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

International Symposium in Honor of Zohar Manna
Taormina, Sicily, Italy
June 29 – July 4, 2003

Talk Outline

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 6^{th} *POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

Complete Lattice of 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 $\{0, 2, 4, 6, \ldots\}$

- 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;

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:

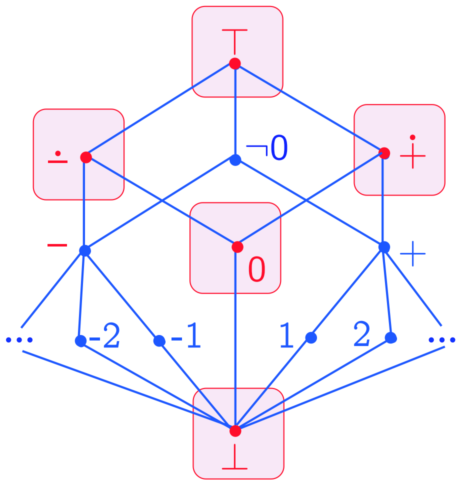
  $\overline{\mathcal{A}}$ is a Moore family

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

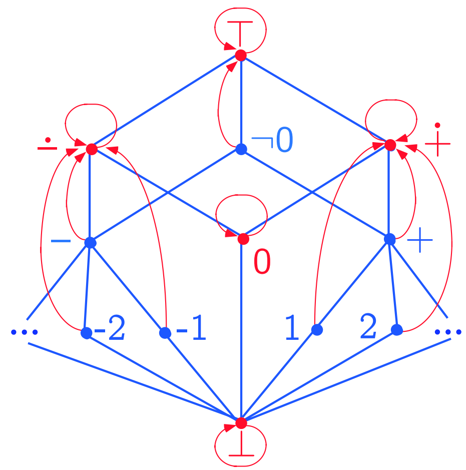
$$\rho_{\overline{\mathcal{A}}}(P) = \bigcap\{\overline{P} \in \overline{\mathcal{A}} \mid P \subseteq \overline{P}\}$$

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



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:

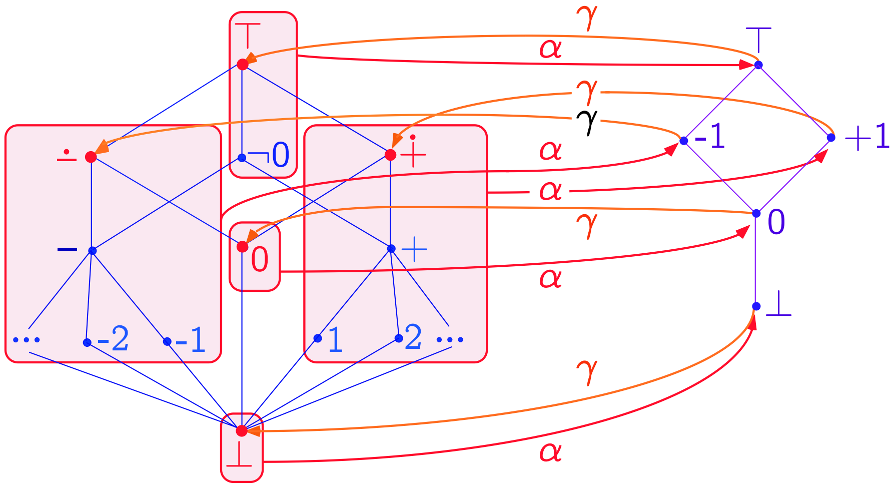
  $$\langle \wp(\Sigma),\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{\mathcal{D}},\ \sqsubseteq\rangle$$

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

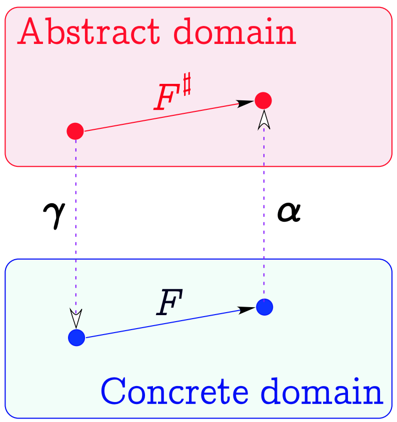
  $$\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \;\Leftrightarrow\; P \subseteq \gamma(\overline{P});$$

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

Example of Galois Connection-Based Abstraction



Function Abstraction



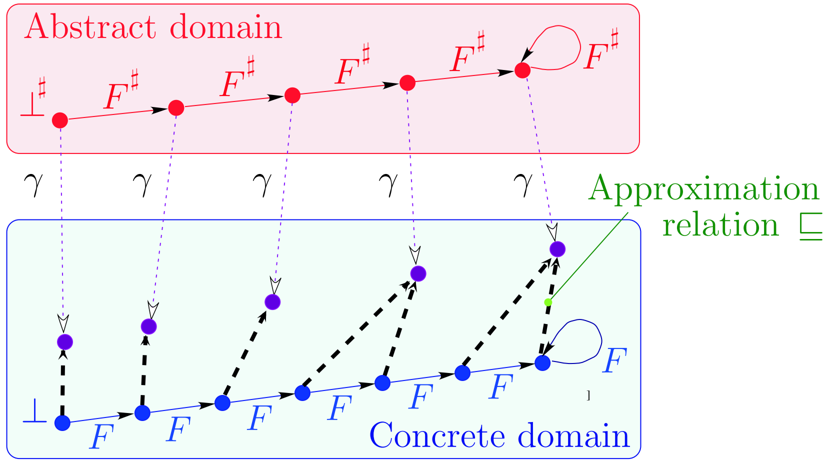
$F^\sharp = \alpha \circ F \circ \gamma$

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

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

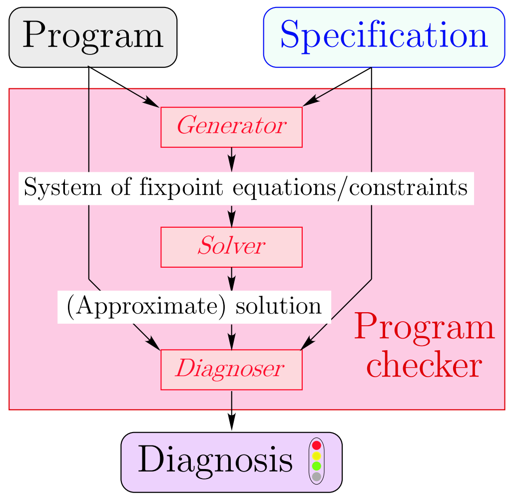
$$\langle P \overset{\mathrm{mon}}{\longmapsto} P, \dot{\subseteq}\rangle \overset{\lambda F^\sharp \cdot \gamma\circ F^\sharp\circ\alpha}{\underset{\lambda F \cdot \alpha\circ F\circ\gamma}{\leftrightarrows}} \langle Q \overset{\mathrm{mon}}{\longmapsto} Q, \dot{\sqsubseteq}\rangle$$

Approximate Fixpoint Abstraction



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

Program Checking by Static Analysis



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: $\mathcal{M} = \mathbb{X} \mapsto \mathcal{V}$

Program Properties [1]

$$P \in \wp(\mathcal{L} \times \mathcal{M})$$

[1] e.g. for reachability.

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:

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

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

- Choose 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:

$$\langle \wp(\mathcal{M}),\ \subseteq \rangle \xrightleftharpoons[\alpha_{\mathbb{P}}]{\gamma_{\mathbb{P}}} \langle \wp(\mathbb{P}),\ \supseteq \rangle$$

where:

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

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

(In practice one uses an isomorphic Boolean encoding)

Pointwise Extension to All program Points

By pointwise extension, we have for all program points:

$$\langle \mathcal{L} \mapsto \wp(\mathcal{M}),\ \dot{\subseteq} \rangle \xrightleftharpoons[\dot{\alpha}_{\mathbb{P}}]{\dot{\gamma}_{\mathbb{P}}} \langle \mathcal{L} \mapsto \wp(\mathbb{P}),\ \dot{\supseteq} \rangle$$

where:

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

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

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

Composition: Pointwise Predicate Abstraction

By composition, we get:

$$\langle \wp(\mathcal{L} \times \mathcal{M}),\ \subseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{L} \mapsto \wp(\mathbb{P}),\ \dot{\supseteq} \rangle$$

where:

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

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

Abstract Predicate Transformer (Sketchy)

$$\alpha \circ \mathrm{post}[\![\mathtt{X:=E}]\!] \circ \gamma(\bigwedge_{i=1}^{n} q_i) \quad \text{where } \{q_1, \ldots, q_n\} \subseteq \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$$

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

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

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

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

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

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

Generic Abstraction

Generic Abstraction in Static Analysis

For program verification, one must discover/compute inductive assertions.

- Ground assertions (e.g. Floyd's invariants on variables attached to program points)
- Atomic assertions (e.g. predicate abstraction so the combination with $\vee$, $\wedge$, $\neg$ and the localization at program points are automated)
- Generic assertions (e.g. parameterized in terms of programs (such as variables))

Static analysis:

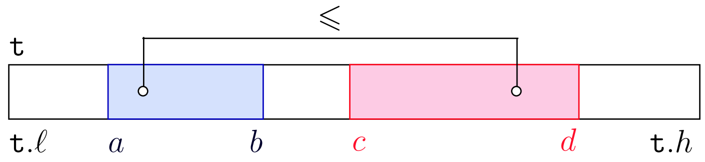
- Generic assertions: Abstract domains
- Combinations: Reduced product ($\wedge$), Disjunctive completion ($\vee$)

Example of generic abstraction: comparison

- Let $\mathcal{D}_{\mathrm{rel}}(X)$ be a generic relational integer abstract domain parameterized by a set X of variables (e.g. octagons or polyhedra);
- We define the generic comparison abstract domain:

$$\mathcal{D}_{\mathrm{lt}}(X) = \{\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle \mid \mathtt{t} \in X \wedge a, b, c, d \notin X \wedge r \in \mathcal{D}_{\mathrm{rel}}(X \cup \{\mathtt{t}.\ell, \mathtt{t}.h, a, b, c, d\})\}\ .$$

- Concretization:



Example: Bubble Sort [2]

var t : array [a, b] of int;
1 : $\{\mathrm{a} \le \mathrm{b}\}$
I := a;
2 : $\{\mathrm{I} = \mathrm{a} \le \mathrm{b}\}$
while (I < b) do
3 : $\{\mathrm{lt}(\mathrm{t}, \mathrm{a}, \mathrm{I}, \mathrm{I}, \mathrm{I}) \wedge \mathrm{I} < \mathrm{b}\}$
if (t[I] > t[I + 1]) then
4 : $\{\mathrm{lt}(\mathrm{t}, \mathrm{a}, \mathrm{I}, \mathrm{I}, \mathrm{I}) \wedge \mathrm{I} < \mathrm{b} \wedge \mathrm{lt}(\mathrm{t}, \mathrm{I}, \mathrm{I}+1, \mathrm{I}, \mathrm{I})\}$
t[I] :=: t[I + 1]
5 : $\{\mathrm{lt}(\mathrm{t}, \mathrm{a}, \mathrm{I}+1, \mathrm{I}+1, \mathrm{I}+1) \wedge \mathrm{I}+1 \le \mathrm{b}\}$
fi;
6 : $\{\mathrm{lt}(\mathrm{t}, \mathrm{a}, \mathrm{I}+1, \mathrm{I}+1, \mathrm{I}+1) \wedge \mathrm{I}+1 \le \mathrm{b}\}$
I := I + 1
7 : $\{\mathrm{lt}(\mathrm{t}, \mathrm{a}, \mathrm{I}, \mathrm{I}, \mathrm{I}) \wedge \mathrm{I} \le \mathrm{b}\}$
od
8 : $\{\mathrm{lt}(\mathrm{t}, \mathrm{a}, \mathrm{I}, \mathrm{I}, \mathrm{I}) \wedge \mathrm{I} = \mathrm{b} \wedge \mathrm{s}(\mathrm{t}, \mathrm{a}, \mathrm{b})\}$

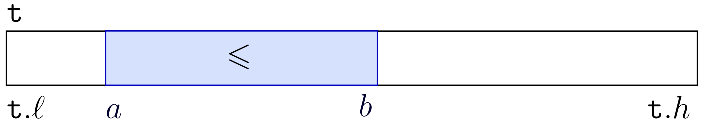
[2] Currently being implemented by Pavol Černy.

Example of generic abstraction: sorted

- Then we define the generic sorting abstract domain:

$$\mathcal{D}_s(X) = \{\langle \mathtt{s}(\mathtt{t}, a, b),\ r\rangle \mid \mathtt{t} \in X \wedge a, b \notin X \wedge r \in \mathcal{D}_{\mathrm{rel}}(X \cup \{\mathtt{t}.\ell, \mathtt{t}.h, a, b\})\}\ .$$

- The meaning $\gamma(\langle \mathtt{s}(\mathtt{t}, a, b),\ r\rangle)$ of an abstract predicate $\langle \mathtt{s}(\mathtt{t}, a, b),\ r\rangle$ is that the elements of $\mathtt{t}$ between indices a and b are sorted:



$$\gamma(\langle \mathtt{s}(\mathtt{t}, a, b),\ r\rangle) = \exists a, b : \mathtt{t}.\ell \leq a \leq b \leq \mathtt{t}.h \wedge \forall i, j \in [a, b] : (i \leq j) \Rightarrow (\mathtt{t}[i] \leq \mathtt{t}[j]) \wedge r\ .$$

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

Reference

[2] 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.

[3] 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.

A Parametric Specializable Static Program Analyzer

- C programs: safety critical embedded real-time synchronous software for 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;
- Semantics: ISO C99 + machine (IEEE 754-1985) + compiler + user;
- Implicit specification: absence of runtime errors, integer arithmetics should not wrap-around, etc;

The Class of Considered Periodic Synchronous Programs

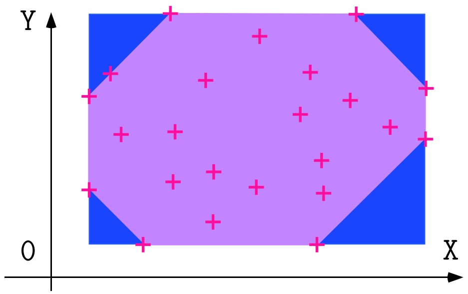
```
declare volatile input, state and output variables;
initialize state variables;
loop forever
    - read volatile input variables,
    - compute output and state variables,
    - write to volatile output variables;
  wait_for_clock ();
end loop
```

- 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.

General-Purpose Abstract Domains: Intervals and Octagons



Intervals:
$$\begin{cases} 1 \le x \le 9 \\ 1 \le y \le 20 \end{cases}$$

Octagons [5]:
$$\begin{cases} 1 \le x \le 9 \\ x + y \le 78 \\ 1 \le y \le 20 \\ x - y \le 03 \end{cases}$$

Difficulties: many global variables, IEEE 754 floating-point arithmetic (in program <u>and</u> analyzer)

Reference

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

Clock Abstract Domain

- Code Sample:

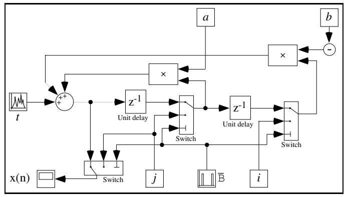
```
R = 0;
while (1) {
  if (I)
    { R = R+1; }
  else
    { R = 0; }
  T = (R>=n);
  wait_for_clock ();
}
```

- 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)*.

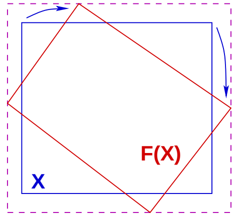
- Solution:
  - We add a phantom variable `clock` in the concrete user semantics to track elapsed clock ticks.
  - For each variable `X`, we abstract three intervals: `X`, `X+clock`, and `X-clock`.
  - If `X+clock` or `X-clock` is bounded, so is `X`.

Ellipsoid Abstract Domain

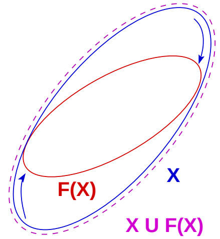
2d Order Filter Sample:



- 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.

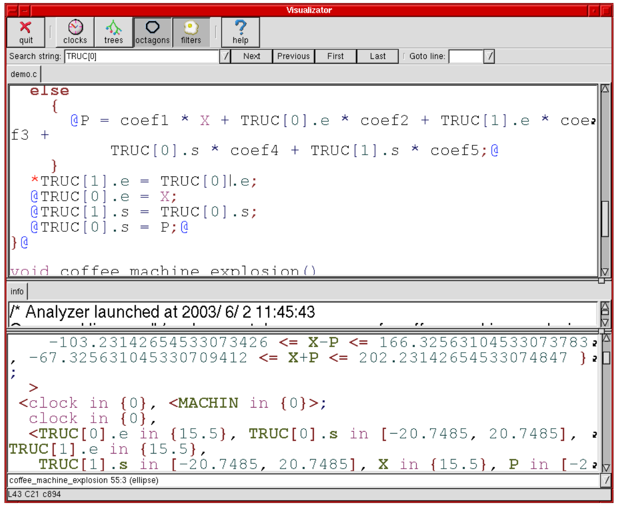


unstable interval



stable ellipsoid

Example of Analysis Session



Benchmarks

- Comparative results (commercial software):

  4,200 (false?) alarms,

  3.5 days;
- Our results:

  3 (false?) alarms,

  48 mn on 2.8 GHz PC,

  200 Megabytes.

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 ($x+\text{clk} \in [a;b] \wedge x-\text{clk} \in [a;b]$)
- 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.

Conclusion

Abstract Interpretation

- Abstract interpretation theory formalizes the idea of sound approximation for mathematical constructs involved in the specification of properties of computer systems.

References

[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, 1977.

[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.

[PO- PL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *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]

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

Conclusion on Verification by Abstraction

- 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 END, THANK YOU

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