Verification of Safety-Critical Control-Command Sofware 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

Radhia COUSOT
École Polytechnique
91128 Palaiseau cedex
France
Radhia.Cousot@polytechnique.fr
stix.polytechnique.fr/~rcousot

CS Dept., Courant Inst. of Math. Sci., NYU
New York, NY, 13 May 2004

Deficiencies of Formal Methods

Talk Outline

Automated Verification of Infinite-State Systems

- The automated verification of infinite-sate systems has made considerable progress these last ten years
- It is yet far from being a common industrial practice
- This might be that most available prototypes and tools are inappropriate
- These prototypes and tools aim at debugging whereas we need automated verification

Defects of Available Prototypes and Tools

- Manual (e.g. require end-users to provide manually a simple-enough model of the complex system), and/or
- User-unfriendly (e.g. require complex interactions with end-users), and/or
- Trivial (e.g. consider immediate essentially syntactic program properties) and/or
- Incorrect/unsound (e.g. do not explore the complete space of executions and so may forget about potential problems at run-time), and/or

- Inefficient (some may not terminate at all but by exhaustion of time/memory resources), and/or
- Imprecise (leading to too many false alarms that is spurious warnings on potential problems that can never occur at run-time).

Can we do better?

A Few Elements of Abstract Interpretation

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.

A Model of Computer Programs

- Syntax : a well-founded set of programs $\langle \mathbb{P}, \prec \rangle$ where $\prec$ is the "strict immediate subcomponent" relation ;
- Semantics of $P \in \mathbb{P}$:
  - Semantic domain : a complete lattice/cpo $\langle \mathcal{D}[\![P]\!], \sqsubseteq, \bot, \sqcup \rangle$
  - Compositional Fixpoint Semantics :

$$\mathcal{S}[\![P]\!] \stackrel{\text{def}}{=} \mathsf{lfp}_{\bot}^{\sqsubseteq}\, \mathcal{F}[\![P]\!] \left(\prod_{P' \prec P} \mathcal{S}[\![P']\!] \right)$$

$\mathsf{lfp}_{\bot}^{\sqsubseteq} f$ is the limit of $X^0 = \bot$, $X^{\delta+1} = f(X^\delta)$, $X^\lambda = \sqcup_{\beta<\lambda} X^\lambda$, λ limit ordinal, if any. Existence e.g. monotony (by Tarski constructive [PACJM '79]).

Example: Syntax of Programs

X	variables $X \in \mathbb{X}$
T	types $T \in \mathbb{T}$
E	arithmetic expressions $E \in \mathbb{E}$
B	boolean expressions $B \in \mathbb{B}$
$D ::= T\ X\,;$	declarations $D \in \mathbb{D}$, $\mathtt{vars}(D) = \{X\}$
$\mid\ T\ X\ ;\ D'$	$X \notin \mathtt{vars}(D')$, $\mathtt{vars}(D) = \{X\} \cup \mathtt{vars}(D')$
$C ::= X\ \mathtt{=}\ E;$	commands $C \in \mathbb{C}$ $(E \prec C)$
$\mid\ \mathtt{while}\ B\ C'$	$(B \prec C,\ C' \prec C)$
$\mid\ \mathtt{if}\ B\ C'$	$(B \prec C,\ C' \prec C)$
$\mid\ \mathtt{if}\ B\ C'\ \mathtt{else}\ C''$	$(B \prec C,\ C' \prec C,\ C'' \prec C)$
$\mid\ \{\ \mathrm{C}_1\ \ldots\ \mathrm{C}_n\ \}, (n \geq 0)$	$(C_1 \prec C, \ldots, C_n \prec C)$
$P ::= D\ C$	program $P \in \mathbb{P}$ $(C \prec P)$

Example: Concrete Semantic Domain of Programs

Reachability properties:

$$\Sigma[\![D\ C]\!] \stackrel{\text{def}}{=} \Sigma[\![D]\!]$$
$$\Sigma[\![T\ X\,;]\!] \stackrel{\text{def}}{=} \{X\} \mapsto T$$
$$\Sigma[\![T\ X\,;\ D]\!] \stackrel{\text{def}}{=} (\{X\} \mapsto T) \cup \Sigma[\![D]\!]$$

states ρ ($\rho(X)$ is the value of X)

$$\mathcal{D}[\![P]\!] \stackrel{\text{def}}{=} \wp(\Sigma[\![P]\!])$$ sets of states
$$\sqsubseteq \stackrel{\text{def}}{=} \subseteq$$ implication
$$\bot \stackrel{\text{def}}{=} \emptyset$$ false
$$\sqcup \stackrel{\text{def}}{=} \cup$$ disjunction

Example: Concrete Semantics of Programs (Reachability)

$$\mathcal{S}[\![X\ \mathtt{=}\ E;]\!]R \stackrel{\text{def}}{=} \{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in R \cap \mathrm{dom}(E)\}$$
$$\rho[X \leftarrow v](X) \stackrel{\text{def}}{=} v, \qquad \rho[X \leftarrow v](Y) \stackrel{\text{def}}{=} \rho(Y)$$
$$\mathcal{S}[\![\mathtt{if}\ B\ C']\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C']\!](\mathcal{B}[\![B]\!]R) \cup \mathcal{B}[\![\neg B]\!]R$$
$$\mathcal{B}[\![B]\!]R \stackrel{\text{def}}{=} \{\rho \in R \cap \mathrm{dom}(B) \mid B \text{ holds in } \rho\}$$
$$\mathcal{S}[\![\mathtt{if}\ B\ C'\ \mathtt{else}\ C'']\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C']\!](\mathcal{B}[\![B]\!]R) \cup \mathcal{S}[\![C'']\!](\mathcal{B}[\![\neg B]\!]R)$$
$$\mathcal{S}[\![\mathtt{while}\ B\ C']\!]R \stackrel{\text{def}}{=} \text{let } \mathcal{W} = \mathsf{lfp}^{\subseteq}_{\emptyset}\ \lambda\mathcal{X}\cdot R \cup \mathcal{S}[\![C']\!](\mathcal{B}[\![B]\!]\mathcal{X}) \text{ in } (\mathcal{B}[\![\neg B]\!]\mathcal{W})$$
$$\mathcal{S}[\![\{\}]\!]R \stackrel{\text{def}}{=} R$$
$$\mathcal{S}[\![\{C_1 \ldots C_n\}]\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C_n]\!] \circ \ldots \circ \mathcal{S}[\![C_1]\!] \quad n > 0$$
$$\mathcal{S}[\![D\ C]\!]R \stackrel{\text{def}}{=} \mathcal{S}[\![C]\!](\Sigma[\![D]\!]) \quad \text{(uninitialized variables)}$$

Not computable (undecidability).

Abstraction

A reasoning/computation which is restricted in 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;

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).

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).

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 $\cap$:

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

(Otherwise see [JLC '92].)

Reference

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

Moore Family

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

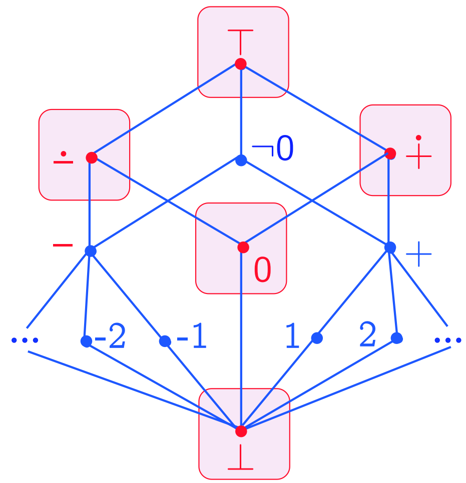
  $\overline{\mathcal{A}}$ is a Moore family

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

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

- In particular $\bigcap \emptyset = \Sigma \in \overline{\mathcal{A}}$ is "I don't know".

Example of Moore Family-Based Abstraction



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 $\overline{\mathcal{A}}$:

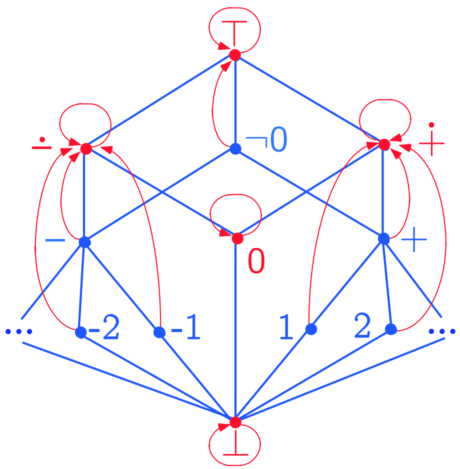
$$\rho_{\overline{\mathcal{A}}}(P) = \bigcap\{\overline{P} \in \overline{\mathcal{A}} \mid P \subseteq \overline{P}\}$$

is a closure operator:

- extensive,
- idempotent,
- isotone/monotonic;

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

Example of Closure Operator-Based Abstraction



The Lattice of Abstract Interpretations

- The set of all possible abstractions that is of all upper closure operators on the complete lattice

  $$\langle \mathcal{D}[\![P]\!],\ \sqsubseteq,\ \bot,\ \top,\ \sqcup,\ \sqcap\rangle$$

  is a complete lattice

  $$\langle \text{uco}(\mathcal{D}[\![P]\!] \mapsto \mathcal{D}[\![P]\!]),\ \dot{\sqsubseteq},\ \lambda x \cdot x,\ \lambda x \cdot \top,\ \lambda R \cdot \text{uco}(\dot{\sqcup} R),\ \dot{\sqcap}\rangle$$

- The meet of abstractions called the reduced product ($\dot{\sqcap}_{i \in \Delta} \rho_i$ is that most abstract abstraction more precise than all ρ_i, $i \in \Delta$)

Galois Connection Between Concrete and Abstract Properties

- For closure operators ρ, we have:

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

  written:

  $$\langle \wp(\Sigma),\ \subseteq\rangle \underset{\rho}{\overset{1}{\leftrightarrows}} \langle \rho(\wp(\Sigma)),\ \subseteq\rangle$$

  where 1 is the identity and:

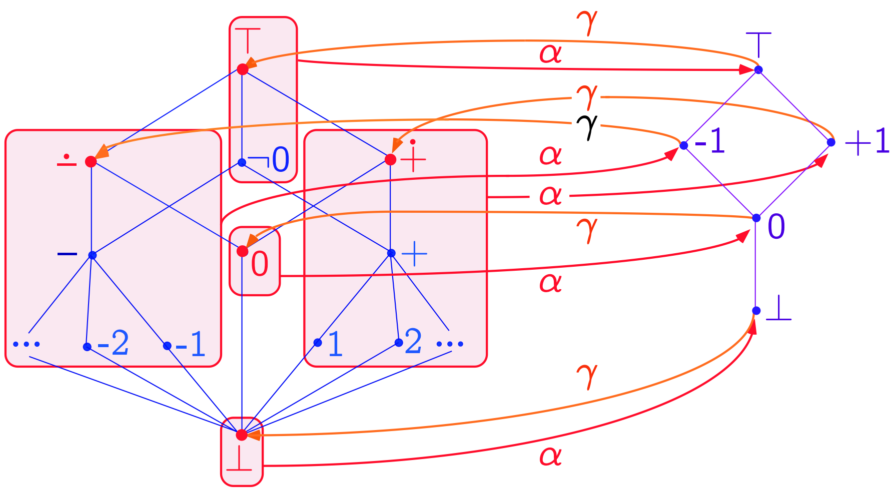
  $$\langle \wp(\Sigma),\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{\mathcal{D}},\ \sqsubseteq\rangle$$

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

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

- A Galois connection defines a closure operator $\rho = \alpha \circ \gamma$, hence a best abstraction.

Example of Galois Connection-Based Abstraction



Example: abstract semantic domain of programs

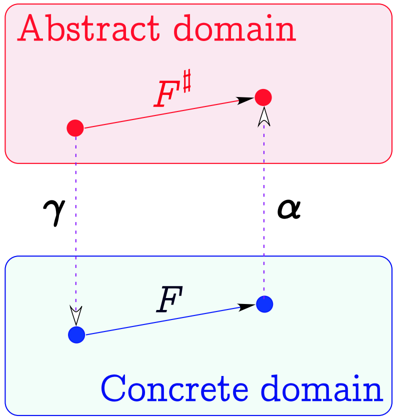
$$\langle \mathcal{D}^\sharp[\![P]\!],\ \sqsubseteq,\ \bot,\ \sqcup\rangle$$

such that:

$$\langle \mathcal{D},\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp[\![P]\!],\ \sqsubseteq\rangle$$

hence $\langle \mathcal{D}^\sharp[\![P]\!],\ \sqsubseteq,\ \bot,\ \sqcup\rangle$ is a complete lattice such that $\bot = \alpha(\emptyset)$ and $\sqcup X = \alpha(\cup\, \gamma(X))$

Function Abstraction



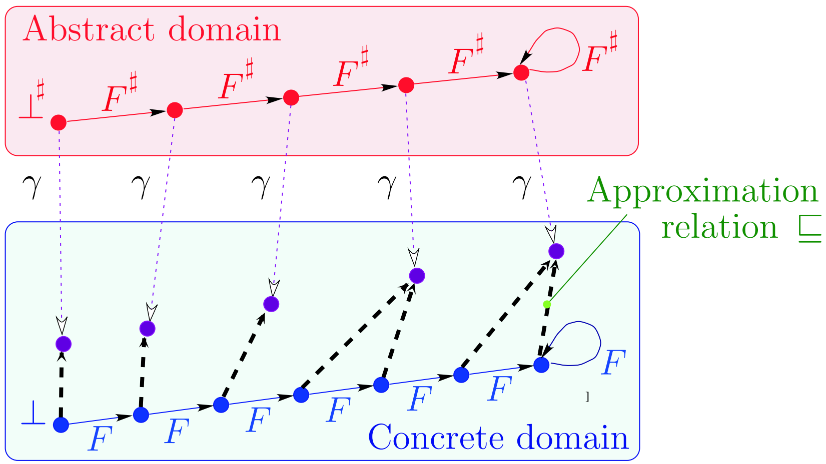
$$F^\sharp = \alpha \circ F \circ \gamma$$

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

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

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

Approximate Fixpoint Abstraction



$$F \circ \gamma \sqsubseteq \gamma \circ F^\sharp \Rightarrow \mathsf{lfp}\ F \sqsubseteq \gamma(\mathsf{lfp}\ F^\sharp)$$

Example: abstract semantics of programs (reachability)

$$\mathcal{S}^\sharp[\![X = E;]\!]R \stackrel{\text{def}}{=} \alpha(\{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in \gamma(R) \cap \text{dom}(E)\})$$

$$\mathcal{S}^\sharp[\![\texttt{if } B\ C']\!]R \stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{B}^\sharp[\![\neg B]\!]R$$

$$\mathcal{B}^\sharp[\![B]\!]R \stackrel{\text{def}}{=} \alpha(\{\rho \in \gamma(R) \cap \text{dom}(B) \mid B \text{ holds in } \rho\})$$

$$\mathcal{S}^\sharp[\![\texttt{if } B\ C' \texttt{ else } C'']\!]R \stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{S}^\sharp[\![C'']\!](\mathcal{B}^\sharp[\![\neg B]\!]R)$$

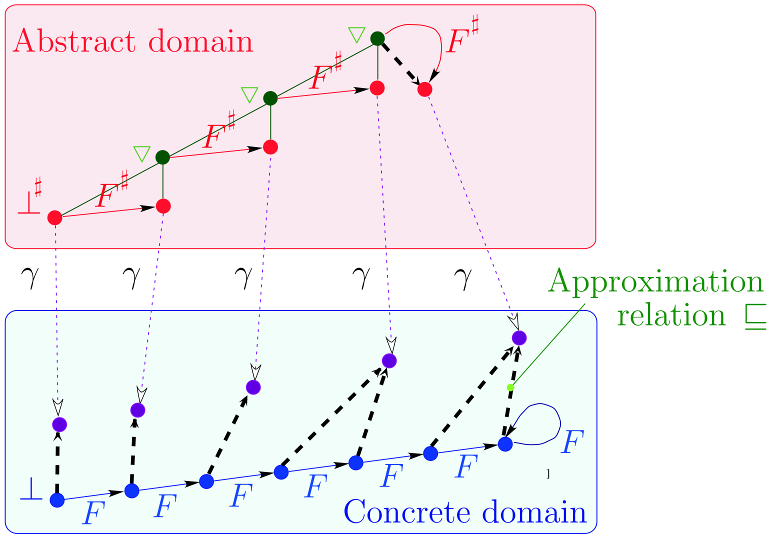
$$\mathcal{S}^\sharp[\![\texttt{while } B\ C']\!]R \stackrel{\text{def}}{=} \text{let } \mathcal{W} = \mathsf{lfp}_{\perp}^{\sqsubseteq}\, \lambda\mathcal{X} \cdot R \sqcup \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]\mathcal{X}) \text{ in } (\mathcal{B}^\sharp[\![\neg B]\!]\mathcal{W})$$

$$\mathcal{S}^\sharp[\![\{\}]\!]R \stackrel{\text{def}}{=} R$$

$$\mathcal{S}^\sharp[\![\{C_1 \ldots C_n\}]\!]R \stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C_n]\!] \circ \ldots \circ \mathcal{S}^\sharp[\![C_1]\!] \quad n > 0$$

$$\mathcal{S}^\sharp[\![D\ C]\!]R \stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C]\!](\top) \quad \text{(uninitialized variables)}$$

Convergence Acceleration with Widening



Widening Operator

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

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

Fixpoint Approximation with Widening

Concergence Theorem:

The upward iteration sequence with widening:

- $X^0 = \perp$ (infimum)
- $X^{i+1} = X^i$ if $F^\sharp(X^i) \sqsubseteq X^i$
  $= X^i \mathbin{\nabla} F^\sharp(X^i)$ otherwise

is ultimately stationary and its limit A is a sound upper approximation of $\mathsf{lfp}_{\perp}^{\sqsubseteq} F^\sharp$:

$$\mathsf{lfp}_{\perp}^{\sqsubseteq} F^\sharp \sqsubseteq A$$

Example: Abstract Semantics with Convergence Acceleration [1]

$$
\begin{aligned}
\mathcal{S}^\sharp[\![X = E;]\!]R &\stackrel{\text{def}}{=} \alpha(\{\rho[X \leftarrow \mathcal{E}[\![E]\!]\rho] \mid \rho \in \gamma(R) \cap \mathrm{dom}(E)\})\\
\mathcal{S}^\sharp[\![\mathtt{if}\ B\ C']\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{B}^\sharp[\![\neg B]\!]R\\
\mathcal{B}^\sharp[\![B]\!]R &\stackrel{\text{def}}{=} \alpha(\{\rho \in \gamma(R) \cap \mathrm{dom}(B) \mid B \text{ holds in } \rho\})\\
\mathcal{S}^\sharp[\![\mathtt{if}\ B\ C'\ \mathtt{else}\ C'']\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]R) \sqcup \mathcal{S}^\sharp[\![C'']\!](\mathcal{B}^\sharp[\![\neg B]\!]R)\\
\mathcal{S}^\sharp[\![\mathtt{while}\ B\ C']\!]R &\stackrel{\text{def}}{=} \text{let } \mathcal{F}^\sharp = \lambda \mathcal{X} \cdot \text{let } \mathcal{Y} = R \sqcup \mathcal{S}^\sharp[\![C']\!](\mathcal{B}^\sharp[\![B]\!]\mathcal{X})\\
&\qquad\qquad \text{in if } \mathcal{Y} \sqsubseteq \mathcal{X} \text{ then } \mathcal{X} \text{ else } \mathcal{X} \nabla \mathcal{Y}\\
&\qquad \text{and } \mathcal{W} = \mathsf{lfp}^{\sqsubseteq}_{\perp} \mathcal{F}^\sharp \text{ in } (\mathcal{B}^\sharp[\![\neg B]\!]\mathcal{W})\\
\mathcal{S}^\sharp[\![\{\}]\!]R &\stackrel{\text{def}}{=} R\\
\mathcal{S}^\sharp[\![\{C_1 \ldots C_n\}]\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C_n]\!] \circ \ldots \circ \mathcal{S}^\sharp[\![C_1]\!] \quad n > 0\\
\mathcal{S}^\sharp[\![D\ C]\!]R &\stackrel{\text{def}}{=} \mathcal{S}^\sharp[\![C]\!](\top) \quad \text{(uninitialized variables)}
\end{aligned}
$$

Extrapolation by Widening is Essentially Not Monotone

Proof by contradiction:

- Let ∇ be a widening operator
- Define $x \nabla' y = \text{if } y \sqsubseteq x \text{ then } x \text{ else } x \nabla y$
- Assume $x \sqsubset y = F(x)$ (during iteration)

then:
$$
\begin{array}{ccccccl}
x & \nabla' & y & = x \nabla y & \sqsupseteq & y & \textit{(soundness)}\\
\sqsubseteq & & \sqsubseteq & & & \sqsubseteq & \textit{(monotony hypothesis)}\\
y & \nabla' & y & = & & y & \textit{(termination)}
\end{array}
$$

$\Rightarrow$ $x \nabla y = y$, by antisymmetry!

$\Rightarrow$ $x \nabla F(x) = F(x)$ during iteration $\Rightarrow$ convergence cannot be enforced with monotone widening (so widening by finite abstraction is less powerful!)

[1] Note: $\mathcal{F}^\sharp$ not monotonic!

Soundness Theorem

- Convergence by extensivity (no longer monotone)
- Improvement by narrowing [POPL '77]
- *Soundness Corollary:* any abstract safety proof is valid in the concrete in that:

$$\mathcal{S}^\sharp[\![P]\!] \sqsubseteq Q \Longrightarrow \mathcal{S}[\![P]\!] \subseteq \gamma(Q)$$

- Example: $\gamma(Q)$ expresses the absence of run-time errors.

Reference

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

Applications of Abstract Interpretation

Applications of Abstract Interpretation

- **Static Program Analysis** [POPL '77], [POPL '78], [POPL '79] including **Dataflow Analysis** [POPL '79], [POPL '00], **Set-based Analysis** [FPCA '95], **Predicate Abstraction** [Manna's festschrift '03]
- **Syntax Analysis** [TCS 290(1) 2002]
- **Hierarchies of Semantics (including Proofs)** [POPL '92], [TCS 277(1–2) 2002]
- **Typing** [TCS 277(1–2) 2002]

Applications of Abstract Interpretation (Cont'd)

- **(Abstract) Model Checking** [POPL '00]
- **Program Transformation** [POPL '02]
- **Software Watermarking** [POPL '04]
- **Bisimulations** [RT-ESOP '04]

All these techniques involve sound approximations that can be formalized by abstract interpretation

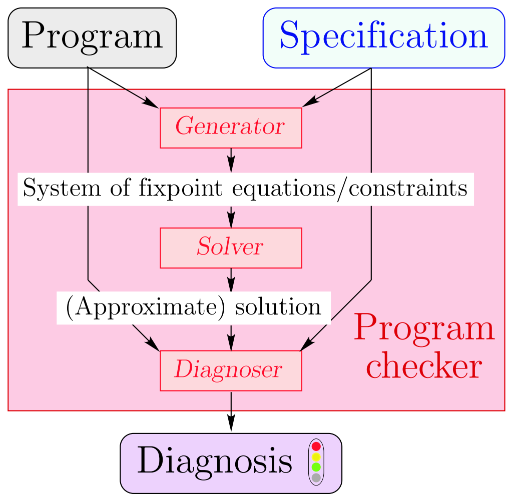
A Practical Application of Abstract Interpretation to the Verification of Safety Critical Embedded Control-Command Software

Reference

[1] 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.

[2] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. PLDI'03, San Diego, June 7–14, ACM Press, 2003.

Static Program Analysis



ASTRÉE: A Sound, Automatic, Specializable, Domain-Aware, Parametric, Modular, Efficient and Precise Static Program Analyzer

www.astree.ens.fr

- C programs:
  - structured C programs;
  - no dynamic memory allocation;
  - no recursion.
- Application Domain: safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems.

Concrete Operational Semantics

- International norm of C (ISO/IEC 9899:1999)
- *restricted by* implementation-specific behaviors depending upon the machine and compiler (e.g. representation and size of integers, IEEE 754-1985 norm for floats and doubles)
- *restricted by* user-defined programming guidelines (such as no modular arithmetic for signed integers, even though this might be the hardware choice)
- *restricted by* program specific user requirements (e.g. assert)

Abstract Semantics

- Reachable states for the concrete operational semantics
- Volatile environment is specified by a *trusted* configuration file.

Implicit Specification: Absence of Runtime Errors

- No violation of the norm of C (e.g. array index out of bounds)
- No implementation-specific undefined behaviors (e.g. maximum short integer is 32767)
- No violation of the programming guidelines (e.g. static variables cannot be assumed to be initialized to 0)
- No violation of the programmer assertions (must all be statically verified).

Example application

- Primary flight control software of the Airbus A340/A380 fly-by-wire system

- C program, automatically generated from a proprietary high-level specification
- A340: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays.

The Class of Considered Periodic Synchronous Programs

declare `volatile input, state and output variables;`
`initialize state and output variables;`
loop forever
- `read volatile input variables,`
- `compute output and state variables,`
- `write to volatile output variables;`

wait_for_clock ();
end loop

- Requirements: the only interrupts are clock ticks;
- Execution time of loop body less than a clock tick [3].

Reference

[3] 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.

Characteristics of the ASTRÉE Analyzer

Static: compile time analysis (≠ run time analysis Rational Purify, Parasoft Insure++)

Program Analyzer: analyzes programs not micromodels of programs (≠ PROMELA in SPIN or Alloy in the Alloy Analyzer)

Automatic: no end-user intervention needed (≠ ESC Java, ESC Java 2)

Sound: covers the whole state space (≠ MAGIC, CBMC) so never omit potential errors (≠ UNO, CMC from coverity.com) or sort most probable ones (≠ Splint)

Characteristics of the ASTRÉE Analyzer (Cont'd)

Multiabstraction: uses many numerical/symbolic abstract domains (≠ symbolic constraints in Bane)

Infinitary: all abstractions use infinite abstract domains with widening/narrowing (≠ model checking based analyzers such as VeriSoft, Bandera, Java PathFinder)

Efficient: always terminate (≠ counterexample-driven automatic abstraction refinement BLAST, SLAM)

Specializable: can easily incorporate new abstractions (and reduction with already existing abstract domains) (≠ general-purpose analyzers PolySpace Verifier)

Characteristics of the ASTRÉE Analyzer (Cont'd)

Domain-Aware: knows about control/command (e.g. digital filters) (as opposed to specialization to a mere programming style in C Global Surveyor)

Parametric: the precision/cost can be tailored to user needs by options and directives in the code

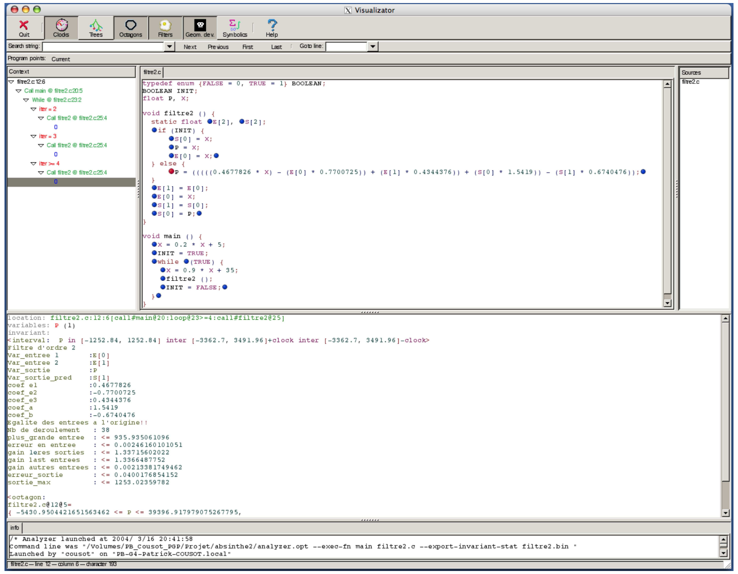
Automatic Parametrization: the generation of parametric directives in the code can be programmed (to be specialized for a specific application domain)

Characteristics of the ASTRÉE Analyzer (Cont'd)

Modular: an analyzer instance is built by selection of O-CAML modules from a collection each implementing an abstract domain

Precise: few or no false alarm when adapted to an application domain ⟶ VERIFIER!

Example of Analysis Session



Benchmarks for the Primary Flight Control Software of the Airbus A340

- Comparative results (commercial software):

  4,200 (false?) alarms,

  3.5 days;

- Our results:

  0 alarm,

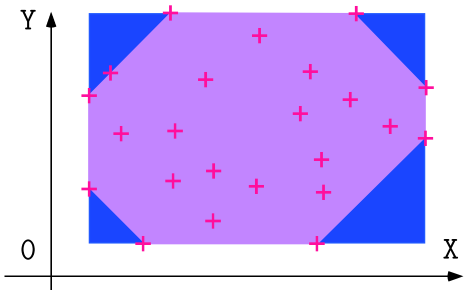
  1h20 on 2.8 GHz PC,

  300 Megabytes

  ⟶ A world première!

Examples of Abstractions

General-Purpose Abstract Domains: Intervals and Octagons



Intervals:

$$\begin{cases} 1 \leq x \leq 9 \\ 1 \leq y \leq 20 \end{cases}$$

Octagons [4]:

$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 20 \\ x - y \leq 04 \end{cases}$$

Difficulties: many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [5]

Reference

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

[5] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In *ESOP'04*, Barcelona, LNCS 2986, pp. 1—17, Springer, 2004.

Floating-Point Computations

- Code Sample:

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
} % gcc float-error.c
% ./a.out
0.000000
```

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951488.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
134217728.000000
```

$$(x + a) - (x - a) \neq 2a$$

Symbolic abstract domain

- Interval analysis: if $x \in [a, b]$ and $y \in [c, d]$ then $x - y \in [a - c, b - d]$ so if $x \in [0, 100]$ then $x - x \in [-100, 100]$!!!
- The symbolic abstract domain propagates the symbolic values of variables and performs simplifications;
- Must maintain the maximal possible rounding error for float computations (overestimated with intervals);

```
% cat -n x-x.c
     1  void main () { int X, Y;
     2          __ASTREE_known_fact(((0 <= X) && (X <= 100)));
     3          Y = (X - X);
     4          __ASTREE_log_vars((Y));
     5  }
```

```
astree -exec-fn main -no-relational x-x.c
Call main@x-x.c:1:5-x-x.c:1:9:
<interval: Y in [-100, 100]>
```

```
astree -exec-fn main x-x.c
Call main@x-x.c:1:5-x-x.c:1:9:
<interval: Y in {0}> <symbolic: Y = (X -i X)>
```

Clock Abstract Domain for Counters

- Code Sample:

```
R = 0;
while (1) {
  if (I)
    { R = R+1; }
  else
    { R = 0; }
  T = (R>=n);
  wait_for_clock ();
}
```

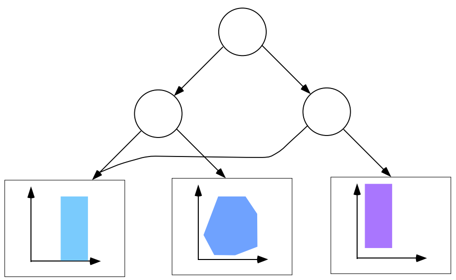
- Output T is true iff the volatile input I has been true for the last n clock ticks.
- The clock ticks every s seconds for at most h hours, thus R is bounded.
- To prove that R cannot overflow, we must prove that R cannot exceed the elapsed clock ticks *(impossible using only intervals)*.

- Solution:
  - We add a phantom variable `clock` in the concrete user semantics to track elapsed clock ticks.
  - For each variable `X`, we abstract three intervals: `X`, `X+clock`, and `X-clock`.
  - If `X+clock` or `X-clock` is bounded, so is `X`.

Boolean Relations for Boolean Control

- Code Sample:

```
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ...
    B = (X == 0);
    ...
    if (!B) {
      Y = 1 / X;
    }
    ...
  }
}
```



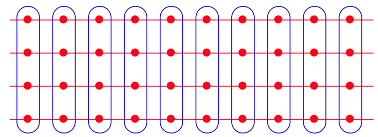
The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leafs

Control Partitionning for Case Analysis

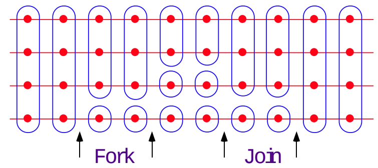
- Code Sample:

```
/* trace_partitionning.c */
void main() {
  float t[5] = {-10.0, -10.0, 0.0, 10.0, 10.0};
  float c[4] = {0.0, 2.0, 2.0, 0.0};
  float d[4] = {-20.0, -20.0, 0.0, 20.0};
  float x, r;
  int i = 0;

  ... found invariant -100 ≤ x ≤ 100 ...

  while ((i < 3) && (x >= t[i+1])) {
    i = i + 1;
  }
  r = (x - t[i]) * c[i] + d[i];
}
```

Control point partitionning:

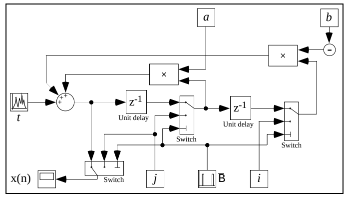


Trace partitionning:



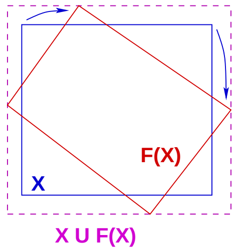
Delaying abstract unions in tests and loops is more precise for non-distributive abstract domains (and much less expensive than disjunctive completion).

2^d Order Digital Filter:

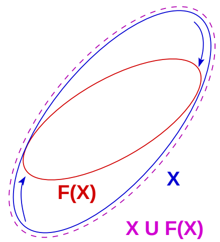


Ellipsoid Abstract Domain for Filters

- Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is bounded, which must be proved in the abstract.
- There is no stable interval or octagon.
- The simplest stable surface is an ellipsoid.



unstable interval



stable ellipsoid

Reference

[6] J. Feret. Static analysis of digital filters. In *ESOP'04*, Barcelona, LNCS 2986, pp. 33—-48, Springer, 2004.

(Automatic) Parameterization

- All abstract domains of ASTRÉE are parameterized, e.g.
  - variable packing for octagones and decision trees,
  - partition/merge program points,
  - loop unrollings,
  - thresholds in widenings, . . . ;
- End-users can either parameterize by hand (analyzer options, directives in the code), or
- choose the automatic parameterization (default options, directives for pattern-matched predefined program schemata).

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
- 60 ellipse invariants, etc . . .

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

Why finite abstractions will not do?

Theoretical reasons on finite abstraction:

- If an abstraction works, then the abstact domain must contain an inductive invariant, so [7]:
  - No finite domain can represent all such necessary inductive invariants for a programming language
  - Finite abstractions will fail on infinitely many programs (undecidability)
  - Whereas well-chosen widenings will always do better or at least as well as any given finite domain

Reference

[7] P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In M. Bruynooghe and M. Wirsing, (Eds), *Proc. 4^{th} Int. Symp. PLILP '92*, Louvain, BE, 26–28 august 1992, LNCS 631, pp. 269–295. Springer, 1992.

Why finite abstractions will not do? (Cont'd)

Theoretical reasons on abstraction refinement:

- Refinement (e.g. counter-example driven) aims at [8]:
  - Computing the most abstract inductive invariant
  - By an iterative fixpoint computation
  - In the concrete
  - Which does not converge/terminate in general (by undecidability)

Reference

[8] P. Cousot. Partial Completeness of Abstract Fixpoint Checking. In B.Y. Choueiry and T. Walsh (Eds), *Proc. 4^{th} Int. Symp. SARA '2000*, Horseshoe Bay, TX, USA, LNAI 1864, pp. 1–25. Springer, 26–29 jul. 2000.

Why finite abstractions will not do? (Cont'd)

Practical reasons on abstraction:

- The adequate abstract domain must be guessed from the program before starting the analysis [9]:
  - E.g. in the form of a finite model
  - Impossible since most abtract predicates do not appear at all in the program text
  - E.g. polyhedral analysis, filter analysis, congruence analysis, etc.

Reference

[9] P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In M. Bruynooghe and M. Wirsing, (Eds), *Proc. 4th Int. Symp. PLILP '92*, Louvain, BE, 26–28 august 1992, LNCS 631, pp. 269–295. Springer, 1992.

Why finite abstractions will not do? (Cont'd)

Practical reasons on refinement:

- Since abstraction by refinement is done using concrete computations, it is unable to synthesize abstract invariants
- e.g. in polyhedral analysis, congruence analysis, filter analysis, etc, the invariant will come out in the form of (infinitely) many points:
  - one by one (counter-example based)
  - simultaneously (abstraction completion [10])

Reference

[10] R. Giacobazzi and E. Quintarelli, Incompleteness, Counterexamples and Refinements in Abstract Model-Checking. In *Proc. Eight International Symposium on Static Analysis, SAS '01*, P. Cousot (Ed), Paris, France, 16–18 July 2001. Lecture Notes in Computer Science 2126, Springer, pp. 356–373.

Example [11]

```
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;
void filter () {
  static float E[2], S[2];
  if (INIT) { S[0] = X; P = X; E[0] = X; }
  else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
             + (S[0] * 1.5)) - (S[1] * 0.7)); }
  E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}
void main () { X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}
```

Reference

[11] J. Feret. Static analysis of digital filters. In *ESOP'04*, Barcelona, LNCS 2986, pp. 33—-48, Springer, 2004.

Possible origins of imprecision and how to fix it

In case of false alarm, the imprecision can come from:

- Abstract transformers (not best possible) ⟶ improve algorithm;
- Automatized parametrization (e.g. variable packing) ⟶ improve pattern-matched program schemata;
- Iteration strategy for fixpoints ⟶ fix widening [2];
- Inexpressivity i.e. indispensable local inductive invariant are inexpressible in the abstract ⟶ add a new abstract domain to the reduced product (e.g. filters).

2 This can be very hard since at the limit only a precise infinite iteration might be able to compute the proper abstract invariant. In that case, it might be better to design a more refined abstract domain.

Conclusion

Conclusion

- Most applications of abstract interpretation tolerate a small rate (typically 5 to 15%) of false alarms:
  - Program transformation → do not optimize,
  - Typing → reject some correct programs, etc,
  - WCET analysis → overestimate;
- Some applications require no false alarm at all:
  - Program verification.
- Theoretically possible [SARA '00], practically feasible [PLDI '03]

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.

[PLDI '03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. PLDI'03, San Diego, June 7–14, ACM Press, 2003.

The Future

- Short term (1 year):
  - Backward analysis (help in locating the origin of alarms)
  - Verification of compiled code (for a given compiler/machine)
  - ADA interface

The Future (Cont'nd)

- Longer term:
  - Asynchronous concurrency (for less critical software)
  - Functional properties (reactivity)
  - Verification of specifications (verification from specifications to machine code)

THE END, THANK YOU

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

References

[POPL '77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY, USA.

[PACJM '79] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. Pacific Journal of Mathematics 82(1):43–57 (1979).

[POPL '78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY, U.S.A.

[POPL '79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY, U.S.A.

[POPL '92] P. Cousot and R. Cousot. Inductive Definitions, Semantics and Abstract Interpretation. In Conference Record of the 19^{th} ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages, pages 83–94, Albuquerque, New Mexico, 1992. ACM Press, New York, U.S.A.

[FPCA '95] P. Cousot and R. Cousot. Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation. In *SIGPLAN/SIGARCH/WG2.8 7^{th} Conference on Functional Programming and Computer Architecture, FPCA'95*. La Jolla, California, U.S.A., pages 170–181. ACM Press, New York, U.S.A., 25-28 June 1995.

[POPL '00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Mass., January 2000. ACM Press, New York, NY.

[POPL '02] P. Cousot and R. Cousot. Systematic Design of Program Transformation Frameworks by Abstract Interpretation. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 178–190, Portland, Oregon, January 2002. ACM Press, New York, NY.

[TCS 277(1–2) 2002] P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science* 277(1–2):47–103, 2002.

[TCS 290(1) 2002] P. Cousot and R. Cousot. Parsing as abstract interpretation of grammar semantics. *Theoret. Comput. Sci.*, 290:531–544, 2003.

[Manna's festschrift '03] P. Cousot. Verification by Abstract Interpretation. *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, N. Dershowitz (Ed.), Taormina, Italy, June 29 – July 4, 2003. Lecture Notes in Computer Science, vol. 2772, pp. 243–268. © Springer-Verlag, Berlin, Germany, 2003.

[POPL '04] P. Cousot and R. Cousot. An Abstract Interpretation-Based Framework for Software Watermarking. In *Conference Record of the Thirtyfirst Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 173–185, Venice, Italy, January 14-16, 2004. ACM Press, New York, NY.

[RT-ESOP '04] F. Ranzato and F. Tapparo. Strong Preservation as Completeness in Abstract Interpretation. Porc. Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, D.A. Schmidt (Ed), Lecture Notes in Computer Science 2986, Springer, 2004, pp. 18–32.