Verification of Safety-Critical Control-Command Software by Abstract Interpretation

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

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

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

Talk Outline

- Deficiencies of formal methods (2 mn) .................. 4
- A few elements of abstract interpretation
  (20 mn) ........................................... 8
- Applications of abstract interpretation (2 mn) ....... 32
- Application to the verification of embedded,
  real-time, synchronous, safety super-critical
  control-command software (10 mn) ..................... 36
- Examples of abstractions (20 mn) ...................... 52
- Conclusion (1 mn) .................................. 68

Deficiencies of Formal Methods

Automated Verification of Infinite-State Systems

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

- **Manual** (e.g. require end-users to provide manually a simple-enough model of the complex system), and/or
- **User-unfriendly** (e.g. require complex interactions with end-users), and/or
- **Trivial** (e.g. consider immediate essentially syntactic program properties) and/or
- **Incorrect/unsound** (e.g. do not explore the complete space of executions and so may forget about potential problems at run-time), and/or
- **Inefficient** (some may not terminate at all but by exhaustion of time/memory resources), and/or
- **Imprecise** (leading to too many false alarms that is spurious warnings on potential problems that can never occur at run-time).

Can we do better?

A Few Elements of Abstract Interpretation

Reference


A Model of Computer Programs

- **Syntax** : a well-founded set of programs $\langle \mathcal{P}, \prec \rangle$ where $\prec$ is the “strict immediate subcomponent” relation;
- **Semantics of $P \in \mathcal{P}$** :
  - **Semantic domain** : a complete lattice/cpo $\langle \mathcal{D}[P], \subseteq, \bot, \sqcup \rangle$
  - **Compositional Fixpoint Semantics** :
    $$ S[P] \overset{\text{def.}}{=} \text{lfp}_{\sqcup} \mathcal{F}[P] \left( \prod_{P' < P} S[P'] \right) $$
    \text{lfp}_{\sqcup} f \text{ is the limit of } X^0 = \bot, \ X^{\delta+1} = f(X^\delta), \ X^\lambda = \sqcup_{\beta < \lambda} X^\beta, \ \lambda \text{ limit ordinal, if any. Existence e.g. monotony (by Tarski constructive [PACJM ’79])}. $
Example: Syntax of Programs

\[ \begin{align*}
X & \quad \text{variables } X \in X \\
T & \quad \text{types } T \in T \\
E & \quad \text{arithmetic expressions } B \in E \\
B & \quad \text{boolean expressions } B \in B \\
D & ::= T \, X; \\
\mid & D' \quad \text{declarations } D \in D, \ \text{vars}(D) = \{X\} \\
C & ::= X = E; \\
\mid & \quad \text{commands } C \in C \quad (E < C) \\
\mid & \quad \text{while } B \, C' \\
\mid & \quad \text{if } B \, C' \\
\mid & \quad \text{if } B \, C' \text{ else } C'' \\
\mid & \{ C_1 \ldots C_n \}, (n \geq 0) \\
P & ::= D \, C \\
\end{align*} \]

Example: Concrete Semantic Domain of Programs

Reachability properties:

\[ \begin{align*}
\Sigma[D] & \quad \text{states } \rho \\
\Sigma[T \, X; \, D] & \quad (\rho(X) \text{ is the value of } X) \\
\Sigma[T \, X; \, D] & \quad (\{X\} \mapsto T) \cup \Sigma[D] \\
D[P] & \quad \rho(\Sigma[P]) \\
\subseteq & \quad \subseteq \\
\bot & \quad \emptyset \\
\cup & \quad \cup
\end{align*} \]

Example: Concrete Semantics of Programs (Reachability)

\[ \begin{align*}
S[X = E; \, R] & \equiv \{ \rho[X \leftarrow E[E]\rho] \mid \rho \in R \cap \text{dom}(E) \} \\
\rho[X \leftarrow v](X) & \equiv v, \quad \rho[X \leftarrow v](Y) \equiv \rho(Y) \\
S[\text{if } B \, C' \, R] & \equiv S[C'](B[B]R) \cup B[\neg B]R \\
B[B]R & \equiv \{ \rho \in R \cap \text{dom}(B) \mid B \text{ holds in } \rho \} \\
S[\text{while } B \, C' \, R] & \equiv \text{let } W = \lambda \rho. R \cup S[C'](B[B]X) \text{ in } (B[\neg B]W) \\
S[\{\} \, R] & \equiv R \\
S[\{C_1 \ldots C_n\}] & \equiv S[C_n] \circ \ldots \circ S[C_1] \quad n > 0 \\
S[D \, C \, R] & \equiv S[C](\Sigma[D]) \quad \text{(uninitialized variables)}
\end{align*} \]

Not computable (undecidability).

Abstraction

A reasoning/computation which is restricted in that:

- only some properties can be used;
- the properties that can be used are called “abstract”;
- so, the (other concrete) properties must be approximated by the abstract ones;
Abstract Properties

- Abstract Properties: a set $\mathcal{A} \subseteq \wp(\Sigma)$ 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 $\overline{P} \in \mathcal{A}$:

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

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

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

(Otherwise see [JLC’92].)

Reference


Moore Family

- This hypothesis that any concrete property $P \in \wp(\Sigma)$ has a best abstraction $\overline{P} \in \mathcal{A}$ implies that: $\overline{A}$ is a Moore family
  i.e. it is closed under intersection $\cap$:

$$\forall S \subseteq \overline{A}: \cap S \in \overline{A}$$

- In particular $\cap 0 = \Sigma \in \overline{A}$ is “I don’t know”.

Example of Moore Family-Based Abstraction
Closure Operator Induced by an Abstraction

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

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

is a closure operator:
- extensive,
- idempotent,
- isotone/monotonic;

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

Example of Closure Operator-Based Abstraction

The Lattice of Abstract Interpretations

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

$$\langle D[P], \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$$

is a complete lattice

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

- The meet of abstractions called the reduced product

$$( \bigwedge_{i \in \Delta} \rho_i \text{ is that most abstract abstraction more precise than all } \rho_i, i \in \Delta )$$

Galois Connection Between Concrete and Abstract Properties

- For closure operators $\rho$, we have:

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

written:

$$\langle \varphi(\Sigma), \subseteq \rangle \overset{1}{\overleftarrow{\rho}} \langle \rho(\varphi(\Sigma)), \subseteq \rangle$$

where $1$ is the identity and:

$$\langle \varphi(\Sigma), \subseteq \rangle \overset{\gamma}{\overleftarrow{\alpha}} \langle D, \sqsubseteq \rangle$$

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

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

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

Abstract domain

Function Abstraction

\[ F^\| = \alpha \circ F \circ \gamma \]

i.e. \[ F^\| = \rho \circ F \]

Concrete domain

Approximate Fixpoint Abstraction

\[ F \circ \gamma \sqsubseteq \gamma \circ F^\| \Rightarrow \text{lfp } F \sqsubseteq \gamma(\text{lfp } F^\|) \]

Example: abstract semantic domain of programs

\[ \langle \mathcal{D}^\# [P], \sqsubseteq, \bot, \sqcup \rangle \]

such that:

\[ \langle \mathcal{D}, \sqsubseteq \rangle \xleftarrow{\gamma} \langle \mathcal{D}^\# [P], \sqsubseteq \rangle \]

hence \[ \langle \mathcal{D}^\# [P], \sqsubseteq, \bot, \sqcup \rangle \] is a complete lattice such that \[ \bot = \alpha(0) \] and \[ \sqcup X = \alpha(\sqcup \gamma(X)) \]
Example: abstract semantics of programs (reachability)

\[
S^J [X = E \searrow R] \equiv \alpha (\{ \rho | X \leftarrow E \rho \mid \rho \in \gamma (R) \cap \text{dom}(E) \})
\]
\[
S^J [\text{if } B \text{ then } C' \text{ else } C''] \equiv S^J [C'] (B^J [B] R) \cup B^J [\neg B] R
\]
\[
B^J [B] R \equiv \alpha (\{ \rho | \rho \in \gamma (R) \cap \text{dom}(B) \mid B \text{ holds in } \rho \})
\]
\[
S^J [\text{while } B \text{ do } C'] \equiv \text{let } W = \text{lfp} \lambda \chi . R \cup S^J [C'] (B^J [B] \chi) \text{ in } (B^J [\neg B] W)
\]

\[
S^J [\{\}] \equiv R
\]
\[
S^J [\{ C_1 \ldots C_n \}] \equiv S^J [C_n] \circ \ldots \circ S^J [C_1], \quad n > 0
\]
\[
S^J [D C] \equiv S^J [C] (\top) \quad \text{(uninitialized variables)}
\]

--- 25 ---

**Convergence Acceleration with Widening**

Widening Operator

A widening operator \( \nabla \in \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L} \) is such that:

- **Correctness:**
  \[ \forall x, y \in \mathcal{L} : \gamma (x) \subseteq \gamma (x \nabla y) \]
  \[ \forall x, y \in \mathcal{L} : \gamma (y) \subseteq \gamma (x \nabla y) \]

- **Convergence:**
  - for all increasing chains \( x_0 \subseteq x_1 \subseteq \ldots \), the increasing chain defined by \( y_0 = x_0, \ldots, y_{i+1} = y_i \nabla x_{i+1}, \ldots \) is not strictly increasing.

--- 27 ---

**Fixpoint Approximation with Widening**

**Convergence Theorem:**

The upward iteration sequence with widening:

- \( X^0 = \bot \) (infimum)
- \( X^{i+1} = X^i \) if \( F^J (X^i) \subseteq X^i \)
- \( X^{i+1} = X^i \nabla F^J (X^i) \) otherwise

is ultimately stationary and its limit \( A \) is a sound upper approximation of \( \text{lfp} \subseteq F^J \):

\[ \text{lfp} \subseteq F^J \subseteq A \]
Example: Abstract Semantics with Convergence Acceleration

\[ S^H[X = E] R \triangleq \alpha(\{\rho | X \gets E[\rho] \mid \rho \in \gamma(R) \cap \text{dom}(E)\}) \]

\[ S^H[\text{if } B \ B' \text{ else } C'] R \triangleq S^H[C'](B^R \cup B^\neg B) \]

\[ S^H[\text{while } B \ B' \text{ else } C''] R \triangleq S^H[C'](B^R \cup S^H[C'](B^\neg B \cdot X)) \]

\[ S^H[\emptyset] R \triangleq R \]

\[ S^H[\{C_1 \ldots C_n\}] R \triangleq S^H[C_n] \ldots \circ S^H[C_1] \]

\[ S^H[D \ C] R \triangleq S^H[C](\top) \quad (\text{uninitialized variables}) \]

Soundness Corollary: any abstract safety proof is valid in the concrete in that:

\[ S^H[P] \subseteq Q \implies S[P] \subseteq \gamma(Q) \]

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

Extrapolation by Widening is Essentially Not Monotone

Proof by contradiction:
- Let \( \triangledown \) be a widening operator
- Define \( x \triangledown y = \text{if } y \subseteq x \text{ then } x \text{ else } x \triangledown y \)
- Assume \( x \supseteq y = F(x) \) (during iteration)

\[ x \triangledown y = x \triangledown y \supseteq y \]

\[ y \triangledown y = y \]

\( x \triangledown y = y, \) by antisymmetry!

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

\[ 1 \text{ Note: } F^H \text{ not monotonic!} \]

Soundness Theorem

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

Applications of Abstract Interpretation

Reference

Applications of Abstract Interpretation

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

Applications of Abstract Interpretation (Cont’d)

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

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

---

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

Reference


---

**Static Program Analysis**

- **Program**
- **Specification**

  - **System of fixpoint equations/constraints**
  - **(Approximate) solution**

- **Generator**
- **Solver**
- **Diagnoser**

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

www.astree.ens.fr

- C programs:
  - structured C programs;
  - no dynamic memory allocation;
  - no recursion.

- Application Domain: safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems.

Abstract Semantics

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

Concrete Operational Semantics

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

Implicit Specification: Absence of Runtime Errors

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

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

- C program, automatically generated from a proprietary high-level specification

- A340: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays.

**The Class of Considered Periodic Synchronous Programs**

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

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

**Characteristics of the ASTRÉE Analyzer**

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

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

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

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

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

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

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

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

---

Characteristics of the ASTRÉE Analyzer (Cont’d)

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

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

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

---

Characteristics of the ASTRÉE Analyzer (Cont’d)

**Modular:** an analyzer instance is built by selection of OCAML modules from a collection each implementing an abstract domain

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

---

Example of Analysis Session

---

Benchmarks for the Primary Flight Control Software of the Airbus A340

- **Comparative results** (commercial software):
  - 4,200 (false?) alarms,
  - 3.5 days;

- **Our results**:
  - 0 alarm,
  - 1h20 on 2.8 GHz PC,
  - 300 Megabytes
  → A world première!
Examples of Abstractions

General-Purpose Abstract Domains: Intervals and Octagons

Intervals:
- $1 \leq x \leq 9$
- $1 \leq y \leq 20$

Octagons [4]:
- $1 \leq x \leq 9$
- $x + y \leq 77$
- $1 \leq y \leq 20$
- $x - y \leq 04$

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

Reference

Floating-Point Computations

• Code Sample:

```c
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%.16f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```c
/* double-error.c */
int main () {
  double x; float y, z, r;
  x = ldexp(1.,50)+ldexp(1.,26); /* x = 1125899973951488.0; */
  y = x + 1;
  z = x - 1;
  r = y - z;
  printf("%.16f\n", r);
}
% gcc double-error.c
% ./a.out
134217728.000000
```

• Interval analysis: if $x \in [a, b]$ and $y \in [c, d]$ then $x - y \in [a - c, b - d]$ so if $x \in [0, 100]$ then $x - x \in [-100, 100]!!$

• The symbolic abstract domain propagates the symbolic values of variables and performs simplifications;

• Must maintain the maximal possible rounding error for float computations (overestimated with intervals);

Symbolic abstract domain

© P. Cousot
13 May 2004
© P. Cousot
Clock Abstract Domain for Counters

- **Code Sample:**
  ```c
  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 \( \text{clock} \) in the concrete user semantics to track elapsed clock ticks.
  - For each variable \( X \), we abstract three intervals: \( X \), \( X+\text{clock} \), and \( X-\text{clock} \).
  - If \( X+\text{clock} \) or \( X-\text{clock} \) is bounded, so is \( X \).

---

Boolean Relations for Boolean Control

- **Code Sample:**
  ```c
  /* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ... 
    B = (X == 0);
    ... 
    if (!B) {
      Y = 1 / X;
    } 
    ... 
  }
}
```  
  The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leafs

---

Control Partitionning for Case Analysis

- **Code Sample:**
  ```c
  /* trace_partitionning.c */
void main() {
  float t[5] = {-10.0, -10.0, 0.0, 10.0, 10.0};
  float c[4] = {0.0, 2.0, 2.0, 0.0};
  float d[4] = {-20.0, -20.0, 0.0, 20.0};
  float x, r;
  int i = 0;
  ... found invariant -100 <= x <= 100 ...
  while ((i < 3) && (x >= t[i+1])) {
    i = i + 1;
    r = (x - t[i]) * c[i] + d[i];
  }
}
```  
  Delaying abstract unions in tests and loops is more precise for non-distributive abstract domains (and much less expensive than disjunctive completion).

---

2\(^{nd}\) Order Digital Filter:

- **Ellipsoid Abstract Domain for Filters**
  * Computes \( X_n = \{ \alpha X_{n-1} + \beta X_{n-2} + Y_n \} \)
  * 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.

---

Reference

(Automatic) Parameterization

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

The main loop invariant

A textual file over 4.5 Mb with

- 6,900 boolean interval assertions ($x \in [0;1]$)
- 9,600 interval assertions ($x \in [a;b]$)
- 25,400 clock assertions ($x+\text{clk} \in [a;b] \land x-\text{clk} \in [a;b]$)
- 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.

Why finite abstractions will not do?

Theoretical reasons on finite abstraction:

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

__Reference__


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

Theoretical reasons on abstraction refinement:

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

__Reference__

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

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

Reference

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

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

Reference

Example [11]

typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;

void filter () {
    static float E[2], S[2];
    if (INIT) { S[0] = X; P = X; E[0] = X; }
    else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4)) + (S[0] * 1.5)) - (S[1] * 0.7)); }
    E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
    /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}

void main () { X = 0.2 * X + 5; INIT = TRUE;
    while (1) {
        X = 0.9 * X + 35; /* simulated filter input */
        filter (); INIT = FALSE;
    }
}

Possible origins of imprecision and how to fix it

In case of false alarm, the imprecision can come from:
• Abstract transformers (not best possible) —> improve algorithm;
• Automatized parametrization (e.g. variable packing) —> improve pattern-matched program schemata;
• Iteration strategy for fixpoints —> fix widening 2;
• Inexpressivity i.e. indispensable local inductive invariant are inexpressible in the abstract —> add a new abstract domain to the reduced product (e.g. filters).

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

Reference
Conclusion

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

Reference


The Future

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

The Future (Cont’nd)

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


