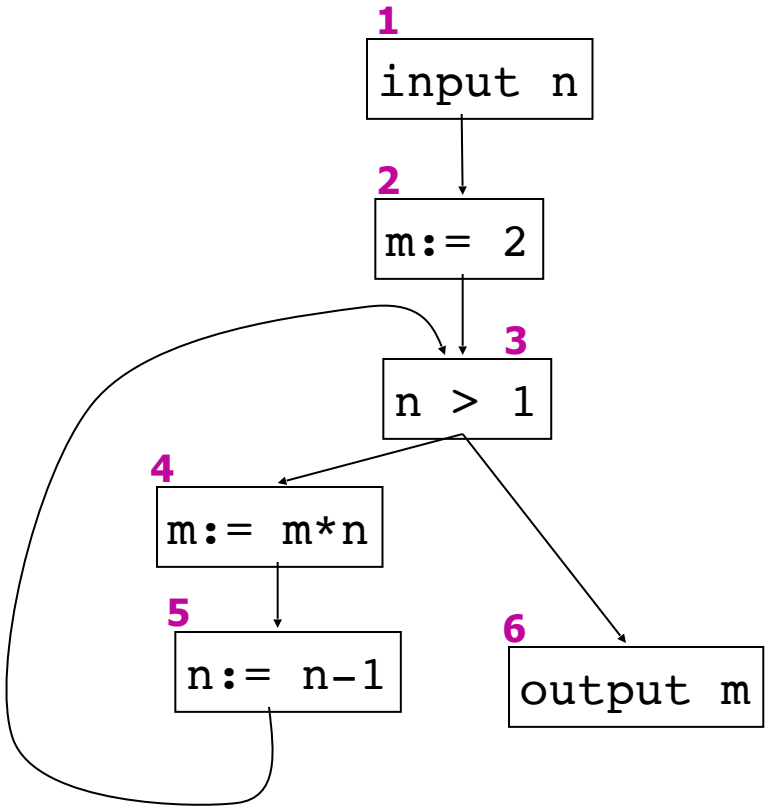
```
{n = k ∧ n≥0}

input n;
m:=2;
while n>1 do
  m:= m*n;
  n:= n-1;
output m;
{m = 2(k!)}
```

```
(1) input n;
(2) m:=2;
    while (3) n>1 do
     (4) m:= m*n;
     (5) n:= n-1;
(6) output m;
```

With program points

Control Flow Graph

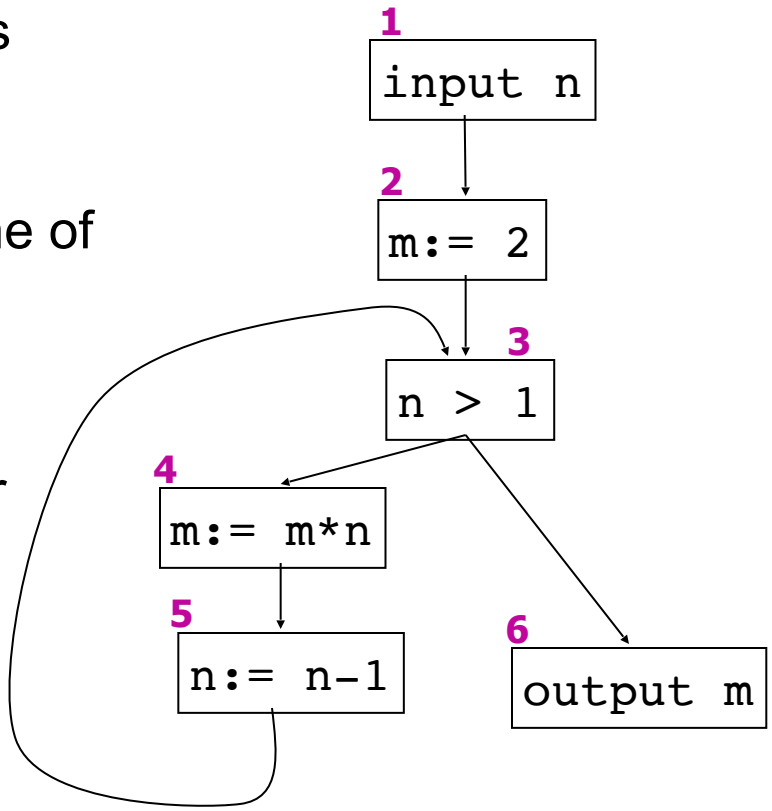
```
(1)  input n;
(2)  m:=2;
     while (3) n>1 do
      (4) m:= m*n;
      (5) n:= n-1;
(6)  output m;
```



- We can infer by static analysis that at program point (6) the value of variable m will be even for any input value n.

- In order to accomplish this, the analysis must propagate "**parity information**"

- We assign at each program variable one of these "values":
  - Even the value is even
  - Odd the value is odd
  - DontKnow the value can be either even or odd



```
(1)  input n;
(2)  m:= 2;
     while (3) n>1 do
      (4) m:= m * n;
      (5) n:= n - 1;
(6)  output m;
```

(1) n DontKnow
    m DontKnow

(2) n DontKnow
    m DontKnow

(3) n DontKnow
    m Even

(4) n DontKnow
    m Even

(5) n DontKnow
    m Even

(6) n DontKnow
    m Even

```
(1)  input n;
(2)  m:= 2;
     while (3) n>1 do
      (4) m:= m * n;
      (5) n:= n - 1;
(6)  output m;
```

Actually, for a given positive n, this program computes the double of the factorial of n, namely, 2*(n!)

- n=1 … output is 2
- n=2 … output is 4
- n=3 … output is 12
- n=4 … output is 48

even

```
(1)  input n;
(2)  m:= 1;
     while (3) n>1 do
      (4) m:= m * n;
      (5) n:= n - 1;
(6)  output m;
```

- For a given positive n, this program computes the factorial n!
- In this case, the previous analysis provides no useful information on the parity of m in (6)
- This analysis is **trivially correct**, since outputs DontKnow
- Even if the input is restricted to even n or n>1, the analysis is not precise: in (3) both n and m are DontKnow, although in (6) m will be always even.

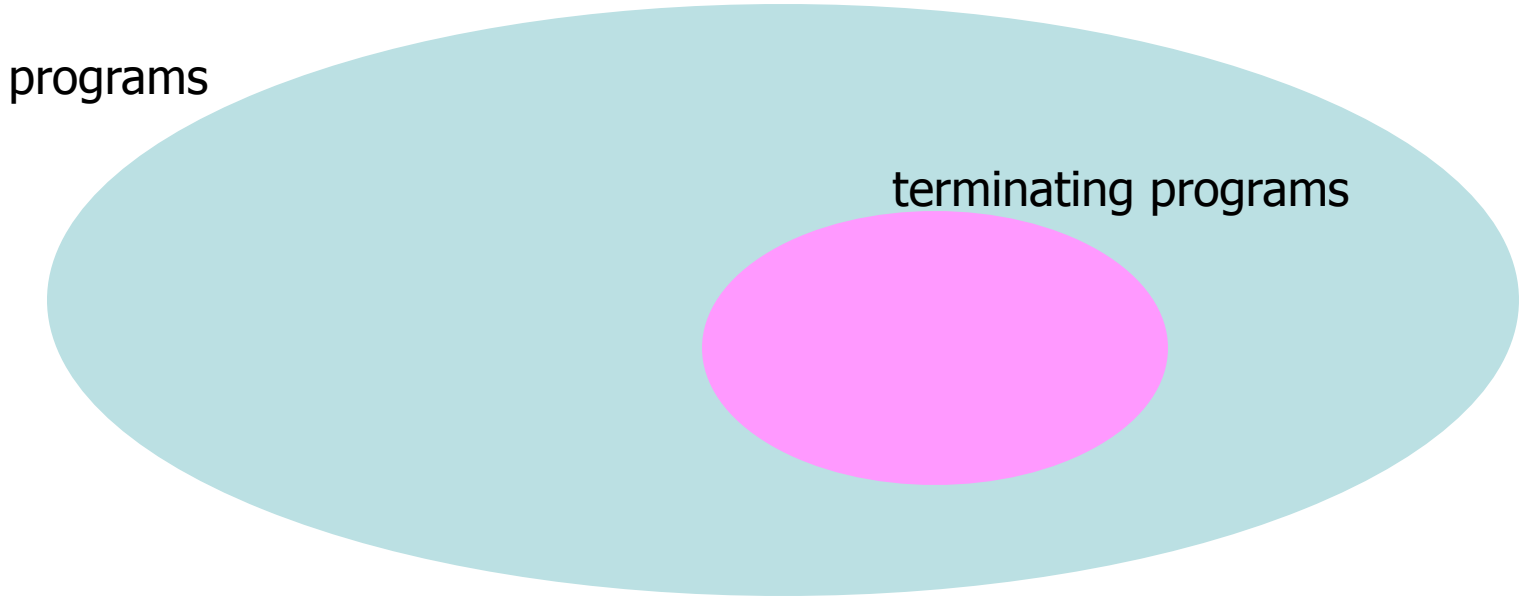
Precision and Correctness

- This loss of precision is a common attribute of static analysis techniques
- Most interesting dynamic properties of programs are undecidable (due to Rice's Theorem), thus we cannot automatically infer them with full precision
- Nevertheless we need to ensure the correctness (a.k.a. soundness) of the output of static analysis:
  - If the analysis **outputs** YES then the property is certainly verified
  - If the analysis **does not output** YES then it may happen that the property is not verified
    ⇒ ¬YES=DontKnow

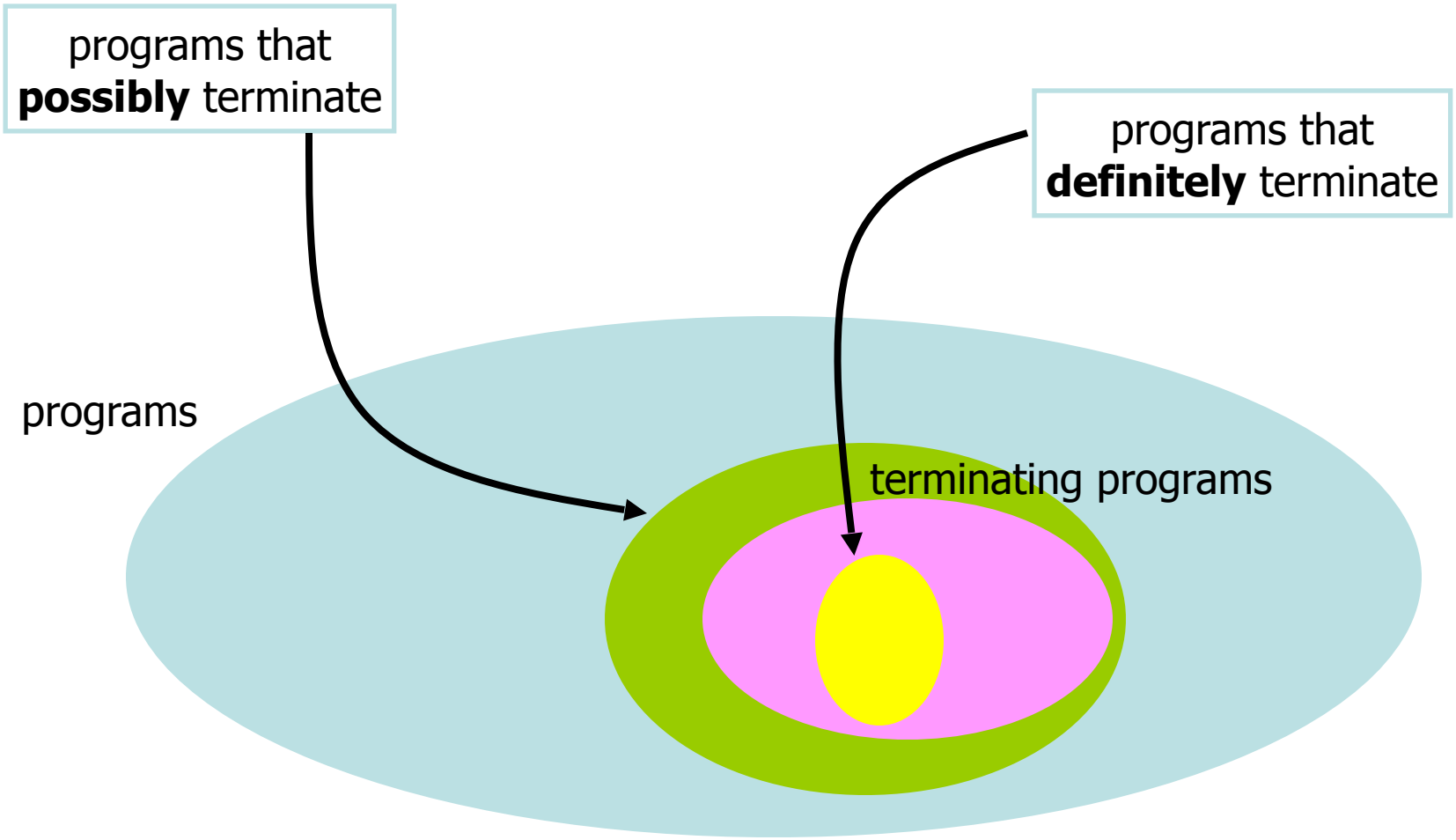
Example

The halting problem for programs is **undecidable**

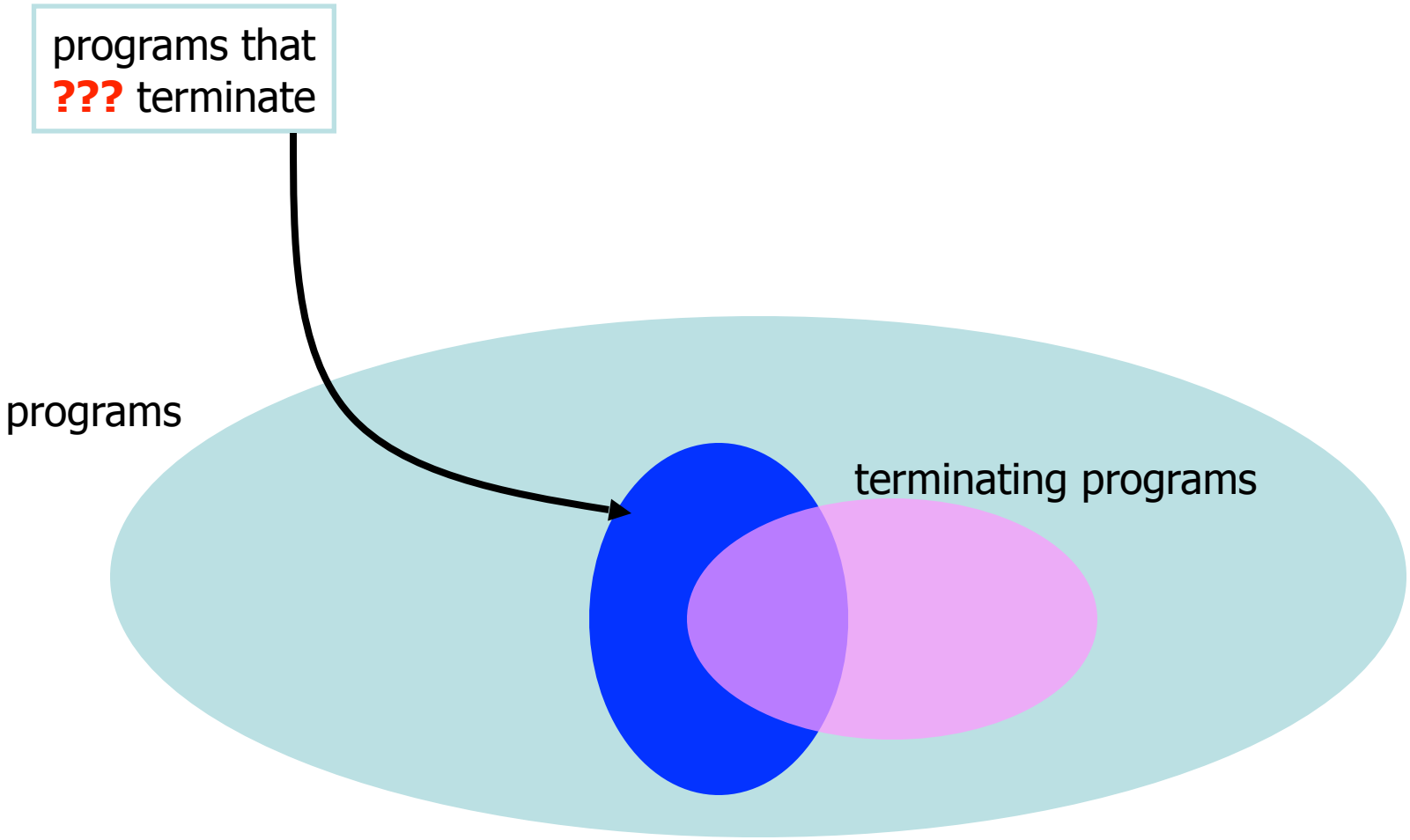
K = {P | P(0) terminates} is not a recursive set



Sound Termination Analysis



Unsound Termination Analysis



Precision and Correctness

- We thus have two types of termination analysis:
- "Definite termination analysis"
  - YES the input program **definitely** terminate
  - NO the input program could not terminate
- "Possible termination analysis"
  - YES the input program could terminate
  - NO the input program **definitely** does not terminate

Precision and Correctness

- We consider semantics-based static analysis techniques
- This means that we provide a **proof of correctness** (also called **soundness**) for the information obtained as output by the static analysis with respect to some program semantics

Coverity

From Wikipedia, the free encyclopedia

Coverity is a static code analysis tool from Synopsys. This product enables engineers and security teams to quickly find and fix defects and security vulnerabilities in custom source code written in C, C++, Java, C#, JavaScript and more.

Before its acquisition by Synopsys, Coverity was an organization founded in the Computer Systems Laboratory at Stanford University in Palo Alto, California and with headquarters in San Francisco. In June 2008, Coverity acquired Solidware Technologies.[1] And in February 2014, Coverity announced an agreement to be acquired by Synopsys, an electronic design automation company, for $350 million net of cash on hand.[2]

Contents [hide]

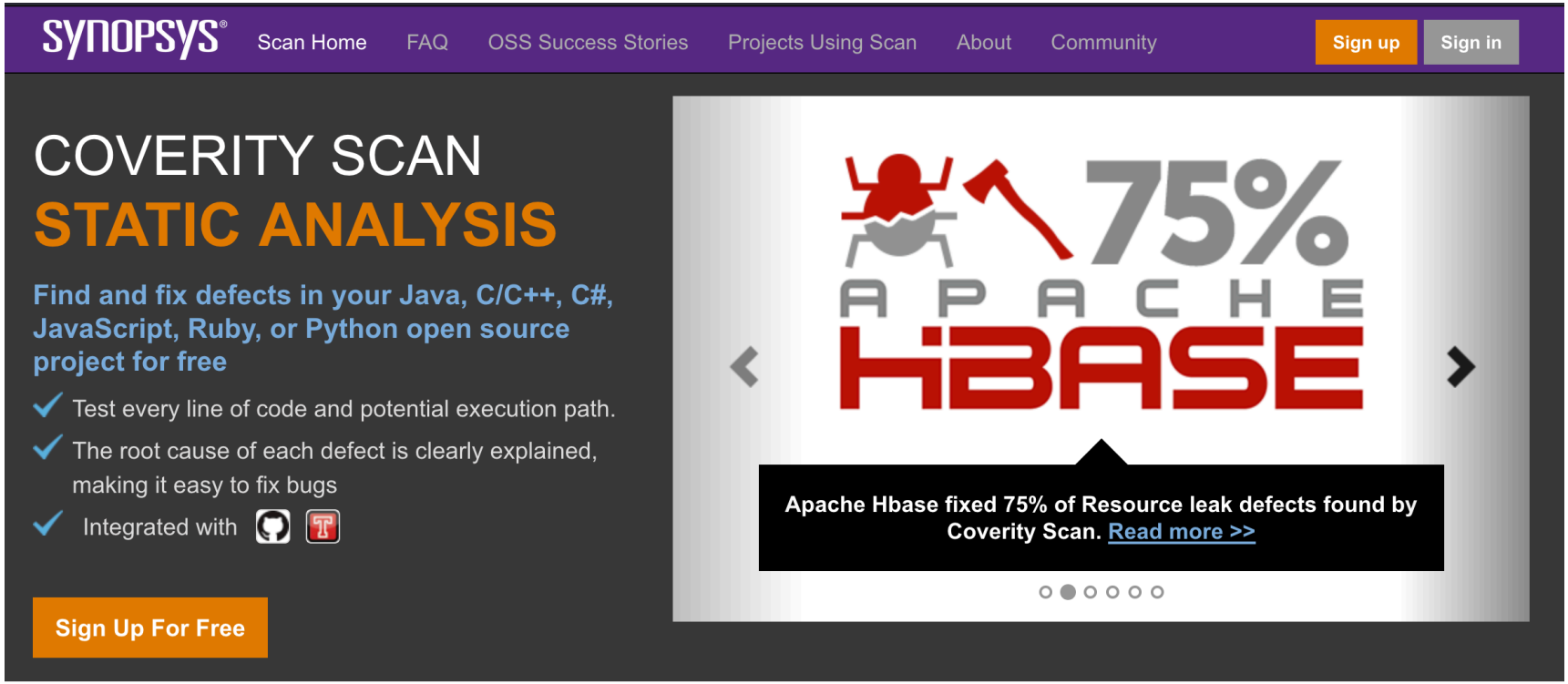
Coverity, Inc. - A Synopsys Company

Type	Public
Industry	Security Testing
Fate	Acquired by Synopsys
Founded	November 2002
Headquarters	San Francisco, CA
Key people	Andy Chou (Cofounder) Andreas Kuehlmann (SVP & GM)
Products	Coverity Code Advisor, Coverity Code Advisor on Demand, Coverity Scan, Coverity Test Advisor, Seeker
Number of employees	250+
Parent	Synopsys, Inc.
Website	synopsys.com/software-integrity.html

Products [edit]

Coverity is a static code analysis tool for C, C++, C#, Java, JavaScript, PHP, Python, .Net Core, ASP.NET, Objective-C, Go, JSP, Ruby, Swift, Fortran, Scala, VP.NET, ioS, and Typescript. It also supports more than 70 different frameworks for Java, JavaScript, C# and other languages. [3]

Coverity Scan is a free static-analysis cloud-based service for the open source community. The tool analyzes over 3900 open-source projects and is integrated with GitHub and Travis CI.[4]



More than 7500 open source projects and 39000 developers use Coverity Scan

“From my experience, I think that Coverity improves the software quality of the NNStreamer project.”

COMMUNICATIONS OF THE ACM

CONTRIBUTED ARTICLES

A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World

By Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler

Communications of the ACM, Vol. 53 No. 2, Pages 66-75
10.1145/1646353.1646374

A glance through the literature reveals many ways to go about static bug finding.[1,2,4,7,8,11] For us, the central religion was results: If it worked, it was good, and if not, not. The ideal: check millions of lines of code with little manual setup and find the maximum number of serious true errors with the minimum number of false reports. As much as possible, we avoided using annotations or specifications to reduce manual labor.

Like the PREfix product,[2] we were also unsound. Our product did not verify the absence of errors but rather tried to find as many of them as possible. Unsoundness let us focus on handling the easiest cases first, scaling up as it proved useful. We could ignore code constructs that led to high rates of false-error messages (false positives) or analysis complexity, in the extreme skipping problematic code entirely (such as assembly statements, functions, or even entire files). Circa 2000, unsoundness was controversial in the research community, though it has since become almost a de facto tool bias for commercial products and many research projects.

Claim: soundness means software **science**

False Positives (Alarms) and False Negatives

Imagine a static program analysis that in order to detect divisions by zero syntactically finds the strings "/0"

```
x = 100/0;
```
⟸**true alarm**

```
print("25/09/2016");
```
⟸**false alarm**

```
y=100/02;
```
⟸**false alarm**

```
z=0;
print(5/z);
```
⟸**false negative**

Static program analyses with false negatives are called **unsound**

Optimizing Compilers

- An optimizing compiler transforms the program code in order to improve its efficiency without affecting its input/output behavior
- Program transformations that improve the efficiency:
  - Removal of common sub-expressions: if some sub-expression is computed more than once then we can avoid re-computations
  - “Dead-code” elimination: remove a piece of code whose result is never used
  - “Constant folding”: if the operands of some expression E turn out to be constant then the computation of E can be done at compile-time
  - …

```
a = b * c + g;
d = b * c * e;
```

it may be worth transforming the code to:

```
tmp = b * c;
a = tmp + g;
d = tmp * e;
```

Examples

```
int foo(void)
{
  int a = 24;
  int b = 25; /* Assignment to dead variable */
  int c;
  c = a * 4;
  return c;
  b = 24; /* Unreachable code */
  return 0;
}
```

```
int x = 14;
int y = 7 - x / 2;
return y * (28 / x + 2);
```

Propagating x yields:

```
int x = 14;
int y = 7 - 14 / 2;
return y * (28 / 14 + 2);
```

Continuing to propagate yields the following

```
int x = 14;
int y = 0;
return 0;
```

gcc -O option flag

Set the compiler's optimization level.

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

+increase ++increase more +++increase even more -reduce --reduce more ---reduce even more

3.10 Options That Control Optimization

These options control various sorts of optimizations.

Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you expect from the source code.

Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Data Flow Analysis

Theory and Practice

Uday P. Khedker
Amitabha Sanyal
Bageshri Karkare

10

Implementing Data Flow Analysis in GCC

This chapter presents a *g*eneric *d*ata *f*low *a*nalyzer for *p*er *f*unction (i.e., intraprocedural) *b*it *v*ector *d*ata *f*low *a*nalysis in GCC 4.3.0. We call this infrastructure *gdfa*. The analyzers implemented using *gdfa* are called *pfbvdfa*. *gdfa* has been used to implement several bit vector data flow analyses.

An optimal compiler?

- A "fully size-optimizing compiler" could be defined as a compiler that transforms a program P into the program Opt(P) which should be "**the smallest**" program having the same input/output behavior as P.
- Given a program Q that does not terminate, it is simple to find Opt(Q): this is the program L1: goto L1;
- Assume that a fully optimizing compiler exists. We could use it to solve the halting problem: in order to determine if there exists an input In such that a program P terminates on input In, it is enough to see whether Opt(P) coincides with L1: goto L1;
- Since the halting problem is undecidable, the above remark implies that a fully optimizing compiler cannot exist.

Static Analysis Design

In order to design a **sound static analysis** **SA** we need to:

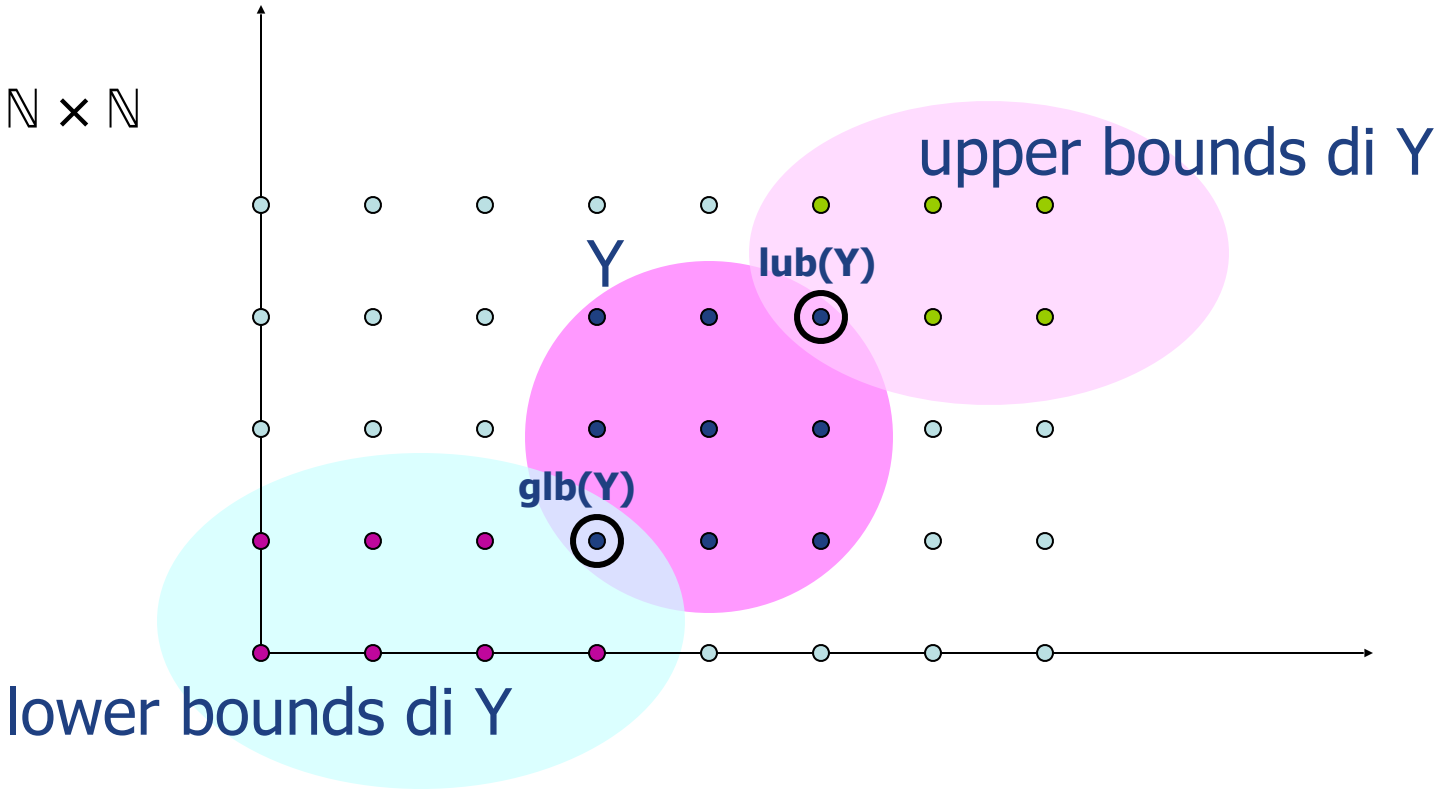
1. have a reference program semantics **Sem**
2. specify what and how the static analysis **SA** computes
3. prove the correctness of **SA** w.r.t. **Sem**
4. efficiently implement a static analyzer for **SA**

Static Analysis Techniques

- **Abstract Interpretation**
- Dataflow Analysis
- Model Checking
- Logical Deductive Systems (e.g. separation logic and SMT/SAT solvers)
- Type Systems

More on Lattices and Fixpoints

Example: lub e glb



$$(x_1,y_1) \leq_{\mathbb{N}\times\mathbb{N}} (x_2,y_2) \;\Leftrightarrow\; x_1 \leq_{\mathbb{N}} x_2 \;\wedge\; y_1 \leq_{\mathbb{N}} y_2$$

Lattices

- A lattice is a poset $(L, \leq)$ such that there exist lub and glb of any pair of elements in L

- If L is a nonempty poset and $x \leq y$ then

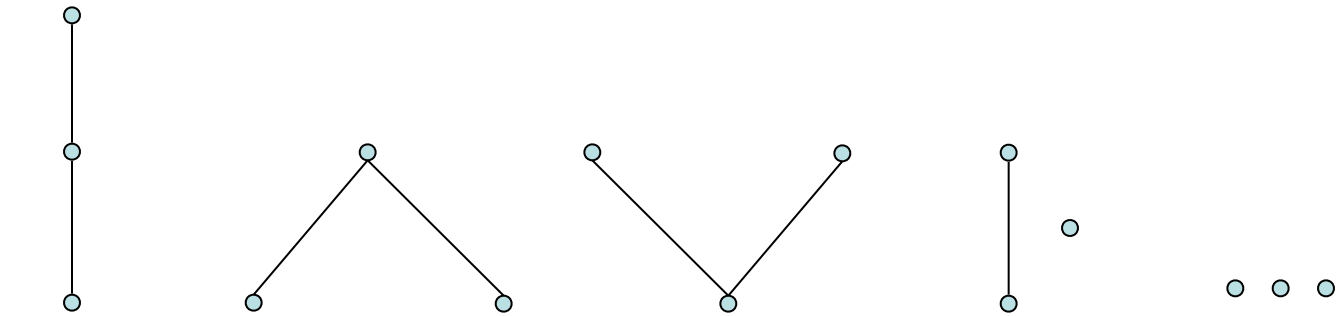
  $\text{lub}(\{x,y\}) = y$

  $\text{glb}(\{x,y\}) = x$

  In order to prove that L is a lattice it is therefore enough to show that both lub and glb exist for any pair of uncomparable elements in L

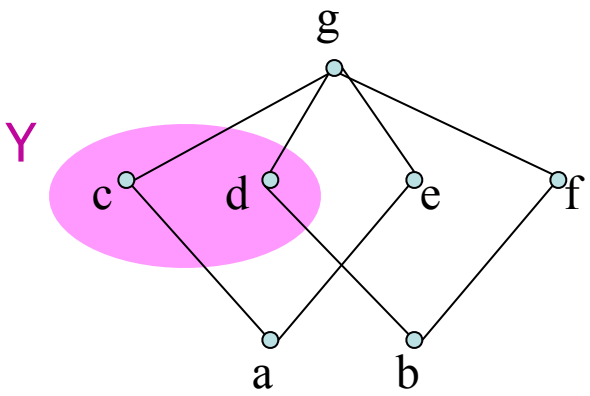
Example

Are these 3-elements posets actually lattices?



YES NO NO NO NO

Example



- L= {a,b,c,d,e,f,g}
- ≤ ={(a,c), (a,e), (b,d), (b,f), (c,g), (d,g), (e,g), (f,g)}^{T}
- (L,≤) is not a lattice: Y does not have lower bounds

Chains

- Given a poset $(L,\leq)$, a subset $Y \subseteq L$ is a chain if

  $$\forall\, y_1, y_2 \in Y : (y_1 \leq y_2) \vee (y_2 \leq y_1)$$

  that is, a subset Y of L is a chain when Y is totally ordered.
- A poset $(L,\leq)$ has finite height $n \in \mathbb{N}$ if the cardinality of the largest chain in L is n+1.
- A poset $(L,\leq)$ does not have infinite chains if every chain in L is finite.
- A denumerable sequence $(y_n)_{n\in\mathbb{N}}$ (i.e., indexed on natural numbers $\mathbb{N}$) of elements in L is an ascending chain if

  $$n \leq m \Rightarrow y_n \leq y_m$$

CPOs

- A poset $(P,\leq_P)$ is a CPO (chain-Complete Partial Order) if every chain in P has the lub.

- In particular, any CPO has the bottom element $\perp$ since this is the lub of the empty chain.

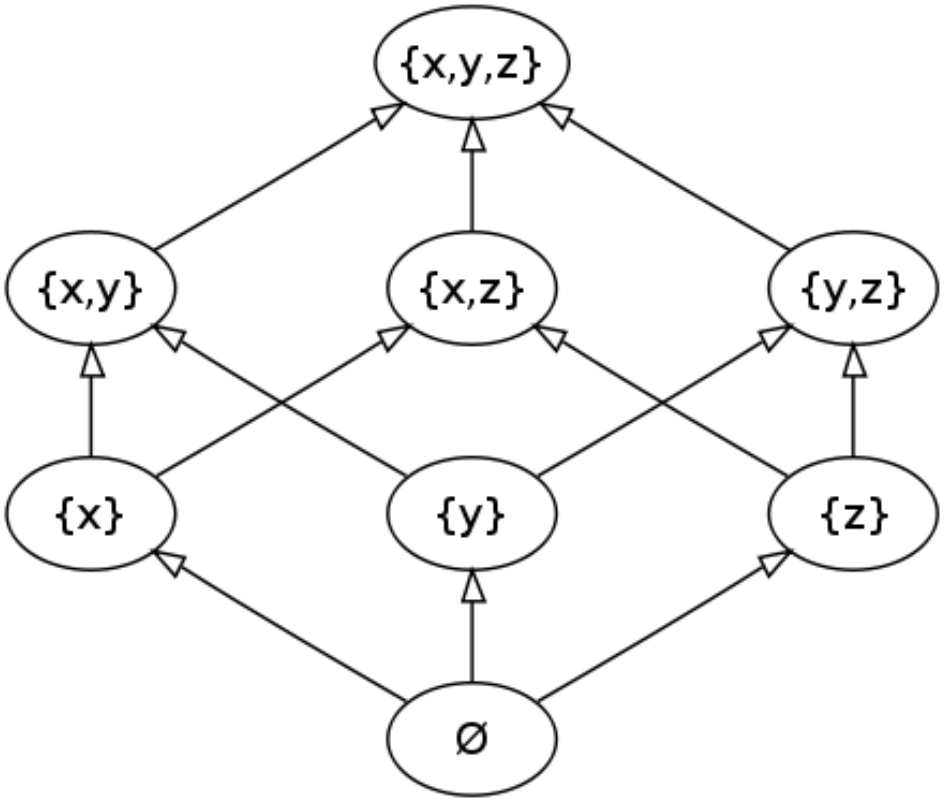
Complete lattices

- A complete lattice is a poset $(L, \leq)$ such that any (possibly empty) subset of L has both lub and glb.
- If $(L, \leq)$ is a complete lattice then we use the following notation:

$$\bot = \text{lub}(\varnothing) = \text{glb}(L) \qquad \text{bottom element}$$
$$\top = \text{lub}(L) = \text{glb}(\varnothing) \qquad \text{top element}$$

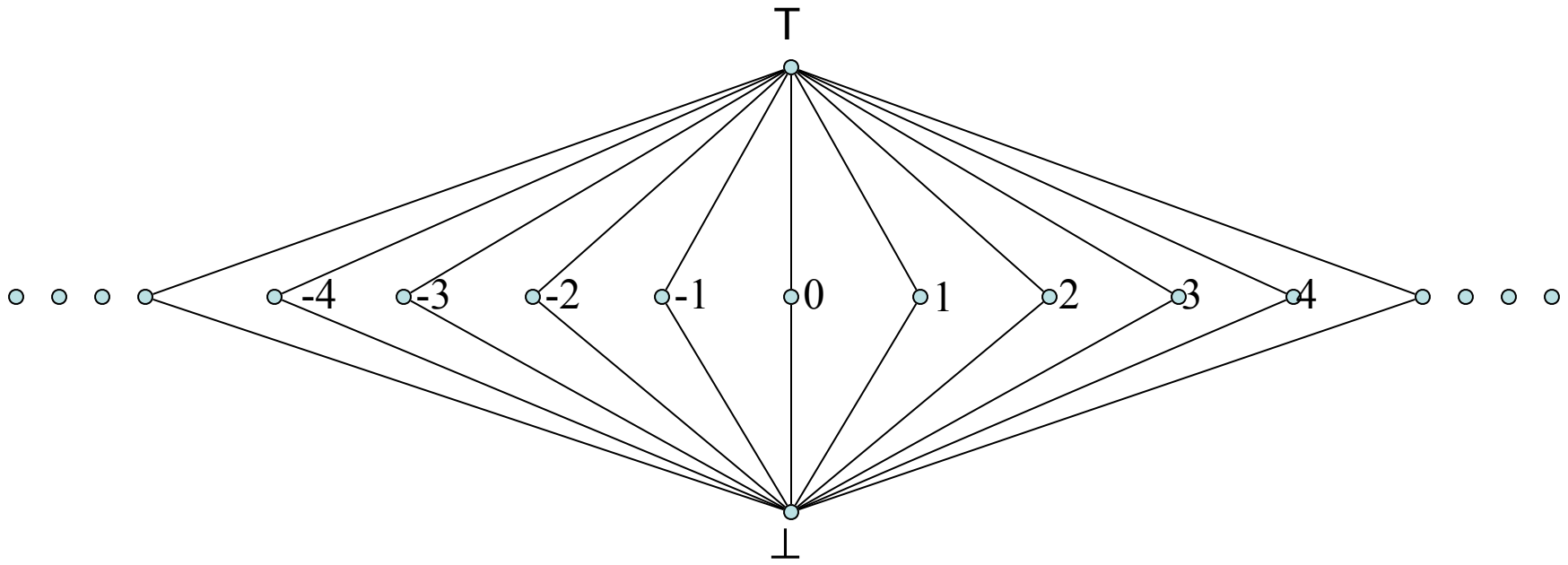
- Each finite lattice is a complete lattice
- Each complete lattice is a CPO

Example



- L= $\wp$ ({x,y,z})
- ≤ = ⊆
- lub(Y) = ∪Y
- glb(Y) = ∩Y

Example



- $L = \mathbb{Z} \cup \{\top,\bot\}$
- $\forall n \in \mathbb{Z} : \bot \leq n \leq \top$

Example

- L = $\mathbb{N}$
- $\leq$ total order on $\mathbb{N}$
- lub = max
- glb = min
- This is a lattice, but it is not a complete lattice and it is not a CPO

Example

- L= $\mathbb{N} \cup \{\top\}$
- $\leq$ total order on $\mathbb{N} \cup \{\top\}$
- lub = sup
- glb = min
- This is a complete lattice

Theorem: Let $(L, \leq)$ be a poset. The following statements are equivalent:

1. L is a complete lattice
2. each subset of L has lub
3. each subset of L has glb

In order to show that $2 \Rightarrow 1$, for any $Y \subseteq L$ we prove that

$$glb(Y) = lub(\{x \in L \mid \forall y \in Y : x \leq y\})$$

Converging Chains

- Recall that a **denumerable** sequence $(x_n)_{n\in\mathbb{N}}$ of elements in L is an ascending chain if

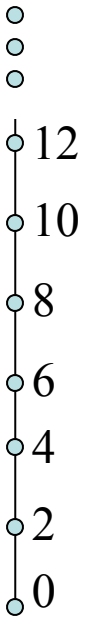
$$n \leq m \Rightarrow x_n \leq x_m$$

- A denumerable sequence $(x_n)_{n\in\mathbb{N}}$ converges if

$$\exists n_0 \in \mathbb{N}.\ \forall n \in \mathbb{N}.\ n_0 \leq n \Rightarrow x_{n_0} = x_n$$

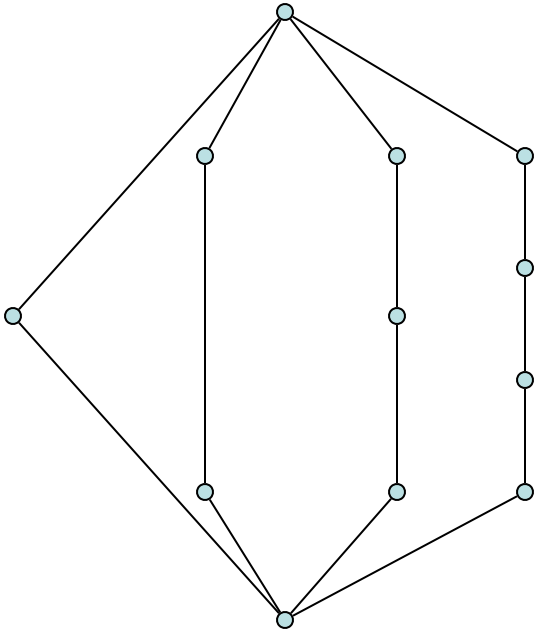
- A poset $(L,\leq)$ satisfies the Ascending Chain Condition (ACC) when each ascending chain in L converges (namely, when L does not have infinite ascending chains).

Example



The poset of even numbers is not ACC

Example



...

This is an infinite complete lattice which is ACC but does not have finite height

Deriving new Lattices

Given a (complete) lattice or partial order $(X, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ we can derive new (complete) lattices or partial orders by:

- duality
  $(X, \sqsupseteq, \sqcap, \sqcup, \top, \bot)$
  - $\sqsubseteq$ is reversed
  - $\sqcup$ and $\sqcap$ are switched
  - $\bot$ and $\top$ are switched

- lifting (adding a smallest element)
  $(X \cup \{ \bot' \}, \sqsubseteq', \sqcup', \sqcap', \bot', \top)$
  - $a \sqsubseteq' b \iff a = \bot' \vee a \sqsubseteq b$
  - $\bot' \sqcup' a = a \sqcup' \bot' = a$, and $a \sqcup' b = a \sqcup b$ if $a, b \neq \bot'$
  - $\bot' \sqcap' a = a \sqcap' \bot' = \bot'$, and $a \sqcap' b = a \sqcap b$ if $a, b \neq \bot'$
  - $\bot'$ replaces $\bot$
  - $\top$ is unchanged

Deriving new Lattices

Given (complete) lattices or partial orders:
$(X_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$ and $(X_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$

We can combine them by:

- product
  $(X_1 \times X_2, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ where
  - $(x, y) \sqsubseteq (x', y') \iff x \sqsubseteq_1 x' \wedge y \sqsubseteq_2 y'$
  - $(x, y) \sqcup (x', y') \stackrel{\text{def}}{=} (x \sqcup_1 x',\ y \sqcup_2 y')$
  - $(x, y) \sqcap (x', y') \stackrel{\text{def}}{=} (x \sqcap_1 x',\ y \sqcap_2 y')$
  - $\bot \stackrel{\text{def}}{=} (\bot_1, \bot_2)$
  - $\top \stackrel{\text{def}}{=} (\top_1, \top_2)$

Fixpoints

- Let $f:(P,\leq_P) \rightarrow (P,\leq_P)$ be a monotonic function on a poset P.
- An element $x \in P$ is a fixpoint of f when $f(x)=x$.
- The set of fixpoints of f is a subset of P which is denoted by Fix(f):

$$Fix(f) = \{x \in P \mid f(x)=x\}$$

- The least fixpoint of f, when it exists, is denoted by $lfp(f) \in P$. Analogously, the greatest fixpoint of f, when this exists, is denoted by $gfp(f) \in P$.

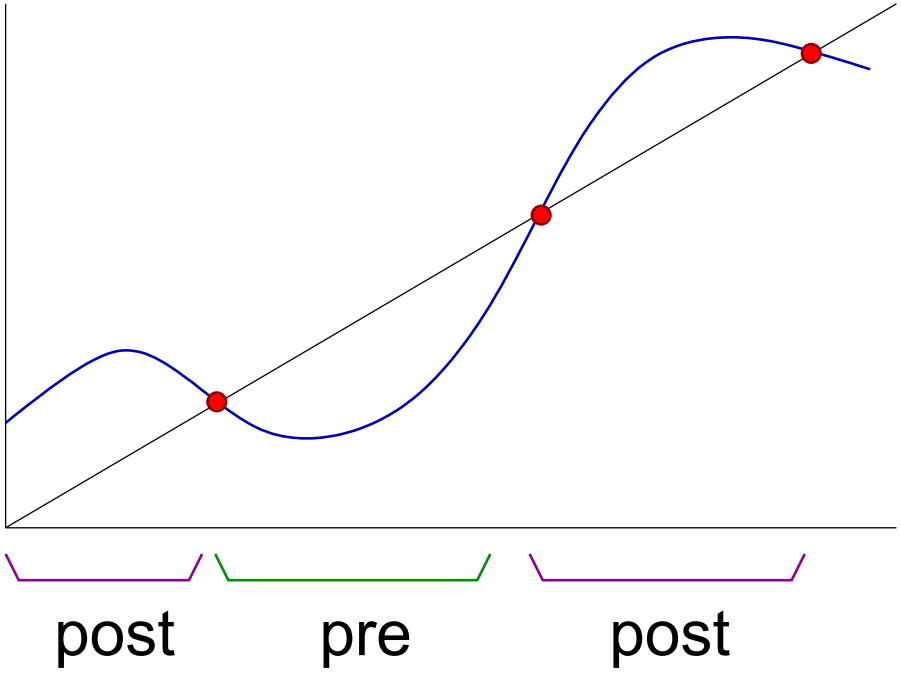
Pre- and Post-Fixpoints

- The set of **pre-fixpoints** of f is denoted by Pre(f):

$$Pre(f) = \{x \in P \mid f(x) \leq x\}$$

- The set of **post-fixpoints** of f is denoted by Post(f):

$$Post(f) = \{x \in P \mid x \leq f(x)\}$$

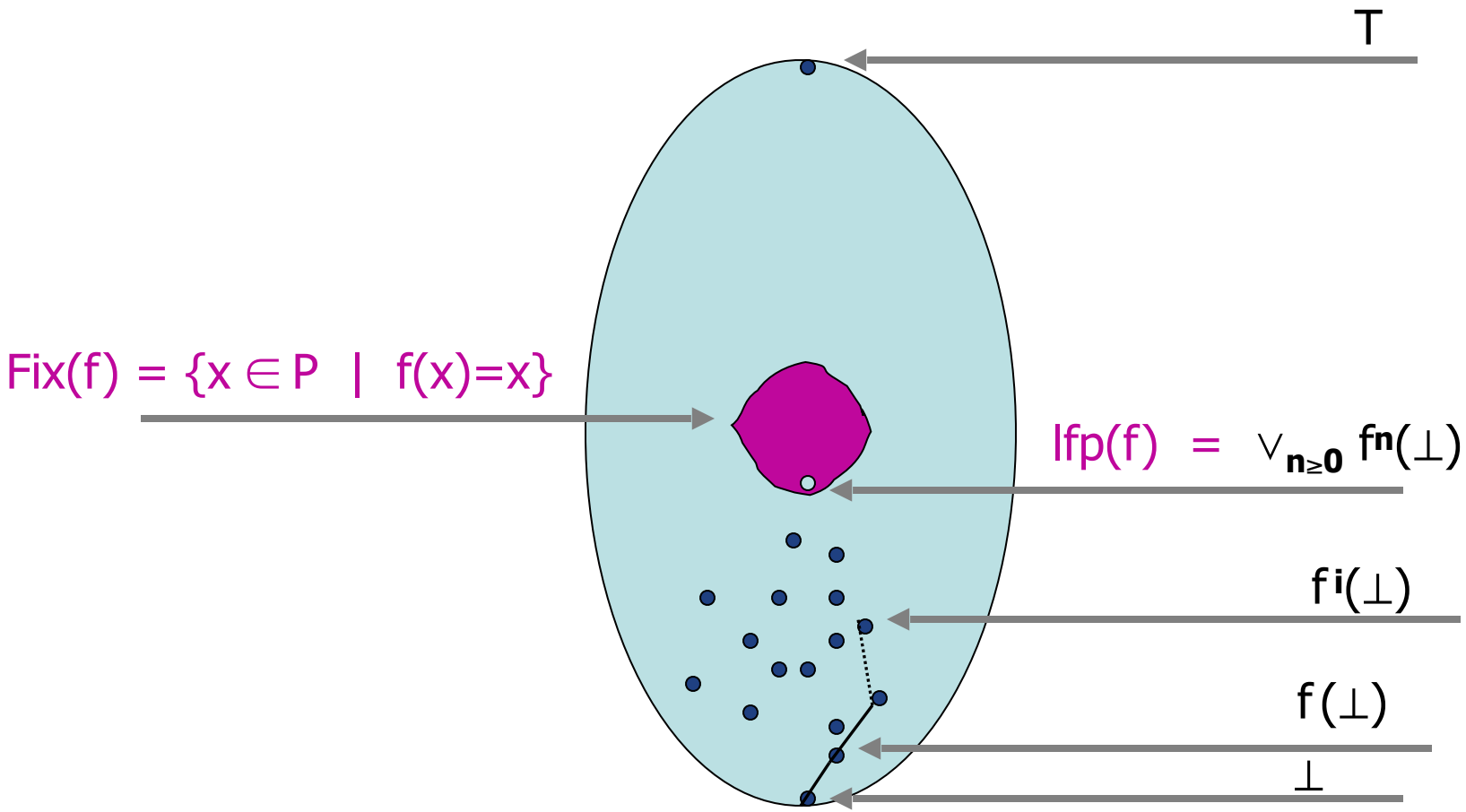


Fixpoints on CPOs

Let $f:(P,\leq_P) \rightarrow (P,\leq_P)$ be a **monotonic** function on a CPO P.
Let $\alpha \triangleq \bigvee_{\mathbf{n\geq 0}} f^n(\bot) \in P$

- If $\alpha \in Fix(f)$ then $\alpha = lfp(f)$
- Knaster-Tarski-Kleene Theorem
  If f is **continuous** then the least fixpoint of f exists and is α.

Fixpoints on CPOs



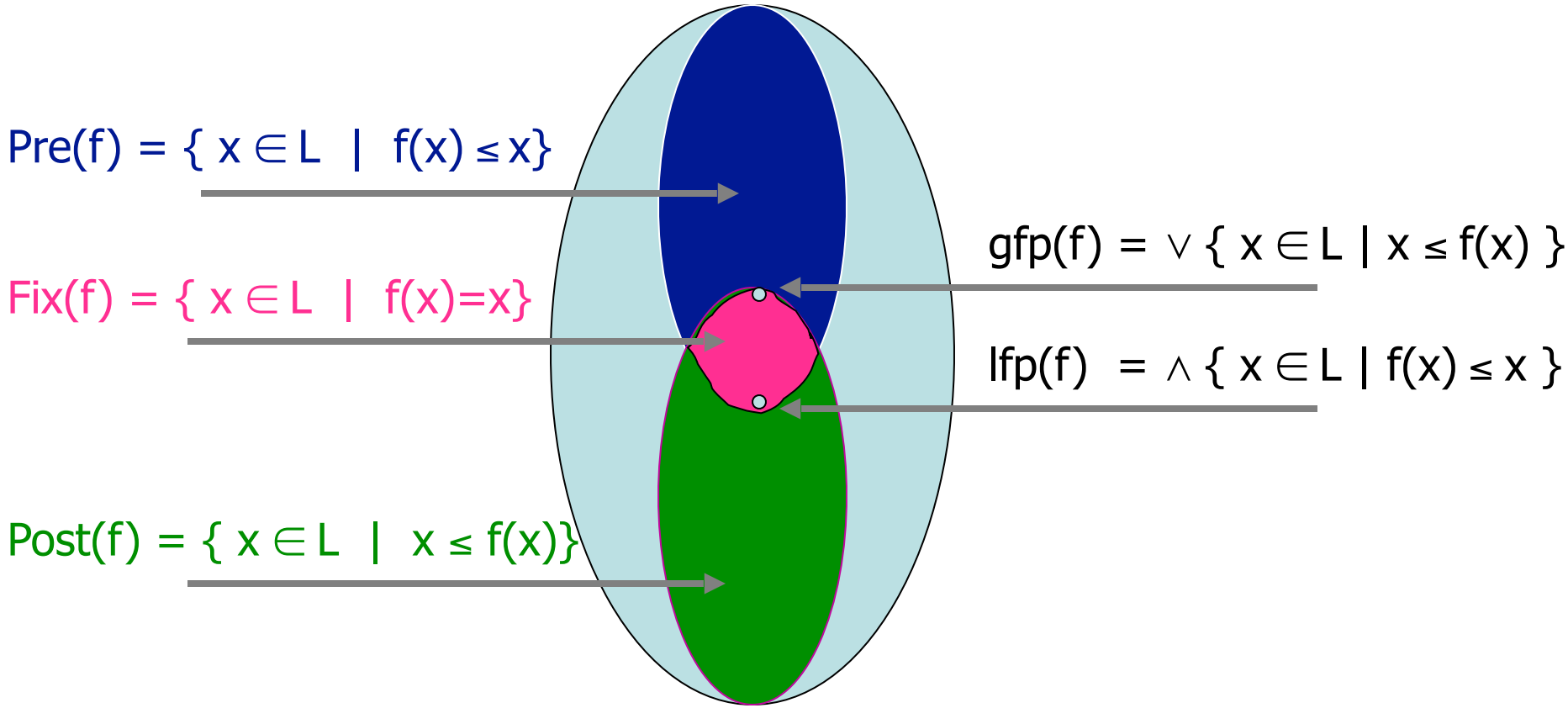
Fixpoints on complete lattices

- Let $f:L \to L$ be a **monotonic** function on a complete lattice L.
- It turns out that $(Fix(f), \leq_L)$ is a complete lattice where:

$$lfp(f) = glb(Fix(f)) \in Fix(f)$$
$$gfp(f) = lub(Fix(f)) \in Fix(f)$$

- **Theorem**
  Let L be a complete lattice. If $f:L \to L$ is a **monotonic** function then
  $lfp(f) = \wedge \{ x \in L \mid f(x) \leq x \}$ (glb of pre-fixpoints)
  $gfp(f) = \vee \{ x \in L \mid x \leq f(x) \}$ (lub of post-fixpoints)

Fixpoints on complete lattices



Fixpoints on CPOs

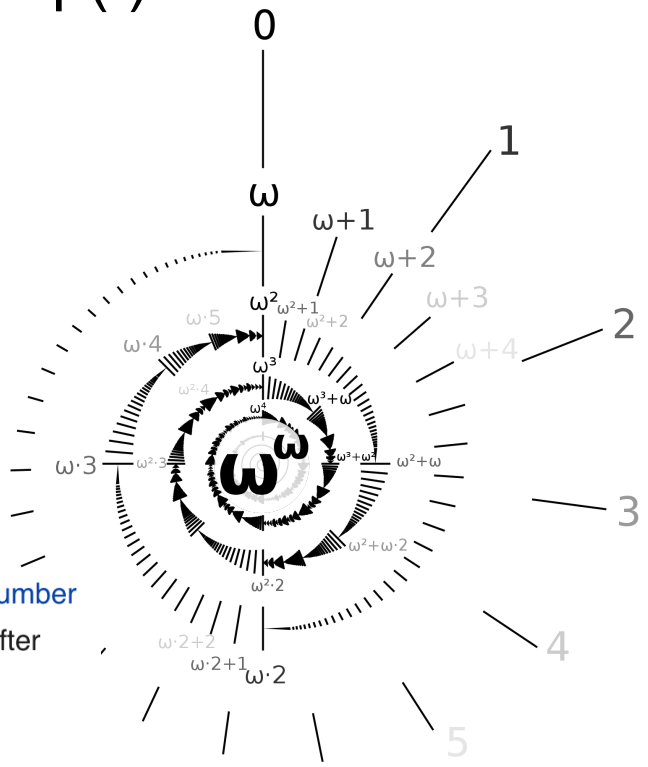
Theorem

Let $f:(P,\leq_P) \to (P,\leq_P)$ be a **monotonic** function on a CPO P. Then, f has the least fixpoint lfp(f).

$$\mathrm{lfp}(f) = \bigvee_{\alpha \in \mathbf{Ord}} f^{\alpha}(\bot)$$

Ord is the class of ordinal numbers

In set theory, an **ordinal number**, or **ordinal**, is one generalization of the concept of a natural number that is used to describe a way to arrange a (possibly infinite) collection of objects in order, one after another.[1][2]



Fixpoints

We thus have three results that guarantee the existence of the least fixpoint:

1. Continuous function on a CPO
2. Monotonic function on a complete lattice
3. Monotonic function on a CPO

Results 1 and 2 have stronger hypotheses and provide an explicit characterization of the least fixpoint. Result 3 relies on ordinal numbers.

Uses of Fixpoints

Express solutions of mutually recursive equation systems

Example:

The solutions of $\begin{cases} x_1 = f(x_1, x_2) \\ x_2 = g(x_1, x_2) \end{cases}$ with x_1, x_2 in lattice X

are exactly the fixpoint of $\vec{F}$ in lattice $X \times X$, where

$$\vec{F}\begin{pmatrix} x_1, \\ x_2 \end{pmatrix} = \begin{pmatrix} f(x_1, x_2), \\ g(x_1, x_2) \end{pmatrix}$$

The least solution of the system is lfp $\vec{F}$.

Abstract Interpretation

Abstract interpretation

From Wikipedia, the free encyclopedia

In computer science, **abstract interpretation** is a theory of sound approximation of the semantics of computer programs, based on monotonic functions over ordered sets, especially lattices. It can be viewed as a partial execution of a computer program which gains information about its semantics (e.g., control-flow, data-flow) without performing all the calculations.

Its main concrete application is formal static analysis, the automatic extraction of information about the possible executions of computer programs; such analyses have two main usages:

- inside compilers, to analyse programs to decide whether certain optimizations or transformations are applicable;
- for debugging or even the certification of programs against classes of bugs.

Abstract interpretation was formalized by the French computer scientist working couple Patrick Cousot and Radhia Cousot in the late 1970s.[1][2]

Contents [hide]

POPL Principles of Programming Languages

The annual Symposium on Principles of Programming Languages is a forum for the discussion of all aspects of programming languages and systems, with emphasis on how principles underpin practice. Both theoretical and experimental papers are welcome, on topics ranging from formal frameworks to experience reports.

Top 8 Most Cited Articles (as of 2018)

Most Popular

Downloaded ***Cited***

1 January 1977
Abstract interpretation: a unified lattice model for static analysis of programs by construction or...
POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on...
Patrick Cousot, Radhia Cousot
Cited 3,415 times

2 January 1979
Systematic design of program analysis frameworks
POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on...
Patrick Cousot, Radhia Cousot
Cited 937 times

3 January 1978
Automatic discovery of linear restraints among variables of a program
POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on...
Patrick Cousot, Nicolas Halbwachs
Cited 931 times

4 January 2002
Lazy abstraction
POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on...
Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Grégoire Sutre
Cited 858 times

5 October 1987
Constraint logic programming
POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on...
J. Jaffar, J.-L. Lassez
Cited 751 times

6 January 1989
On the synthesis of a reactive module
POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on...
A. Pnueli, R. Rosner
Cited 750 times

7 January 1997
Proof-carrying code
POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on...
George C. Necula
Cited 679 times

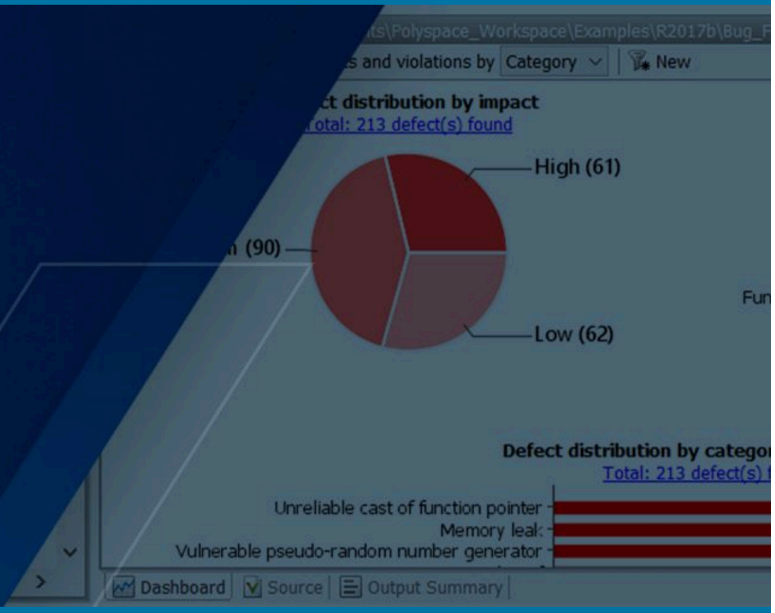
8 January 1995
Precise interprocedural dataflow analysis via graph reachability
POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on...
Thomas Reps, Susan Horwitz, Mooly Sagiv
Cited 672 times

Abstract Interpretation Tools

Polyspace Bug Finder

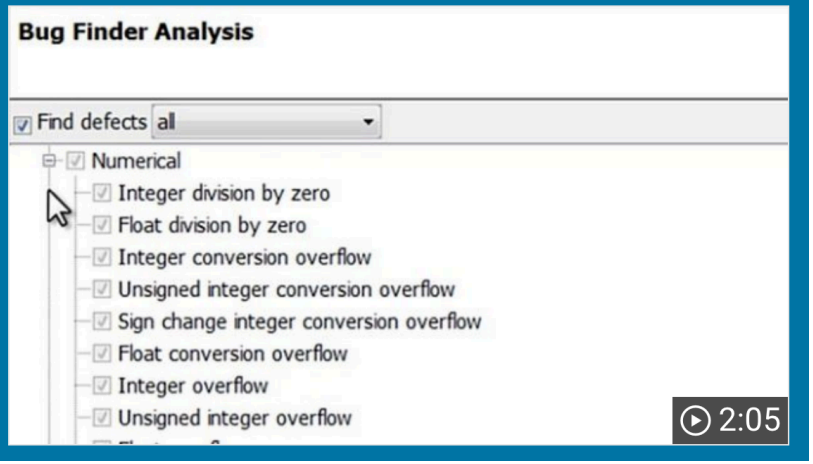
Identify software bugs using static analysis

Request a trial Request a quote



Polyspace Bug Finder™ identifies run-time errors, concurrency issues, security vulnerabilities, and other defects in C and C++ embedded software. Using static analysis, including semantic analysis, Polyspace Bug Finder analyzes software control, data flow, and interprocedural behavior. By highlighting defects as soon as they are detected, it lets you triage and fix bugs early in the development process.

Polyspace Bug Finder checks compliance with coding rule standards such as MISRA C and MISRA C++, AUTOSAR C++ 14, CERT® C, CERT® C++, and custom naming conventions. It generates reports consisting of bugs found, code-rule violations, and code quality metrics, including cyclomatic complexity. Polyspace Bug Finder can be used with the



The Astrée Static Analyzer

Centre National de la Recherche Scientifique — École Normale Supérieure — INRIA (since Sep. 2007)

Participants:

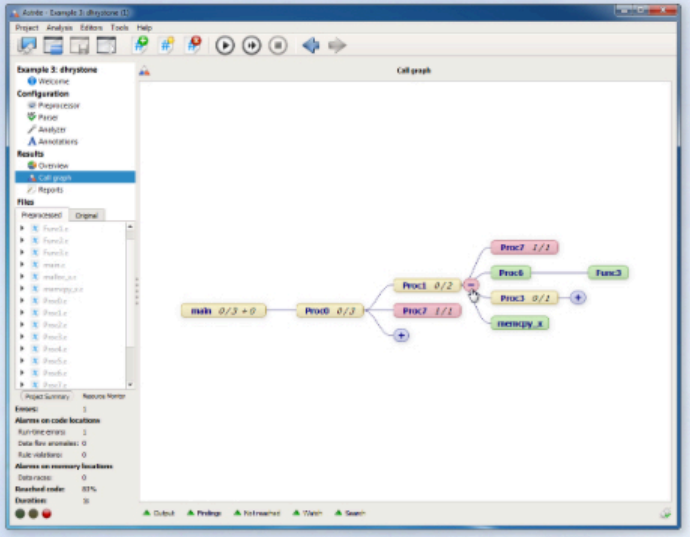
Patrick Cousot (project leader), Radhia Cousot, Jérôme Feret, Antoine Miné, Xavier Rival

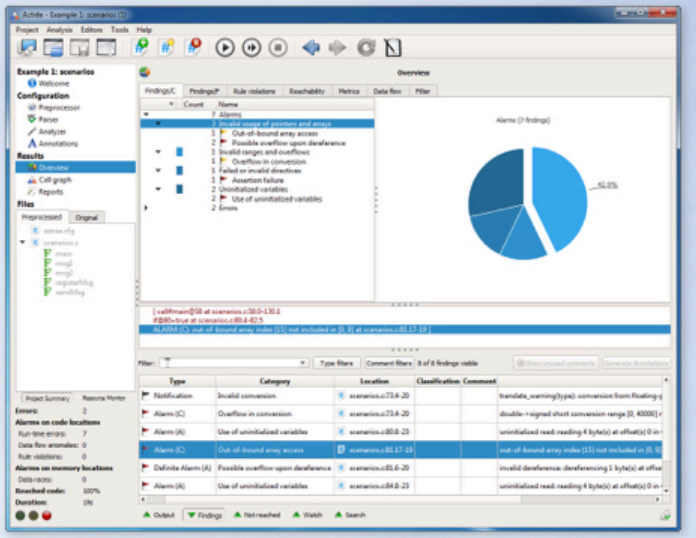
Former participants:

Bruno Blanchet (Nov. 2001 — Nov. 2003), David Monniaux (Nov. 2001 — Aug. 2007), Laurent Mauborgne (Nov. 2001 — Aug. 2010).

Fast and sound runtime error analysis

Astrée is a static code analyzer that proves the absence of runtime errors and invalid concurrent behavior in safety-critical software written or generated in C.





Astrée primarily targets embedded applications as found in aeronautics, earth transportation, medical instrumentation, nuclear energy, and space flight. Nevertheless, it can just as well be used to analyze any structured C programs, handwritten or generated, with complex memory usages, dynamic memory allocation, and recursion.

Astrée is sound — that is, if no errors are signaled, the absence of errors has been proved.

Interproc

by Gaël Lalire, Mathias Argoud, and Bertrand Jeannet

About

Interproc is an interprocedural analyzer for a small imperative language with (recursive) procedure calls. It infers invariants on the numerical variables of analyzed program.

It it intended as a pedagogical and experimental tool. It also demonstrates the features of the APRON library and the use of the Fixpoint libraries.

It is implemented in OCaml.

License

Try it online

You can play with Interproc online.

Download

- Tar-gzipped sources (includes documentation)
- Subversion repository (see Up for the organisation of the repository)

Documentation

- On-line
- PDF

Analysis Result

Run interprocweb or interprocwebf ?

Result

```
Annotated program after forward analysis
var m : int, n : int;
begin
  /* (L2 C5) top */
  m = 2; /* (L3 C8) [|m-2>=0|] */
  while n > 1 do
    /* (L4 C16) [|m-2>=0; n-2>=0|] */
    m = m * n; /* (L5 C10) [|m-4>=0; n-2>=0|] */
    n = n - 1; /* (L6 C9) [|m-4>=0; n-1>=0|] */
  done; /* (L7 C7) [|m-2>=0; -n+1>=0|] */
end
```

Source

```
var m:int, n:int;
begin
  m = 2;
  while (n>1) do
    m=m*n;
   n=n-1;
  done;
end
```

A tool to detect bugs in Java and C/C++/Objective-C code before it ships

Infer is a static analysis tool - if you give Infer some Java or C/C++/Objective-C code it produces a list of potential bugs. Anyone can use Infer to intercept critical bugs before they have shipped to users, and help prevent crashes or poor performance.

Get Started | Learn More

Android and Java

Infer checks for null pointer exceptions, resource leaks, annotation reachability, missing lock guards, and concurrency race conditions in Android and Java code.

C, C++, and iOS/Objective-C

Infer checks for null pointer dereferences, memory leaks, coding conventions and unavailable API's.

About Infer

Infer is a static program analyzer for Java, C, and Objective-C, written in OCaml. Infer is deployed within Facebook and it is running continuously to verify select properties of every code modification for the main Facebook apps for Android and iOS, Facebook Messenger, Instagram, and other apps. It can be used for other code too: Infer can also analyze C code, and Java code that is not Android. At present Infer is tracking problems caused by null pointer dereferences and resource and memory leaks, which cause some of the more important problems on mobile.

Infer came to Facebook with the acquisition of the verification startup Monoidics in 2013. Monoidics was itself based on recent academic research, particularly on separation logic and bi-abduction.

We have broadened Infer's scope within the past few years. We now refer to the original separation logic analysis as Infer.SL. We now also have Infer.AI, a general analysis framework which is an interface to the modular analysis engine which can be used by other kinds of program analyses (technically, called "abstract interpretations", hence the AI monicker).
This added generality has been used to develop instantiations of Infer.AI for security, concurrency and in other domains. Additionally, we have Infer linters for describing shallow syntactic analyses, using the AL language, because sometimes linters are just what you need.

facebook / infer Public

Watch 603 | Unstar 12.7k | Fork 1.7k

Code | Issues 311 | Pull requests 5 | Actions | Projects | Wiki | Security | Insights

main | 9 branches | 32 tags | Go to file | Add file | Code

da319 and facebook-github-bot [pulse] Add new issue types for late... ... 71b5144 19 hours ago 10,244 commits

.github/workflows	[github] badges for the github actions	2 months ago
dependencies	[nullsafe] Mark **@nullsafe** with @TypeQualifierDefault&Co for Kotlin i...	2 years ago
docker	Readme.md (#1541)	15 days ago
examples	[copyright] Remove years	2 years ago
facebook-clang-plugins	Add patch to make clang work with xcode13	2 days ago
infer	[pulse] Add new issue types for latent issues	19 hours ago
m4	[opam] Move the opam files to an opam directory at repo root	8 months ago
opam	[opam] upgrade dependencies	8 days ago
scripts	[caml] upgrade to 4.12 and most recent versions of dependencies	5 months ago
sledge	[sledge] Strengthen canonization of Extract terms	16 days ago
website	Bump axios from 0.21.1 to 0.21.4 in /website (#1527)	15 days ago
.buckconfig	[infer][genrule] Add example of Buck DEFS macro to generate Infer a...	5 years ago

About

A static analyzer for Java, C, C++, and Objective-C

fbinfer.com/

c | java | objective-c | cpp | static-code-analysis | static-analysis | code-quality

Readme

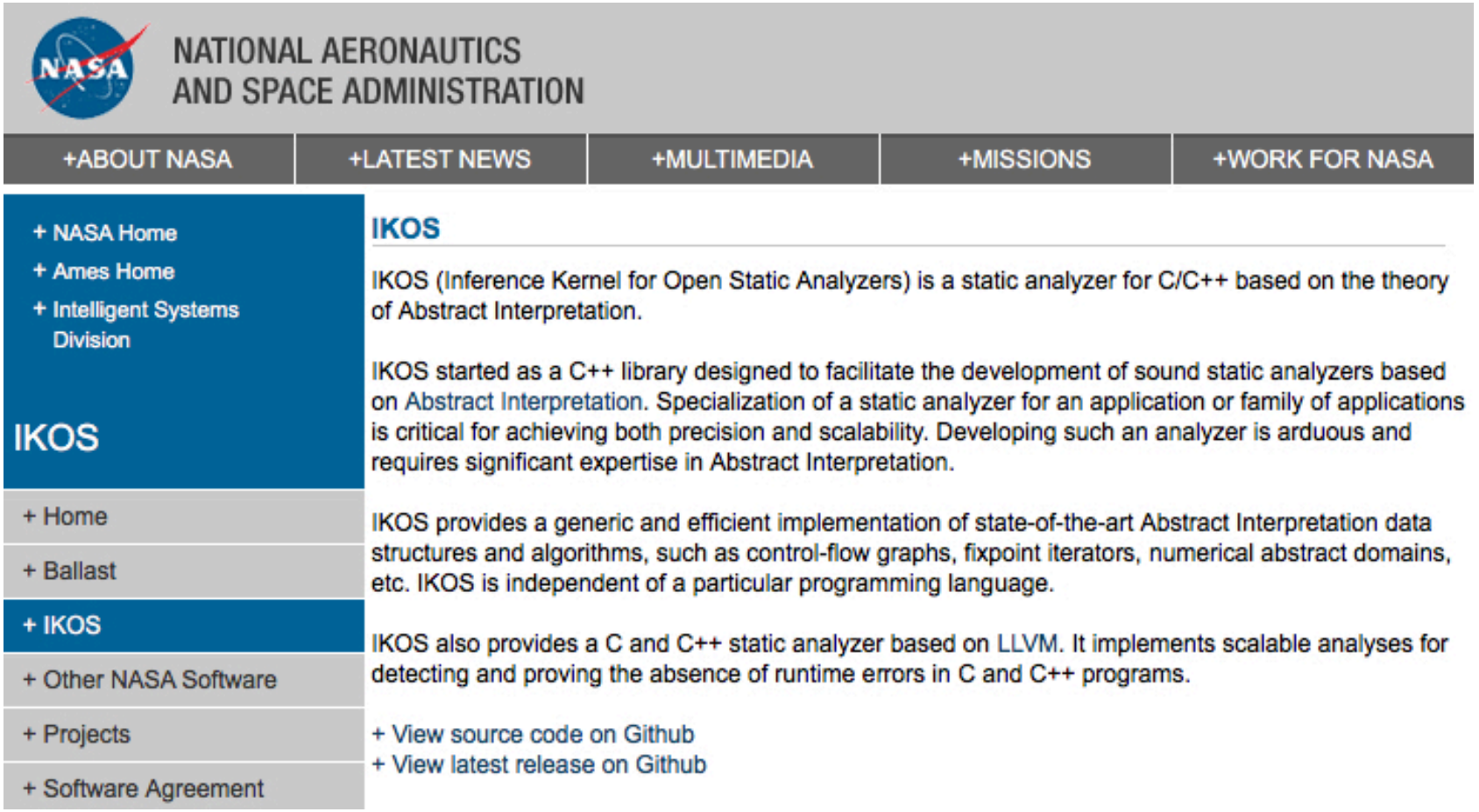
MIT License

Code of conduct

Releases 29

Infer version v1.1.0 Latest
on Mar 26

+ 28 releases



https://github.com/nasa-sw-vnv/ikos

NASA-SW-VnV / ikos

Watch 40 | Unstar 1.2k | Fork 77

Code | Issues 29 | Pull requests 0 | Projects 0 | Wiki | Security | Insights

Static analyzer for C/C++ based on the theory of Abstract Interpretation.

static-analysis software-verification abstract-interpretation program-analysis

372 commits | 1 branch | 4 releases | 2 contributors | View license

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

arthaud Upgrade to LLVM 9 (closes #154) Latest commit ca70f94 5 hours ago

analyzer	Upgrade to LLVM 9 (closes #154)	5 hours ago
ar	[ar] Allow partitioning on functions returning common integer types	14 days ago
cmake	[cmake] Add an option to enable assertions	2 months ago
core	[core] Assign the offset of null to zero in the composite scalar domain	18 days ago
doc	Upgrade to LLVM 9 (closes #154)	5 hours ago
frontend/llvm	Upgrade to LLVM 9 (closes #154)	5 hours ago
script	Upgrade to LLVM 9 (closes #154)	5 hours ago
test/install	Upgrade to LLVM 9 (closes #154)	5 hours ago

SPARTA

SPARTA is a library of software components specially designed for building high-performance static analyzers based on the theory of Abstract Interpretation.

Abstract Interpretation

Abstract Interpretation is a theory of semantic approximation that provides a foundational framework for the design of static program analyzers. Static analyzers built following the methodology of Abstract Interpretation are mathematically sound, i.e., the semantic information they compute is guaranteed to hold in all possible execution contexts considered. Moreover, these analyzers are able to infer complex properties of programs, the expressiveness of which can be finely tuned in order to control the analysis time. Static analyzers based on Abstract Interpretation are routinely used to perform the formal verification of flight software in the aerospace industry, for example.

Why SPARTA?

Building an industrial-grade static analysis tool based on Abstract Interpretation from scratch is a daunting task that requires the attention of experts in the field. The purpose of SPARTA is to drastically simplify the engineering of Abstract Interpretation by providing a set of software components that have a simple API, are highly performant and can be easily assembled to build a production-quality static analyzer. By encapsulating the complex implementation details of Abstract Interpretation, SPARTA lets the tool developer focus on the three fundamental axes in the design of an analysis:

- **Semantics:** the program properties to analyze (range of numerical variables, aliasing relation, etc.)
- **Partitioning:** the granularity of the properties analyzed (intraprocedural, interprocedural, context-sensitive, etc.)
- **Abstraction:** the representation of the program properties (sign, interval, alias graph, etc.)

SPARTA is an acronym that stands for **S**emantics, **PART**itioning and **A**bstraction.

facebook / SPARTA Public

Watch 21 | Unstar 502 | Fork 33

Code | Issues | Pull requests | Actions | Projects | Security | Insights

main | 1 branch | 0 tags | Go to file | Add file | Code

Xazax-hun and **facebook-github-bot** Fix typo in comments. (#16) ... 5d8b40b on Jun 21 | 98 commits

.circleci	Change boost download script + Fix OSS CI	6 months ago
cmake_modules	Upgrade to C++17 (#15)	6 months ago
include	Fix typo in comments. (#16)	5 months ago
test	Apply clang-format update fixes	10 months ago
.clang-format	Initial commit	3 years ago
.gitignore	Initial commit	3 years ago
CMakeLists.txt	change how tests are linked and run	13 months ago
CODE_OF_CONDUCT.md	Adopt Contributor Covenant	2 years ago
CONTRIBUTING.md	Initial commit	3 years ago
LICENSE	Initial commit	3 years ago
README.md	change how tests are linked and run	13 months ago
SPARTA.png	Initial commit	3 years ago
get_boost.sh	Change boost download script + Fix OSS CI	6 months ago
run_all_tests.sh	CI should fail when there's a test failing	13 months ago

About

SPARTA is a library that provides the basic blocks for building high-performance static code analyzers based on Abstract Interpretation.

Readme

MIT License

Code of conduct

Releases

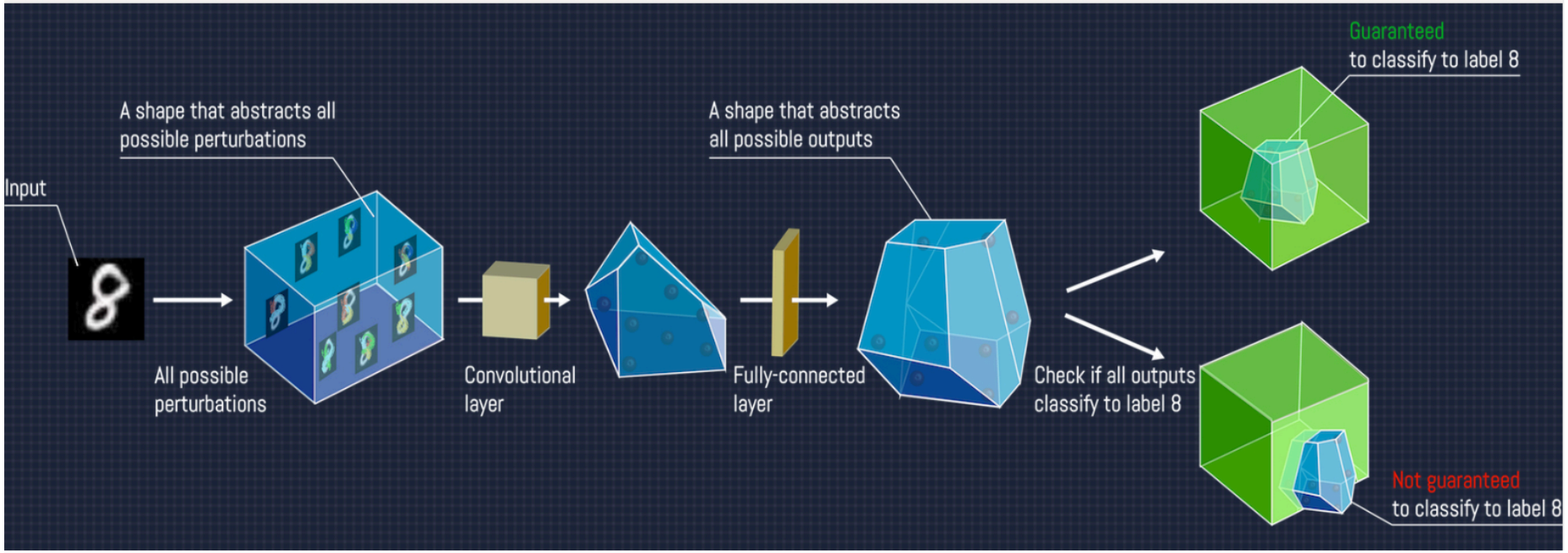
No releases published

Packages

No packages published

Contributors 16

In the SafeAI project at the SRI lab, ETH Zurich, we explore new methods and systems which can ensure Artificial Intelligence (AI) systems such as deep neural networks are more robust, safe and interpretable. Our work tends to sit at the intersection of machine learning, optimization and symbolic reasoning methods. For example, among other results, we recently introduced new approaches and systems that can certify and train deep neural networks with symbolic methods, illustrated in the figure below.



eth-sri / eran

Watch 16 | Unstar 94 | Fork 17

Code | Issues 3 | Pull requests 0 | Projects 0 | Wiki | Security | Insights

ETH Robustness Analyzer for Deep Neural Networks

116 commits | **2** branches | **0** releases | **5** contributors | Apache-2.0

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

maurerjo Remove call of not yet implemeted method. Latest commit `4c3de1a` 4 days ago

data	added support for acasxu	7 months ago
testing	adhere to flag conventions	7 days ago
tf_verify	Remove call of not yet implemeted method.	4 days ago
LICENSE	initial commit	last year
README.md	Add normalization option in readme	19 days ago
gurobi_setup_path.sh	Fix setup instructions	5 months ago
install.sh	added area heuristic enable/disable flag for deeppoly	6 months ago
overview.png	added overview	last year
requirements.txt	Fix ONNX translation, without gather	2 months ago

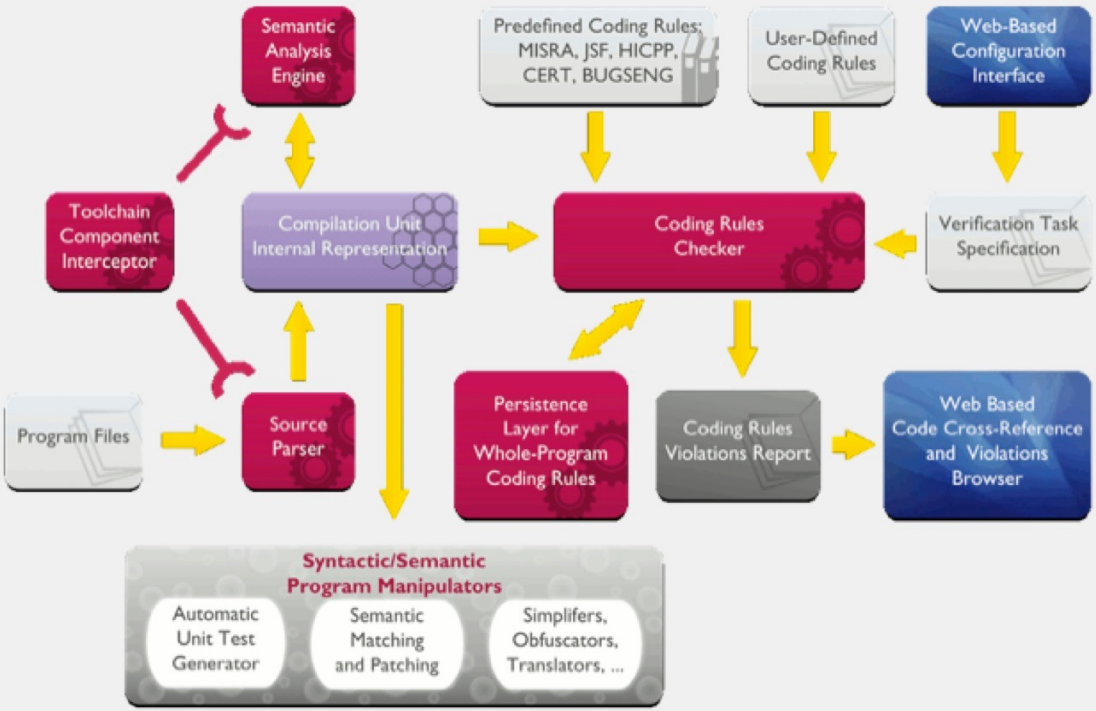
bugSeng

Latest news

2012/04/16 PPL 0.12.1 has been released
2012/03/30 New paper to appear on Information and Computation
2012/02/27 PPL 0.12 has been released
2011/11/04 The PPL has a new logo and web site

eclair

A flexible and extensible framework for the analysis of C and C++ source-code. Among the many possible applications, ECLAIR automatically finds violations of coding rules, either user-defined or taken from popular coding standards such as MISRA C/C++, High-Integrity C++, CERT C/C++, JSF C++, Netrino Embedded C.



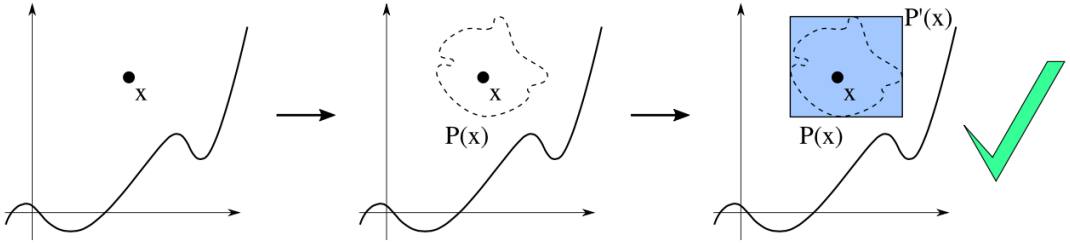
Catch the bugs before they bite you

README.md

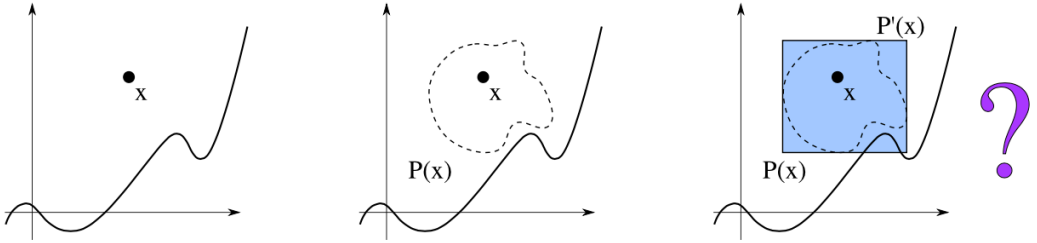
SAVer

SAVer (**S**VM **A**bstract **Ver**ifier) is an abstract interpretation based tool for proving properties of SVMs, in particular we aim at proving robustness or vulnerability properties of classifiers.

Given a point *x* and perturbation function *P*, SAVer symbolically computes an overapproximation of *P(x)*, the region of (possibly infinite) points which corresponds to perturbations of *x*, and runs an abstract version of the SVM on it, returning a superset of the labels associated to points in *P(x)*. Whenever such set contains a single label, the concrete SVM classifier is definitively robust on point *x* for perturbation *P*.



When SAVer returns more than one label, it may happen either due to the classifier really being not robust, or because of a loss of precision induced by the abstraction process.



More information can be found on Robustness Verification of Support Vector Machines.

marco-zanella Added support for custom perturbations from file `b98355d` on Sep 20 **12** commits

doc	Initial commit	12 months ago
src	Added support for custom perturbations from file	2 months ago
.gitignore	Added .gitignore	4 months ago
LICENSE	Initial commit	12 months ago
README.md	Added link to paper	12 months ago

README.md

Silva

Silva (**S**ilvarum **I**nterpretatione **L**ator **V**alens **A**nalysis) is an abstract interpretation based tool for proving properties of tree ensembles classifiers, in particural we aim at proving stability-related properties of classifiers.

Given a point *x* and a perturbation *P*, silva symbolically computes an overapproximation of *P(x)*, the region of (possibly infinite) points which corresponds to perturbations of *x*, and runs an abstract version of the forest classifier on it, returning a superset of the labels associated to points in *P(x)*. Whenever such set yelds the same output as the classification ona single point in *P(x)*, the concrete classifier is definitively stable on point *x* for perturbation *P*.

When silva returns more labels, it may happen due to the classifier really being not stable, or because of a loss of precision induced by the abstract process. For forests consisting of univariate hard splits only (i.e. *x_i < k*) and *l_|inf* perturbation silva becomes *complete*: every single label in the output set is guaranteed to be associated to at least one point in *P(x)*, thus no false results can be returned.

More information can be found in Abstract interpretation of decision tree ensemble classifiers.

Why not all software developers use static analysis tools?

Irresponsibility

Computer engineering is the only technology where developers are not responsible for their errors, even the trivial ones:

DISCLAIMER OF WARRANTIES. *. . . MICROSOFT AND ITS SUPPLIERS PROVIDE THE SOFTWARE, AND SUPPORT SERVICES (IF ANY) AS IS AND WITH ALL FAULTS, AND MICROSOFT AND ITS SUPPLIERS HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE. . . .*

The future

- Safety and security does matter to the general public
- Computer scientists will ultimately be held responsible for **their** errors
- At least the automatically discoverable ones
- Since this is now part of the state of the art
- Automatic static analysis, verification, etc has a brilliant future.

by Patrick Cousot

CONTRIBUTED ARTICLES

Scaling Static Analyses at Facebook

By Dino Distefano, Manuel Fähndrich, Francesco Logozzo, Peter W. O'Hearn

Communications of the ACM, August 2019, Vol. 62 No. 8, Pages 62-70
10.1145/3338112

Comments

VIEW AS: SHARE:

Static analysis tools are programs that examine, and attempt to draw conclusions about, the source of other programs without running them. At Facebook, we have been investing in advanced static analysis tools that employ reasoning techniques similar to those from program verification. The tools we describe in this article (Infer and Zoncolan) target issues related to crashes and to the security of our services, they perform sometimes complex reasoning spanning many procedures or files, and they are integrated into engineering workflows in a way that attempts to bring value while minimizing friction.

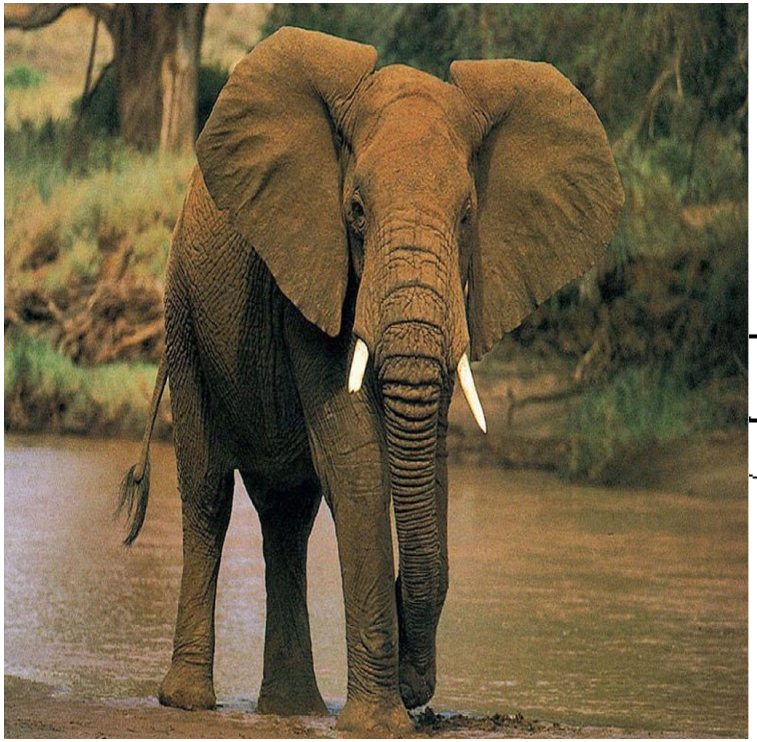
Back to Top

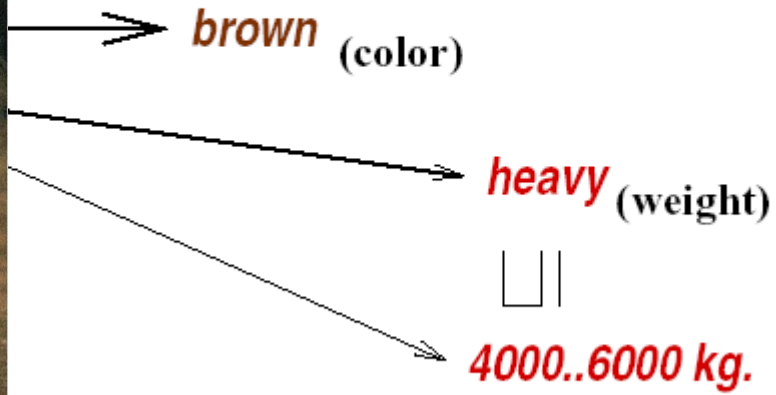
Key Insights

- Advanced static analysis techniques performing deep reasoning about source code can scale to large industrial codebases, for example, with 100-million LOC.
- Static analyses should strike a balance between missed bugs (false negatives) and un-actioned reports (false positives).
- A "diff time" deployment, where issues are given to developers promptly as part of code review, is important to catching bugs early and getting high fix rates.

Abstract Interpretation: Basic Ideas

Abstraction and correctness



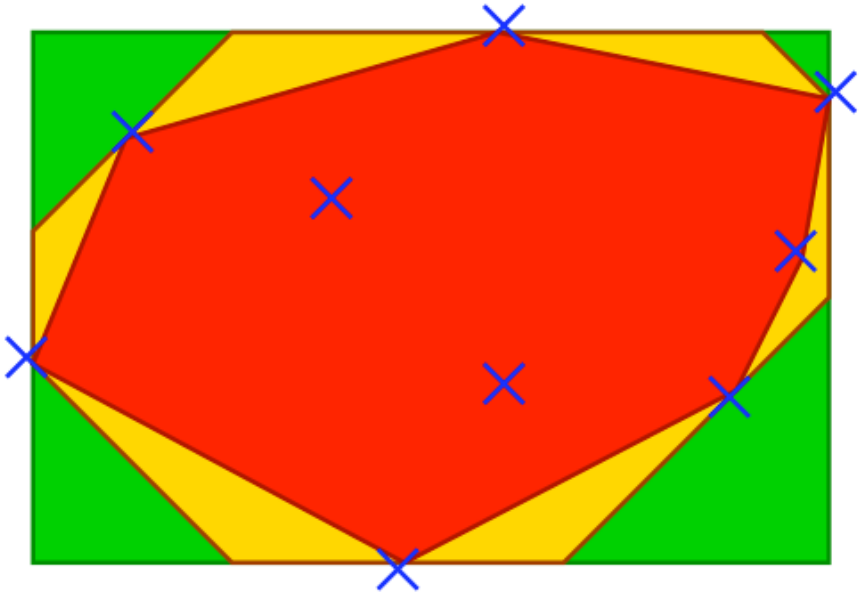


Abstracting a set of 2D points

concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

e.g. run-time values of two float variables

Abstracting a set of 2D points

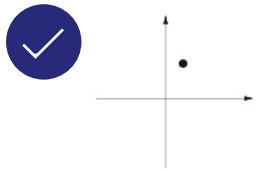


concrete sets $\mathcal{D}$:	$\{(0,3),(5.5,0),(12,7),\ldots\}$	not computable
abstract polyhedra $\mathcal{D}^\sharp_p$:	$6X + 11Y \geq 33 \wedge \cdots$	exponential cost
abstract octagons $\mathcal{D}^\sharp_o$:	$X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$	cubic cost
abstract intervals $\mathcal{D}^\sharp_i$:	$X \in [0,12] \wedge Y \in [0,8]$	linear cost

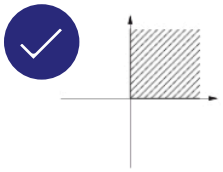
Trade-off between cost and expressiveness / precision

wrt number of variables

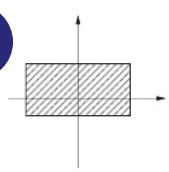
Abstracting a set of 2D points



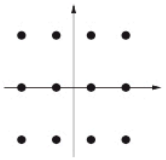
Constant Propagation
$X_i = c_i$
[Kil73]



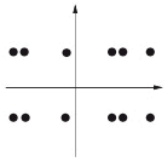
Signs
$X_i \geq 0,\ X_i \leq 0$
[CC76]



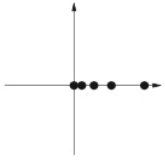
Intervals
$X_i \in [a_i, b_i]$
[CC76]



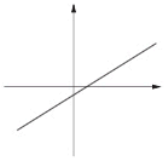
Simple Congruences
$X_i \equiv a_i\ [b_i]$
[Gra89, Gra97]



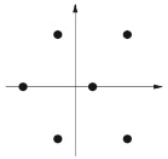
Interval Congruences
$X_i \in \alpha_i[a_i, b_i]$
[Mas93]



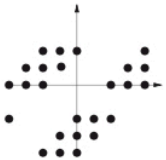
Power Analysis
$X_i \in \alpha_i^{a_i\mathbb{Z}+b_i}, \alpha_i^{[a_i,b_i]}$, etc.
[Mas01]



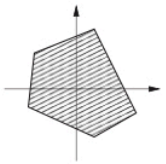
Linear Equalities
$\sum_i \alpha_{ij} X_i = \beta_j$
[Kar76]



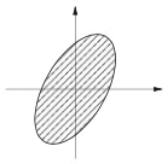
Linear Congruences
$\sum_i \alpha_{ij} X_i \equiv \beta_j\ [\gamma_j]$
[Gra91]



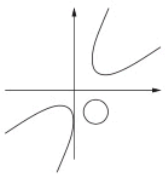
Trapezoidal Congruences
$X_i = \sum_j \lambda_j \alpha_{ij} + \beta_j$
[Mas92]



Polyhedra
$\sum_i \alpha_{ij} X_i \leq \beta_j$
[CH78]

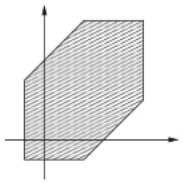


Ellipsoids
$\alpha X^2 + \beta Y^2 + \gamma XY \leq \delta$
[Fer04b]

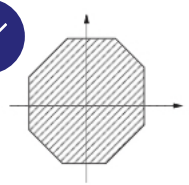


Varieties
$P_i(\vec{X}) = 0,\ P_i \in \mathbb{R}[\mathcal{V}]$
[RCK04a]

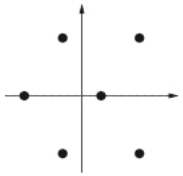
Abstracting a set of 2D points



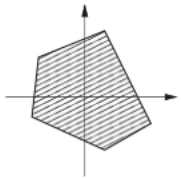
Zones
$X_i - X_j \leq c_{ij}$
[Min01a]



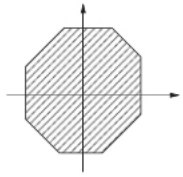
Octagons
$\pm X_i \pm X_j \leq c_{ij}$
[Min01b]



Zone Congruences
$X_i \equiv X_j + \alpha_{ij}\,[\beta_{ij}]$
[Min02]



TVPLI
$\alpha_{ij} X_i + \beta_{ij} X_j \leq c_{ij}$
[SKH02]



Octahedra
$\sum_i \epsilon_{ij} X_i \leq \beta_j,\ \epsilon_{ij} \in \{-1, 0, 1\},\ X_i \geq 0$
[CC04]

Abstract interpretation

Patrick Cousot[12]

THESE

présentée à

Université Scientifique et Médicale de Grenoble
Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR ES SCIENCES MATHEMATIQUES

par

Patrick COUSOT

sur

METHODES ITERATIVES DE CONSTRUCTION
ET D'APPROXIMATION DE POINTS FIXES
D'OPERATEURS MONOTONES SUR UN TREILLIS,
ANALYSE SEMANTIQUE DES PROGRAMMES.

Abstract Interpretation

- A formal technique used since 40+ years (Patrick e Radhia Cousot, 1977) to systematically design abstractions and approximations

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

Most cited POPL paper ever

10. *Conclusion*

It is our feeling that most program analysis techniques may be understood as abstract interpretations of programs. Let us point out global data flow analysis in optimizing compilers (Kildall[73], Morel and Renvoise[76] , Schwartz[75], Ullman[75], Wegbreit[75], ...), type verification (Naur[65], ...), type discovery (Cousot[76'], Sintzoff[72], Tenenbaum[74], ...), program testing (Henderson [75], ...) symbolic evaluation of programs (Hewitt et al.[73], Karr[76], ...), program performance analysis (Wegbreit[76], ...), formalization of program semantics (Hoare and Lauer[74], Ligler[75], Manna and Shamir[75], ...), verification of program correctness (Floyd[67], Park[69], Sintzoff[75], ...), discovery of inductive invariants (Katz and Manna[76], ...), proofs of program termination (Sites[74], ...), program transformation (Sintzoff [76], ...), ...

There is a fundamental unity between all these apparently unrelated program analysis techniques : a new interpretation is given to the program text which allows to built an often implicit system of equations. The problem is either to verify that a solution provided by the user is correct, or to discover or approximate such a solution.

The mathematical model we studied in this paper is certainly the weakest which is necessary to unify these techniques, and therefore should be of very general scope. It can be considerably enriched for particular applications so that more powerful results may be obtained.

The general idea

- The starting point is a concrete semantics that provides the meaning of program commands into a given computational domain
- We need an abstract domain, which models some properties of interest of concrete computations and leaves out the remaining information
- We then have an abstract semantics that allows us “to abstractly execute” a program on the abstract domain in order to compute the program properties modeled by the abstract domain.
- The computation of the abstract semantics typically involves fixpoint computations
- When necessary, e.g. by efficiency reasons, one could perform further correct approximations of the abstract semantics

Abstract interpretation

```
(S0)
assume X in [0,1000];
(S1)
I := 0;
(S2)
while (S3) I < X do
    (S4)
    I := I + 2;
    (S5)
(S6)
```

program

$$\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I}, \mathtt{X}\} \to \mathbb{Z})$$

$$\begin{aligned}
\mathcal{S}_0 &= \{\, (i,x) \mid i,x \in \mathbb{Z} \,\} && = \top \\
\mathcal{S}_1 &= \{\, (i,x) \in \mathcal{S}_0 \mid x \in [0,1000] \,\} && = F_1(\mathcal{S}_0) \\
\mathcal{S}_2 &= \{\, (0,x) \mid \exists i,\ (i,x) \in \mathcal{S}_1 \,\} && = F_2(\mathcal{S}_1) \\
\mathcal{S}_3 &= \mathcal{S}_2 \cup \mathcal{S}_5 \\
\mathcal{S}_4 &= \{\, (i,x) \in \mathcal{S}_3 \mid i < x \,\} && = F_4(\mathcal{S}_3) \\
\mathcal{S}_5 &= \{\, (i+2,x) \mid (i,x) \in \mathcal{S}_4 \,\} && = F_5(\mathcal{S}_4) \\
\mathcal{S}_6 &= \{\, (i,x) \in \mathcal{S}_3 \mid i \geq x \,\} && = F_6(\mathcal{S}_3)
\end{aligned}$$

semantics

F_i are called transfer functions

Concrete semantics $\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I}, \mathtt{X}\} \to \mathbb{Z})$:

- strongest invariant (and an inductive invariant)
- not computable in general
- smallest solution of a system of equations

Abstract interpretation

```
(S0)
assume X in [0,1000];
(S1)
I := 0;
(S2)
while (S3) I < X do
    (S4)
    I := I + 2;
    (S5)
(S6)
```

program

$$
\begin{aligned}
&\mathcal{S}_i^\sharp \in \mathcal{D}^\sharp \\
&\mathcal{S}_0^\sharp = \top^\sharp \\
&\mathcal{S}_1^\sharp = F_1^\sharp(\mathcal{S}_0^\sharp) \\
&\mathcal{S}_2^\sharp = F_2^\sharp(\mathcal{S}_1^\sharp) \\
&\mathcal{S}_3^\sharp = \mathcal{S}_2^\sharp \cup^\sharp \mathcal{S}_5^\sharp \\
&\mathcal{S}_4^\sharp = F_4^\sharp(\mathcal{S}_3^\sharp) \\
&\mathcal{S}_5^\sharp = F_5^\sharp(\mathcal{S}_4^\sharp) \\
&\mathcal{S}_6^\sharp = F_6^\sharp(\mathcal{S}_3^\sharp)
\end{aligned}
$$

semantics

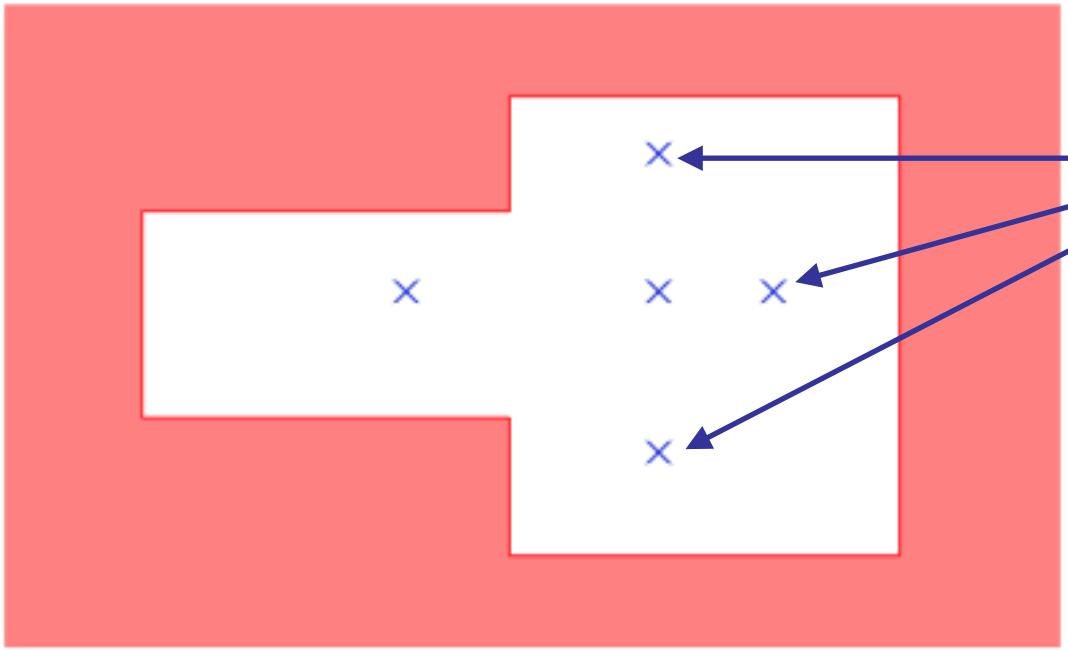
$F_i^\#$ are called abstract transfer functions

Abstract semantics $\mathcal{S}_i^\sharp \in \mathcal{D}^\sharp$:

- $\mathcal{D}^\sharp$ is a subset of properties of interest with a machine representation (approximation)
- $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ over-approximates the effect of $F : \mathcal{D} \to \mathcal{D}$ in $\mathcal{D}^\sharp$ (with effective algorithms)

Correctness proof and false alarms

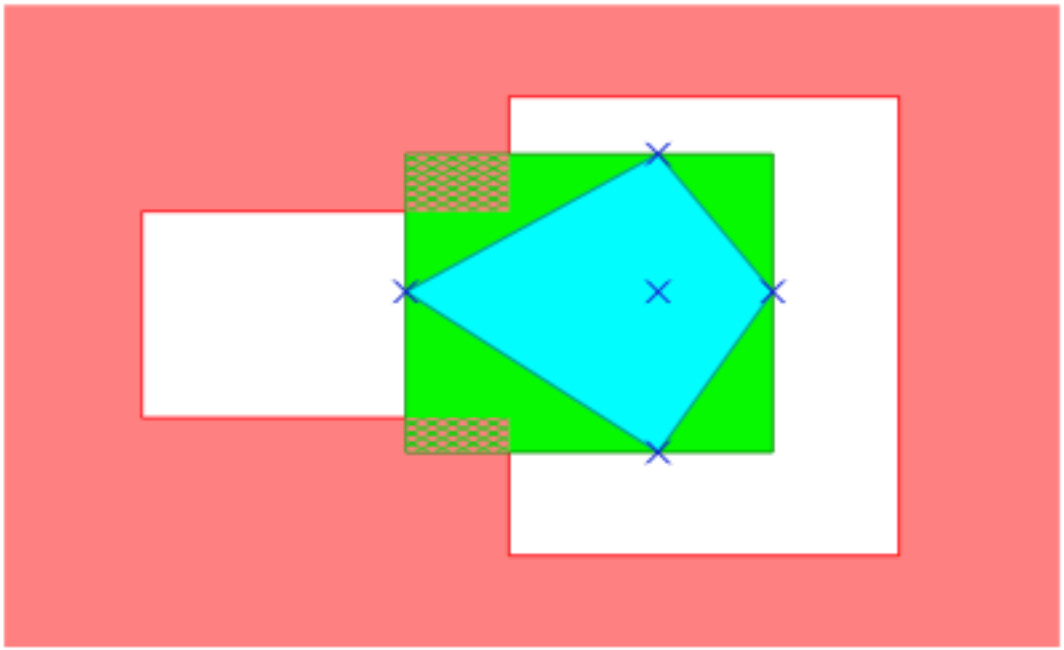
red area of bad states



The program is correct (blue ∩ red = ∅).

Correctness proof and false alarms

red area of bad states

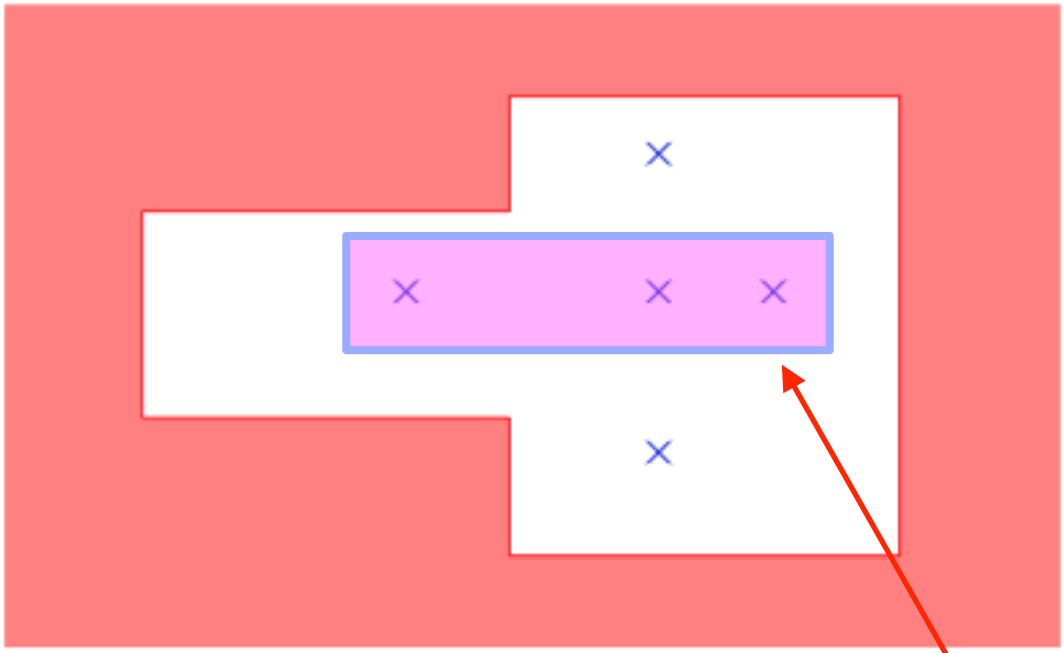


The program is correct (blue $\cap$ red $= \emptyset$).
The polyhedra domain can prove the correctness (cyan $\cap$ red $= \emptyset$).
The interval domain cannot (green $\cap$ red $\neq \emptyset$, false alarm).

false alarm
=
false positive

Correctness proof and false alarms

red area of bad states

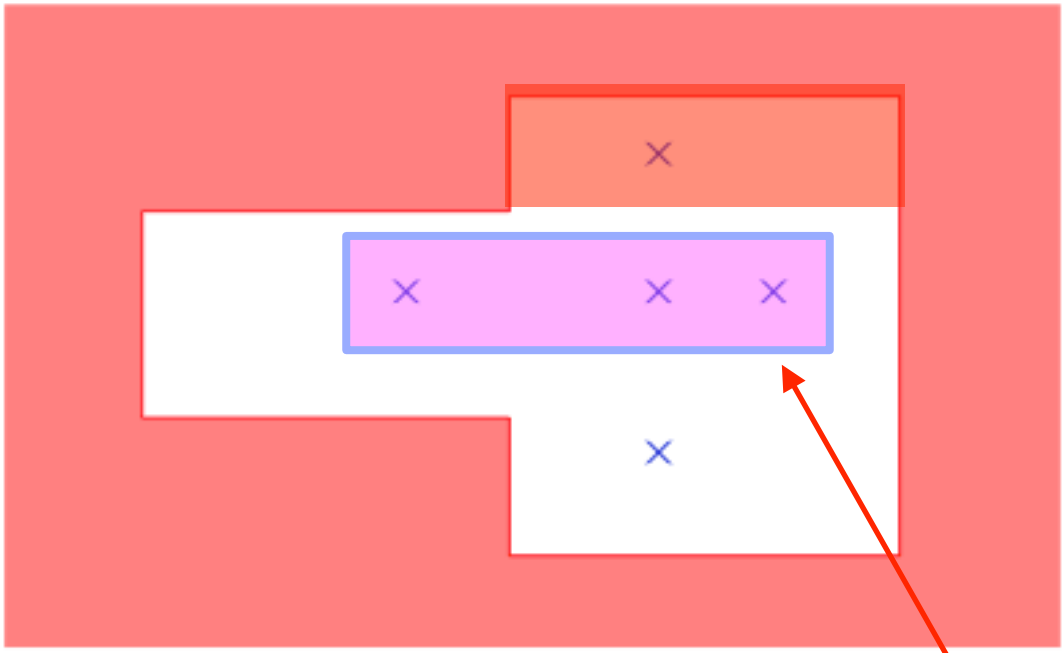


(lucky) unsound analysis

The program is correct (blue $\cap$ red $= \emptyset$).

Correctness proof and false alarms

red area of bad states



Unsound analysis ⇒ False negative

The program is bugged

Brahmagupta

Brahmagupta (Sanskrit: ब्रह्मगुप्त; (598–c.670 CE) was an Indian mathematician and astronomer who wrote two important works on Mathematics and Astronomy: the *Brāhmasphuṭasiddhānta* (Extensive Treatise of Brahma) (628), a theoretical treatise, and the *Khaṇḍakhādyaka*, a more practical text.

Brahmagupta

Born	598 CE
Died	c.670 CE
Fields	Mathematics, Astronomy
Known for	Zero, modern Number system

The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative; [...]

18.32. A negative minus zero is negative, a positive [minus zero] positive; zero [minus zero] is zero. When a positive is to be subtracted from a negative or a negative from a positive, then it is to be added.

18.33. The product of a negative and a positive is negative, of two negatives positive, and of positives positive; the product of zero and a negative, of zero and a positive, or of two zeros is zero.

18.34. A positive divided by a positive or a negative divided by a negative is positive; a zero divided by a zero is zero; a positive divided by a negative is negative; a negative divided by a positive is [also] negative.

wrong

Concrete semantics

- We will first consider the denotational semantics of our **While** language
- Let us begin with basic expressions that only allow to multiply integer numbers.

  Exp $\ni$ e ::= n | e*e

- The concrete semantics of these expressions is given by a function $\mathcal{S}$ defined as follows:

  $\mathcal{S}$: **Exp** $\rightarrow$ $\mathbb{Z}$

  $\mathcal{S}(n) = n_{\mathbb{Z}}$

  $\mathcal{S}(e_1 * e_2) = \mathcal{S}(e_1) \times \mathcal{S}(e_2)$

Abstract semantics

We consider a **compositional** abstract semantics σ that computes the **sign** of expressions:

$$\sigma : \textbf{Exp} \rightarrow \{-,0,+\}$$

$$\sigma(n) = \begin{cases} - & \text{if } n<0 \\ 0 & \text{if } n=0 \\ + & \text{if } n>0 \end{cases}$$

$\times^a$	-	0	+
-	+	0	-
0	0	0	0
+	-	0	+

$$\sigma(e_1 * e_2) = \sigma(e_1) \times^a \sigma(e_2)$$

compositional definition

Correctness

- This abstract semantics σ is **correct** (or **sound**), namely, it correctly computes the sign of expressions

- This obvious proof relies on structural induction on the expression e and exploits the rule of signs for integer multiplications (a product of positive numbers is positive, etc.).

For any expression $e \in$ **Exp**:

$$\sigma(e) = - \Rightarrow \mathcal{S}(e) < 0 \qquad \text{(indeed } \Leftrightarrow \text{ holds, completeness)}$$

$$\sigma(e) = 0 \Rightarrow \mathcal{S}(e) = 0 \qquad \text{(indeed } \Leftrightarrow \text{ holds, completeness)}$$

$$\sigma(e) = + \Rightarrow \mathcal{S}(e) > 0 \qquad \text{(indeed } \Leftrightarrow \text{ holds, completeness)}$$

A different perspective

Let us map each abstract value in {-,0,+} to the set of concrete values that it represents:

$\gamma : \{-,0,+\} \rightarrow \wp(\mathbb{Z})$

γ concretization function

$\gamma(-) = \{x \in \mathbb{Z} \mid x<0\}$

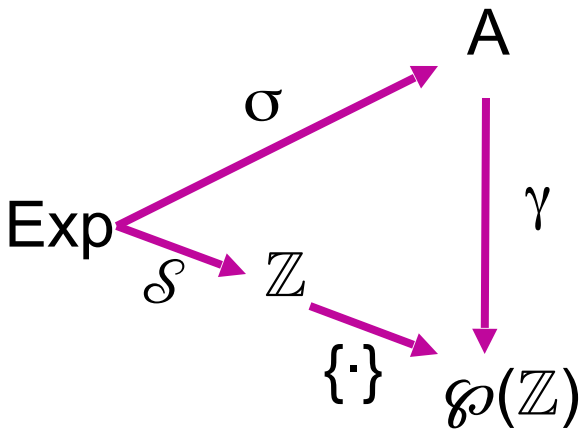
$\gamma(0) = \{0\}$

$\gamma(+) = \{x \in \mathbb{Z} \mid x>0\}$

Concretization

- This concretization function γ maps an abstract value to a set of concrete values
- Let us denote the abstract domain by A

$$\mathcal{S}(e) \in \gamma(\sigma(e))$$



Abstract interpretation

- We have specified a **very naive abstract interpretation**.
  - Abstract computations happen in an abstract domain
  - In this case, the abstract domain is {+,0,-}

- The abstract semantics is correct, meaning that σ is a correct approximation of the concrete semantics $\mathcal{S}$, i.e.,
  $\{\mathcal{S}(e)\} \subseteq \gamma(\sigma(e))$

- The concretization map establishes the relationship between the notions of approximation in the concrete and abstract domains

Adding -

- Let us add to expressions the unary operator that changes the sign of an expression to its opposite

Exp ∋ e ::= n | e*e | -e

$\mathcal{S}(-e) = -\mathcal{S}(e)$

$\sigma(-e) = -^a\sigma(e)$ where $-^a(-) = +,\ -^a(0) = 0,\ -^a(+) = -$

compositional definition

Adding +

- The addition operation needs some care, since the abstract domain is not closed with respect to this operation

$$\mathcal{S}(e_1 + e_2) = \mathcal{S}(e_1) + \mathcal{S}(e_2)$$

$$\sigma(e_1 + e_2) = \sigma(e_1) +^a \sigma(e_2)$$

compositional definition

$+^a$	-	0	+
-	-	-	?
0	-	0	+
+	?	+	+

- Which abstract value should we use for the sum of two integer numbers having opposite signs?

Solution

We add a new abstract value $\top$ that represents any integer number

$$\gamma(\top) = \mathbb{Z}$$

$+^a$	-	0	+	$\top$
-	-	-	$\top$	$\top$
0	-	0	+	$\top$
+	$\top$	+	+	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

Extending operations

Since the abstract domain contains a new element, we need to extend the previously defined abstract operations

$\times^a$	-	0	+	$\top$
-	+	0	-	$\top$
0	0	0	0	0
+	-	0	+	$\top$
$\top$	$\top$	0	$\top$	$\top$

$-^a(-) = +, \quad -^a(0) = 0,$
$-^a(+) = -, \quad -^a(\top) = \top$

Examples

- In some cases we observe a **loss of information** in the abstract semantics

  $\mathcal{S}((1+2) - 3) = 0$

  $\sigma((1+2)+ -3) = (+ +^a +) +^a - \; = \; + +^a - \; = \; \top$

- In some cases this loss of information does not happen, the abstract computation is **complete**

  $\mathcal{S}((5*4) + 6) = 26$

  $\sigma((5*4) + 6) = (+ \times^a +) +^a + \; = \; + +^a + \; = \; +$

Adding divisions

- Integer divisions do not raise issues except for the case of divisions by zero
- What is the result of dividing by zero a set of integer numbers? The empty set, which therefore represents a run-time error.
- Thus, the concrete semantics needs to assume its values on the powerset of $\mathbb{Z}$, namely, $\mathcal{S}$: **Exp** $\rightarrow \wp(\mathbb{Z})$

- The empty set of integers is therefore represented by a new abstract element $\perp$, whose meaning is **sure abnormal termination**, and consequently we need to extend the operations to take care of $\perp$

$\gamma(\perp)=\emptyset$

/a	-	0	+	$\top$	$\perp$
-	+	0	-	$\top$	$\perp$
0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
+	-	0	+	$\top$	$\perp$
$\top$	$\top$	0	$\top$	$\top$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

$\perp +^a x = \perp = x +^a \perp$

$\perp \times^a x = \perp = x \times^a \perp$

$-^a (\perp) = \perp$

Adding divisions

$\gamma(\bot)=\emptyset$

$/^a$	-	0	+	$\top$	$\bot$
-	+	0	-	$\top$	$\bot$
0	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
+	-	0	+	$\top$	$\bot$
$\top$	$\top$	0	$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Thus, $\bot$ means sure abnormal termination (run-time error)

Notice that + $/^a$ $\top$must be defined to be $\top$ in order to be correct:

$\mathcal{S}(2/(3\text{-}1))=\{1\} \subseteq \gamma(\sigma(2/(3\text{-}1))) = \gamma(+ /^a \top)$

$\mathcal{S}(2/(3\text{-}3))=\emptyset \subseteq \gamma(\sigma(2/(3\text{-}3))) = \gamma(+ /^a \top)$

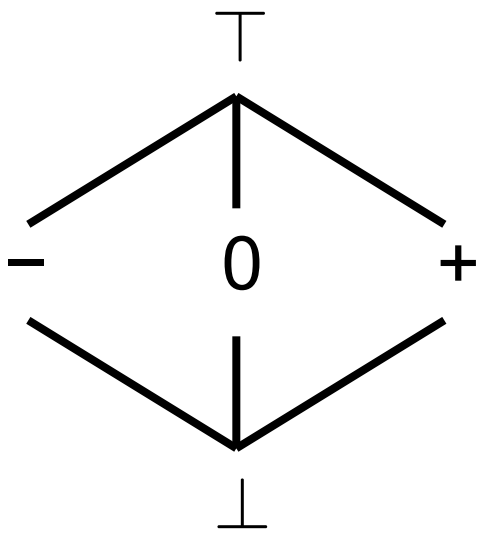
$\mathcal{S}(2/(3\text{-}5))=\{\text{-}1\} \subseteq \gamma(\sigma(2/(3\text{-}5))) = \gamma(+ /^a \top)$

The abstract domain

- The abstract domain is a poset whose partial order represents the notion of approximation/ precision
- The partial order must be coherent with the concretization map:

$$x \leq y \Leftrightarrow \gamma(x) \subseteq \gamma(y)$$

- Each subset of the abstract domain has lub and glb: it is a lattice



The abstraction function

- The abstraction function α is the counterpart of the concretization map γ.
- The function α maps a set S of concrete values into the most precise abstract value that represents S.
- In our example

$$\alpha : \wp(\mathbb{Z}) \rightarrow A \qquad \alpha(S) = \begin{cases} \bot & \text{if } S = \emptyset \\ - & \text{if } S \neq \emptyset,\ S \subseteq \mathbb{Z}_{<0} \\ 0 & \text{if } S = \{0\} \\ + & \text{if } S \neq \emptyset,\ S \subseteq \mathbb{Z}_{>0} \\ \top & \text{otherwise} \end{cases}$$

General Definition

- An Abstract Interpretation consists of:
  - An abstract domain A and a concrete domain C
  - A and C are (usually) complete lattices. The order encodes the precision/approximation relation (smaller means more precise)
  - Monotone concretization and abstraction maps γ and α, that give rise to a Galois insertion.
  - Abstract operations that correctly mimick on A their concrete behavior on C.

$\gamma_{sign}(\bot)=\emptyset$, if $x = \bot$

$\gamma_{sign}(+)=\{y\in\mathbb{Z}|y>0\}$

$\gamma_{sign}(0+)=\{y\in\mathbb{Z}|y\geq 0\}$

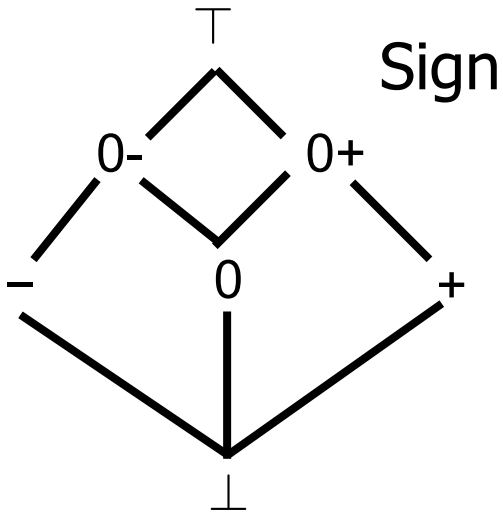
$\gamma_{sign}(0)=\{0\}$

$\gamma_{sign}(0-)=\{y\in\mathbb{Z}|y\leq 0\}$

$\gamma_{sign}(-)=\{y\in\mathbb{Z}|y<0\}$

$\gamma_{sign}(\top)=\mathbb{Z}$

Further example



$\alpha_{sign}(S) = \wedge\{a\in Sign \mid S \subseteq \gamma_{sign}(a)\}$

$\alpha_{sign}(\{0,1,3\}) = 0+$

$\gamma_{sign}(0+) \supseteq \{0,1,3\}, \{3,5,2\}, \ldots$

$\gamma_{sign}(\alpha_{sign}(\{0,1,3\})) \supseteq \{0,1,3\}$

$\alpha_{sign}(\gamma_{sign}(0+)) = 0+$

Galois connection

Galois connection

From Wikipedia, the free encyclopedia

In mathematics, especially in order theory, a **Galois connection** is a particular correspondence (typically) between two partially ordered sets (posets). Galois connections find applications in various mathematical theories. They generalize the fundamental theorem of Galois theory about the correspondence between subgroups and subfields, discovered the French mathematician Évariste Galois.

Galois connection

(α, C, A, γ) **Galois connection** (GC) if

(1) A and C poset

(2) α : C $\rightarrow$ A monotone (abstraction map)

(3) γ : A $\rightarrow$ C monotone (concretization map)

(4) $\forall c \in C.\ c \leq_C \gamma(\alpha(c))$

(5) $\forall a \in A.\ \alpha(\gamma(a)) \leq_A a$

Approximation relations: $\alpha(c) \leq_A a$ in A and $c \leq_C \gamma(a)$ in C

Intuition: (2) and (3) mean that α and γ preserve the approximation ordering; (4) means that $\alpha(c)$ provides a correct abstract approximation in A for the concrete value *c*; (5) means that $\gamma(a)$ is correctly approximated in A by *a*.

Adjunctions

(α, C, A, γ) **adjunction** if

(1) A and C poset
(2) $\alpha : C \rightarrow A$ (abstraction map)

(3) $\gamma : A \rightarrow C$ (concretization map)

(4) $\forall c \in C.\ \forall a \in A.\ \alpha(c) \leq_A a \Leftrightarrow c \leq_C \gamma(a)$

Intuition: the approximation relations $\alpha(c) \leq_A a$ in A and $c \leq_C \gamma(a)$ in C are equivalent

Theorem

(α,C,A,γ) is a GC $\Leftrightarrow$ (α,C,A,γ) is an adjunction

Galois insertion

A GC (α, C, A, γ) is a **Galois insertion** (GI) if one of the following equivalent conditions holds:

1) α surjective
2) γ injective
3) $\forall a \in A.\ \alpha(\gamma(a)) = a$

GC but not GI

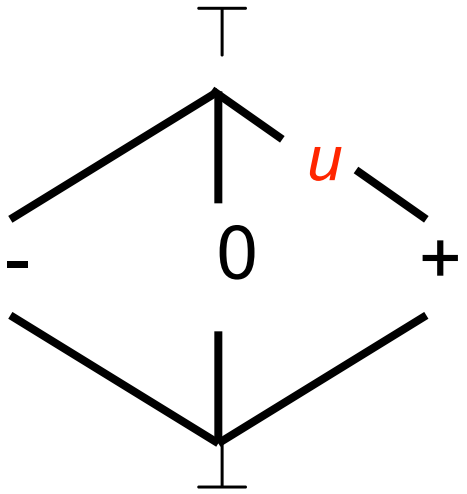
$\gamma(\bot)=\emptyset$

$\gamma(0)=\{0\}$

$\gamma(-)=\{y\in\mathbb{Z}|y<0\}$

$\gamma(u)=\gamma(+)=\{z\in\mathbb{Z}|z > 0\}$

$\gamma(\top)=\mathbb{Z}$



$\alpha(\{1,3\}) = +$

Abstract value *u* is "useless"

GIs from GCs

How to “transform” a GC (α, C, A, γ) into a GI by eliminating the “useless” abstract values?

We can simply consider a “reduced” abstract domain $A^{red} = \alpha(C)$ and the corresponding restrictions on A^{red} of the abstraction and concretization functions:

$\alpha^{red} : C \rightarrow A^{red}$ where $\alpha^{red}(c) = \alpha(c)$

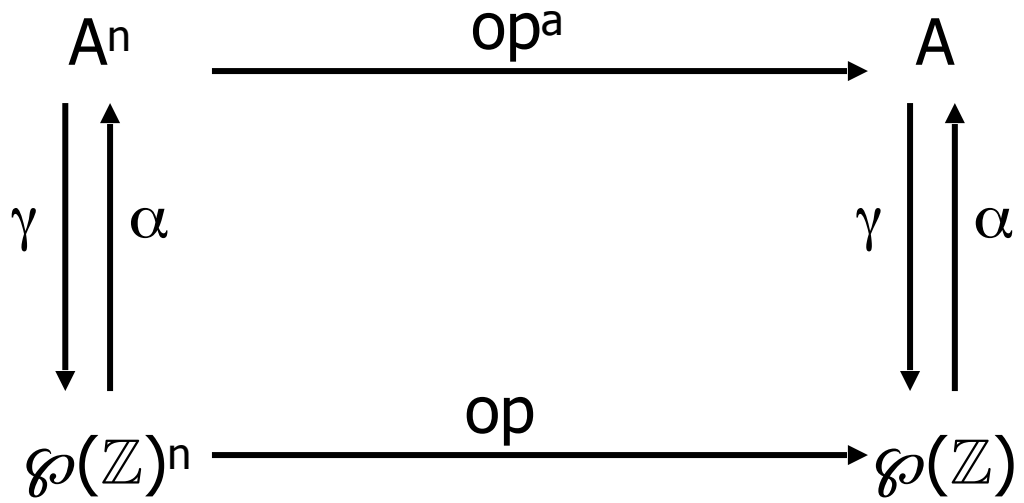
$\gamma^{red} : A^{red} \rightarrow C$ where $\gamma^{red}(a) = \gamma(a)$

(α^{red}, C, A^{red}, γ^{red}) is now a GI which has the same “expressive power” of the original GC since $\alpha^{red}(C) = \alpha(C)$.

By Galois connection, we therefore have two equivalent ways of stating soundness of the abstract semantics:

$$\{\mathcal{S}(e)\} \subseteq \gamma(\sigma(e)) \Leftrightarrow \alpha(\{\mathcal{S}(e)\}) \leq \sigma(e)$$

Correctness of abstract operations



Correctness of abstract operations

Let op:$\wp(\mathbb{Z})^n \to \wp(\mathbb{Z})$ be a monotonic n-ary concrete operation.

Let op^a:$A^n \to A$ be a corresponding monotonic n-ary abstract operation.

The abstract operation op^a is **correct** w.r.t. op when:
(1) for any $\langle a_1,...,a_n\rangle \in A^n$, $op(\gamma(a_1),...,\gamma(a_n)) \leq_C \gamma(op^a(a_1,...,a_n))$

Correctness for op^a can be equivalently defined as follows:
(2) for any $\langle c_1,...,c_n\rangle \in C^n$, $\alpha(op(c_1,...,c_n)) \leq_A op^a(\alpha(c_1),...,\alpha(c_n))$

(3) for any $\langle c_1,...,c_n\rangle \in C^n$, $op(c_1,...,c_n) \leq_C \gamma(op^a(\alpha(c_1),...,\alpha(c_n))$

(4) for any $\langle a_1,...,a_n\rangle \in A^n$, $\alpha(op(\gamma(a_1),...,\gamma(a_n))) \leq_A op^a(a_1,...,a_n)$

Best correct approximation

- Correctness guarantees that the result of applying an abstract operation is a correct approximation of the result of applying the corresponding concrete operation.
- For any concrete operation op, one can always define the so-called best correct approximation op^A of op on the abstract domain A as follows:

$$op^A(a_1,...,a_n) \triangleq \alpha(op(\gamma(a_1),...,\gamma(a_n))$$

Correctness proof

Let us prove by structural induction on e$\in$**Exp** (to simplify the proof, division is not included) that we have:

$$\{\mathcal{S}(e)\} \subseteq \gamma(\sigma(e))$$

or, equivalently by GI, $\alpha(\{\mathcal{S}(e)\}) \leq \sigma(e)$

- op^s is a binary syntactic operation, i.e., $e_1\ op^s\ e_2 \in$**Exp**
- op : $\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}$ is a corresponding binary integer operation
- op^c : $\wp(\mathbb{Z})\times\wp(\mathbb{Z}) \to \wp(\mathbb{Z})$ is the monotone pointwise lifting of op, where

  $op^c(S_1,S_2) \triangleq \{op(z_1,z_2) \in \mathbb{Z} \mid z_1\in S_1,\ z_2\in S_2\}$
- op^a : $A\times A \to A$ is a correct abstract operation w.r.t. op^c

Correctness proof

Let us prove by structural induction on $e \in \mathbf{Exp}$ that:

$$\{\mathcal{S}(e)\} \subseteq \gamma(\sigma(e))$$

- **Basis**:
  $\{\mathcal{S}(n)\} = \{n\} \subseteq \gamma(\alpha(\{n\})) = \gamma(\sigma(n))$ by Galois connection
- **Inductive step**:
  $\{\mathcal{S}(e_1\ op^s\ e_2)\} = \{\mathcal{S}(e_1)\ op\ \mathcal{S}(e_2)\} = \{\mathcal{S}(e_1)\}\ op^c\ \{\mathcal{S}(e_2)\}$

- by inductive hypothesis on e_1 and e_2, we have $\{\mathcal{S}(e_i)\} \subseteq \gamma(\sigma(e_i))$

- by monotonicity of op^c, we have $\{\mathcal{S}(e_1)\}\ op^c\ \{\mathcal{S}(e_2)\} \subseteq \gamma(\sigma(e_1))\ op^c\ \gamma(\sigma(e_2))$

- by correctness of op^a, $\gamma(\sigma(e_1))\ op^c\ \gamma(\sigma(e_2)) \subseteq \gamma(\sigma(e_1)\ op^a\ \sigma(e_2))$

Thus: $\{\mathcal{S}(e_1)\}\ op^c\ \{\mathcal{S}(e_2)\} \subseteq \gamma(\sigma(e_1)\ op^a\ \sigma(e_2)) = \gamma(\sigma(e_1\ op^s\ e_2))$

We conclude that: $\{\mathcal{S}(e_1\ op^s\ e_2)\} \subseteq \gamma(\sigma(e_1\ op^s\ e_2))$

Abstract denotational semantics

Denotational semantics

$\mathbf{Aexp} \ni e ::= x \mid n \mid e_1 \text{ op } e_2$

$\mathbf{Bexp} \ni b ::= \mathit{true} \mid \mathit{false} \mid e_1 = e_2 \mid e_1 < e_2 \mid b_1 \wedge b_2 \mid \neg b$

$\mathbf{While} \ni S ::= x := e \mid \mathbf{skip} \mid S_1; S_2 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \mid \mathbf{while}\ b\ \mathbf{do}\ S$

$\mathcal{A} : \mathbf{Aexp} \to \mathbf{State} \to \mathbb{Z}$

$\mathcal{B} : \mathbf{Bexp} \to \mathbf{State} \to \mathbb{T}$

$\mathcal{S}_{\text{ds}} : \mathbf{While} \to \mathbf{State} \hookrightarrow \mathbf{State}$

Collecting denotational semantics

"Collecting" means lifting the denotational semantics to the powerset of states $\wp(\mathbf{State})$. Since sets in $\wp(\mathbf{State})$ denote state properties, the collecting semantics can be viewed as a property transformer.

$$\mathcal{A}_c : \mathbf{Aexp} \to \wp(\mathbf{State}) \to \wp(\mathbb{Z})$$

$$\mathcal{A}_c[\![e]\!]T \triangleq \{\mathcal{A}[\![e]\!]s \mid s \in T\}$$

$$\mathcal{B}_c : \mathbf{Bexp} \to \wp(\mathbf{State}) \to \wp(\mathbf{State})$$

$$\mathcal{B}_c[\![b]\!]T \triangleq \{s \in T \mid \mathcal{B}[\![b]\!]s = tt\}$$

$\mathcal{B}_c[\![e]\!]$ transforms a state property into a state property

Thus, $\mathcal{B}_c[\![b]\!]T$ selects those states in T which make the guard b true

Collecting denotational semantics

$\mathcal{D} : \mathbf{While} \to \wp(\mathbf{State}) \to \wp(\mathbf{State})$

Let $T \in \wp(\mathbf{State})$

- $\mathcal{D}[\![x := e]\!]T \triangleq \{s[x \mapsto \mathcal{A}[\![e]\!]s] \mid s \in T\}$
- $\mathcal{D}[\![\mathbf{skip}]\!]T \triangleq T$
- $\mathcal{D}[\![S_1; S_2]\!]T \triangleq (\mathcal{D}[\![S_2]\!] \circ \mathcal{D}[\![S_1]\!])T$
- $\mathcal{D}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]T \triangleq (\mathcal{D}[\![S_1]\!] \circ \mathcal{B}_c[\![b]\!])T \ \cup \ (\mathcal{D}[\![S_2]\!] \circ \mathcal{B}_c[\![\neg b]\!])T$

Collecting denotational semantics

$\mathcal{D} : \mathbf{While} \to \wp(\mathbf{State}) \to \wp(\mathbf{State})$

$(\wp(\mathbf{State}), \subseteq)$ is a complete lattice. In order to define the collecting semantics of $\mathbf{while}$ loop, we consider a least fixpoint of a continuous function on this complete lattice. Let $T \in \wp(\mathbf{State})$.

– $\mathcal{D}[\![\mathbf{while}\, b\, \mathbf{do}\, S]\!]T \triangleq \mathcal{B}_c[\![\neg b]\!]\big(\mathrm{lfp}(\lambda T'.T \cup (\mathcal{D}[\![S]\!] \circ \mathcal{B}_c[\![b]\!])T')\big)$

Observe that $\mathrm{lfp}(\lambda T'.T \cup (\mathcal{D}[\![S]\!] \circ \mathcal{B}_c[\![b]\!])T')$ is the strongest loop invariant of the $\mathbf{while}\, b\, \mathbf{do}\, S$ loop for the precondition $T \in \wp(\mathbf{State})$.

Fixpoint: $X = T \cup \mathcal{D}[\![S]\!](\mathcal{B}_c[\![b]\!]X) \quad$ iff $\quad T \subseteq X \;\&\; \mathcal{D}[\![S]\!](\mathcal{B}_c[\![b]\!]X) \subseteq X$

Collecting denotational semantics: example

$$\mathcal{D}[\![\textbf{while}\ b\ \textbf{do}\ S]\!]T \triangleq \mathcal{B}_c[\![\neg b]\!]\big(\mathrm{lfp}(\lambda T'.T \cup (\mathcal{D}[\![S]\!] \circ \mathcal{B}_c[\![b]\!])T')\big)$$

$$T = \{\{x \mapsto 5\}, \{x \mapsto 6\}\} \in \wp(\textbf{State})$$

$$\mathcal{D}[\![\textbf{while}\ \neg(x = 0)\ \textbf{do}\ x := x - 2]\!]T = \ ?$$

$$F \triangleq \lambda T'.T \cup \{s[x \mapsto sx - 2] \mid sx \neq 0,\ s \in T'\}$$

$$\mathrm{lfp}(F) = \{..., \{x \mapsto -5\}, \{x \mapsto -3\}, \{x \mapsto -1\}, \{x \mapsto 0\}, \{x \mapsto 1\},$$
$$\{x \mapsto 2\}, \{x \mapsto 3\}, \{x \mapsto 4\}, \{x \mapsto 5\}, \{x \mapsto 6\}\}$$

$$\mathcal{B}_c[\![\neg\neg(x = 0)]\!](\mathrm{lfp}(F)) = \{\{x \mapsto 0\}\}$$

Collecting denotational semantics

Theorem.

For any $S \in \mathbf{While}$, $T \in \wp(\mathbf{State})$,
$\mathcal{D}[\![S]\!]T = \{\mathcal{S}_{\mathrm{ds}}[\![S]\!]s \mid s \in T, \mathcal{S}_{\mathrm{ds}}[\![S]\!]s \neq \mathit{undef}\}$.

Thus, $\mathcal{D}[\![S]\!]T = \varnothing \Leftrightarrow$ for any $s \in T$, the evaluation of S from s does not terminate

Abstract domains

We need:

(1) An abstract domain A which abstracts the concrete domain $\langle \wp(\mathbb{Z}), \subseteq \rangle$

(2) An abstract domain $\mathbb{S}$ which abstracts the concrete domain $\langle \wp(\mathbf{State}), \subseteq \rangle$

Thus, we consider GIs $(\alpha_A, \wp(\mathbb{Z}), A, \gamma_A)$ and $(\alpha_{\mathbb{S}}, \wp(\mathbf{State}), \mathbb{S}, \gamma_{\mathbb{S}})$

Abstract semantics of arithmetic expressions

Abstract semantics $\mathcal{A}^\sharp : \mathbf{Aexp} \to \mathbb{S} \to A$

$\mathcal{A}^\sharp$ must be correct: for any $e \in \mathbf{Aexp}$, for any $T \in \wp(\mathbf{State})$,

$$\alpha_A(\mathcal{A}_c[\![e]\!]T) \leq_A \mathcal{A}^\sharp[\![e]\!]\alpha_{\mathbb{S}}(T)$$

Equivalently: for any $e \in \mathbf{Aexp}$, for any $s^\sharp \in \mathbb{S}$,

$$\mathcal{A}_c[\![e]\!]\gamma_{\mathbb{S}}(s^\sharp) \subseteq \gamma_A(\mathcal{A}^\sharp[\![e]\!]s^\sharp)$$

Abstract semantics of Boolean expressions

Abstract semantics $\mathcal{B}^\sharp : \mathbf{Bexp} \to \mathbb{S} \to \mathbb{S}$

$\mathcal{B}^\sharp$ must be correct: for any $b \in \mathbf{Bexp}$, for any $T \in \wp(\mathbf{State})$,

$$\alpha_{\mathbb{S}}(\mathcal{B}_c[\![b]\!]T) \leq_{\mathbb{S}} \mathcal{B}^\sharp[\![b]\!]\alpha_{\mathbb{S}}(T)$$

Equivalently: for any $b \in \mathbf{Bexp}$, for any $s^\sharp \in \mathbb{S}$,

$$\mathcal{B}_c[\![b]\!]\gamma_{\mathbb{S}}(s^\sharp) \subseteq \gamma_{\mathbb{S}}(\mathcal{B}^\sharp[\![b]\!]s^\sharp)$$

Abstract State Update

We need an abstract version of the concrete state update $s[x \mapsto z]$

This is an abstract operation which given an abstract state $s^\sharp \in \mathbb{S}$, a variable $x \in \mathbf{Var}$, an abstract value $a \in A$, provides an updated abstract state $s^\sharp[x \mapsto a]$.

This abstract update must be correct: for any $s \in \gamma_{\mathbb{S}}(s^\sharp)$, $x \in \mathbf{Var}$, $z \in \gamma_A(a)$,

$$s[x \mapsto z] \in \gamma_{\mathbb{S}}\big(s^\sharp[x \mapsto a]\big)$$

Abstract semantics

Abstract semantics $\mathcal{D}^\sharp$ of $\mathbf{While}$ programs

$\mathcal{D}^\sharp : \mathbf{While} \to \mathbb{S} \to \mathbb{S}$

- $\mathcal{D}^\sharp[\![x := e]\!]s^\sharp \triangleq s^\sharp[x \mapsto \mathcal{A}^\sharp[\![e]\!]s^\sharp]$
- $\mathcal{D}^\sharp[\![\mathbf{skip}]\!]s^\sharp \triangleq s^\sharp$
- $\mathcal{D}^\sharp[\![S_1; S_2]\!]s^\sharp \triangleq (\mathcal{D}^\sharp[\![S_2]\!] \circ \mathcal{D}^\sharp[\![S_1]\!])s^\sharp$
- $\mathcal{D}^\sharp[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]s^\sharp \triangleq (\mathcal{D}^\sharp[\![S_1]\!] \circ \mathcal{B}^\sharp[\![b]\!])s^\sharp\ \vee_{\mathbb{S}}\ (\mathcal{D}^\sharp[\![S_2]\!] \circ \mathcal{B}^\sharp[\![\neg b]\!])s^\sharp$
- $\mathcal{D}^\sharp[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]s^\sharp \triangleq \mathcal{B}^\sharp[\![\neg b]\!]\big(\mathrm{lfp}(\lambda x.s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![S]\!] \circ \mathcal{B}^\sharp[\![b]\!])x)\big)$

Soundness of Abstract semantics

Abstract semantics $\mathcal{D}^\sharp : \mathbf{While} \to \mathbb{S} \to \mathbb{S}$

Soundness Theorem: for any $S \in \mathbf{While}$, $T \in \wp(\mathbf{State})$,

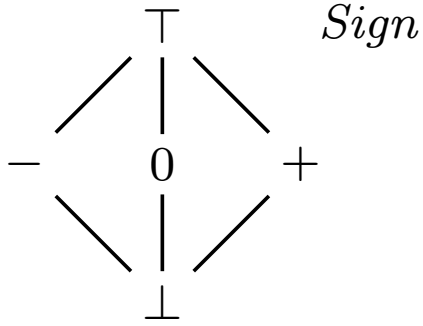
$$\alpha_{\mathbb{S}}(\mathcal{D}[\![S]\!]T) \leq_{\mathbb{S}} \mathcal{D}^\sharp[\![S]\!]\alpha_{\mathbb{S}}(T)$$

Equivalently: for any $S \in \mathbf{While}$, $s^\sharp \in \mathbb{S}$,

$$\mathcal{D}[\![S]\!]\gamma_{\mathbb{S}}(s^\sharp) \subseteq \gamma_{\mathbb{S}}(\mathcal{D}^\sharp[\![S]\!]s^\sharp)$$

Example

The abstract domain $Sign$ encodes sign properties of integer program variables



$(\alpha, \wp(\mathbb{Z}), Sign, \gamma)$ GI

$\alpha(\{-1,-3\}) = -;\ \ \alpha(\{0,3\}) = \top;\ \ \alpha(\{z \in \mathbb{Z} \mid z \geq 3\}) = +.$

$\gamma(\bot) = \varnothing;\ \ \gamma(-) = \{z \in \mathbb{Z} \mid z < 0\};\ \ \gamma(\top) = \mathbb{Z}.$

Example

The corresponding domain $\mathbb{S} \triangleq Var \to Sign$ abstracts $\wp(\mathbf{State})$

$(\mathbb{S}, \sqsubseteq, \bot_{\mathbb{S}}, \top_{\mathbb{S}})$ is a finite lattice (for a finite set Var)

$s_1^\sharp \sqsubseteq s_2^\sharp$ iff $\forall x \in Var. s_1^\sharp(x) \leq_{Sign} s_2^\sharp(x)$

$\bot_{\mathbb{S}} = \lambda x. \bot; \quad \top_{\mathbb{S}} = \lambda x. \top$

$(\alpha_{\mathbb{S}}, \wp(\mathbf{State}), \mathbb{S}, \gamma_{\mathbb{S}})$ is a GC where:

Remark: it is not a GI

$\alpha_{\mathbb{S}}(T) = \lambda x \in Var.\ \alpha(\{s(x) \mid s \in T\})$

$\gamma_{\mathbb{S}}(s^\sharp) = \{s \in \mathbf{State} \mid \forall x \in Var. \alpha(\{s(x)\}) \leq_{Sign} s^\sharp(x)\}$

$\gamma_{\mathbb{S}}$ is not injective, thus it is not a GI:

$\gamma_{\mathbb{S}}(\{x \mapsto -, y \mapsto \bot\}) = \varnothing = \gamma_{\mathbb{S}}(\{x \mapsto \bot, y \mapsto +\})$

Example

Abstract semantics of expressions $\mathcal{A}^\sharp : \mathbf{Aexp} \to \mathbb{S} \to Sign$

- $\mathcal{A}^\sharp[\![n]\!]s^\sharp \triangleq \alpha(\{n\})$
- $\mathcal{A}^\sharp[\![x]\!]s^\sharp \triangleq s^\sharp(x)$
- $\mathcal{A}^\sharp[\![e_1\ op\ e_2]\!]s^\sharp \triangleq \mathcal{A}^\sharp[\![e_1]\!]s^\sharp\ \ op^{Sign}\ \ \mathcal{A}^\sharp[\![e_2]\!]s^\sharp$
  where $op^{Sign} : Sign \times Sign \to Sign$ is a correct abstract operation for op

Examples:

$$\mathcal{A}^\sharp[\![2 * y + 3 * x]\!]\{x \mapsto -, y \mapsto -\} = -$$
$$\mathcal{A}^\sharp[\![x * y - 2 * z]\!]\{x \mapsto -, y \mapsto -, z \mapsto +\} = \top$$

Example

Abstract semantics of Boolean expressions $\mathcal{B}^\sharp : \mathbf{Bexp} \to \mathbb{S} \to \mathbb{S}$

$\mathcal{B}^\sharp[\![tt]\!]s^\sharp \triangleq s^\sharp$

$\mathcal{B}^\sharp[\![ff]\!]s^\sharp \triangleq \bot_{\mathbb{S}}$

$$\mathcal{B}^\sharp[\![e_1 = e_2]\!]s^\sharp \triangleq \begin{cases} \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_i]\!]s^\sharp \neq \top, \bot, \mathcal{A}^\sharp[\![e_1]\!]s^\sharp \neq \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_1]\!]s^\sharp = \bot \text{ or } \mathcal{A}^\sharp[\![e_2]\!]s^\sharp = \bot \\ s^\sharp & \text{otherwise} \end{cases}$$

This is not the best approximation. For example:

$\mathcal{B}^\sharp[\![x = 0]\!]\{x \mapsto \top\} \triangleq \{x \mapsto 0\}$

is (the best) correct approximation

Example

Abstract semantics of Boolean expressions $\mathcal{B}^\sharp : \mathbf{Bexp} \to \mathbb{S} \to \mathbb{S}$

$$\mathcal{B}^\sharp[\![tt]\!]s^\sharp \triangleq s^\sharp$$

$$\mathcal{B}^\sharp[\![ff]\!]s^\sharp \triangleq \bot_{\mathbb{S}}$$

$$\mathcal{B}^\sharp[\![e_1 = e_2]\!]s^\sharp \triangleq \begin{cases} \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_i]\!]s^\sharp \neq \top, \bot, \mathcal{A}^\sharp[\![e_1]\!]s^\sharp \neq \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_1]\!]s^\sharp = \bot \text{ or } \mathcal{A}^\sharp[\![e_2]\!]s^\sharp = \bot \\ s^\sharp & \text{otherwise} \end{cases}$$

$$\mathcal{B}^\sharp[\![e_1 \neq e_2]\!]s^\sharp \triangleq \begin{cases} \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_1]\!]s^\sharp = 0 = \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_1]\!]s^\sharp = \bot \text{ or } \mathcal{A}^\sharp[\![e_2]\!]s^\sharp = \bot \\ s^\sharp & \text{otherwise} \end{cases}$$

Example

$$\mathcal{B}^\sharp[\![e_1 < e_2]\!]s^\sharp \triangleq \begin{cases} \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_i]\!]s^\sharp \neq \top,\ \mathcal{A}^\sharp[\![e_1]\!]s^\sharp \text{ ``}>\text{''} \ \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_1]\!]s^\sharp = \bot = \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ s^\sharp & \text{otherwise} \end{cases}$$

This is not the best approximation. For example:

$\mathcal{B}^\sharp[\![x < 0]\!]\{x \mapsto \top\} \triangleq \{x \mapsto -\}$

is (the best) correct approximation

Exercise: Give the table defining the b.c.a. for $x < y$ on $Sign$

Example

$$\mathcal{B}^\sharp[\![e_1 < e_2]\!]s^\sharp \triangleq \begin{cases} \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_i]\!]s^\sharp \neq \top,\ \mathcal{A}^\sharp[\![e_1]\!]s^\sharp \text{ ``}>\text{''} \ \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_1]\!]s^\sharp = \bot = \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ s^\sharp & \text{otherwise} \end{cases}$$

$$\mathcal{B}^\sharp[\![e_1 \leq e_2]\!]s^\sharp \triangleq \begin{cases} \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_i]\!]s^\sharp \neq \top,\ \mathcal{A}^\sharp[\![e_1]\!]s^\sharp \text{ ``}>\text{''} \ \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ \bot_{\mathbb{S}} & \text{if } \mathcal{A}^\sharp[\![e_1]\!]s^\sharp = \bot = \mathcal{A}^\sharp[\![e_2]\!]s^\sharp \\ s^\sharp & \text{otherwise} \end{cases}$$

$\mathcal{B}^\sharp[\![b_1 \wedge b_2]\!]s^\sharp \triangleq (\mathcal{B}^\sharp[\![b_2]\!] \circ \mathcal{B}^\sharp[\![b_1]\!])s^\sharp$ $\quad$ $\mathcal{B}^\sharp[\![b_1 \vee b_2]\!]s^\sharp \triangleq \mathcal{B}^\sharp[\![b_1]\!]s^\sharp \vee_{\mathbb{S}} \mathcal{B}^\sharp[\![b_2]\!]s^\sharp$

Examples:

$\mathcal{B}^\sharp[\![x < -1]\!]\{x \mapsto +\} = \bot_{\mathbb{S}}$

$\mathcal{B}^\sharp[\![x \neq 2 * y]\!]\{x \mapsto +, y \mapsto +\} = \{x \mapsto +, y \mapsto +\}$

This definition of $\mathcal{B}^\sharp$ can be further improved, for example:

$\mathcal{B}^\sharp[\![x < y]\!]\{x \mapsto \top, y \mapsto 0\}$ could be set to $\{x \mapsto -, y \mapsto 0\}$

Example

Abstract update $\cdot[\cdot \mapsto \cdot] : \mathbb{S} \times \mathbf{Var} \times Sign \to \mathbb{S}$

$$s^\sharp[x \mapsto a] \triangleq \lambda y. \begin{cases} a & \text{if } y = x \\ s^\sharp(y) & \text{if } y \neq x \end{cases}$$

Example

$\mathcal{D}^\sharp[\![\mathbf{while}\ x \neq 0\ \mathbf{do}\ x := x+1]\!]s^\sharp \triangleq$

$\mathcal{B}^\sharp[\![x = 0]\!]\big(\mathrm{lfp}(\lambda t^\sharp.s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![x := x+1]\!] \circ \mathcal{B}^\sharp[\![x \neq 0]\!])t^\sharp)\big)$

- $s_1^\sharp = \{x \mapsto -\}$
  - $\mathcal{D}^\sharp[\![x := x+1]\!] \circ \mathcal{B}^\sharp[\![x \neq 0]\!]s_1^\sharp = \mathcal{D}^\sharp[\![x := x+1]\!]s_1^\sharp = \{x \mapsto \top\}$
  - $s_1^\sharp \vee_{\mathbb{S}} \{x \mapsto \top\} = \{x \mapsto \top\}$
  - $\mathrm{lfp}\big(\lambda t^\sharp.s_1^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![x := x+1]\!] \circ \mathcal{B}^\sharp[\![x \neq 0]\!])t^\sharp\big) = \{x \mapsto \top\}$
  - $\mathcal{B}^\sharp[\![x = 0]\!]\{x \mapsto \top\} = \{x \mapsto 0\}$

⇒No interesting information

Example

- $s_2^\sharp = \{x \mapsto 0\}$
  - $\mathcal{D}^\sharp[\![x := x+1]\!] \circ \mathcal{B}^\sharp[\![x \neq 0]\!] s_2^\sharp = \mathcal{D}^\sharp[\![x := x+1]\!]\{x \mapsto \bot\} = \{x \mapsto \bot\}$
  - $s_2^\sharp \vee_{\mathbb{S}} \{x \mapsto \bot\} = \{x \mapsto 0\}$
  - $\mathrm{lfp}\big(\lambda t^\sharp . s_2^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![x := x+1]\!] \circ \mathcal{B}^\sharp[\![x \neq 0]\!]) t^\sharp\big) = \{x \mapsto 0\}$
  - $\mathcal{B}^\sharp[\![x = 0]\!]\{x \mapsto 0\} = \{x \mapsto 0\}$

⇒No interesting information

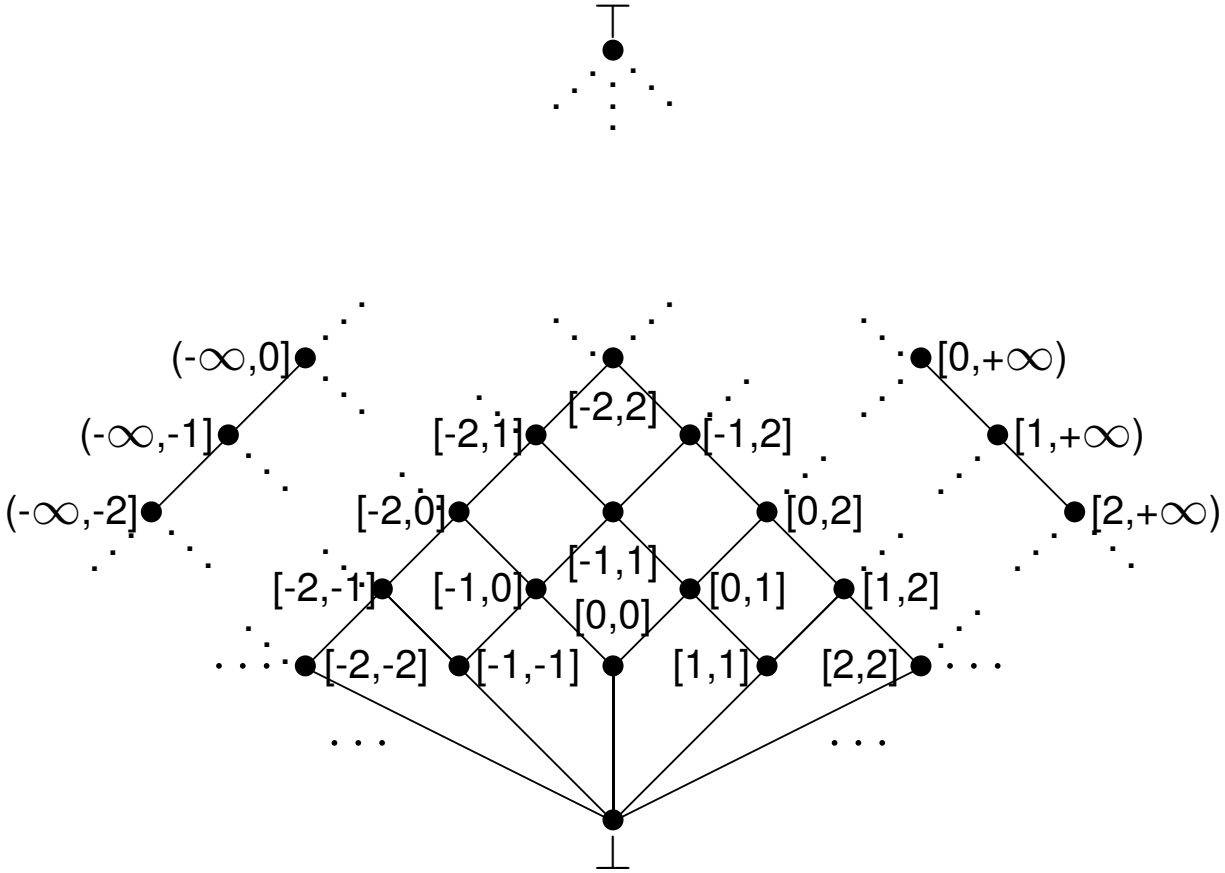
Example

- $s_3^\sharp = \{x \mapsto +\}$
  - $\mathcal{D}^\sharp[\![x := x + 1]\!] \circ \mathcal{B}^\sharp[\![x \neq 0]\!] s_3^\sharp = \mathcal{D}^\sharp[\![x := x + 1]\!]\{x \mapsto +\} = \{x \mapsto +\}$
  - $s_3^\sharp \vee_{\mathbb{S}} \{x \mapsto +\} = \{x \mapsto +\}$
  - $\mathrm{lfp}\big(\lambda t^\sharp . s_3^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![x := x + 1]\!] \circ \mathcal{B}^\sharp[\![x \neq 0]\!]) t^\sharp\big) = \{x \mapsto +\}$
  - $\mathcal{B}^\sharp[\![x = 0]\!]\{x \mapsto +\} = \{x \mapsto \bot\}$

$\Rightarrow$ If the initial value of x is positive then the program does not terminate

Example

Abstract domain of integer intervals: Int



Example

Abstract domain Int

$(\alpha, \wp(\mathbb{Z}), Int, \gamma)$ GI

α(*X*) is the least interval containing *X*

$\alpha(\{-1, 3\}) = [-1, 3]$

$\alpha(\{z \in \mathbb{Z} \mid z \text{ is even}\}) = \top$

$\alpha(\{z \in \mathbb{Z} \mid z \geq 3, z \neq 9\}) = [3, +\infty)$.

Abstract domain of abstract states: $\mathbb{S} \triangleq Var \to Int$

In this example, we do not give explicit abstract operations on Int

Example

Program to analyze:

$$\mathbf{while}\ x \geq 0\ \mathbf{do}\ \{x := x - 1; y := y + 1\}$$

Abstract semantics:

$$\mathcal{D}^\sharp[\![\mathbf{while}\ x \geq 0\ \mathbf{do}\ \{x := x - 1; y := y + 1\}]\!]s^\sharp \triangleq$$

$$\mathcal{B}^\sharp[\![x < 0]\!]\big(\mathrm{lfp}(\lambda t^\sharp.s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!])t^\sharp)\big)$$

$$\textbf{while}\ x \geq 0\ \textbf{do}\ \{x := x - 1; y := y + 1\}$$

Example

Initial abstract state:

$$s^\sharp = \{x \mapsto [10, 10], y \mapsto [0, 0]\}$$

Let us compute:

$$\mathrm{lfp}(\lambda t^\sharp . s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!])t^\sharp))$$

* $\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!]s^\sharp =$
  $\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!]\{x \mapsto [10, 10], y \mapsto [0, 0]\} =$
  $\mathcal{D}^\sharp[\![y := y + 1]\!]\{x \mapsto [9, 9], y \mapsto [0, 0]\} = \{x \mapsto [9, 9], y \mapsto [1, 1]\}$

* $s^\sharp \vee_{\mathbb{S}} \{x \mapsto [9, 9], y \mapsto [1, 1]\} = \{x \mapsto [9, 10], y \mapsto [0, 1]\}$

because $[9, 9] \vee_{\mathsf{Int}} [10, 10] = [9, 10]$ and $[0, 0] \vee_{\mathsf{Int}} [1, 1] = [0, 1]$

$$\textbf{while } x \geq 0 \textbf{ do } \{x := x - 1; y := y + 1\}$$

Example

* $\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!]\{x \mapsto [9, 10], y \mapsto [0, 1]\} =$
  $\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!]\{x \mapsto [9, 10], y \mapsto [0, 1]\} =$
  $\mathcal{D}^\sharp[\![y := y + 1]\!]\{x \mapsto [8, 9], y \mapsto [0, 1]\} = \{x \mapsto [8, 9], y \mapsto [1, 2]\}$
* $\{x \mapsto [9, 10], y \mapsto [0, 1]\} \vee_{\mathbb{S}} \{x \mapsto [8, 9], y \mapsto [1, 2]\} = \{x \mapsto [8, 10], y \mapsto [0, 2]\}$
* ...

Remark: this is an infinite chain

* $\mathrm{lfp}(\lambda t^\sharp . s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!])t^\sharp) =$
  $\{x \mapsto [-1, 10], y \mapsto [0, +\infty)\}$

$$\textbf{while}\ x \geq 0\ \textbf{do}\ \{x := x - 1; y := y + 1\}$$

Example

$$\mathrm{lfp}(\lambda t^\sharp . s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!])t^\sharp) =$$
$$\{x \mapsto [-1, 10], y \mapsto [0, +\infty)\}$$

Thus:

$$\mathcal{B}^\sharp[\![x < 0]\!]\{x \mapsto [-1, 10], y \mapsto [0, +\infty)\} = \{x \mapsto [-1, -1], y \mapsto [0, +\infty)\}$$

Problem

The Kleene iteration sequence for:

$$\mathrm{lfp}(\lambda t^\sharp . s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!])t^\sharp) =$$
$$\{x \mapsto [-1, 10], y \mapsto [0, +\infty)\}$$

is infinite. This analysis is not terminating.

We need a technique to extrapolate in a finite number of steps that $y \mapsto [0, +\infty)$

$$s^\sharp = \{x \mapsto [10, 10], y \mapsto [0, 0]\}$$
$$\mathrm{lfp}(\lambda t^\sharp . s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!])t^\sharp))$$

* $\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!]s^\sharp =$
  $\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!]\{x \mapsto [10, 10], y \mapsto [0, 0]\} =$
  $\mathcal{D}^\sharp[\![y := y + 1]\!]\{x \mapsto [9, 9], y \mapsto [0, 0]\} = \{x \mapsto [9, 9], y \mapsto [1, 1]\}$
* $s^\sharp \vee_{\mathbb{S}} \{x \mapsto [9, 9], y \mapsto [1, 1]\} = \{x \mapsto [9, 10], y \mapsto [0, 1]\}$

Here is the problem for ***y***

Problem

The Kleene iteration sequence for:

$$\mathrm{lfp}(\lambda t^\sharp . s^\sharp \vee_{\mathbb{S}} (\mathcal{D}^\sharp[\![y := y + 1]\!] \circ \mathcal{D}^\sharp[\![x := x - 1]\!] \circ \mathcal{B}^\sharp[\![x \geq 0]\!])t^\sharp) =$$
$$\{x \mapsto [-1, 10], y \mapsto [0, +\infty)\}$$

Why is the analysis of Interproc able to terminate?

Analysis Result

Run interprocweb or interprocwebf ?

Result

```
Annotated program after forward analysis
var x : int, y : int;
begin
  /* (L2 C5) top */
  x = 10; /* (L3 C9) [|x-10=0|] */
  y = 0; /* (L4 C8)
            [|x+1>=0; -x+10>=0; y>=0|] */
  while x >= 0 do
    /* (L5 C17) [|x>=0; -x+10>=0; y>=0|] */
    x = x - 1; /* (L6 C12)
                  [|x+1>=0; -x+9>=0; y>=0|] */
    y = y + 1; /* (L7 C12)
                  [|x+1>=0; -x+9>=0; y-1>=0|] */
  done; /* (L8 C7) [|x+1=0; y>=0|] */
end
```

Source

```
var x:int, y:int;
begin
  x = 10;
  y = 0;
  while (x>=0) do
    x = x-1;
    y = y+1;
  done;
end
```

Widening in Abstract Interpretation

An operator $\nabla: L \times L \rightarrow L$ on a complete lattice $(L, \leq_L)$ is called a widening operator if:

- It is a **finite upper bound**, namely:
  for any $x,y \in L$, $x,y \leq_L x \nabla y$
- It **enforces convergence**, namely for any ascending chain $(x_n)_{n \geq 0}$ in L, the ascending chain $(y_n)_{n \geq 0}$ inductively defined by:

  $y_0 \triangleq x_0 \qquad y_{k+1} \triangleq y_k \nabla x_{k+1}$

  is not strictly increasing, i.e., the chain finitely converges

Example

- Let us consider the complete lattice of integer intervals **Int** where the lub is defined by: $[a,b] \sqcup [c,d] = [\min(a,c),\max(b,d)]$
- Let K be some given interval in **Int**
- Let us define the operator Ω_K: **Int**×**Int** $\rightarrow$ **Int** as follows:

$$\Omega_K([a,b],[c,d]) \triangleq \begin{cases} [\min(a,c),\max(b,d)] & \text{if } [\min(a,c),\max(b,d)] \subseteq K \\ [-\infty, +\infty] & \text{otherwise} \end{cases}$$

- If K is a finite interval then Ω_K is a widening

Example

$$\Omega_K([a,b],[c,d]) = \begin{cases} [\min(a,c),\max(b,d)] & \text{if } [\min(a,c),\max(b,d)] \subseteq K \\ [-\infty, +\infty] & \text{otherwise} \end{cases}$$

For $K=[0, +\infty]$ then $\Omega_{[0, +\infty]}$ is not a widening. The chain:

[0,0], [0,1], [0,2], [0,3], [0,4], [0,5], [0,6],…
becomes the chain
[0,0], [0,1], [0,2], [0,3], [0,4], [0,5], [0,6],…
hence it does not converge

Example

- In general, a widening is **neither commutative nor monotone**
- We consider a **standard widening** on the **Int** abstract domain
- We define ∇: **Int**$\times$**Int** $\rightarrow$ **Int** as follows:

$$\bot \nabla X \triangleq X$$

$$X \nabla \bot \triangleq X$$

$$[x_0,y_0] \nabla [x_1,y_1] \triangleq [\text{if } x_0 \leq x_1 \text{ then } x_0 \text{ else } -\infty,$$

$$\text{if } y_1 \leq y_0 \text{ then } y_0 \text{ else } +\infty]$$

- Thus, unstable left and right bounds are widened, respectively, to $-\infty$ and $+\infty$

Example

- In general, a widening is **neither commutative nor monotone**
- We consider a **standard widening** on the **Int** abstract domain
- We define ∇: **Int**$\times$**Int** $\rightarrow$ **Int** as follows:

$$\bot \nabla X \triangleq X$$

$$X \nabla \bot \triangleq X$$

$$[x_0,y_0] \nabla [x_1,y_1] \triangleq [\text{if } x_0 \leq x_1 \text{ then } x_0 \text{ else } -\infty,$$

$$\text{if } y_1 \leq y_0 \text{ then } y_0 \text{ else } +\infty]$$

- Thus, unstable left and right bounds are widened, respectively, to $-\infty$ and $+\infty$
- We have that $[0,1] \subseteq [0,2]$ but $[0,1] \nabla [0,2] = [0,+\infty] \not\subseteq [0,2] = [0,2] \nabla [0,2]$
- We have that $[0,2] \nabla [0,1] = [0,2]$ while $[0,1] \nabla [0,2] = [0,+\infty]$

Widening and fixpoints

Let f be a **continuous** function f: (L,≤) → (L,≤) on a complete lattice L and consider a widening ∇ on L.

We define the following ascending chain depending on ∇:

$$x^0 \triangleq \bot$$

$$x^{n+1} \triangleq \begin{cases} x^n & \text{if } f(x^n) \leq x^n \\ x^n \nabla f(x^n) & \text{if } f(x^n) \not\leq x^n \end{cases}$$

Lemma: $\{x^n\}_{n \in \mathbb{N}}$ is an ascending chain which converges in a **finite number of steps** to a limit z such that f(z)≤z (i.e., z is a pre-fixpoint) and **lfp(f)≤z holds**

Example

Let us consider the program:

```
int x = 1; while (x <= 100) do x := x+1;
```

The interval analysis on **Int** requires the lfp computation of the following function:

F^a : **Int** $\rightarrow$ **Int**

$F^a(X) \triangleq [1,1] \sqcup ((X \sqcap [-\infty,100]) \oplus [1,1])$

The ascending chain induced by the standard widening ∇on Int is as follows:

$X_0 = \bot$

$X_1 = X_0 \nabla F^a(X_0) = \bot \nabla ([1,1] \sqcup ((\bot \sqcap [-\infty,100]) \oplus [1,1])) = \bot \nabla [1,1] = [1,1]$

$X_2 = X_1 \nabla F^a(X_1) = [1,1] \nabla ([1,1] \sqcup (([1,1] \sqcap [-\infty,100]) \oplus [1,1])) = [1,1] \nabla ([1,1] \sqcup [2,2]) =$

$= [1,1] \nabla [1,2] = [1,+\infty]$

$X_3 = X_2$ because $F^a(X_2) = [1,1] \sqcup (([1,+\infty] \sqcap [-\infty,100]) \oplus [1,1]) = [1,101] \leq X_2$

Example

Let us consider the program:

```
int x = 1; while (x <= 100) do x := x+1;
```

The interval analysis on **Int** requires the lfp computation of the following function:

F^a : Int $\rightarrow$ Int

$F^a(X) \triangleq [1,1] \sqcup ((X \sqcap [-\infty,100]) \oplus [1,1])$

The interval widening ∇provides $[1,+\infty]$ in 3 iterations as approximation of lfp(F^a).

This allows us to conclude as post-condition that
$[101,+\infty] \sqcap [1,+\infty] = [101,+\infty]$ holds, i.e. that `x >100` holds

Without widening, in 101 iterations we would get the precise least fixpoint
lfp(F^a)=[1,101] and therefore the post-condition `x=101`

Abstract Interpretation with Control Flow Graphs

by Antoine Minè
(Sorbonne University Paris)

Syntax

Expression syntax

- fixed, finite set of variables $\mathbb{V}$,
- one datatype: scalars in $\mathbb{I}$, with $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$
  (and later, floating-point numbers $\mathbb{F}$)
- no procedure

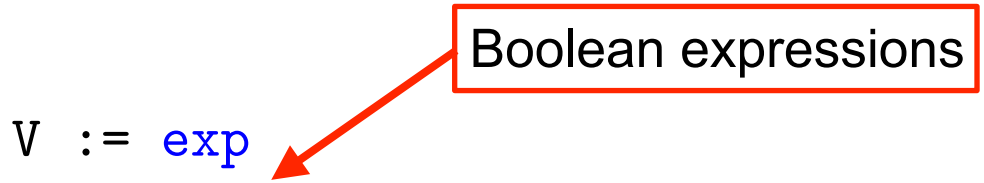
arithmetic expressions:

exp	::=	V	variable $\mathtt{V} \in \mathbb{V}$
	|	$-$exp	negation
	|	exp $\diamond$ exp	binary operation: $\diamond \in \{+, -, \times, /\}$
	|	$[c, c']$	constant range, $c, c' \in \mathbb{I} \cup \{\pm\infty\}$
			c is a shorthand for $[c, c]$

divisions included

Programs (structured syntax)

programs: as syntax trees



`prog` ::=		
|	`V := exp`	assignment
|	`if exp` $\bowtie$ `0 then prog else prog fi`	test
|	`while exp` $\bowtie$ `0 do prog done`	loop
|	`prog; prog`	sequence
|	ε	no-op

comparison operators: $\bowtie \in \{ =, <, >, <=, >=, <> \}$.

Programs (as control-flow graphs)

commands:

com	::=	V := exp	assignment into $\mathtt{V} \in \mathbb{V}$
	|	exp $\bowtie$ 0	test, $\bowtie \in \{=, <, >, <=, >=, <>\}$

programs: as control-flow graphs

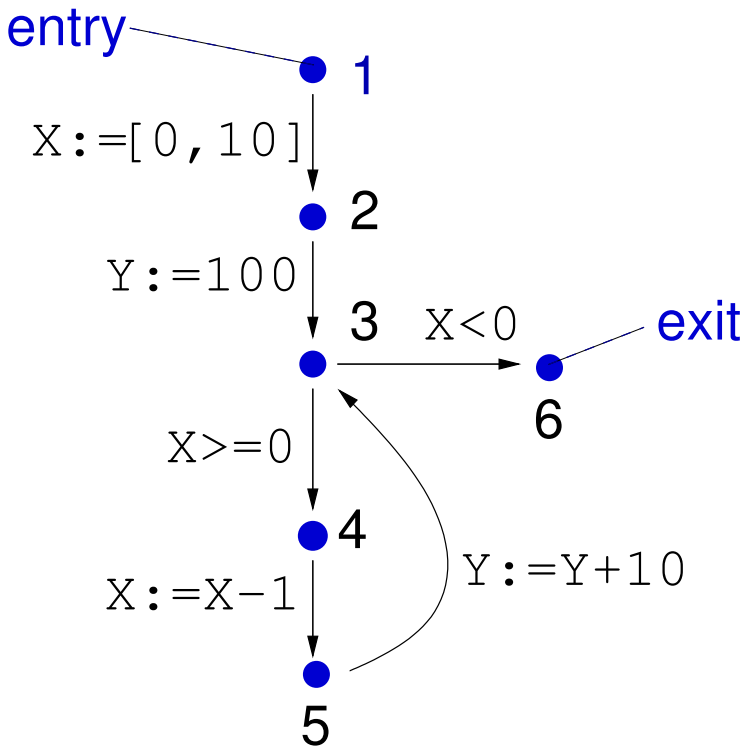
$$P \stackrel{\text{def}}{=} (L, e, x, A)$$

L	program points (labels)
e	entry point: $e \in L$
x	exit point: $x \in L$
A	arcs: $A \subseteq L \times \mathtt{com} \times L$

Example

```
1X:=[0,10];2
 Y:=100;
 while 3X>=0 do4
   X:=X-1;5
   Y:=Y+10
 done6
```

structured program



control flow graph

Concrete semantics

Forward concrete semantics

State → $\mathbb{V} \to \mathbb{I}$; Set of values → $\mathcal{P}(\mathbb{I})$

Semantics of expressions: $\mathbb{E}[\![e]\!] : (\mathbb{V} \to \mathbb{I}) \to \mathcal{P}(\mathbb{I})$

The evaluation of e in ρ gives a set of values:

because intervals $[x,y]$ are values

$$
\begin{array}{lcl}
\mathbb{E}[\![[c, c']]\!]\, \rho & \stackrel{\text{def}}{=} & \{ x \in \mathbb{I} \mid c \leq x \leq c' \} \\
\mathbb{E}[\![\mathtt{V}]\!]\, \rho & \stackrel{\text{def}}{=} & \{ \rho(\mathtt{V}) \} \\
\mathbb{E}[\![-e]\!]\, \rho & \stackrel{\text{def}}{=} & \{ -v \mid v \in \mathbb{E}[\![e]\!]\, \rho \} \\
\mathbb{E}[\![e_1 + e_2]\!]\, \rho & \stackrel{\text{def}}{=} & \{ v_1 + v_2 \mid v_1 \in \mathbb{E}[\![e_1]\!]\, \rho, v_2 \in \mathbb{E}[\![e_2]\!]\, \rho \} \\
\mathbb{E}[\![e_1 - e_2]\!]\, \rho & \stackrel{\text{def}}{=} & \{ v_1 - v_2 \mid v_1 \in \mathbb{E}[\![e_1]\!]\, \rho, v_2 \in \mathbb{E}[\![e_2]\!]\, \rho \} \\
\mathbb{E}[\![e_1 \times e_2]\!]\, \rho & \stackrel{\text{def}}{=} & \{ v_1 \times v_2 \mid v_1 \in \mathbb{E}[\![e_1]\!]\, \rho, v_2 \in \mathbb{E}[\![e_2]\!]\, \rho \} \\
\mathbb{E}[\![e_1 / e_2]\!]\, \rho & \stackrel{\text{def}}{=} & \{ v_1 / v_2 \mid v_1 \in \mathbb{E}[\![e_1]\!]\, \rho, v_2 \in \mathbb{E}[\![e_2]\!]\, \rho, \; v_2 \neq 0 \}
\end{array}
$$

$$\mathbb{E}[\![e_i]\!] = \varnothing \Rightarrow \mathbb{E}[\![e_1 \diamond e_2]\!] = \varnothing$$

Nontermination ~ run-time error

Forward concrete semantics (cont.)

Semantics of commands: $\mathsf{C}[\![\, c \,]\!] : \mathcal{P}(\mathbb{V} \to \mathbb{I}) \to \mathcal{P}(\mathbb{V} \to \mathbb{I})$

A transfer function for c defines a relation on environments:

$$\mathsf{C}[\![\, \mathtt{V} := e \,]\!]\, \mathcal{X} \overset{\text{def}}{=} \{\, \rho[\, \mathtt{V} \mapsto v \,] \mid \rho \in \mathcal{X},\ v \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\}$$

$$\mathsf{C}[\![\, e \bowtie 0 \,]\!]\, \mathcal{X} \overset{\text{def}}{=} \{\, \rho \mid \rho \in \mathcal{X},\ \exists v \in \mathsf{E}[\![\, e \,]\!]\, \rho,\ v \bowtie 0 \,\}$$

filtering of states making $e \bowtie 0$ true

It relates the environments after the execution of a command to the environments before.

Complete join morphism: $\mathsf{C}[\![\, c \,]\!]\, \mathcal{X} = \bigcup_{\rho \in \mathcal{X}} \mathsf{C}[\![\, c \,]\!]\, \{\, \rho \,\}$.

this is a collecting semantics

Forward concrete semantics (cont.)

Semantics of commands: $\mathsf{C}[\![c]\!] : \mathcal{P}(\mathbb{V} \to \mathbb{I}) \to \mathcal{P}(\mathbb{V} \to \mathbb{I})$

A transfer function for c defines a relation on environments:

$$\mathsf{C}[\![\mathtt{V} := e]\!]\,\mathcal{X} \overset{\text{def}}{=} \{ \rho[\, \mathtt{V} \mapsto v \,] \mid \rho \in \mathcal{X},\ v \in \mathsf{E}[\![e]\!]\,\rho \}$$
$$\mathsf{C}[\![e \bowtie 0]\!]\,\mathcal{X} \overset{\text{def}}{=} \{ \rho \mid \rho \in \mathcal{X},\ \exists v \in \mathsf{E}[\![e]\!]\,\rho,\ v \bowtie 0 \}$$

$$\mathsf{C}[\![x := 1/0]\!]\mathcal{X} = \varnothing$$
$$\mathsf{C}[\![1/0 < 1]\!]\mathcal{X} = \varnothing$$

$\varnothing$ models run-time error or non-termination

Forward concrete semantics (cont.)

Semantics of programs: $\mathrm{P}[\![(L, e, x, A)]\!] : L \to \mathcal{P}(\mathbb{V} \to \mathbb{I})$

$\mathrm{P}[\![(L, e, x, A)]\!]\, \ell$ is the most precise invariant at $\ell \in L$.

It is the smallest solution of a recursive equation system $(\mathcal{X}_\ell)_{\ell \in L}$:

Semantic equation system: one equation for program point

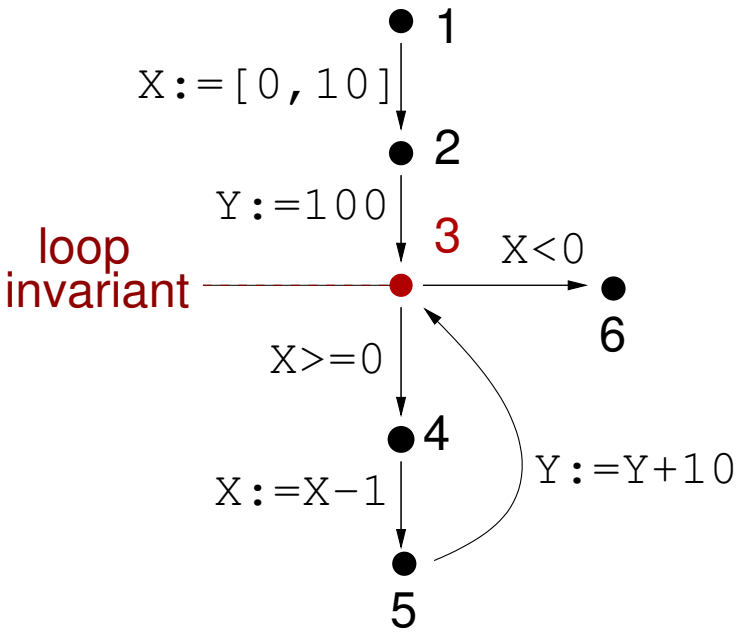
$$\begin{array}{lll} \mathcal{X}_e & & \text{(given initial state)} \\ \mathcal{X}_{\ell \neq e} & = \bigcup_{(\ell', c, \ell) \in A} \mathrm{C}[\![c]\!]\, \mathcal{X}_{\ell'} & \text{(transfer function)} \end{array}$$

Tarski's Theorem: this smallest solution exists and is unique.

- $\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{P}(\mathbb{V} \to \mathbb{I}), \subseteq, \cup, \cap, \emptyset, (\mathbb{V} \to \mathbb{I}))$ is a complete lattice,
- each $M_\ell : \mathcal{X}_\ell \mapsto \bigcup_{(\ell', c, \ell) \in A} \mathrm{C}[\![c]\!]\, \mathcal{X}_{\ell'}$ is monotonic in $\mathcal{D}$.

  $\Rightarrow$ the solution is the least fixpoint of $(M_\ell)_{\ell \in L}$.

Forward concrete semantics (example)



control flow graph

$$\begin{cases} \mathcal{X}_1 = (\{\, \mathtt{X}, \mathtt{Y} \,\} \to \mathbb{Z}) \\ \mathcal{X}_2 = \mathrm{C}[\![\, \mathtt{X} := [0, 10] \,]\!]\, \mathcal{X}_1 \\ \mathcal{X}_3 = \mathrm{C}[\![\, \mathtt{Y} := 100 \,]\!]\, \mathcal{X}_2 \cup \\ \qquad\quad \mathrm{C}[\![\, \mathtt{Y} := \mathtt{Y} + 10 \,]\!]\, \mathcal{X}_5 \\ \mathcal{X}_4 = \mathrm{C}[\![\, \mathtt{X} \geq 0 \,]\!]\, \mathcal{X}_3 \\ \mathcal{X}_5 = \mathrm{C}[\![\, \mathtt{X} := \mathtt{X} - 1 \,]\!]\, \mathcal{X}_4 \\ \mathcal{X}_6 = \mathrm{C}[\![\, \mathtt{X} < 0 \,]\!]\, \mathcal{X}_3 \end{cases}$$

equation system

Loop invariant:

$$\mathcal{X}_3 = \{\, \rho \mid \rho(\mathtt{X}) \in [\text{-}1,10],\ 10\rho(\mathtt{X}) + \rho(\mathtt{Y}) \in [100, 200] \cap 10\mathbb{Z} \,\}$$

Resolution

Resolution by increasing iterations: Kleene iterates

$$\begin{cases} \mathcal{X}_e^0 \stackrel{\text{def}}{=} \mathcal{X}_e \\ \mathcal{X}_{\ell\neq e}^0 \stackrel{\text{def}}{=} \emptyset \end{cases} \qquad \begin{cases} \mathcal{X}_e^{n+1} \stackrel{\text{def}}{=} \mathcal{X}_e \\ \mathcal{X}_{\ell\neq e}^{n+1} \stackrel{\text{def}}{=} \bigcup\limits_{(\ell',c,\ell)\in A} \mathsf{C}[\![\, c \,]\!]\, \mathcal{X}_{\ell'}^n \end{cases}$$

Converges in ω iterations to a least solution,
because each $\mathsf{C}[\![\, c \,]\!]$ is continuous in the CPO $\mathcal{D}$.
(Kleene-Knaster-Tarski theorem)

Resolution (example)

$$\begin{cases} & \text{iteration } 0 \\ \mathcal{X}_1 = \mathbb{Z}^2 & \mathbb{Z}^2 \\ \mathcal{X}_2 = \mathrm{C}[\![\, \mathtt{X} := [0,10] \,]\!]\, \mathcal{X}_1 & \emptyset \\ \mathcal{X}_3 = \mathrm{C}[\![\, \mathtt{Y} := 100 \,]\!]\, \mathcal{X}_2 \cup \mathrm{C}[\![\, \mathtt{Y} := \mathtt{Y} + 10 \,]\!]\, \mathcal{X}_5 & \emptyset \\ \mathcal{X}_4 = \mathrm{C}[\![\, \mathtt{X} \geq 0 \,]\!]\, \mathcal{X}_3 & \emptyset \\ \mathcal{X}_5 = \mathrm{C}[\![\, \mathtt{X} := \mathtt{X} - 1 \,]\!]\, \mathcal{X}_4 & \emptyset \\ \mathcal{X}_6 = \mathrm{C}[\![\, \mathtt{X} < 0 \,]\!]\, \mathcal{X}_3 & \emptyset \end{cases}$$

Resolution (example)

$$\begin{cases} & \text{iteration 1} \\ \mathcal{X}_1 = \mathbb{Z}^2 & \mathbb{Z}^2 \\ \mathcal{X}_2 = \mathrm{C}[\![\, \mathtt{X} := [0,10] \,]\!]\, \mathcal{X}_1 & [0,10] \times \mathbb{Z} \\ \mathcal{X}_3 = \mathrm{C}[\![\, \mathtt{Y} := 100 \,]\!]\, \mathcal{X}_2 \cup \mathrm{C}[\![\, \mathtt{Y} := \mathtt{Y} + 10 \,]\!]\, \mathcal{X}_5 & \emptyset \\ \mathcal{X}_4 = \mathrm{C}[\![\, \mathtt{X} \geq 0 \,]\!]\, \mathcal{X}_3 & \emptyset \\ \mathcal{X}_5 = \mathrm{C}[\![\, \mathtt{X} := \mathtt{X} - 1 \,]\!]\, \mathcal{X}_4 & \emptyset \\ \mathcal{X}_6 = \mathrm{C}[\![\, \mathtt{X} < 0 \,]\!]\, \mathcal{X}_3 & \emptyset \end{cases}$$

Resolution (example)

$$
\begin{cases}
 & & \text{iteration 2} \\
\mathcal{X}_1 = \mathbb{Z}^2 & & \mathbb{Z}^2 \\
\mathcal{X}_2 = \mathrm{C}[\![\, \mathtt{X} := [0,10] \,]\!]\, \mathcal{X}_1 & & [0,10] \times \mathbb{Z} \\
\mathcal{X}_3 = \mathrm{C}[\![\, \mathtt{Y} := 100 \,]\!]\, \mathcal{X}_2 \cup \mathrm{C}[\![\, \mathtt{Y} := \mathtt{Y} + 10 \,]\!]\, \mathcal{X}_5 & & \{\, (0,100), \ldots, (10,100) \,\} \\
\mathcal{X}_4 = \mathrm{C}[\![\, \mathtt{X} \geq 0 \,]\!]\, \mathcal{X}_3 & & \emptyset \\
\mathcal{X}_5 = \mathrm{C}[\![\, \mathtt{X} := \mathtt{X} - 1 \,]\!]\, \mathcal{X}_4 & & \emptyset \\
\mathcal{X}_6 = \mathrm{C}[\![\, \mathtt{X} < 0 \,]\!]\, \mathcal{X}_3 & & \emptyset
\end{cases}
$$

Resolution (example)

$$
\begin{cases}
 & & \text{iteration } \ldots \\
\mathcal{X}_1 = & \mathbb{Z}^2 & \mathbb{Z}^2 \\
\mathcal{X}_2 = & \mathrm{C}[\![\, \mathtt{X} := [0,10] \,]\!]\, \mathcal{X}_1 & [0,10] \times \mathbb{Z} \\
\mathcal{X}_3 = & \mathrm{C}[\![\, \mathtt{Y} := 100 \,]\!]\, \mathcal{X}_2 \cup & \{\, (0,100), \ldots, (10,100), \\
 & \mathrm{C}[\![\, \mathtt{Y} := \mathtt{Y} + 10 \,]\!]\, \mathcal{X}_5 & \quad (-1,110), \ldots, (9,110), \\
 & & \quad (-1,120), \ldots, (8,120), \ldots \,\} \\
\mathcal{X}_4 = & \mathrm{C}[\![\, \mathtt{X} \geq 0 \,]\!]\, \mathcal{X}_3 & \{\, (0,100), \ldots, (10,100), \\
 & & \quad (0,110), \ldots, (9,110), \\
 & & \quad (0,120), \ldots, (8,120), \ldots \,\} \\
\mathcal{X}_5 = & \mathrm{C}[\![\, \mathtt{X} := \mathtt{X} - 1 \,]\!]\, \mathcal{X}_4 & \{\, (-1,100), \ldots, (9,100), \\
 & & \quad (-1,110), \ldots, (8,110), \\
 & & \quad (-1,120), \ldots, (7,120), \ldots \,\} \\
\mathcal{X}_6 = & \mathrm{C}[\![\, \mathtt{X} < 0 \,]\!]\, \mathcal{X}_3 & \{\, (-1,110), (-1,120), \ldots \,\}
\end{cases}
$$

Limit to automation

We wish to perform automatic numerical invariant discovery.

Theoretical problems

- elements of $\mathcal{P}(\mathbb{V} \to \mathbb{I})$ are not computer representable
- transfer functions $\mathsf{C}[\![c]\!]$ are not computable
- lattice iterations in $\mathcal{P}(\mathbb{V} \to \mathbb{I})$ can be infinite

Finding the best invariant is an undecidable problem

Note:

Even when $\mathbb{I}$ is finite, a concrete analysis is not tractable:

- representing elements in $\mathcal{P}(\mathbb{V} \to \mathbb{I})$ in extension is expensive
- computing $\mathsf{C}[\![c]\!]$ explicitly is expensive
- the lattice $\mathcal{P}(\mathbb{V} \to \mathbb{I})$ has a large height ($\Rightarrow$ many iterations)

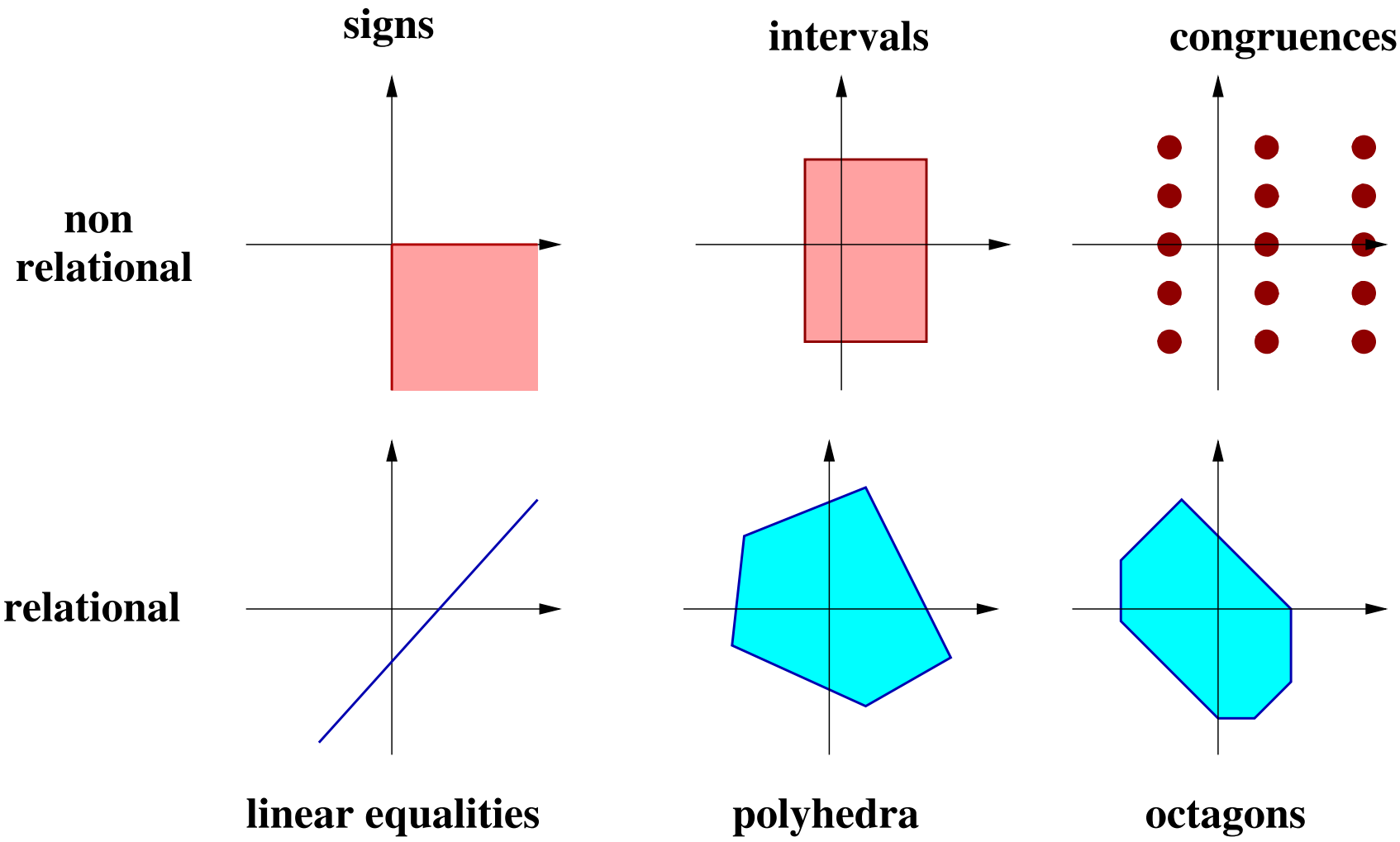
Abstraction

Numerical abstract domains

A numerical abstract domain is given by:

- a subset of $\mathcal{P}(\mathbb{V} \to \mathbb{I})$
  (a set of environment sets)
  together with a machine encoding,
- effective and sound abstract operators,
- an iteration strategy
  ensuring convergence in finite time.

Numerical abstract domain examples



Numerical abstract domains (cont.)

Representation: given by

minimal requirements

- a set $\mathcal{D}^\sharp$ of machine-representable abstract values,
- a partial order $(\mathcal{D}^\sharp, \sqsubseteq, \bot^\sharp, \top^\sharp)$
  relating the amount of information given by abstract values,
- a concretization function γ: $\mathcal{D}^\sharp \to \mathcal{P}(\mathbb{V} \to \mathbb{I})$
  giving a concrete meaning to each abstract element.

Required algebraic properties:

- γ should be monotonic for $\sqsubseteq$: $\mathcal{X}^\sharp \sqsubseteq \mathcal{Y}^\sharp \Longrightarrow \gamma(\mathcal{X}^\sharp) \subseteq \gamma(\mathcal{Y}^\sharp)$,
- $\gamma(\bot^\sharp) = \emptyset$, ☚ **unreachability**
- $\gamma(\top^\sharp) = \mathbb{V} \to \mathbb{I}$.

Note: γ need not be one-to-one.

Numerical abstract domains (cont.)

Abstract operators: we require:

- sound, effective, abstract transfer functions $C^\sharp[\![c]\!]$ for all commands c,
- sound, effective, abstract set operators $\cup^\sharp$, $\cap^\sharp$, ← not necessarily lub and glb
- an algorithm to decide the ordering $\sqsubseteq$.

Soundness criterion:

$F^\sharp$ is a sound abstraction of a $n-$ary operator F if:

$$\forall \mathcal{X}_1^\sharp, \ldots, \mathcal{X}_n^\sharp \in D^\sharp,\ F(\gamma(\mathcal{X}_1^\sharp), \ldots, \gamma(\mathcal{X}_n^\sharp)) \subseteq \gamma(F^\sharp(\mathcal{X}_1^\sharp, \ldots, \mathcal{X}_n^\sharp))$$

Abstract semantics

Abstract semantic equation system

$$\mathcal{X}^\sharp : L \to \mathcal{D}^\sharp$$

$$\mathcal{X}^\sharp_\ell \sqsupseteq \begin{cases} \mathcal{X}^\sharp_e & \text{if } \ell = e \quad \text{(where } \mathcal{X}_e \subseteq \gamma(\mathcal{X}^\sharp_e)\text{)} \\ \bigcup^\sharp_{(\ell',c,\ell)\in A} \mathsf{C}^\sharp[\![\, c \,]\!]\, \mathcal{X}^\sharp_{\ell'} & \text{if } \ell \neq e \quad \text{(abstract transfer function)} \end{cases}$$

$\sqsupseteq$ is not necessarily =, any approximate solution is accepted

Abstract semantics

Abstract semantic equation system

$$\mathcal{X}^\sharp : L \to \mathcal{D}^\sharp$$

$$\mathcal{X}^\sharp_\ell \supseteq \begin{cases} \mathcal{X}^\sharp_e & \text{if } \ell = e \quad \text{(where } \mathcal{X}_e \subseteq \gamma(\mathcal{X}^\sharp_e)\text{)} \\ \bigcup\limits^\sharp_{(\ell',c,\ell)\in A} \mathsf{C}^\sharp[\![\, c \,]\!]\, \mathcal{X}^\sharp_{\ell'} & \text{if } \ell \neq e \quad \text{(abstract transfer function)} \end{cases}$$

Soundness Theorem

Any solution $(\mathcal{X}^\sharp_\ell)_{\ell\in L}$ is a **sound over-approximation** of the concrete collecting semantics:

$$\forall \ell \in L,\ \gamma(\mathcal{X}^\sharp_\ell) \supseteq \mathcal{X}_\ell$$

where $\mathcal{X}_\ell$ is the smallest solution of

$$\begin{cases} \mathcal{X}_e & \text{given} \\ \mathcal{X}_\ell = \bigcup\limits_{(\ell',c,\ell)\in A} \mathsf{C}[\![\, c \,]\!]\, \mathcal{X}_{\ell'} & \text{if } \ell \neq e \end{cases}$$

Iteration strategy

Resolution by iterations in $\mathcal{D}^\sharp$:

To effectively solve the abstract system, we require:

- an iteration ordering on abstract equations
  (which equation(s) are applied at a given iteration)

Chaotic iteration: any ordering of the abstract equations provide the same result.
We do not consider iteration orderings (e.g. top-down vs bottom-up CFG traversal) and we apply in parallel all the abstract equations

Iteration strategy

Resolution by iterations in $\mathcal{D}^\sharp$:

To effectively solve the abstract system, we require:

- an iteration ordering on abstract equations
  (which equation(s) are applied at a given iteration)
- a widening operator ∇ to speed-up the convergence, if there are infinite strictly increasing chains in $\mathcal{D}^\sharp$

  $\nabla : (\mathcal{D}^\sharp \times \mathcal{D}^\sharp) \to \mathcal{D}^\sharp$ is a widening if:
  - it is sound: $\gamma(\mathcal{X}^\sharp) \cup \gamma(\mathcal{Y}^\sharp) \subseteq \gamma(\mathcal{X}^\sharp \nabla \mathcal{Y}^\sharp)$
  - it enforces termination:
    $\forall$ sequence $(\mathcal{Y}_i^\sharp)_{i \in \mathbb{N}}$
    the sequence $\mathcal{X}_0^\sharp = \mathcal{Y}_0^\sharp,\ \mathcal{X}_{i+1}^\sharp = \mathcal{X}_i^\sharp \nabla \mathcal{Y}_{i+1}^\sharp$
    stabilizes in finite time: $\exists n < \omega,\ \mathcal{X}_{n+1}^\sharp = \mathcal{X}_n^\sharp$

Abstract analysis

$\mathcal{W} \subseteq L$ is a set of widening points if every CFG cycle has a point in $\mathcal{W}$.

Forward analysis:

$\mathcal{X}^{\sharp 0}_e \stackrel{\text{def}}{=} \mathcal{X}^\sharp_e$ given, such that $\mathcal{X}_e \subseteq \gamma(\mathcal{X}^\sharp_e)$

$\mathcal{X}^{\sharp 0}_{\ell \neq e} \stackrel{\text{def}}{=} \bot^\sharp$

$$\mathcal{X}^{\sharp n+1}_\ell \stackrel{\text{def}}{=} \begin{cases} \mathcal{X}^\sharp_e & \text{if } \ell = e \\ \bigcup^\sharp_{(\ell',c,\ell)\in A} \mathsf{C}^\sharp[\![c]\!] \mathcal{X}^{\sharp n}_{\ell'} & \text{if } \ell \notin \mathcal{W},\ \ell \neq e \\ \mathcal{X}^{\sharp n}_\ell \mathbin{\triangledown} \bigcup^\sharp_{(\ell',c,\ell)\in A} \mathsf{C}^\sharp[\![c]\!] \mathcal{X}^{\sharp n}_{\ell'} & \text{if } \ell \in \mathcal{W},\ \ell \neq e \end{cases}$$

- termination: for some δ, $\forall \ell,\ \mathcal{X}^{\sharp \delta+1}_\ell = \mathcal{X}^{\sharp \delta}_\ell$
- soundness: $\forall \ell \in L,\ \mathcal{X}_\ell \subseteq \gamma(\mathcal{X}^{\sharp \delta}_\ell)$
- can be refined by decreasing iterations with narrowing $\vartriangle$ (presented later)

Exact and best abstractions: Reminders

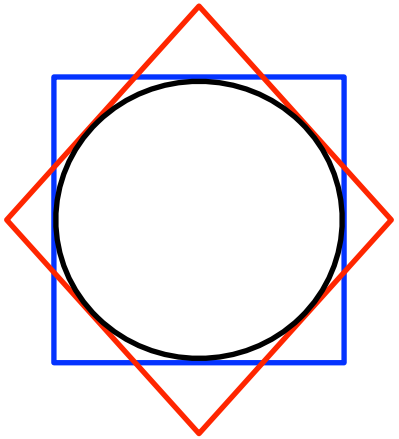
Galois connection: $(\mathcal{D}, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{D}^\sharp, \sqsubseteq)$

- α, γ monotonic and $\forall \mathcal{X}, \mathcal{Y}^\sharp$, $\alpha(\mathcal{X}) \sqsubseteq \mathcal{Y}^\sharp \iff \mathcal{X} \subseteq \gamma(\mathcal{Y}^\sharp)$
- $\Rightarrow$ elements $\mathcal{X}$ have a best abstraction: $\alpha(\mathcal{X})$
- $\Rightarrow$ operators F have a best abstraction: $F^\sharp = \alpha \circ F \circ \gamma$

Sometimes, no α exists:

- $\{ \gamma(\mathcal{Y}^\sharp) \mid \mathcal{X} \subseteq \gamma(\mathcal{Y}^\sharp) \}$ has no greatest lower bound

Counterexample
Convex polyhedra



Exact and best abstractions: Reminders

Galois connection: $(\mathcal{D}, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{D}^\sharp, \sqsubseteq)$

- α, γ monotonic and $\forall \mathcal{X}, \mathcal{Y}^\sharp$, $\alpha(\mathcal{X}) \sqsubseteq \mathcal{Y}^\sharp \iff \mathcal{X} \subseteq \gamma(\mathcal{Y}^\sharp)$
- $\Rightarrow$ elements $\mathcal{X}$ have a best abstraction: $\alpha(\mathcal{X})$
- $\Rightarrow$ operators F have a best abstraction: $F^\sharp = \alpha \circ F \circ \gamma$

Sometimes, no α exists:

- $\{ \gamma(\mathcal{Y}^\sharp) \mid \mathcal{X} \subseteq \gamma(\mathcal{Y}^\sharp) \}$ has no greatest lower bound

Concretization-based optimality:

- sound abstraction: $\gamma \circ F^\sharp \supseteq F \circ \gamma$
- exact abstraction: $\gamma \circ F^\sharp = F \circ \gamma$

α-based optimality

$F^\sharp \circ \alpha = \alpha \circ F$

Non-relational domains

No relational information
between different variables

Value abstract domain

Idea: start from an abstraction of values $\mathcal{P}(\mathbb{I})$

Numerical value abstract domain:

$\mathcal{B}^\sharp$	abstract values, machine-representable
$\gamma_b \colon \mathcal{B}^\sharp \to \mathcal{P}(\mathbb{I})$	concretization
$\sqsubseteq_b$	partial order
$\bot_b^\sharp$, $\top_b^\sharp$	represent $\emptyset$ and $\mathbb{I}$
$\cup_b^\sharp$, $\cap_b^\sharp$	sound abstractions of $\cup$ and $\cap$
∇_b	widening operator
$\alpha_b \colon \mathcal{P}(\mathbb{I}) \to \mathcal{B}^\sharp$	abstraction (optional)

Derived abstract domain

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} (\mathbb{V} \to (\mathcal{B}^\sharp \setminus \{ \bot_b^\sharp \})) \cup \{ \bot^\sharp \}$$

- point-wise extension: $\mathcal{X}^\sharp \in \mathcal{D}^\sharp$ is a vector of elements in $\mathcal{B}^\sharp$
  (e.g. using arrays of size $|\mathbb{V}|$)
- smashed $\bot^\sharp$ (avoids redundant representations of $\emptyset$)

Derived abstract domain (cont.)

$$\mathcal{X}^\sharp \sqsubseteq \mathcal{Y}^\sharp \overset{\text{def}}{\iff} \mathcal{X}^\sharp = \bot^\sharp \vee (\mathcal{X}^\sharp, \mathcal{Y}^\sharp \neq \bot^\sharp \wedge \forall \mathtt{v},\ \mathcal{X}^\sharp(\mathtt{v}) \sqsubseteq_b \mathcal{Y}^\sharp(\mathtt{v}))$$

$$\mathcal{X}^\sharp \cup^\sharp \mathcal{Y}^\sharp \overset{\text{def}}{=} \begin{cases} \mathcal{Y}^\sharp & \text{if } \mathcal{X}^\sharp = \bot^\sharp \\ \mathcal{X}^\sharp & \text{if } \mathcal{Y}^\sharp = \bot^\sharp \\ \lambda \mathtt{v}.\mathcal{X}^\sharp(\mathtt{v}) \cup_b^\sharp \mathcal{Y}^\sharp(\mathtt{v}) & \text{otherwise} \end{cases}$$

$$\mathcal{X}^\sharp \triangledown \mathcal{Y}^\sharp \overset{\text{def}}{=} \begin{cases} \mathcal{Y}^\sharp & \text{if } \mathcal{X}^\sharp = \bot^\sharp \\ \mathcal{X}^\sharp & \text{if } \mathcal{Y}^\sharp = \bot^\sharp \\ \lambda \mathtt{v}.\mathcal{X}^\sharp(\mathtt{v}) \triangledown_b \mathcal{Y}^\sharp(\mathtt{v}) & \text{otherwise} \end{cases}$$

$$\mathcal{X}^\sharp \cap^\sharp \mathcal{Y}^\sharp \overset{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } \mathcal{X}^\sharp = \bot^\sharp \text{ or } \mathcal{Y}^\sharp = \bot^\sharp \\ \bot^\sharp & \text{if } \exists \mathtt{v},\ \mathcal{X}^\sharp(\mathtt{v}) \cap_b^\sharp \mathcal{Y}^\sharp(\mathtt{v}) = \bot_b^\sharp \\ \lambda \mathtt{v}.\mathcal{X}^\sharp(\mathtt{v}) \cap_b^\sharp \mathcal{Y}^\sharp(\mathtt{v}) & \text{otherwise} \end{cases}$$

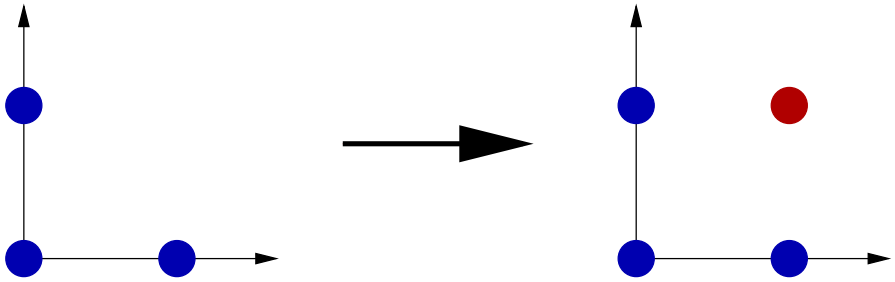
We will see later how to derive $\mathrm{C}^\sharp[\![\, c \,]\!]$ using:

- abstract operators $+_b^\sharp$, ... for $\mathrm{C}^\sharp[\![\, \mathtt{V} := e \,]\!]$

Cartesian abstraction

Non-relational domains "forget" all relationships between variables.

Example: $\rho_c(\{(X,Y) \mid X \in \{0,2\}, Y \in \{0,2\}, X + Y \leq 2\}) = \{0,2\} \times \{0,2\}$.



Generic non-relational abstract assignments

Given: sound abstract versions in $\mathcal{B}^\sharp$ of all arithmetic operators:

$$\begin{array}{rrcl}
[c,c']_b^\sharp: & \{x \mid c \leq x \leq c'\} & \subseteq & \gamma_b([c,c']_b^\sharp) \\
-_b^\sharp: & \{-x \mid x \in \gamma_b(\mathcal{X}_b^\sharp)\} & \subseteq & \gamma_b(-_b^\sharp\, \mathcal{X}_b^\sharp) \\
+_b^\sharp: & \{x+y \mid x \in \gamma_b(\mathcal{X}_b^\sharp), y \in \gamma_b(\mathcal{Y}_b^\sharp)\} & \subseteq & \gamma_b(\mathcal{X}_b^\sharp +_b^\sharp \mathcal{Y}_b^\sharp) \\
\vdots & & &
\end{array}$$

We can define:

- an abstract semantics of expressions: $\mathsf{E}^\sharp[\![e]\!] : \mathcal{D}^\sharp \to \mathcal{B}^\sharp$

$$\mathsf{E}^\sharp[\![e]\!]\, \bot^\sharp \stackrel{\text{def}}{=} \bot_b^\sharp$$

strictness

if $\mathcal{X}^\sharp \neq \bot^\sharp$:

$$\begin{array}{rcl}
\mathsf{E}^\sharp[\![[c,c']]\!]\, \mathcal{X}^\sharp & \stackrel{\text{def}}{=} & [c,c']_b^\sharp \\
\mathsf{E}^\sharp[\![\mathtt{V}]\!]\, \mathcal{X}^\sharp & \stackrel{\text{def}}{=} & \mathcal{X}^\sharp(\mathtt{V}) \\
\mathsf{E}^\sharp[\![-e]\!]\, \mathcal{X}^\sharp & \stackrel{\text{def}}{=} & -_b^\sharp\, \mathsf{E}^\sharp[\![e]\!]\, \mathcal{X}^\sharp \\
\mathsf{E}^\sharp[\![e_1+e_2]\!]\, \mathcal{X}^\sharp & \stackrel{\text{def}}{=} & \mathsf{E}^\sharp[\![e_1]\!]\, \mathcal{X}^\sharp +_b^\sharp \mathsf{E}^\sharp[\![e_2]\!]\, \mathcal{X}^\sharp \\
\vdots & &
\end{array}$$

compositional
by structural induction

Generic non-relational abstract assignments (cont.)

We can then define:

- an abstract assignment:

$$C^\sharp[\![\, \mathtt{V} := e \,]\!]\, \mathcal{X}^\sharp \overset{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } \mathcal{V}_b^\sharp = \bot_b^\sharp \\ \mathcal{X}^\sharp[\mathtt{V} \mapsto \mathcal{V}_b^\sharp] & \text{otherwise} \end{cases}$$

strictness

where $\mathcal{V}_b^\sharp = E^\sharp[\![\, e \,]\!]\, \mathcal{X}^\sharp$.

Using a Galois connection (α_b, γ_b):

We can define best abstract arithmetic operators:

$$\begin{aligned} [c, c']_b^\sharp &\overset{\text{def}}{=} \alpha_b(\{\, x \mid c \leq x \leq c' \,\}) \\ -_b^\sharp\, \mathcal{X}_b^\sharp &\overset{\text{def}}{=} \alpha_b(\{\, -x \mid x \in \gamma(\mathcal{X}_b^\sharp) \,\}) \\ \mathcal{X}_b^\sharp +_b^\sharp \mathcal{Y}_b^\sharp &\overset{\text{def}}{=} \alpha_b(\{\, x+y \mid x \in \gamma(\mathcal{X}_b^\sharp),\, y \in \gamma(\mathcal{Y}_b^\sharp) \,\}) \\ &\vdots \end{aligned}$$

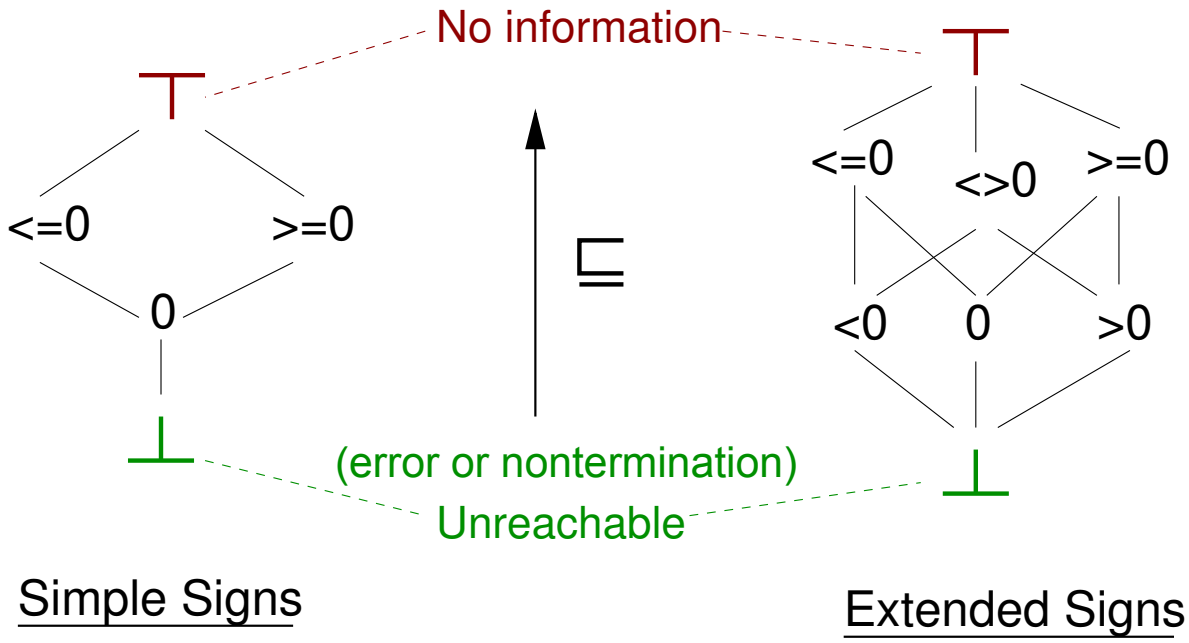
Note: in general, $E^\sharp[\![\, e \,]\!]$ is less precise than $\alpha_b \circ E[\![\, e \,]\!] \circ \gamma$

e.g. $e = \mathtt{V} - \mathtt{V}$ and $\gamma_b(\mathcal{X}^\sharp(\mathtt{V})) = [0, 1]$ $\qquad E^\sharp[\![\mathtt{V} - \mathtt{V}]\!]\mathcal{X}^\sharp = [-1, 1]$

The sign domain

The sign lattices

Hasse diagram: for the lattice $(\mathcal{B}^\sharp, \sqsubseteq_b, \bot_b^\sharp, \top_b^\sharp)$



The extended sign domain is a refinement of the simple sign domain.

The diagram implicitly defines $\cup^\sharp$ and $\cap^\sharp$ as the least upper bound and greatest lower bound for $\sqsubseteq$.

Operations on simple signs

Abstraction α: there is a Galois connection between $\mathcal{B}^\sharp$ and $\mathcal{P}(\mathbb{I})$:

$$\alpha_b(S) \stackrel{\text{def}}{=} \begin{cases} \bot_b^\sharp & \text{if } S = \emptyset \\ 0 & \text{if } S = \{0\} \\ \geq 0 & \text{else if } \forall s \in S,\ s \geq 0 \\ \leq 0 & \text{else if } \forall s \in S,\ s \leq 0 \\ \top_b^\sharp & \text{otherwise} \end{cases}$$

Derived abstract arithmetic operators:

$$c_b^\sharp \stackrel{\text{def}}{=} \alpha_b(\{c\}) = \begin{cases} 0 & \text{if } c = 0 \\ \leq 0 & \text{if } c < 0 \\ \geq 0 & \text{if } c > 0 \end{cases}$$

$$X^\sharp +_b^\sharp Y^\sharp \stackrel{\text{def}}{=} \alpha_b(\{\, x + y \mid x \in \gamma_b(X^\sharp),\ y \in \gamma_b(Y^\sharp)\,\})$$

$$= \begin{cases} \bot_b^\sharp & \text{if } X \text{ or } Y^\sharp = \bot_b^\sharp \quad \boxed{\text{strictness}} \\ 0 & \text{if } X^\sharp = Y^\sharp = 0 \\ \leq 0 & \text{else if } X^\sharp \text{ and } Y^\sharp \in \{0, \leq 0\} \\ \geq 0 & \text{else if } X^\sharp \text{ and } Y^\sharp \in \{0, \geq 0\} \\ \top_b^\sharp & \text{otherwise} \end{cases}$$

Operations on simple signs (cont.)

Abstract test examples:

$$C^\sharp[\![X \leq 0]\!]\, \mathcal{X}^\sharp \overset{\text{def}}{=} \left(\begin{cases} \mathcal{X}^\sharp[X \mapsto 0] & \text{if } \mathcal{X}^\sharp(X) \in \{0, \geq 0\} \\ \mathcal{X}^\sharp[X \mapsto \leq 0] & \text{if } \mathcal{X}^\sharp(X) \in \{\top_b^\sharp, \leq 0\} \\ \bot^\sharp & \text{if } \mathcal{X}^\sharp = \bot^\sharp \end{cases} \right)$$

$$C^\sharp[\![X - c \leq 0]\!]\, \mathcal{X}^\sharp \overset{\text{def}}{=} \left(\begin{cases} C^\sharp[\![X \leq 0]\!]\, \mathcal{X}^\sharp & \text{if } c \leq 0 \\ \mathcal{X}^\sharp & \text{otherwise} \end{cases} \right)$$

$$C^\sharp[\![X - Y \leq 0]\!]\, \mathcal{X}^\sharp \overset{\text{def}}{=}$$

$$\begin{cases} C^\sharp[\![X \leq 0]\!]\, \mathcal{X}^\sharp & \text{if } \mathcal{X}^\sharp(Y) \in \{0, \leq 0\} \\ \mathcal{X}^\sharp & \text{otherwise} \end{cases} \quad \cap^\sharp \quad \begin{cases} C^\sharp[\![Y \geq 0]\!]\, \mathcal{X}^\sharp & \text{if } \mathcal{X}^\sharp(X) \in \{0, \geq 0\} \\ \mathcal{X}^\sharp & \text{otherwise} \end{cases}$$

best

Example: $C^\sharp[\![X - Y \leq 0]\!]\{X \geq 0, Y \leq 0\} =$
$= \{X = 0, Y \leq 0\} \cap^\sharp \{X \geq 0, Y = 0\} = \{X = 0, Y = 0\}$

Operations on simple signs (cont.)

Abstract test examples:

$$\mathrm{C}^\sharp[\![\, \mathtt{X} \leq 0 \,]\!]\, \mathcal{X}^\sharp \overset{\text{def}}{=} \left(\begin{cases} \mathcal{X}^\sharp[\mathtt{X} \mapsto 0] & \text{if } \mathcal{X}^\sharp(\mathtt{X}) \in \{0, \geq 0\} \\ \mathcal{X}^\sharp[\mathtt{X} \mapsto \leq 0] & \text{if } \mathcal{X}^\sharp(\mathtt{X}) \in \{\top_b^\sharp, \leq 0\} \\ \bot^\sharp & \text{if } \mathcal{X}^\sharp = \bot^\sharp \end{cases} \right)$$

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - c \leq 0 \,]\!]\, \mathcal{X}^\sharp \overset{\text{def}}{=} \left(\begin{cases} \mathrm{C}^\sharp[\![\, \mathtt{X} \leq 0 \,]\!]\, \mathcal{X}^\sharp & \text{if } c \leq 0 \\ \mathcal{X}^\sharp & \text{otherwise} \end{cases} \right)$$

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - \mathtt{Y} \leq 0 \,]\!]\, \mathcal{X}^\sharp \overset{\text{def}}{=}$$
$$\begin{cases} \mathrm{C}^\sharp[\![\, \mathtt{X} \leq 0 \,]\!]\, \mathcal{X}^\sharp & \text{if } \mathcal{X}^\sharp(\mathtt{Y}) \in \{0, \leq 0\} \\ \mathcal{X}^\sharp & \text{otherwise} \end{cases} \bigcap^\sharp \begin{cases} \mathrm{C}^\sharp[\![\, \mathtt{Y} \geq 0 \,]\!]\, \mathcal{X}^\sharp & \text{if } \mathcal{X}^\sharp(\mathtt{X}) \in \{0, \geq 0\} \\ \mathcal{X}^\sharp & \text{otherwise} \end{cases}$$

Other cases: $\mathrm{C}^\sharp[\![\, \mathit{expr} \bowtie 0 \,]\!]\, \mathcal{X}^\sharp \overset{\text{def}}{=} \mathcal{X}^\sharp$ is always a sound abstraction (but not necessarily the best)

Simple sign analysis example

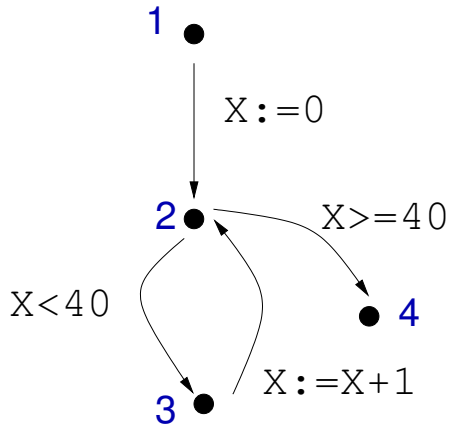
Example analysis using the simple sign domain:

```
X:=0;
while X<40 do
  X:=X+1
done
```

Program

$$\begin{cases} \mathcal{X}_2^{\sharp i+1} &= \mathrm{C}^\sharp[\![\, \mathtt{X} := 0 \,]\!]\, \mathcal{X}_1^{\sharp i} \cup \\ & \quad\ \mathrm{C}^\sharp[\![\, \mathtt{X} := \mathtt{X} + 1 \,]\!]\, \mathcal{X}_3^{\sharp i} \\ \mathcal{X}_3^{\sharp i+1} &= \mathrm{C}^\sharp[\![\, \mathtt{X} < 40 \,]\!]\, \mathcal{X}_2^{\sharp i} \\ \mathcal{X}_4^{\sharp i+1} &= \mathrm{C}^\sharp[\![\, \mathtt{X} \geq 40 \,]\!]\, \mathcal{X}_2^{\sharp i} \end{cases}$$

Iteration system



CFG

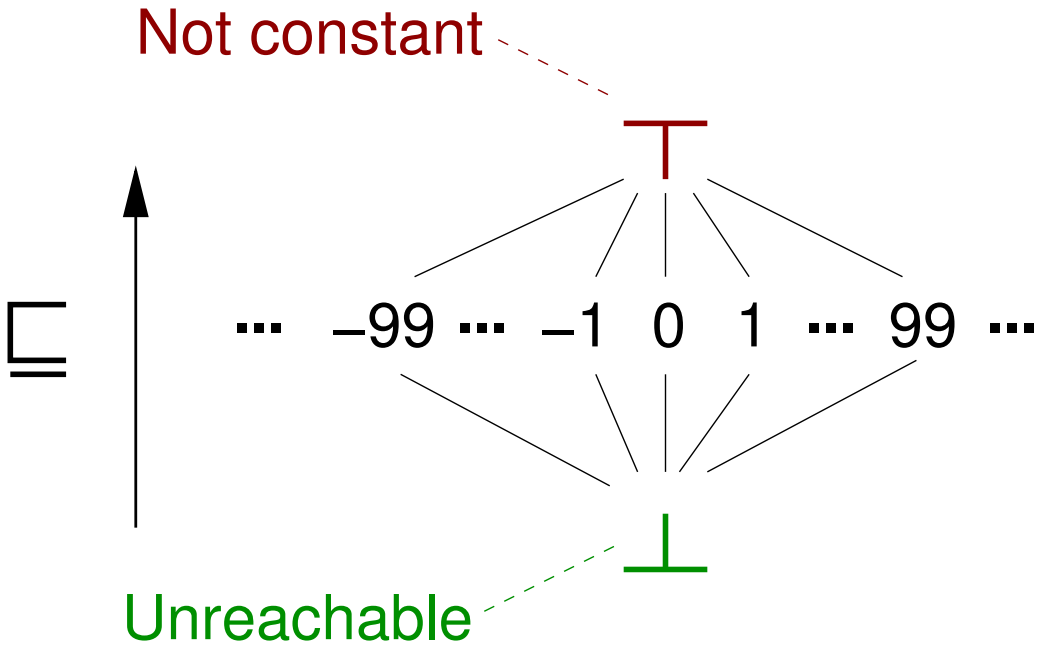
ℓ	$\mathcal{X}_\ell^{\sharp 0}$	$\mathcal{X}_\ell^{\sharp 1}$	$\mathcal{X}_\ell^{\sharp 2}$	$\mathcal{X}_\ell^{\sharp 3}$	$\mathcal{X}_\ell^{\sharp 4}$	$\mathcal{X}_\ell^{\sharp 5}$
1	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$
2	$\bot^\sharp$	$\mathtt{X} = 0$	$\mathtt{X} = 0$	$\mathtt{X} \geq 0$	$\mathtt{X} \geq 0$	$\mathtt{X} \geq 0$
3	$\bot^\sharp$	$\bot^\sharp$	$\mathtt{X} = 0$	$\mathtt{X} = 0$	$\mathtt{X} \geq 0$	$\mathtt{X} \geq 0$
4	$\bot^\sharp$	$\bot^\sharp$	$\mathtt{X} = 0$	$\mathtt{X} = 0$	$\mathtt{X} \geq 0$	$\mathtt{X} \geq 0$

Iterations

The constant domain

The constant lattice

Hasse diagram:



$\mathcal{B}^\sharp = \mathbb{I} \cup \{\top_b^\sharp; \bot_b^\sharp\}$

The lattice is flat but infinite.

Operations on constants

Abstraction α: there is a Galois connection:

$$\alpha_b(S) \stackrel{\text{def}}{=} \begin{cases} \bot_b^\sharp & \text{if } S = \emptyset \\ c & \text{if } S = \{c\} \\ \top_b^\sharp & \text{otherwise} \end{cases}$$

Thus: $\alpha_b(\{1,3\}) = \top_b^\sharp$

Derived abstract arithmetic operators:

$$c_b^\sharp \stackrel{\text{def}}{=} c$$

$$(X^\sharp) +_b^\sharp (Y^\sharp) \stackrel{\text{def}}{=} \begin{cases} \bot_b^\sharp & \text{if } X^\sharp \text{ or } Y^\sharp = \bot_b^\sharp \\ \top_b^\sharp & \text{else if } X^\sharp \text{ or } Y^\sharp = \top_b^\sharp \\ X^\sharp + Y^\sharp & \text{otherwise} \end{cases}$$

strictness

$$(X^\sharp) \times_b^\sharp (Y^\sharp) \stackrel{\text{def}}{=} \begin{cases} \bot_b^\sharp & \text{if } X^\sharp \text{ or } Y^\sharp = \bot_b^\sharp \\ 0 & \text{else if } X^\sharp \text{ or } Y^\sharp = 0 \\ \top_b^\sharp & \text{else if } X^\sharp \text{ or } Y^\sharp = \top_b^\sharp \\ X^\sharp \times Y^\sharp & \text{otherwise} \end{cases}$$

strictness

Operations on constants (cont.)

Abstract test examples:

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - c = 0 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } \mathcal{X}^\sharp(\mathtt{X}) \notin \{c, \top_b^\sharp\} \\ \mathcal{X}^\sharp[\mathtt{X} \mapsto c] & \text{otherwise} \end{cases}$$

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - \mathtt{Y} - c = 0 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=}$$

best

$$\left(\begin{cases} \mathrm{C}^\sharp[\![\, \mathtt{X} - (\mathcal{X}^\sharp(\mathtt{Y}) + c) = 0 \,]\!]\, \mathcal{X}^\sharp & \text{if } \mathcal{X}^\sharp(\mathtt{Y}) \notin \{\bot_b^\sharp, \top_b^\sharp\} \\ \mathcal{X}^\sharp & \text{otherwise} \end{cases} \right) \cap^\sharp$$

$$\left(\begin{cases} \mathrm{C}^\sharp[\![\, \mathtt{Y} - (\mathcal{X}^\sharp(\mathtt{X}) - c) = 0 \,]\!]\, \mathcal{X}^\sharp & \text{if } \mathcal{X}^\sharp(\mathtt{X}) \notin \{\bot_b^\sharp, \top_b^\sharp\} \\ \mathcal{X}^\sharp & \text{otherwise} \end{cases} \right)$$

Constant analysis example

$\mathcal{B}^\sharp$ has finite height, the $(\mathcal{X}_\ell^{\sharp i})$ converge in finite time.

(even though $\mathcal{B}^\sharp$ is infinite...)

Analysis example:

Loop invariant:
$[\mathtt{X} \mapsto 0, \mathtt{Y} \mapsto 10] \sqcup^\sharp [\mathtt{X} \mapsto 7, \mathtt{Y} \mapsto 10] =$
$[\mathtt{X} \mapsto \top_b^\sharp, \mathtt{Y} \mapsto 10]$

```
X:=0; Y:=10;
while X<100 do
  Y:=Y-3;
  X:=X+Y; •
  Y:=Y+3
done
```

The constant analysis finds, at •, the invariant: $\begin{cases} \mathtt{X} = \top_b^\sharp \\ \mathtt{Y} = 7 \end{cases}$

Note: the analysis can find constants that do not appear syntactically in the program.

gcc compiler

`-fgcse`

Perform a global common subexpression elimination pass. This pass also performs global constant and copy propagation.

```
int x = 14;
int y = 7 - x / 2;
return y * (28 / x + 2);
```

Propagating x yields:

```
int x = 14;
int y = 7 - 14 / 2;
return y * (28 / 14 + 2);
```

Continuing to propagate yields the following

```
int x = 14;
int y = 0;
return 0;
```

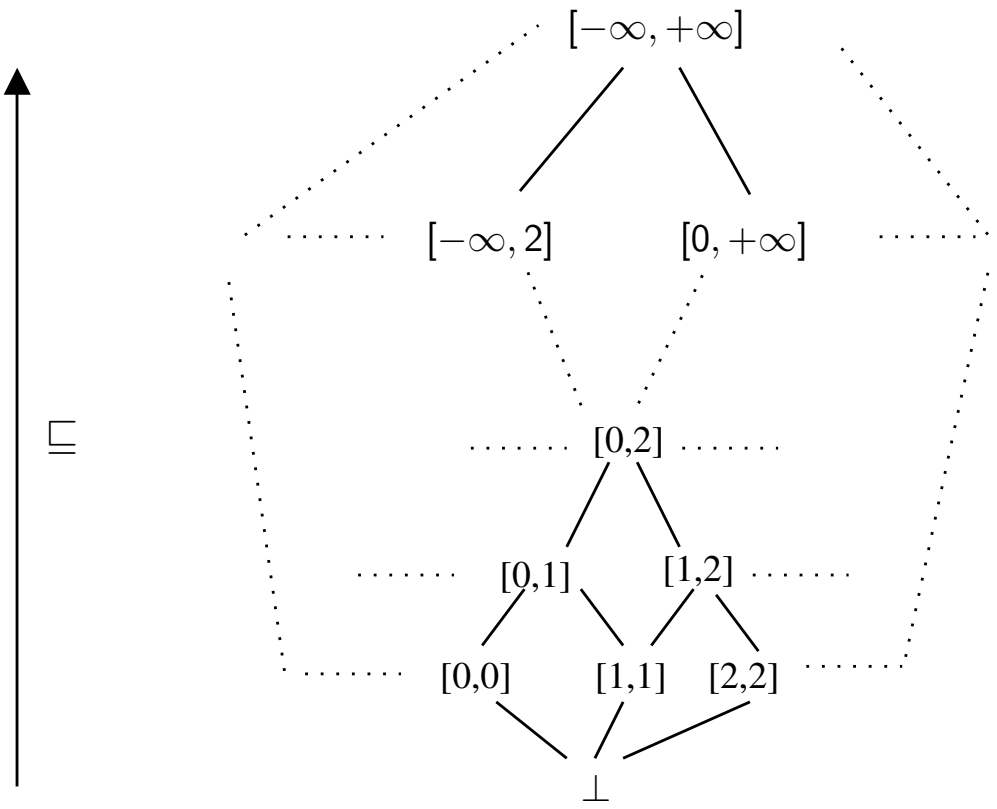
Enabled at levels `-O2`, `-O3`, `-Os`.

The interval domain

The interval lattice

Introduced by [Cous76].

$$\mathcal{B}^\sharp \stackrel{\text{def}}{=} \{\, [a,b] \mid a \in \mathbb{I} \cup \{-\infty\},\ b \in \mathbb{I} \cup \{+\infty\},\ a \leq b \,\} \cup \{\, \bot^\sharp_b \,\}$$



Note: intervals are open at infinite bounds $+\infty$, $-\infty$.

The interval lattice (cont.)

Galois connection (α_b, γ_b):

$$\gamma_b([a,b]) \stackrel{\text{def}}{=} \{ x \in \mathbb{I} \mid a \leq x \leq b \}$$

$$\alpha_b(\mathcal{X}) \stackrel{\text{def}}{=} \begin{cases} \bot_b^\sharp & \text{if } \mathcal{X} = \emptyset \\ [\min \mathcal{X}, \max \mathcal{X}] & \text{otherwise} \end{cases}$$

If $\mathbb{I} = \mathbb{Q}$, α_b is not always defined $\quad \alpha_b(\{x \in \mathbb{Q} \mid x < \sqrt{2}\}) = ?$

Partial order:

$$[a,b] \sqsubseteq_b [c,d] \stackrel{\text{def}}{\iff} a \geq c \text{ and } b \leq d$$

$$\top_b^\sharp \stackrel{\text{def}}{=}]-\infty, +\infty[$$

$$[a,b] \cup_b^\sharp [c,d] \stackrel{\text{def}}{=} [\min(a,c), \max(b,d)]$$

$$[a,b] \cap_b^\sharp [c,d] \stackrel{\text{def}}{=} \begin{cases} [\max(a,c), \min(b,d)] & \text{if } \max \leq \min \\ \bot_b^\sharp & \text{otherwise} \end{cases}$$

If $\mathbb{I} \neq \mathbb{Q}$, it is a complete lattice.

Interval abstract arithmetic operators

$$[c, c']^\sharp_b \stackrel{\text{def}}{=} [c, c']$$

$$-^\sharp_b [a, b] \stackrel{\text{def}}{=} [-b, -a]$$

$$[a, b] +^\sharp_b [c, d] \stackrel{\text{def}}{=} [a + c, b + d] \quad \text{best}$$

$$[a, b] -^\sharp_b [c, d] \stackrel{\text{def}}{=} [a - d, b - c] \quad \text{best}$$

$$[a, b] \times^\sharp_b [c, d] \stackrel{\text{def}}{=} [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \quad \text{best}$$

Example: $[-3, -2] \times^\sharp_b [-3, 6] = [-18, 9]$

Interval abstract arithmetic operators

$$[c, c']^\sharp_b \stackrel{\text{def}}{=} [c, c']$$

$$-^\sharp_b [a, b] \stackrel{\text{def}}{=} [-b, -a]$$

$$[a, b] +^\sharp_b [c, d] \stackrel{\text{def}}{=} [a + c, b + d]$$

$$[a, b] -^\sharp_b [c, d] \stackrel{\text{def}}{=} [a - d, b - c]$$

$$[a, b] \times^\sharp_b [c, d] \stackrel{\text{def}}{=} [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$$

$$[a, b] /^\sharp_b [c, d] \stackrel{\text{def}}{=} \begin{cases} \bot^\sharp_b & \text{if } [c,d] = [0,0] \\ [\min(a/c, a/d, b/c, b/d), \\ \ \max(a/c, a/d, b/c, b/d)] & \text{else if } 0 \leq c \\ [-b, -a]/^\sharp_b[-d, -c] & \text{else if } d \leq 0 \\ ([a, b]/^\sharp_b[c, 0]) \cup^\sharp_b ([a, b]/^\sharp_b[0, d]) & \text{otherwise} \end{cases}$$

best

when $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$

where $\pm\infty \times 0 = 0, \quad 0/0 = 0, \quad \forall x,\ x/\pm\infty = 0$
$\forall x > 0,\ x/0 = +\infty, \quad \forall x < 0,\ x/0 = -\infty$

Operators are strict: $-^\sharp_b \bot^\sharp_b = \bot^\sharp_b$, $[a, b] +^\sharp_b \bot^\sharp_b = \bot^\sharp_b$, etc. strictness clause

Interval abstract tests (non-generic)

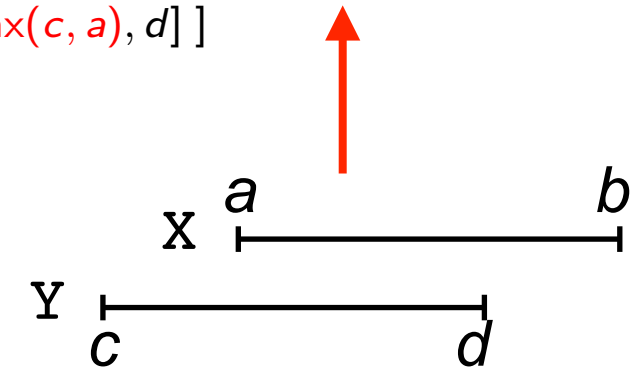
If $\mathcal{X}^\sharp(\mathtt{X}) = [a, b]$ and $\mathcal{X}^\sharp(\mathtt{Y}) = [c, d]$, we can define:

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - c \leq 0 \,]\!]\, \mathcal{X}^\sharp \quad \stackrel{\text{def}}{=} \quad \begin{cases} \bot^\sharp & \text{if } a > c \\ \mathcal{X}^\sharp[\, \mathtt{X} \mapsto [a, \min(b, c)] \,] & \text{otherwise} \end{cases}$$

best

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - \mathtt{Y} \leq 0 \,]\!]\, \mathcal{X}^\sharp \quad \stackrel{\text{def}}{=} \quad \begin{cases} \bot^\sharp & \text{if } a > d \\ \mathcal{X}^\sharp[\, \mathtt{X} \mapsto [a, \min(b, d)], \\ \qquad \mathtt{Y} \mapsto [\max(c, a), d] \,] & \text{otherwise} \end{cases}$$

best



Interval abstract tests (non-generic)

If $\mathcal{X}^\sharp(\mathtt{X}) = [a, b]$ and $\mathcal{X}^\sharp(\mathtt{Y}) = [c, d]$, we can define:

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - c \leq 0 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } a > c \\ \mathcal{X}^\sharp[\, \mathtt{X} \mapsto [a, \min(b, c)] \,] & \text{otherwise} \end{cases}$$

best

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - \mathtt{Y} \leq 0 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } a > d \\ \mathcal{X}^\sharp[\, \mathtt{X} \mapsto [a, \min(b, d)], \; \mathtt{Y} \mapsto [\max(c, a), d] \,] & \text{otherwise} \end{cases}$$

best

$$\mathrm{C}^\sharp[\![\, e \bowtie 0 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \mathcal{X}^\sharp \quad \text{otherwise}$$

always sound

Note: fall-back operators

- $\mathrm{C}^\sharp[\![\, e \bowtie 0 \,]\!]\, \mathcal{X}^\sharp = \mathcal{X}^\sharp$ is always sound.
- $\mathrm{C}^\sharp[\![\, \mathtt{X} := e \,]\!]\, \mathcal{X}^\sharp = \mathcal{X}^\sharp[\mathtt{X} \mapsto \top^\sharp_b]$ is always sound.

In pratice: More sophisticated algorithms for abstract tests and assignments

Generic non-relational abstract test

Abstract test algorithm: $\mathrm{C}^\sharp[\![\, e \bowtie 0 \,]\!]\, \mathcal{X}^\sharp$

Associate to each expression node an abstract value in $\mathcal{B}^\sharp$ using two traversals of the expression tree:

- first, a bottom-up evaluation using forward operators $\diamond_b^\sharp$,
- apply $\overleftarrow{\bowtie}\, 0_b^\sharp$ to the root,
- then, a top-down refinement using backward operators $\overleftarrow{\diamond}_b^\sharp$.

For each expression leaf, we get an abstract value $\mathcal{V}_b^\sharp$:

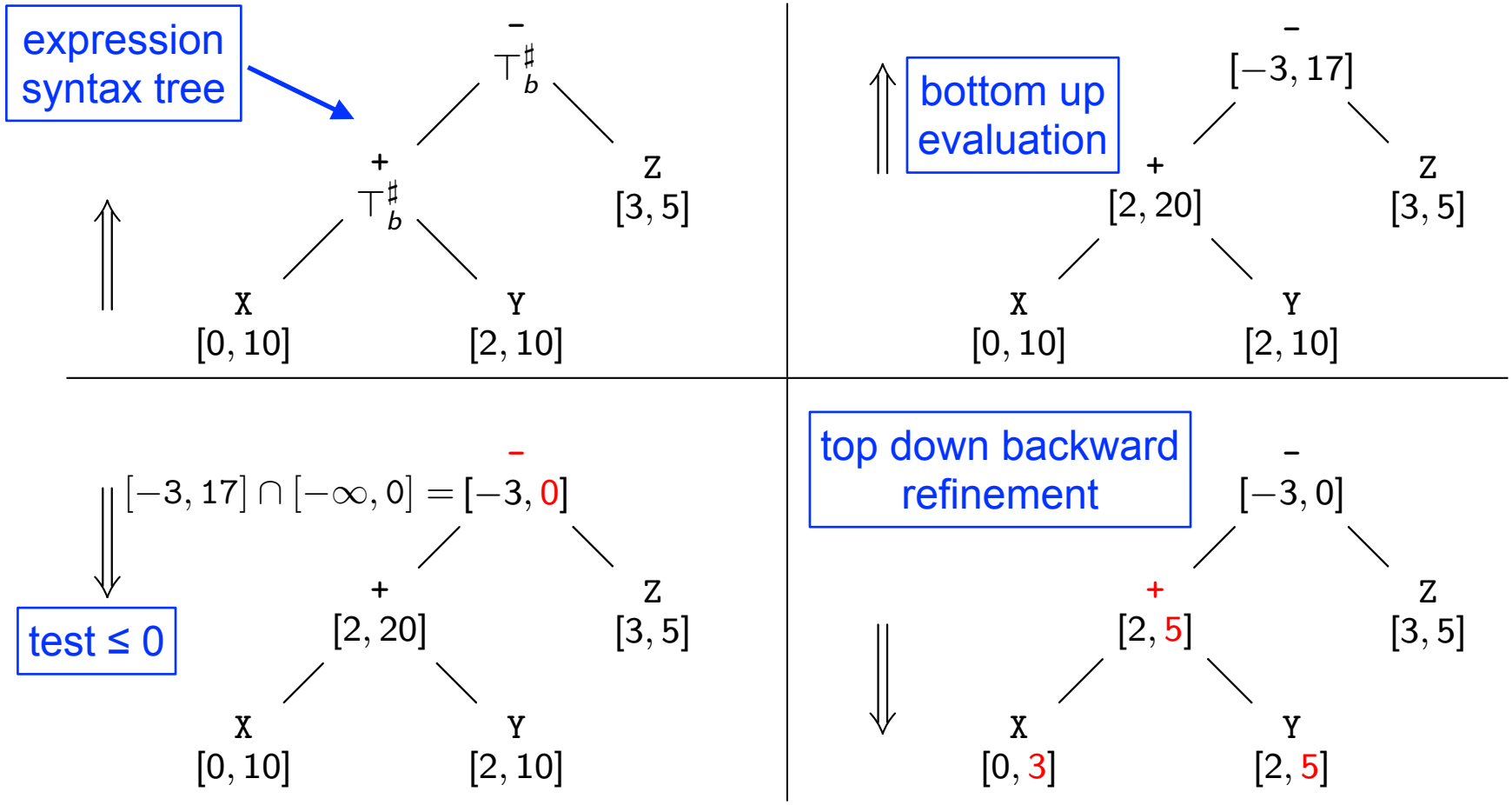
- for a variable V, replace $\mathcal{X}^\sharp(\mathtt{V})$ with $\mathcal{X}^\sharp(\mathtt{V}) \cap_b^\sharp \mathcal{V}_b^\sharp$,
- for a constant $[c, c']$, check that $[c, c']_b^\sharp \cap_b^\sharp \mathcal{V}_b^\sharp \neq \bot_b^\sharp$,
- $\Longrightarrow$ return $\bot^\sharp$ if some $\cap_b^\sharp \mathcal{V}_b^\sharp$ returns $\bot_b^\sharp$.

Improvement: local iterations [Gran92].

Interval test example

Example: $C^\sharp[\![X + Y - Z \leq 0]\!] \mathcal{X}^\sharp$

with $\mathcal{X}^\sharp = \{ X \mapsto [0, 10], Y \mapsto [2, 10], Z \mapsto [3, 5] \}$



Interval widening

Widening on non-relational domains:

Given a value widening $\nabla_b : \mathcal{B}^\sharp \times \mathcal{B}^\sharp \to \mathcal{B}^\sharp$,
we extend it point-wisely into a widening $\nabla : \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp$:

$$\mathcal{X}^\sharp \nabla \mathcal{Y}^\sharp \stackrel{\text{def}}{=} \lambda \mathtt{V}.(\mathcal{X}^\sharp(\mathtt{V}) \nabla_b \mathcal{Y}^\sharp(\mathtt{V}))$$

Interval widening example:

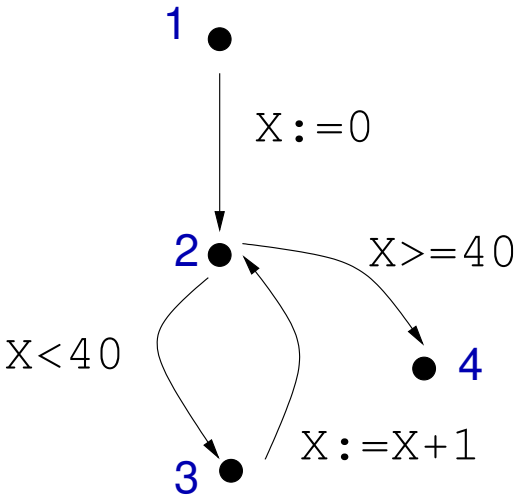
$$\bot^\sharp \quad \nabla_b \quad X^\sharp \quad \stackrel{\text{def}}{=} \quad X^\sharp$$

$$[a,b] \quad \nabla_b \quad [c,d] \quad \stackrel{\text{def}}{=} \quad \left[\begin{cases} a & \text{if } a \leq c \\ -\infty & \text{otherwise} \end{cases} , \begin{cases} b & \text{if } b \geq d \\ +\infty & \text{otherwise} \end{cases} \right]$$

Unstable bounds are set to $\pm\infty$.

Analysis with widening example

Analysis example with $\mathcal{W} = \{2\}$



ℓ	$\mathcal{X}_\ell^{\sharp 0}$	$\mathcal{X}_\ell^{\sharp 1}$	$\mathcal{X}_\ell^{\sharp 2}$	$\mathcal{X}_\ell^{\sharp 3}$	$\mathcal{X}_\ell^{\sharp 4}$	$\mathcal{X}_\ell^{\sharp 5}$
1	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$
2 $\triangledown$	$\bot^\sharp$	$= 0$	$= 0$	≥ 0	≥ 0	≥ 0
3	$\bot^\sharp$	$\bot^\sharp$	$= 0$	$= 0$	$\in [0, 39]$	$\in [0, 39]$
4	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	≥ 40	≥ 40

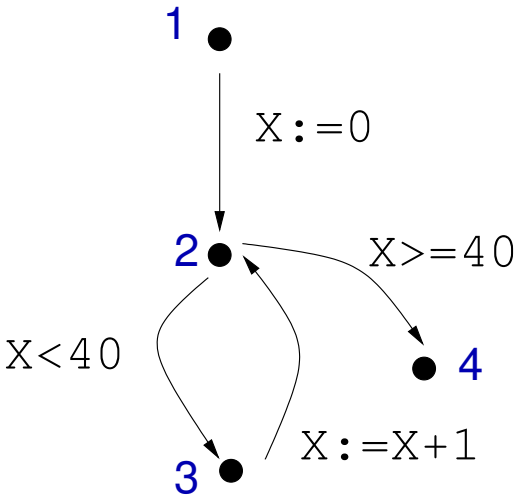
More precisely, at the widening point:

$$\begin{array}{llllllll}
\mathcal{X}_2^{\sharp 1} & = & \bot^\sharp & \triangledown_b\,([0,0] \cup_b^\sharp \bot^\sharp) & = & \bot^\sharp & \triangledown_b\,[0,0] & = & [0,0] \\
\mathcal{X}_2^{\sharp 2} & = & [0,0] & \triangledown_b\,([0,0] \cup_b^\sharp \bot^\sharp) & = & [0,0] & \triangledown_b\,[0,0] & = & [0,0] \\
\mathcal{X}_2^{\sharp 3} & = & [0,0] & \triangledown_b\,([0,0] \cup_b^\sharp [1,1]) & = & [0,0] & \triangledown_b\,[0,1] & = & [0,+\infty[\\
\mathcal{X}_2^{\sharp 4} & = & [0,+\infty[& \triangledown_b\,([0,0] \cup_b^\sharp [1,40]) & = & [0,+\infty[& \triangledown_b\,[0,40] & = & [0,+\infty[
\end{array}$$

Note that the most precise interval abstraction would be $\mathtt{X} \in [0, 40]$ at 2, and $\mathtt{X} = 40$ at 4.

Analysis with widening example

Analysis example with $\mathcal{W} = \{2\}$



Please type a program, upload a file from your hard-drive, or choose one the provided examples:

Choose File no file selected

user-supplied

```
/* type your program here ! */

var X:int;
begin
  X = 0;
  while (X<40) do
    X = X+1;
  done;
end
```

Numerical Abstract Domain: box

Kind of Analysis: f (sequence of forward and/or backward analysis)

Iterations/Widening options:

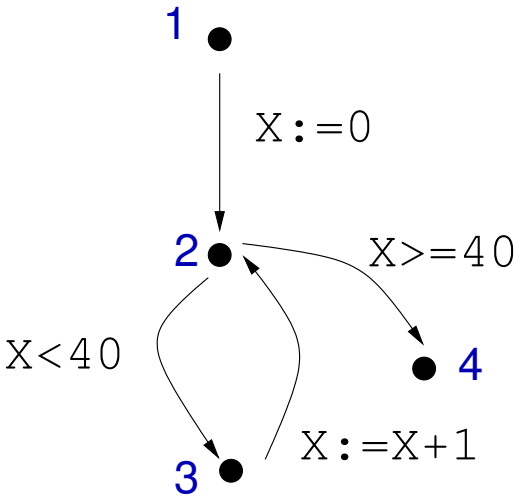
guided iterations 1 widening delay 0 descending steps

0 debugging level (0 to 6)

Hit the OK button to proceed: OK ! Reset

Analysis with widening example

Analysis example with $\mathcal{W} = \{2\}$



ℓ	$\mathcal{X}_\ell^{\sharp 0}$	$\mathcal{X}_\ell^{\sharp 1}$	$\mathcal{X}_\ell^{\sharp 2}$	$\mathcal{X}_\ell^{\sharp 3}$	$\mathcal{X}_\ell^{\sharp 4}$	$\mathcal{X}_\ell^{\sharp 5}$
1	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$
2 $\triangledown$	$\bot^\sharp$	$= 0$	$= 0$	≥ 0	≥ 0	≥ 0
3	$\bot^\sharp$	$\bot^\sharp$	$= 0$	$= 0$	$\in [0, 39]$	$\in [0, 39]$
4	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	≥ 40	≥ 40

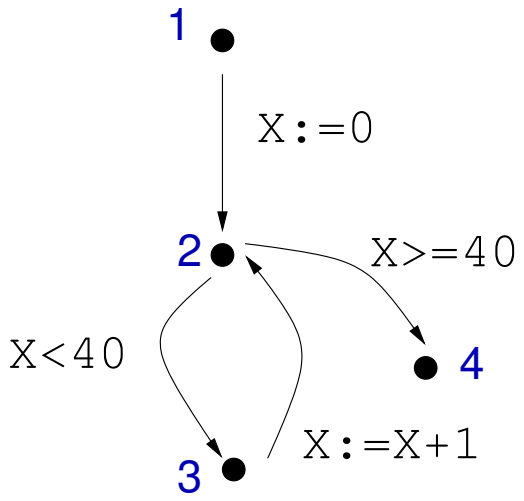
Result

```
Annotated program after forward analysis
var X : int;
begin
  /* (L4 C5) top */
  X = 0; /* (L5 C8) [|X>=0|] */
  while X < 40 do
    /* (L6 C17) [|X>=0; -X+39>=0|] */
    X = X + 1; /* (L7 C12)
                  [|X-1>=0; -X+40>=0|] */
  done; /* (L8 C7) [|X-40>=0|] */
end
```

Influence of the widening point and iteration strategy

Changing $\mathcal{W}$ changes the analysis result

Example: The analysis is less precise for $\mathcal{W} = \{3\}$.



ℓ	$\mathcal{X}_\ell^{\sharp 1}$	$\mathcal{X}_\ell^{\sharp 2}$	$\mathcal{X}_\ell^{\sharp 3}$	$\mathcal{X}_\ell^{\sharp 4}$	$\mathcal{X}_\ell^{\sharp 5}$	$\mathcal{X}_\ell^{\sharp 6}$
1	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$
2	$= 0$	$= 0$	$\in [0,1]$	$\in [0,1]$	≥ 0	≥ 0
3 $\triangledown$	$\bot^\sharp$	$= 0$	$= 0$	≥ 0	≥ 0	≥ 0
4	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	≥ 40

Intuition: extrapolation to $+\infty$ is no longer contained by the tests.

Narrowing

Using a widening makes the analysis less precise.

Some precision can be retrieved by using a narrowing $\vartriangle$.

Definition: narrowing $\vartriangle$

Binary operator $\mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ such that:

- $(\mathcal{X}^\sharp \cap^\sharp \mathcal{Y}^\sharp) \sqsubseteq (\mathcal{X}^\sharp \vartriangle \mathcal{Y}^\sharp) \sqsubseteq \mathcal{X}^\sharp$,
- for all sequences $(\mathcal{X}_i^\sharp)$, the decreasing sequence $(\mathcal{Y}_i^\sharp)$ defined by $\begin{cases} \mathcal{Y}_0^\sharp & \stackrel{\text{def}}{=} & \mathcal{X}_0^\sharp \\ \mathcal{Y}_{i+1}^\sharp & \stackrel{\text{def}}{=} & \mathcal{Y}_i^\sharp \vartriangle \mathcal{X}_{i+1}^\sharp \end{cases}$ is stationary.

This is not the dual of a widening!

Narrowing examples

Trivial narrowing:

$\mathcal{X}^\sharp \mathbin{\triangle} \mathcal{Y}^\sharp \overset{\text{def}}{=} \mathcal{X}^\sharp$ is a correct narrowing.

do nothing

Finite-time intersection narrowing:

$$\mathcal{X}^{\sharp i} \mathbin{\triangle} \mathcal{Y}^\sharp \overset{\text{def}}{=} \begin{cases} \mathcal{X}^{\sharp i} \cap^\sharp \mathcal{Y}^\sharp & \text{if } i \leq N \\ \mathcal{X}^{\sharp i} & \text{if } i > N \end{cases}$$

do a finite number of abstract intersections

Interval narrowing:

$$[a,b] \mathbin{\triangle_b} [c,d] \overset{\text{def}}{=} \left[\begin{cases} c & \text{if } a = -\infty \\ a & \text{otherwise} \end{cases} , \begin{cases} d & \text{if } b = +\infty \\ b & \text{otherwise} \end{cases} \right]$$

(refine only infinite bounds)

Point-wise extension to $\mathcal{D}^\sharp$: $\mathcal{X}^\sharp \mathbin{\triangle} \mathcal{Y}^\sharp \overset{\text{def}}{=} \lambda \mathtt{V}.(\mathcal{X}^\sharp(\mathtt{V}) \mathbin{\triangle_b} \mathcal{Y}^\sharp(\mathtt{V}))$

Iterations with narrowing

Let $\mathcal{X}^{\sharp\delta}_\ell$ be the result after widening stabilisation, *i.e.*:

$$\mathcal{X}^{\sharp\delta}_\ell \sqsupseteq \begin{cases} \top^\sharp & \text{if } \ell = e \\ \bigcup\limits^\sharp_{(\ell',c,\ell)\in A} \mathrm{C}^\sharp[\![\, c \,]\!]\, \mathcal{X}^{\sharp\delta}_{\ell'} & \text{if } \ell \neq e \end{cases}$$

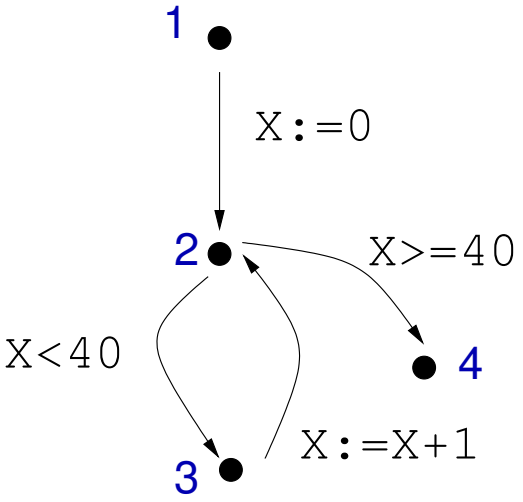
The following sequence is computed:

$$\mathcal{Y}^{\sharp 0}_\ell \stackrel{\text{def}}{=} \mathcal{X}^{\sharp\delta}_\ell \qquad \mathcal{Y}^{\sharp i+1}_\ell \stackrel{\text{def}}{=} \begin{cases} \top^\sharp & \text{if } \ell = e \\ \bigcup\limits^\sharp_{(\ell',c,\ell)\in A} \mathrm{C}^\sharp[\![\, c \,]\!]\, \mathcal{Y}^{\sharp i}_{\ell'} & \text{if } \ell \notin \mathcal{W} \\ \mathcal{Y}^{\sharp i}_\ell \mathbin{\triangle} \bigcup\limits^\sharp_{(\ell',c,\ell)\in A} \mathrm{C}^\sharp[\![\, c \,]\!]\, \mathcal{Y}^{\sharp i}_{\ell'} & \text{if } \ell \in \mathcal{W} \end{cases}$$

- the sequence $(\mathcal{Y}^{\sharp i}_\ell)$ is decreasing and converges in finite time,
- all $(\mathcal{Y}^{\sharp i}_\ell)$ are solutions of the abstract semantic system.

Analysis with narrowing example

Example with $\mathcal{W} = \{2\}$



ℓ	$\mathcal{Y}_\ell^{\sharp 0}$	$\mathcal{Y}_\ell^{\sharp 1}$	$\mathcal{Y}_\ell^{\sharp 2}$	$\mathcal{Y}_\ell^{\sharp 3}$
1	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$
2 $\triangle$	≥ 0	$\in [0, 40]$	$\in [0, 40]$	$\in [0, 40]$
3	$\in [0, 39]$	$\in [0, 39]$	$\in [0, 39]$	$\in [0, 39]$
4	≥ 40	≥ 40	$= 40$	$= 40$

Narrowing at 2 gives:

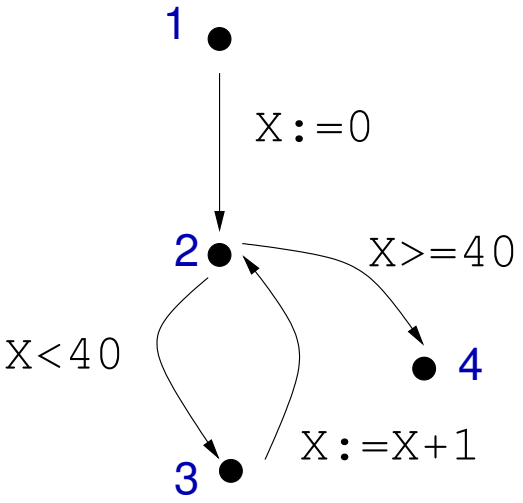
$$\begin{array}{lcllcll} \mathcal{Y}_2^{\sharp 1} & = & [0, +\infty[\triangle_b ([0,0] \cup_b^\sharp [1,40]) & = & [0, +\infty[\triangle_b [0,40] & = & [0,40] \\ \mathcal{Y}_2^{\sharp 2} & = & [0,40] \quad \triangle_b ([0,0] \cup_b^\sharp [1,40]) & = & [0,40] \quad \triangle_b [0,40] & = & [0,40] \end{array}$$

Then $\mathcal{Y}_2^{\sharp 2} : \mathtt{X} \in [0, 40]$ gives $\mathcal{Y}_4^{\sharp 3} : \mathtt{X} = 40$.

We found the most precise invariants!

Analysis with narrowing example

Example with $\mathcal{W} = \{2\}$



ℓ	$\mathcal{Y}^{\sharp 0}_\ell$	$\mathcal{Y}^{\sharp 1}_\ell$	$\mathcal{Y}^{\sharp 2}_\ell$	$\mathcal{Y}^{\sharp 3}_\ell$
1	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$
2 △	≥ 0	$\in [0, 40]$	$\in [0, 40]$	$\in [0, 40]$
3	$\in [0, 39]$	$\in [0, 39]$	$\in [0, 39]$	$\in [0, 39]$
4	≥ 40	≥ 40	$= 40$	$= 40$

```
/* type your program here ! */

var X:int;
begin
  X = 0;
  while (X<40) do
    X = X+1;
  done;
end
```

Numerical Abstract Domain: box

Kind of Analysis: f (sequence of forward and/or backward analysis)

Iterations/Widening options:

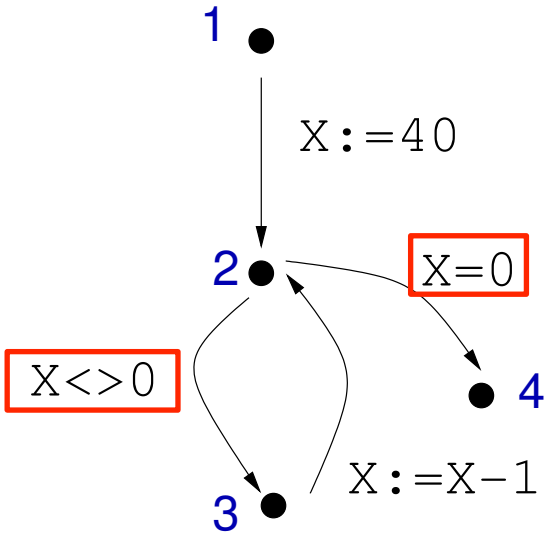
guided iterations 1 widening delay 1 descending steps

Result

```
*** Analysis...
.. in 3,1 iterations without stabilization ***
Annotated program after forward analysis
var X : int;
begin
  /* (L4 C5) top */
  X = 0; /* (L5 C8) [|X>=0; -X+40>=0|] */
  while X < 40 do
    /* (L6 C17) [|X>=0; -X+39>=0|] */
    X = X + 1; /* (L7 C12)
                  [|X-1>=0; -X+40>=0|] */
  done; /* (L8 C7) [|X-40=0|] */
end
```

Improving the widening

Example of imprecise analysis



ℓ	intervals with ∇_b	extended signs
1	$\top^\sharp$	$\top^\sharp$
2 ∇	$X \leq 40$	$X \geq 0$
3	$X \leq 40$	$X > 0$
4	$X = 0$	$X = 0$

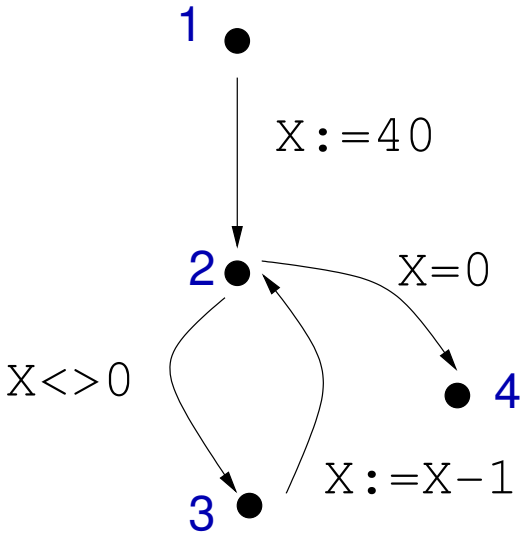
$X<>0 \sqcap^\sharp X \geq 0 = X > 0$

Result

```
*** Analysis...
.. in 3,1 iterations with stabilization ***
Annotated program after forward analysis
var X : int;
begin
  /* (L4 C5) top */
  X = 40; /* (L5 C9) [|-X+40>=0|] */
  while not X == 0 do
    /* (L6 C22) [|-X+40>=0|] */
    X = X - 1; /* (L7 C12) [|-X+39>=0|] */
  done; /* (L8 C7) [|X=0|] */
end
```

Improving the widening

Example of imprecise analysis



ℓ	intervals with ∇_b	extended signs	intervals with ∇'_b
1	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$
2 ∇	$\mathtt{X} \leq 40$	$\mathtt{X} \geq 0$	$\mathtt{X} \in [0, 40]$
3	$\mathtt{X} \leq 40$	$\mathtt{X} > 0$	$\mathtt{X} \in [0, 40]$
4	$\mathtt{X} = 0$	$\mathtt{X} = 0$	$\mathtt{X} = 0$

The interval domain cannot prove that $\mathtt{X} \geq 0$ at 2, while the (less powerful) sign domain can!

Solution: improve the interval widening

$$[a,b] \nabla'_b [c,d] \stackrel{\text{def}}{=} \left[\begin{cases} a & \text{if } a \leq c \\ 0 & \text{if } 0 \leq c < a \\ -\infty & \text{otherwise} \end{cases} , \begin{cases} b & \text{if } b \geq d \\ 0 & \text{if } 0 \geq b > d \\ +\infty & \text{otherwise} \end{cases} \right]$$

(∇'_b checks the stability of 0)

Widening with thresholds

Analysis problem:

```
X:=0;
while • 1=1 do
  if [0,1]=0 then
    X:=X+1;
    if X>40 then X:=0 fi
  fi
done
```

[0,1] =? 0
is a random choice

We wish to prove that $X \in [0, 40]$ at •.

Widening with thresholds

Analysis problem:

```
X:=0;
while • 1=1 do
  if [0,1]=0 then
    X:=X+1;
    if X>40 then X:=0 fi
  fi
done
```

We wish to prove that $X \in [0, 40]$ at •.

Result

```
*** Analysis...
.. in 3,1 iterations with stabilization ***
Annotated program after forward analysis
var X : int, Y : int;
begin
  /* (L4 C5) top */
  X = 0; /* (L5 C4) [|X>=0|] */
  while true do
    /* (L6 C15) [|X>=0|] */
    Y = [-inf; inf]; /* (L7 C11) [|X>=0|] */
    if Y == 0 then
      /* (L8 C18) [|X>=0; Y=0|] */
      X = X + 1; /* (L9 C9) [|X-1>=0; Y=0|] */
      if X > 40 then
        /* (L10 C16) [|X-41>=0; Y=0|] */
        X = 0; /* (L10 C21) [|X=0; Y=0|] */
      endif; /* (L10 C28)
                [|X>=0; -X+40>=0; Y=0|] */
    endif; /* (L11 C8) [|X>=0|] */
  done; /* (L12 C5) [|X>=0|] */
end
```

Widening with thresholds

Analysis problem:

```
X:=0;
while • 1=1 do
  if [0,1]=0 then
    X:=X+1;
    if X>40 then X:=0 fi
  fi
done
```

We wish to prove that $X \in [0, 40]$ at •.

- Widening at • finds the loop invariant $\mathtt{X} \in [0, +\infty[$.
  $\mathcal{X}_\bullet^\sharp = [0,0] \nabla_b ([0,0] \cup^\sharp [0,1]) = [0,0] \nabla_b [0,1] = [0, +\infty[$

[0,0] when [0,1]=0 is false
[0,1] = [1,1] $\cup^\sharp$ [0,0] when [0,1]=0 is true

Widening with thresholds

Analysis problem:

```
X:=0;
while • 1=1 do
  if [0,1]=0 then
    X:=X+1;
    if X>40 then X:=0 fi
  fi
done
```

We wish to prove that $X \in [0, 40]$ at •.

- Widening at • finds the loop invariant $\mathtt{X} \in [0, +\infty[$.
  $\mathcal{X}^\sharp_\bullet = [0,0] \nabla_b ([0,0] \cup^\sharp [0,1]) = [0,0] \nabla_b [0,1] = [0, +\infty[$
- Narrowing is unable to refine the invariant:
  $\mathcal{Y}^\sharp_\bullet = [0, +\infty[\Delta_b ([0,0] \cup^\sharp [0, +\infty[) = [0, +\infty[$

  (the code that limits X is not executed at every loop iteration)

Widening with thresholds (cont.)

Solution:

Choose a finite set T of thresholds containing $+\infty$ and $-\infty$.

Definition: widening with thresholds ∇_b^T

$$[a,b] \nabla_b^T [c,d] \stackrel{\text{def}}{=} \left[\begin{cases} a & \text{if } a \leq c \\ \max\{x \in T \mid x \leq c\} & \text{otherwise} \end{cases} , \right.$$
$$\left. \begin{cases} b & \text{if } b \geq d \\ \min\{x \in T \mid x \geq d\} & \text{otherwise} \end{cases} \right]$$

The widening tests and stops at the first stable bound in T.

Widening with thresholds (cont.)

Applications:

- On the previous example, we find:
  $\mathtt{X} \in [\, 0,\ \min\{x \in T \mid x \geq 40\}\,]$.

$T = \{0,40\}$
constants occurring in P

```
X:=0;
while • 1=1 do
  if [0,1]=0 then
    X:=X+1;
    if X>40 then X:=0 fi
  fi
done
```

First iteration: $[0,0] \nabla_b [0,1] = [0,40]$
Second iteration:
$[0,40] \oplus 1 = [1,41]$
$([1,41] \cap^\sharp (-\infty, 40]) \cup^\sharp ([0,0]) = [1,40] \cup^\sharp [0,0] = [0,40]$

Widening with thresholds (cont.)

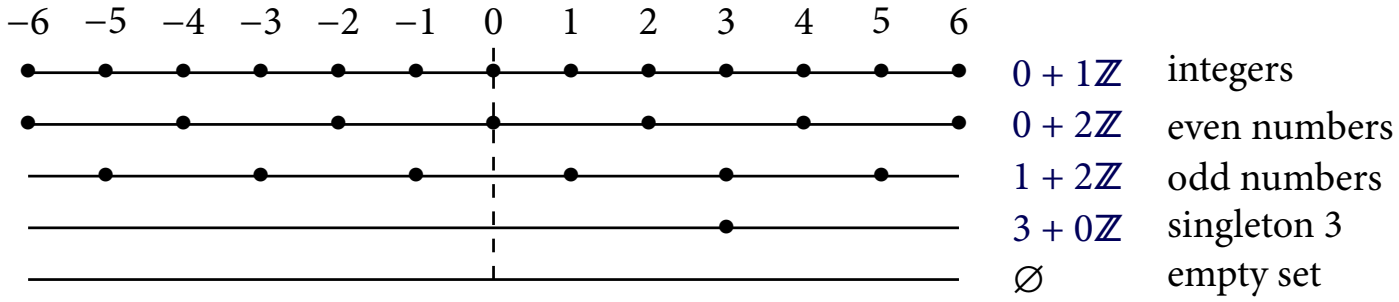
Applications:

- On the previous example, we find:
  $\mathtt{X} \in [\, 0,\ \min\, \{x \in T \mid x \geq 40\}\,]$.

- Useful when it is easy to find a 'good' set T.

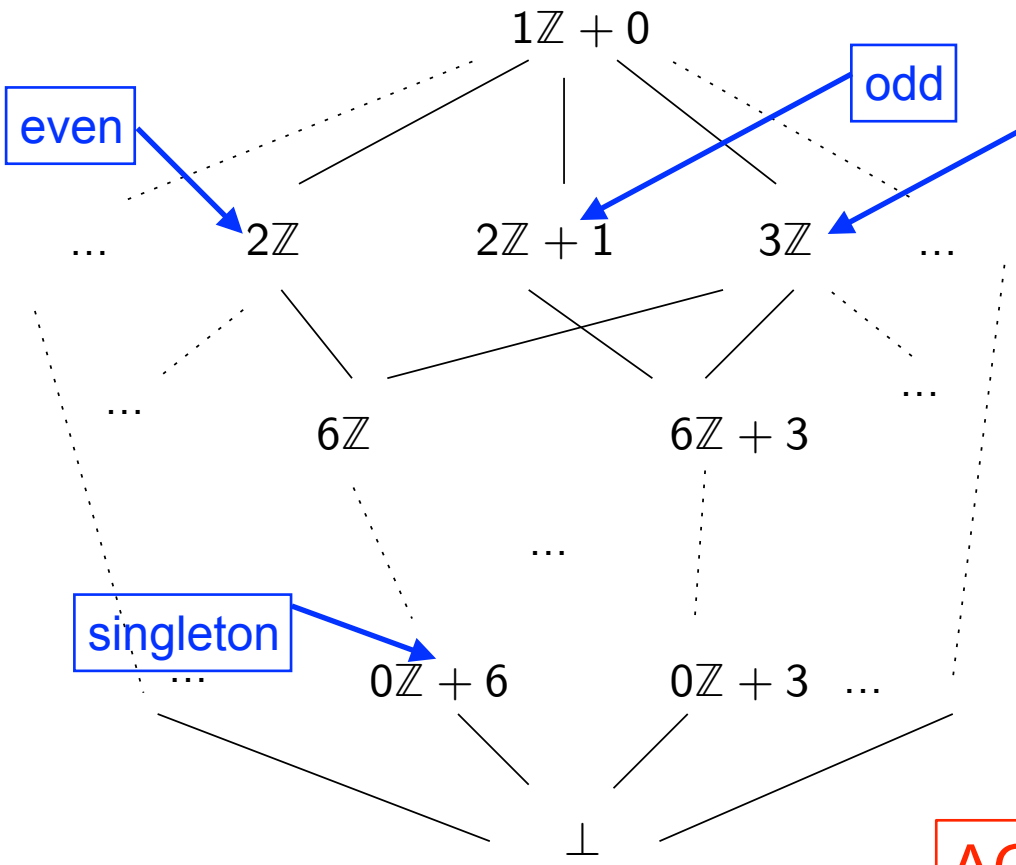
  Example: array bound-checking

The congruence domain



The congruence lattice

$$\mathcal{B}^\sharp \stackrel{\text{def}}{=} \{ (a\mathbb{Z} + b) \mid a \in \mathbb{N},\ b \in \mathbb{Z} \} \cup \{ \bot_b^\sharp \}$$



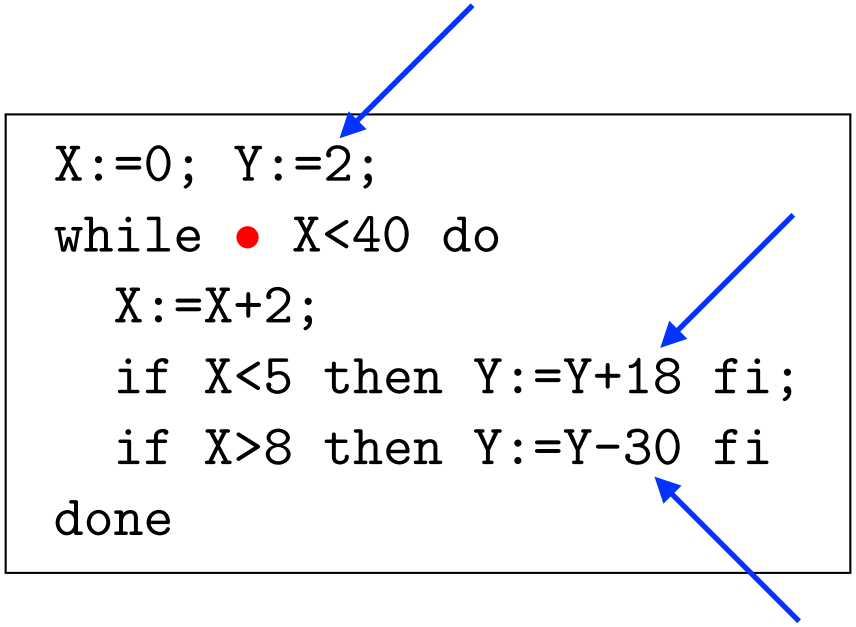
*a*ℤ+*b* is the congruence class of *b* modulo *a*, i.e. $x \in a\mathbb{Z}+b$ when *a* divides *x*-*b*

Introduced by Granger [Gran89].
We take $\mathbb{I} = \mathbb{Z}$.

ACC but not DCC

$2\mathbb{Z} \supseteq 4\mathbb{Z} \supseteq 8\mathbb{Z} \supseteq \ldots$

Congruence analysis example



We find, at •, the loop invariant $\begin{cases} X \in 2\mathbb{Z} \\ Y \in 6\mathbb{Z} + 2 \end{cases}$

6 = greatest common divisor of 18 and 30

Congruence analysis example

```
*** Analysis...
.. in 4,1 iterations with stabilization ***
Annotated program after forward analysis
var X : int, Y : int;
begin
  /* (L4 C5) [|Y=0 mod 1; X=0 mod 1|] */
  X = 0; /* (L5 C4) [|Y=0 mod 1; X=0|] */
  Y = 2; /* (L5 C9)
            [|Y-2=0 mod 6; 3X=0 mod 6|] */
  while X < 40 do
    /* (L6 C15) [|Y-2=0 mod 6; 3X=0 mod 6|] */
    X = X + 2; /* (L7 C8)
                  [|Y-2=0 mod 6; 3X=0 mod 6|] */
    if X < 5 then
      /* (L8 C15) [|Y-2=0 mod 6; 3X=0 mod 6|] */
      Y = Y + 18; /* (L8 C23)
                     [|Y-2=0 mod 6; 3X=0 mod 6|] */
    endif; /* (L8 C30)
              [|Y-2=0 mod 6; 3X=0 mod 6|] */
    if X > 8 then
      /* (L9 C15) [|Y-2=0 mod 6; 3X=0 mod 6|] */
      Y = Y - 30; /* (L9 C23)
                     [|Y-2=0 mod 6; 3X=0 mod 6|] */
    endif; /* (L9 C30)
              [|Y-2=0 mod 6; 3X=0 mod 6|] */
  done; /* (L10 C5)
           [|Y-2=0 mod 6; 3X=0 mod 6|] */
end
```

```
X:=0; Y:=2;
while • X<40 do
  X:=X+2;
  if X<5 then Y:=Y+18 fi;
  if X>8 then Y:=Y-30 fi
done
```

We find, at •, the loop invariant $\begin{cases} X \in 2\mathbb{Z} \\ Y \in 6\mathbb{Z}+2 \end{cases}$

Reduced products of domains

Non-reduced product of domains

Product representation:

Cartesian product $\mathcal{D}^\sharp_{1\times 2}$ of $\mathcal{D}^\sharp_1$ and $\mathcal{D}^\sharp_2$:

- $\mathcal{D}^\sharp_{1\times 2} \stackrel{\text{def}}{=} \mathcal{D}^\sharp_1 \times \mathcal{D}^\sharp_2$
- $\gamma_{1\times 2}(\mathcal{X}^\sharp_1, \mathcal{X}^\sharp_2) \stackrel{\text{def}}{=} \gamma_1(\mathcal{X}^\sharp_1) \cap \gamma_2(\mathcal{X}^\sharp_2)$
- $\alpha_{1\times 2}(\mathcal{X}) \stackrel{\text{def}}{=} (\alpha_1(\mathcal{X}), \alpha_2(\mathcal{X}))$
- $(\mathcal{X}^\sharp_1, \mathcal{X}^\sharp_2) \sqsubseteq_{1\times 2} (\mathcal{Y}^\sharp_1, \mathcal{Y}^\sharp_2) \stackrel{\text{def}}{\iff} X^\sharp_1 \sqsubseteq_1 \mathcal{Y}^\sharp_1$ and $X^\sharp_2 \sqsubseteq_2 \mathcal{Y}^\sharp_2$

Abstract operators: performed in parallel on both components:

- $(\mathcal{X}^\sharp_1, \mathcal{X}^\sharp_2) \cup^\sharp_{1\times 2} (\mathcal{Y}^\sharp_1, \mathcal{Y}^\sharp_2) \stackrel{\text{def}}{=} (\mathcal{X}^\sharp_1 \cup^\sharp_1 \mathcal{Y}^\sharp_1, \mathcal{X}^\sharp_2 \cup^\sharp_2 \mathcal{Y}^\sharp_2)$
  and the same for $\nabla^\sharp_{1\times 2}$ and $\triangle^\sharp_{1\times 2}$
- $C^\sharp[\![c]\!]_{1\times 2}(\mathcal{X}^\sharp_1, \mathcal{X}^\sharp_2) \stackrel{\text{def}}{=} (C^\sharp[\![c]\!]_1(\mathcal{X}^\sharp_1), C^\sharp[\![c]\!]_2(\mathcal{X}^\sharp_2))$

Non-reduced product example

The product analysis is no more precise than two separate analyses.

Example: interval–congruence product:

Loop invariant:
$[1, +\infty) \nabla [1, 12] = [1, 12]$

```
X:=1;
while X-10<=0 do
  X:=X+2
done;
•if X-12>=0 then♦ X:=0★ fi
```

	interval	congruence	product γ
•	$X \in [11, 12]$	$X \equiv 1\ [2]$	$X = 11$ ←
♦	$X = 12$	$X \equiv 1\ [2]$	$\emptyset$ ←
★	$X = 0$	$X = 0$	$X = 0$

We cannot prove that the `if` branch is never taken!

Fully-reduced product

Definition:

Given the Galois connections (α_1, γ_1) and (α_2, γ_2) on $\mathcal{D}_1^\sharp$ and $\mathcal{D}_2^\sharp$ we define the reduction operator ρ as:

$$\rho : \mathcal{D}_{1\times 2}^\sharp \to \mathcal{D}_{1\times 2}^\sharp$$
$$\rho(\mathcal{X}_1^\sharp, \mathcal{X}_2^\sharp) \stackrel{\text{def}}{=} (\alpha_1(\gamma_1(\mathcal{X}_1^\sharp) \cap \gamma_2(\mathcal{X}_2^\sharp)), \alpha_2(\gamma_1(\mathcal{X}_1^\sharp) \cap \gamma_2(\mathcal{X}_2^\sharp)))$$

ρ propagates information between domains.

$$\rho(\mathtt{X} \in [11, 12], \mathtt{X} \equiv 1[2])) = (\mathtt{X} \in [11, 11], \mathtt{X} \equiv 1[2])$$

$$\rho(\mathtt{X} \in [12, 12], \mathtt{X} \equiv 1[2])) = \bot^\sharp$$

Fully-reduced product

Definition:

Given the Galois connections (α_1, γ_1) and (α_2, γ_2) on $\mathcal{D}_1^\sharp$ and $\mathcal{D}_2^\sharp$ we define the reduction operator ρ as:

$$\rho : \mathcal{D}_{1\times 2}^\sharp \to \mathcal{D}_{1\times 2}^\sharp$$
$$\rho(\mathcal{X}_1^\sharp, \mathcal{X}_2^\sharp) \stackrel{\text{def}}{=} (\alpha_1(\gamma_1(\mathcal{X}_1^\sharp) \cap \gamma_2(\mathcal{X}_2^\sharp)), \alpha_2(\gamma_1(\mathcal{X}_1^\sharp) \cap \gamma_2(\mathcal{X}_2^\sharp)))$$

ρ propagates information between domains.

Application:

We can reduce the result of each abstract operator, except ∇:

- $(\mathcal{X}_1^\sharp, \mathcal{X}_2^\sharp) \cup_{1\times 2}^\sharp (\mathcal{Y}_1^\sharp, \mathcal{Y}_2^\sharp) \stackrel{\text{def}}{=} \rho(\mathcal{X}_1^\sharp \cup_1^\sharp \mathcal{Y}_1^\sharp, \mathcal{X}_2^\sharp \cup_2^\sharp \mathcal{Y}_2^\sharp)$,
- $\mathsf{C}^\sharp[\![c]\!]_{1\times 2}(\mathcal{X}_1^\sharp, \mathcal{X}_2^\sharp) \stackrel{\text{def}}{=} \rho(\mathsf{C}^\sharp[\![c]\!]_1(\mathcal{X}_1^\sharp), \mathsf{C}^\sharp[\![c]\!]_2(\mathcal{X}_2^\sharp))$.

We refrain from reducing after a widening ∇,
this may threaten the convergence

Fully-reduced product example

Reduction example: between the interval and congruence domains:

Noting: $a' \stackrel{\text{def}}{=} \min \{ x \geq a \mid x \equiv d\ [c] \}$
$b' \stackrel{\text{def}}{=} \max \{ x \leq b \mid x \equiv d\ [c] \}$

We get:

$$\rho_b([a,b], c\mathbb{Z}+d) \stackrel{\text{def}}{=} \begin{cases} (\bot_b^\sharp, \bot_b^\sharp) & \text{if } a' > b' \\ ([a',a'], 0\mathbb{Z}+a') & \text{if } a' = b' \\ ([a',b'], c\mathbb{Z}+d) & \text{if } a' < b' \end{cases}$$

extended point-wisely to ρ on $\mathcal{D}^\sharp$.

Application:

- $\rho_b([10,11], 2\mathbb{Z}+1) = ([11,11], 0\mathbb{Z}+11)$
  (proves that the branch is never taken on our example)
- $\rho_b([1,3], 4\mathbb{Z}) = (\bot_b^\sharp, \bot_b^\sharp)$

Shortcomings of non-relational domains

Accumulated loss of precision

Non-relational domains cannot represent variable relationships

Rate limiter

```
Y:=0; while • 1=1 do
  X:=[-128,128]; D:=[0,16];
  S:=Y; Y:=X; R:=X-S;
  if R<=-D then Y:=S-D fi;
  if R>=D then Y:=S+D fi
done
```

X: input signal
Y: output signal
S: last output
R: delta Y−S
D: max. allowed for |R|

Iterations in the interval domain (without widening):

$\mathcal{X}_\bullet^{\sharp 0}$	$\mathcal{X}_\bullet^{\sharp 1}$	$\mathcal{X}_\bullet^{\sharp 2}$	...	$\mathcal{X}_\bullet^{\sharp n}$
$\mathtt{Y} = 0$	$\lvert\mathtt{Y}\rvert \leq 144$	$\lvert\mathtt{Y}\rvert \leq 160$	...	$\lvert\mathtt{Y}\rvert \leq 128 + 16n$

In fact, $\mathtt{Y} \in [-128, 128]$ always holds.

To prove that, e.g. $\mathtt{Y} \geq -128$, we must be able to:

- represent the properties $\mathtt{R} = \mathtt{X} - \mathtt{S}$ and $\mathtt{R} \leq -\mathtt{D}$
- combine them to deduce $\mathtt{S} - \mathtt{X} \geq \mathtt{D}$, and then $\mathtt{Y} = \mathtt{S} - \mathtt{D} \geq \mathtt{X}$

The need for relational loop invariants

To prove some invariant after the end of a loop,
we often need to find a loop invariant of a more complex form

relational loop invariant

```
X:=0; I:=1;
while • I<5000 do
  if [0,1]=1 then X:=X+1 else X:=X-1 fi;
  I:=I+1
done ⧫
```

A non-relational analysis finds at ⧫ that $\mathtt{I} = 5000$ and $\mathtt{X} \in \mathbb{Z}$

The best invariant is: $(\mathtt{I} = 5000) \wedge (\mathtt{X} \in [-4999, 4999]) \wedge (\mathtt{X} \equiv 1\ [2])$

To find this non-relational invariant, we must find a relational loop invariant at •: $(-\mathtt{I} < \mathtt{X} < \mathtt{I}) \wedge (\mathtt{X} + \mathtt{I} \equiv 1\ [2]) \wedge (\mathtt{I} \in [1, 5000])$,
and apply the loop exit condition $\mathrm{C}^\sharp[\![\ \mathtt{I} >= 5000\ ]\!]$

Modular analysis

store the maximum of X,Y,0 into Z'

```
max(X,Y,Z)
  X':=X; Y':=Y; Z':=Z;
  Z':=X';
  if Y' > Z' then Z':=Y';
  if Z' < 0 then Z':=0;
```

$(Z' \geq X \wedge Z' \geq Y \wedge Z' \geq 0 \wedge X' = X \wedge Y' = Y)$

Modular analysis:

- analyze a procedure once (procedure summary)
- reuse the summary at each call site (instantiation)
  $\Longrightarrow$ improved efficiency
- infer a relation between input X,Y,Z and output X′,Y′,Z′ values
  $\mathcal{P}((\mathbb{V} \to \mathbb{R}) \times (\mathbb{V}' \to \mathbb{R})) \equiv \mathcal{P}((\mathbb{V} \cup \mathbb{V}') \to \mathbb{R})$
- requires inferring relational information

[Anco10], [Jean09]

Linear equality domain

The affine equality domain

Here $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$.

We look for invariants of the form:

$$\bigwedge_j \left(\sum_{i=1}^{n} \alpha_{ij} \mathtt{V}_i = \beta_j\right),\ \alpha_{ij}, \beta_j \in \mathbb{I}$$

where all the α_{ij} and β_j are inferred automatically.

With 3 variables x, y, z

$(x = 3, y = -2, z = 0)$	point (dim 0)
$x + 2y - 3z = 1$	line (dim 1)
$x + 2z = 3$	plane (dim 2)
$y = 3$	plane (dim 2)

The affine equality domain

Here $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$.

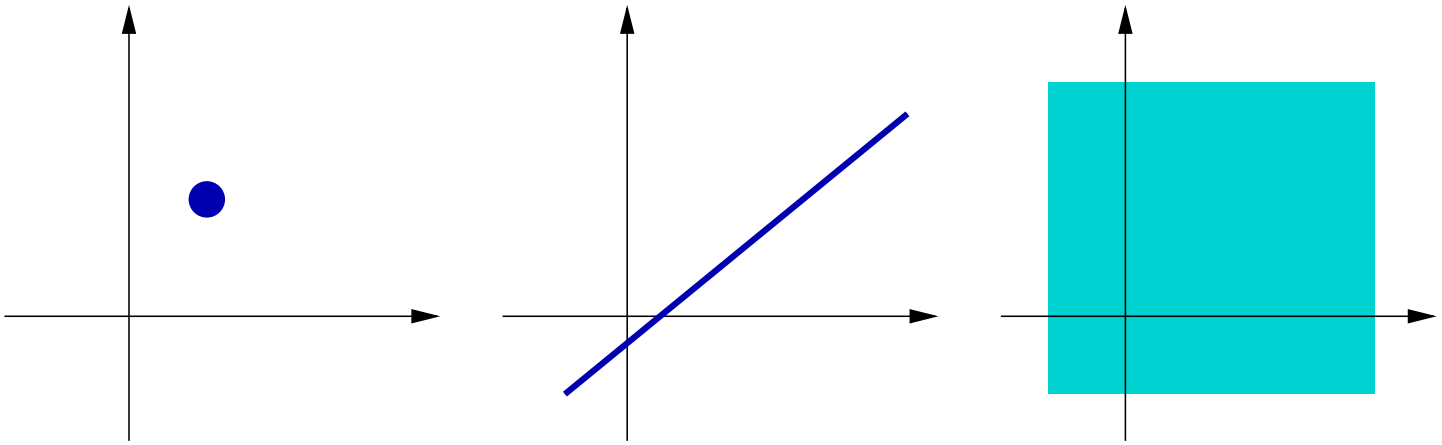
We look for invariants of the form:

$$\bigwedge_j \left(\sum_{i=1}^{n} \alpha_{ij} \mathtt{V}_i = \beta_j\right),\ \alpha_{ij}, \beta_j \in \mathbb{I}$$

where all the α_{ij} and β_j are inferred automatically.

We use a domain of affine spaces proposed by [Karr76]:

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\text{ affine subspaces of } \mathbb{V} \to \mathbb{I} \}$$



Galois connection

Galois connection: (actually, a Galois insertion)

between arbitrary subsets and affine subsets

$$(\mathcal{P}(\mathbb{I}^n), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathit{Aff}(\mathbb{I}^n), \subseteq)$$

- $\gamma(X) \overset{\text{def}}{=} X$ (identity)
- $\alpha(X) \overset{\text{def}}{=}$ smallest affine subset containing X

  $\mathit{Aff}(\mathbb{I}^n)$ is closed under arbitrary intersections, so we have:

  $$\alpha(X) = \cap \{ Y \in \mathit{Aff}(\mathbb{I}^n) \mid X \subseteq Y \}$$

  $\mathit{Aff}(\mathbb{I}^n)$ contains every point in $\mathbb{I}^n$

  we can also construct $\alpha(X)$ by abstract union:

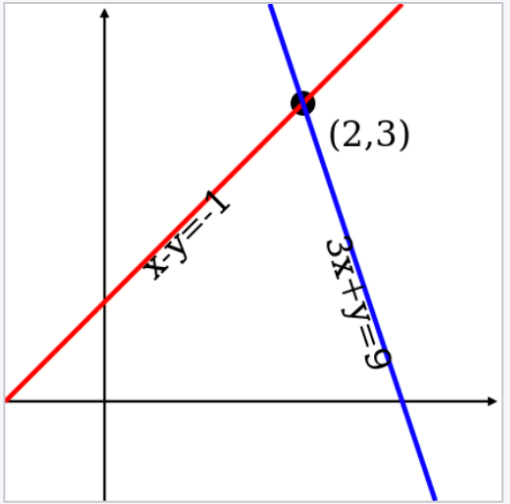
  $$\alpha(X) = \cup^\sharp \{ \{x\} \mid x \in X \}$$

Solution set [edit]

A **solution** of a linear system is an assignment of values to the variables x_1, x_2, ..., x_n such that each of the equations is satisfied. The set of all possible solutions is called the **solution set**.

A linear system may behave in any one of three possible ways:

1. The system has *infinitely many solutions*.
2. The system has a single *unique solution*.
3. The system has *no solution*.



The solution set for the equations $x - y = -1$ and $3x + y = 9$ is the single point (2, 3).

Geometric interpretation [edit]

For a system involving two variables (x and y), each linear equation determines a line on the xy-plane. Because a solution to a linear system must satisfy all of the equations, the solution set is the intersection of these lines, and is hence either a line, a single point, or the empty set.

For three variables, each linear equation determines a plane in three-dimensional space, and the solution set is the intersection of these planes. Thus the solution set may be a plane, a line, a single point, or the empty set. For example, as three parallel planes do not have a common point, the solution set of their equations is empty; the solution set of the equations of three planes intersecting at a point is single point; if three planes pass through two points, their equations have at least two common solutions; in fact the solution set is infinite and consists in all the line passing through these points.[6]

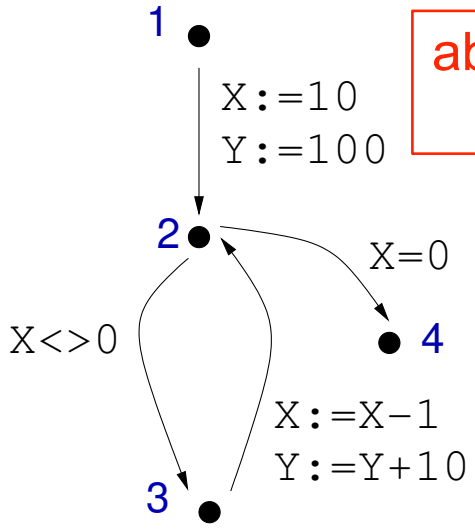
For n variables, each linear equation determines a hyperplane in n-dimensional space. The solution set is the intersection of these hyperplanes, and is a flat, which may have any dimension lower than n.

Analysis example

No infinite increasing chain: we can iterate without widening.

Forward analysis example:

```
[1]X:=10; Y:=100;
 while [2]X<>0 do[3]
   X:=X-1;
   Y:=Y+10
 done[4]
```



abstract operations/tests need cubic time

Result

```
Annotated program after forward analysis
var X : int, Y : int;
begin
  /* (L4 C5) top */
  X = 10; /* (L5 C5) [|X-10=0|] */
  Y = 100; /* (L5 C12) [|10X+Y-200=0|] */
  while not X == 0 do
    /* (L6 C20) [|10X+Y-200=0|] */
    X = X - 1; /* (L7 C6) [|10X+Y-190=0|] */
    Y = Y + 10; /* (L8 C7) [|10X+Y-200=0|] */
  done; /* (L9 C5) [|Y-200=0; X=0|] */
end
```

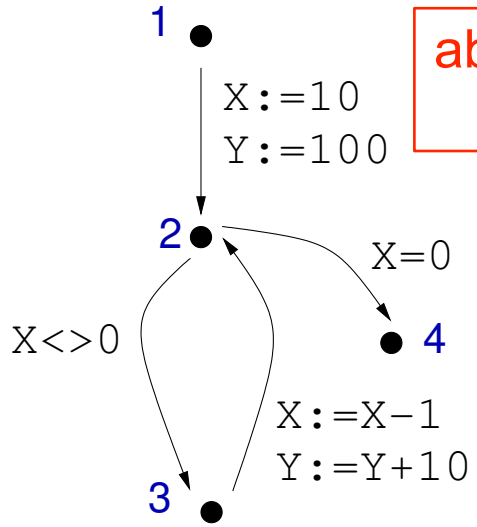
Analysis example

No infinite increasing chain: we can iterate without widening.

Forward analysis example:

abstract operations/tests need cubic time

```
[1]X:=10; Y:=100;
 while [2]X<>0 do[3]
   X:=X-1;
   Y:=Y+10
 done[4]
```



ℓ	$\mathcal{X}_\ell^{\sharp 0}$	$\mathcal{X}_\ell^{\sharp 1}$	$\mathcal{X}_\ell^{\sharp 2}$	$\mathcal{X}_\ell^{\sharp 3}$	$\mathcal{X}_\ell^{\sharp 4}$
1	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$	$\top^\sharp$
2	$\bot^\sharp$	$(10, 100)$	$(10, 100)$	$10\mathtt{X} + \mathtt{Y} = 200$	$10\mathtt{X} + \mathtt{Y} = 200$
3	$\bot^\sharp$	$\bot^\sharp$	$(10, 100)$	$(10, 100)$	$10\mathtt{X} + \mathtt{Y} = 200$
4	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	$\bot^\sharp$	$(0, 200)$

Note in particular:

$$\mathcal{X}_2^{\sharp 3} = \{(10, 100)\} \cup^\sharp \{(9, 110)\} = \{\ (\mathtt{X}, \mathtt{Y}) \mid 10\mathtt{X} + \mathtt{Y} = 200\ \}$$

line including (10,100) and (9,110)

Polyhedron domain

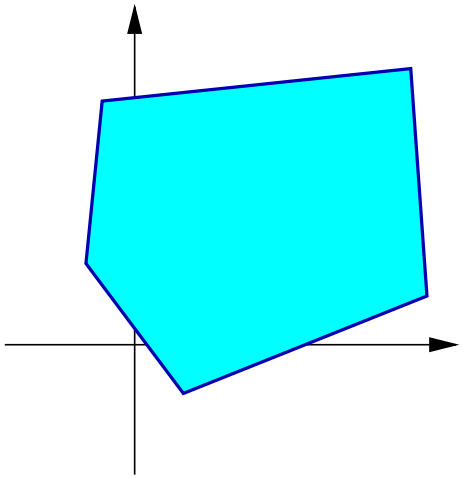
The polyhedron domain

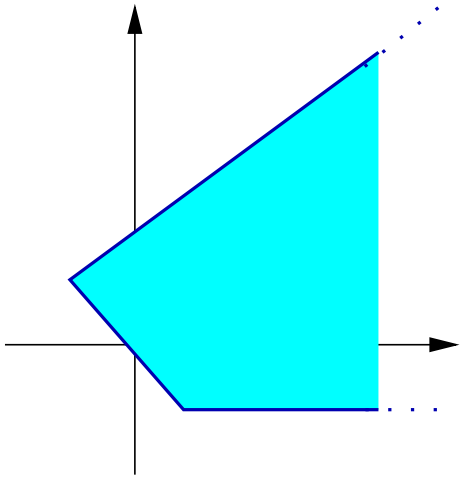
Here again, $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$.

We look for invariants of the form: $\bigwedge_j \left(\sum_{i=1}^{n} \alpha_{ij} \mathtt{V}_i \geq \beta_j \right)$.

We use the polyhedron domain proposed by [Cous78]:

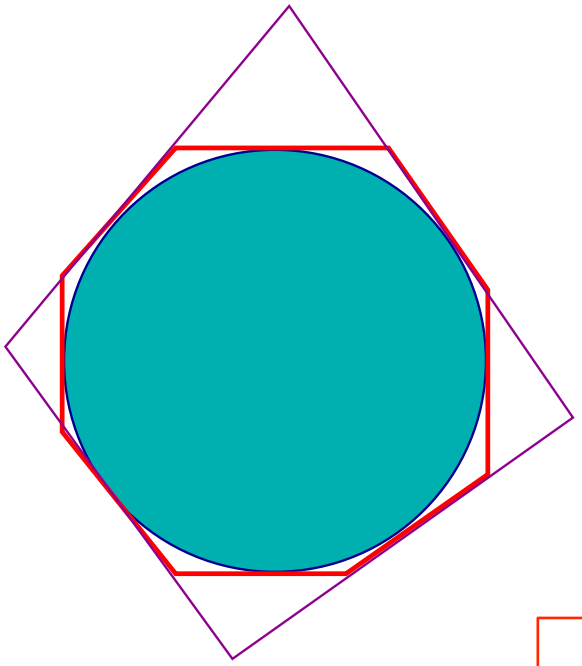
$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\text{closed convex polyhedra of } \mathbb{V} \to \mathbb{I}\}$$





Note: polyhedra need not be bounded ($\neq$ polytopes).

Polyhedra representations



abstract operations
need **exponential** time

- No best abstraction α ←
  (e.g., a disc has infinitely many polyhedral over-approximations, but no best one)
- No memory bound on the (complex) representations ←

Example analysis

Example program

```
X:=2; I:=0;
while • I<10 do
  if [0,1]=0 then X:=X+2 else X:=X-3 fi;
  I:=I+1
done♦
```

Best loop invariant?

$\mathtt{I} \in [0,10] \wedge \mathtt{X} \in [-28,22]$ is not the best

$\mathtt{I} \in [0,10] \wedge \mathtt{2-3I} \leq \mathtt{X} \leq \mathtt{2I+2}$ is more precise

(yet not the concrete one)

Example analysis

Example program

```
X:=2; I:=0;
while • I<10 do
  if [0,1]=0 then X:=X+2 else X:=X-3 fi;
  I:=I+1
done♦
```

We use a finite number (one) of intersections $\cap^\sharp$ as narrowing.

Iterations with widening and narrowing at • give:

$$\mathcal{X}_\bullet^{\sharp 1} = \{\mathtt{X}=2, \mathtt{I}=0\}$$

$$\begin{aligned}\mathcal{X}_\bullet^{\sharp 2} &= \{\mathtt{X}=2, \mathtt{I}=0\} \mathbin{\triangledown} (\{\mathtt{X}=2, \mathtt{I}=0\} \cup^\sharp \{\mathtt{X}\in[-1,4],\ \mathtt{I}=1\}) \\ &= \{\mathtt{X}=2, \mathtt{I}=0\} \mathbin{\triangledown} \{\mathtt{I}\in[0,1],\ 2-3\mathtt{I}\leq \mathtt{X}\leq 2\mathtt{I}+2\} \\ &= \{\mathtt{I}\geq 0,\ 2-3\mathtt{I}\leq \mathtt{X}\leq 2\mathtt{I}+2\}\end{aligned}$$

$$\begin{aligned}\mathcal{X}_\bullet^{\sharp 3} &= \{\mathtt{I}\geq 0,\ 2-3\mathtt{I}\leq \mathtt{X}\leq 2\mathtt{I}+2\} \cap^\sharp \\ &\quad (\{\mathtt{X}=2, \mathtt{I}=0\} \cup^\sharp \{\mathtt{I}\in[1,10],\ 2-3\mathtt{I}\leq \mathtt{X}\leq 2\mathtt{I}+2\}) \\ &= \{\mathtt{I}\in[0,10],\ 2-3\mathtt{I}\leq \mathtt{X}\leq 2\mathtt{I}+2\}\end{aligned}$$

At ♦ we find eventually: $\mathtt{I}=10 \wedge \mathtt{X}\in[-28,22]$.

sophisticated widening operator

lub of polyhedra

abstract operations need **exponential** time

Weakly relational domains

Zone domain

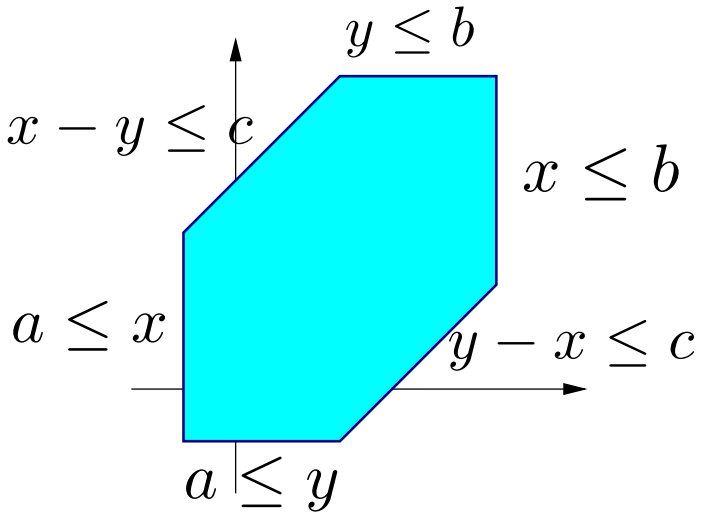
The zone domain

Here, $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$.

We look for invariants of the form:

$$\bigwedge \mathtt{V}_i - \mathtt{V}_j \leq c \text{ or } \pm \mathtt{V}_i \leq c, \quad c \in \mathbb{I}$$

A subset of $\mathbb{I}^n$ bounded by such constraints is called a **zone**.

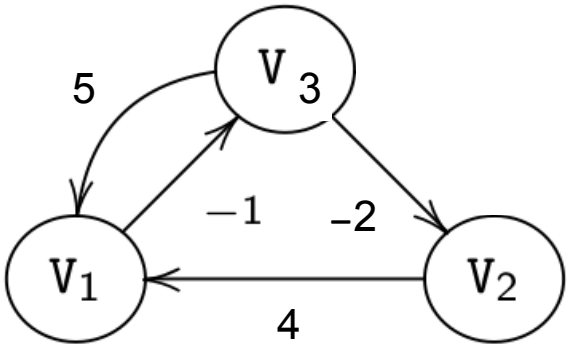


Machine representation

A potential constraint has the form: $\mathtt{V}_j - \mathtt{V}_i \leq c$.

Potential graph: directed, weighted graph $\mathcal{G}$

- nodes are labelled with variables in $\mathbb{V}$,
- we add an arc with weight c from $\mathtt{V}_i$ to $\mathtt{V}_j$ for each constraint $\mathtt{V}_j - \mathtt{V}_i \leq c$.



$\mathtt{V}_1 - \mathtt{V}_3 \leq 5$

$\mathtt{V}_3 - \mathtt{V}_1 \leq -1$

$\mathtt{V}_2 - \mathtt{V}_3 \leq -2$

$\mathtt{V}_1 - \mathtt{V}_2 \leq 4$

Machine representation

A potential constraint has the form: $\mathtt{V}_j - \mathtt{V}_i \leq c$.

Potential graph: directed, weighted graph $\mathcal{G}$

- nodes are labelled with variables in $\mathbb{V}$,
- we add an arc with weight c from $\mathtt{V}_i$ to $\mathtt{V}_j$ for each constraint $\mathtt{V}_j - \mathtt{V}_i \leq c$.

Difference Bound Matrix (DBM)

Adjacency matrix **m** of $\mathcal{G}$:

- **m** is square, with size $n \times n$, and elements in $\mathbb{I} \cup \{+\infty\}$,
- $m_{ij} = c < +\infty$ denotes the constraint $\mathtt{V}_j - \mathtt{V}_i \leq c$,
- $m_{ij} = +\infty$ if there is no upper bound on $\mathtt{V}_j - \mathtt{V}_i$.

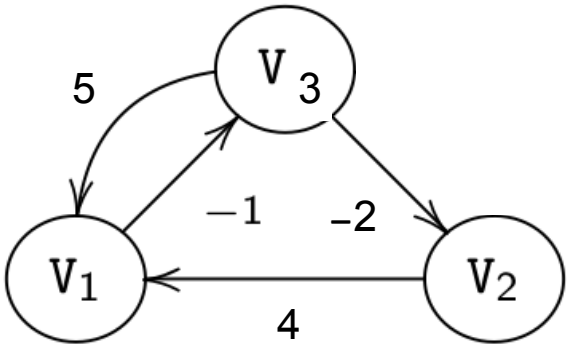
Machine representation

A potential constraint has the form: $\mathtt{V}_j - \mathtt{V}_i \leq c$.

Potential graph: directed, weighted graph $\mathcal{G}$

- nodes are labelled with variables in $\mathbb{V}$,
- we add an arc with weight c from $\mathtt{V}_i$ to $\mathtt{V}_j$ for each constraint $\mathtt{V}_j - \mathtt{V}_i \leq c$.

	$\mathtt{V}_1$	$\mathtt{V}_2$	$\mathtt{V}_3$
$\mathtt{V}_1$	∞	∞	-1
$\mathtt{V}_2$	4	∞	-2
$\mathtt{V}_3$	5	∞	∞



$\mathtt{V}_1 - \mathtt{V}_3 \leq 5$

$\mathtt{V}_3 - \mathtt{V}_1 \leq -1$

$\mathtt{V}_2 - \mathtt{V}_3 \leq -2$

$\mathtt{V}_1 - \mathtt{V}_2 \leq 4$

Machine representation

A potential constraint has the form: $\mathtt{V}_j - \mathtt{V}_i \leq c$.

Potential graph: directed, weighted graph $\mathcal{G}$

- nodes are labelled with variables in $\mathbb{V}$,
- we add an arc with weight c from $\mathtt{V}_i$ to $\mathtt{V}_j$ for each constraint $\mathtt{V}_j - \mathtt{V}_i \leq c$.

Difference Bound Matrix (DBM)

Adjacency matrix $\mathbf{m}$ of $\mathcal{G}$:

- $\mathbf{m}$ is square, with size $n \times n$, and elements in $\mathbb{I} \cup \{+\infty\}$,
- $m_{ij} = c < +\infty$ denotes the constraint $\mathtt{V}_j - \mathtt{V}_i \leq c$,
- $m_{ij} = +\infty$ if there is no upper bound on $\mathtt{V}_j - \mathtt{V}_i$.

Concretization:

$$\gamma(\mathbf{m}) \stackrel{\text{def}}{=} \{ (v_1, \ldots, v_n) \in \mathbb{I}^n \mid \forall i, j,\ v_j - v_i \leq m_{ij} \}.$$

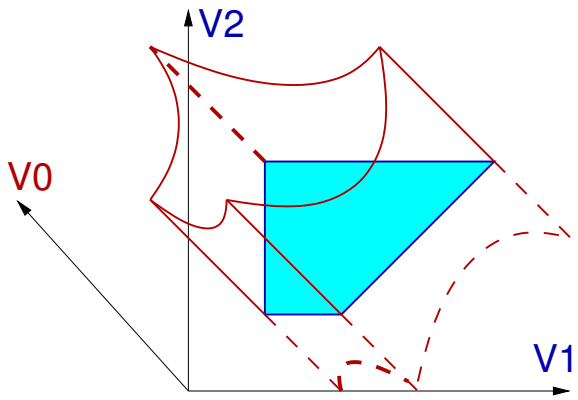
Machine representation (cont.)

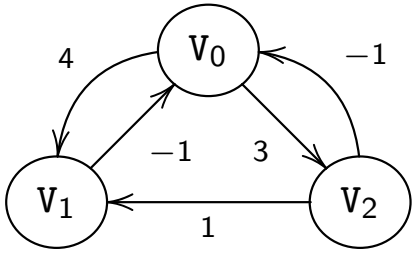
Unary constraints add a constant null variable $\mathtt{V}_0$.

- $\mathbf{m}$ has size $(n+1) \times (n+1)$;
- $\mathtt{V}_i \leq c$ is denoted as $\mathtt{V}_i - \mathtt{V}_0 \leq c$, i.e., $m_{i0} = c$;
- $\mathtt{V}_i \geq c$ is denoted as $\mathtt{V}_0 - \mathtt{V}_i \leq -c$, i.e., $m_{0i} = -c$;
- γ is now: $\gamma_0(\mathbf{m}) \stackrel{\text{def}}{=} \{ (v_1, \ldots, v_n) \mid (0, v_1, \ldots, v_n) \in \gamma(\mathbf{m}) \}$.

Example:

$$\mathtt{V}_1 \leq 4,\ \mathtt{V}_2 \leq 3,$$
$$\mathtt{V}_1 \geq 1,\ \mathtt{V}_2 \geq 1,$$
$$\mathtt{V}_1 - \mathtt{V}_2 \leq 1$$





	$\mathtt{V}_0$	$\mathtt{V}_1$	$\mathtt{V}_2$
$\mathtt{V}_0$	$+\infty$	4	3
$\mathtt{V}_1$	-1	$+\infty$	$+\infty$
$\mathtt{V}_2$	-1	1	$+\infty$

The DBM lattice

$\mathcal{D}^\sharp$ contains all DBMs, plus $\bot^\sharp$.

$\leq$ on $\mathbb{I} \cup \{+\infty\}$ is extended point-wisely.

If $\mathbf{m}, \mathbf{n} \neq \bot^\sharp$:

$$\begin{array}{rcl}
\mathbf{m} \subseteq^\sharp \mathbf{n} & \overset{\text{def}}{\iff} & \forall i,j,\ m_{ij} \leq n_{ij} \\
\mathbf{m} =^\sharp \mathbf{n} & \overset{\text{def}}{\iff} & \forall i,j,\ m_{ij} = n_{ij} \\
\left[\mathbf{m} \cap^\sharp \mathbf{n}\right]_{ij} & \overset{\text{def}}{=} & \min(m_{ij}, n_{ij}) \\
\left[\mathbf{m} \cup^\sharp \mathbf{n}\right]_{ij} & \overset{\text{def}}{=} & \max(m_{ij}, n_{ij}) \\
\left[\top^\sharp\right]_{ij} & \overset{\text{def}}{=} & +\infty
\end{array}$$

$(\mathcal{D}^\sharp, \subseteq^\sharp, \cup^\sharp, \cap^\sharp, \bot^\sharp, \top^\sharp)$ is a lattice.

Remarks:

- $\mathcal{D}^\sharp$ is complete if $\leq$ is ($\mathbb{I} = \mathbb{R}$ or $\mathbb{Z}$, but not $\mathbb{Q}$),
- $\mathbf{m} \subseteq^\sharp \mathbf{n} \Longrightarrow \gamma_0(\mathbf{m}) \subseteq \gamma_0(\mathbf{n})$, but not the converse,
- $\mathbf{m} =^\sharp \mathbf{n} \Longrightarrow \gamma_0(\mathbf{m}) = \gamma_0(\mathbf{n})$, but not the converse.

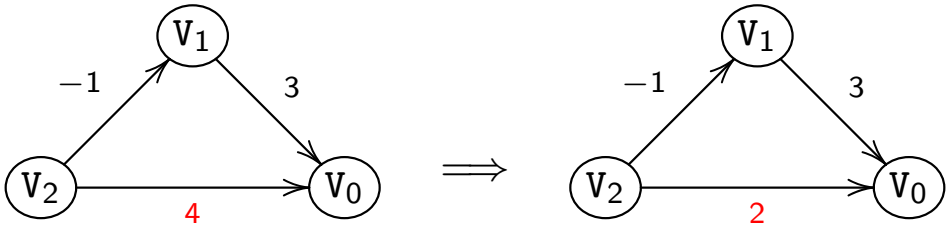
γ_0 is not injective

Normal form, equality and inclusion testing

Issue: how can we compare $\gamma_0(\mathbf{m})$ and $\gamma_0(\mathbf{n})$?

Idea: find a normal form by propagating/tightening constraints.

$$\begin{cases} V_0 - V_1 \leq 3 \\ V_1 - V_2 \leq -1 \\ V_0 - V_2 \leq 4 \end{cases} \qquad \begin{cases} V_0 - V_1 \leq 3 \\ V_1 - V_2 \leq -1 \\ V_0 - V_2 \leq 2 \end{cases}$$



Definition: shortest-path closure $\mathbf{m}^*$

$$\forall\ i,j \text{ compute } \quad m^*_{ij} \overset{\text{def}}{=} \min_{\substack{N \\ \langle i = i_1, \ldots, i_N = j \rangle}} \sum_{k=1}^{N-1} m_{i_k\, i_{k+1}}$$

Exists only when $\mathbf{m}$ has no cycle with strictly negative weight.

Floyd–Warshall algorithm

From Wikipedia, the free encyclopedia

> *"Floyd's algorithm" redirects here. For cycle detection, see Floyd's cycle-finding algorithm. For computer graphics, see Floyd–Steinberg dithering.*

In computer science, the **Floyd–Warshall algorithm** is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles).[1][2] A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between *all* pairs of vertices. Although it does not return details of the paths themselves, it is possible to reconstruct the paths with simple modifications to the algorithm. Versions of the algorithm can also be used for finding the transitive closure of a relation R, or (in connection with the Schulze voting system) widest paths between all pairs of vertices in a weighted graph.

Floyd–Warshall algorithm

Class	All-pairs shortest path problem (for weighted graphs)
Data structure	Graph
Worst-case performance	$\Theta(\|V\|^3)$
Best-case performance	$\Theta(\|V\|^3)$
Average performance	$\Theta(\|V\|^3)$
Worst-case space complexity	$\Theta(\|V\|^2)$

Abstract operations, abstract tests and widening all rely on shortest-path closure **m***

Floyd-Warshall algorithm $\Theta(|Var|^3)$

Octagon domain

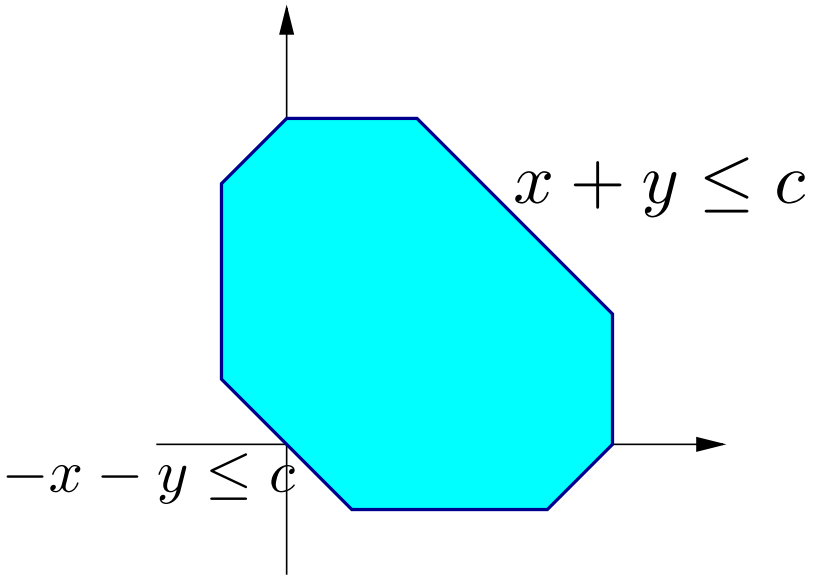
The octagon domain

Now, $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$.

We look for invariants of the form: $\bigwedge \pm \mathtt{V}_i \pm \mathtt{V}_j \leq c, \quad c \in \mathbb{I}$

A subset of $\mathbb{I}^n$ defined by such constraints is called an octagon.

It is a generalisation of zones (more symmetric).



[Mine01b]

Machine representation

Idea: use a variable change to get back to potential constraints.

Let $\mathbb{V}' \stackrel{\text{def}}{=} \{\mathtt{V}'_1, \ldots, \mathtt{V}'_{2n}\}$.

$$\mathtt{V_i} \mapsto \mathtt{V}'_{2\mathtt{i}-1} \qquad -\mathtt{V_i} \mapsto \mathtt{V}'_{2\mathtt{i}}$$

the constraint:		is encoded as:				
$\mathtt{V}_i - \mathtt{V}_j \leq c$	$(i \neq j)$	$\mathtt{V}'_{2i-1} - \mathtt{V}'_{2j-1} \leq$	c	and	$\mathtt{V}'_{2j} - \mathtt{V}'_{2i} \leq c$	
$\mathtt{V}_i + \mathtt{V}_j \leq c$	$(i \neq j)$	$\mathtt{V}'_{2i-1} - \mathtt{V}'_{2j} \leq$	c	and	$\mathtt{V}'_{2j-1} - \mathtt{V}'_{2i} \leq c$	
$-\mathtt{V}_i - \mathtt{V}_j \leq c$	$(i \neq j)$	$\mathtt{V}'_{2j} - \mathtt{V}'_{2i-1} \leq$	c	and	$\mathtt{V}'_{2i} - \mathtt{V}'_{2j-1} \leq c$	
$\mathtt{V}_i \leq c$		$\mathtt{V}'_{2i-1} - \mathtt{V}'_{2i} \leq$	$2c$			
$\mathtt{V}_i \geq c$		$\mathtt{V}'_{2i} - \mathtt{V}'_{2i-1} \leq$	$-2c$			

$$(\mathtt{V}_i) - (\mathtt{V}_j) \leq c \qquad (-\mathtt{V}_j) - (-\mathtt{V}_i) \leq c$$

Machine representation

Idea: use a variable change to get back to potential constraints.

Let $\mathbb{V}' \stackrel{\text{def}}{=} \{\mathtt{V}'_1, \ldots, \mathtt{V}'_{2n}\}$. $\quad \boxed{\mathtt{V_i} \mapsto \mathtt{V}'_{2\mathtt{i}-1} \quad -\mathtt{V_i} \mapsto \mathtt{V}'_{2\mathtt{i}}}$

the constraint:		is encoded as:				
$\mathtt{V}_i - \mathtt{V}_j \leq c$	$(i \neq j)$	$\mathtt{V}'_{2i-1} - \mathtt{V}'_{2j-1} \leq$	c	and	$\mathtt{V}'_{2j} - \mathtt{V}'_{2i} \leq c$	
$\mathtt{V}_i + \mathtt{V}_j \leq c$	$(i \neq j)$	$\mathtt{V}'_{2i-1} - \mathtt{V}'_{2j} \leq$	c	and	$\mathtt{V}'_{2j-1} - \mathtt{V}'_{2i} \leq c$	
$-\mathtt{V}_i - \mathtt{V}_j \leq c$	$(i \neq j)$	$\mathtt{V}'_{2j} - \mathtt{V}'_{2i-1} \leq$	c	and	$\mathtt{V}'_{2i} - \mathtt{V}'_{2j-1} \leq c$	
$\mathtt{V}_i \leq c$		$\mathtt{V}'_{2i-1} - \mathtt{V}'_{2i} \leq$	$2c$			
$\mathtt{V}_i \geq c$		$\mathtt{V}'_{2i} - \mathtt{V}'_{2i-1} \leq$	$-2c$			

We use a matrix $\mathbf{m}$ of size $(2n) \times (2n)$ with elements in $\mathbb{I} \cup \{+\infty\}$ and $\gamma_\pm(\mathbf{m}) \stackrel{\text{def}}{=} \{ (v_1, \ldots, v_n) \mid (v_1, -v_1, \ldots, v_n, -v_n) \in \gamma(\mathbf{m}) \}$.

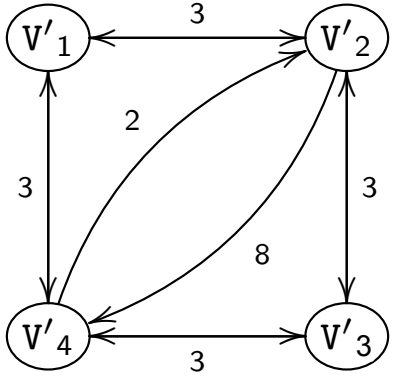
Note:
Two distinct $\mathbf{m}$ elements can represent the same constraint on $\mathbb{V}$.

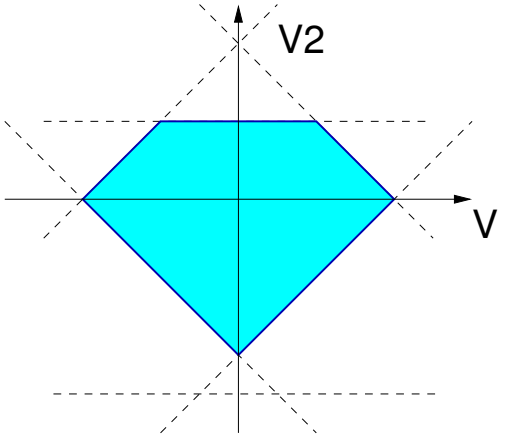
To avoid this, we impose that $\forall i, j,\ m_{ij} = m_{\bar{\jmath}\,\bar{\imath}}$ where $\bar{\imath} = i \oplus 1$.

Machine representation (cont.)

Example:

$$\begin{cases} V_1 + V_2 \leq 3 \\ V_2 - V_1 \leq 3 \\ V_1 - V_2 \leq 3 \\ -V_1 - V_2 \leq -3 \\ 2V_2 \leq 2 \\ -2V_2 \leq 8 \end{cases}$$





Lattice

Constructed by point-wise extension of $\leq$ on $\mathbb{I} \cup \{+\infty\}$.

```
var X: int, Y: int;
begin
 X=10;
 Y=0;
 while (X>=0) do
   X = X-1;
   if brandom then Y = Y+1; endif;
 done;
end
```

Loop invariant:

$-1 \le X \le 10,\ 0 \le Y \le 11$

$-12 \le X - Y \le 10$

$-1 \le X + Y \le 10$

```
Annotated program after forward analysis
var X : int, Y : int;
begin
  /* (L2 C5) top */
  X = 10; /* (L3 C6) [|X-10>=0; -X+10>=0|] */
  Y = 0; /* (L4 C5)
            [|X+1>=0; -X+10>=0; -X+Y+10>=0; X+Y+1>=0; Y>=0; -X-Y+10>=0;
              X-Y+12>=0; -Y+11>=0|] */
  while X >= 0 do
    /* (L5 C16)
       [|X>=0; -X+10>=0; -X+Y+10>=0; X+Y>=0; Y>=0; -X-Y+10>=0; X-Y+10>=0;
         -Y+10>=0|] */
    X = X - 1; /* (L6 C11)
                  [|X+1>=0; -X+9>=0; -X+Y+9>=0; X+Y+1>=0; Y>=0; -X-Y+9>=0;
                    X-Y+11>=0; -Y+10>=0|] */
    if brandom then
      /* (L7 C18)
         [|X+1>=0; -X+9>=0; -X+Y+9>=0; X+Y+1>=0; Y>=0; -X-Y+9>=0; X-Y+11>=0;
           -Y+10>=0|] */
      Y = Y + 1; /* (L7 C27)
                    [|X+1>=0; -X+9>=0; -X+Y+8>=0; X+Y>=0; Y-1>=0; -X-Y+10>=0;
                      X-Y+12>=0; -Y+11>=0|] */
    endif; /* (L7 C34)
              [|X+1>=0; -X+9>=0; -X+Y+9>=0; X+Y+1>=0; Y>=0; -X-Y+10>=0;
                X-Y+12>=0; -Y+11>=0|] */
  done; /* (L8 C6)
           [|X+1>=0; -X-1>=0; -X+Y-1>=0; X+Y+1>=0; Y>=0; -X-Y+10>=0;
             X-Y+12>=0; -Y+11>=0|] */
end
```

Summary

Summary of numerical domains

domain	non-relational	linear equalities	polyhedra	octagons
invariants	$\mathtt{V} \in \mathcal{B}_b^\sharp$	$\sum_i \alpha_i \mathtt{V}_i = \beta$	$\sum_i \alpha_i \mathtt{V}_i \leq \beta$	$\pm\mathtt{V}_i \pm \mathtt{V}_j \leq c$
memory cost	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(2^n)$	$\mathcal{O}(n^2)$
time cost	$\mathcal{O}(n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(2^n)$	$\mathcal{O}(n^3)$

The Interproc Analyzer

This is a web interface to the Interproc analyzer connected to the APRON Abstract Domain Library and the Fixpoint Solver Library, whose goal is to demonstrate the features of the APRON library and, to a less extent, of the Analyzer fixpoint engine, in the static analysis field.

There are two compiled versions: interprocweb, in which all the abstract domains use underlying multiprecision integer/rational numbers, and interprocwebf, in which box and octagon domains use underlying floating-point numbers in safe way.

This is the **Interproc** version

Arguments

Please type a program, upload a file from your hard-drive, or choose one the provided examples:

Choose File no file selected

user-supplied

/* type your program here ! */

Choose an Abstract Domain:
box
box with policy iteration
octagon
✓ convex polyhedra (polka)
convex polyhedra (PPL)
strict convex polyhedra (polka)
strict convex polyhedra (PPL)
linear equalities (polka)
linear congruences (PPL)
convex polyhedra + linear congruences

Numerical Abstract Domain

Kind of Analysis: f (s analysis)

Iterations/Widening options:
guided iterations 1 wid
0 debugging level (0 to 6)

Hit the OK button to proceed: OK ! Reset

it can be used in homework

``Simple'' language syntax

Here are some program examples: incr ackerman fact numerical numerical2 maccarthy91 heapsort symmetricalstairs

Results

In order not to flood our web-server, analysis computation time is limited to 1min in this demonstration. Also note that result files are temporary files stored on our server that have a very short life-time.

The analysis computes an invariant at each program point.

Exams

- Agree on a date with the teacher (contact him at least 1 week in advance), any date of 2022 is possible. You can discuss HW1 and HW2 separately in two different dates, or together in a single date.

- Please subscribe the Uniweb list for the exam of January/ February 2022, even if you plan to discuss your homework in a later date. This is needed for completing your course evaluation by the deadline of February 26th 2022

Final Guest Lecture

- The final lecture of Tuesday 21st Dec 10:30am will be given by dr. Marco Zanella, Ph.Dr. in Computer Science, who will talk about "**Verification of Machine Learning models by abstract interpretation**". The lecture will be given through Zoom only.