Abstract Interpretation: Past, Present, and Future

Patrick Cousot  Radhia Cousot

cims.nyu.edu/~pcousot di.ens.fr/~rcousot

CSL — LICS
Vienna, Austria — July 14 – 18, 2014

Formal methods

• Reasonings on programs are
  • Reasonings on properties of their semantics (i.e. execution behaviors)
  • Always involve some form of abstraction

Abstract interpretation

• A theory establishing a correspondance between
  • Concrete semantic properties
    ↑ what you want to prove on the semantics
  • Abstract properties
    ↑ how to prove it in the abstract

• Objective: formalize
  • formal methods
  • algorithms for reasoning on programs
Fundamental motivations

Scientific research

• in Mathematics/Physics:
  trend towards unification and synthesis through universal principles

• in Computer science:
  trend towards dispersion and parcelization through a collection of local techniques for specific applications

An exponential process, will stop!

Example: reasoning on computational structures

<table>
<thead>
<tr>
<th>WCET</th>
<th>Security protocol verification</th>
<th>Systems biology analysis</th>
<th>Operational semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axiomatic semantics</td>
<td>Dataflow analysis</td>
<td>Model checking</td>
<td>Abstraction refinement</td>
</tr>
<tr>
<td>Confidentiality analysis</td>
<td>Database query</td>
<td>Dependence analysis</td>
<td>Type inference</td>
</tr>
<tr>
<td>Program synthesis</td>
<td>Denotational semantics</td>
<td>CEGAR</td>
<td>Separation logic</td>
</tr>
<tr>
<td>Grammar analysis</td>
<td>Theories combination</td>
<td>Program transformation</td>
<td>Termination proof</td>
</tr>
<tr>
<td>Statistical model-checking</td>
<td>Code Interpolants</td>
<td>Abstract model checking</td>
<td>Shape analysis</td>
</tr>
<tr>
<td>Invariance proof</td>
<td>Symbolic execution</td>
<td>Quantitative Bismulation</td>
<td>Malware detection</td>
</tr>
<tr>
<td>Provable security</td>
<td>Code contracts</td>
<td>SMT solvers</td>
<td>Code refactoring</td>
</tr>
<tr>
<td>Parsing</td>
<td>Type theory</td>
<td>Steganography</td>
<td>Tautology testers</td>
</tr>
</tbody>
</table>

Example: reasoning on computational structures

<table>
<thead>
<tr>
<th>WCET</th>
<th>Security protocol verification</th>
<th>Systems biology analysis</th>
<th>Operational semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axiomatic semantics</td>
<td>Dataflow analysis</td>
<td>Model checking</td>
<td>Abstraction refinement</td>
</tr>
<tr>
<td>Confidentiality analysis</td>
<td>Database query</td>
<td>Dependence analysis</td>
<td>Type inference</td>
</tr>
<tr>
<td>Program synthesis</td>
<td>Denotational semantics</td>
<td>CEGAR</td>
<td>Separation logic</td>
</tr>
<tr>
<td>Grammar analysis</td>
<td>Theories combination</td>
<td>Program transformation</td>
<td>Termination proof</td>
</tr>
<tr>
<td>Statistical model-checking</td>
<td>Code Interpolants</td>
<td>Abstract model checking</td>
<td>Shape analysis</td>
</tr>
<tr>
<td>Invariance proof</td>
<td>Symbolic execution</td>
<td>Quantitative Bismulation</td>
<td>Malware detection</td>
</tr>
<tr>
<td>Provable security</td>
<td>Code contracts</td>
<td>SMT solvers</td>
<td>Code refactoring</td>
</tr>
<tr>
<td>Parsing</td>
<td>Type theory</td>
<td>Steganography</td>
<td>Tautology testers</td>
</tr>
</tbody>
</table>
Example: reasoning on computational structures

Abstract interpretation

Informal examples of abstraction

Practical motivations

All computer scientists have experienced bugs

Ariane 5.01 failure (overflow)
Patriot failure (float rounding)
Mars orbiter loss (unit error)
Heartbleed (buffer overrun)

• Checking the presence of bugs by debugging is great
• Proving their absence by static analysis is even better!
• Undecidability and complexity is the challenge for automation
Abstractions of Dora Maar by Picasso

Pixelation

An old idea...

20,000 years old picture in a Spanish cave:

(the concrete is unknown)

Abstractions of a man / crowd

Height
Fingerprint
Eye color
DNA
Phone metadata

Individual heights
min, max
Numerical abstractions in Astrée

Properties and their Abstractions

Concrete properties

- A concrete property is represented by the set of elements which have that property:
  - universe (set of elements) $\mathcal{D}$ (e.g. a semantic domain)
  - properties of these elements: $P \in \wp(\mathcal{D})$
  - "x has property P" is $x \in P$
  - $(\wp(\mathcal{D}), \subseteq, \cup, \cap, \ldots)$ is a complete lattice for inclusion $\subseteq$ (i.e. logical implication)

Collecting semantics:
- partial traces

Intervals: $x \in [a, b]$

Simple congruences: $x \equiv a[b]$

Octagons:
- $\pm x \pm y \leq a$

Ellipses:
- $x^2 + by^2 - axy \leq d$

Exponentials:
- $-a^{bt} \leq y(t) \leq a^{bt}$

A very short introduction to abstract interpretation
Abstract properties

• Abstract properties: \( Q \in \mathcal{A} \)

• Abstract domain \( \mathcal{A} \): encodes a subset of the concrete properties (e.g. a program logic, type terms, linear algebra, etc)

• Poset: \( \langle \mathcal{A}, \subseteq, \sqcup, \sqcap, ... \rangle \)

• Partial order: \( \subseteq \) is abstract implication

Concretization

• Concretization \( \gamma \in \mathcal{A} \rightarrow \wp(\mathcal{A}) \)

• \( \gamma(Q) \) is the semantics (concrete meaning) of \( Q \)

• \( \gamma \) is increasing (so \( \subseteq \) abstracts \( \subseteq \))

• The concrete properties in \( \gamma(\mathcal{A}) \) are exactly representable in the abstract \( \mathcal{A} \), all others in \( \wp(\mathcal{A}) \setminus \gamma(\mathcal{A}) \) can only be approximated in \( \mathcal{A} \)

Best abstraction

• A concrete property \( P \in \wp(\mathcal{D}) \) has a best abstraction \( Q \in \mathcal{A} \) iff

  • it is sound (over-approximation):
    \[ P \subseteq \gamma(Q) \]

  • and more precise than any sound abstraction:
    \[ P \subseteq \gamma(Q') \implies Q \subseteq Q' \implies \gamma(Q) \subseteq \gamma(Q') \]

• The best abstraction is unique (by antisymmetry)

• Under-approximation is order-dual

Galois connection

• Any \( P \in \wp(\mathcal{D}) \) has a (unique) best abstraction \( \alpha(P) \) in \( \mathcal{A} \) if and only if

  \[ \forall P \in \wp(\mathcal{D}): \forall Q \in \mathcal{A}: \; \alpha(P) \subseteq Q \iff P \subseteq \gamma(Q) \]

  \( \Rightarrow \): over-approximation

  \( \Leftarrow \): best abstraction

written

\[ \langle \wp(\mathcal{D}), \subseteq \rangle \overset{\gamma}{\underset{\alpha}{\leftrightarrow}} \langle \mathcal{A}, \subseteq \rangle \]
Examples

- Needness/strictness analysis (80's)
  
  ![Diagram](true, false)

- Similar abstraction \((\gamma(T) \triangleq \{\text{true, false}\})\) for scalable hardware symbolic trajectory evaluation STE (90)

  Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

  Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

Properties of Galois connections

- \(\alpha\) preserves existing lubs (by order-duality, \(\gamma\) preserves existing glbs)
- One adjoint uniquely determine the other
- \(\alpha\) is surjective (iff \(\gamma\) injective iff \(\alpha \circ \gamma = 1\)), written
  \[
  \langle P, \leq \rangle \xrightarrow{\alpha} \langle Q, \subseteq \rangle
  \]
- The composition of Galois connections is a Galois connection
- \(\alpha(x)\) is the best over-approximation of \(x \in P:\)
  - \(x \leq \gamma(\alpha(x))\) over-approximation
  - \(x \leq \gamma(y) \implies \alpha(x) \subseteq y\) more precise than any other over-approximation

Example: Homomorphic abstraction \(\varphi(D) \rightarrow \varphi(A)\)

- \(h \in D \rightarrow A\)
  \[
  \alpha \triangleq \lambda x \cdot \{h(x) | x \in X\}
  \]
  \[
  \gamma \triangleq \lambda y \cdot \{x \in D | h(x) \in Y\}
  \]
  \[
  \implies \langle \varphi(D), \subseteq \rangle \xrightarrow{\gamma} \langle \varphi(A), \subseteq \rangle \quad (\text{iff } h \text{ onto})
  \]

- Example (*) rule of signs: \(A = \mathbb{Z}, B = \{-1, 0, 1\}, h(z) = z/|z|\)

- Counter-example (**) intervals (octagons, polyhedra, etc)

Equivalent mathematical structures

- Join morphism
- Meet morphism
- Upper closure
- Downset family
- Congruence
- Soundness relation
- Relation postimage

---

Example: Homomorphic abstraction \(\varphi(D) \rightarrow \varphi(A)\)

- \(h \in D \rightarrow A\)
  \[
  \alpha \triangleq \lambda x \cdot \{h(x) | x \in X\}
  \]
  \[
  \gamma \triangleq \lambda y \cdot \{x \in D | h(x) \in Y\}
  \]
  \[
  \implies \langle \varphi(D), \subseteq \rangle \xrightarrow{\gamma} \langle \varphi(A), \subseteq \rangle \quad (\text{iff } h \text{ onto})
  \]

- Example (*) rule of signs: \(A = \mathbb{Z}, B = \{-1, 0, 1\}, h(z) = z/|z|\)

- Counter-example (**) intervals (octagons, polyhedra, etc)

---

Example: Homomorphic abstraction \(\varphi(D) \rightarrow \varphi(A)\)

- \(h \in D \rightarrow A\)
  \[
  \alpha \triangleq \lambda x \cdot \{h(x) | x \in X\}
  \]
  \[
  \gamma \triangleq \lambda y \cdot \{x \in D | h(x) \in Y\}
  \]
  \[
  \implies \langle \varphi(D), \subseteq \rangle \xrightarrow{\gamma} \langle \varphi(A), \subseteq \rangle \quad (\text{iff } h \text{ onto})
  \]

- Example (*) rule of signs: \(A = \mathbb{Z}, B = \{-1, 0, 1\}, h(z) = z/|z|\)

- Counter-example (**) intervals (octagons, polyhedra, etc)
In absence of best abstraction?

- Best abstraction of a disk by a rectangular parallelogram (intervals)

- No best abstraction of a disk by a polyhedron (Euclid)

use only abstraction or concretization or widening (*)

Best abstract semantics

- If \( \langle \varrho(\mathcal{D}), \subseteq \rangle \xrightarrow{\gamma} \langle \mathcal{A}, \subseteq \rangle \) then the **best abstract semantics** is the abstraction of the collecting semantics

\[
\tilde{S}[P] \triangleq \alpha(S[P])
\]

- Proof:
  - It is **sound**: \( \tilde{S}[P] \triangleq \alpha(S[P]) \subseteq \tilde{S}[P] \implies \{S[P]\} \subseteq \gamma(\tilde{S}[P]) \implies S[P] \in \gamma(\tilde{S}[P]) \)
  - It is the **most precise**: \( S[P] \in \gamma(\tilde{S}[P]) \implies \{S[P]\} \subseteq \gamma(\tilde{S}[P]) \implies \tilde{S}[P] \triangleq \alpha(\{S[P]\}) \subseteq \tilde{S}[P] \)

Sound semantics abstraction

- program \( P \in \mathcal{L} \) programming language

- standard semantics \( \{S[P]\} \in \mathcal{D} \) semantic domain

- collecting semantics \( \{S[P]\} \in \varrho(\mathcal{D}) \) semantic property

- abstract semantics \( \tilde{S}[P] \in \mathcal{A} \) abstract domain

- concretization \( \gamma \in \mathcal{A} \rightarrow \mathcal{D} \)

- soundness \( \{S[P]\} \subseteq \gamma(\tilde{S}[P]) \)

i.e. \( \tilde{S}[P] \in \gamma(\tilde{S}[P]), \ P \ has \ abstract \ property \tilde{S}[P] \)

Calculational design of the abstract semantics

- The (standard hence collecting) semantics are defined by composition of **mathematical structures** (such as set unions, products, functions, fixpoints, etc)

- If you know **best abstractions of properties**, you also know **best abstractions of these mathematical structures**

- So, by composition, you also know the **best abstraction of the collecting semantics** --- calculational design of the abstract semantics

- Orthogonally, there are many styles of
  - **semantics** (traces, relations, transformers,…)
  - **induction** (transitional, structural, segmentation [POPL 2012])
  - **presentations** (fixpoints, equations, constraints, rules [CAV 1995])
Example: functional connector

- If \( g = \langle \mathcal{C}, \subseteq \rangle \xrightarrow{\gamma} \langle \mathcal{A}, \subseteq \rangle \) then

\[
g \mapsto g = \langle \mathcal{C} \xrightarrow{\gamma} \mathcal{C}, \subseteq \rangle \xrightarrow{\lambda F, \gamma \circ F} \langle \mathcal{A} \xrightarrow{\gamma} \mathcal{A}, \subseteq \rangle
\]

(\( \mapsto \) is a called a \textit{Galois connector}.)

Fixpoint abstraction

**Theorem 1** If \( \langle C, \sqsubseteq \rangle \xrightarrow{\gamma} \langle A, \sqsubseteq \rangle \) in cpos for infinite/transfinite chains, \( F \in C \mapsto C \) and \( G \in A \mapsto A \) are continuous/increasing then

\[
\begin{align*}
\alpha(\text{lfp} F) &= \text{lfp} \gamma G \iff \alpha \circ F = G \circ \alpha & \text{(commutation condition)} \\
G &= \alpha \circ F \circ \gamma \\
\alpha(\text{lfp} F) &\leq \text{lfp} \gamma G \iff \alpha \circ F \preceq G \circ \alpha & \text{(semi-commutation condition)}
\end{align*}
\]

[\text{Cousot and Cousot, 1979b, theorem 7.1.0.4(2–3)], see also [de Bakker et al., 1984, lemma 4.3], [Apt and Plotkin, 1986, fact 2.3], [Backhouse, 2000, theorem 95], etc.}

[\text{Cousot and Cousot, 1979b} \text{ Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979, 269-282}]

**Fixpoint abstraction**

- **Best abstraction** (completeness case)

\[
\text{if } \alpha \circ F = \bar{F} \circ \alpha \text{ then } \bar{F} = \alpha \circ F \circ \gamma \text{ and } \alpha(\text{lfp} F) = \text{lfp} \bar{F}
\]
e.g. semantics, proof methods, static analysis of finite state systems

- **Best approximation** (incompleteness case)

\[
\text{if } \bar{F} = \alpha \circ F \circ \gamma \text{ but } \alpha \circ F \sqsubseteq \bar{F} \circ \alpha \text{ then } \alpha(\text{lfp} F) \sqsubseteq \text{lfp} \bar{F}
\]
e.g. static analysis of infinite state systems

- idem for equations, constraints, rule-based deductive systems, etc

**Exact fixpoint abstraction**

Abstract domain

\[
\begin{align*}
\alpha \quad F &\quad F \quad F \quad F \quad F \\
\alpha \quad F &\quad F \quad F \quad F \quad F
\end{align*}
\]

Concrete domain

\[
\alpha \circ F = F^\sharp \circ \alpha \Rightarrow \alpha(\text{lfp} F) = \text{lfp} F^\sharp
\]
Approximate fixpoint abstraction

Abstract domain

Concrete domain

Approximation relation $\sqsubseteq$

$\text{lfp } F \sqsubseteq \gamma(\text{lfp } F^\#)$

Duality

- **Order duality**: join ($\cup$) or meet ($\cap$)
- **Inversion duality**: forward ($\rightarrow$) or backward ($\leftarrow = (\rightarrow)^{-1}$)
- **Fixpoint duality**: least ($\downarrow$) or greatest ($\uparrow$)

Why abstracting properties of semantics, not semantics?

1. **Abstract interpretation** = a non-standard semantics
   (computations on values in the standard semantics are replaced by computations on abstract values) $\implies$ extremely limited

2. **Abstract interpretation** = an abstraction of the standard semantics $\implies$ limited

3. **Abstract interpretation** = an abstraction of properties of the standard semantics $\implies$ more general
   
i.e. (1) is an abstraction of (2), (2) is an abstraction of (3)

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282
Example: trace semantics properties

- Domain of [in]finite traces on states: $\Pi$
- “Standard” trace semantics domain: $\mathcal{D} = \wp(\Pi)$
- “Standard” trace semantics $S[\mathcal{P}] \in \mathcal{D} = \wp(\Pi)$
- Domain of semantics properties is $\wp(\mathcal{D}) = \wp(\wp(\Pi))$
- Collecting semantics $C[\mathcal{P}] \triangleq \{ S[\mathcal{P}] \} \in \wp(\mathcal{D}) = \wp(\wp(\Pi))$

How to abstract the standard semantics?

- The join abstraction:
  $\langle \wp(\wp(\Pi)), \subseteq \rangle \xrightarrow{\gamma_U} \langle \wp(\Pi), \subseteq \rangle$
  $\alpha_U(X) \triangleq \bigcup X$
  $\gamma_U(Y) \triangleq \wp(Y)$
- Join abstraction of the collecting semantics:
  $\alpha_U(\mathcal{C}[\mathcal{P}]) \triangleq \bigcup \{ S[\mathcal{P}] \} \triangleq S[\mathcal{P}]$
  (i.e. the semantics is the join abstraction of its strongest property)

Loss of information

- “Always terminate with the same value, either 0 or 1”

<table>
<thead>
<tr>
<th>Always terminate with the same value, either 0 or 1</th>
</tr>
</thead>
</table>
| $P = \begin{array}{c}
\begin{array}{c}
\mathbb{0} \\
\mathbb{0} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{1} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\end{array}
\end{array}$ |
| $P \in \wp(\wp(\Pi))$ |

- Join abstraction:

  $\alpha_U(P) = \begin{array}{c}
\begin{array}{c}
\mathbb{0} \\
\mathbb{0} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{1} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\mathbb{\ldots} \\
\end{array}
\end{array}$

  $\alpha_U(P) \in \wp(\Pi)$

  “Always terminate, either with 0 or 1”

Limitations of the union abstraction

- Complete iff any property of the semantics $S[\mathcal{P}]$ is also valid for any subset $\gamma(S[\mathcal{P}]) = \wp(S[\mathcal{P}])$:
  - Examples: safety, liveness
  - Counter-example: security (e.g. authentication using a random cryptographic nonce)
Exact abstractions

- The concrete properties of the standard semantics $S[P]$ that you want to prove can always be proved in the abstract (which is simpler):

$$\forall Q \in S : S[P] \in \gamma(Q) \iff S[P] \subseteq Q$$

where

$$S[P] \triangleq \alpha \circ S[P] \circ \gamma$$

Example: Grammars

- Context-free grammar on alphabet $A = \text{Num} \cup \text{Var} \cup \{+,-,(,),\ldots\}$:

$$E ::= \text{Num} | \text{Var} | E + E | -E | (E)$$

- Chomsky-Schützenberger fixpoint semantics:

$$S[E] = \text{lfp} \subseteq \mathcal{F}[E]$$

$$\mathcal{F}[E]X \triangleq S[\text{Num}] \cup S[\text{Var}]$$

$$\cup \{e_1 + e_2 \mid e_1, e_2 \in X\}$$

$$\cup \{-e \mid e \in X\} \cup \{(e) \mid e \in X\}$$

Example 1 of exact abstraction: grammars

Example: Grammars (cont’d)

- **FIRST abstraction** of a language \( X \in A^* \):
  \[
  \alpha_F(X) \triangleq \{ \ell \mid \exists \sigma \in A^* : \ell \sigma \in X \} \cup \{ \epsilon \mid \epsilon \in X \}
  \]

- Galois connection:
  \[
  \langle \varphi(A^*) , \subseteq \rangle \xrightarrow{\gamma_F} \langle \varphi(A \cup \{ \epsilon \}) , \subseteq \rangle
  \]
  where
  \[
  \gamma_F(Y) \triangleq \{ \ell \sigma \mid \ell \in Y \land \sigma \in A^* \} \cup \{ \epsilon \mid \epsilon \in Y \}
  \]

**Machine-checkable calculational design**

\[
\begin{align*}
\alpha_F \circ \mathcal{F}[E] &= \lambda X \cdot \alpha_F(\mathcal{F}[E](X)) \\
&= \lambda X \cdot (\{ \ell \mid \exists \sigma \in A^* : \ell \sigma \in \mathcal{F}[E](X) \} \cup \{ \epsilon \mid \epsilon \in \mathcal{F}[E](X) \}) & \text{(def. \( \circ \))} \\
&= \lambda X \cdot (\{ \ell \mid \exists \sigma \in A^* : \ell \sigma \in \mathcal{F}[E](X) \}) & \text{(since \( \forall X : \epsilon \notin \mathcal{F}[E](X) \))} \\
&= \lambda X \cdot (\{ \ell \mid \exists \sigma \in A^* : \ell \sigma \in S[\text{Num}] \cup S[\text{Var}] \cup \{ e_1 + e_2 \mid e_1, e_2 \in X \} \cup \{ - \epsilon \mid \epsilon \in X \} \}) & \text{(def. \( \mathcal{F}[E] \))} \\
&= \lambda X \cdot (S[\text{Num}] \cup S[\text{Var}] \cup \{ \ell \mid \exists \sigma \in A^* : \ell \sigma \in X \} \cup \{ + \mid \epsilon \in X \} \cup \{ - \mid \epsilon \in X \} \}) & \text{(def. \( \mathcal{F}[E] \))} \\
&= \lambda X \cdot (S[\text{Num}] \cup S[\text{Var}] \cup (\alpha_F(X) \setminus \{ \epsilon \}) \cup \{ + \mid \epsilon \in \alpha_F(X) \} \cup \{ - \mid \epsilon \in \alpha_F(X) \}) & \text{(def. \( \alpha_F \) and \( \epsilon \in X \iff \epsilon \in \alpha_F(X) \))} \\
&= \lambda X \cdot (\mathcal{F}[E](\alpha_F(X)) & \text{(by defining \( \mathcal{F}[E] \))} \\
&= \mathcal{F}[E] \circ \alpha_F & \text{(def. \( \circ \))}
\end{align*}
\]

Example: Grammars (cont’d)

- **Commutation:**
  \[
  \alpha_F \circ \mathcal{F}[E] = \mathcal{F}[E] \circ \alpha_F
  \]
  where for \( E ::= \text{Num} \mid \text{Var} \mid E + E \mid -E \mid (E) \)
  \[
  \mathcal{F}[E]Y \triangleq S[\text{Num}] \cup S[\text{Var}] \cup (Y \setminus \{ \epsilon \}) \cup \{ + \mid \epsilon \in Y \} \cup \{ - \mid \epsilon \in Y \}
  \]

- **FIRST abstract semantics:**
  \[
  \mathcal{S}[E] \triangleq \alpha_F(S[E])
  \]
  \[
  \begin{align*}
  &= \alpha_F(\text{if}^= \mathcal{F}[E]) & \text{(Chomsky-Schützenberger)} \\
  &= \text{if}^= \mathcal{F}[E] & \text{(fixpoint abstraction th.)}
  \end{align*}
  \]

Algorithm

- Read the grammar \( G \), establish the system of equations \( Y = \mathcal{F}[G](Y) \), solve by chaotic iterations

- This is, up to [en]coding details, the classical algorithm:
  
  ```
  for each \( \alpha \in (T \cup \epsilon) \\
  \text{FIRST}(\alpha) \leftarrow \alpha \\
  \text{for each } A \in \text{NT} \\
  \text{FIRST}(A) \leftarrow 0 \\
  while \text{ (FIRST sets are still changing) } \\
  \text{for each } p \in P, \text{ where } p \text{ has the form } A \rightarrow \beta \\
  \text{if } \beta \text{ is } \beta_1 \beta_2 \ldots \beta_k, \text{ where } \beta_i \in T \cup \text{NT}, \text{ then} \\
  \text{FIRST}(A) \leftarrow \text{FIRST}(A) \cup \text{FIRST}(\beta_1) - \{ \epsilon \} \\
  i \leftarrow 1 \\
  while (\epsilon \in \text{FIRST}(A) \land i \leq k) \\
  \text{FIRST}(A) \leftarrow \text{FIRST}(A) \cup \text{FIRST}(\beta_{i+1}) - \{ \epsilon \} \\
  i \leftarrow i + 1 \\
  if i = k \text{ and } \epsilon \in \text{FIRST}(\beta_k) \\
  \text{then } \text{FIRST}(A) \leftarrow \text{FIRST}(A) \cup \{ \epsilon \}
  ```

Keith D. Cooper, Linda Torczon: Engineering a Compiler. Morgan Kaufmann 2004
Hierarchies of abstractions

Hierarchies of Grammar Semantics

Chomsky–Schützenberger terminal language

Example of proto-derivation tree

Example of proto-derivation

Hierarchies of Semantics for Resolution-based Languages

Comparison of abstractions

- \( \langle P, \leq \rangle \xrightarrow{\gamma_1} \langle Q, \sqsubseteq \rangle \)
  - is more precise than
  - \( \langle P, \leq \rangle \xrightarrow{\gamma_2} \langle R, \preceq \rangle \)

  iff \( \gamma_2(R) \subseteq \gamma_1(Q) \)

  (every abstraction in \( R \) is exactly expressible by \( Q \))

- We say that \( Q \) is a refinement of \( R \) and \( R \) that is a abstraction of \( Q \)

- A pre-order

Example II of exact abstraction: graphs

Finite paths

- Finite paths:
  \[ \Theta^+ \triangleq \{ \sigma_0 \xrightarrow{A_0} \sigma_1 \ldots \sigma_{n-1} \xrightarrow{A_{n-1}} \sigma_n \mid n \geq 0 \land \forall i \in [0, n] : \sigma_i \in \Sigma \land \forall i \in [0, n) : A_i \in \Lambda \} \]

- Paths between two vertices:
  \[ \Pi \in (\Sigma \times \Sigma) \mapsto \varphi(\Theta^+) \]
  \[ \Pi(\sigma, \sigma') \triangleq \{ \sigma_0 \xrightarrow{A_0} \sigma_1 \ldots \sigma_{n-1} \xrightarrow{A_{n-1}} \sigma_n \mid \sigma = \sigma_0 \land n \geq 0 \land \forall i \in [0, n-1] : \sigma_i \xrightarrow{A_i} \sigma_{i+1} \land \sigma_n = \sigma' \} \]

Transition system

- Transition system: \( \langle \Sigma, \Lambda, \rightarrow \rangle \)
  transition relation: \( \rightarrow \in \varphi(\Sigma \times \Lambda \times \Sigma) \)
  transitions/edges: \( \sigma \xrightarrow{\Lambda} \sigma' \)

- Example: non-negatively weighted graphs \( \Lambda \triangleq \mathbb{N} \)

Fixpoint characterization

- Pointwise fixpoint characterization:
  \[ \Pi = \text{fp}^\varsigma F \]
  \[ F \in ((\Sigma \times \Sigma) \mapsto \varphi(\Theta^+)) \mapsto ((\Sigma \times \Sigma) \mapsto \varphi(\Theta^+)) \]
  \[ F(X)(\sigma, \sigma') = \{ \sigma = \sigma' \Rightarrow \{ \sigma \} \cup \left\{ \sigma \xrightarrow{\Delta A} \sigma'' \pi \mid \sigma \xrightarrow{\Delta A} \sigma'' \land \sigma'' \pi \in X(\sigma'', \sigma') \} \right\} \]

(a path of \( n \) transitions is either a single vertex \( n = 0 \) or an edge followed by a path of \( n - 1 \) transitions)
Minimal path length abstraction

- Edges have non-negative lengths $A = \mathbb{N}$
- Abstraction:
  \[
  \alpha \in \Theta^+ \mapsto \mathbb{N}
  \]
  \[
  \alpha(\sigma) \triangleq 0
  \]
  \[
  \alpha(\sigma \xrightarrow{n} \sigma'\pi) \triangleq n + \alpha(\sigma'\pi)
  \]
  \[
  \alpha \in \varphi(\Theta^+) \mapsto \mathbb{N}^\infty
  \]
  \[
  \alpha(X) \triangleq \min \{\alpha(\pi) \mid \pi \in X\}
  \]
  where
  \[
  \min \emptyset = +\infty
  \]
  \[
  \mathbb{N}^\infty \triangleq \mathbb{N} \cup \{+\infty\}
  \]
  \[
  \langle \mathbb{N}^\infty, \geq, \min \rangle \text{ is a complete lattice}
  \]

Shortest distance

- Shortest distance $\Delta(\sigma, \sigma')$ between any two vertices
- $\Delta \in (\Sigma \times \Sigma) \mapsto \mathbb{N}^\infty$
- $\Delta \triangleq \hat{\alpha}(\Pi) = \hat{\alpha}(\text{lfp} \hat{\alpha} F)$

Galois connection

- $\langle \varphi(\Theta^+), \subseteq \rangle \xrightarrow{\gamma} \langle \mathbb{N}^\infty, \geq \rangle$
- Pointwise extension:
  \[
  \hat{\alpha} \in (\Sigma \times \Sigma \mapsto \varphi(\Theta^+)) \mapsto (\Sigma \times \Sigma \mapsto \mathbb{N}^\infty)
  \]
  \[
  \hat{\alpha}(X)(\sigma, \sigma') \triangleq \alpha(X(\sigma, \sigma'))
  \]
- Pointwise Galois connection:
  \[
  \langle (\Sigma \times \Sigma) \mapsto \varphi(\Theta^+), \subseteq \rangle \xrightarrow{\gamma} \langle (\Sigma \times \Sigma) \mapsto \mathbb{N}^\infty, \geq \rangle
  \]

Calculational design of the shortest distance algorithm

\[
\hat{\alpha} \circ F
\]
\[
= \lambda X \cdot \hat{\alpha}(F(X))
\]
\[
\triangleq \text{def. } \hat{\alpha} F
\]
\[
\lambda(\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(F(X))(\sigma, \sigma')
\]
\[
\triangleq \text{def. } \lambda x \cdot e
\]
\[
\lambda(\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(\lambda(\sigma, \sigma')) \cdot \{\sigma = \sigma' \ ? \ \{\sigma\} \ & \bigcup \{\sigma \xrightarrow{n} \sigma'' \pi \mid \sigma \xrightarrow{n} \sigma'' \land \sigma''\pi \in X(\sigma'', \sigma')\}\}
\]
\[
\triangleq \text{def. } F
\]
\[
\lambda(\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(\{\sigma = \sigma' \ ? \ \{\sigma\} \ & \bigcup \{\sigma \xrightarrow{n} \sigma'' \pi \mid \sigma \xrightarrow{n} \sigma'' \land \sigma''\pi \in X(\sigma'', \sigma')\}\})
\]
\[
\triangleq \text{def. } \hat{\alpha}(X)(\sigma, \sigma') \triangleq \alpha(\Delta(\sigma, \sigma'))
\]
\[
\lambda(\sigma, \sigma') \cdot \lambda X \cdot \{\sigma = \sigma' \ ? \ \{\sigma\} \ & \bigcup \{\sigma \xrightarrow{n} \sigma'' \pi \mid \sigma \xrightarrow{n} \sigma'' \land \sigma''\pi \in X(\sigma'', \sigma')\}\}
\]
\[
\triangleq \text{def. conditional } \{\ldots \ ? \ \ldots \ ? \ \ldots \}
\]
\[
\lambda(\sigma, \sigma') \cdot \lambda X \cdot \{\sigma = \sigma' \ ? \ \{\sigma\} \ & \bigcup \{\sigma \xrightarrow{n} \sigma'' \pi \mid \sigma \xrightarrow{n} \sigma'' \land \sigma''\pi \in X(\sigma'', \sigma')\}\}
\]
\[
\triangleq \text{def. join preservation in Galois C.}
\]
\[
\lambda(\sigma, \sigma') \cdot \lambda X \cdot \{\sigma = \sigma' \ ? \ \min \{\alpha(\pi) \mid \pi \in \{\sigma\}\} \ & \bigcup \{\sigma \xrightarrow{n} \sigma'' \pi \mid \sigma \xrightarrow{n} \sigma'' \land \sigma''\pi \in X(\sigma'', \sigma')\}\}
\]
\[
\triangleq \text{def. } \alpha(X) \triangleq \min \{\alpha(\pi) \mid \pi \in X\}
\]
Calcutational design of the shortest distance algorithm

\[
\lambda (\sigma, \sigma') \cdot \lambda X \cdot \{ \sigma = \sigma' \iff \min \{ \alpha(\sigma) \} \iff \min \min \{ \alpha(\sigma) \to \sigma'' \mid \sigma \to \sigma'' 
\}
\}
\]

\[
= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \{ \sigma = \sigma' \iff \min \{ \alpha(\sigma) \} \iff \min \{ \alpha(\sigma) \to \sigma'' \mid \sigma \to \sigma'' \mid \sigma'' \in X(\sigma, \sigma') \} \}
\]

\[
= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \{ \sigma = \sigma' \iff \min \{ \alpha(\sigma) \} \iff \min \{ \alpha(\sigma) \to \sigma'' \mid \sigma \to \sigma'' \mid \sigma'' \in X(\sigma, \sigma') \} \}
\]

\[
= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \{ \sigma = \sigma' \iff \min \{ \alpha(\sigma) \} \iff \min \{ \alpha(\sigma) \to \sigma'' \mid \sigma \to \sigma'' \mid \sigma'' \in X(\sigma, \sigma') \} \}
\]

\[
\text{by defining}
\]

\[
G(X)(\sigma, \sigma') = \{ \sigma = \sigma' \iff \min \{ n + X(\sigma', \sigma') \mid \sigma \to \sigma' \} \}
\]

\[
= \lambda X \cdot \lambda (\sigma, \sigma') \cdot \lambda X \cdot G(\sigma(\sigma'))
\]

\[
= G \circ X
\]

\[
\text{Shortest distance in fixpoint form}
\]

\[
\Delta = \alpha(1_{fp} \subseteq F)
\]

\[
= 1_{fp} \triangleright G
\]

\[
= \min_{\sigma \in \Sigma} (\lambda (\sigma, \sigma') \cdot + \infty)
\]

\[
\text{where the iterates are}
\]

\[
\bullet \quad G^0(X) = X
\]

\[
\bullet \quad G^{n+1} = G \circ G^n, \quad n \in \mathbb{N}
\]

\[
\text{Example III of exact abstractions: semantics}
\]

These equations define the shortest distance algorithm, which computes the minimum distance between states. The algorithm uses a fixpoint abstraction theorem to find the shortest path. The iterates are defined by the function \(G\), which updates the distance as \(G^n(X) = X\) and \(G^{n+1} = G \circ G^n\).
Trace semantics

Abstraction to denotational/natural semantics

Abstraction to small-steps operational semantics

Abstraction to reachability/invariance
Abstraction to Hoare logic

Initial states: $P$  
Intermediate states: $Q$  
Final states of the finite traces

Infinite traces

$\{P\}C\{Q\} \Leftrightarrow \{ \bullet \mid \bullet \in P \land \ldots \bullet \in [C]\} \subseteq Q$

Poset of semantics

Hoare logics

Weakest precondition semantics

Denotational semantics

Relational semantics

Trace semantics

Lattice of Semantics

Verification/static analysis by abstract interpretation

- Define the syntax of programs $P \in L$
- Define the concrete semantics of programs:
  - $\mathcal{D}[P]$  
  - $\forall P \in L: S[P] \in \mathcal{D}[P]$  
- Concrete/semantic properties: $\mathcal{D}(\mathcal{D}[P])$
- Collecting semantics: $\{S[P]\} \in \mathcal{D}(\mathcal{D}[P])$
  (the strongest property of the semantics, which implies all other semantic properties)
Verification/static analysis by abstract interpretation

- Define the abstraction:
  \[ \langle \wp(\wp[P]), \subseteq \rangle \xrightarrow{\alpha[P]} \langle \wp[P], \subseteq \rangle \]

- Calculate the abstract semantics:
  \[ S\#[P] = \alpha[P](S[P]) \] exact abstraction
  \[ S\#[P] \subseteq \alpha[P](S[P]) \] approximate abstraction

- Soundness (by construction):
  \[ \forall P \in L : \forall Q \in \wp[P] : S[P] \subseteq Q \implies S[P] \in \gamma[P](Q) \]

- Completeness (for exact abstractions only)
  \[ \forall P \in L : \forall Q \in \wp[P] : S[P] \in \gamma[P](Q) \implies S\#[P] \subseteq Q \]

- Methodology:
  - Structural induction on programs \( P \)
  - Compositional definition\(^(*)\) of \( \wp[P] \) and \( \alpha[P]/\gamma[P] \)
  - Fixpoint abstraction/approximation for recursion
  - Verification for fixpoints is the main problem:
    \[ \text{lfp} \subseteq F\#[P] \subseteq Q \]


Verification/static analysis by abstract interpretation

- Method: find \( I \in \wp[P] \) such that \( F\#[P]I \subseteq I \land I \subseteq Q \) (so that \( \text{lfp} \subseteq F\#[P] \subseteq Q \), by Tarski)

- Verification/deductive/proof methods:
  - ask the end-user for the inductive argument \( I \)

- Static analysis:
  1. compute \( I \) knowing \( F\#[P] \) and \( Q \)
  2. compute \( I \) knowing \( F\#[P] \) (and later given any \( Q \) check that \( I \subseteq Q \))
Approximate abstractions

• The **concrete properties** of the standard semantics $S[P]$ that you want to prove **may not always be provable** in the abstract:

  $\forall Q \in \mathcal{A}: S[P] \in \gamma(Q) \not\Rightarrow S[P] \subseteq Q$

where

$S[P] \triangleq \alpha \circ S[P] \circ \gamma$

Why abstraction may be approximate?

• Example

  $\{ x = y \land 0 \leq x \leq 10 \}$

  $x := x - y;$

  $\{ x = 0 \land 0 \leq y \leq 10 \}$

  Interval abstraction:

  $\{ x \in [0, 10] \land y \in [0, 10] \}$

  $x := x - y;$

  $\{ x \in [-10, 10] \land y \in [0, 10] \}$

  (but for constants, the interval abstraction can’t express equality)

Refinement: good news

• **Problem:** how to prove a valid abstract property $\alpha(\{ \text{lfp } F[P] \}) \subseteq Q$ when $\alpha \circ F \subseteq F^\# \circ \alpha$ but $\text{lfp } F^\# \not\subseteq Q$?

• It is **always** possible to refine $\langle \mathcal{A}, \subseteq \rangle$ into a most abstract more precise abstraction $\langle \mathcal{A}', \subseteq' \rangle$ such that $\langle \mathcal{A}, \subseteq \rangle \xrightarrow{\gamma} \langle \mathcal{A}', \subseteq' \rangle$ and $\alpha' \circ F = F' \circ \alpha$ with $\text{lfp } F'[P] \subseteq \alpha' \circ \gamma(Q)$

(thus proving $\text{lfp } F[P] \in \gamma'(Q)$ which implies $\text{lfp } F[P] \in \gamma(Q)$)

Refinement: bad news

• But, refinements of an abstraction can be intrinsically incomplete

• The only complete refinement of that abstraction for the collecting semantics is:
  • the identity (i.e. no abstraction at all)
  • In that case, the only complete refinement of the abstraction is to the collecting semantics and any other refinement is always imprecise

Example of intrinsic approximate refinement

• Consider the temporal specification language $\mathcal{L}$ (containing LTL, CTL, CTL*, and Kozen’s μ-calculus as fragments):

| $\varphi$ ::= | $\sigma_S$ | $S \in \varphi(S)$ | state predicate |
| $\pi_t$ | $t \in \varphi(S \times S)$ | transition predicate |
| $\oplus \varphi_1$ | next |
| $\varphi_1^\text{rev}$ | reversal |
| $\varphi_1 \lor \varphi_2$ | disjunction |
| $\neg \varphi_1$ | negation |
| $X$ | $X \in X$ | variable |
| $\mu X \cdot \varphi_1$ | least fixpoint |
| $\nu X \cdot \varphi_1$ | greatest fixpoint |
| $\forall \varphi_1 : \varphi_2$ | universal state closure |

Example of intrinsic approximate refinement

• Consider universal model-checking abstraction:

$$MC_M^V(\phi) = \sigma_M^V(\langle \phi \rangle) \in \varphi(\text{Traces}) ightarrow \varphi(\text{States})$$

$$= \{ s \in \text{States} \mid \forall \langle i, \sigma \rangle \in \text{Traces}_M . (\sigma_i = s) \Rightarrow \langle i, \sigma \rangle \in [\phi] \}$$

where $M$ is defined by a transition system

(and dually the existential model-checking abstraction)
Example of intrinsic approximate refinement

- The abstraction from a set of traces to a trace of sets is sound but incomplete, even for finite systems \(^{(\text{3})}\)

- Any refinement of this abstraction is incomplete (but to the infinite past/future trace semantics itself) \(^{(\text{28})}\)

\(^{(\text{3})}\) Patrick Cousot, Radhia Cousot: Temporal Abstract Interpretation. POPL 2000: 12-25


---

In general refinement does not terminate

- Example: filter invariant abstraction:
  
  2nd order filter:

  ![Image](image1.png)

  Unstable polyhedral abstraction:

  ![Image](image2.png)

  Counter-example guided refinement will indefinitely add missing points according to the execution trace:

  ![Image](image3.png)

  Stable ellipsoidal abstraction:

  ![Image](image4.png)

---

Intrinsic approximate refinement

\[
\lambda P \in \wp(D) \cdot D
\]

\[
\lambda P \in \wp(D) \cdot P
\]

Poset of abstractions

---

In general refinement does not terminate

- Narrowing is needed to stop infinite iterated automatic refinements:
  
  e.g. SLAM stops refinement after 20mn

- Intelligence is needed for refinement:
  
  e.g. human-driven refinement of Astrée

---


Finite versus infinite abstractions

[In]finite abstractions

• Given a program \( P \) and a program property \( Q \) which holds (i.e. \( \text{lfp } F[\![P]\!] \in Q \)) there exists a most abstract abstraction in a finite domain \( \mathcal{A}[\![P]\!] \) to prove it (*)

• Example:
  \[
  x=0;\text{ while } x<1 \text{ do } x++ \rightarrow \{\bot, [0,0], [0,1], [-\infty, \infty]\}
  
  x=0;\text{ while } x<2 \text{ do } x++ \rightarrow \{\bot, [0,0], [0,1], [0,2], [-\infty, \infty]\}
  
  \ldots
  
  x=0;\text{ while } x<n \text{ do } x++ \rightarrow \{\bot, [0,0], [0,1], [0,2], [0,3], \ldots, [0,n], [-\infty, \infty]\}
  
  \ldots
  
  (*) \text{Patrick Cousot: Partial Completeness of Abstract Fixpoint Checking. SARA 2000: 1-25}

• No such domain exists for infinitely many programs
  
  \[ \bigcup_{P \in L} \mathcal{A}[\![P]\!] \text{ is infinite} \]

  Example: \( \{\bot, [0,0], [0,1], [0,2], [0,3], \ldots, [0,\infty], [0,\infty+1], \ldots, [-\infty, \infty]\} \)

  \[ \lambda P \in L. \mathcal{A}[\![P]\!] \text{ is not computable (for undecidable properties)} \]

  \[ \Rightarrow \text{finite abstractions will fail infinitely often while infinite abstractions will succeed!} \]
Abstract Induction
(in non-Noetherian domains)

Convergence acceleration

Infinite iteration

Accelerated iteration with widening
(e.g. with a widening based on the derivative as in Newton-Raphson method)

Problem with infinite abstractions

- For non-Noetherian iterations, we need
  - finitary abstract induction, and
  - finitary passage to the limit

\[ X^0 = \bot, \ldots, X^{n+1} = \exists (X^0, \ldots, X^n, F(X^0), \ldots, F(X^n)), \ldots, \lim_{n \to \infty} X^n \]

\( \exists \) iteration converging

\begin{tabular}{|c|c|c|}
\hline
\( \exists \) & above the limit & below the limit \\
\hline
below the limit & widening \( \searrow \) & dual narrowing \( \bar{\triangle} \) \\
\hline
above the limit & narrowing \( \triangle \) & dual widening \( \bar{\triangledown} \) \\
\hline
\end{tabular}

[Semi-]dual abstract induction methods

\[
\begin{align*}
X \sqsubseteq F(X) & \quad \forall x \in \mathcal{L}_S, \bot \sqsubseteq V_S(j) x = x \sqcup V_S(j) \bot = x \\
[1, u_1] V_S(j) [l_1, u_2] & = \begin{cases} 
\text{if } 0 \leq l_1 < l_1 \text{ then } \neg \text{elsif } l_1 < l_1 \text{ then } \neg b \neg 1 \text{ else } l_1 \text{ fi,} \\
\text{if } u_1 < u_2 \leq 0 \text{ then } \neg \text{elsif } u_1 < u_2 \text{ then } b \text{ else } u_1 \text{ fi} 
\end{cases}
\end{align*}
\]

\( \bot \sqsubseteq X \sqsubseteq \top \)

\{ co-induction \}

\{ induction \}

(separate from termination conditions)

On widening/narrowing/and their duals

- Because the abstract domain is non-Noetherian, any widening/narrowing/duals can be strictly improved infinitely many times (i.e. no best widening)

E.g. widening with thresholds \([1]\)

\[
[a_1, b_1] \uparrow [a_2, b_2] = \\
[\text{if } a_1 < a_2 \text{ then } \neg \text{elsif } a_2 \text{ then } \neg \text{elsif } b_2 \text{ then } \neg \text{else } b_1 \text{ fi}]
\]

\[ [a_4, b_1] \downarrow [a_2, b_2] = \\
[\text{if } a_1 = \neg \text{then } a_2 \text{ else } \text{MIN } (a_1, a_2), \\
\text{if } b_1 = \neg \text{then } b_2 \text{ else } \text{MAX } (b_1, b_2)]
\]

- Any terminating widening is not increasing (in its 1st parameter)

- Any abstraction done with Galois connections can be done with widenings (i.e. a widening calculus)

\([1]\)  Patrick Cousot, Semantic foundations of program analysis, Ch. 10 of Program flow analysis: theory and practice, N. Jones & S. Muchnick (eds), Prentice Hall, 1981.
**Widening**

- \(\langle \mathcal{A}, \sqsubseteq \rangle\) poset
- \(\nabla \in \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}\)
- **Sound widening** (upper bound):
  \[
  \forall x, y \in \mathcal{A} : x \sqsubseteq x \nabla y \land y \sqsubseteq x \nabla y
  \]
- **Terminating widening**: for any \(\langle x^n \in \mathcal{A}, n \in \mathbb{N} \rangle\), the sequence \(y^0 \equiv x^0, \ldots, y^{n+1} \equiv y^n \nabla x^n, \ldots\) is **ultimately stationary** (\(\exists \varepsilon \in \mathbb{N}: \forall n \geq \varepsilon: y^n = y^\varepsilon\))

(Note: sound and terminating are independent properties)

**Example: (simple) widening for polyhedra**

- **Iterates**
- **Widening**

**Iteration with widening for static analysis**

- **Problem**: compute \(I\) such that \(\text{lfp} \sqsubseteq F \sqsubseteq I \sqsubseteq Q\)
- **Compute** \(I\) as the limit of the iterates:
  - \(X^0 \equiv \bot\),
  - \(X^{n+1} \equiv X^n\) when \(F(X^n) \sqsubseteq X^n\) so \(I = X^n\)
  - \(X^{n+1} \equiv (X^n \nabla F(X^n)) \triangle Q\) otherwise
- **I** can be improved by an iteration with narrowing \(\triangle\)
- **Check that** \(F(I) \sqsubseteq Q\)

**Dual narrowing**

- \(\langle \mathcal{A}, \sqsubseteq \rangle\) poset
- \(\sim \in \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}\)
- **Sound dual narrowing** (interpolation):
  \[
  \forall x, y \in \mathcal{A} : x \sqsubseteq y \implies x \sqsubseteq x \sim y \sqsubseteq y
  \]
- **Terminating dual narrowing**: for any \(\langle x^n \in \mathcal{A}, n \in \mathbb{N} \rangle\), the sequence \(y^0 \equiv x^0, \ldots, y^{n+1} \equiv y^n \sim x^n, \ldots\) is **ultimately stationary** (\(\exists \varepsilon \in \mathbb{N}: \forall n \geq \varepsilon: y^n = y^\varepsilon\))

(Note: sound and terminating are independent properties)
Iteration with dual narrowing for static checking

- Problem: find \( I \) such that \( \text{lfp} F \subseteq I \subseteq Q \)
- Compute \( I \) as the limit of the iterates:
  - \( X^0 \triangleq \bot \)
  - \( X^{n+1} \triangleq X^n \) \( \cup \) when \( F(X^n) \subseteq X^n \) so \( I = X^n \)
  - \( X^{n+1} \triangleq F(X^n) \triangleq Q \), otherwise
- Check that \( F(I) \subseteq Q \)
- Example: First-order logic + Craig interpolation (with some choice of one of the solutions, control of combinatorial explosion, and convergence enforcement)

Example of domain-specific abstraction: ellipses

```c
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;
    }
}
```
Code Contract Static Checker (cccheck)

- Available within MS Visual Studio

A screenshot from Clousot/cccheck on the classic binary search.
- The screenshot shows from left to right and top to bottom
  1. C# code + CodeContracts with a buggy BinarySearch
  2. cccheck integration in VS (right pane with all the options integrated in the VS project system)
  3. cccheck messages in the VS error list
- The features of cccheck that it shows are:
  1. Basic abstract interpretation:
     a. the loop invariant to prove the array access correct and that the arithmetic operation may
        overflow is inferred fully automatically
     b. different from deductive methods as e.g. ESC/Java or Boogie where the loop invariant must
        be provided by the end-user
  2. Inference of necessary preconditions:
     a. Clousot finds that array may be null (message 3)
     b. Clousot suggests and propagates a necessary precondition invariant (message 1)
  3. Array analysis (+ disjunctive reasoning):
     a. to prove the postcondition should infer property of the content of the array
     b. please note that the postcondition is true even if there is no precondition requiring the
        array to be sorted.
  4. Verified code repairs:
     a. from the inferred loop invariant does not follow that index computation does not
        overflow
     b. suggest a code fix for it (message 2)

Abstract interpretation

- Intellectual tool (not to be confused with its specific application to iterative static analysis with \( \nabla \) & \( \triangle \))
- No cathedral would have been built without plumb-line and square, certainly not enough for skyscrapers:
  Powerful tools are needed for progress and applicability of formal methods
Abstract interpretation

• See further developments in the proceedings with many (but unfortunately not all) references:

1. Abstraction
2. Scope
3. Static analysis
4. Acceleration of loop invariant
5. Semantics
6. Preservation of semantics
7. Combinations of semantics
8. Proof methods, verification and inference
9. Abstraction and under-approximation
10. Abstract domains
11. Refinement of abstract domains
12. Combinations of abstract domains
13. Equational design of abstract domains
14. Galois connections for best abstraction
15. In absence of best abstraction
16. Equational design of abstract domains
17. Syntactic abstractions
18. Abstraction of syntax
19. Temporal abstraction
20. Languages
21. Control-flow analysis
22. Parallelism
23. Types
24. Binary abstraction and hardware analysis
25. Numerical abstractions
26. Symbolic abstractions
27. Simulations
28. Probabilistic abstractions
29. Program transformation
30. Termination
31. Modularity
32. Generalist versus domain-aware static analyzers
33. Industrial applications
34. Security
35. Unexpected applications
36. Conclusion

Varieties of researchers in formal methods:

(i) explicitly use abstract interpretation, and are happy to extend its scope and broaden its applicability
(ii) implicitly use abstract interpretation, and hide it
(iii) pretend to use abstract interpretation, but misuse it
(iv) don’t know that they use abstract interpretation, but would benefit from it

Never too late to upgrade

The End

Thank You