« The Rôle of Abstract Interpretation in Formal Methods »

Patrick Cousot
École normale supérieure
45 rue d'Ulm, 75230 Paris cedex 05, France

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

The 5th IEEE International Conference on Software Engineering and Formal Methods
IEEE SEFM 2007 — London, United Kingdom
September 10–14, 2007

Contents

1. Abstract Interpretation

Formal methods

- Objective: specification, development and verification of software and hardware systems
- Problem: establish the satisfaction of a specification by the semantics (set of behaviors) of a computer system.

Abstract Interpretation

– Abstract interpretation [1], [2]: a theory of sound approximation of mathematical structures, in particular those involved in the description of the specification and the semantics of computer systems.

⇒ Abstract interpretation is used (often *implicitly*) everywhere in formal methods

References

[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 1978.

[2] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th ACM POPL*, 269–282, 1979.

Applications of Abstract Interpretation

– Abstract interpretation is used for the systematic derivation of sound methods and algorithms for approximating undecidable or highly complex problems in various areas of computer science

– Applications:

- semantics,
- verification and proof,
- model-checking,
- static analysis,
- typing,
- program transformation and optimization,
- software steganography,
- software obfuscation,
- etc.

Main Current Application of Abstract Interpretation

Safety and security of complex hardware and software computer systems.

2. Verification

Semantics

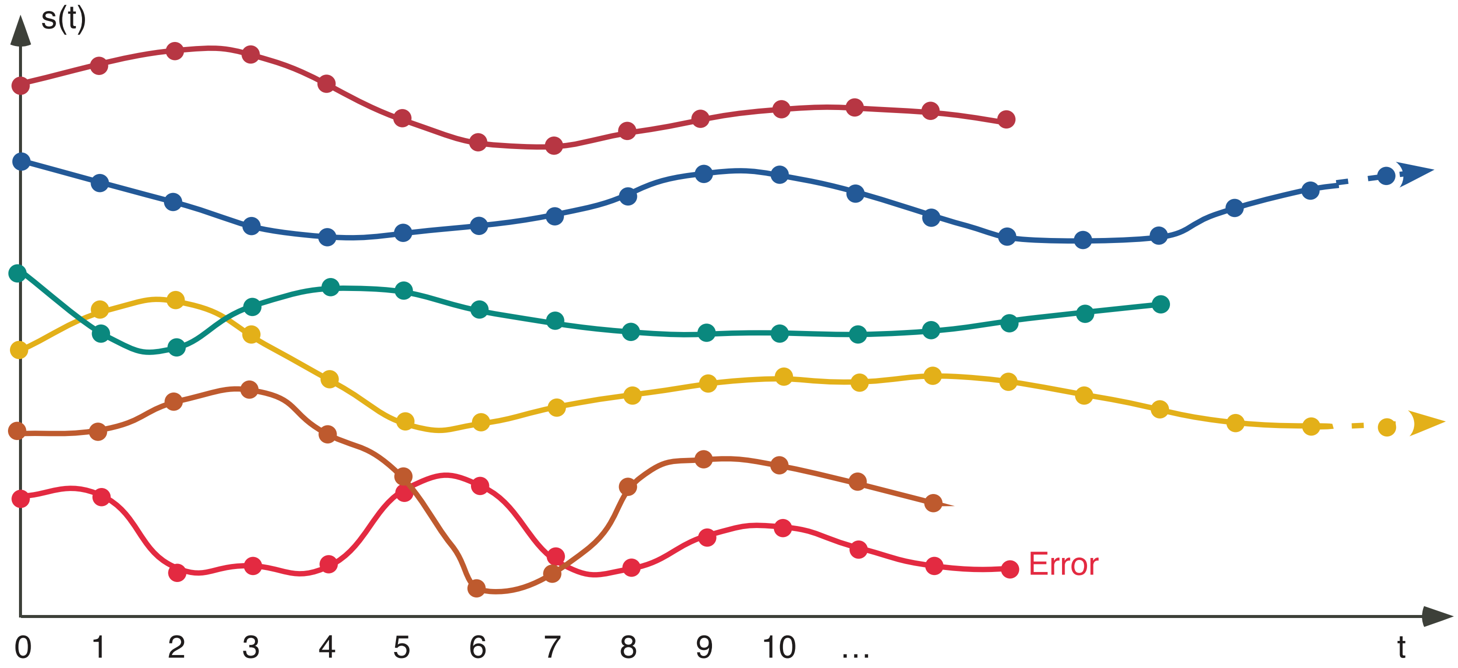
- The semantics $\mathcal{S}[\![\mathbb{p}]\!]$ of a software and hardware system $\mathbb{p} \in \mathbb{P}$ is a formal model of the execution of this system $\mathbb{p}$.
- A semantic domain $\mathcal{D}$ is a set of such formal models, so

$$\forall \mathbb{p} \in \mathbb{P} : \mathcal{S}[\![\mathbb{p}]\!] \in \mathcal{D}\ ^{1}$$

1 To be more precise one might consider $\mathcal{D}[\![\mathbb{p}]\!], \mathbb{p} \in \mathbb{P}$.

Example: Operational Semantics

– The operational semantics describes all possible program executions as a set of *maximal execution traces*



States and Traces

– States in Σ, describe an instantaneous snapshot of the execution

– Traces are finite or infinite sequences of states in Σ, two successive states corresponding to an elementary program step.

– In that case

- $\Sigma^n \triangleq [0, n[\mapsto \Sigma$ traces of length $n = 1, \ldots, +\infty$ [2].
- $\mathcal{T} \triangleq \bigcup_{n=1}^{+\infty} \Sigma^n$ all possible traces
- $\mathcal{D} \triangleq \wp(\mathcal{T})$ [3] semantic domain

---

[2] $[0, n[= \{0, 1, \ldots, n-1\}$ with $[0, 0[= \varnothing$.

[3] $\wp(S) \triangleq \{S' \mid S' \subseteq S\}$ is the powerset of S.

Properties and Specifications

- A specification is a required *property* of the semantics of the system.
- The interpretation of a property is therefore a set of semantic models that satisfy this property
- Formally, the set of properties is

$$\mathcal{P} \triangleq \wp(\mathcal{D}) \; .$$

Example: Properties of a Trace Semantics

- $\mathcal{T}$ — all possible traces
- $\mathcal{D} \triangleq \wp(\mathcal{T})$ — semantic domain (sets of traces)
- $\mathcal{P} \triangleq \wp(\mathcal{D}) \triangleq \wp(\wp(\mathcal{T}))$ — properties (sets of sets of traces)

Collecting Semantics

– The strongest property of a system $\mathbb{p} \in \mathbb{P}$ is its semantics $\{\mathcal{S}[\![\mathbb{p}]\!]\}$, called the collecting semantics

$$\mathcal{C}[\![\mathbb{p}]\!] \triangleq \{\mathcal{S}[\![\mathbb{p}]\!]\} \ .$$

Verification

– The satisfaction of a specification $P \in \mathcal{P}$ by a system $\mathbb{p}$ (more precisely by the system semantics $\mathcal{S}[\![\mathbb{p}]\!]$) is

$$\mathcal{S}[\![\mathbb{p}]\!] \in P$$

– Satisfaction can equivalently be defined as the proof that

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P$$

i.e. *the strongest program property implies its specification.*

Undecidability

– The proof that

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P$$

is not mechanizable (Gödel, Turing).

3. Abstract Verification

Abstraction

To prove

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P$$

one can use a sound over-approximation of the collecting semantics

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq \mathcal{C}^\sharp[\![\mathbb{p}]\!]$$

and a sound under-approximation of the property

$$P^\sharp \subseteq P$$

and make the correctness proof *in the abstract*

$$\mathcal{C}^\sharp[\![\mathbb{p}]\!] \subseteq P^\sharp$$

Abstract Domain

- For automated proofs, $C^\sharp[\![\mathbb{p}]\!]$ and $P^\sharp$ must be computer-representable
- Hence, they are not chosen in the mathematical concrete domain

$$\langle \mathcal{P},\ \subseteq \rangle$$

but in a computer-representable abstract domain

$$\langle \mathcal{P}^\sharp,\ \sqsubseteq \rangle$$

Concretization Function

- The abstract to concrete correspondence is given by a concretization function

$$\gamma \in \mathcal{P}^\sharp \mapsto \mathcal{P}$$

providing the meaning $\gamma(P^\sharp)$ of abstract properties $P^\sharp$

- For abstract reasonings to be valid in the concrete, γ should preserve the abstract implication

$$\forall Q_1, Q_2 \in \mathcal{P}^\sharp : (Q_1 \sqsubseteq Q_2) \Longrightarrow (\gamma(Q_1) \subseteq \gamma(Q_2))$$

Soundness of the Abstraction

– The soundness of the abstract over-approximation of the collecting semantics is now

$$\mathcal{C}[\![\mathrm{p}]\!] \subseteq \gamma(\mathcal{C}^{\sharp}[\![\mathrm{p}]\!])$$

– The soundness of the abstract under-approximation of the property is now

$$\gamma(P^{\sharp}) \subseteq P$$

Abstract Proofs

– Then, the abstract proof

$$\mathcal{C}^\sharp[\![\mathbb{p}]\!] \sqsubseteq P^\sharp$$

implies

$$\gamma(\mathcal{C}^\sharp[\![\mathbb{p}]\!]) \subseteq \gamma(P^\sharp)$$

and by soundness of the abstraction

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq \gamma(\mathcal{C}^\sharp[\![\mathbb{p}]\!]) \quad \text{and} \quad \gamma(P^\sharp) \subseteq P$$

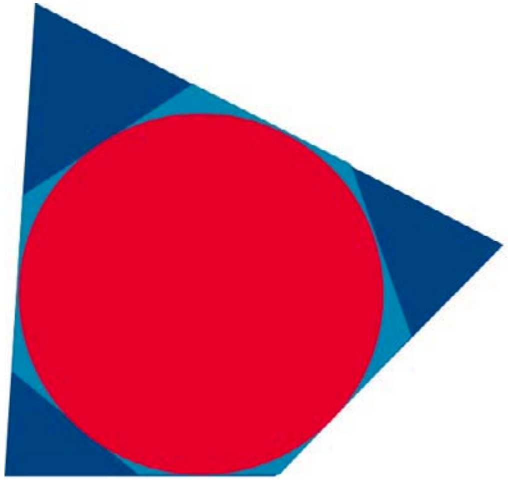
we have *proved correctness in the concrete*

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P \ .$$

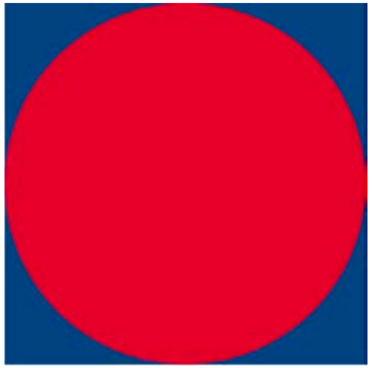
4. Best Abstraction

Best Abstraction

– If we want to over-approximate a disk in two dimensions by a polyhedron there is no best (smallest) one, as shown by Euclid.



– However if we want to over-approximate a disk by a rectangular parallelepiped which sides are parallel to the axes, then there is definitely a best (smallest) one.



Best Abstraction (Cont'd)

– In case of best over-approximation, there is an abstraction function

$$\alpha \in \mathcal{P} \mapsto \mathcal{P}^\sharp$$

such that

- for all $P \in \mathcal{P}$, $\alpha(P) \in \mathcal{P}^\sharp$ is an abstract over-approximation of P, so

$$P \subseteq \gamma(\alpha(P))$$

and,

- it is the most precise abstract over-approximation, so

$$\forall Q \in \mathcal{P}^\sharp : P \subseteq \gamma(Q) \Longrightarrow \alpha(P) \sqsubseteq Q$$

(whence $\gamma(\alpha(P)) \subseteq \gamma(Q)$ by monotony of γ).

Best Abstraction and Galois Connection

– It follows in that case of existence of a best abstraction, that the pair $\langle\alpha, \gamma\rangle$ is a Galois connection [3].

$$\forall P \in \mathcal{P} : \forall Q \in \mathcal{P}^\sharp : P \subseteq \gamma(Q) \iff \alpha(P) \sqsubseteq Q$$

Reference

[3] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th ACM POPL*, 269–282, 1979.

5. Examples of Traditional (Implicit) Abstractions

Abstraction is very often implicit, as shown by the following classical examples.

Looseness Abstraction

Traditional View of Program Properties

– In the operational trace semantics example $\mathcal{D} \triangleq \wp(\mathcal{T})$ so properties are

$$\mathcal{P} \triangleq \wp(\wp(\mathcal{T}))$$

where $\mathcal{T}$ is the set of traces.

– The traditional view of program properties as set of traces [4], [5] is an abstraction.

References

[4] B. Alpern and F. Schneider. Defining liveness. *Inf. Process. Lett.*, 21:181–185, 1985.

[5] A. Pnueli. The temporal logic of programs. *18th ACM FOCS*, 46–57, 1977.

Example of Program Properties

– An example of progam property is

$$P_{01} \triangleq \{\{\sigma 0 \mid \sigma \in \mathcal{T}\}, \{\sigma 1 \mid \sigma \in \mathcal{T}\}\} \in \mathcal{P}$$

specifying that executions of the system always terminate with 0 or always terminate with 1.

– This cannot be expressed in the traditional view of program properties as set of traces [4], [5].

Looseness Abstraction

– This traditional understanding of a program property is given by the looseness abstraction

$$\alpha_{\cup} \in \wp(\wp(\mathcal{T})) \mapsto \wp(\mathcal{T}),$$
$$\alpha_{\cup}(P) \triangleq \bigcup P$$

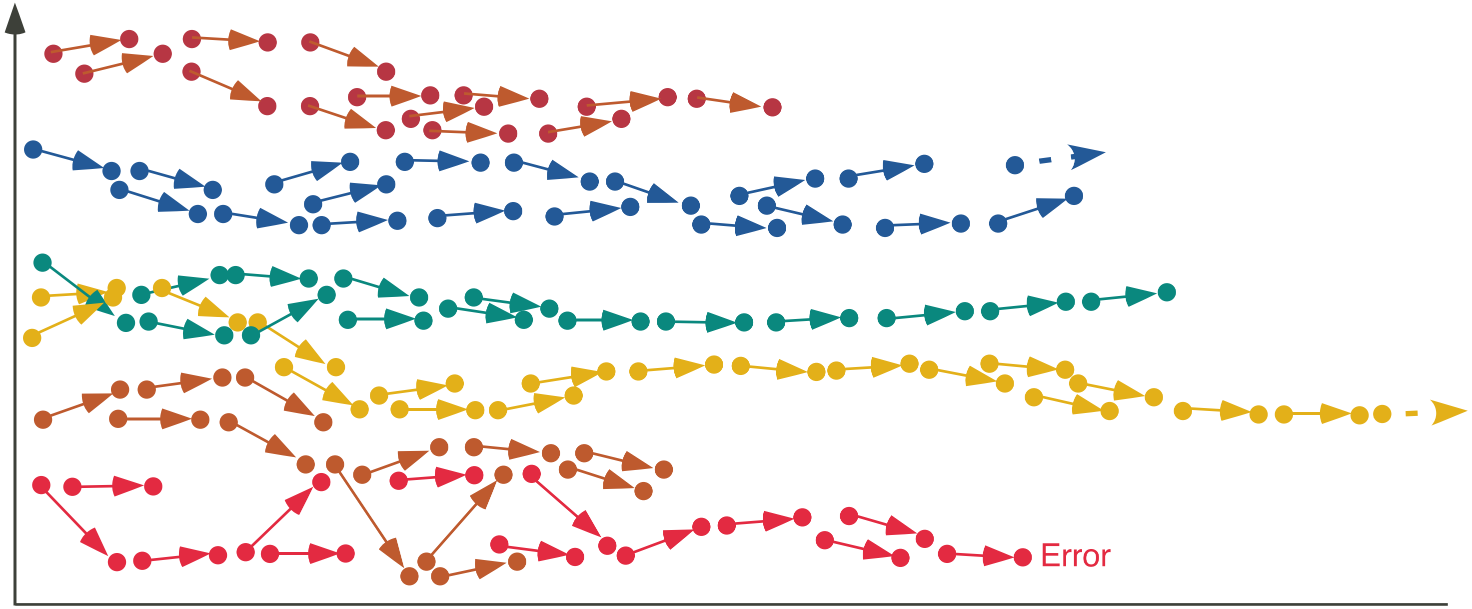
with concretization

$$\gamma_{\cup} \in \wp(\mathcal{T}) \mapsto \wp(\wp(\mathcal{T})),$$
$$\gamma_{\cup}(Q) \triangleq \wp(Q) .$$

– An example is $\alpha_{\cup}(P_{01}) = \{\sigma 0, \sigma 1 \mid \sigma \in \mathcal{T}\}$ specifying that execution always terminate, either with 0 or with 1.

Transition Abstraction

Transition Abstraction



Transition Abstraction

– The transition abstraction

$$\alpha_{\mathcal{T}} \in \wp(\mathcal{T}) \mapsto \wp(\Sigma \times \Sigma)$$

collects transitions along traces.

$$\alpha_{\mathcal{T}}(\sigma_0 \ldots \sigma_n) \triangleq \{\sigma_i \to \sigma_{i+1} \mid 0 \leqslant i < n\},$$

$$\alpha_{\mathcal{T}}(\sigma_0 \ldots \sigma_i \ldots) \triangleq \{\sigma_i \to \sigma_{i+1} \mid i \geqslant 0\}, \qquad \text{and}$$

$$\alpha_{\mathcal{T}}(T) \triangleq \bigcup \{\alpha(\sigma) \mid \sigma \in T\}\ .$$

– The concretization $\gamma_{\mathcal{T}} \in \wp(\Sigma \times \Sigma) \mapsto \wp(\mathcal{T})$ is

$$\gamma_{\mathcal{T}}(\tau) \triangleq \bigcup_{n=1}^{+\infty} \{\sigma \in [0, n[\mapsto \Sigma \mid \forall i < n : \langle \sigma_i,\ \sigma_{i+1} \rangle \in \tau\}\ .$$

Transition System Abstraction

– The abstraction may also collect initial states

$$\alpha_\iota(T) \triangleq \{\sigma_0 \mid \sigma \in T\}$$

$$\text{so} \quad \alpha_{\iota\tau}(T) \triangleq \langle \alpha_\iota(T),\ \alpha_\tau(T) \rangle \ .$$

– We let

$$\gamma_{\iota\tau} \triangleq \gamma_\iota(\iota) \cap \gamma_\tau(\tau)$$

$$\text{where} \quad \gamma_\iota(\iota) \triangleq \{\sigma \in \mathcal{T} \mid \sigma_0 \in \iota\}$$

– $\langle \alpha_{\iota\tau},\ \gamma_{\iota\tau} \rangle$ is a Galois connection.

Transition System Abstraction (Cont'd)

– The transition system abstraction [6] underlies small-step operational semantics.

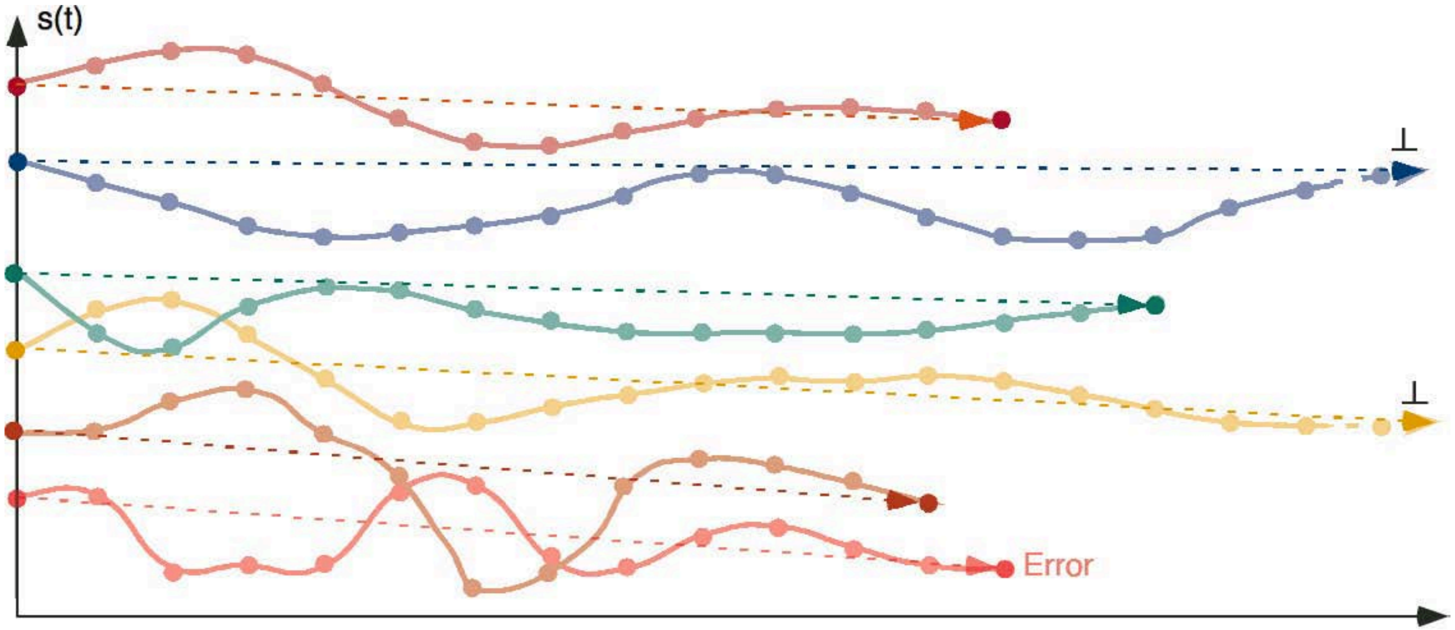
– This is an approximation since traces can express properties not expressible by a transition system (like fairness of parallel processes).

References

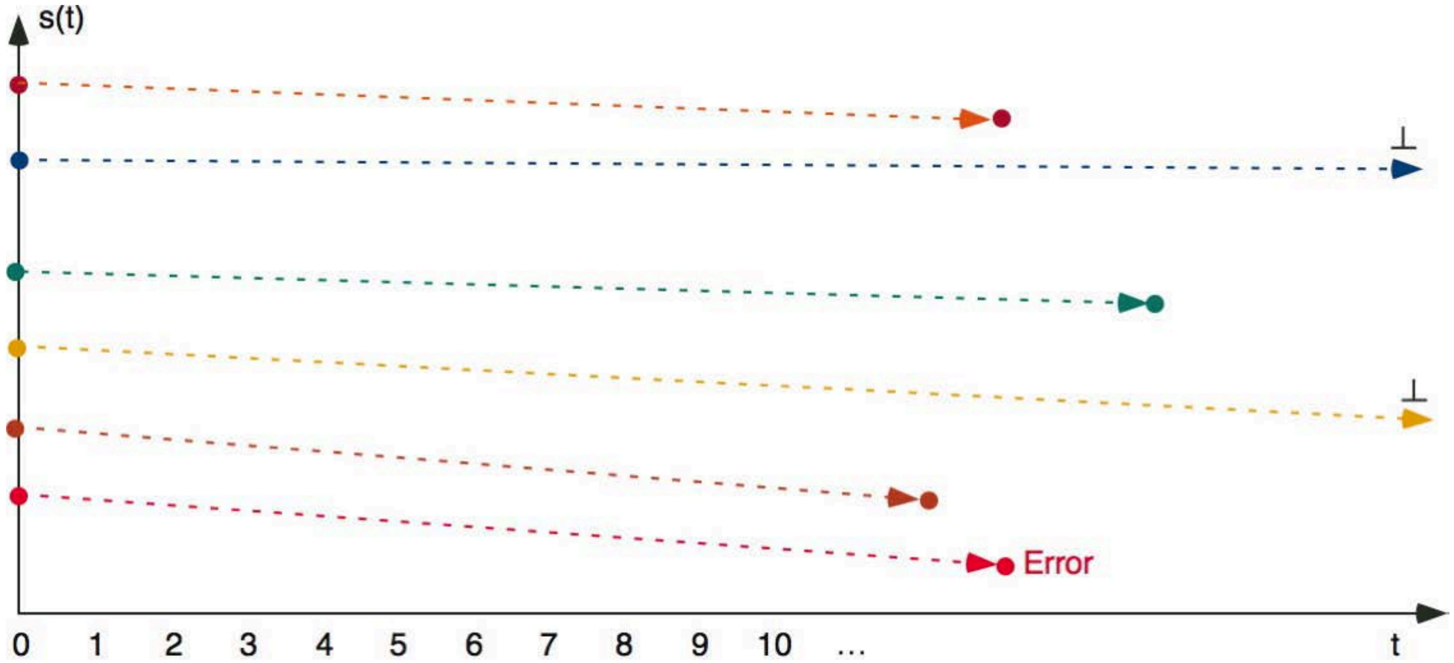
[6] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)*. Thèse d'État ès sci. math., Univ. sci. et médicale de Grenoble, 1978.

Input-Output Abstraction

Input-Output Abstraction



Input-Output Abstract Semantics



Input-Output Abstraction

– The input-output abstraction

$$\alpha_{io} \in \wp(\mathcal{T}) \mapsto \wp(\Sigma \times (\Sigma \cup \{\bot\}))$$

collects initial and final states of traces (and maybe $\bot$ for infinite traces to track nontermination).

$$\alpha_{io}(\sigma_0 \ldots \sigma_n) = \langle \sigma_0,\ \sigma_n \rangle,$$
$$\alpha_{io}(\sigma_0 \ldots \sigma_i \ldots) = \langle \sigma_0,\ \bot \rangle,$$

and
$$\alpha_{io}(T) = \{\alpha_{io}(\sigma) \mid \sigma \in T\}\ .$$

Input-Output Abstraction (Cont'd)

– The input-output abstraction α_{io} underlies
  - denotational semantics, as well as big-step operational, predicate transformer and axiomatic semantics extended to nontermination [10], and
  - interprocedural static analysis using relational procedure summaries [7], [8], [9].

References

[7] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. *IFIP Conf. on Formal Description of Programming Concepts*, 237–277, North-Holland, 1977.

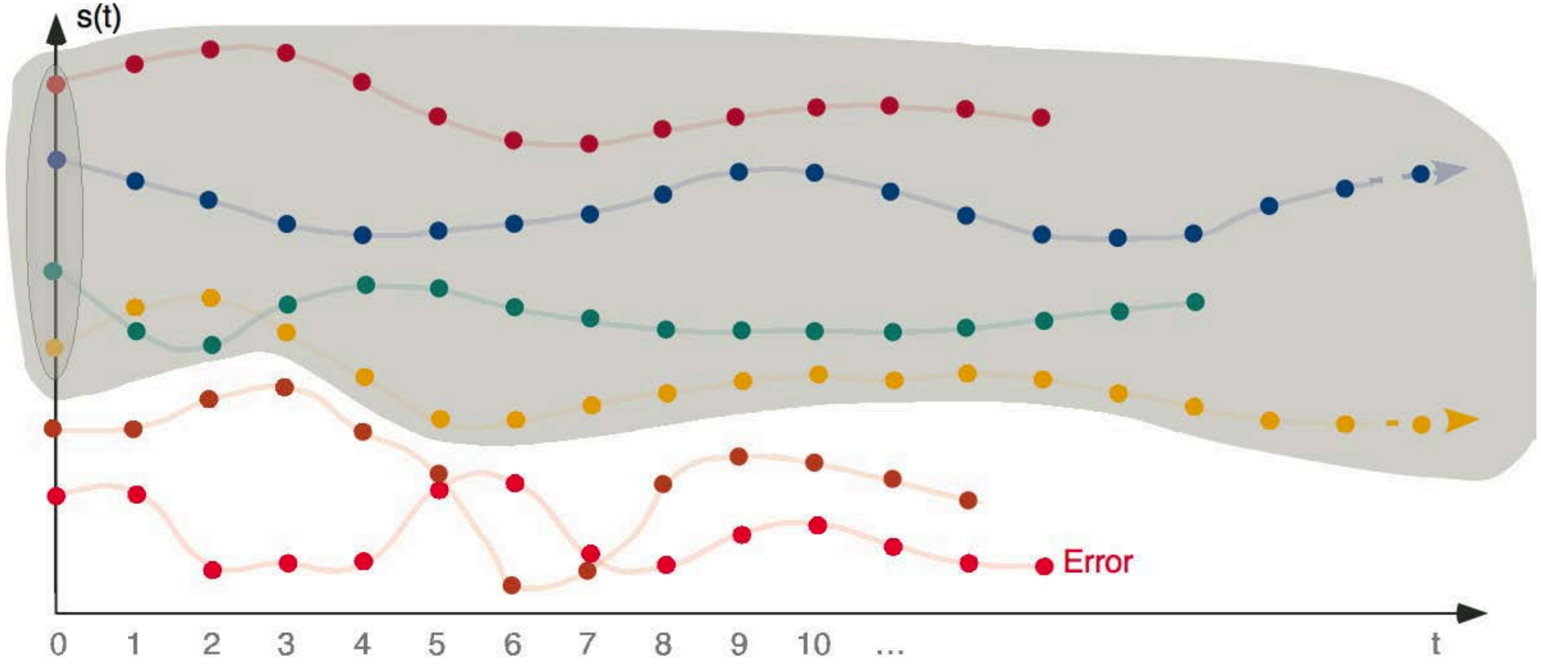
[8] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 1978.

[9] P. Cousot and R. Cousot. Modular static program analysis. 11^{th} CC, LNCS 2304, 159–178, Springer, 2002.

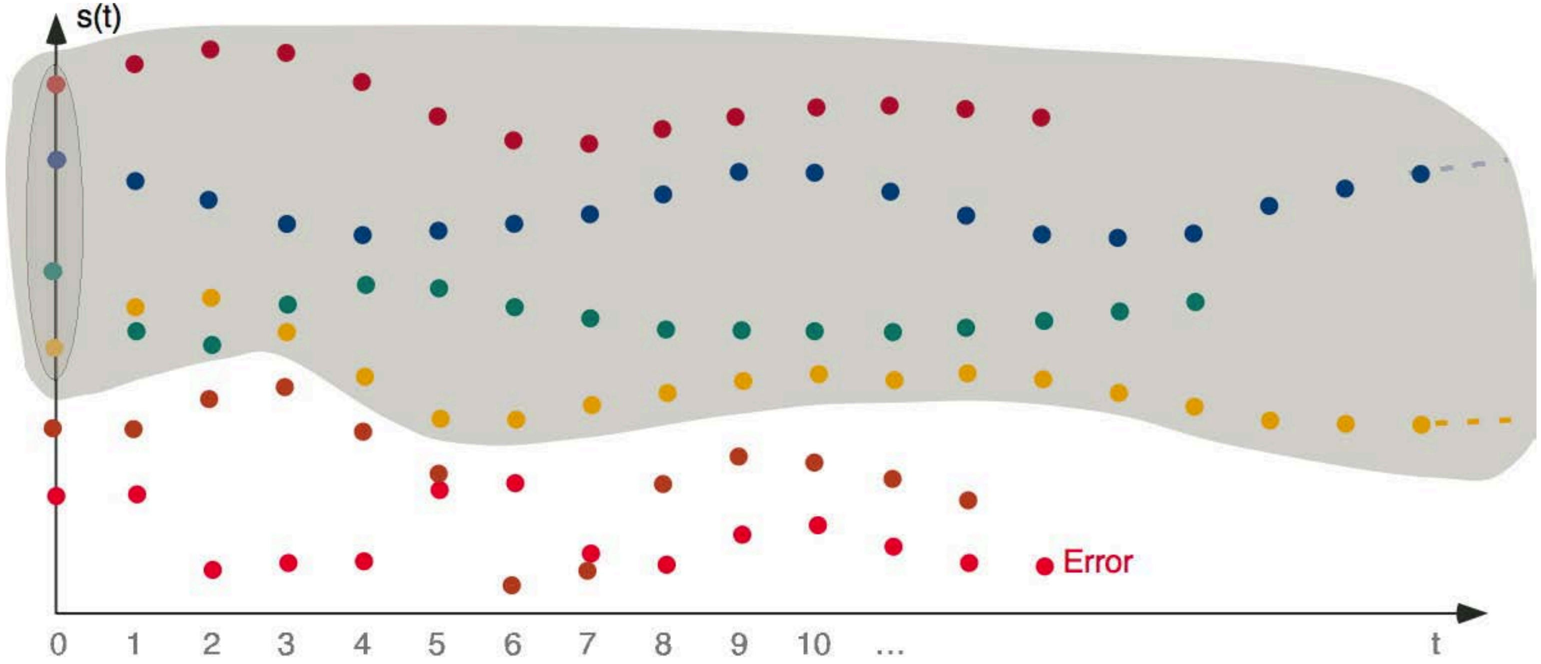
[10] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoret. Comput. Sci.*, 277(1—2):47–103, 2002.

Reachability Abstraction

Reachability Abstraction



Reachability Semantics (System Invariant)



Reachability Abstraction

– The reachability abstraction collects states along traces.

$$
\begin{aligned}
\alpha_r &\in \wp(\mathcal{T}) \mapsto \wp(\Sigma) \\
\alpha_r(T) &\triangleq \{\sigma_i \mid \exists n \in [0, +\infty] : \sigma \in \Sigma^n \cap T \wedge \\
&\qquad\qquad i \in [0, n[\} \\
&\subseteq^{4} \{s' \in \Sigma \mid \exists s \in \iota : \langle s,\ s' \rangle \in \tau^\star\}
\end{aligned}
$$

where $\alpha_{\iota\tau}(T) = \langle \iota,\ \tau \rangle$ is the transition abstraction

and $\tau^\star$ is the reflexive transitive closure of τ.

4 We may have $\subsetneq$ when $T \neq \gamma_{\iota\tau}(\alpha_{\iota\tau}(T))$. We assume $T = \gamma_{\iota\tau}(\alpha_{\iota\tau}(T))$ in the rest of the talk.

Invariants

– Expressed in logical form, the reachability abstraction α provides a system invariant

$$\alpha(\mathcal{C}[\![\mathbb{p}]\!])$$

that is the set of all states that can be reached along some execution of the system $\mathbb{p}$ [11], [12].

References

[11] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 1978.

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

Floyd's Proof Method

– Floyd's method [13] to prove a reachability property

$$\alpha_r(T) \subseteq P$$

consists in finding an invariant I stronger than P, i.e.

$$I \subseteq P$$

which is inductive, i.e.

$$\iota \subseteq I$$

and $$\tau[I] \subseteq I$$

where $$\tau[I] \triangleq \{s' \mid \exists s \in I : \langle s,\ s'\rangle \in \tau\}$$

is the right-image transformer for the transition system $\langle \iota,\ \tau \rangle = \alpha_{\iota\tau}(T)$.

References

[13] R. Floyd. Assigning meaning to programs. *Proc. Symp. in Applied Math.*, vol. 19, 19–32. AMS, 1967.

Floyd's Proof Method (Cont'd)

– This induction principle has many equivalent variants [14], all underlying different static analysis methods (the equivalence may not be preserved by abstraction).

– In particular backward analyzes are based on

$$\langle \tau^{-1}, \alpha_{\varphi}(T) \rangle$$

where τ^{-1} is the inverse of τ

and $\alpha_{\varphi}(T) \triangleq \{\sigma_{n-1} \mid n < +\infty \wedge \sigma \in T \cap \Sigma^{n}\}$

collects final states.

References

[14] P. Cousot and R. Cousot. Induction principles for proving invariance properties of programs. *Tools & Notions for Program Construction*, 43–119. Cambridge U. Press, 1982.

6. Properties of Abstractions

Soundness of Abstractions

– An abstraction is sound [15] if the proof in the abstract implies the concrete property

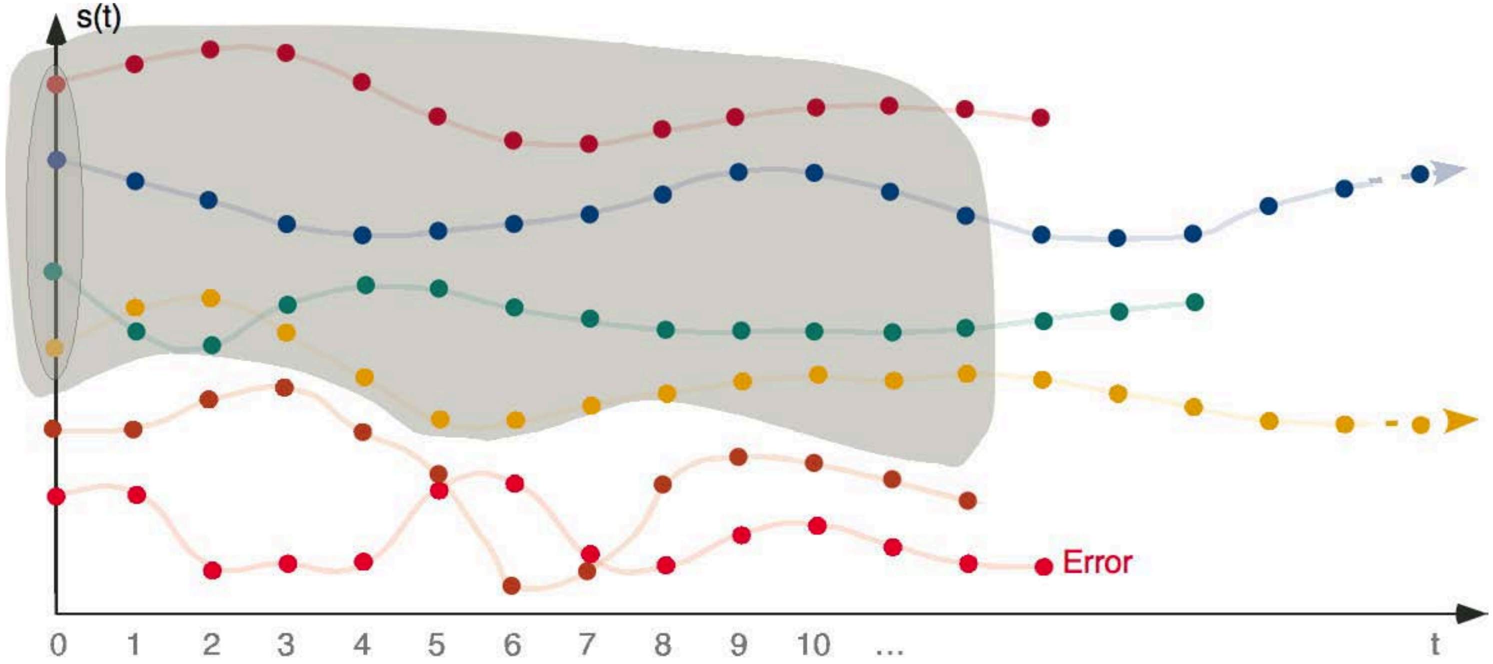
$$\mathcal{C}^\sharp[\![\mathbb{p}]\!] \sqsubseteq P^\sharp \Longrightarrow \mathcal{C}[\![\mathbb{p}]\!] \subseteq P \ .$$

– Abstract interpretation provides an effective theory to design sound abstractions.

References

[15] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th ACM POPL*, 269–282, 1979.

Example of Unsound Abstraction (Bounded Model Checking)



Completeness of Abstractions

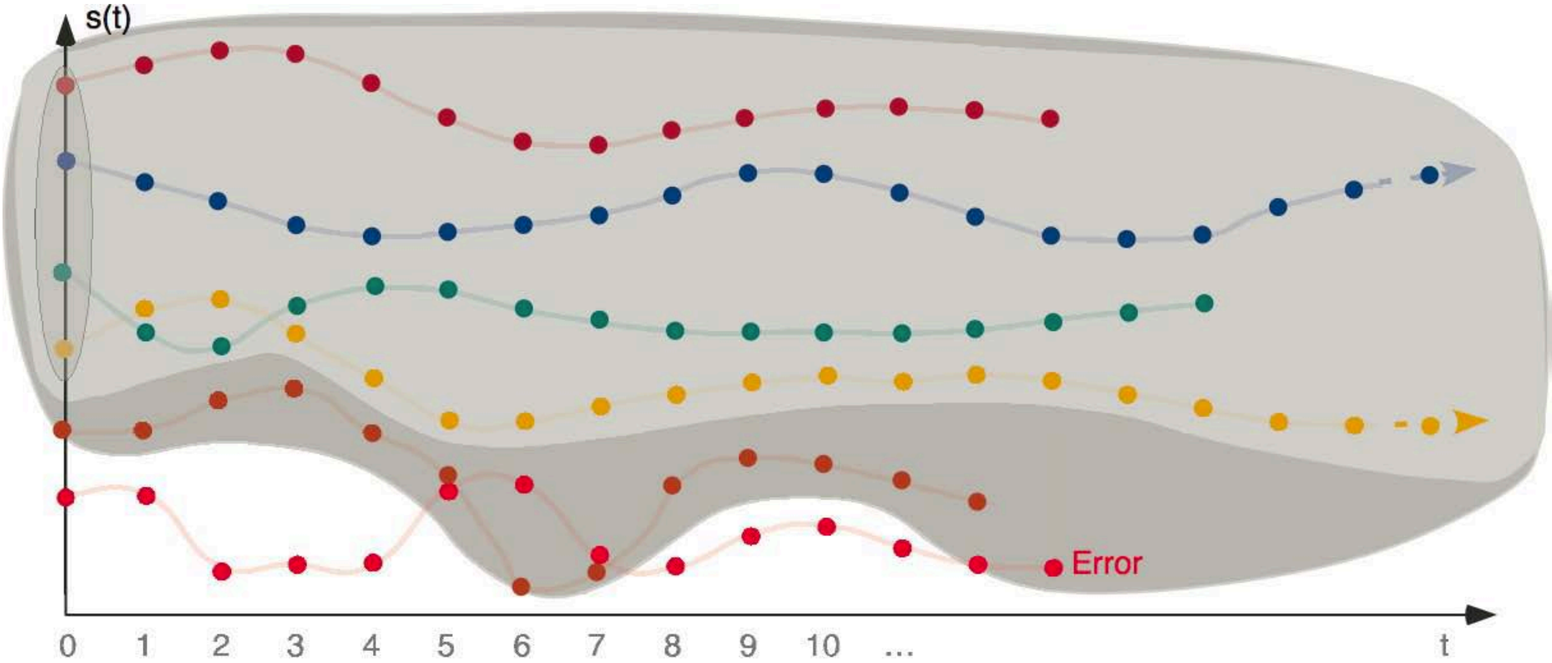
– An abstraction is complete [16] if the fact that the system is correct can always be proved in the abstract

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P \Longrightarrow \mathcal{C}^\sharp[\![\mathbb{p}]\!] \sqsubseteq P^\sharp.$$

References

[16] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th ACM POPL*, 269–282, 1979.

Example of Incomplete Abstraction (Static Analysis)



No error is reachable in the concrete but an error is reachable in the abstract $\Rightarrow$ the proof fails in the abstract (false alarm)!

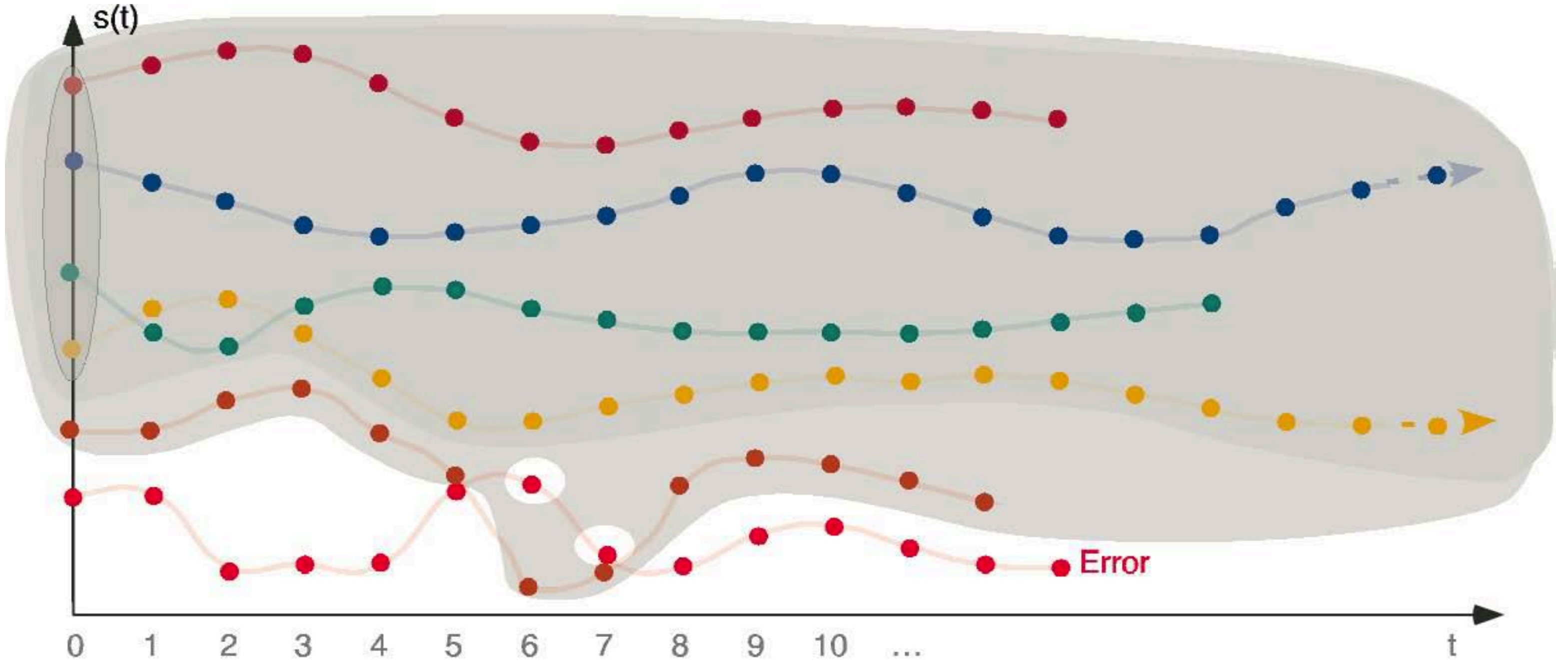
Refinement of Abstractions

– False alarms can always be avoided by refinement of the abstraction [17].

References

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

Example of Refined Abstraction (Static Analysis)



No error is reachable in the abstract whence in the concrete $\Rightarrow$ the proof succeeds in the abstract!

Incompleteness of the Refinement of Abstractions

– This refinement is not effective (i.e. the algorithm does not terminate in general).

– For example in model-checking any abstraction of a trace logic may be incomplete [18].

References

[18] R. Giacobazzi and F. Ranzato. Incompleteness of states w.r.t. traces in model checking. *Inform. and Comput.*, 204(3):376–407, Mar. 2006.

Adequation of Abstractions

- The reachability abstraction is sound and complete for invariance/safety proofs [5].
- That means that if $S \subseteq \Sigma$ is a set of safe states so that $\gamma_r(S)$ is a set of safe traces then the safety proof $\mathcal{C}[\![\mathbb{p}]\!] \subseteq \gamma_r(S)$ can always be done as $\alpha_r(\mathcal{C}[\![\mathbb{p}]\!]) \subseteq S$.
- This is the fundamental remark of Floyd [19] that it is not necessary to reason on traces to prove invariance properties.

References

[19] R. Floyd. Assigning meaning to programs. *Proc. Symp. in Applied Math.*, vol. 19, 19–32. AMS, 1967.

5 Again, assuming $T = \gamma_{\iota\tau}(\alpha_{\iota\tau}(T))$

Adequation of Abstractions (Cont'd)

– This does not mean that this abstraction is adequate, that is, informally, the most simple way to do the proof.

– For example Burstall's intermittent assertions may be simpler than Floyd's invariant assertions [20]

– or, in static analysis trace partitioning may be more adequate that state-based reachability analysis [21].

References

[20] P. Cousot and R. Cousot. Sometime = always + recursion ≡ always: on the equivalence of the intermittent and invariant assertions methods for proving inevitability properties of programs. *Acta Informat.*, 24:1–31, 1987.

[21] L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzer. *14th ESOP*, LNCS 3444, 5–20. Springer, 2005.

7. Static Analysis

Principle of Static Analysis

Static Analysis

– Static code analysis is the analysis of computer system
  - by direct inspection of the source or object code describing this system
  - with respect to a semantics of this code (no user-provided model)
  - without executing programs as in dynamic analysis.
– The static code analysis is performed by an automated tool, as opposed to program understanding or program comprehension by humans.

Verification by Static Analysis

– The proof

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P$$

is done in the abstract

$$\mathcal{C}^\sharp[\![\mathbb{p}]\!] \sqsubseteq P^\sharp ,$$

which involves the static analysis of $\mathbb{p}$ that is the effective computation of the abstract semantics

$$\mathcal{C}^\sharp[\![\mathbb{p}]\!] ,$$

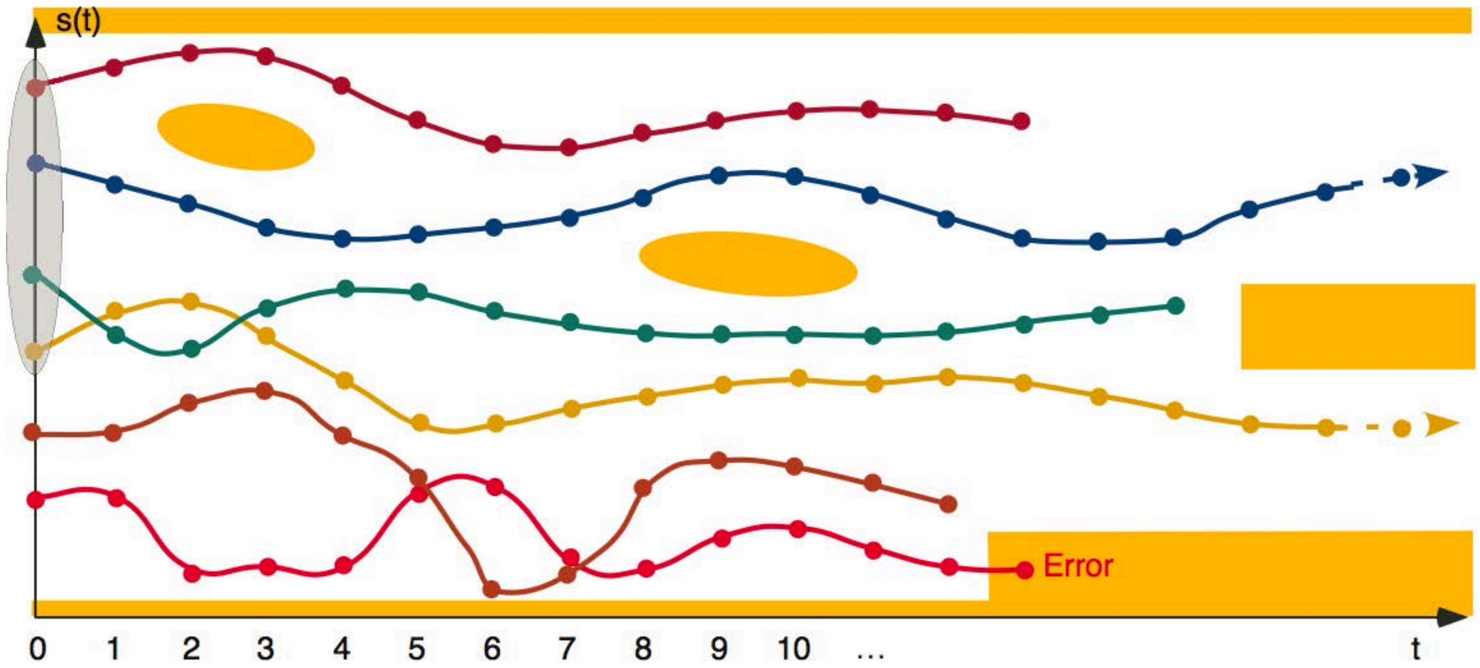
as formalized by abstract interpretation [22], [23].

References

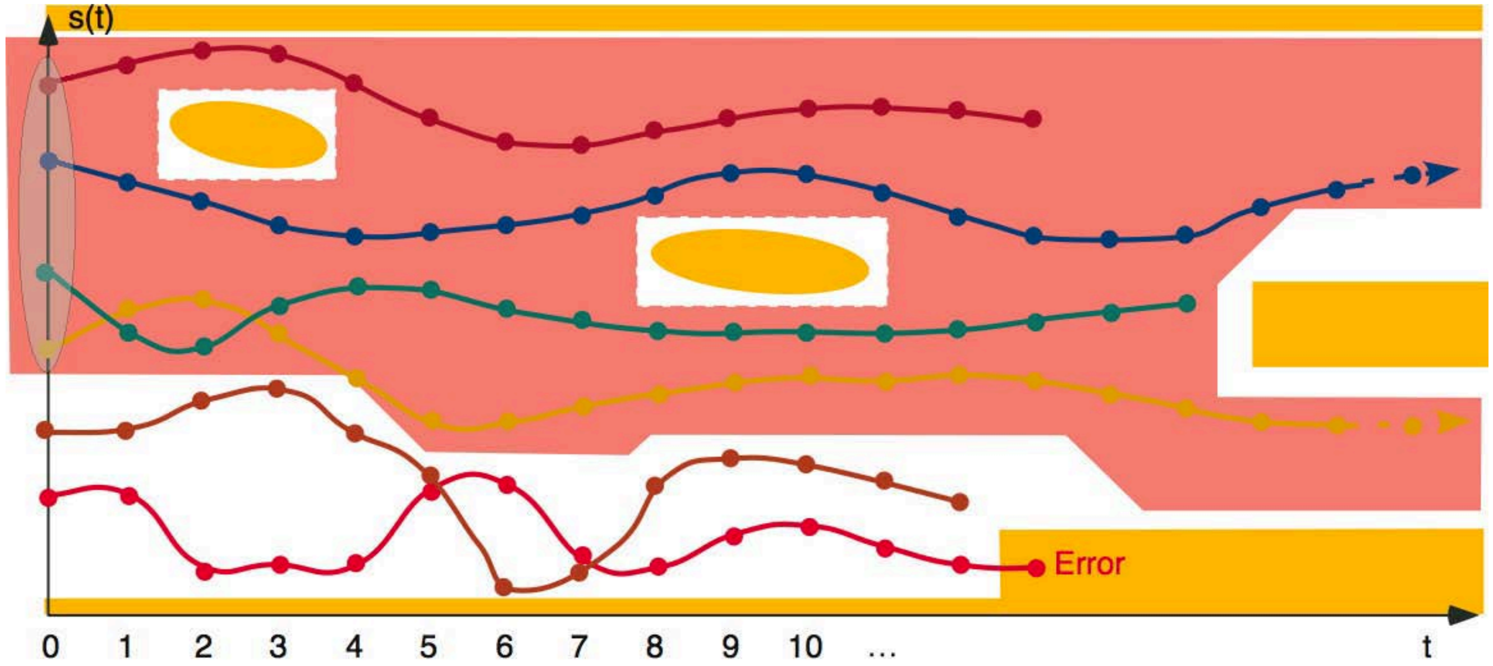
[22] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 1978.

[23] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th ACM POPL*, 269–282, 1979.

Semantics and Specification



Abstract Semantics and Verification



Model versus Property and Program versus Language-based Abstraction

Property-based Abstraction

– Property-based abstraction over approximate the collecting semantics in the abstract

  - $\mathcal{C}[\![\mathbb{p}]\!] = \{\mathcal{S}[\![\mathbb{p}]\!]\} \in \mathcal{P}$ collecting semantics
  - $\langle \mathcal{P}, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{P}^\sharp, \sqsubseteq^\sharp \rangle$ abstraction
  - $\mathcal{C}^\sharp[\![\mathbb{p}]\!] \in \mathcal{P}^\sharp$ abstract semantics
  - $\mathcal{C}[\![\mathbb{p}]\!] \subseteq \gamma(\mathcal{C}^\sharp[\![\mathbb{p}]\!])$ soundness

$\Rightarrow$ an abstract proof ($\mathcal{C}^\sharp[\![\mathbb{p}]\!] \sqsubseteq^\sharp P^\sharp$) is valid in the concrete ($\mathcal{C}[\![\mathbb{p}]\!] \subseteq \gamma(P^\sharp)$).

Model-based Abstraction

– Let $\langle \iota, \ \tau \rangle$ be a transition system model of a software or hardware system $\mathbb{p} \in \mathbb{P}$ (so that $\mathcal{S}[\![\mathbb{p}]\!] \triangleq \gamma_{\iota\tau}(\langle \iota, \ \tau \rangle))$.

– A model-based abstraction is an abstract transition system $\langle \iota^\sharp, \tau^\sharp \rangle$ which over-approximates $\langle \iota, \tau \rangle$ (so that, up to concretization, $\iota \subseteq \iota^\sharp$ and $\tau \subseteq \tau^\sharp$).

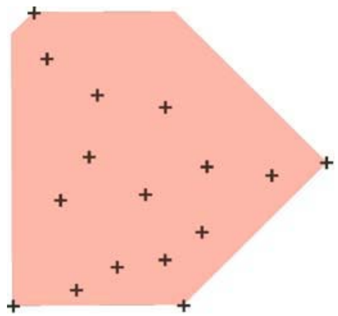
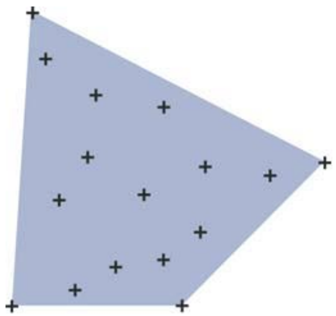
– The set of reachable abstract states for $\langle \iota^\sharp, \ \tau^\sharp \rangle$ over-approximate the reachable concrete states of $\langle \iota, \ \tau \rangle$

– Hence the model-based abstractions yields sound abstractions of the concrete reachability states.

Is the model-based abstraction "adequate"?

Limitations of Model-based Abstractions

– Some abstractions defined by a Galois connection of sets of (reachable) states are not be model-based abstractions, in particular when the abstract domain is not a representable as a powerset of states, e.g.

Octagons [25]   Polyhedra [24]

References

[24] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. 5^{th} POPL, pp. 84–97, ACM Press, 1978.

[25] A. Miné. The octagon abstract domain. *Higher-Order and Symb. Comp.*, 19:31–100, 2006.

Program-based versus Language-based Abstraction

- Static analysis has to define an abstraction $\alpha[\![\mathbb{p}]\!]$ for all programs $\mathbb{p} \in \mathbb{P}$ of a language $\mathbb{P}$.
- This is different from defining an abstraction specific to a *given* program (or model).

Program-based versus Language-based Abstraction

– An abstraction specific to a given program can always be refined to be complete using a finite abstract domain [26].

– This is impossible in general for a language-based abstraction for which infinite abstract domains have been shown to always produce better results [27].

References

[26] P. Cousot. Partial completeness of abstract fixpoint checking. *SARA*, LNAI 1864, 1–25. Springer, 2000.

[27] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. *PLILP '92*, LNCS 631, 269–295. Springer, 1992.

False Alarms

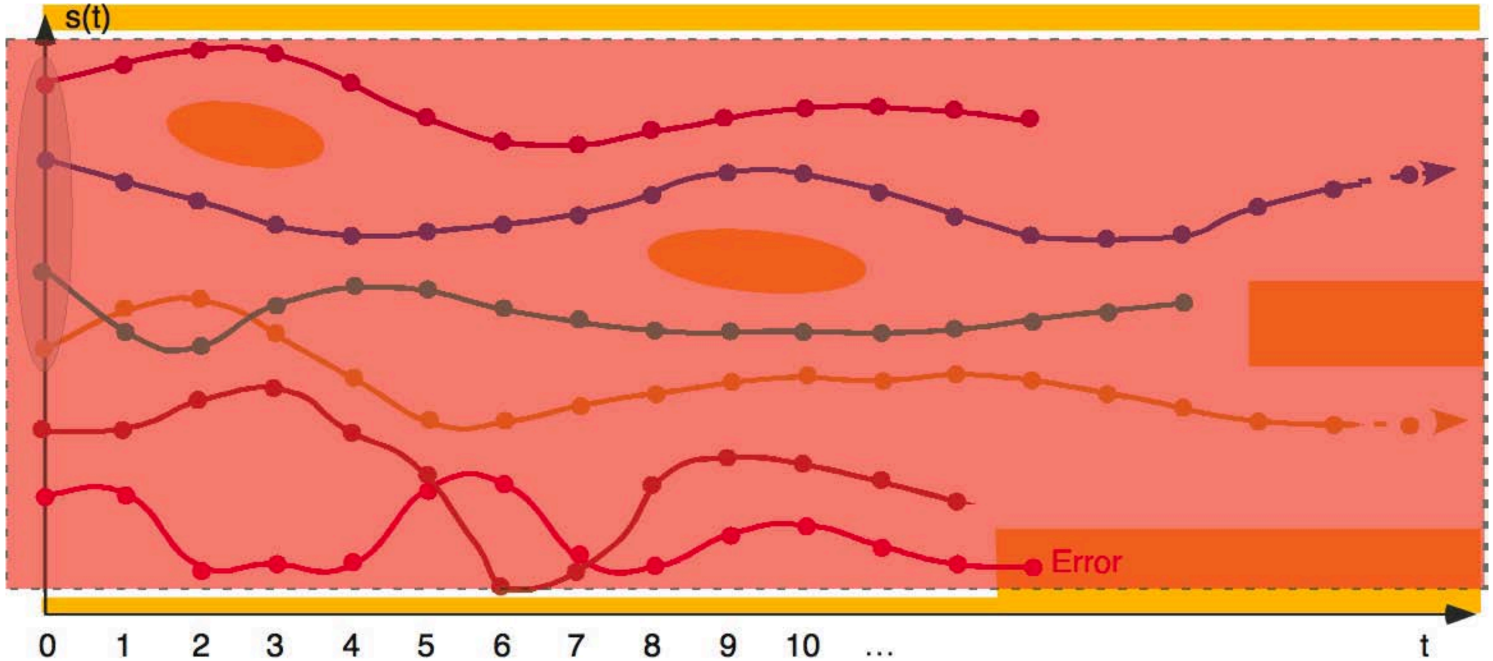
False Alarms

– Static analysis being undecidable, it relies on incomplete language-based abstractions.

– A false alarm is a case when a concrete property holds but this cannot be proved in the abstract for the given abstraction.

False Alarm

– An example in reachability analysis is when no inductive invariant can be expressed in the abstract.



False Alarms (Cont'd)

- The experience of ASTRÉE (`www.astree.ens.fr`, [28]) shows that it is possible to design precise language-based abstractions which produce no false alarm on a well defined *families of programs*[6].

- Nevertheless, by indecidability, the analyzer will produce false alarms on infinitely many programs (which can even be generated automatically).

References

[28] 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. *ACM PLDI*, 196–207, 2003.

[6] Synchronous, time-triggered, real-time, safety critical, embedded software written or automatically generated in the C programming language for ASTRÉE.

Design of Abstractions

Design of Abstractions

– The design of a sound and precise language-based abstraction is difficult.

– First from a mathematical point of view, one must discover the appropriate set of abstract properties that are needed to represent the necessary inductive invariants.

– Of course mathematical completion techniques could be used [29] but because of undecidability, they do not terminate in general.

References

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

Design of Abstractions (Cont'd)

– Second, from a computer-science point of view, one must find an appropriate computer representation of abstract properties and abstract transformers.

– Universal representations (e.g. using symbolic terms, automata or BDDs) are in general inefficient

– The discovery of appropriate computer representations is far from being automatized.

Local versus Global Abstractions

– A simple approach to static analysis is to use the same global abstraction everywhere in the program, which hardly scales up.

– More sophisticated abstractions, as used in ASTRÉE are not uniform, different local abstractions being in different program regions [30].

References

[30] 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. *ACM PLDI*, 196–207, 2003.

Multiple versus Single Abstractions

– Because of the complexity of abstractions, it is simpler to design a precise abstraction by composing many elementary abstractions which are simple to understand and implement.

Abstractions in Astrée

Multiple versus Single Abstractions (Cont'd)

– ASTRÉE uses many weakly relational domains (such as octagons [33], digital filters [31], arithmetico-geometric progressions [32], etc)
– These abstract domains could hardly be encoded efficiently using a universal representation of program properties as found in theorem provers, proof assistants or model-checkers.

References

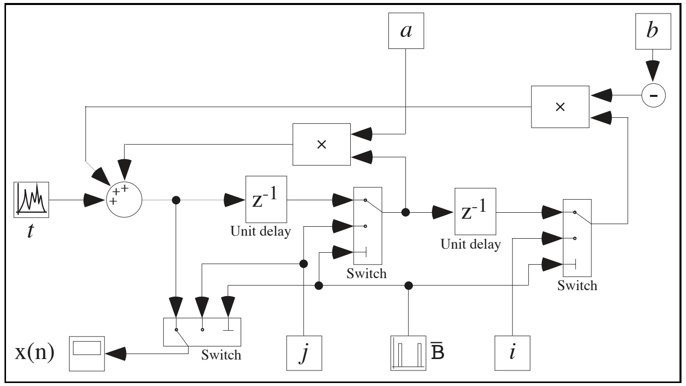
[31] J. Feret. Static analysis of digital filters. *30th ESOP*, LNCS 2986, 33–48. Springer, 2004.

[32] J. Feret. The arithmetic-geometric progression abstract domain. *6th VMCAI*, LNCS 3385, 42–58, Springer, 2005.

[33] A. Miné. The octagon abstract domain. *Higher-Order and Symb. Comp.*, 19:31–100, 2006.

Ellipsoid Abstract Domain for Filters

2^{d} Order Digital Filter:

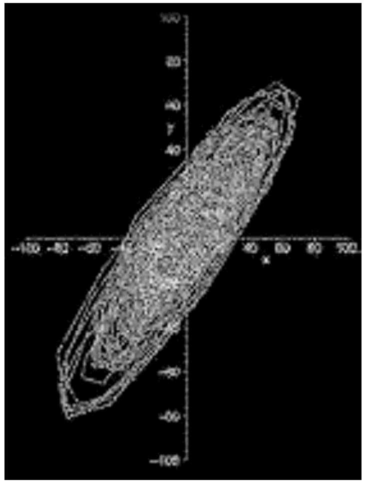


– Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$

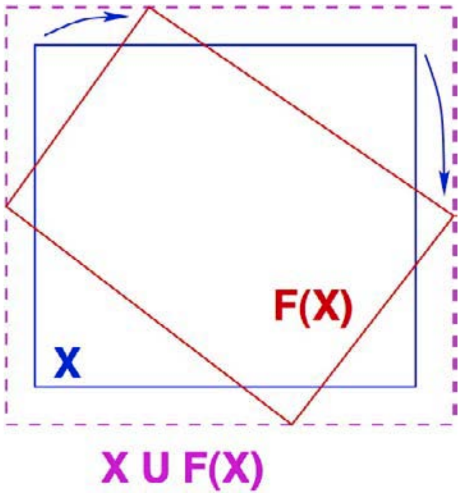
– The concrete computation is bounded, which must be proved in the abstract.

– There is no stable interval or octagon.

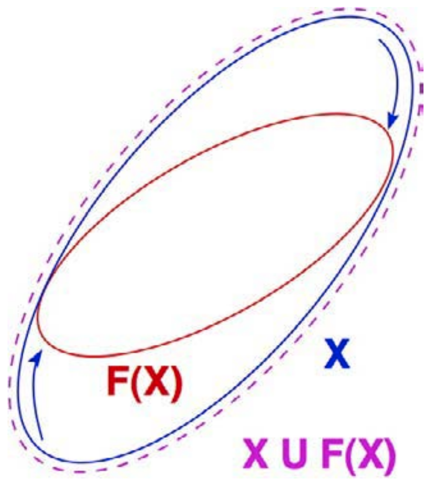
– The simplest stable surface is an ellipsoid.



execution trace



unstable interval



stable ellipsoid

Filter Example [34]

```
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

[34] J. Feret. Static analysis of digital filters. *30th ESOP*, LNCS 2986, 33–48. Springer, 2004.

Abstraction Reduction

- If several abstractions are used, the static analyzer must implement their conjunction in the concrete, that is their reduced product in the abstract [35].
- The implementation must be extensible, allowing for the easy incorporation of new abstractions [36].

References

[35] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th ACM POPL*, 269–282, 1979.

[36] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the ASTRÉE static analyzer, invited paper. *11th ASIAN*, LNCS, Springer (to appear).

Refinement

– For a single program-based abstraction, refinement consists in strengthening the abstract invariant until it is inductive, through a combination of fixpoint computations [37] which in general does not terminate or explodes combinatorially.

– The problem is even harder for language-based abstractions.

References

[37] P. Cousot, P. Ganty, and J.-F. Raskin. Fixpoint-Guided Abstraction Refinements. In *Proc. Fourteenth International Symposium on Static Analysis (SAS '07)*, G. Filé & H. Riis-Nielson (Eds), pages 333–348, Kongens Lyngby, Denmark, 22–24 August 2007. Lecture Notes in Computer Science, volume 4634, Springer, Berlin, pp. 333–348.

Refinement (Cont'd)

- The pragmatic approach used in ASTRÉE is to manually design new abstractions which are incorporated in the reduced product of all abstractions used by the analyzer
- This allowing for the strengthening of the abstract invariants for all programs until no false alarm is left [38].

References

[38] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the ASTRÉE static analyzer, invited paper. *11th ASIAN*, LNCS, Springer (to appear).

Arithmetic-geometric progressions[7] [39]

– Abstract domain: $(\mathbb{R}^+)^5$

– Concretization:

$$\gamma \in (\mathbb{R}^+)^5 \longmapsto \wp(\mathbb{N} \mapsto \mathbb{R})$$

$$\gamma(M, a, b, a', b') =$$
$$\{f \mid \forall k \in \mathbb{N} : |f(k)| \leq \left(\boldsymbol{\lambda} x \cdot ax + b \circ (\boldsymbol{\lambda} x \cdot a'x + b')^k\right)(M)\}$$

i.e. any function bounded by the arithmetic-geometric progression.

Reference

[39] J. Feret. The arithmetic-geometric progression abstract domain. 6^{th} *VMCAI*, LNCS 3385, 42–58, Springer, 2005.

[7] here in $\mathbb{R}$

Arithmetic-Geometric Progressions (Example 1)

```
% cat count.c
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
volatile BOOLEAN I; int R; BOOLEAN T;
void main() {
  R = 0;
  while (TRUE) {
    __ASTREE_log_vars((R));
    if (I) { R = R + 1; }          ← potential overflow!
    else { R = 0; }
    T = (R >= 100);
    __ASTREE_wait_for_clock(());
  }}
% cat count.config
__ASTREE_volatile_input((I [0,1]));
__ASTREE_max_clock((3600000));
% astree -exec-fn main -config-sem count.config count.c|grep '|R|'
|R| <= 0. + clock *1. <= 3600001.
```

Arithmetic-geometric progressions (Example 2)

```
% cat retro.c
typedef enum {FALSE=0, TRUE=1} BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

void dev( )
{ X=E;
  if (FIRST) { P = X; }
  else
    { P =  (P - ((((2.0 * P) - A) - B)
              * 4.491048e-03)); };
  B = A;
  if (SWITCH) {A = P;}
  else {A = X;}
}
```

```
void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock(());
  }}
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));
```

```
|P| <= (15.  + 5.87747175411e-39
/ 1.19209290217e-07) * (1
+ 1.19209290217e-07)^clock
- 5.87747175411e-39 /
1.19209290217e-07 <=
23.0393526881
```

The Breakthrough of ASTRÉE

Industrial Application of ASTRÉE

- Automatic (no user specification, no user model, no necessary user intercaction with the analysis)
- Scales up (to 1.000.000 LOCs)
- Precise (refinable by parametrization and analysis directives whence no false alarm!)
- Relatively fast (2/3 hours per 100.000 LOCs)
- Used by Airbus France [40]

References

[40] D. Delmas and J. Souyris. ASTRÉE from research to industry. *14th SAS*, LNCS 4634, Springer, Aug. 2007, pp. 437–451.

8. Conclusion

Conclusion

– The behaviors of computer systems are too large and complex for enumeration (state/combinatorial explosion);

– Abstraction is therefore necessary to reason or compute behaviors of computer systems;

– Making explicit the rôle of abstract interpretation in formal methods might be fruitful;

– In particular to apply formal methods to complex industrial applications.

THE END, THANK YOU

9. Bibliography

Bibliography

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21:181–185, 1985.

[BCC+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. In *Proc. ACM SIGPLAN '2003 Conf. PLDI*, pages 196–207, San Diego, CA, US, 7–14 June 2003. ACM Press.

[CC77a] P. Cousot and 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.

[CC77b] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E.J. Neuhold, editor, *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA*, pages 237–277. North-Holland, 1977.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6^{th} POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

[CC82] P. Cousot and R. Cousot. Induction principles for proving invariance properties of programs, invited chapter. In D. Néel, editor, *Tools & Notions for Program Construction*, pages 43–119. Cambridge U. Press, 1982.

[CC87] P. Cousot and R. Cousot. Sometime = always + recursion ≡ always: on the equivalence of the intermittent and invariant assertions methods for proving inevitability properties of programs. *Acta Informat.*, 24:1–31, 1987.

[CC92] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proc. 4^{th} Int. Symp. PLILP '92*, Leuven, BE, 26–28 Aug. 1992, LNCS 631, pages 269–295. Springer, 1992.

[CC02] P. Cousot and R. Cousot. Modular static program analysis, invited paper. In R.N. Horspool, editor, *Proc. 11^{th} Int. Conf. CC '2002*, pages 159–178, Grenoble, FR, 6–14 Apr. 2002. LNCS 2304, Springer.

[CCF+06] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the Astrée static analyzer, invited paper. In M. Okada and I. Satoh, editors, *11^{th} ASIAN 06*, Tokyo, JP, 6–8 Dec. 2006. LNCS , Springer. To appear.

[CGR07] P. Cousot, P. Ganty, and J.-F. Raskin. Fixpoint-guided abstraction refinements. In G. Filé and H. Riis-Nielson, editors, *Proc. 14th Int. Symp. SAS '07*, Kongens Lyngby, DK, LNCS 4634, pages 333–348. Springer, 22–24 Aug. 2007.

[CH78] 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.

[Cou78] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, FR, 21 Mar. 1978.

[Cou00] P. Cousot. Partial completeness of abstract fixpoint checking, invited paper. In B.Y. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, TX, US, LNAI 1864, pages 1–25. Springer, 26–29 Jul. 2000.

[Cou02] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoret. Comput. Sci.*, 277(1—2):47–103, 2002.

[DS07] D. Delmas and J. Souyris. ASTRÉE: from research to industry. In G. Filé and H. Riis-Nielson, editors, *Proc. 14th Int. Symp. SAS '07*, Kongens Lyngby, DK, LNCS 4634. Springer, 22–24 Aug. 2007.

[Fer04] J. Feret. Static analysis of digital filters. In D. Schmidt, editor, *Proc. 30th ESOP '2004, Barcelona, ES*, volume 2986 of *LNCS*, pages 33–48. Springer, Mar. 27 – Apr. 4, 2004.

[Fer05] J. Feret. The arithmetic-geometric progression abstract domain. In R. Cousot, editor, *Proc. 6th Int. Conf. VMCAI 2005*, pages 42–58, Paris, FR, 17–19 Jan. 2005. LNCS 3385, Springer.

[Flo67] R.W. Floyd. Assigning meaning to programs. In J.T. Schwartz, editor, *Proc. Symposium in Applied Mathematics*, volume 19, pages 19–32. AMS, 1967.

[GR06] R. Giacobazzi and F. Ranzato. Incompleteness of states w.r.t traces in model checking. *Inform. and Comput.*, 204(3):376–407, Mar. 2006.

[GRS00] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

[Min06] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19:31–100, 2006.

[MR05] L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzer. In M. Sagiv, editor, *Proc. 14th ESOP '2005, Edinburg, UK*, volume 3444 of *LNCS*, pages 5–20. Springer, Apr. 2—-10, 2005.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57, Providence, RI, Nov. 1977.