ABSTRACT INTERPRETATION: THEORY AND APPLICATIONS

P. Cousot

Patrick.Cousot@ens.fr http://www.di.ens.fr/~cousot

Second International Summer School in Computational Logic, ISCL 2002

25th—30th August 2002, Acquafredda di Maratea (Basilicata, Italy)

LESSON 3 (3/3, Static Analysis)

A Potpourri of Applications of Abstract Interpretation

Application to Static Program Analysis[29]

- P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques. Grenoble, 21 Mar. 1978.
- P. Cousot. *Semantic Foundations of Program Analysis*. Ch. 10 of *Program Flow Analysis: Theory and Applications*, S.S. Muchnick & N.D. Jones, pp. 303–342. Prentice-Hall, 1981.

[29] Now called *software model checking*!

What is static program analysis?

- Automatic static/compile time determination of dynamic/run-time properties of programs;

What is static program analysis?

- Automatic static/compile time determination of dynamic/run-time properties of programs;
- **Basic idea:** use effective computable approximations of the program semantics;

  Advantage: fully automatic, no need for error-prone user designed model or costly user interaction;

  Drawback: can only handle properties captured by the approximation.

Collecting Semantics Abstractions

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \subseteq \rangle \xleftarrow[\alpha]{\gamma} \langle \wp(\Sigma), \subseteq \rangle$$

Collecting Semantics Abstractions

$$\langle \wp(\Sigma^{+} \cup \Sigma^{\omega}), \subseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \wp(\Sigma), \subseteq \rangle$$

Example 1: reachable states (forward analysis)

$$\alpha_I(X) \stackrel{\text{def}}{=} \{\sigma_i \mid \sigma \in X \wedge \sigma_0 \in I \wedge i \in \mathrm{Dom}(\sigma)\}$$

Collecting Semantics Abstractions

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \subseteq \rangle \xleftarrow[\alpha]{\gamma} \langle \wp(\Sigma), \subseteq \rangle$$

Example 1: reachable states (forward analysis)

$$\alpha_I(X) \stackrel{\text{def}}{=} \{\sigma_i \mid \sigma \in X \wedge \sigma_0 \in I \wedge i \in \mathrm{Dom}(\sigma)\}$$

Example 2: ancestor states (backward analysis)

$$\alpha_F(X) \stackrel{\text{def}}{=} \{\sigma_i \mid \sigma \in X \wedge \exists n \in \mathrm{Dom}(\sigma) : 0 \leq i \leq n \wedge \sigma_n \in F\}$$

Partitioning

- If $\Sigma = C \times M$ (control and store state) and C is finite [30], we can partition:

$$\langle \wp(C \times M), \subseteq \rangle \underset{\alpha_C}{\overset{\gamma_C}{\leftrightarrows}} \langle C \mapsto \wp(M), \dot{\subseteq} \rangle$$

$$\alpha_C(S) = \boldsymbol{\lambda} c \in C \cdot \{m \mid \langle c, m \rangle \in S\}$$

30 use e.g. dynamic partitioning if C is infinite

Partitioning

- If $\Sigma = C \times M$ (control and store state) and C is finite [30], we can partition:

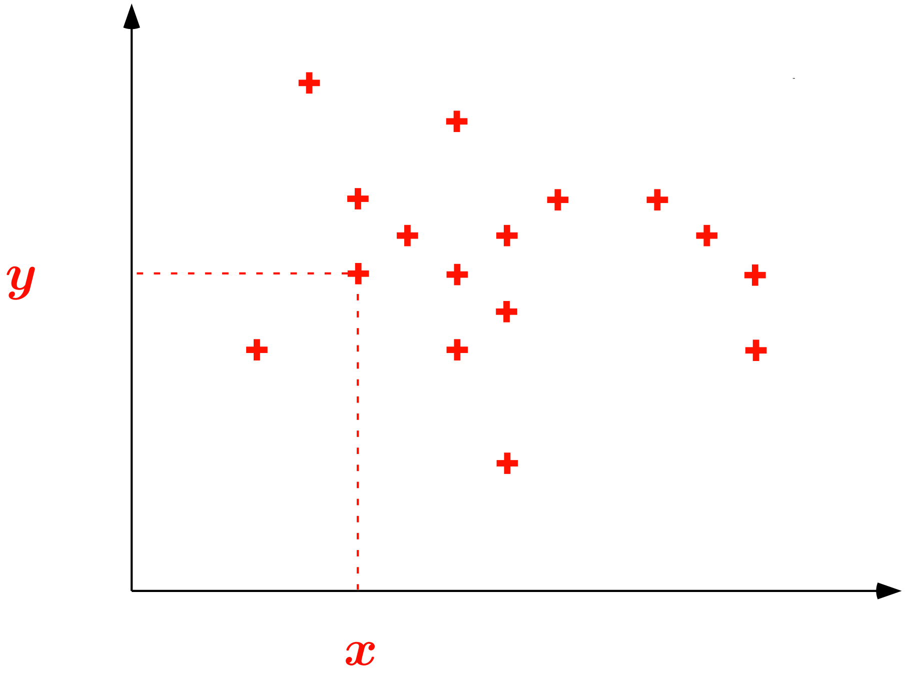
$$\langle \wp(C \times M), \subseteq \rangle \xleftrightarrow[\alpha_C]{\gamma_C} \langle C \mapsto \wp(M), \dot{\subseteq} \rangle$$

$$\alpha_C(S) \;=\; \boldsymbol{\lambda} c \in C \cdot \{m \mid \langle c, m \rangle \in S\}$$

- It remains to find abstractions of store properties $\wp(M)$ where $M = V \mapsto D$ (variables to data) e.g. of [in]finite set of points of the euclidian space.

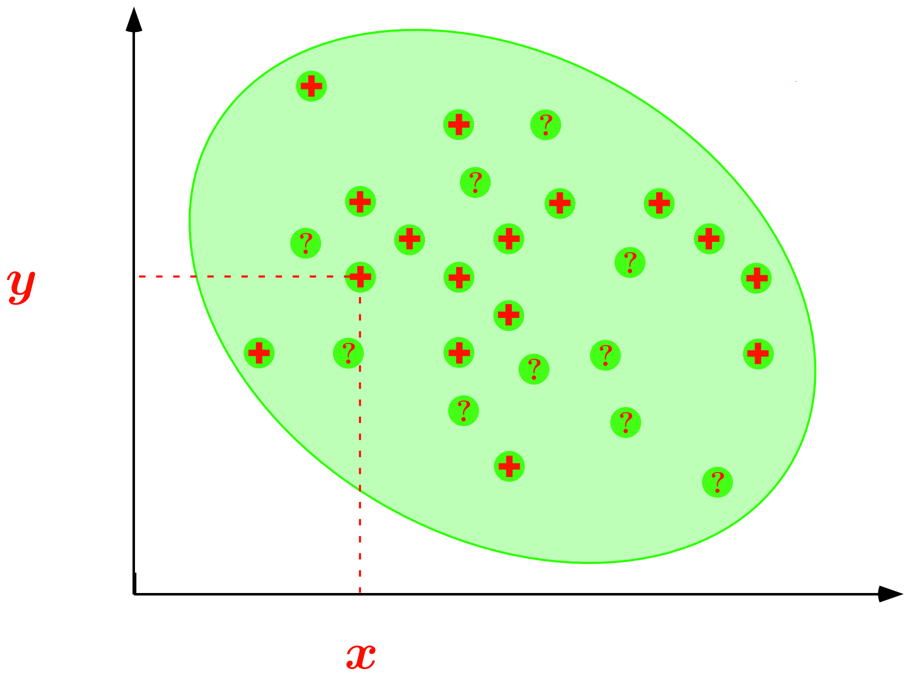
30 use e.g. dynamic partitioning if C is infinite.

Approximations of an [in]finite set of points:



$$\{\ldots, \langle 19,\ 77\rangle, \ldots,\\ \langle 20,\ 02\rangle, \ldots\}$$

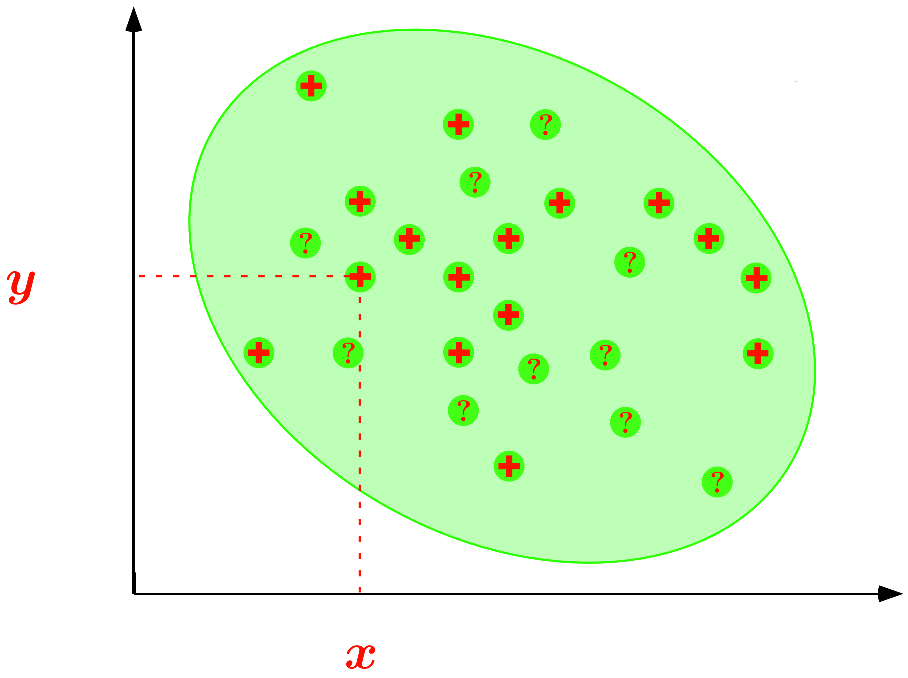
Approximations of an [in]finite set of points: From Above



$$\{\ldots, \langle 19,\ 77\rangle, \ldots,$$

$$\langle 20,\ 02\rangle, \langle ?,\ ?\rangle, \ldots\}$$

Approximations of an [in]finite set of points: From Above



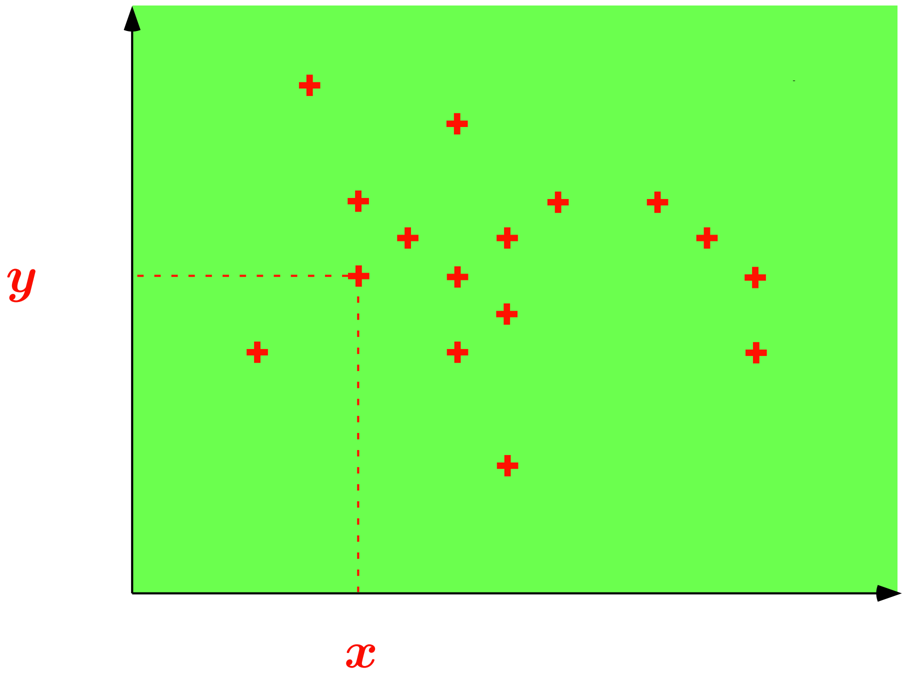
$\{\ldots, \langle 19,\ 77\rangle, \ldots,$

$\langle 20,\ 02\rangle, \langle ?,\ ?\rangle, \ldots\}$

From Below: dual[31] + combinations.

31 Trivial for finite states (liveness model-checking), more difficult for infinite states (variant functions).

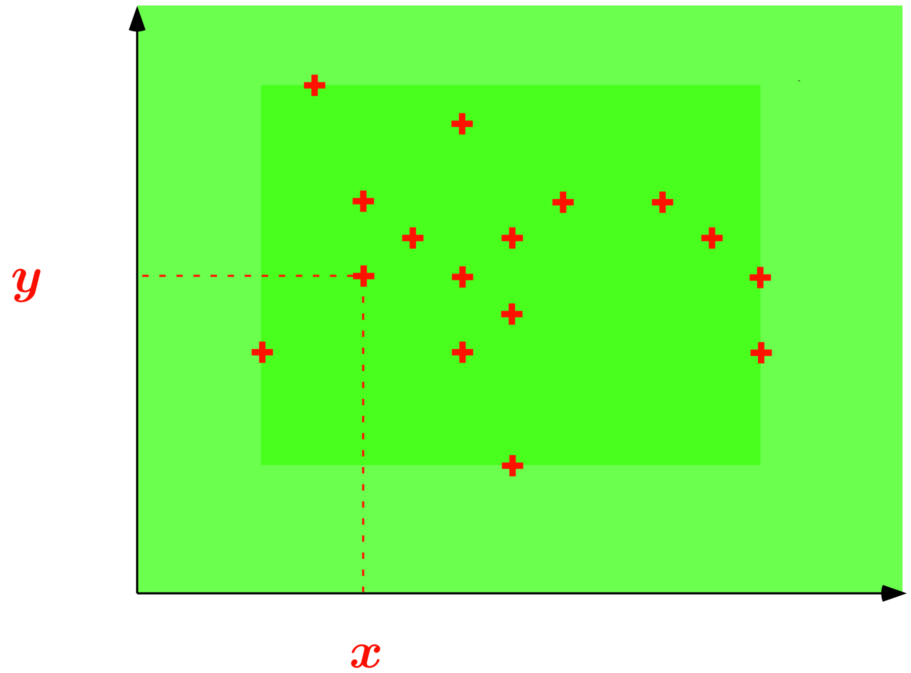
Effective computable approximations of an [in]finite set of points; Signs[32]



$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

32 P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979.

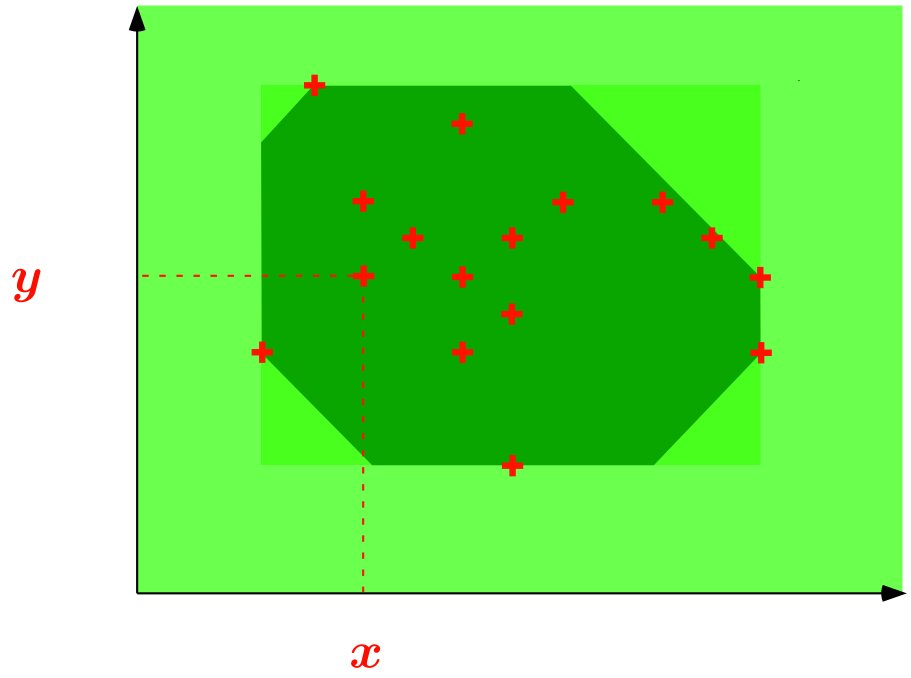
Effective computable approximations of an [in]finite set of points; Intervals[33]



$$\begin{cases} x \in [19,\ 77] \\ y \in [20,\ 02] \end{cases}$$

[33] P. Cousot & R. Cousot. *Static determination of dynamic properties of programs.* Proc. 2nd Int. Symp. on Programming, Dunod, 1976.

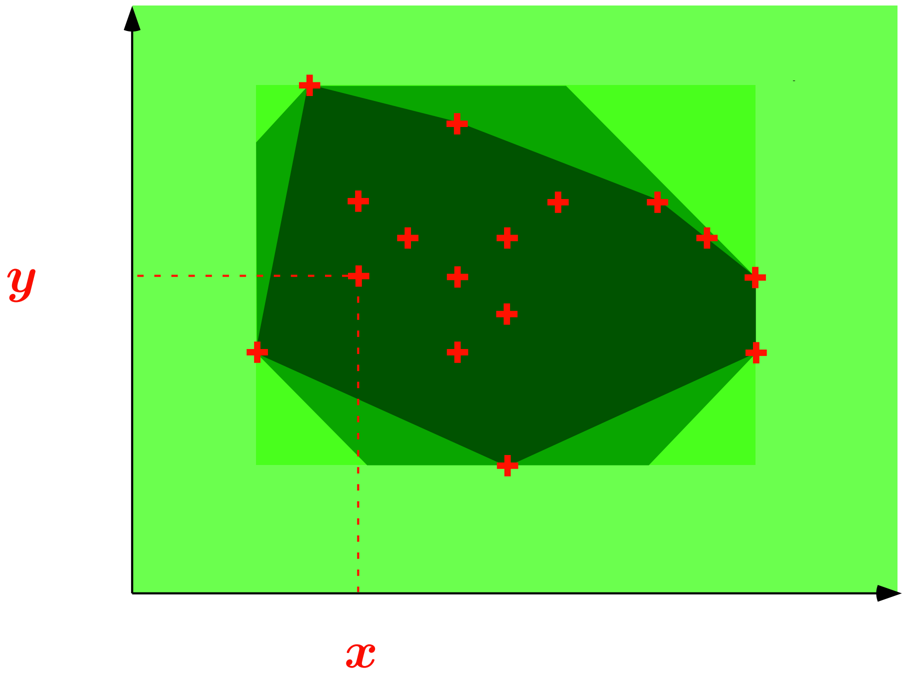
Effective computable approximations of an [in]finite set of points; Octagons[34]



$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

[34] A. Miné. *A New Numerical Abstract Domain Based on Difference-Bound Matrices*. PADO'2001. LNCS 2053, pp. 155–172. Springer 2001.

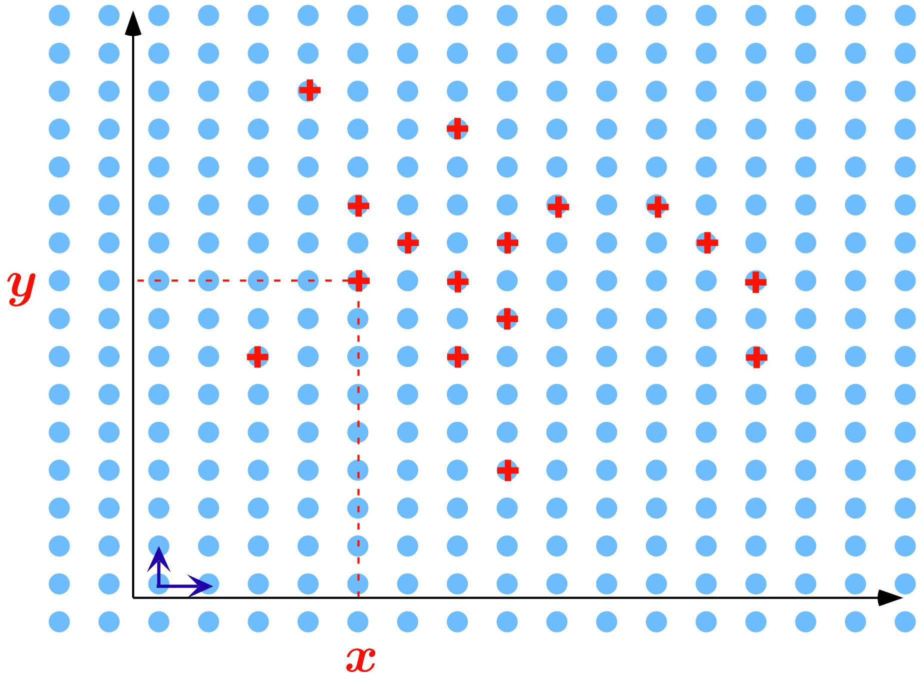
Effective computable approximations of an [in]finite set of points; Polyhedra [35]



$$\begin{cases} 19x + 77y \leq 2002 \\ 20x + 02y \geq 0 \end{cases}$$

[35] P. Cousot & N. Halbwachs. *Automatic discovery of linear restraints among variables of a program*. ACM POPL, 1978, pp. 84–97.

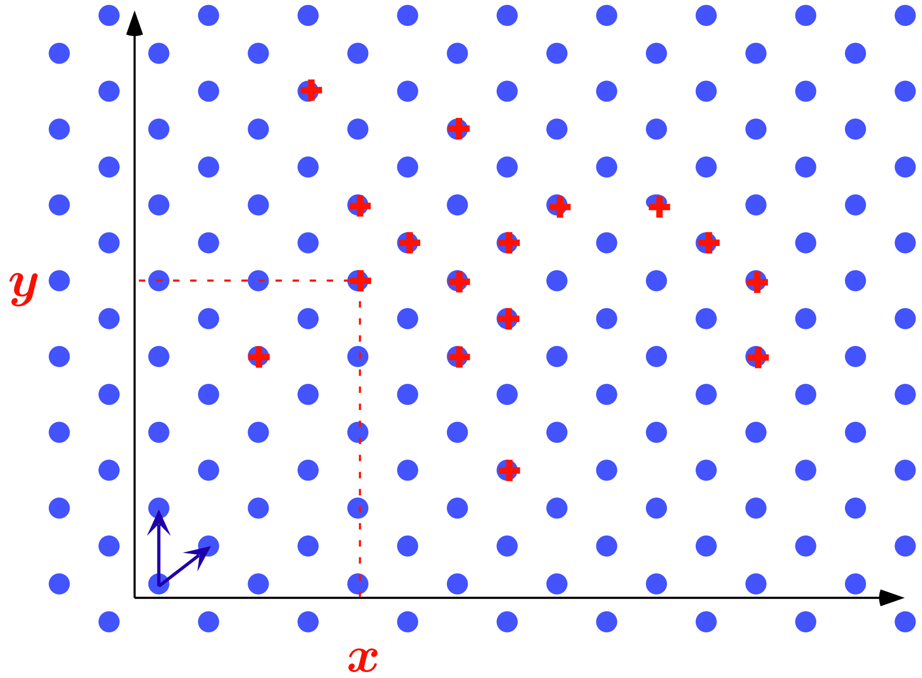
Effective computable approximations of an [in]finite set of points; Simple congruences [36]



$$\begin{cases} x = 19 \bmod 77 \\ y = 20 \bmod 99 \end{cases}$$

36 Ph. Granger. *Static Analysis of Arithmetical Congruences*. Int. J. Comput. Math. 30, 1989, pp. 165–190.

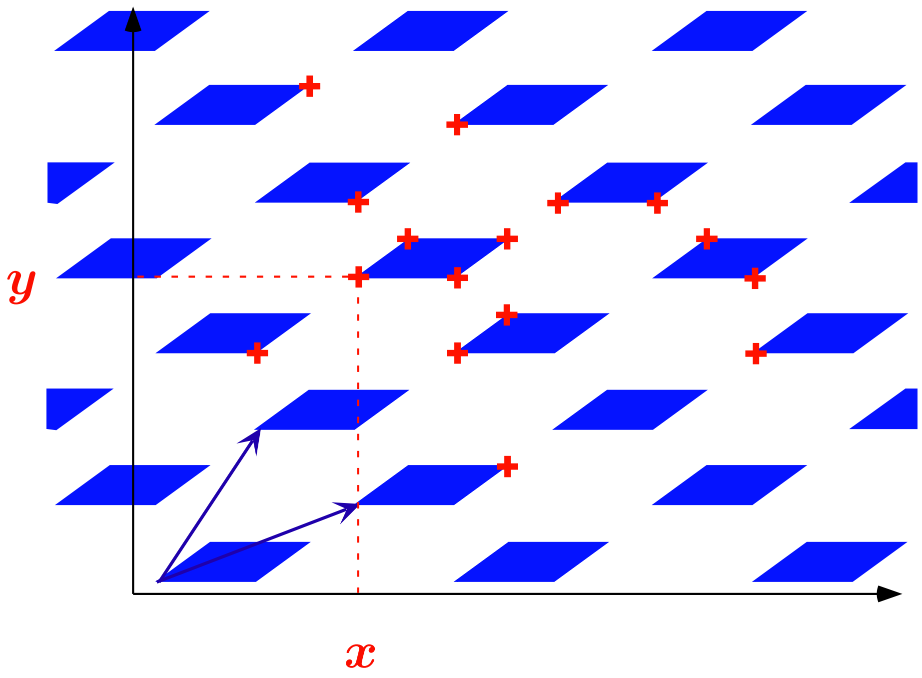
Effective computable approximations of an [in]finite set of points; Linear congruences[37]



$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x - 1y = 9 \bmod 9 \end{cases}$$

37 Ph. Granger. *Static Analysis of Linear Congruence Equalities among Variables of a Program*. TAPSOFT '91, pp. 169–192. LNCS 493, Springer, 1991.

Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences[38]



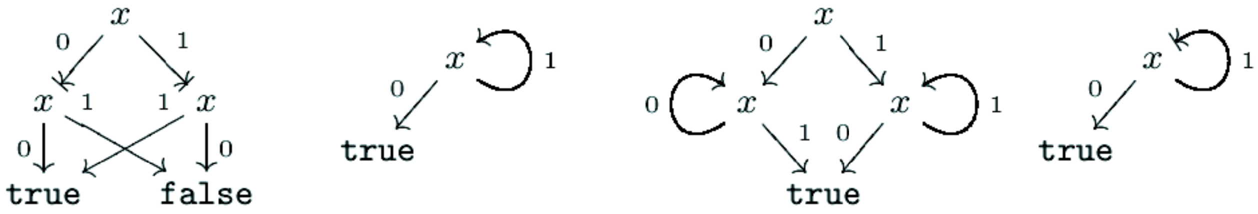
$$\begin{cases} 1x + 9y \in [0, 77] \bmod 10 \\ 2x - 1y \in [0, 99] \bmod 11 \end{cases}$$

---

[38] F. Masdupuy. *Array Operations Abstraction Using Semantic Analysis of Trapezoid Congruences*. ACM ICS '92.

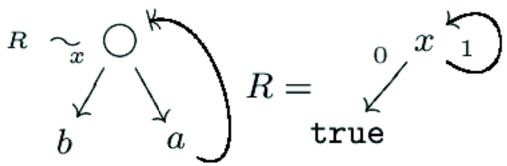
Example of Effective Abstractions of Infinite Sets of Infinite Trees [39]

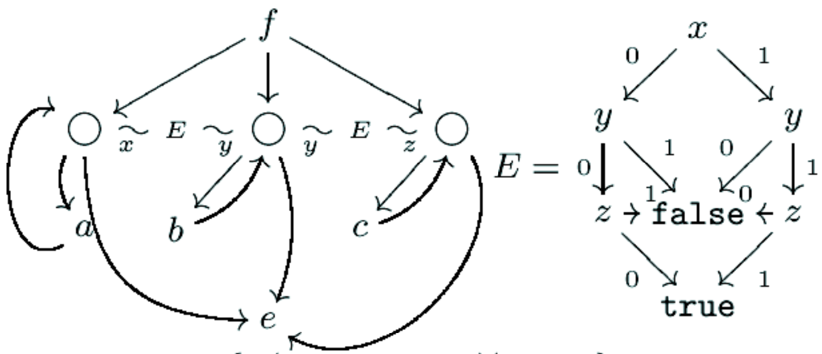
Binary Decision Graphs:



$\{0^\omega, 1^\omega\}$ infinite number of 0's fair vectors ends by 0^ω

Tree Schemata:





$\{f(a^n e, b^n e, c^n e) | n \in \mathbb{N}\}$

$\{a^n b | n \in \mathbb{N}\}$

Note that E is the equality relation.

---

39 L. Mauborgne. *Improving the Representation of Infinite Trees to Deal with Sets of Trees*. ESOP'00. LNCS 1782, pp. 275–289, Springer, 2000.

On Widenings [48]

[48] P. Cousot, R. Cousot: Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. PLILP 1992: 269-295

Widening Operator

A widening operator $\nabla \in \overline{L} \times \overline{L} \mapsto \overline{L}$ is such that:

- Correctness:
  - $\forall x, y \in \overline{L} : \gamma(x) \sqsubseteq \gamma(x \nabla y)$
  - $\forall x, y \in \overline{L} : \gamma(y) \sqsubseteq \gamma(x \nabla y)$
- Convergence:
  - for all increasing chains $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$, the increasing chain defined by $y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}, \ldots$ is not strictly increasing.

Fixpoint Approximation with Widening

The upward iteration sequence with widening:

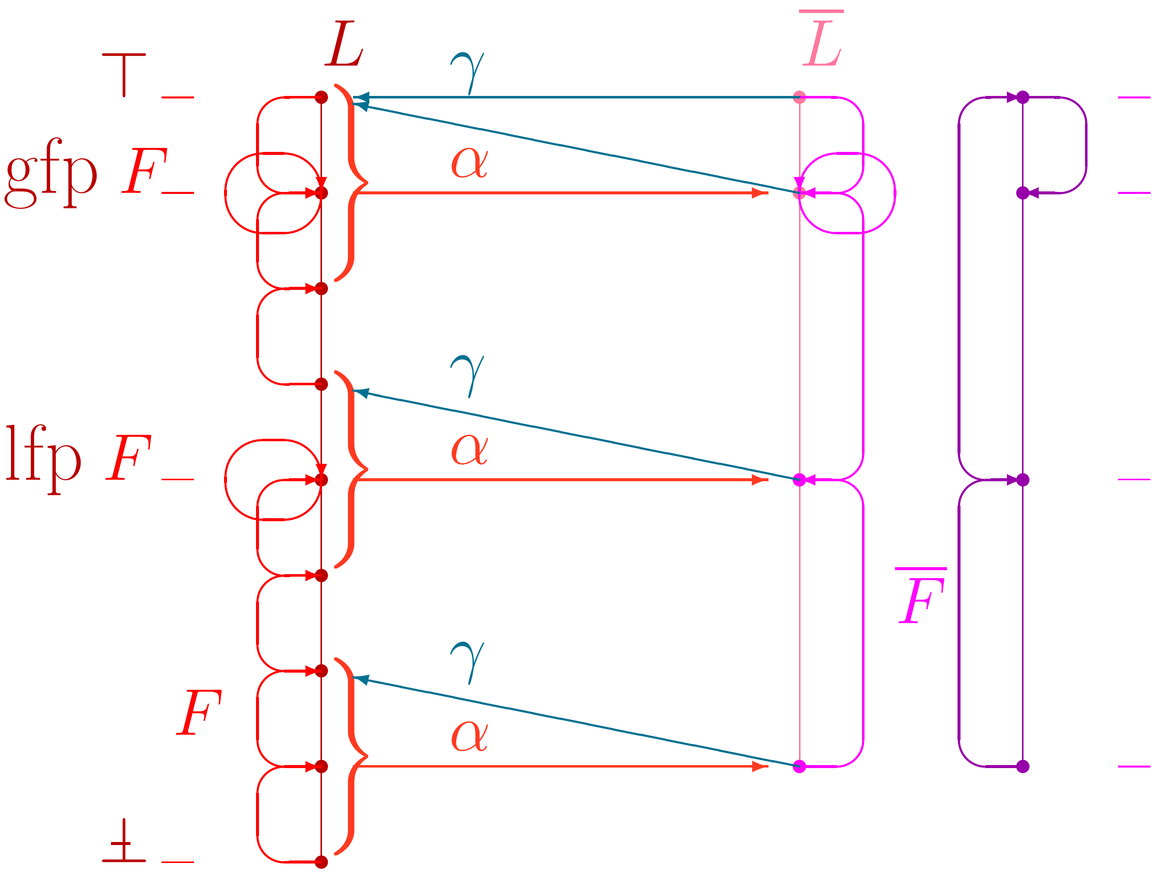
- $\hat{X}^0 = \overline{\bot}$ (infimum)
- $\hat{X}^{i+1} = \hat{X}^i$ if $\overline{F}(\hat{X}^i) \sqsubseteq \hat{X}^i$

  $= \hat{X}^i \nabla F(\hat{X}^i)$ otherwise

is ultimately stationary and its limit $\hat{A}$ is a sound upper approximation of $\mathrm{lfp}^{\overline{\bot}}\ \overline{F}$:

$$\mathrm{lfp}^{\overline{\bot}}\ \overline{F} \sqsubseteq \hat{A}$$

Fixpoint Approximation with Widening/Narrowing



$$\hat{X}^2 = \hat{X}^1 \nabla \overline{F}(\hat{X}^1)$$
$$= \overline{\top} = \check{X}^0$$
$$\check{X}^1 = \check{X}^0 \, \Delta \overline{F}(\check{X}^0)$$
$$= \mathrm{gfp}\overline{F} = \mathrm{lfp}\overline{F}$$

$$\hat{X}^1 = \hat{X}^0 \nabla \overline{F}(\hat{X}^0)$$

$$\hat{X}^0 = \overline{\bot}$$

Interval Widening

- $\overline{L} = \{\bot\} \cup \{[\ell, u] \mid \ell \in \mathbb{Z} \cup \{-\infty\} \wedge u \in \mathbb{Z} \cup \{+\infty\} \wedge \ell \leq u\}$
- The widening extrapolates unstable bounds to infinity:

$$
\begin{aligned}
\bot \mathbin{\nabla} X &= X \\
X \mathbin{\nabla} \bot &= X \\
[\ell_0,\ u_0] \mathbin{\nabla} [\ell_1,\ u_1] &= [\text{if } \ell_1 < \ell_0 \text{ then } -\infty \text{ else } \ell_0, \\
&\qquad \text{if } \mathrm{u}_1 > \mathrm{u}_0 \text{ then } +\infty \text{ else } \mathrm{u}_0]
\end{aligned}
$$

Not monotone. For example $[0,\ 1] \sqsubseteq [0,\ 2]$ but $[0,\ 1] \mathbin{\nabla} [0,\ 2] = [0,\ +\infty] \not\sqsubseteq [0,\ 2] = [0,\ 2] \mathbin{\nabla} [0,\ 2]$

Interval Widening with Thresholds

- Extrapolate to thresholds, zero, one or infinity:

$$
\begin{aligned}
[\ell_0,\, u_0] \nabla [\ell_1,\, u_1] = [&\text{if } \ell \leq \ell_1 < \ell_0 \ \wedge\ \ell \in \{1, 0, -1\} \text{ then } \mathrm{l} \\
&\text{elsif } \ell_1 < \ell_0 \text{ then } -\infty \\
&\text{else } \ell_0, \\
&\text{if } \mathrm{u}_0 < \mathrm{u}_1 \leq \mathrm{u} \ \wedge\ \mathrm{u} \in \{-1, 0, 1\} \text{ then } \mathrm{u} \\
&\text{elsif } \mathrm{u}_0 < \mathrm{u}_1 \text{ then } +\infty \\
&\text{else } \mathrm{u}_0]
\end{aligned}
$$

- So the analysis is always as good as the sign analysis.

Non-Existence of Finite Abstractions

Let us consider the infinite family of programs parameterized by the *mathematical constants* n_1, n_2 $(n_1 \leq n_2)$:

```
X := n1;
while X <= n2 do
   X := X + 1;
od
```

- An interval analysis with widening/narrowing will discover the loop invariant $\mathtt{X} \in [n_1, n_2]$;
- To handle all programs in the family without false alarm, the abstract domain must contain all such intervals;

  $\Rightarrow$ No single finite abstract domain will do for all programs!

- Yes, but predicate abstraction with refinement will do (?) for each program in the family (since it is equivalent to a widening)[49]!
- Indeed no, since:
  - Predicate abstraction is unable to express limits of infinite sequences of predicates;
  - Not all widening proceed by eliminating constraints:
  - A narrowing is necessary anyway in the refinement loop (to avoid infinitely many refinements);
  - Not speaking of costs!

---

49 T. Ball, A. Podelski, S.K. Rajamani. Relative Completeness of Abstraction Refinement for Software Model Checking. TACAS 2002: 158-172.

On the Design of Program Static Analyzers

- P. Cousot. *The Calculational Design of a Generic Abstract Interpreter*. In *Calculational System Design*, M. Broy and R. Steinbrüggen (Eds). Vol. 173 of NATO Science Series, Series F: Computer and Systems Sciences. IOS Press, pp. 421–505, 1999.
- The corresponding *generic abstract interpreter* (written in Ocaml) is available at URL www.di.ens.fr/~cousot

On the Design of Program Analyzers

- The abstract interpretation theory provides the design principles;

On the Design of Program Analyzers

- The abstract interpretation theory provides the design principles;
- In practice, one must find the appropriate tradeoff between generality, precision and efficiency;

On the Design of Program Analyzers

- The abstract interpretation theory provides the design principles;
- In practice, one must find the appropriate tradeoff between generality, precision and efficiency;
- There is a full range of program analyzers from

  general purpose analyzers for programming languages

  to

  specific analyzers for a given program (software model checking).

Specific Static Program Analyzers

- A complete specific analyzer[50] (for a given software or hardware program) can always use a finite abstract domain[51];

50 Called a *software model checker?*

51 P. Cousot. *Partial completeness of abstract fixpoint checking*. SARA'2000. LNAI 1864, pp. 1–25. Springer.

Specific Static Program Analyzers

- A complete specific analyzer[50] (for a given software or hardware program) can always use a finite abstract domain[51];
- The design of a complete specific analyzer is logically equivalent to a correctness proof of the program;

---

50 Called a *software model checker?*

51 P. Cousot. *Partial completeness of abstract fixpoint checking*. SARA'2000. LNAI 1864, pp. 1–25. Springer.

Specific Static Program Analyzers

- A complete specific analyzer [50] (for a given software or hardware program) can always use a finite abstract domain [51];
- The design of a complete specific analyzer is logically equivalent to a correctness proof of the program;
- Such analyzers are precise but not reusable hence very costly to develop.

---

50 Called a *software model checker?*

51 P. Cousot. *Partial completeness of abstract fixpoint checking*. SARA'2000. LNAI 1864, pp. 1–25. Springer.

General-Purpose Static Program Analyzers

- To handle infinitely many programs for non-trivial properties, a general-purpose analyser must use an infinite abstract domain [52];

[52] P. Cousot & R. Cousot. *Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation*. PLILP'92. LNCS 631, pp. 269–295. Springer.

General-Purpose Static Program Analyzers

- To handle infinitely many programs for non-trivial properties, a general-purpose analyser must use an infinite abstract domain [52];
- Such analyzers are huge for complex languages hence very costly to develop but reusable;

[52] P. Cousot & R. Cousot. *Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation*. PLILP'92. LNCS 631, pp. 269–295. Springer.

General-Purpose Static Program Analyzers

- To handle infinitely many programs for non-trivial properties, a general-purpose analyser must use an infinite abstract domain [52];
- Such analyzers are huge for complex languages hence very costly to develop but reusable;
- There are always programs for which they lead to false alarms;

52 P. Cousot & R. Cousot. *Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation*. PLILP'92. LNCS 631, pp. 269–295. Springer.

General-Purpose Static Program Analyzers

- To handle infinitely many programs for non-trivial properties, a general-purpose analyser must use an infinite abstract domain [52];
- Such analyzers are huge for complex languages hence very costly to develop but reusable;
- There are always programs for which they lead to false alarms;
- Although incomplete, they are very useful for verifying/testing/debugging.

[52] P. Cousot & R. Cousot. *Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation*. PLILP'92. LNCS 631, pp. 269–295. Springer.

Parametric Specializable Static Program Analyzers

- The abstraction can be tailored to significant classes of programs (e.g. critical synchronous real-time embedded systems);

Parametric Specializable Static Program Analyzers

- The abstraction can be tailored to significant classes of programs (e.g. critical synchronous real-time embedded systems);
- This leads to *very efficient analyzers* with ~~*almost*~~ *zero-false alarm* even for large programs.

Experience Report on a Parametric Specializable Program Static Analyzer

B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, X. Rival.

Example of Parametric Specializable Static Program Analyzers

Analyzer under development, very first results!

- C programs: safety critical embedded real-time synchronous software for non-linear control of complex systems;
- 10 000 LOCs, 1300 global variables (booleans, integers, real, arrays, macros, non-recursive procedures);
- Implicit specification: absence of runtime errors (no integer/floating point arithmetic overflow, no array bound overflow);
- Initial design: 2h, 110 false alarms (general purpose analyzer);

Experience report

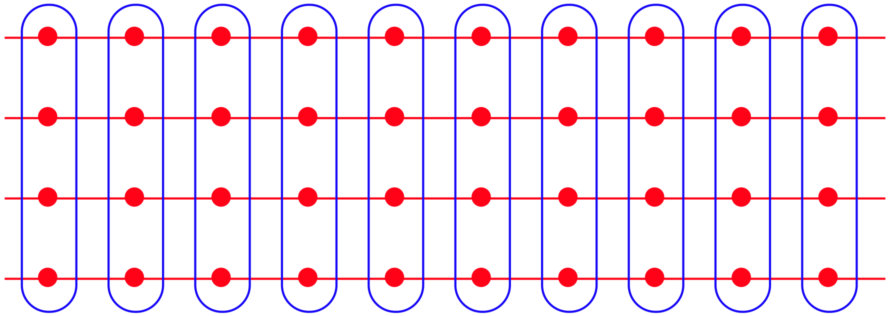
- Comparative results (commercial software):
  - 70 false alarms, 2 days, 500 Megabytes;
- Initial redesign:
  - Weak relational domain with time;
- Parametrisation:
  - Hypotheses on volatile inputs;
  - Staged widenings with thresholds;
  - Local refinements of the parameterized abstract domains;

Experience report

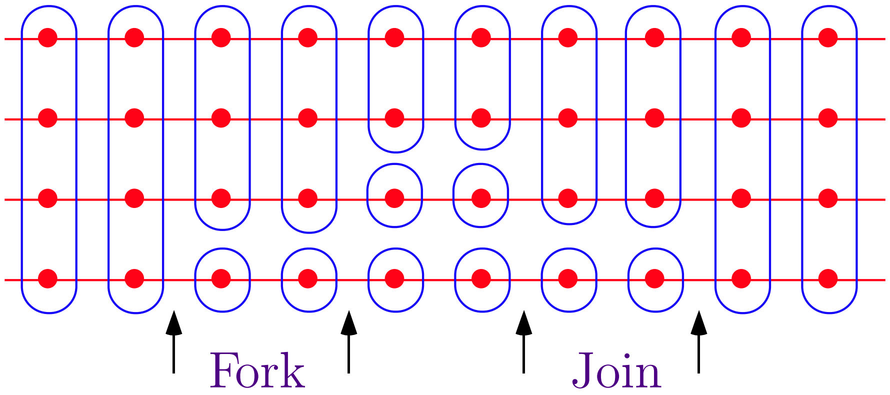
- Comparative results (commercial software):
  - 70 false alarms, 2 days, 500 Megabytes;
- Initial redesign:
  - Weak relational domain with time;
- Parametrisation:
  - Hypotheses on volatile inputs;
  - Staged widenings with thresholds;
  - Local refinements of the parameterized abstract domains;
- Results:
  - No false alarm, 14s, 20 Megabytes.

Example of refinement: trace partitionning

Control point partitionning:

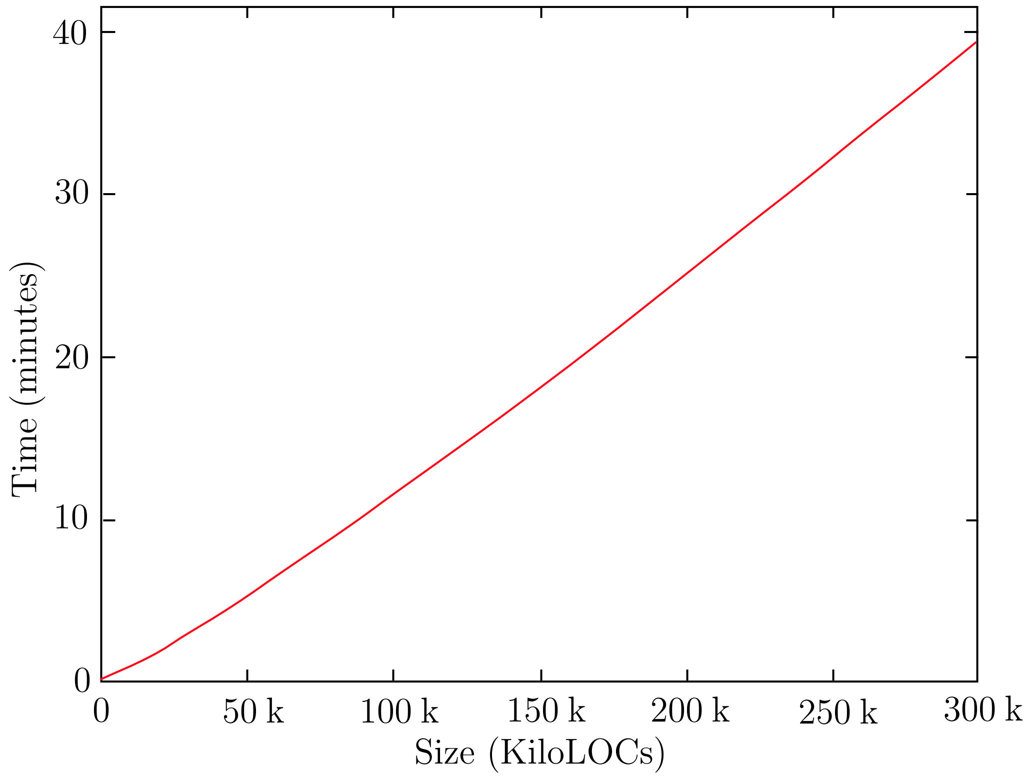


Trace partitionning:



Performance: Space and Time

$$\text{Space} = \mathcal{O}(\text{LOCs})$$
$$\text{Time} = \mathcal{O}(\text{LOCs} \times (\ln(\text{LOCs}))^{1.5})$$



Conclusion

Conclusion on Formal Methods

- Formal methods concentrate on the deductive/exhaustive verification of (abstract) models of the execution of programs;
- Most often this abstraction into a model is *manual* and left completely *informal*, if not tortured to meet the tool limitations;
- Semantics concentrates on the rigorous formalization of the execution of programs;
- So models should abstract the program semantics.

Conclusion on Formal Methods

- Formal methods concentrate on the deductive/exhaustive verification of (abstract) models of the execution of programs;
- Most often this abstraction into a model is *manual* and left completely *informal*, if not tortured to meet the tool limitations;
- Semantics concentrates on the rigorous formalization of the execution of programs;
- So models should abstract the program semantics. This is the whole purpose of Abstract Interpretation!

Conclusion on Abstract Interpretation

- Abstract interpretation provides mathematical foundations of most semantics-based program verification and manipulation techniques;

Conclusion on Abstract Interpretation

- Abstract interpretation provides mathematical foundations of most semantics-based program verification and manipulation techniques;
- In abstract interpretation, the abstraction of the program semantics into an approximate semantics is automated so that one can go *much beyond* examples modelled by hand;
- The abstraction can be tailored to classes of programs so as to design *very efficient analyzers* with ~~*almost*~~ *zero-false alarm*.