Automatic Verification by Abstract Interpretation

Patrick COUSOT
École Normale Supérieure
45 rue d'Ulm
75230 Paris cedex 05, France
Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

VMCAI'03, Courant Institute, NYU, New York
Jan. 10, 2003

Abstract Interpretation

Abstract Interpretation

- Abstract interpretation theory [Thesis, POPL '77, POPL '79, JLC '92] formalizes the idea of abstraction for mathematical constructs involved in the specification of properties of computer systems.

References

[Thesis] 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, Université scientifique et médicale de Grenoble, Grenoble, 21 Mar. 1978.

[POPL '77] P. Cousot & R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, 1977.

[PO- PL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, 1979.

[JLC '92] P. Cousot & R. Cousot. Abstract interpretation frameworks. *J. Logic and Comp.*, 2(4):511–547, 1992.

Applications of Abstract Interpretation

- **Static Program Analysis** [POPL '77,78,79] inluding **Dataflow Analysis** [POPL '79,00], **Set-based Analysis** [FPCA '95]
- **Syntax Analysis** [TCS 290(1) 2002]
- **Hierarchies of Semantics (including Proofs)** [POPL '92, TCS 277(1–2) 2002]
- **Typing** [POPL '97]
- **Model Checking** [POPL '00]
- **Program Transformation** [POPL '02]

The Abstract Interpretation Methodology

- All these techniques involve approximations that can be formalized by abstract interpretation;
- Consequently, sound (and complete) abstracts semantics, including abstract models, algorithms, etc can be derived systematically in a mathematically constructive way by algebraic calculation.

A Challenge for Abstract Interpretation

- Most applications of abstract interpretation tolerate a small rate (typically 5 to 15%) of false alarms:
  - Run-time checks elimination, Partial evaluation $\rightarrow$ do not optimize,
  - Typing $\rightarrow$ reject some correct programs, etc;
- Some applications require no false alarm at all:
  - Program verification.
- Theoretically possible [SARA '00]; Practically feasible?

Reference

[SARA '00] P. Cousot. Partial Completeness of Abstract Fixpoint Checking, invited paper. In 4^{th} *Int. Symp. SARA '2000*, LNAI 1864, Springer, pp. 1–25, 2000.

Requirements for Verification

- *Correctness*[1] (excludes non exhaustive methods like simulation or test),
- *Automation* (no manual production of a program model, no human assistance for provers),
- *Precision* (general-purpose static program analyzers produce too many false alarms),
- *Scaling up* (to a few hundred thousand lines), and
- *Efficiency* (with minimal space and time requirements for verification during software production).

[1] Automatic verification for proving the absence of errors, not their presence (i.e. not debugging).

Content

- A short introduction to abstract interpretation
- Application to predicate abstraction
- A practical application of abstract interpretation to the verification of safety critical embedded software
- Would automatic predicate abstraction have done it?
- Conclusion

A Short Introduction to Abstract Interpretation (based on [POPL '79, Sec. 5])

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Moore Family-Based Abstraction [POPL '79, Sec. 5.1]

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Properties

- We represent properties P of objects $s \in \Sigma$ as sets of objects $P \in \wp(\Sigma)$ (which have the property in question);

  Example: the property "*to be an even natural number*" is $\{0, 2, 4, 6, \ldots\}$

Complete Lattice of Properties

- The set of properties of objects Σ is a complete boolean lattice:

$$\langle \wp(\Sigma),\ \subseteq,\ \emptyset,\ \Sigma,\ \cup,\ \cap,\ \neg \rangle\ .$$

Abstraction

A reasoning/computation such that:

- only some properties can be used;
- the properties that can be used are called "*abstract*";
- so, the (other concrete) properties must be approximated by the abstract ones;

Direction of Approximation

- Approximation from above: approximate P by $\overline{P}$ such that $P \subseteq \overline{P}$;
- Approximation from below: approximate P by $\underline{P}$ such that $\underline{P} \subseteq P$ (dual).

Abstract Properties

- Abstract Properties: a set $\overline{\mathcal{A}} \subsetneq \wp(\Sigma)$ of properties of interest (the only one which can be used to approximate others).

In Absence of (Upper) Approximation

- What to say when some property has no (computable) abstraction?
  - loop?
  - block?
  - ask for help?
  - say something!

I don't know

- Any property should be approximable from above by I don't know (i.e. "true" or Σ).

Minimal Approximations

- A concrete property $P \in \wp(\Sigma)$ is most precisely abstracted by any minimal upper approximation $\overline{P} \in \overline{\mathcal{A}}$:

$$P \subseteq \overline{P}$$
$$\nexists \overline{P'} \in \overline{\mathcal{A}} : P \subseteq \overline{P'} \subsetneq \overline{P}$$

- So, an abstract property $\overline{P} \in \overline{\mathcal{A}}$ is best approximated by itself.

Which Minimal Approximation is Most Useful?

- Which minimal approximation is most useful depends upon the circumstances;
- **Example** (rule of signs):
  - 0 is better approximated as positive in "$\ \ 3 + 0$";
  - 0 is better approximated as negative in "$-3 + 0$".

Avoiding Backtracking

- We don't want to exhaustively try all minimal approximations;
- We want to use only one of the minimal approximations;

Which Minimal Abstraction to Use?

- Which minimal abstraction to choose?
  - make a circumstantial choice [2];
  - make a definitive arbitrary choice [3];
  - require the existence of a best choice [4].

Reference

[JLC '92] P. Cousot & R. Cousot. Abstract interpretation frameworks. *J. Logic and Comp.*, 2(4):511–547, 1992.

2 [JLC '92] uses a concretization function.
3 [JLC '92] uses an abstraction function.
4 [JLC '92] uses an abstraction/concretization Galois connection (this talk).

Best Abstraction

- We require that all concrete property $P \in \wp(\Sigma)$ have a best abstraction $\overline{P} \in \overline{\mathcal{A}}$:

$$P \subseteq \overline{P}$$
$$\forall \overline{P'} \in \overline{\mathcal{A}} : (P \subseteq \overline{P'}) \Longrightarrow (\overline{P} \subseteq \overline{P'})$$

- So, by definition of the greatest lower bound/meet $\cap$:

$$\overline{P} = \bigcap \{\overline{P'} \in \overline{\mathcal{A}} \mid P \subseteq \overline{P'}\} \in \overline{\mathcal{A}}$$

Moore Family

- So, the hypothesis that any concrete property $P \in \wp(\Sigma)$ has a best abstraction $\overline{P} \in \overline{\mathcal{A}}$ implies that:

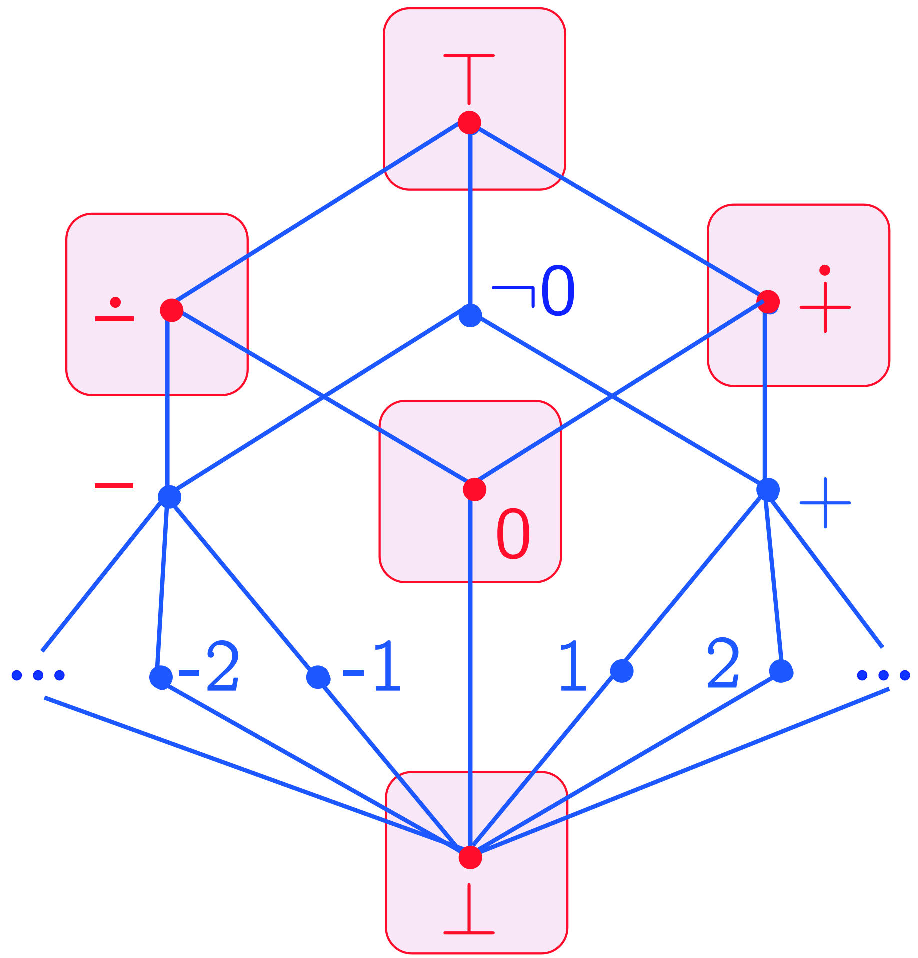
  $\overline{\mathcal{A}}$ is a Moore family

  i.e. it is closed under intersection $\bigcap$:

$$\forall S \subseteq \overline{\mathcal{A}} : \bigcap S \in \overline{\mathcal{A}}$$

- In particular $\bigcap \emptyset = \Sigma \in \overline{\mathcal{A}}$.

Example of Moore Family-Based Abstraction



The Lattice of Abstractions (1)

- The set $\mathcal{M}(\wp(\wp(\Sigma)))$ of all abstractions i.e. of Moore families on the set $\wp(\Sigma)$ of concrete properties is the complete lattice of abstractions

$$\langle \mathcal{M}(\wp(\wp(\Sigma))),\ \supseteq,\ \wp(\Sigma),\ \{\Sigma\},\ \lambda S \cdot \mathcal{M}(\cup S),\ \cap \rangle$$

where:

$$\mathcal{M}(\overline{\mathcal{A}}) = \{\bigcap S \mid S \subseteq \overline{\mathcal{A}}\}$$

is the $\subseteq$-least Moore family containing $\overline{\mathcal{A}}$.

Closure Operator-Based Abstraction [POPL '79, Sec. 5.2]

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Closure Operator Induced by an Abstraction

The map $\rho_{\overline{\mathcal{A}}}$ mapping a concrete property $P \in \wp(\Sigma)$ to its best abstraction $\rho_{\overline{\mathcal{A}}}(P)$ in $\overline{\mathcal{A}}$ is:

$$\rho_{\overline{\mathcal{A}}}(P) = \bigcap\{\overline{P} \in \overline{\mathcal{A}} \mid P \subseteq \overline{P}\} \ .$$

It is a closure operator:

- extensive,
- idempotent,
- isotone/monotonic;

such that $P \in \bar{\mathcal{A}} \iff P = \rho_{\overline{\mathcal{A}}}(P)$

hence $\overline{\mathcal{A}} = \rho_{\overline{\mathcal{A}}}(\wp(\Sigma))$.

Abstraction Induced by a Closure Operator

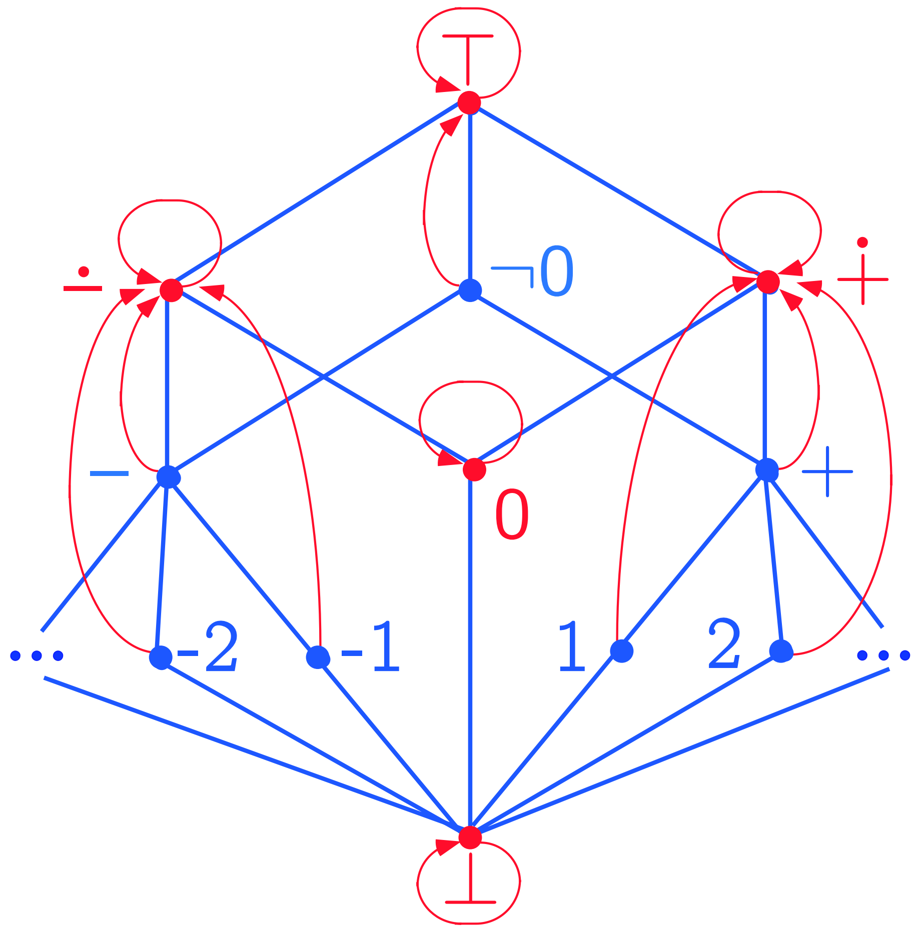
- Any closure operator ρ on the set of properties $\wp(\Sigma)$ induces an abstraction $\rho(\wp(\Sigma))$.

  Examples:

  - $\lambda P \cdot P$ the most precise abstraction (identity),
  - $\lambda P \cdot \Sigma$ the most imprecise abstraction (I don't know).

- Closure operators are isomorphic to the Moore families (i.e. their fixpoints).

Example of Closure Operator-Based Abstraction



The Lattice of Abstractions (2)

- The set $\mathbf{clo}(\wp(\Sigma) \mapsto \wp(\Sigma))$ of all abstractions, i.e. isomorphically, closure operators ρ on the set $\wp(\Sigma)$ of concrete properties is the complete lattice of abstractions for pointwise inclusion [5]:

$$\langle \mathbf{clo}(\wp(\Sigma) \mapsto \wp(\Sigma)),\ \dot{\subseteq},\ \lambda P \cdot P,\ \lambda P \cdot \Sigma,\ \lambda S \cdot \mathbf{ide}(\dot{\cup} S),\ \dot{\cap} \rangle$$

where:

 - the glb $\dot{\cap}$ is the reduced product;
 - $\mathbf{ide}(\rho) = \mathsf{lfp}^{\rho}_{\dot{\subseteq}}\ \lambda f \cdot f \circ f$ is the $\dot{\subseteq}$-least idempotent operator on $\wp(\Sigma)$ $\dot{\subseteq}$-greater than ρ.

---

5 M. Ward, *The closure operators of a lattice*, Annals Math., 43(1942), 191–196.

Local Completion (see [POPL '79, Sec. 9.2])

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Non Distributivity [POPL '79]

- An abstraction ρ is $\cup$-complete or distributive, whenever the union of abstract properties is abstract:

$$\forall S \subseteq \wp(\Sigma) : \bigcup_{P \in S} \rho(P) = \rho(\bigcup_{P \in S} \rho(P))$$

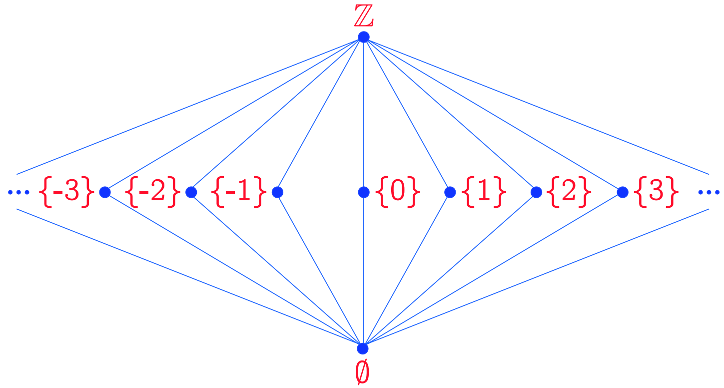
- Hence, the abstract union of abstract properties looses no information with respect to their concrete one;
- Otherwise it is $\cup$-incomplete or non-distributive.

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Example of Non Distributivity [POPL '79]

- Kildall's constant propagation $\langle \{\emptyset, \mathbb{Z}\} \cup \{\{i\} \mid i \in \mathbb{Z}\}, \subseteq \rangle$



is <u>not</u> distributive:

$$\rho(\{1\}) \cup \rho(\{2\}) = \{1, 2\} \neq \mathbb{Z} = \rho(\rho(\{1\}) \cup \rho(\{2\})) \ .$$

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Disjunctive Completion [POPL '79]

- The $\cup$-completion or disjunctive completion $\mathfrak{C}^{\cup}(\overline{\mathcal{A}})$ of an abstract domain $\overline{\mathcal{A}}$ is the smallest distributive abstract domain containing $\overline{\mathcal{A}}$;
- The disjunctive completion adds all missing joins to the abstract domain:

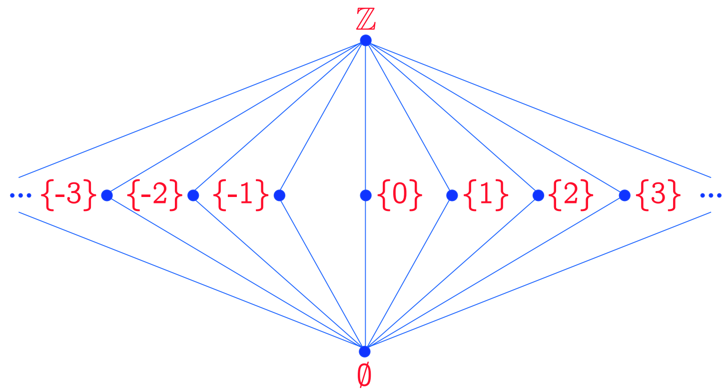
$$\mathfrak{C}^{\cup}(\overline{\mathcal{A}}) = \mathsf{lfp}^{\subseteq}_{\overline{\mathcal{A}}}\, \lambda A \cdot \mathcal{M}\Big(A \cup \{\bigcup_{P \in S} \rho_A(P) \mid \rho_A(\bigcup_{P \in S} \rho_A(P)) \neq \bigcup_{P \in S} \rho_A(P)\}\Big)$$

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Example of Disjunctive Completion [POPL '79]

- Kildall's constant propagation $\langle\{\emptyset, \mathbb{Z}\} \cup \{\{i\} \mid i \in \mathbb{Z}\}, \subseteq\rangle$



  is not distributive;

- The disjunctive completion is $\langle\wp(\mathbb{Z}), \subseteq\rangle$ (i.e. identity abstraction!).

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Local Completeness [POPL '79]

- Given $f \in \wp(\Sigma) \mapsto \wp(\Sigma)$, the abstraction ρ is f-complete iff the f-transformation of abstract properties is abstract:

$$\forall P \in \wp(\Sigma) : \rho \circ f \circ \rho(P) = f \circ \rho(P)$$

- Hence, the abstract transformation of an abstract property looses no information with respect to the concrete one;
- Otherwise ρ is f-incomplete.

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Local Completion [6]

- The f-completion $\mathfrak{C}^f(\overline{\mathcal{A}})$ of an abstract domain $\overline{\mathcal{A}}$ is the smallest f-complete abstract domain containing $\overline{\mathcal{A}}$;
- The local completion adds all missing abstract elements to the abstract domain:

$$\mathfrak{C}^f(\overline{\mathcal{A}}) = \mathsf{lfp}^{\subseteq}_{\overline{\mathcal{A}}}\, \lambda A \cdot \mathcal{M}\Big(A \cup \{f \circ \rho_A(P) \mid \rho_A \circ f \circ \rho_A(P) \neq f \circ \rho_A(P)\}\Big)$$

[6] See other completion methods in:

P. Cousot. Partial Completeness of Abstract Fixpoint Checking, invited paper. In 4^{th} *Int. Symp. SARA '2000*, LNAI 1864, Springer, pp. 1–25, 2000.

R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

Galois Connection-Based Abstraction [POPL '79, Sec. 5.3]

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Correspondance Between Concrete and Abstract Properties

- For closure operators ρ, we have:

$$\rho(P) \subseteq \rho(P') \;\Leftrightarrow\; P \subseteq \rho(P')$$

written:

$$\langle \wp(\Sigma),\ \subseteq\rangle \xleftarrow[\rho]{\;1\;} \!\!\!\!\rightarrow\!\!\!\!\rightarrow \langle \rho(\wp(\Sigma)),\ \subseteq\rangle$$

where 1 is the identity and:

$$\langle \wp(\Sigma),\ \subseteq\rangle \xleftarrow[\alpha]{\;\gamma\;} \!\!\!\!\rightarrow\!\!\!\!\rightarrow \langle \overline{\mathcal{D}},\ \sqsubseteq\rangle$$

means that $\langle \alpha,\ \gamma\rangle$ is a Galois connection:

– $\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \;\Leftrightarrow\; P \subseteq \gamma(\overline{P})$;

– α is onto (equivalently $\alpha \circ \gamma = 1$ or γ is one-to-one).

Abstract Domain

- Abstract Domain: an isomorphic representation $\overline{\mathcal{D}}$ of the set $\overline{\mathcal{A}} \subsetneq \wp(\Sigma) = \rho(\wp(\Sigma))$ of abstract properties (up to some order-isomorphism ι).

Galois Surjection [7]

- We have the Galois surjection:

$$\langle \wp(\Sigma),\ \subseteq\rangle \xleftarrow[\iota\circ\rho]{\iota^{-1}} \!\!\!\twoheadrightarrow \langle \overline{\mathcal{D}},\ \sqsubseteq\rangle$$

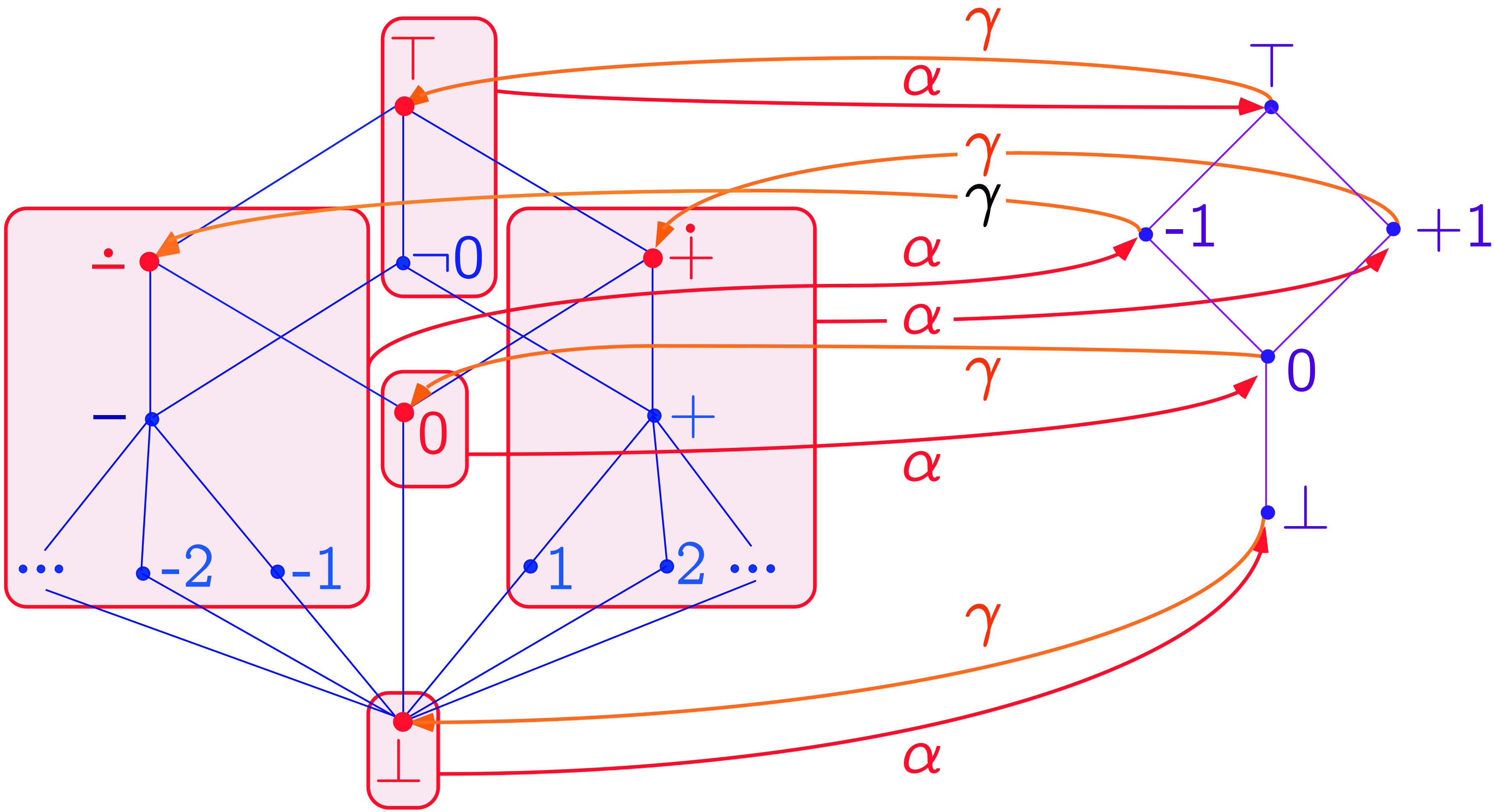
- More generally:

$$\langle \wp(\Sigma),\ \subseteq\rangle \xleftarrow[\alpha]{\gamma} \!\!\!\twoheadrightarrow \langle \overline{\mathcal{D}},\ \sqsubseteq\rangle$$

denoting (again) the fact that:
  - $\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \ \Leftrightarrow\ P \subseteq \gamma(\overline{P})$;
  - α is onto (equivalently $\alpha \circ \gamma = 1$ or γ is one-to-one).

[7] Also called Galois insertion since γ is injective.

Example of Galois Surjection-Based Abstraction



Galois Connection

- Relaxing the condition that α is onto:

$$\langle \wp(\Sigma),\ \subseteq\rangle \xleftrightarrow[\alpha]{\gamma} \langle \overline{\mathcal{D}},\ \sqsubseteq\rangle$$

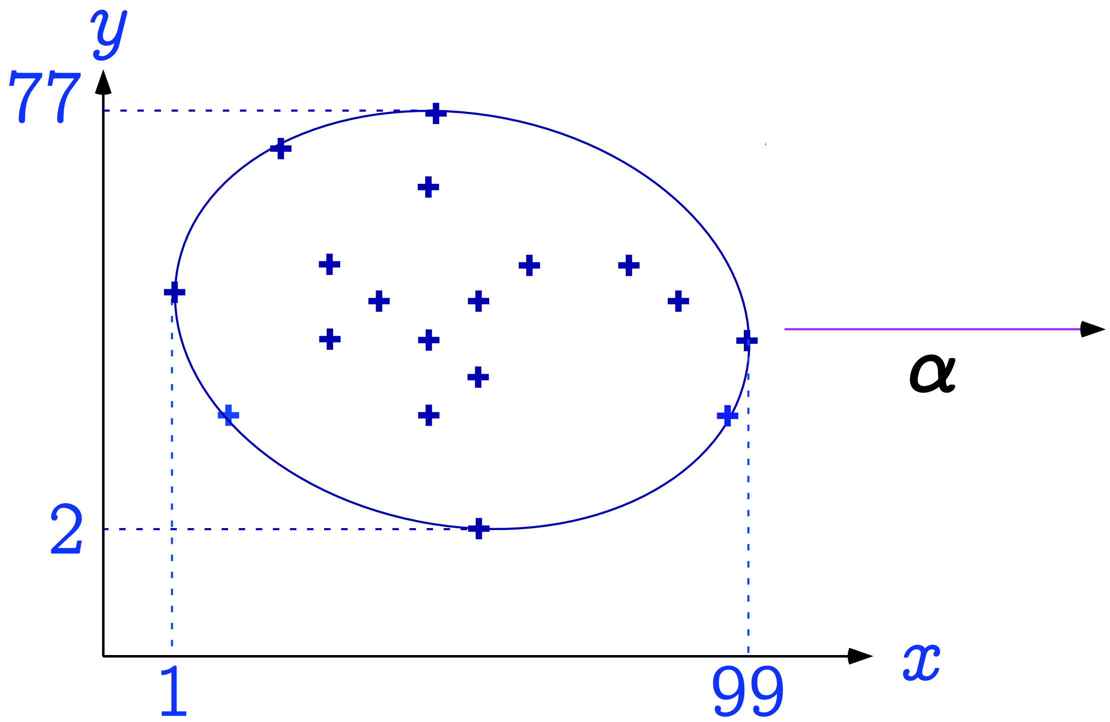
  that is to say:

$$\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \iff P \subseteq \gamma(\overline{P});$$

- i.e. ρ is now $\gamma \circ \alpha$;

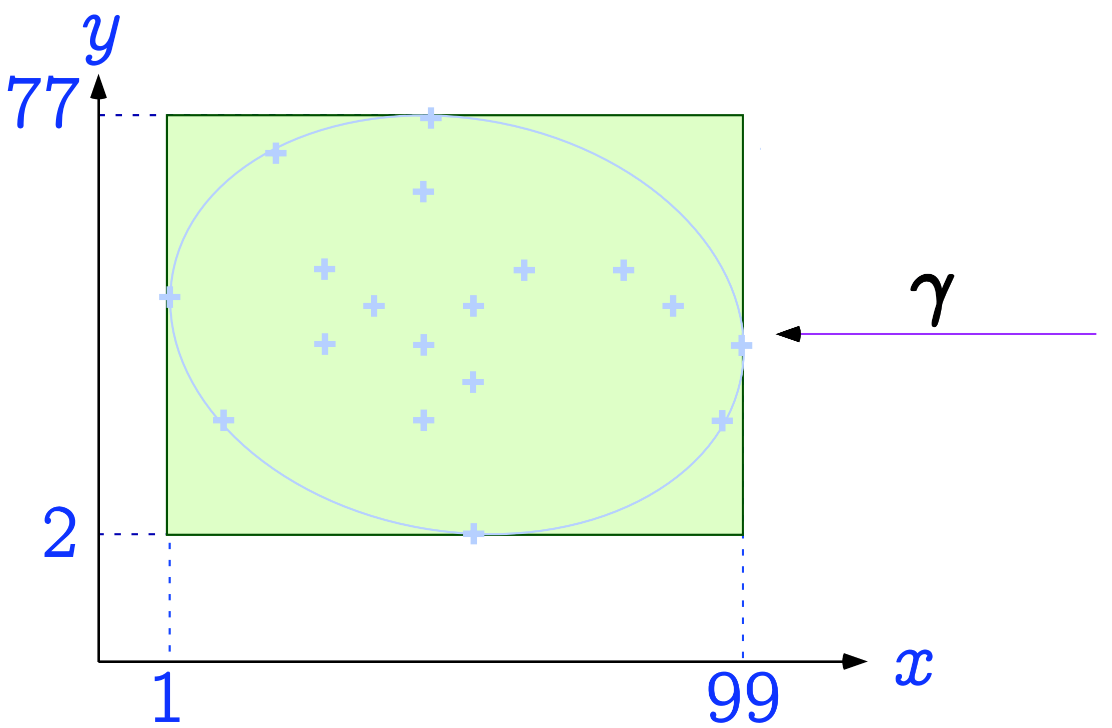
  We can now have different representations of the same abstract property.

Abstraction α



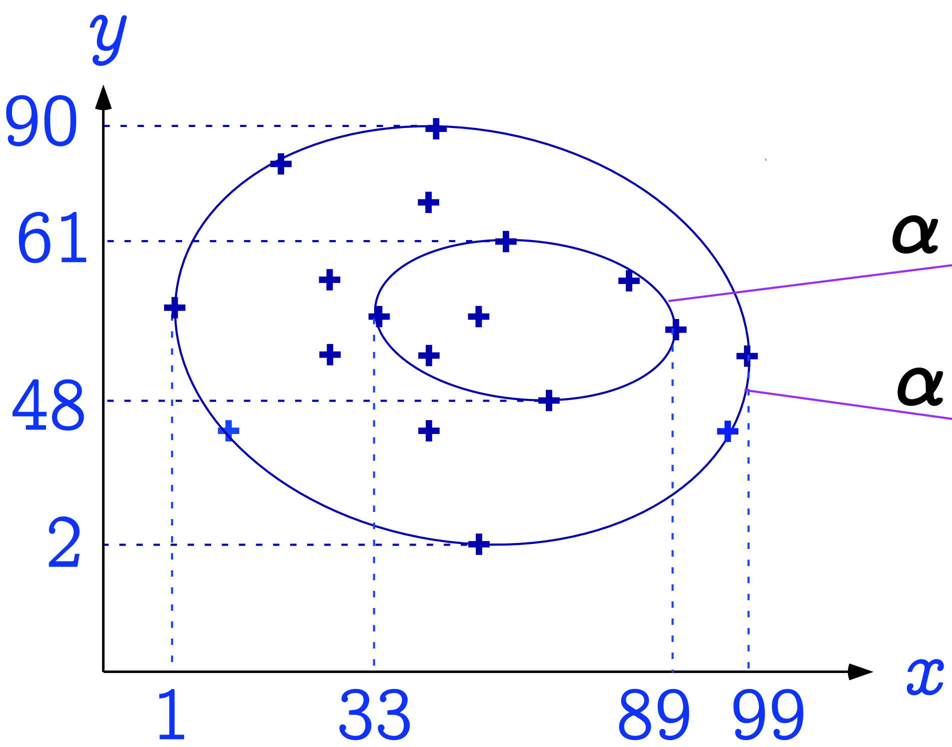
$$\{x : [1, 99], y : [2, 77]\}$$

Concretization γ



$\{x : [1, 99], y : [2, 77]\}$

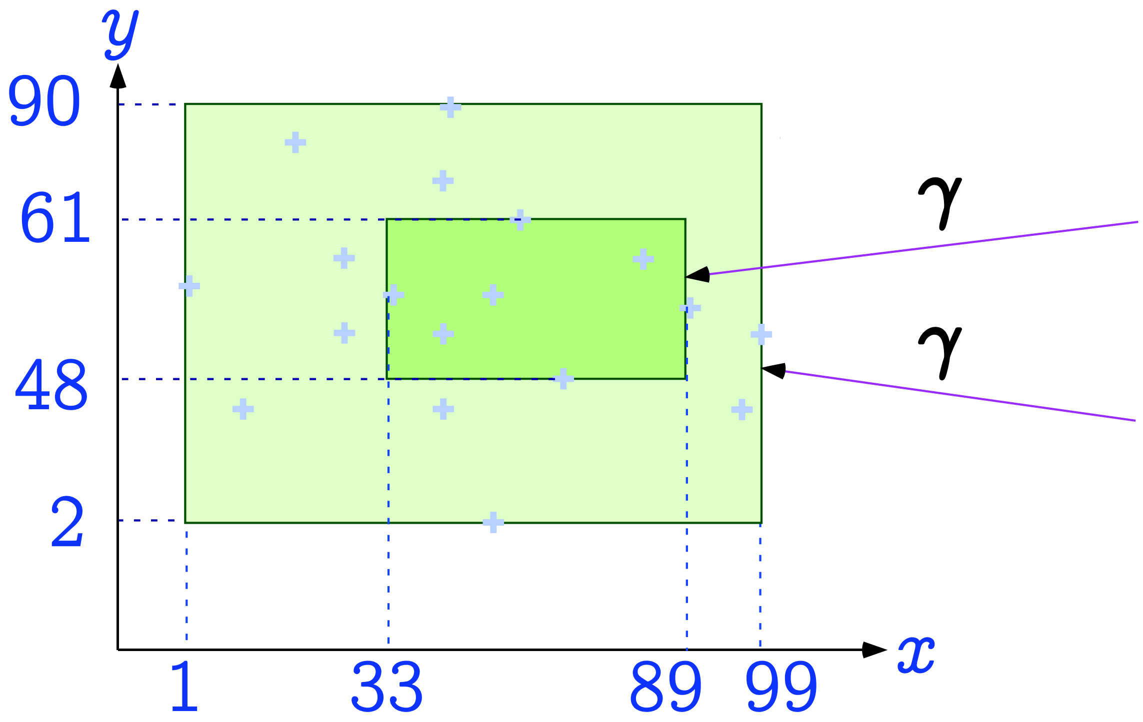
The Abstraction α is Monotone



$$\{x : [33, 89], y : [48, 61]\}$$
$$\sqsubseteq$$
$$\{x : [1, 99], y : [2, 90]\}$$

$$X \subseteq Y \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$$

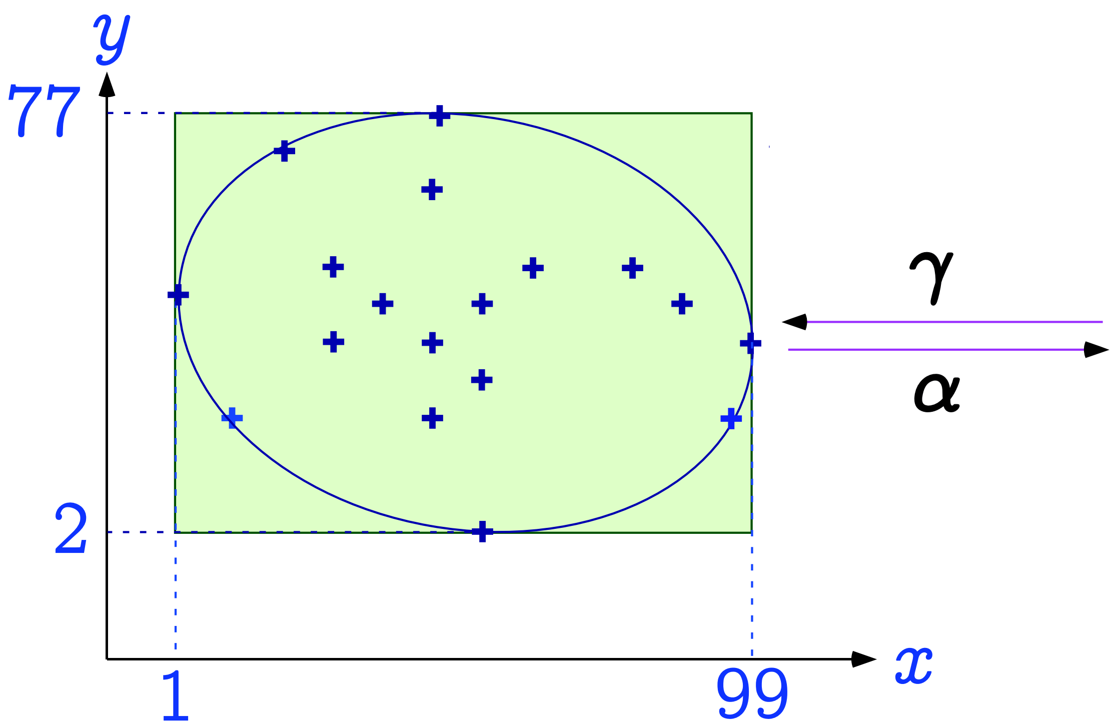
The Concretization γ is Monotone



$$\{x : [33, 89], y : [48, 61]\}$$
$$\sqsubseteq$$
$$\{x : [1, 99], y : [2, 90]\}$$

$$X \sqsubseteq Y \Rightarrow \gamma(X) \subseteq \gamma(Y)$$

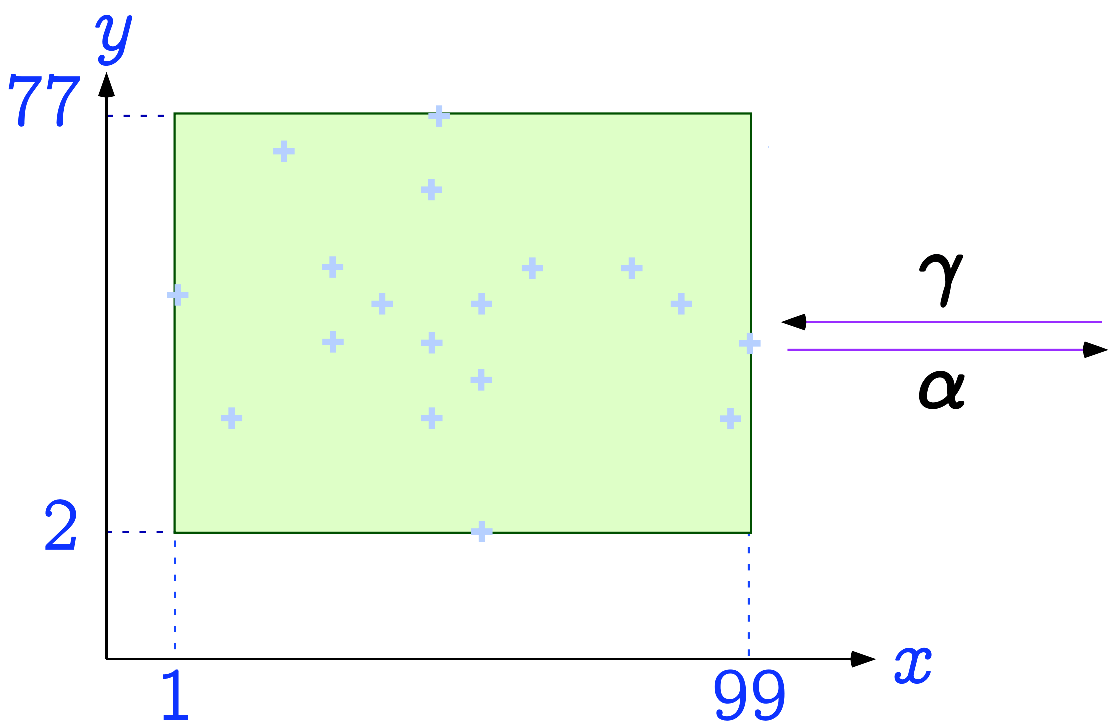
The $\gamma \circ \alpha$ Composition is Extensive



$\{x : [1, 99], y : [2, 77]\}$

$$X \subseteq \gamma \circ \alpha(X)$$

The $\alpha \circ \gamma$ Composition is Reductive



$$\{x : [1, 99], y : [2, 77]\}$$
$$=/\sqsubseteq$$
$$\{x : [1, 99], y : [2, 77]\}$$

$$\alpha \circ \gamma(Y) =/\sqsubseteq Y$$

Composition of Galois Connections

The composition of Galois connections:

$$\langle L,\ \leq\rangle \xleftarrow[\alpha_1]{\gamma_1} \xrightarrow{} \langle M,\ \sqsubseteq\rangle$$

and:

$$\langle M,\ \sqsubseteq\rangle \xleftarrow[\alpha_2]{\gamma_2} \xrightarrow{} \langle N,\ \preceq\rangle$$

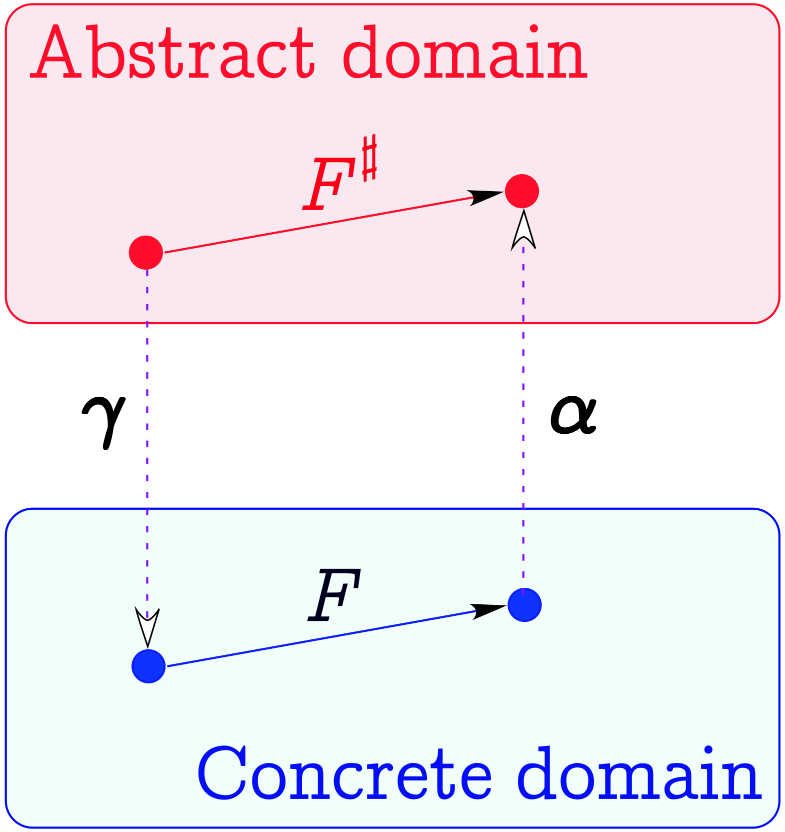
is a Galois connection:

$$\langle L,\ \leq\rangle \xleftarrow[\alpha_2\circ\alpha_1]{\gamma_1\circ\gamma_2} \xrightarrow{} \langle N,\ \preceq\rangle$$

Function Abstraction [POPL '79, Sec. 7.2]

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.



Function Abstraction

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

i.e. $F^\sharp = \rho \circ F$

$$\langle P,\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle Q,\ \sqsubseteq\rangle \Rightarrow$$

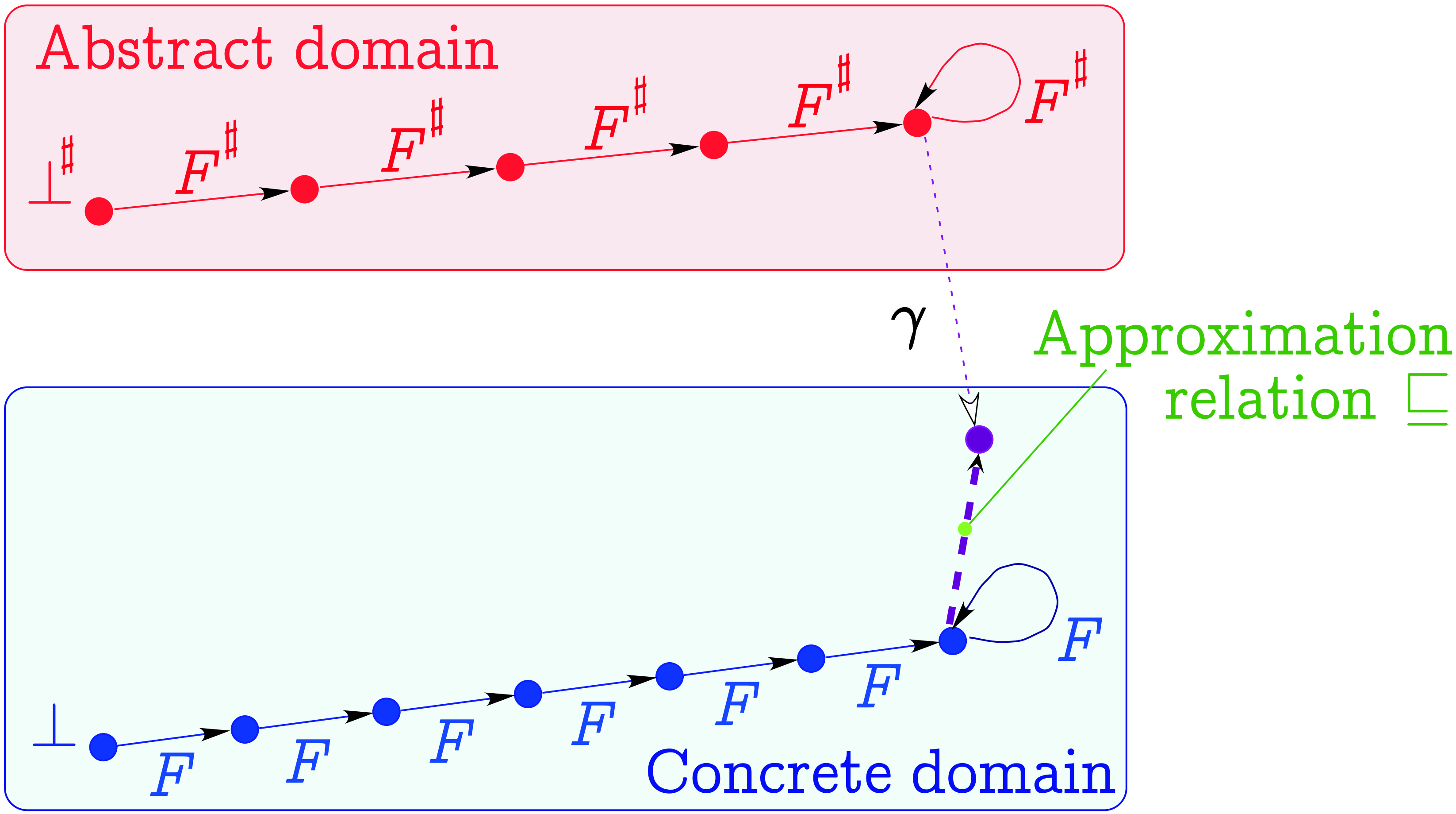
$$\langle P \overset{\text{mon}}{\longmapsto} P,\ \dot{\subseteq}\rangle \underset{\lambda F \cdot \alpha\circ F\circ\gamma}{\overset{\lambda F^\sharp \cdot \gamma\circ F^\sharp\circ\alpha}{\leftrightarrows}} \langle Q \overset{\text{mon}}{\longmapsto} Q,\ \dot{\sqsubseteq}\rangle$$

Fixpoint Abstraction [POPL '79, Sec. 7.1]

Reference

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Approximate Fixpoint Abstraction



$$\alpha(\mathsf{lfp}\ F) \sqsubseteq \mathsf{lfp}\ F^\sharp$$

Approximate/Exact Fixpoint Abstraction

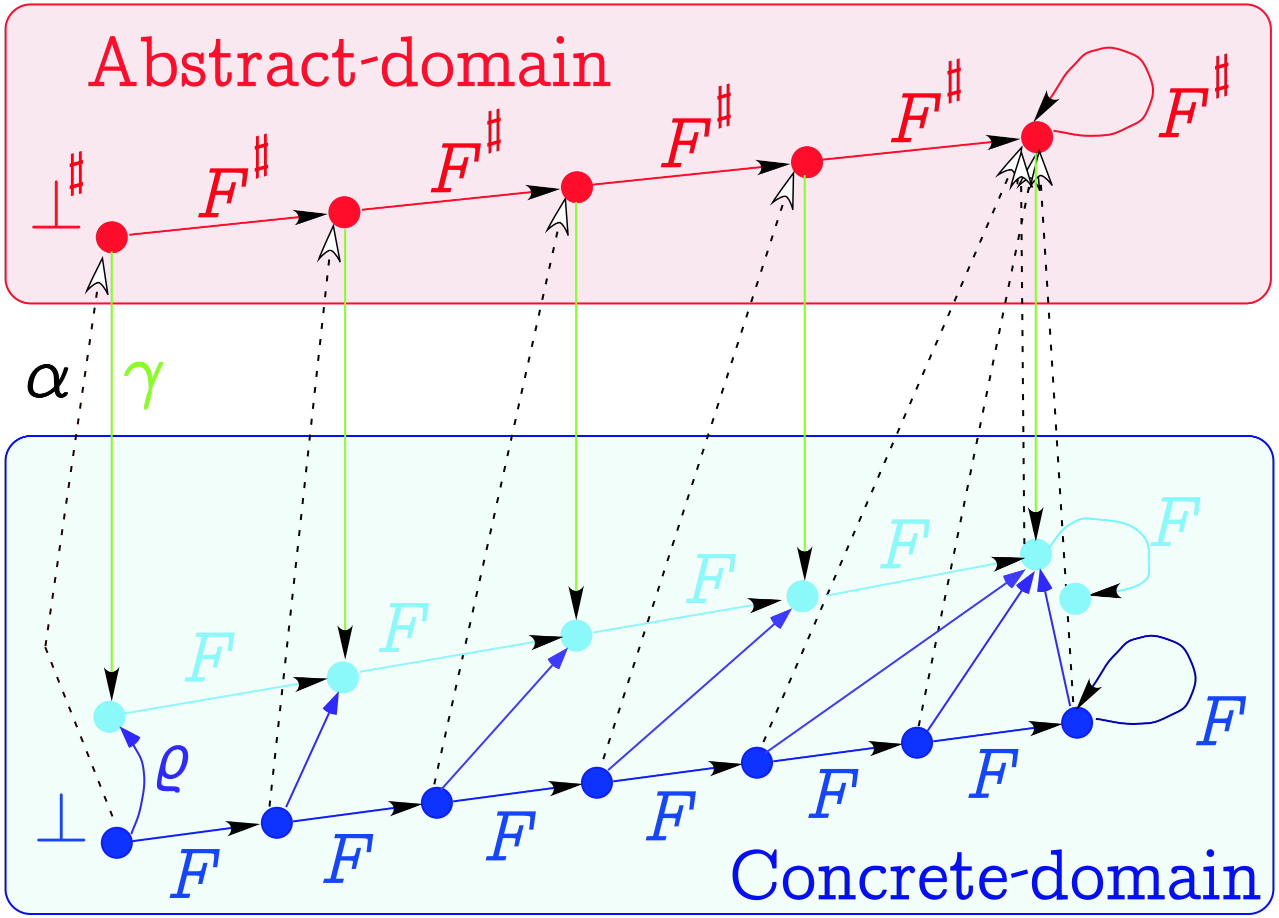
Exact Abstraction:

$$\alpha(\mathsf{lfp}\ F) = \mathsf{lfp}\ F^\sharp$$

Approximate Abstraction:

$$\alpha(\mathsf{lfp}\ F) \sqsubseteq^\sharp \mathsf{lfp}\ F^\sharp$$

Exact Fixpoint Abstraction



$$F \circ \gamma = \gamma \circ F^\sharp \Rightarrow \alpha(\mathsf{lfp}\, F) = \mathsf{lfp}\, F^\sharp$$

Fixpoint Completion

- We want to prove $\mathsf{lfp}\, F \subseteq \gamma(I)$ i.e. $\alpha(\mathsf{lfp}\, F) \sqsubseteq^\sharp I$
- The abstraction is in general incomplete so $\mathsf{lfp}\, F^\sharp \not\sqsubseteq^\sharp I$
- Hence we look for the most abstract abstraction $\bar{\alpha}$ which is more precise than α and is fixpoint complete:
$$\bar{\alpha}(\mathsf{lfp}\, F) = \mathsf{lfp}\, \bar{F}^\sharp \qquad \text{where} \qquad \bar{F}^\sharp = \bar{\alpha} \circ F \circ \bar{\gamma}$$
- This is sound since $\mathsf{lfp}\, \bar{F}^\sharp \sqsubseteq^\sharp I$ implies $\alpha(\mathsf{lfp}\, F) \sqsubseteq^\sharp I$ that is $\mathsf{lfp}\, F \subseteq \gamma(I)$
- This is complete since $\mathsf{lfp}\, F \subseteq \bar{\gamma}(I) = \gamma(I)$ so $\bar{\alpha}(\mathsf{lfp}\, F) \sqsubseteq^\sharp I$ i.e. $\mathsf{lfp}\, \bar{F}^\sharp \sqsubseteq^\sharp I$ is now provable in the abstract.

Local F-Completion

A sufficient condition to ensure exact fixpoint abstraction $\bar{\alpha}(\mathsf{lfp}\, F) = \mathsf{lfp}\, \bar{F}^\sharp$ is:

- Local completeness that is $F \circ \bar{\gamma} = \bar{\gamma} \circ \bar{F}^\sharp$, or $F \circ \bar{\rho} = \bar{\rho} \circ F \circ \bar{\rho}$ where $\bar{\rho} = \bar{\gamma} \circ \bar{\alpha}$
- Therefore F-local completion can be used to determine $\bar{\rho}$ (i.e. $\langle \bar{\alpha}, \bar{\gamma} \rangle$) from $\rho = \gamma \circ \alpha$ by a fixpoint computation.

Notes:

- The F-local completion can be restricted to the fixpoint iterates;
- In general, the completed domain does not satisfy the ascending chain condition (see the previous constant propagation example).

Application to Predicate Abstraction

Reference

[1] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Proc. 9th Int. Conf. CAV '97*,LNCS 1254, pp. 72–83. Springer, 1997.

The Structure of Program States

- States: $\Sigma = \mathcal{L} \times \mathcal{M}$
- Program points/labels: $\mathcal{L}$ is finite
- Variables: $\mathbb{X}$ is finite (for a given program)
- Set of values: $\mathcal{V}$
- Memory states: $\mathcal{M} = \mathbb{X} \mapsto \mathcal{V}$

Local Versus Global Assertions

- Isomorphism between global and local assertions:

$$\langle \wp(\mathcal{L} \times \mathcal{M}), \subseteq \rangle \underset{\alpha_\downarrow}{\overset{\gamma_\downarrow}{\leftrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathcal{M}), \dot{\subseteq} \rangle$$

where:

$$\alpha_\downarrow(P) = \lambda \ell \cdot \{m \mid \langle \ell, m \rangle \in P\}$$
$$\gamma_\downarrow(Q) = \{\langle \ell, m \rangle \mid \ell \in \mathcal{L} \wedge m \in Q_\ell\}$$

and $\dot{\subseteq}$ is the pointwise ordering:

$Q \mathrel{\dot{\subseteq}} Q'$ if and only if $\forall \ell \in \mathcal{L} : Q_\ell \subseteq Q'_\ell$.

Syntactic Predicates

- a set $\mathbb{P}$ of syntactic predicates p such that:

$$\forall S \subseteq \mathbb{P} : (\bigwedge S) \in \mathbb{P}$$

- an interpretation $\mathcal{I} \in \mathbb{P} \mapsto \wp(\mathcal{M})$ such that:

$$\forall S \subseteq \mathbb{P} : \mathcal{I}(\bigwedge S) = \bigcap_{p \in S} \mathcal{I}[\![p]\!]$$

- It follows that $\{\mathcal{I}[\![p]\!] \mid p \in \mathbb{P}\}$ is a Moore family.

Predicate Abstraction

A memory state property $Q \in \wp(\mathcal{M})$ is approximated by the subset of predicates p of $\mathbb{P}$ which holds when Q holds (formally $Q \subseteq \mathcal{I}[\![p]\!]$). This defines a Galois connection:

$$\langle \wp(\mathcal{M}), \subseteq \rangle \underset{\alpha_{\mathbb{P}}}{\overset{\gamma_{\mathbb{P}}}{\leftrightarrows}} \langle \wp(\mathbb{P}), \supseteq \rangle$$

where:

$$\alpha_{\mathbb{P}}(Q) \stackrel{\text{def}}{=} \{p \in \mathbb{P} \mid Q \subseteq \mathcal{I}[\![p]\!]\}$$

$$\gamma_{\mathbb{P}}(P) \stackrel{\text{def}}{=} \bigcap \{\mathcal{I}[\![p]\!] \mid p \in P\}$$

Pointwise Extension to All program Points

By pointwise extension, we have for all program points:

$$\langle \mathcal{L} \mapsto \wp(\mathcal{M}),\ \dot{\subseteq} \rangle \underset{\dot{\alpha}_{\mathbb{P}}}{\overset{\dot{\gamma}_{\mathbb{P}}}{\leftrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathbb{P}),\ \dot{\supseteq} \rangle$$

where:

$$\dot{\alpha}_{\mathbb{P}}(Q) = \lambda \ell \cdot \alpha_{\mathbb{P}}(Q_\ell)$$

$$\dot{\gamma}_{\mathbb{P}}(P) = \lambda \ell \cdot \gamma_{\mathbb{P}}(P_\ell)$$

$$P \mathrel{\dot{\supseteq}} P' = \forall \ell \in \mathcal{L} : P_\ell \supseteq P'_\ell$$

Boolean Encoding

- $\mathbb{P} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$ is finite
- $\mathbb{B} = \{tt, ff\}$ is the set of booleans with $ff \Rightarrow ff \Rightarrow tt \Rightarrow tt$
- We can use a boolean encoding of subsets of $\mathbb{P}$:

$$\langle \wp(\mathbb{P}),\ \supseteq \rangle \underset{\alpha_b}{\overset{\gamma_b}{\leftrightarrows}} \langle \prod_{i=1}^{k} \mathbb{B},\ \dot{\Leftarrow} \rangle$$

where:

$$\alpha_b(P) = \prod_{i=1}^{k} (\mathfrak{p}_i \in P)$$

$$\gamma_b(Q) = \{\mathfrak{p}_i \mid 1 \leq i \leq k \wedge Q_i\}$$

$$Q \dot{\Leftarrow} Q' = \forall i : 1 \leq i \leq k : Q_i \Leftarrow Q'_i$$

Pointwise Extension to All program Points

By pointwise extension, we have for all program points:

$$\langle \mathcal{L} \mapsto \wp(\mathbb{P}),\ \dot{\supseteq} \rangle \underset{\dot{\alpha}_b}{\overset{\dot{\gamma}_b}{\leftleftarrows\!\!\!\twoheadrightarrow}} \langle \mathcal{L} \mapsto \prod_{i=1}^{k} \mathbb{B},\ \dot{\Leftarrow} \rangle$$

where:

$$\dot{\alpha}_b(P) = \lambda \ell \cdot \alpha_b(P_\ell)$$

$$\dot{\gamma}_b(Q) = \lambda \ell \cdot \gamma_b(Q_\ell)$$

$$Q \dot{\Leftarrow} Q' = \forall \ell \in \mathcal{L} : Q_\ell \Leftarrow Q'_\ell$$

Composition: Pointwise Boolean Encoded Predicate Abstraction

By composition, we get:

$$\langle \wp(\mathcal{L} \times \mathcal{M}),\ \subseteq\rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{L} \mapsto \prod_{i=1}^{k} \mathbb{B},\ \dot{\Leftarrow}\rangle$$

where:

$$\alpha(P) = \dot{\alpha}_b \circ \dot{\alpha}_{\mathbb{P}} \circ \alpha_{\downarrow}(P)$$

$$\gamma(Q) = \gamma_{\downarrow} \circ \dot{\gamma}_{\mathbb{P}} \circ \dot{\gamma}_b(Q)$$

Abstract Predicate Transformer (Sketchy)

$$\alpha \circ \text{post}[\![\mathtt{X:=E}]\!] \circ \gamma(\bigwedge_{i=1}^{n} q_i)$$

where $\{q_1, \ldots, q_n\} \subseteq \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$

$$= \alpha \circ \text{post}[\![\mathtt{X:=E}]\!](\bigcap_{i=1}^{n} \mathcal{I}[\![q_i]\!])$$ def. γ

$$= \alpha(\{\rho[\mathtt{X}/[\![\mathtt{E}]\!]\rho] \mid \rho \in \bigcap_{i=1}^{n} \mathcal{I}[\![q_i]\!]\})$$ def. $\text{post}[\![\mathtt{X:=E}]\!]$

$$= \alpha(\bigcap_{i=1}^{n} \mathcal{I}[\![q_i[\mathtt{X}/\mathtt{E}]]\!])$$ def. substitution

$$= \bigwedge\{\mathfrak{p}_j \mid \mathcal{I}[\![q_i[\mathtt{X}/\mathtt{E}] \Rightarrow \mathfrak{p}_j]\!]\}$$ def. α

$$\Rightarrow \bigwedge\{\mathfrak{p}_j \mid \text{theorem\_prover}[\![q_i[\mathtt{X}/\mathtt{E}] \Rightarrow \mathfrak{p}_j]\!]\}$$

since $\text{theorem\_prover}[\![q_i[\mathtt{X}/\mathtt{E}] \Rightarrow \mathfrak{p}_j]\!]$ implies $\mathcal{I}[\![q_i[\mathtt{X}/\mathtt{E}] \Rightarrow \mathfrak{p}_j]\!]$

Predicate Abstraction Completion

- Principle:
  - Start from $\mathbb{P} = \{\mathtt{true}\}$ (or some more refined abstraction such as intervals)
  - Iteratively repeat local completion until verification done
- A few convincing practical experiences e.g. [2]
- Can this scale up for more precise abstractions?

Reference

[2] T. Ball, R. Majumdar, T.D. Millstein, and S.K. Rajamani. Automatic predicate abstraction of C programs. In *Proc. ACM SIGPLAN 2001 Conf. PLDI. ACM SIGPLAN Not. 36(5)*, pages 203–213. ACM Press, June 2001.

A Practical Application of Abstract Interpretation to the Verification of Safety Critical Embedded Software

Reference

[3] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566, pages 85–108. Springer, 2002.

General-Purpose versus Specializable Static Program Analysis

General-Purpose Static Program Analyzers

- To handle infinitely many programs for non-trivial properties, a general-purpose analyser must use an infinite abstract domain[8];
- 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.

---

[8] 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);
- This leads to *very efficient analyzers* with *zero (or almost no) false alarm* even for large programs.

The Class of Periodic Synchronous Programs

declare `volatile input, state and output variables;`

`initialize state variables;`

loop forever

```
- read volatile input variables,
- compute output and state variables,
- write to volatile output variables;
```

wait for next clock tick;

end loop

- The only allowed interrupts are clock ticks;
- Execution time of loop body less than a clock tick [4].

Reference

[4] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. *ESOP (2001)*, LNCS 2211, 469–485.

First Experience

Reference

[5] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566, pages 85–108. Springer, 2002.

A First Experience of Parametric Specializable Static Program Analyzers

- C programs: safety critical embedded real-time synchronous software for non-linear control of complex systems;
- 10 000 LOCs, 1300 global variables (booleans, integers, floats, arrays, macros, non-recursive procedures);
- Implicit specification: absence of runtime errors (no integer/floating point arithmetic overflow, no array bound overflow);
- Comparative results (commercial software):
  - 70 false alarms, 2 days, 500 Megabytes;

- Initial design: 2h, 110 false alarms (general purpose interval-based analyzer);
- Main redesign:
  - Reduced product with 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 a Simple Idea That Does Not Scale Up

- Represent abstract environments $\bar{\mathcal{M}} = \mathbb{X} \mapsto \bar{\mathcal{D}}$ where $\bar{\mathcal{D}}$ is the abstract domain as arrays/functional arrays;
- $\mathcal{O}(1)$ to access/change the abstract value of an identifier <u>but</u>, most variables are locally unchanged so a lot of time is lost in unions $P \cup P = P$ and widenings $P \nabla P = P$;
- Solution: shared balanced binary tree (`maps` in CAML);
- $\mathcal{O}(\ln n)$ among n to access/change the abstract value of an identifier <u>but</u>, most of the tree is unchanged in unions and widenings (gained factor 7 in time).

Example 1 of refinement: widenings

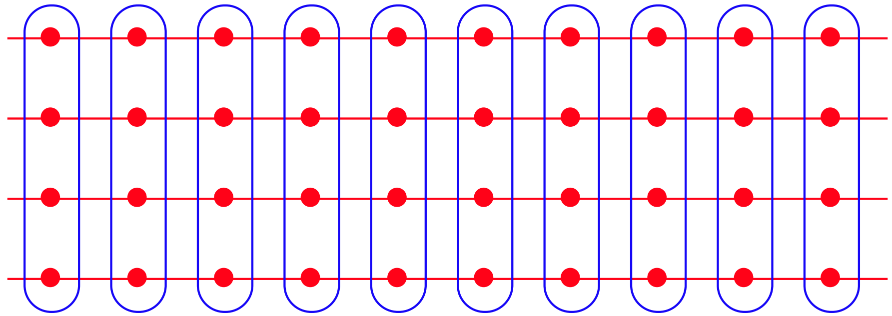
- Interval analysis with naïve widening to $\pm\infty$ can be less precise than sign analysis;
- For example $[2, +\infty] \mathbin{\nabla} [1, +\infty] = [-\infty, +\infty]$ whereas sign analysis would first try $[0, +\infty]$ (i.e. "positive");
- Solution: widening with threshold set.

Widening with threshold set

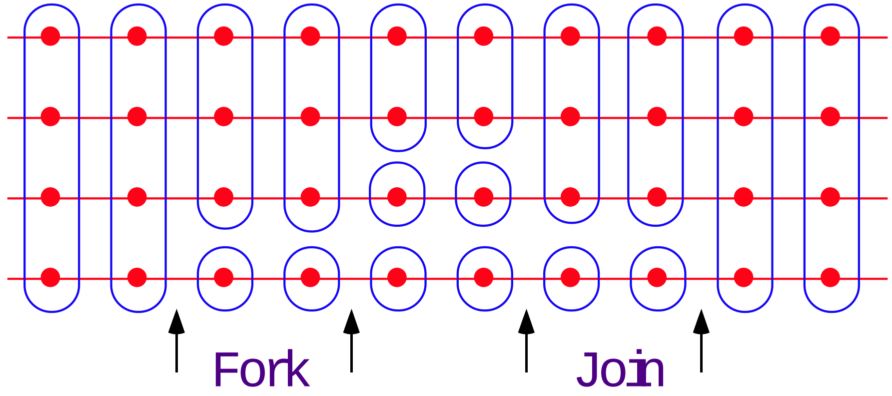
- The threshold set T is a finite set of numbers (plus $+\infty$ and $-\infty$),
- $$[a, b] \nabla_T [a', b'] = [\mathit{if}\ a' < a\ \mathit{then}\ \max\{\ell \in T \mid \ell \leq a'\}\ \mathit{else}\ a,\ \mathit{if}\ b' > b\ \mathit{then}\ \min\{h \in T \mid h \geq b'\}\ \mathit{else}\ b]\ .$$
- Examples (intervals):
  - sign analysis: $T = \{-\infty, 0, +\infty\}$;
  - strict sign analysis: $T = \{-\infty, -1, 0, +1, +\infty\}$;
- T is a parameter of the analysis.

Example 2 of refinement: trace partitionning

Control point partitionning:

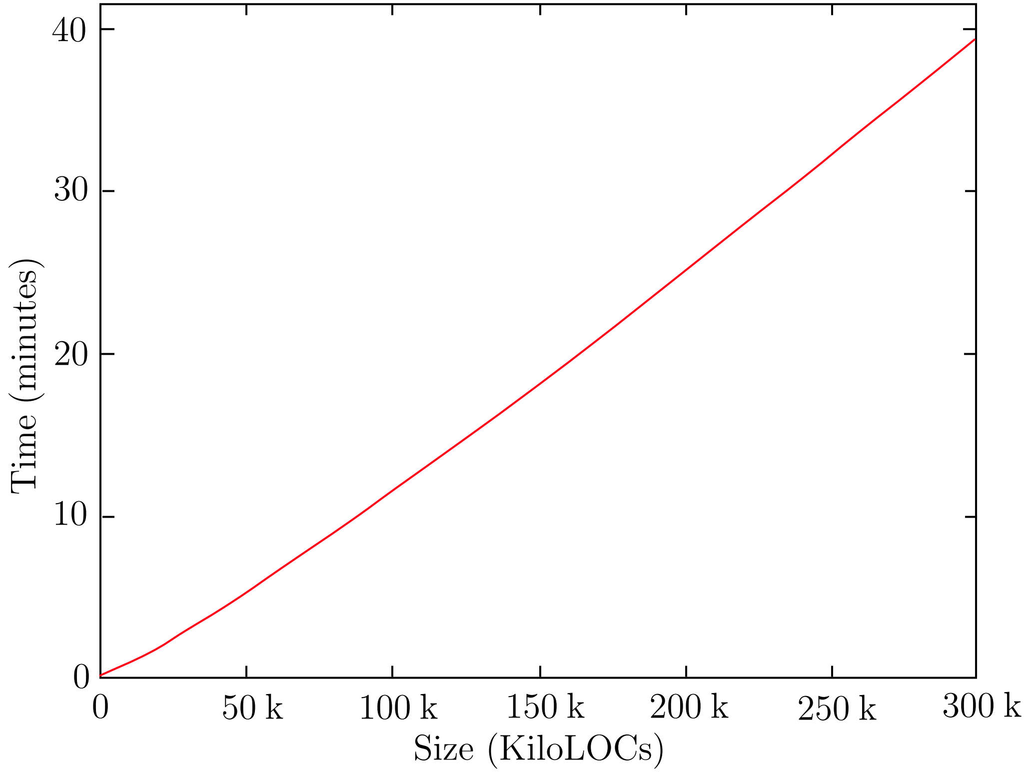


Trace partitionning:



Performance: Space and Time

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



Second Experience

A Second Experience of Parametric Specializable Static Program Analyzers

- Same C programs for synchronous non-linear control of very complex systems;
- 132,000 lines of C, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays;
- Same implicit specification: absence of runtime errors;
- Analyzer of first experience: 30mn, 1,200 false alarms;

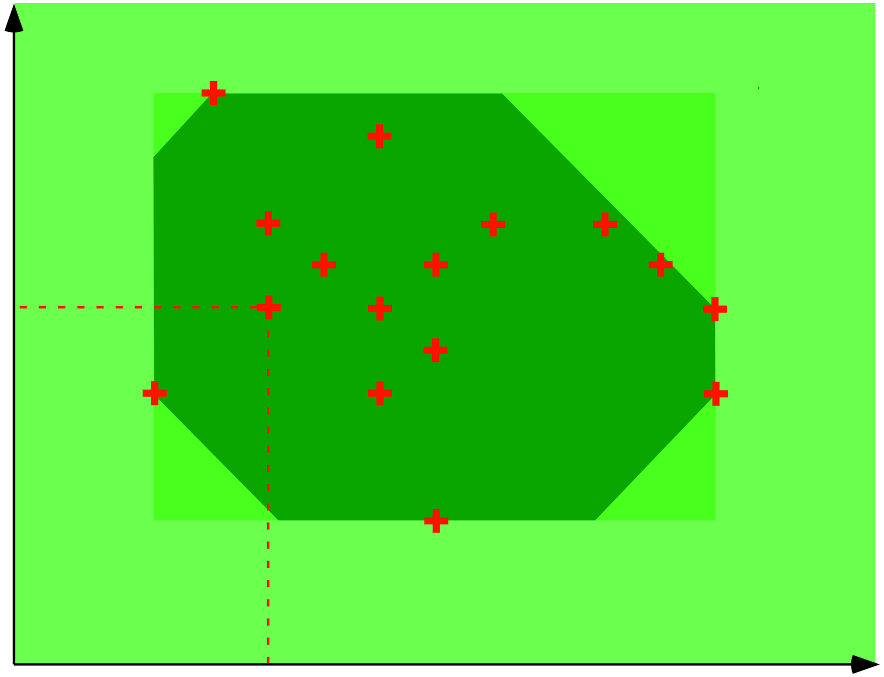
Some Difficulties (Among Others)

- Ignoring the value of any variable at any program point creates false alarms;
- Most precise abstract domains (e.g. polyhedra [6]) simply do not scale up;
- Tracing the fixpoint computation will produce huge log files crashing usual text editors;

Reference

[6] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th POPL*, pages 84–97, Tucson, AZ, 1978. ACM Press.

Example of Refinement: Octagons



$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 78 \\ 1 \leq y \leq 20 \\ x - y \leq 03 \end{cases}$$

Reference

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

Difficulty 1 with Octagons

- Most operations are $\mathcal{O}(n^2)$ in space and $\mathcal{O}(n^3)$ in time, so does not scale up;
- Solution:
  - Parameterize with packs of variables/program points where to use octagons,
  - Automatize the determination of the packs by experimentation (to eliminate the useless ones);

Difficulty 2 with Octagons [9]

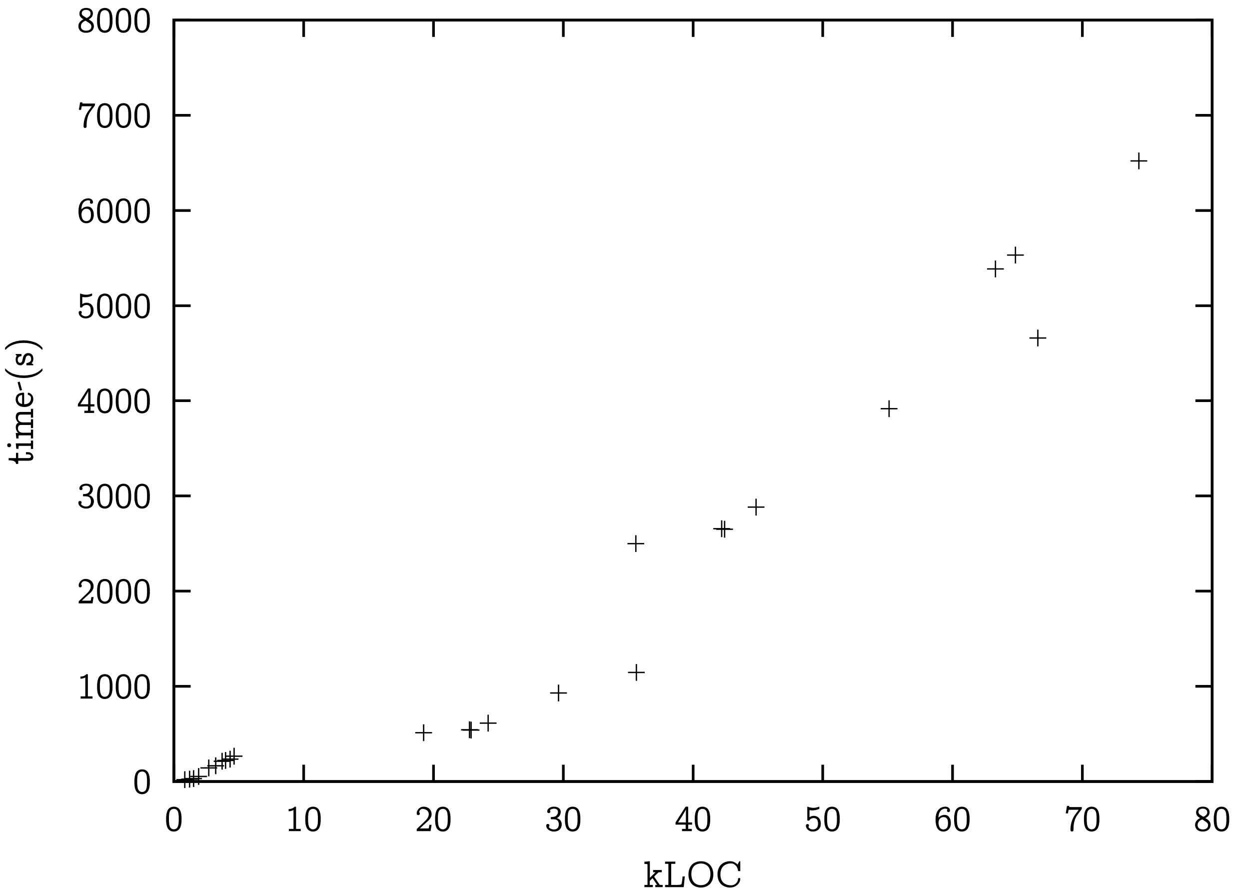
- Must be correct with respect to the IEEE 754 floating-point arithmetic norm;
- Solution: sophisticated algorithmic to correctly handle concrete and abstract rounding errors

9 An opened problem with polyhedra.

Second Experience (Preliminary) Report

- Comparative results (commercial software):
  - 4,200 (false?) alarms, 5 days;
- Results: 20 (false?) alarms, 1h30mn, 500 Megabytes.

Benchmarks


8000
7000
6000
5000
4000
3000
2000
1000
0
time (s)
0
10
20
30
40
50
60
70
80
kLOC

Would Automatic Predicate Abstraction Have Done It?

Yes, Predicate Abstraction Can Do It!

- Yes, because their exists a finite domain that can do it (as proved in [SARA '00])!
- So this finite abstract domain can be encoded by predicate abstraction!

Reference

[SARA '00] P. Cousot. Partial Completeness of Abstract Fixpoint Checking, invited paper. In 4^{th} *Int. Symp. SARA '2000*, LNAI 1864, Springer, pp. 1–25, 2000.

Yes, But What About Automatic Predicate Abstraction!

- Yes, because one can use a widening on the concrete domain which, for a *given* program, will extract from this infinite domain a finite, subset which can be used as an abstract domain for a finite analysis as proved in [PLILP '92]!
- So this finite abstract domain can be encoded by predicate abstraction!

This is good old theory, but so what in practice?

Reference

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

Problems of Semantics

- For C programs, the prover which is used to automatically design abstract transfer functions has to take the machine-level semantics into account;
- For example:
  - floating-point arithmetic with rounding errors as opposed to real numbers (e.g. $A+B < C \wedge D-B \leq C \not\Rightarrow A+D < 2 \times C$);
  - ESC is simply unsound with respect to modulo arithmetics [8].

Reference

[8] Flanagan, C., Leino, K.R.M., Lillibridge, M., Nelson, G., Saxe, J., Stata, R.: *Extended static checking for Java*. PLDI'02, ACM SIGPLAN Not. 37(5), (2002) 234–245.

Prognosticating a State Explosion Problem

The main loop invariant: a textual file over 4.5 Mb with

- 6,900 boolean interval assertions ($x \in [0; 1]$)
- 9,600 interval assertions ($x \in [a; b]$)
- 25,400 clock assertions ($x + \mathrm{clk} \in [a; b] \wedge x - \mathrm{clk} \in [a; b]$)
- 19,100 additive octagonal assertions ($a \leq x + y \leq b$)
- 19,200 subtractive octagonal assertions ($a \leq x - y \leq b$)
- 100 decision trees
- etc, ...

involving over 16,000 floating point constants (only 550 appearing in the program text) $\times$ 75,000 LOCs.

Conclusion

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 (as in software model-checking)*;
- The abstraction can be tailored to classes of programs so as to design *very efficient analyzers* with *almost no and even zero-false alarm*.

Conclusion on Verification by Abstraction

Beyond Static Analysis, Abstract Interpretation is Efficacious for Automatic Verification in the Large.

THE END

More references at URL www.di.ens.fr/~cousot.