1. The Endless “Software Failure” Problem

Example 1: Overflow

All Computer Scientists Have Experienced Bugs

Ariane 5.01
Patriot
Mars orbiter

Mars Global Surveyor

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

Embedded Systems Week, International Conference on Embedded Software — Salzburg, Austria
September 30th, 2007
Modular integer arithmetics...

- Today, computers avoid integer overflows thanks to modular arithmetic
- Example: integer 2’s complement encoding on 8 bits

Modular arithmetics is not very intuitive (cont’d)

In C:
% cat -n modulo.c.c
1 #include <stdio.h>
2 int main () {
3 int x,y;
4 x = -2147483647 / -1;
5 y = ((-x) -1) / -1;
6 return 0;
7 }
8
% gcc modulo.c.c
% ./a.out
x = 2147483647, y = -2147483648

Static Analysis with ASTRÉE

% cat -n overflow.c
1 void main () {
2 double x,y;
3 x = 1.0e+256 * 1.0e+256;
4 y = 1.0e+256 * -1.0e+256;
5 __ASTREE_log_vars((x,y));
6 }
7
% gcc overflow.c
% ./a.out
x = inf, y = -inf

% astree -exec-fn main -unroll 0 overflow.c
% egrep -A 1 "WARN"
overflow.c:3.4-23::[call#main1::]: WARN: double arithmetic range
{1.79769e+308, inf] not included in [-1.79769e+308, 1.79769e+308]
overflow.c:4.4-24::[call#main1::]: WARN: double arithmetic range
[-inf, -1.79769e+308] not included in [-1.79769e+308, 1.79769e+308]

ASTRÉE signals the overflow and goes on with an unknown value.

Float Arithmetics does Overflow

In C:
% cat -n overflow.c
1 void main () {
2 double x,y;
3 x = 1.0e+256 * 1.0e+256;
4 y = 1.0e+256 * -1.0e+256;
5 __ASTREE_log_vars((x,y));
6 }
7
% gcc overflow.c
% ./a.out
x = inf, y = -inf

% astree -exec-fn main
% grep "WARN"
overflow.c:3.4-23::[call#main1::]: WARN: double arithmetic range
[1.79769e+308, inf] not included in [-1.79769e+308, 1.79769e+308]
overflow.c:4.4-24::[call#main1::]: WARN: double arithmetic range
[-inf, -1.79769e+308] not included in [-1.79769e+308, 1.79769e+308]
The Ariane 5.01 maiden flight
– June 4th, 1996 was the maiden flight of Ariane 5

A 16 bit piece of code of Ariane 4 had been reused within the new 32 bit code for Ariane 5. This caused an uncaught overflow, making the launcher uncontrollable.

Example 2: Rounding

Rounding
– Computations returning reals that are not floats, must be rounded
– Most mathematical identities on $\mathbb{R}$ are no longer valid with floats
– Rounding errors may either compensate or accumulate in long computations
– Computations converging in the reals may diverge with floats (and ultimately overflow)
Example of rounding error

/* float-error.c */
int main () {
    float x, y, z, r;
    x = 1.0000000019e+38;
y = x + 1.0e21;
z = x - 1.0e21;
r = y - z;
    printf("%.9f\n", r);
} % gcc float-error.c % ./a.out

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

/* double-error.c */
int main () {
    double x, y, z, r;
    /* x = ldexp(1,.50)+ldexp(1,.26); */
x = 1125899973951488.0;
y = x + 1;
z = x - 1;
r = y - z;
    printf("%.9f\n", r);
} % gcc double-error.c % ./a.out

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

Explanation of the huge rounding error

(1) Floats
Reals
Rounding
x-ar \rightarrow x-ar+2^1

(2) Doubles
Reals
Floats
Rounding
x-ar = 2^6 \rightarrow x-ar+2^1

Static analysis with ASTRÉE

% cat -n double-error.c
2 int main () {
3 double x, float y, z, r;
4 /* x = ldexp(1,.50)+ldexp(1,.26); */
5 x = 1125899973951488.0;
6 y = x + 1;
7 z = x - 1;
8 r = y - z;
9 __ASTREE_log_vars((r));
10 }
% gcc double-error.c % ./a.out
134217728.000000

% astree -exec-fn main -print-float-digits 10 double-error.c |& grep "r in ">
direct = <float-interval: r in [-134217728, 134217728] >

\[ ASTRÉE \] makes a worst-case assumption on the rounding (+\infty, -\infty, 0, nearest) hence the possibility to get -134217728.
Example of accumulation of small rounding errors

```c
#include <stdio.h>
int main () {
    int i; double x; x = 0.0;
    for (i=1; i<=1000000000; i++) {
        x = x + 1.0/10.0;
    }
    printf("x = %f\n", x);
}
```

```
$ cat -n rounding-c.c
1 #include <stdio.h>
2 int main () {
3     int i; double x; x = 0.0;
4     for (i=1; i<=1000000000; i++) {
5         x = x + 1.0/10.0;
6     }
7     printf("x = %f\n", x);
8 }
```

```c
$ gcc rounding-c.c
$ ./a.out
x = 99999998.745418
```

since \((0.1)_{10} = (0.0001100110011001100\ldots)_{2}\)

The Patriot missile failure

- “On February 25th, 1991, a Patriot missile ... failed to track and intercept an incoming Scud (1).”
- The software failure was due to accumulated rounding error (2)

(1) This Scud subsequently hit an Army barracks, killing 28 Americans.
(2) “Time is kept continuously by the system’s internal clock in tenths of seconds”
- “The system had been in operation for over 100 consecutive hours”
- “Because the system had been on so long, the resulting inaccuracy in the time calculation caused the range gate to shift so much that the system could not track the incoming Scud”

Static analysis with ASTRÉE

```c
int main () {
    double x; x = 0.0;
    while (1) {
        x = x + 1.0/10.0;
        __ASTREE_log_vars((x));
        __ASTREE_wait_for_clock();
    }
}
```

```c
$ cat -n rounding.c
1 int main () {
2     double x; x = 0.0;
3     while (1) {
4         x = x + 1.0/10.0;
5         __ASTREE_log_vars((x));
6         __ASTREE_wait_for_clock();
7     }
8 }
```

```c
$ gcc rounding.c
$ __ASTREE_max_clock((1000000000));
$ astree -exec-fn main -config-sem rounding.config -unroll 0 rounding.c\n|& egrep "(x in)|(|x|)|(WARN)" | tail -2
direct = <float-interval: x in [0.1, 200000000.938]>
|x| <= 1.0*(((0.0 + 0.1/(1.-1)))*(1.)~clock - 0.1/(1.-1)) + 0.1
    <= 200000000.938
```

Other Examples
The NASA’s Climate Orbiter Loss on September 23, 1999

- A **metric confusion error** led to the loss of NASA’s $125 million, Lockheed Martin built Mars Climate Orbiter on September 23, 1999.

- “People sometimes make errors,” said Edward Weiler, NASA’s Associate Administrator for Space Science in a written statement. “The problem here was not the error, it was the failure of NASA’s systems engineering, and the checks and balances in our processes to detect the error. That’s why we lost the spacecraft.”

---

Static Analysis with ASTRÉE

```c
% cat -n scale.c
1 int main () {
2 float x; x = 0.70000001;
3 while (1) {
4 x = x / 3.0;
5 x = x * 3.0;
6 __ASTREE_log_vars((x));
7 __ASTREE_wait_for_clock();
8 }
9 }
% gcc scale.c
% ./a.out
x = 0.699999888079071
% cat scale.config
__ASTREE_max_clock((1000000000));
% astree –exec-fn main –config-sem scale.config –unroll 0 scale.c\n|& grep "x in" | tail -1
direct = <float-interval: x in [0.69999986887, 0.700000047684]>
%```

---

Is the metric system better?

```c
while (1) {
    ... 
    /* x in meters */
    x = x * 100.0;
    /* x in centimeters */
    ... 
    x = x / 100.0;
    /* back to x in meters */
    ...
}
```

Scaling in general can be the source of cumulated rounding errors.

---

Bugs Now Show-Up in Everyday Life

- **Bugs** now appear frequently in everyday life (banks, cars, telephones, ...)

- Example (HSBC bank ATM at 19 Boulevard Sébastopol in Paris, failure on Nov. 21st 2006 at 8:30 am):

---

4 cash machine, cash dispenser, automatic teller machine.
2. What can be done about bugs?

A Strong Need for Software Better Quality

- Poor software quality is not acceptable in safety and mission critical software applications.
- The present state of the art in software engineering does not offer sufficient quality guarantees

Tool-Based Software Design Methods

- New tool-based software design methods will have to emerge to face the unprecedented growth and complexity of critical software
  - E.g. FCPC (Flight Control Primary Computer)
    - A220: 20 000 LOCs,
    - A340: 130 000 LOCs (V1), 250 000 LOCs (V2),
    - A380: 1.000.000 LOCs

Product-based Software Qualification

- An avenue is therefore opened for formal methods which are product-based
  - The software is shown to satisfy a specification
- Main approaches:
  - theorem-proving & proof checking
  - model-checking
  - static analysis
Bug Finding versus Bug Absence Proving

- **Bug-finding methods**: unit, integration, and system testing, dynamic verification, bounded model-checking, error pattern mining, ...
  → Helpful but very partial
- **Absence of bug proving methods**: formally prove that the semantics of a program satisfies a specification
  → Successful in the small but must scale up in the large

Avantages of Static Analysis

- **Formal specifications** are implicit (no need for explicit, user-provided specifications)
- **Formal semantics** are approximated by the static analyzer (no user-provided models of the program)
- **Formal proofs** are automatic (no required user-interaction)
- **Costs** are low (no modification of the software production methodology)
- **Scales up** to 100.000 to 1.000.000 LOCS
- **Large diffusion** in embedded software production industries

Problems with Formal Methods

- **Formal specifications** (abstract machines, temporal logic, ...) are costly, complex, error-prone, difficult to maintain, not mastered by casual programmers
- **Formal semantics** of the specification and programming language are inexistant, informal, unrealistic or complex
- **Formal proofs** are partial (static analysis), do not scale up (model checking) or need human assistance (theorem proving & proof assistants)
- **High costs** (for specification, proof assistance, etc).

Disadvantages of Static Analysis and Remedies

- **Imprecision** (acceptable in some applications like WCET or program optimization)
- **Incomplete** for program verification
- **False alarms** are due to unsuccessful automatic proofs in 5 to 15% of the cases
  → specialization to specific program properties
  → specialization to specific families of programs
  → possibility of refinement

5 For example, Asterée is specialized for runtime errors
6 For example, Asterée is designed for the proof of runtime-errors in real-time synchronous control/command programs
7 For example, Asterée offers parametrizations and analysis directives
3. Informal Introduction to Abstract Interpretation

Abstract Interpretation
There are two fundamental concepts in computer science (and in sciences in general):

- **Abstraction**: to reason on complex systems
- **Approximation**: to make effective undecidable computations

These concepts are formalized by abstract interpretation

---

Applications of Abstract Interpretation

- **Static Program Analysis** [CC77a], [CH78], [CC79] including Dataflow Analysis; [CC79], [CC00], Set-based Analysis [CC95], Predicate Abstraction [Cou03], ...
- **Grammar Analysis and Parsing** [CC03];
- **Hierarchies of Semantics and Proof Methods** [CC92b], [Cou02];
- **Typing & Type Inference** [Cou97];
- **(Abstract) Model Checking** [CC00];
- **Program Transformation** (including program optimization, partial evaluation, etc) [CC02b];

Applications of Abstract Interpretation (Cont’d)

- **Software Watermarking** [CC04];
- **Bisimulations** [RT04];
- **Language-based security** [GM04];
- **Semantics-based obfuscated malware detection** [PCJD07].
- **Databases** [AGM93, BPC01, BS97]
- **Computational biology** [Dan07]
- **Quantum computing** [JP06, Per06]

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

---

References

Approximation

Operational semantics

Safety property

Test/Debugging is Unsafe
Correctness Proof

The correctness proof has two phases.
- In the first analysis phase, the program trace semantics is computed iteratively.\(^6\)
- The verification phase then checks that none of these execution traces can reach a state in which a runtime error can occur.

\(^6\) From a purely mathematical point of view, the set of all execution traces can in principle be formally constructed starting from initial states, then extending iteratively the partial traces from one state to the next one according to the program transition steps until termination on final or error states or passing to the limit for infinite traces (corresponding to non-terminating executions).
Soundness Requirement: Erroneous Abstraction

\( x(t) \)

This situation is always excluded in static analysis by abstract interpretation.

Imprecision \(\Rightarrow\) False Alarms

\( x(t) \)

Soundness Requirement: Erroneous Abstraction

\( x(t) \)

This situation is always excluded in static analysis by abstract interpretation.

The Most Abstract Sound Abstraction
Global Interval Abstraction → False Alarms

Local Interval Abstraction → False Alarms

Refinement by Partitionning

Intervals with Partitionning
Iterator and Abstract Domains

- an iterator for approximating the step by step iterative computation of traces [1], and
- abstract domains representing the effect of program steps and passage to the limit (widening/narrowing [1]).

References


A small graphical language

- objects;
- operations on objects.

Objects

An object is a pair:

- an origin (a reference point ×);
- a finite set of black pixels (on a white background).
Example of an object: a flower

Operations on objects: rotation

- rotation $r[a](o)$ of objects $o$ (of some angle $a$ around the origin):

Example 1 of rotation

- constant objects;
  for example:

  petal =  

  petal $=$  

  $r[45](\text{petal})$ $=$
Example 2 of rotation

\[
\text{flower} = r[-45](\text{flower}) =
\]

Operations on objects: union

- **union** \( o_1 \cup o_2 \) of objects \( o_1 \) and \( o_2 \) = superposition at the origin;
  
  for example:
  
  \[
  \text{corolla} = \text{petal} \cup r[45](\text{petal}) \cup r[90](\text{petal}) \cup r[135](\text{petal}) \cup r[180](\text{petal}) \cup r[225](\text{petal}) \cup r[270](\text{petal}) \cup r[315](\text{petal})
  \]

Operations on objects: add a stem

- stem(\( o \)) adds a **stem** to an object \( o \) (up to the origin, with new origin at the root);

\[
\text{Flower}
\]

\[
\text{flower} = \text{stem(\text{corolla})}
\]
Fixpoints

- corolla = lfp \subseteq F
  \[ F(X) = \text{petal} \cup r[45](X) \]

Contraints

- A corolla is the \( \subseteq \)-least object \( X \) satisfying the two constraints:
  - A corolla contains a petal:
    \( \text{petal} \subseteq X \)
  - and, a corolla contains its own rotation by 45 degrees:
    \( r[45](X) \subseteq X \)
- Or, equivalently \(^{11}\):
  \[ F(X) \subseteq X, \quad \text{where} \quad F(X) = \text{petal} \cup r[45](X) \]

Iterates to fixpoints

- The iterates of \( F \) from the infimum \( \emptyset \) are:
  \[
  X^0 = \emptyset, \\
  X^1 = F(X^0), \\
  \ldots \ldots \ldots, \\
  X^{n+1} = F(X^n), \\
  \ldots \ldots \ldots, \\
  \text{lfp} \subseteq F = X^\omega = \bigcup_{n \geq 0} X^n.
  \]

Iterates for the corolla

\(^{11}\) By Tarski's fixpoint theorem, the least solution is lfp \( \subseteq F \).
The bouquet

- bouquet = r[-45](flower) ∪ flower ∪ r[45](flower)
- The bouquet:

Examples of upper-approximations of flowers

Upper-approximation

- An upper-approximation of an object is an object with:
  - same origin;
  - more pixels.

Abstract objects

- an abstract object is a mathematical/computer representation of an approximation of a concrete object;
Abstract domain

– an abstract domain is a set of abstract objects plus abstract operations (approximating the concrete ones);

Abstraction

– an abstraction function $\alpha$ maps a concrete object $o$ to an approximation represented by an abstract object $\alpha(o)$.

Example 1 of abstraction

Example 2 of abstraction
Comparing abstractions

- larger pen diameters: more abstract;
- different pen shapes: may be non comparable abstractions.

Concretization

- a concretization function $\gamma$ maps an abstract object $\mathfrak{a}$ to the concrete object $\gamma(\mathfrak{a})$ that represents (that is to its concrete meaning/semantics).

Example of concretization

Galois connection 1/4

- $\alpha$ is monotonic.
Galois connection 2/4

- $\gamma$ is monotonic.

\[
\begin{array}{c c c}
\text{flower} & \alpha(\text{flower}) & \gamma(\alpha(\text{flower})) \\
\end{array}
\]

Galois connection 3/4

- for all concrete objects $x$, $\gamma \circ \alpha(x) \supseteq x^{12}$.

\[
\begin{array}{c c c}
\text{flower} & \alpha(\text{flower}) & \gamma(\alpha(\text{flower})) \\
\end{array}
\]

Galois connection 4/4

- for all abstract objects $y$, $\alpha \circ \gamma(y) \subseteq y$.

\[
\begin{array}{c c c}
\text{abstract flower} & \gamma(\text{abstract flower}) & \alpha(\gamma(\text{abstract flower})) \\
\end{array}
\]

Galois connections

\[
\langle \mathcal{D}, \subseteq \rangle \xleftarrow{\gamma} \langle \mathcal{D}, \subseteq \rangle
\]

iff

\[
\begin{align*}
\forall x, y \in \mathcal{D} : x \subseteq y & \Rightarrow \alpha(x) \subseteq \alpha(y) \\
\land \forall x, y \in \mathcal{D} : x \subseteq y & \Rightarrow \gamma(x) \subseteq \gamma(y) \\
\land \forall x \in \mathcal{D} : x \subseteq \gamma(\alpha(x)) & \\
\land \forall y \in \mathcal{D} : \alpha(\gamma(y)) \subseteq x & \\
\end{align*}
\]

iff

\[
\begin{align*}
\forall x \in \mathcal{D}, y \in \mathcal{D} : \alpha(x) \subseteq y & \iff x \subseteq \gamma(y) \\
\end{align*}
\]

\footnote{f : g = \lambda x . f(g(x))}

\[\text{EMSOFT 2007, ESWS, Salzburg, Austria, Sep. 30, 2007} \quad \text{© P. Cousot} \quad \text{EMSOFT 2007, ESWS, Salzburg, Austria, Sep. 30, 2007} \quad \text{© P. Cousot} \quad \text{EMSOFT 2007, ESWS, Salzburg, Austria, Sep. 30, 2007} \quad \text{© P. Cousot} \quad \text{EMSOFT 2007, ESWS, Salzburg, Austria, Sep. 30, 2007} \quad \text{© P. Cousot} \]
Abstract ordering

- $x \sqsubseteq y$ is defined as $\gamma(x) \subseteq \gamma(y)$.

Abstract petal

$\alpha(\includegraphics[width=0.1\textwidth]{petal}) = \includegraphics[width=0.1\textwidth]{petal}$

Specification of abstract operations

- $\text{op}/0 \triangleq \alpha(\text{op}/0)$  
  0-ary
- $\text{op}/1(y) \triangleq \alpha(\text{op}/1(\gamma(y)))$  
  unary
- $\text{op}/2(y, z) \triangleq \alpha(\text{op}/2(\gamma(y), \gamma(z)))$  
  binary
- ...  

Abstract rotations

- $\tau[a](y) \triangleq \alpha(\tau[a](\gamma(y)))$
Abstract rotations

- $\hat{\mathfrak{r}}[a](y) \triangleq \alpha(\hat{\mathfrak{r}}[a](\gamma(y)))$
- $\gamma(\hat{\mathfrak{r}}[a](\alpha(x)))$

A commutation theorem on abstract rotations

- $\alpha(\hat{\mathfrak{r}}[a](x))$
- $\alpha(\gamma(\hat{\mathfrak{r}}[a](\alpha(x))))$
- $\alpha(\hat{\mathfrak{r}}[a](\gamma(\alpha(x))))$
- $\hat{\mathfrak{r}}[a](\alpha(x))$

13 In a Galois connection: $\alpha = \alpha \circ \gamma \circ \alpha$
14 Rotation is the same before or after abstraction
15 Rotation is the same before or after concretization
16 Def. $\hat{\mathfrak{r}}[a]$
Abstract bouquet: (cont’d)

\[\text{abstract flower} = \alpha(\text{concrete flower})\]

\[\text{abstract bouquet} = r[-45](\text{abstract flower}) \cup \text{abstract flower} \cup r[-45](\text{abstract flower})\]

\[= r[-45](\alpha(\text{concrete flower})) \cup \alpha(\text{concrete flower}) \cup r[-45](\alpha(\text{concrete flower}))\]

\[= \alpha(r[-45](\text{concrete flower})) \cup \alpha(\text{concrete flower}) \cup \alpha(r[-45](\text{concrete flower}))\]

\[= \alpha(\text{concrete bouquet})\]
Abstract fixpoint

- abstract corolla = α(concrete corolla) = α(lfp \subseteq F)
  where F(X) = petal \cup r[45](X))

Abstract transformer F

- α(F(X))
  = α(petal \cup r[45](X))
  = α(petal) \sqcup α(r[45](X))
  = α(petal) \sqcup F[45](α(X))
  = abstract petal \sqcup F[45](α(X))
  = Fα(X)
by defining

F(X) = abstract petal \sqcup F[45](X)
and so:
- abstract corolla = α(concrete corolla) = α(lfp \subseteq F) = lfp \subseteq F

Iterates for the abstract corolla

Abstract interpretation of the (graphic) language

- Similar, but by syntactic induction on the structure of programs of the language;
On abstracting properties of graphic objects

- A graphic object is a set of (black) pixels (ignoring the origin for simplicity);
- So a property of graphic objects is a set of graphic objects that is a set of sets of (black) pixels (always ignoring the set of origins for simplicity);
- Was there something wrong?

No, because we implicitly used the following implicit looseness abstraction:

\[
\langle \wp(\wp(\mathcal{P})), \subseteq \rangle \xleftarrow{\gamma_0} \xrightarrow{\alpha_0} \langle \wp(\mathcal{P}), \subseteq \rangle
\]

where:

- \(\mathcal{P}\) is a set of pixels (e.g. pairs of coordinates)
- \(\alpha_0(X) = \bigcup X\)
- \(\gamma_0(Y) = \{G \in \mathcal{P} | G \subseteq Y\}\)
Semantics

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

States and Traces

- States in $\Sigma$, describe an instantaneous snapshot of the execution.
- Traces are finite or infinite sequences of states in $\Sigma$, two successive states corresponding to an elementary program step.
- In that case
  - $\Sigma^n \triangleq \{0, n\} \rightarrow \Sigma$ traces of length $n = 1, \ldots, +\infty$.
  - $T \triangleq \bigcup_{n=1}^{+\infty} \Sigma^n$ all possible traces.
  - $D \triangleq \wp(T)$ semantic domain.

Example: Operational Semantics

- The operational semantics describes all possible program executions as a set of maximal execution traces.
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 \( \mathcal{P} \triangleq \mathcal{P}(\mathcal{D}) \).

The Complete Lattice of Semantic Properties

The semantic properties have a **complete lattice** (indeed Boolean lattice) structure:

\[ (\mathcal{P}(\mathcal{D}), \subseteq, \varnothing, \mathcal{D}, \cup, \cap, \neg) \]

The implication/set inclusion \( \subseteq \) is a **partial order**:

- reflexive: \( \forall X \in \mathcal{P}(\mathcal{D}) : X \subseteq X \)
- antisymmetric:
  \[
  \forall X, Y \in \mathcal{P}(\mathcal{D}) : X \subseteq Y \land Y \subseteq X \implies X = Y
  \]
- transitive:
  \[
  \forall X, Y, Z \in \mathcal{P}(\mathcal{D}) : X \subseteq Y \land Y \subseteq Z \implies X \subseteq Z
  \]

The join/union \( \cup \) is the **least upper bound** (lub):

- \( \cup \) is an upper bound: \( \forall (X_i \in \mathcal{P}(\mathcal{D}), i \in \Delta) : \forall j \in \Delta : X_j \subseteq \bigcup_{i \in \Delta} X_i \)
- \( \cup \) is the least one: \( \forall (X_i \in \mathcal{P}(\mathcal{D}), i \in \Delta) : \forall Y \in \mathcal{P}(\mathcal{D}) : (\forall j \in \Delta : X_j \subseteq Y) \implies (\bigcup_{i \in \Delta} X_i \subseteq Y) \)

Example: Properties of a Trace Semantics

- \( \mathcal{T} \) all possible traces
- \( \mathcal{D} \triangleq \mathcal{P}(\mathcal{T}) \) semantic domain (sets of traces)
- \( \mathcal{P} \triangleq \mathcal{P}(\mathcal{D}) \triangleq \mathcal{P}(\mathcal{P}(\mathcal{T})) \) properties (sets of sets of traces)
The meet/union $\cap$ is the greatest lower bound (glb):
- $\cap$ is a lower bound: $\forall (X_i \in \mathcal{P}(D), i \in \Delta) : \forall j \in \Delta : \bigcap_{i \in \Delta} X_i \subseteq X_j$
- $\cap$ is the greatest one: $\forall (X_i \in \mathcal{P}(D), i \in \Delta) : \forall Y \in \mathcal{P}(D) : (\forall j \in \Delta : Y \subseteq X_j) \Rightarrow (Y \subseteq \bigcap_{i \in \Delta} X_i)$

The infimum/empty set is $\emptyset$ such that
- $\forall X \in \mathcal{P}(D) : \emptyset \subseteq X$
- $\emptyset = \bigcap \mathcal{P}(D) = \bigcup \emptyset$

The supremum is $D$ such that
- $\forall X \in \mathcal{P}(D) : X \subseteq D$
- $D = \bigcup \mathcal{P}(D) = \bigcap \emptyset$

The complement $\neg X \triangleq D \setminus X$ satisfies
- $X \cap \neg X = \emptyset$
- $X \cup \neg X = D$

and is unique

Lattice Theory
- Lattice theory was introduced by Garrett Birkhoff [2]
- Weakens set theory while keeping essential results
Partial Order

\( \langle L, \subseteq \rangle \)

- \( L \) is a set
- The relation \( \subseteq \) on \( L \) is a reflexive, antisymmetric and transitive

The lub/glb might not exist for finite subsets of \( L \).

Complete Lattices

\( \langle L, \subseteq, \bot, \top, \cup, \cap \rangle \)

- \( \langle L, \subseteq \rangle \) is a partial order
- The lub \( \bigcup X \) does exist for all subsets \( X \) of \( L \)
- It follows that the glb \( \bigcap X \triangleq \bigcup \{ y \mid \forall x \in X : y \subseteq x \} \) does exist for all subsets \( X \) of \( L \)
- It follows that \( L \) has an infimum \( \bot = \bigcap L = \bigcup \emptyset \) and a supremum \( \top = \bigcup L = \bigcap \emptyset \)

The complement may not exist for all elements of \( L \) and may not be unique. Any finite lattice is complete.

Lattices

\( \langle L, \subseteq, \cup, \cap \rangle \)

- \( \langle L, \subseteq \rangle \) is a partial order
- The lub \( x \cup y \) exists for all \( x, y \in L \) (whence for any finite subset of \( L \))
- The glb \( x \cap y \) exists for all \( x, y \in L \) (whence for any finite subset of \( L \))

The lub/glb might not exist for infinite subsets of \( L \).

Examples of (Complete) Lattices

\[ \text{Partial order} \quad \text{Lattices} \quad \text{Complete lattice} \]
Duality Principle

- The dual of $\langle L, \subseteq, \bot, \top, \sqcup, \sqcap \rangle$ is $\langle L, \supseteq, \top, \bot, \sqcap, \sqcup \rangle$.
- If a statement is true in lattice theory, its dual is also true.
- Hence, there is no need for a dual of abstract interpretation theory.\(^\text{20}\)

\(^{20}\) Despite numerous counter-examples, see e.g. E.M. Clarke, O. Grumberg, and D.E. Long, Model Checking and Abstraction, TOPLAS 16:5(1512–1542), 1994.

Collecting Semantics

- The strongest property of a system $p \in P$ is its semantics $\{S[p]\}$, called the collecting semantics $C[p] \triangleq \{S[p]\}$.

Verification

- The satisfaction of a specification $P \in P$ by a system $p$ (more precisely by the system semantics $S[p]$) is $S[p] \in P$.
- Satisfaction can equivalently be defined as the proof that $C[p] \subseteq P$ i.e. the strongest program property implies its specification.
Undecidability

- The proof that 
  \[ \mathcal{C}[\mathcal{P}] \subseteq \mathcal{P} \]
  is not mechanizable (Gödel, Turing).

Abstraction

To prove 
  \[ \mathcal{C}[\mathcal{P}] \subseteq \mathcal{P} \]
one can use a sound over-approximation of the collecting semantics

\[ \mathcal{C}[\mathcal{P}] \subseteq \mathcal{C}[\mathcal{P}] \]

and a sound under-approximation of the property

\[ \mathcal{P} \subseteq \mathcal{P} \]

and make the correctness proof in the abstract

\[ \mathcal{C}[\mathcal{P}] \subseteq \mathcal{P} \]

Abstract Domain

- For automated proofs, \( \mathcal{C}[\mathcal{P}] \) and \( \mathcal{P} \) 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

\[ \langle \mathcal{P}, \subseteq \rangle \]
Concretization Function

- The abstract to concrete correspondence is given by a concretization function
\[ \gamma \in \overline{P} \mapsto p \]
providing the meaning \( \gamma(P) \) of abstract properties \( P \).

- For abstract reasonings to be valid in the concrete, \( \gamma \) should preserve the abstract implication
\[ \forall Q_1, Q_2 \in \overline{P} : (Q_1 \subseteq Q_2) \Rightarrow (\gamma(Q_1) \subseteq \gamma(Q_2)) \]

Soundness of the Abstraction

- The soundness of the abstract over-approximation of the collecting semantics is now
\[ C[p] \subseteq \gamma(C[p]) \]

- The soundness of the abstract under-approximation of the property is now
\[ \gamma(P) \subseteq P \]

Abstract Proofs

- Then, the abstract proof
\[ C[p] \subseteq P \]

implies
\[ \gamma(C[p]) \subseteq \gamma(P) \]
and by soundness of the abstraction
\[ C[p] \subseteq \gamma(C[p]) \quad \text{and} \quad \gamma(P) \subseteq P \]
we have proved correctness in the concrete
\[ C[p] \subseteq P \]

Galois Connections
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
  \[ \alpha \in \mathcal{P} \mapsto \overline{P} \]
such that
  - for all \( P \in \mathcal{P} \), \( \alpha(P) \in \overline{P} \) is an abstract over-approximation of \( P \), so
  \[ P \subseteq \gamma(\alpha(P)) \]

  and,
  \[ \forall Q \in \overline{P} : P \subseteq \gamma(Q) \implies \alpha(P) \subseteq Q \]
  (whence \( \gamma(\alpha(P)) \subseteq \gamma(Q) \) by monotony of \( \gamma \)).

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].

  \[ \forall P \in \mathcal{P} : \forall Q \in \overline{P} : P \subseteq \gamma(Q) \iff \alpha(P) \subseteq Q \]

written

\[ \langle \mathcal{P}, \subseteq \rangle \xrightarrow{\gamma} \overline{\mathcal{P}}, \subseteq \]

Reference

Galois Connection Preserve Existing Joins

If

\[ \text{poset} \rightarrow \langle L, \sqsubseteq \rangle \xleftarrow{\gamma} \langle L, \sqsubseteq \rangle \xleftarrow{\alpha} \text{poset} \] (1)

and \( \bigsqcup_i X_i \) exists in \( \langle L, \sqsubseteq \rangle \) then

\[ \alpha(\bigsqcup_i X_i) = \bigsqcup_i \alpha(X_i) \]

Reciprocally, if \( \alpha \) preserves existing joins then it has a unique adjoint \( \gamma \) satisfying Eq. (1)

---

I.1 — Traditional View of Program Properties

- In the operational trace semantics example \( D \triangleq \rho(T) \) so properties are

\[ \mathcal{P} \triangleq \rho(\rho(T)) \]

where \( T \) is the set of traces.

- The traditional view of program properties as set of traces \([4],[5]\) is an abstraction.

References


---

Examples of Abstractions I

I.1 — Example of Program Properties

- An example of program property is

\[ P_{01} \triangleq \{\{s0 \mid s \in T\}, \{s1 \mid s \in T\}\} \in \mathcal{P} \]

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 \([4],[5]\).
I.1 — Looseness Abstraction

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

\[ \alpha_{\|} \in \wp(\wp(T)) \mapsto \wp(T), \]
\[ \alpha_{\|}(P) \triangleq \bigcup P \]

with concretization

\[ \gamma_{\|} \in \wp(T) \mapsto \wp(\wp(T)), \]
\[ \gamma_{\|}(Q) \triangleq \wp(Q). \]

- An example is \( \alpha_{\|}(P_{01}) = \{\sigma_0, \sigma_1 \mid \sigma \in \mathcal{T}\} \) specifying that execution always terminate, either with 0 or with 1.

I.2 — Transition Abstraction

- The transition abstraction

\[ \alpha_{\tau} \in \wp(T) \mapsto \wp(\Sigma \times \Sigma) \]

collects transitions along traces.

\[ \alpha_{\tau}(\sigma_0 \ldots \sigma_n) \triangleq \{\sigma_i \rightarrow \sigma_{i+1} \mid 0 \leq i < n\}, \]
\[ \alpha_{\tau}(\sigma_0 \ldots \sigma_i \ldots) \triangleq \{\sigma_i \rightarrow \sigma_{i+1} \mid i \geq 0\}, \]
and

\[ \alpha_{\tau}(T) \triangleq \bigcup \{\alpha(\sigma) \mid \sigma \in T\}. \]

- The concretization \( \gamma_{\tau} \in \wp(\Sigma \times \Sigma) \mapsto \wp(T) \) is

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

I.2 — Transition System Abstraction

- The abstraction may also collect initial states

\[ \alpha_i(T) \triangleq \{\sigma_0 \mid \sigma \in T\} \]

so

\[ \alpha_{i\tau}(T) \triangleq \langle \alpha_i(T), \alpha_{\tau}(T) \rangle. \]

- We let

\[ \gamma_{i\tau} \triangleq \gamma_i(\iota) \cap \gamma_{\tau}(\tau) \]

where

\[ \gamma_i(\iota) \triangleq \{\sigma \in \mathcal{T} \mid \sigma_0 \in \iota\} \]

and

\[ \langle \alpha_{i\tau}, \gamma_{i\tau} \rangle \] is a Galois connection.
I.2 — Transition System Abstraction (Cont’d)

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

References

I.3 — Input-Output Abstract Semantics

– The input-output abstraction

\( \alpha_{io} : p(\mathcal{T}) \mapsto p(\Sigma \times (\Sigma \cup \{\bot\})) \)

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

\[
\alpha_{io}(\sigma_0 \ldots \sigma_n) = (\sigma_0, \sigma_n),
\]

\[
\alpha_{io}(\sigma_0 \ldots \sigma_i \ldots) = (\sigma_0, \bot),
\]

and

\[
\alpha_{io}(T) = \{ \alpha_{io}(\sigma) \mid \sigma \in T \}.
\]
I.3 — Input-Output Abstraction (Cont’d)

- The input-output abstraction $\alpha_{io}$ 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


I.4 — Reachability Abstraction

- The reachability abstraction collects states along traces.

\[
\alpha_r \in \mathcal{P}(T) \mapsto \mathcal{P}(\Sigma),
\]

\[
\alpha_r(T) \triangleq \{ \sigma_i : \exists n \in [0, +\infty] : \sigma \in \Sigma^n \cap T \land i \in [0, n] \} \subseteq \{ s' \in \Sigma | \exists s \in i : \langle s, s' \rangle \in \tau^* \}
\]

where $\alpha_r(T) = \langle i, \tau \rangle$ is the transition abstraction and $\tau^*$ is the reflexive transitive closure of $\tau$.

21 We may have $\subseteq$ when $T \neq \gamma_r(T_\alpha(T))$. We assume $T = \gamma_r(T_\alpha(T))$ in the rest of the talk.
I.4 — Invariants

Expressed in logical form, the reachability abstraction $\alpha$ provides a system invariant $\alpha(C[p])$
that is the set of all states that can be reached along some execution of the system $p$ [11], [12].

References


Example of invariant (I)

Not inductive (and too weak!)

Example of invariant (II)

Inductive and precise enough!
Example of Absence of Best Abstraction

- \( Z \): set of integers
- \( \wp(Z) \): integer properties \(^{22}\)
- \( \{\bot, +, \cdot, T\} \): abstract signs, with

\[
\begin{align*}
\gamma(\bot) &= \emptyset \\
\gamma(+) &= \{n \in \mathbb{Z} \mid n \geq 0\} \\
\gamma(\cdot) &= \{n \in \mathbb{Z} \mid n \leq 0\}
\end{align*}
\]

- 0 has no best abstraction (can be either + or \( \cdot \))

What to do in Absence of Best Abstraction

1. Close the abstract domain by intersection (Moore family)
   e.g. \( \{\bot, +, 0, \cdot, T\} \): abstract signs (with \( \gamma(0) = \{0\} \) so
   \( 0 + + = + \) and \( 0 + \cdot = \cdot \))
   \( \Rightarrow \) In general, there are infinitely many possible choices
   so the Moore closure is quite complex \(^{23}\)

2. Try all possible choices and locally keep the best one \(^{24}\)
   \( \Rightarrow \) Make arbitrary choices \(^{25}\)

\(^{22}\) e.g. possible values of an integer variable at runtime

\(^{23}\) e.g. polyhedra can be closed in convex sets much harder to represent in machines

\(^{24}\) In general impossible due to combinatorial explosion

\(^{25}\) Using a concretization function \( \gamma \) this choice can be made locally, while with an \( \alpha \) it is made globally, once

In Absence of Best Abstraction (Cont’d)

- Among the possible choices, one may be locally preferable, e.g.
  - \( 0 + + \), 0 should be abstracted to + (since + + = +
    while + + = T)
  - \( 0 + \cdot \), 0 should be abstracted to \( \cdot \) (since \( \cdot + \cdot = \cdot \)
    while \( + + = T \))
Effective computable approximations of an [in]finite set of points; Signs

\[ \begin{align*}
&\{ x \geq 0 \\
&\{ y \geq 0 \\
&x < 0 \\
y < 0 \\
\end{align*} \]

Non-relational
Best abstraction (with 0).

---

Effective computable approximations of an [in]finite set of points; Intervals

\[ \begin{align*}
&\{ x \in [19, 77] \\
&\{ y \in [20, 07] \\
\end{align*} \]

Non-relational
Best abstraction.
Effective computable approximations of an [in]finite set of points; Octagons

\[ \begin{align*}
1 & \leq x \leq 9 \\
1 & \leq y \leq 9 \\
x + y & \leq 77 \\
x - y & \leq 99
\end{align*} \]

Weakly relational
Best abstraction.


Effective computable approximations of an [in]finite set of points; Simple congruences

\[ \begin{align*}
x & = 19 \mod 77 \\
y & = 20 \mod 99
\end{align*} \]

Non-relational
Best abstraction.


Effective computable approximations of an [in]finite set of points; Polyhedra

\[ \begin{align*}
19x + 77y & \leq 2004 \\
20x + 03y & \geq 0
\end{align*} \]

Relational
No best abstraction.


Effective computable approximations of an [in]finite set of points; Linear congruences

\[ \begin{align*}
x & = 9 \mod 8 \\
2x - 1y & = 9 \mod 9
\end{align*} \]

Relational
Best abstraction.

Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences

\[ \begin{align*}
12x + 9y &\in [0, 77] \mod 10 \\
2x - 1y &\in [0, 99] \mod 11
\end{align*} \]

Relational
No best abstraction.

\[ P \xrightarrow{\sigma} Q \implies \bar{C}[P] \subseteq \bar{P} \]

Soundness of Abstractions

- An abstraction is sound \([14]\) if the proof in the abstract implies the concrete property

\[ \bar{C}[P] \subseteq \bar{P} \implies \bar{C}[P] \subseteq P. \]

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

Properties of Abstractions

Example of Unsound Abstraction (Bounded Model Checking)
Completeness of Abstractions

- An abstraction is complete [15] if the fact that the system is correct can always be proved in the abstract

\[ C[p] \subseteq P \implies \overline{C[p]} \subseteq \overline{P}. \]

References

Refinement of Abstractions

- False alarms can always be avoided by refinement of the abstraction [16].

References

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)!

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** [17].

**References**


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 [19]
- or, in static analysis **trace partitioning** may be more adequate that state-based reachability analysis [20].

**References**


Adequation of Abstractions

- The **reachability abstraction** is **sound and complete** for invariance/safety proofs.
- 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 $C[p] \subseteq \gamma_r(S)$ can always be done as $\alpha_r(C[p]) \subseteq S$.
- This is the fundamental remark of Floyd [18] that it is not necessary to reason on traces to prove invariance properties.

**References**


Combination of Abstractions

[33] Again, assuming $T = \gamma_r(\alpha_r(T))$.
Reduced Product of Abstract Domains

To combine abstractions

\[ \langle D_1, \subseteq \rangle \xrightarrow{\alpha_1} \langle D_1^i, \subseteq_1 \rangle \text{ and } \langle D_2, \subseteq \rangle \xrightarrow{\alpha_2} \langle D_2^i, \subseteq_2 \rangle \]

the reduced product is

\[ \alpha(X) \triangleq \cap \{ \langle x, y \rangle \mid X \subseteq \gamma_1(x) \land X \subseteq \gamma_2(y) \} \]

such that \( \subseteq \triangleq \subseteq_1 \times \subseteq_2 \) and

\[ \langle D, \subseteq \rangle \xrightarrow{\alpha \times \gamma} \langle \alpha(D), \subseteq \rangle \]

Example: \( x \in [1, 9] \land x \mod 2 = 0 \) reduces to \( x \in [2, 8] \land x \mod 2 = 0 \)

Reduction in Astéé

- The computation of an abstract transformer \( F_1 \) for an abstract domain \( D_1 \) can use an abstract invariant computed by another abstract domain \( D_2 \).
- The two abstract domains communicate symbolically through a channel.\(^{34}\)
- A fixed communication order is used (so reduction cannot prevent to widening/narrowing convergence enforcement).

\(^{34}\) using a common language to communicate whereas the representation of invariants may be quite different.

Semantic Transformer

- The concrete/semantic transformer \( F \) describes the effect of program commands: if \( P \) describes behaviors before/after a command, then \( F(P) \) describes behaviors after/before this command

\[ F \in \mathcal{P} \xrightarrow{\text{mon}} \mathcal{P} \]

- Assumed to be monotonic: \( \forall P, P' \in \mathcal{P} : (P \subseteq P') \implies (F(P) \subseteq F(P')) \).
- Intuition: the more behaviors before/after a command, the more after/before the command.
Abstract Transformer

- The abstract transformer $\overline{F}$ is

$$\overline{F} \in \mathcal{P} \mapsto \mathcal{P}$$

- Might not be monotonic (because of non-monotonic widening/narrowing, see later)

Best Abstract Transformer

Given $\langle \subseteq, \mathcal{P} \rangle$, $\langle \subseteq, \overline{\mathcal{P}} \rangle$, $\gamma \in \overline{\mathcal{P}} \mapsto \mathcal{P}$ and $F \in \mathcal{P} \overset{\text{mon}}{\mapsto} \mathcal{P}$, $\overline{F} \in \overline{\mathcal{P}} \mapsto \overline{\mathcal{P}}$ is the best approximation of $F$ iff

1. $\overline{F}$ is an over approximation of $F$:

$$\forall P \in \mathcal{P}, \overline{P} \in \overline{\mathcal{P}} : (P \subseteq \gamma(P)) \implies (F(P) \subseteq \gamma(\overline{F}(\overline{P})))$$

2. $\overline{F}$ is the most precise over approximation of $F$: if

$$\forall P \in \mathcal{P}, \overline{P} \in \overline{\mathcal{P}} : (P \subseteq \gamma(P)) \implies (F(P) \subseteq \gamma(\overline{F}(\overline{P})))$$

then

$$\forall P \in \overline{\mathcal{P}} : F(\overline{P}) \subseteq \mathcal{G}(\overline{P})$$

Given $\gamma$, the best abstract transformer might not exist!
Existence of a Best Abstract Transformer for Galois Connections

If 

\[ \langle P, \subseteq \rangle \xrightarrow{\alpha} \langle \overline{P}, \subseteq \rangle \]

then the best overapproximation of \( F \in \mathcal{P} \xrightarrow{\text{mon}} \overline{\mathcal{P}} \) is

\[ F' \triangleq \alpha \circ F \circ \gamma. \]

**Proof**

- \( P \subseteq \gamma(\overline{P}) \)
  \[ \implies F(P) \subseteq F(\gamma(\overline{P})) \]
  \[ \implies \alpha(F(P)) \subseteq \alpha(F(\gamma(\overline{P}))) \]
  \[ \implies F(\gamma(P)) \subseteq \gamma(\alpha \circ F \circ \gamma(P)) \]
  monotony of \( F \)
  monotony of \( \alpha \)
  def. Galois connection
  proving \( \alpha \circ F \circ \gamma \) to be an abstract overapproximation of \( F \).

- If \( \overline{G} \) is an abstract overapproximation of \( F \):
  \[ \forall P \in \mathcal{P}, \overline{P} \in \overline{\mathcal{P}} : (P \subseteq \gamma(\overline{P})) \implies (F(P) \subseteq \gamma(\overline{G}(\overline{P}))) \]
  then
  \[ \implies F(\gamma(\overline{P})) \subseteq \gamma(\overline{G}(\overline{P})) \]
  \[ \implies \alpha \circ F \circ \gamma(\overline{P}) \subseteq \overline{G}(\overline{P}) \]
  def. Galois connection
  proving \( \alpha \circ F \circ \gamma \) to be the best abstract overapproximation of \( F \).

**Fixpoints**

A fixpoint of \( F \) is \( X \) such that \( X = F(X) \)

- May not exist, may have [infinitely] many
- May have a least one \( \text{lfp} \subseteq F \) for a partial order \( \sqsubseteq \):
  \[ F(\text{lfp} \subseteq F) = \text{lfp} \subseteq F \]
  \[ X = F(X) \implies \text{lfp} \subseteq F \sqsubseteq X \]
  least one
Fixpoint Theorem I (Tarski)

- The set of fixpoints of a monotone operator $F \in L \xrightarrow{\text{mon}} L$ on a complete lattice $(L, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ is a complete lattice.  
- The least fixpoint is the least post-fixpoint:

$$\text{lfp} \sqsubseteq F = \bigcap \{x \in L \mid F(x) \sqsubseteq x\}$$

35 Hence, not empty!

Fixpoint Theorem II (Kleene)

- The transfinite iterates of $F$ on a poset $(L, \sqsubseteq)$
  - $X^0 = \bot$
  - $X^{\eta+1} \triangleq F(X^\eta)$ $\eta + 1$ successor ordinal
  - $X^\lambda \triangleq \bigsqcup_{\eta < \lambda} X^\eta$ $\lambda$ limit ordinal
- If $F$ is monotone and the lubs $\sqcup$ do exist then the iterates are increasing, ultimately stationary, with limit $\text{lfp} \sqsubseteq F$

So $\text{lfp} \sqsubseteq F$ can always be computed iteratively.

36 e.g. in a complete lattice or a cpo for which lubs of increasing chains do exist.

Example of Fixpoint:
Reflexive Transitive Closure
Example: Reflexive Transitive Closure

- $\tau \in \mathcal{P}(\Sigma \times \Sigma)$

Transition relation

Example: Reflexive Transitive Closure (Cont’d)

- $\tau^* \in \mathcal{P}(\Sigma \times \Sigma)$

Reflexive transitive closure

Example: Reflexive Transitive Closure (Cont’d)

$$\tau^* = \text{lfp} \subseteq F \quad \text{where} \quad F(X) = 1_\Sigma \cup \tau \circ X$$

$$= \bigcup_{n \geq 0} t^n$$

and

- $t^0 = 1_\Sigma = \{\langle x, x \rangle \mid x \in \Sigma\}$,
- $t^{n+1} = t^n \circ t = t \circ t^n$

Example: Reflexive Transitive Closure (Cont’d)

$$\tau^* = \text{lfp} \subseteq \lambda X . \tau^0 \cup X \circ \tau$$

Proof

$X^0 = \emptyset \quad \text{basis}$

$X^1 = \tau^0 \cup X^0 \circ \tau = \tau^0$

$X^2 = \tau^0 \cup X^1 \circ \tau = \tau^0 \cup \tau^0 \circ \tau = \tau^0 \cup \tau^1$

$\ldots \quad \ldots$

$X^n = \bigcup_{0 \leq i < n} \tau^i \quad (\text{induct on hypotheses})$
\[
X^{n+1} = r^0 \cup X^n \circ \tau \\
\text{induction}
\]
\[
= r^0 \cup \left( \bigcup_{0 \leq i < n} \tau^i \circ \tau \right)
\]
\[
= r^0 \cup \bigcup_{0 \leq i < n} (\tau^i \circ \tau)
\]
\[
= r^0 \cup \bigcup_{1 \leq j \leq n+1} (\tau^{j-1})
\]
\[
= r^0 \cup \left( \bigcup_{1 \leq j \leq n+1} \tau^j \right) \circ \tau
\]
\[
= \bigcup_{0 \leq i < n+1} \tau^i
\]
\[
\ldots \ldots
\]
\[
X^{\omega+1} = r^0 \cup X^\omega \circ \tau \\
\text{convergence}
\]
\[
= r^0 \cup \left( \bigcup_{n \geq 0} \left( \bigcup_{0 \leq i < n} \tau^i \circ \tau \right) \right)
\]
\[
= r^0 \cup \left( \bigcup_{n \geq 0} \left( \bigcup_{0 \leq i < n} \tau^i \right) \right)
\]
\[
= r^0 \cup \left( \bigcup_{n \geq 0} \tau^{n+1} \right)
\]
\[
= r^0 \cup \bigcup_{n \geq 0} \tau^n
\]
\[
= \tau^*
\]
\[
X^\omega = \bigcup_{n \geq 0} X^n \\
\text{limit}
\]
\[
= \bigcup_{n \geq 0} \left( \bigcup_{0 \leq i < n} \tau^i \right)
\]
\[
= \bigcup_{n \geq 0} \tau^n
\]
\[
= \tau^*
\]
\[
\text{Iterates}
\]
\[
X^0 \\
X^1 \\
X^2 \\
X^3 \\
X^4 \\
X^5 = \tau^*
\]
Exact Fixpoint Abstraction

- $F \in L \mapsto L$ monotonic on the complete lattice $\langle L, \sqsubseteq, \bot, \top, \cup, \cap \rangle$
- $\bar{F} \in \bar{L} \mapsto \bar{L}$ monotonic on $\langle \bar{L}, \sqsubseteq, \bar{\top}, \bar{\bot}, \bar{\cup}, \bar{\cap} \rangle$
- $\langle L, \sqsubseteq \rangle \xrightarrow{\gamma} \langle L, \sqsubseteq \rangle$ Galois connection
- $\alpha \circ F = \bar{F} \circ \alpha$ Commutation

implies

$$\alpha(\uparrow F) = \uparrow \bar{F}$$

Example of Exact Fixpoint Abstraction: Reachable States

Example: Transition System

Transition system

- $\langle \Sigma, \tau \rangle$
Example: Reachable States

- \( \mathcal{I} \subseteq \Sigma \)
- \( \mathcal{R} \triangleq \{ s' | \exists s \in \mathcal{I} : \tau^*(s, s') \} \)

Initial states
Reachable states

Example: Post-Image

\[
\text{post}[\tau^*]\mathcal{I} = \{ s' | \exists s \in \mathcal{I} : (s, s') \in \tau \}
\]

We have \( \text{post}[\bigcup_{i \in \Delta} \tau^i]\mathcal{I} = \bigcup_{i \in \Delta} \text{post}[\tau^i]\mathcal{I} \) so \( \alpha = \lambda \tau \cdot \text{post}[\tau^*] \mathcal{I} \) is the lower adjoint of a Galois connection.

Example: Postimage Galois Connection

Given \( \mathcal{I} \in \wp(\Sigma) \),

\[
\langle \wp(\Sigma \times \Sigma), \subseteq \rangle \xrightarrow{\gamma} \langle \wp(\Sigma), \subseteq \rangle \xrightarrow{\lambda \tau \cdot \text{post}[\tau^*] \mathcal{I}}
\]

where

\[
\gamma(R) \triangleq \{ (s, s') | (s \in \mathcal{I}) \Rightarrow (s' \in R) \}
\]
Example: Reachable States (Cont’d)

Reachability is an abstraction of the transitive closure:

\[ \alpha \in \wp(\Sigma \times \Sigma) \rightarrow \wp(\Sigma) \]

\[ \alpha(t) \triangleq \text{post}[t] \subseteq \{ s' \mid \exists s \in \mathcal{I} : t(s, s') \} \]

\[ \mathcal{R} = \alpha(\tau^*) \]

\[ = \alpha(\text{lfp} F) \quad \text{where} \quad F(X) = 1_\Sigma \cup \tau \circ X \]

Example: Discovering \( F \) by calculus

\[ \alpha \circ F \]

\[ \alpha \circ (\lambda X \cdot \tau^0 \cup X \circ \tau) \]

\[ = \lambda X \cdot \alpha(\tau^0 \cup X \circ \tau) \]

\[ = \lambda X \cdot \alpha(\tau^0) \cup \alpha(X \circ \tau) \]

\[ = \lambda X \cdot \text{post}[\tau^0] \mathcal{I} \cup \text{post}[X \circ \tau] \mathcal{I} \]

We go on by cases.

Example: Reachable states in fixpoint form

\[ \text{post}[\tau^*] \mathcal{I}, \mathcal{I} \subseteq \Sigma \text{ g ven} \]

\[ = \alpha(\tau^*) \quad \text{where} \quad \alpha(\tau) = \text{post}[\tau] \mathcal{I} = \{ s' \mid \exists s \in \mathcal{I} : \langle s, s' \rangle \in \tau \} \]

\[ = \alpha(\text{lfp} \subseteq \lambda X \cdot \tau^0 \cup X \circ \tau) \]

\[ = \text{lfp} \subseteq \lambda X \cdot \tau^0 \cup X \circ \tau \]

\[ = \text{post}[\tau^0] \mathcal{I} \]

\[ = \{ s' \mid \exists s \in \mathcal{I} : \langle s, s' \rangle \in \tau^0 \} \]

\[ = \{ s' \mid \exists s \in \mathcal{I} : \langle s, s' \rangle \in \{ \langle s \rangle : s \in \mathcal{S} \} \} \]

\[ = \{ s' \mid \exists s \in \mathcal{I} \} \]

\[ = \mathcal{I} \]
\[ \text{post}[X \circ \tau]|I \]
\[ = \{ s' \mid \exists s \in I : (s, s') \in (X \circ \tau) \} \]
\[ = \{ s' \mid \exists s \in I : (s, s') \in \{(s, s'') \mid \exists s' : (s, s'') \in X \land (s', s'') \in \tau \} \} \]
\[ = \{ s' \mid \exists s \in I : \exists s'' \in S : (s, s'') \in X \land (s', s'') \in \tau \} \]
\[ = \{ s' \mid \exists s'' \in S : (s'' \in \{(s'') \mid \exists s \in I : (s, s'') \in X \land (s', s'') \in \tau \} \]
\[ = \{ s' \mid \exists s'' \in S : s'' \in (s'' \mid \exists s \in I : (s, s'') \in X \land (s', s'') \in \tau \} \]
\[ = \text{post}[\tau](\alpha(X)) \]

\[ \alpha \circ F = \alpha \circ (\lambda X \cdot \tau^0 \cup X \circ \tau) \]
\[ = \ldots \]
\[ = \lambda X \cdot \text{post}[\tau^0]|I \cup \text{post}[X \circ \tau]|I \]
\[ = \lambda X \cdot I \cup \text{post}[\tau](\alpha(X)) \]
\[ = \lambda X \cdot \overline{F}(\alpha(X)) \]

by defining:
\[ F = \lambda X \cdot I \cup \text{post}[\tau](X) \]
proving:
\[ \text{post}[\tau^*](I) = \text{ifp} \subseteq \lambda X \cdot I \cup \text{post}[\tau](X) \]
Fixpoint Approximation

- \( F \in L \mapsto L \) monotonic on the complete lattice \( \langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle \)
- \( \overline{F} \in L \mapsto L \) on \( \langle L, \sqsubseteq \rangle \)
- \( \gamma \in L \mapsto L \), monotonic, such that \( F \circ \gamma \sqsubseteq \gamma \circ \overline{F} \)

implies

\[ \overline{F}(X) \sqsubseteq X \implies \text{lfp} F \sqsubseteq \gamma(X) \]

Example: Sign Analysis

1) Reachable states of \( X \) in

\[ X_0 = \mathbb{Z} \]
\[ X_1 = \{100\} \cup X_3 \]
\[ X_2 = X_1 \cap \{z \in \mathbb{Z} | z > 0\} \]
\[ X_3 = \{z - 1 | z \in X_2\} \]
\[ X_4 = X_1 \cap \{z \in \mathbb{Z} | z \leq 0\} \]

of the form:

\[ X = \overline{F}(\bar{X}) \]

where

\[ \bar{X} = (X_0, X_1, \ldots, X_4) \]

Example: Sign Analysis (Cont’d)

2) Overapproximation by the sign of \( X \) in

\[ X_0 = \top \]
\[ X_1 = + \cup X_3 \]
\[ X_2 = X_1 \cap + \]
\[ X_3 = X_2 \cup 1 \]
\[ X_4 = X_1 \cup \bot \]
Example: Sign Analysis (Cont’d)

3) Iterative resolution

\[
\begin{align*}
X_0 &= T \\
X_1 &= + \sqcap X_3 \\
X_2 &= X_1 \sqcap + \\
X_3 &= X_2 \sqcup 1 \\
X_4 &= X_1 \sqcap \bar{1}
\end{align*}
\]

of the form

\[
\overline{X} = \overline{F}(\overline{X}) \quad \text{where} \quad \overline{X} = (\overline{X}_0, \overline{X}_1, \ldots, \overline{X}_4)
\]

Iterate | 0 | 1 | 2 | 3
---------|---|---|---|---
\(X_0\)  | \(\perp\) | \(T\) | \(T\) | \(T\)
\(X_1\)  | \(\perp\) | \(+\) | \(+\) | \(+\)
\(X_2\)  | \(\perp\) | \(+\) | \(+\) | \(+\)
\(X_3\)  | \(\perp\) | \(+\) | \(+\) | \(+\)
\(X_4\)  | \(\perp\) | \(0\) | \(0\) | \(0\)

\[\text{Reference}
\]

Convergence Problem

- The iterates of a monotone transformer \(F \in L \mapsto L\) on a cpo \((L, \sqsubseteq)\) may not converge
- The Interval analysis of

\[
x := 1; \text{ while true do } x := x + 2 \text{ od}
\]

consists in solving

\[
X = [1, 1] \sqcup (X + [2, 2])
\]

Iteratively,

\[
\emptyset, [1, 1], [1, 3], \ldots, [1, 2n + 1], \ldots
\]

Convergence Hypotheses

- We can assume \(L\)
  - to be finite\(^{37}\), or
  - to satisfy the ascending chain condition (ACC) (so \(X^0 = \perp, \ldots, X^{n+1} = F(X^n), \ldots\) which is ascending is finite)
- This is provably less precise than using \(L\) not satisfying the ACC

\[\text{Widening/Narrowing}
\]

\(^{37}\) As in Boolean model-checking
Interval Abstract Domain
The interval abstract domain does not satisfy the ACC

Enforcing Convergence
- The convergence of the iterates
  \[ X^0 = \bot, \ldots, X^{n+1} = F(X^n), \ldots \]
of a monotone transformer \( F \in \mathcal{L} \mapsto \mathcal{L} \) on a cpo \( \langle \mathcal{L}, \sqsubseteq \rangle \) can be forced to converge to an over-approximation of \( \text{lfp} \sqsubseteq F \) using a widening
  - \( X^0 = \bot \)
  - \( X^{n+1} = X^n \uparrow F(X^n) \) if \( F(X^n) \not\sqsubseteq X^n \)
  - \( X^{\ell+1} = X^\ell \) convergence to \( X^\ell \) if \( F(X^\ell) \sqsubseteq X^\ell \)

Definition of the Widening
- The widening overapproximates:
  \[ x \sqsubseteq x \uparrow y \quad y \sqsubseteq x \uparrow y \]
- The widening enforces convergence:
  for all increasing chains
  \[ x^0 \sqsubseteq x^1 \sqsubseteq \ldots, \]
  the increasing chain defined by
  \[ y^0 = x^0, \ldots, y^{i+1} = y^i \uparrow x^{i+1}, \ldots \]
is not strictly increasing.
Example of Widening for the Interval Abstract Domain

- \( \mathcal{L} = \{\perp\} \cup \{[\ell, u] \mid \ell \in \mathbb{Z} \cup \{-\infty\} \land u \in \mathbb{Z} \cup \{+\infty\} \land \ell \leq u\} \)

- The widening extrapolates unstable bounds to infinity:

\[ \bot \triangledown X = X \]
\[ X \triangledown \bot = X \]
\[ [\ell_0, u_0] \triangledown [\ell_1, u_1] = \begin{cases} f \ell_1 < \ell_0 \text{ then } -\infty \text{ else } \ell_0, \\ f u_1 > u_0 \text{ then } +\infty \text{ else } u_0 \end{cases} \]

Soundness and Convergence of the Iterates with Widening

- **Soundness:** convergence to \( X^\ell \) such that \( \overline{F}(X^\ell) \subseteq X^\ell \)

- **Convergence:** \( X^0 = \perp, \ldots, X^{i+1} = X^i \triangledown \overline{F}(X^i), \ldots \)

is not strictly increasing.

Example of Iteration with Widening

- \( x:=1; \) while \( x \leq 1000 \) do \( x:=x+2 \) od

- \( X = ([1, 1] \cup (X + [2, 2])) \triangledown [-\infty, 1000] = \overline{F}(X) \)

- \( X^0 = \perp, \)

- \( X^1 = X^0 \triangledown ((([1, 1] \cup (X^0 + [2, 2])) \triangledown [-\infty, 1000]) = \perp \triangledown [1, 1] = [1, 1] \)

- \( X^2 = X^1 \triangledown ((([1, 1] \cup (X^1 + [2, 2])) \triangledown [-\infty, 1000]) = [1, 1] \triangledown [3, 3] = [1, +\infty] \)

\[ \text{convergence}^{38} \text{ accelerated to } X^\ell = [1, +\infty], \ell = 2 \]

Widening is not Monotone

- **Not** monotone.

- For example \([0, 1] \subseteq [0, 2]\) but \([0, 1] \triangledown [0, 2] = [0, +\infty] \)

\[ \not\subseteq [0, 2] = [0, 2] \triangledown [0, 2] \]

- The limit \( X^\ell \) depends upon the iteration strategy!
Widening Cannot Be 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 $x \triangledown y = \overline{F}(x)$
then: $x \triangledown y = x \triangledown y$ (soundness)
$\subseteq \triangledown \subseteq \triangledown$ (monotony hypothesis)
y $\triangledown y = y$ (termination)
$\Rightarrow x \triangledown y = y$, by antisymmetry!
$\Rightarrow x \triangledown (x \triangledown x) = \overline{F}(x)$ during iteration $\Rightarrow$ convergence cannot be enforced with monotone widening (so widening by finite abstraction is less powerful!)

Convergence Problem, Again

- The decreasing iterates $\overline{F}^n(Y)$, $n \geq 0$ may not converge
- We can assume
  - $\overline{L}$ to be finite, or
  - to satisfy the descending chain condition (DCC)
    (so $X^0 = Y$, $\ldots$, $X^{n+1} = \overline{F}(X^n)$, $\ldots$ which is descending is finite)
- This is provably less precise than using $\overline{L}$ not satisfying the DCC

Improving a Fixpoint Overapproximation

- If $X = \overline{F}(X)$ and $X \subseteq \overline{F}(Y) \subseteq Y$ then
- $X = \overline{F}(X) \subseteq \overline{F}(\overline{F}(Y)) \subseteq \overline{F}(\overline{F}(Y)) \subseteq \overline{F}(Y)$ and hypothesis
- Hence $X = \overline{F}(X) \subseteq \overline{F}(\overline{F}(Y)) \subseteq \overline{F}(\overline{F}(Y)) \subseteq \overline{F}(Y)$ by monotony
- So $X = \overline{F}(X) \subseteq \overline{F}(\overline{F}(Y)) \subseteq \overline{F}(\overline{F}(Y)) \subseteq Y$
- Proving $X = \overline{F}(X) \subseteq \bigcap_{n \geq 0} \overline{F}(Y)$
  by def. $\cap$ and $\overline{F}(0)(Y) = Y$

Enforcing Convergence

- The convergence of the iterates (where $\overline{F}(X^\ell) \subseteq X^\ell$)
  $Y^0 = X^\ell$, $\ldots$, $Y^{n+1} = \overline{F}(Y^n)$, $\ldots$,
of a monotone $\overline{F} \in \overline{L} \subseteq \overline{L}$ on a cpo $\langle \overline{L}, \sqsubseteq \rangle$ can be forced to converge to an over-approximation of $\mathbb{M} \subseteq \overline{F}$
  using a narrowing
- $Y^0 = X^\ell$
- $Y^{n+1} = Y^n \triangle \overline{F}(Y^n)$ if $\overline{F}(Y^n) \neq Y^n$
- $Y^{n+1} = Y^n$ convergence to $Y^n$ if $\overline{F}(Y^n) = Y^n$
Definition of the Narrowing

- A narrowing operator $\Delta$ is such that:
  - $\forall x, y : x \subseteq y \implies x \subseteq x \Delta y \subseteq y$;
  - for all decreasing chains
    \[ x^0 \sqsupseteq x^1 \sqsupseteq \ldots \]
    the decreasing chain defined by
    \[ y^0 = x^0, \ldots, y^{i+1} = y^i \Delta x^{i+1}, \ldots \]
    is not strictly decreasing.

Example of Narrowing for the Interval Abstract Domain

- The narrowing improves infinite bounds only:
  \[ \bot \Delta X = \bot \]
  \[ [\ell_0, u_0] \Delta [\ell_1, u_1] = [(\ell_0 = -\infty \ ? \ \ell_1 : \ell_0) \sqsupseteq, \]
  \[ (u_0 = +\infty \ ? u_1 : u_0)] \]

Example of Iteration with Narrowing

- $x := 1$; while ($x \leq 1000$) do $x := x + 2$ od
- $X = ([1, 1] \sqcup (X + [2, 2])) \sqcap [-\infty, 1000] = \overline{F}(X)$
- $Y^0 = X^\ell = [1, +\infty]$
- $Y^1 = Y^0 \Delta(([1, 1] \sqcup (Y^0 + [2, 2])) \sqcap [-\infty, 1000]) = [1, +\infty] \Delta [1, 1000] = [1, 1000]$
- $\overline{F}(Y^1) = ([1, 1] \sqcup (Y^1 + [2, 2])) \sqcap [-\infty, 1000] = Y^1 = [1, 1000]$
  convergence accelerated to $Y^\eta = [1, +1000]$, $\eta = 1$

Widening/Narrowing May be Too Imprecise

- $x := 1$; while ($x <> 1000$) do $x := x + 2$ od
- $X = ([1, 1] \sqcup (X + [2, 2])) \sqcap [-\infty, 999] \sqcup [1001, +\infty] = [1, 1] \sqcup (X + [2, 2])$
- Iteration with widening: $[1, +\infty]$
- Iteration with narrowing: $[1, +\infty] \Delta ([1, 1] \sqcup ([1, +\infty] + [2, 2])) = [1, +\infty]$ no improvement!
  $\Rightarrow$ Need to widen to threshold 1000!
Improving the Widening: Cutpoints

- Widen only at loop cutpoints (only once around each loop)
- ASTRÉE proceeds by structural induction on the program abstract syntax (so inner loops are stabilized first)

Improving the Widening: Thresholds

- Extrapolate to thresholds in $T \supseteq \{-\infty, +\infty\}$:

  $[\ell_0, u_0] \uplus [\ell_1, u_1] = \begin{cases} [f \, \ell_1 \geq \ell_0 \text{ then } \ell_0, \\ \text{else } \max\{l \in T | l \leq \ell_1\}, \\ f \, u_1 < u_0 \text{ then } u_0 \\ \text{else } \max\{u \in T | u_1 \leq u\}\end{cases}$

- ASTRÉE’s widening thresholds (parametrizable): -1, 0, 1, 2, 3, 4, 5, 17, 32767, 32768, 65535, 65536;

Improving the Widening: Delays

- Do not widen an interval (more generally an abstract predicate) at each iteration, but delay the widening to given numbers of changes
- ASTRÉE’s widening delays (parametrizable): 3, 6, 9, 10, 12, ..., 150, 150 + 1 * Z

Improving the Widening: History-based Widening

- Do not widen/narrow an abstract predicate which was computed for the first time since the last widening/narrowing;
- Otherwise, do not widen/narrow the abstract values of variables which were not “assigned to” \(^{42}\) since the last widening/narrowing.

\(^{42}\) more precisely which did not appear in abstract transformers corresponding to an assignment to these variables.
Example with Widening/Warrowing at Cut-points:

\[
\begin{align*}
\{i: \bot; j: \bot\} \\
\quad i := 1; \\
\{i: [1, \infty); j: [1, \infty)\} \quad & \text{\_\_0\ is the “unini-} \\
\quad \text{\tialized” value} \\
\text{while } (i < 1000) \text{ do} \\
\quad \{i: [1,999]; j: [1, \infty)\} \\
\quad \quad j := 1; \\
\quad \{i: [1, \infty); j: [1, \infty)\} \\
\text{while } (j < i) \text{ do} \\
\quad \{i: [2, \infty); j: [1, 1073741822]\} \\
\quad \quad j := (j + 1) \\
\quad \{i: [2, \infty); j: [2, \infty]\} \\
\text{od;} \\
\quad \{i: [1, \infty); j: [1, \infty)\} \\
\quad i := (i + 1); \\
\{i: [2, \infty); j: [1, \infty)\} \\
\text{od;} \\
\{i: [1000, \infty); j: [1, \infty)\} \\
\end{align*}
\]

Example with History-Based Widening/Narrowing:

\[
\begin{align*}
\{i: \bot; j: \bot\} \\
\quad i := 1; \\
\{i: [1,1000]; j: [1,999]\} \quad & \text{\_\_0\ is the “unini-} \\
\quad \text{\tialized” value} \\
\text{while } (i < 1000) \text{ do} \\
\quad \{i: [1,999]; j: [1,999]\} \\
\quad \quad j := 1; \\
\quad \{i: [1,999]; j: [1,999]\} \\
\text{while } (j < i) \text{ do} \\
\quad \{i: [2,999]; j: [1,998]\} \\
\quad \quad j := (j + 1) \\
\quad \{i: [2,999]; j: [2,999]\} \\
\text{od;} \\
\quad \{i: [1,999]; j: [1,999]\} \\
\quad i := (i + 1); \\
\{i: [2,1000]; j: [1,999]\} \\
\text{od;} \\
\{i: [1000,1000]; j: [1,999]\} \\
\end{align*}
\]

Iteration with Widening/Narrowing (Cont’d)

\[
\begin{align*}
\hat{X}^2 = \hat{X}^1 \vee_F (\hat{X}^0) \\
T = \hat{X}^0 \\
\hat{X}^1 = \hat{X}^0 \Delta_F (\hat{X}^0) \\
gfp F = \text{gfp } F \\
\text{lfp } F = \text{lfp } F \\
\end{align*}
\]
Widening/narrowing are not Dual

- The iteration with **widening** starts from **below** the least fixpoint and stabilizes **above** to a postfixpoint;
- The iteration with **narrowing** starts from **above** the least fixpoint and stabilizes **above**;
- The iteration with **dual widening** starts from **above** the greatest fixpoint and stabilizes **below** to a prefixpoint;
- The iteration with **dual narrowing** starts from **below** the greatest fixpoint and stabilizes **below**;

On Monotony

- The abstract transformer \( \overline{F} \) need not be monotone\(^43\)
- So it can contain \( \nabla \) and \( \Delta \)
- The monotony is required only for concrete transformers (which is the case since predicate transformers are monotonic)

\(^43\) Contrary to what is often assumed for simplicity.

Non-Existence of Finite Abstractions

Let us consider the infinite family of programs parameterized by the mathematical constants \( n_1, n_2 \) \((n_1 \leq n_2)\):

\[
X := n_1; \quad \text{while } X \leq n_2 \text{ do } X := X + 1; \quad \text{od}
\]

- An Interval analysis with widening/narrowing will discover the loop invariant \( X \in [n_1, n_2] \);
- The abstract domain must contain all such intervals to avoid false alarm for all programs in the family;

\[ \Rightarrow \text{ No single finite abstract domain will do for all programs!} \]

References

– Yes, but predicate abstraction with refinement will do (?) for each program in the family (since it is equivalent to a widening)\textsuperscript{44}!

– Indeed no, since:
  - Predicate abstraction is unable to express limits of infinite sequences of predicates;
  - Not all widening proceed by eliminating constraints:
    - A narrowing is necessary anyway in the refinement loop (to avoid infinitely many refinements);
  - Not speaking of costs!


5. 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

\[ C[p] \subseteq P \]

is done in the abstract

\[ C^\mathbb{d}[p] \subseteq P^\mathbb{d} \]

which involves the static analysis of \( p \) that is the effective computation of the abstract semantics

\[ C^\mathbb{d}[p] \]

as formalized by abstract interpretation [23], [24].

References


Example 1: CBMC

- CBMC is a Bounded Model Checker for ANSI-C programs (started at CMU in 1999).
- Allows verifying array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions.
- Aimed for embedded software, also supports dynamic memory allocation using malloc.
- Done by unwinding the loops in the program and passing the resulting equation to a SAT solver.
- Problem (a.o.): does not scale up!
Example 2: Astrée

- Astrée is an abstract interpretation-based static analyzer for ANSI-C programs (started at ENS in 2001).
- Allows verifying array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions.
- Aimed for embedded software, does not support dynamic memory allocation.
- Done by abstracting the reachability fixpoint equations for the program operational semantics.
- Advantage (a.o.): Does scale up!

Design of a Static Analyzer

1. Design of the concrete semantics of programs
2. Definition of properties of programs (collecting semantics)
3. Definition of properties to be verified (specification)
4. Choice of abstractions and their combinations
5. Design of the abstract semantics of programs (iterator and abstract properties)
6. Design of the abstract semantics overapproximation (iteration acceleration)
7. Design of the abstract specification verification algorithm (proof)

Model versus Property and Program versus Language-based Abstraction

Property-based Abstraction

- Property-based abstraction over approximate the collecting semantics in the abstract
  - $C[p] = \{S[p]\} \in P$ collecting semantics
  - $(P, \subseteq) \xrightarrow{\gamma_a} (P^\#, \subseteq^\#)$ abstraction
  - $C^\#[p] \in P^\#$ abstract semantics
  - $C[p] \subseteq \gamma(C^\#[p])$ soundness
  $\Rightarrow$ an abstract proof $(C^\#[p] \subseteq^\# P^\#)$ is valid in the concrete $(C[p] \subseteq \gamma(P^\#))$. 
Model-based Abstraction

- Let $(\ell, \tau)$ be a transition system model of a software or hardware system $p \in P$ (so that $S[p] \triangleq \gamma_p((\ell, \tau))$).
- A model-based abstraction is an abstract transition system $(\ell^\sharp, \tau^\sharp)$ which over-approximates $(\ell, \tau)$ (so that, up to concretization, $\ell \subseteq \ell^\sharp$ and $\tau \subseteq \tau^\sharp$).
- The set of reachable abstract states for $(\ell^\sharp, \tau^\sharp)$ over-approximate the reachable concrete states of $(\ell, \tau)$.
- 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 [26] Polyhedra [25]

Program-based versus Language-based Abstraction

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

- An abstraction specific to a given program can always be refined to be complete using a finite abstract domain [27].
- This is impossible in general for a language-based abstraction for which infinite abstract domains have been shown to always produce better results [28].
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 Alarms (Cont’d)

- The experience of ASTRÉE (www.astree.ens.fr, [29]) shows that it is possible to design precise language-based abstractions which produce no false alarm on a well defined families of programs.
- Nevertheless, by indecidability, the analyzer will produce false alarms on infinitely many programs (which can even be generated automatically).

References


45 Synchronous, time-triggered, real-time, safety critical, embedded software written or automatically generated in the C programming language for Astrée.
Abstract Domains

An Abstract Domain

An abstract domain defines
- The computer representation of abstract properties (corresponding to given concrete properties)
- The abstract operations requested by the iterator (including lattice operations $\sqsubseteq, \ldots$, operations involved in the abstract transformer $F$, convergence acceleration $\nabla$, etc)

Design of Abstractions

An Abstract Domain in Astrée

```plaintext
module type INTEGER = sig
  type t
  val zero: t
  val one: t
  val is_zero: t->bool
  val max: t->t->t
  val min: t->t->t
  val add: t->t->t
  val sub: t->t->t
  val mul: t->t->t
  val div: t->t->t
  val rem: t->t->t
  val neg: t->t
  val sgn: t->int
  val abs: t->t
  val compare: t->t->int
  val print: Format.format->t->unit
  val succ: t->t
  val pred: t->t->bool
  val widening_sequence: t list
  val quick_widening_sequence: t list
  val shift_left: t->int->t
  val shift_right: t->int->t
  val shift_right_logical: t->int->t
  val to_int: t->int
  val of_int: int->t
  val logand: t->t->t
  val logor: t->t->t
  val logxor: t->t->t
  val lognot: t->t
  val to_string: t->string
end
```

More precisely, interface with the iterator.
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 [30] but because of undecidability, they do not terminate in general.

References

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 [31].

References

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.

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.
– These abstractions could hardly be encoded efficiently using a universal representation of program properties as found in theorem provers, proof assistants or model-checkers.
6. The ASTRÊE Static Analyzer

www.astree.ens.fr/ Nov. 2001—Nov. 2007

Project Members

Bruno Blanchet 47  Patrick Cousot  Radhia Cousot  Jérôme Feret

Laurent Mauborgne  Antoine Miné  David Monniaux 48  Xavier Rival

Programs Analyzed by ASTRÊE

Programs analysed by ASTRÊE

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

– C programs:
  - with
    - basic numeric datatypes, structures and arrays
    - pointers (including on functions),
    - floating point computations
    - tests, loops and function calls
    - limited branching (forward goto, break, continue)
- with (cont’d)
  - union [Min06a] NEW
  - pointer arithmetics & casts [Min06a] NEW
- without
  - dynamic memory allocation
  - recursive function calls
  - unstructured/backward branching
  - conflicting side effects
  - C libraries, system calls (parallelism)

Such limitations are quite common for embedded safety-critical software.

-- Challenging aspects

- **Size:** 100/1000 kLOC, 10/150 000 global variables
- **Floating point computations**
  - including interconnected networks of filters, non linear control with feedback, interpolations...
- **Interdependencies among variables:**
  - Stability of computations should be established
  - Complex relations should be inferred among numerical and boolean data
  - Very long data paths from input to outputs

---

**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 output variables;
    __ASTREE_wait_for_clock ();
end loop
```

Task scheduling is static:
- **Requirements:** the only interrupts are clock ticks;
- **Execution time of loop body less than a clock tick,**
  as verified by the aiT WCET Analyzers [FHL+01].

---

**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, execution stops on first runtime error \(^{49}\))

\(^{49}\) semantics of C unclear after an error, equivalent if no alarm
Different Classes of Run-time Errors

1. **Errors terminating the execution**. **ASTRÉE** warns and continues by taking into account only the executions that did not trigger the error.

2. **Errors not terminating the execution with predictable outcome**. **ASTRÉE** warns and continues with worst-case assumptions.

3. **Errors not terminating the execution with unpredictable outcome**. **ASTRÉE** warns and continues by taking into account only the executions that did not trigger the error.

⇒ **ASTRÉE** is sound with respect to C **standard**, unsound with respect to C **implementation**, unless no false alarm.

---

Specification Proved by **ASTRÉE**

**Trace semantics**

- From this small-step semantics we derive a **discrete-time complete trace semantics**;
- This trace semantics is abstracted into many different abstract properties as implemented by various abstract domains defining compat finite representations of specifica properties;
- **ASTRÉE** computes a **weak reduced product** for these abstractions.

---

Implicit Specification: Absence of Runtime Errors

- No violation of the **norm of C** (e.g. array index out of bounds, division by zero)
- No implementation-specific **undefined behaviors** (e.g. maximum short integer is 32767, NaN)
- 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).
Characteristics of the ASTRÉE Analyzer (Cont’d)

**Sound:** – ASTRÉE is a **bug eradicator**: finds all bugs in a well-defined class (runtime errors)
– ASTRÉE is **not** a **bug hunter**: finding some bugs in a well-defined class (e.g. by **bug pattern detection** like FindBugs™, PREfast or PMD)
– ASTRÉE is **exhaustive**: covers the whole state space (**≠** MAGIC, CBMC)
– ASTRÉE is **comprehensive**: never omits potential errors (**≠** UNO, CMC from coverity.com) or sort most probable ones to avoid overwhelming messages (**≠** Splint)

**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), or PREfast (annotate functions with intended use)

**Multiabstraction:** uses many numerical/symbolic abstract domains (**≠** symbolic constraints in Bane or the canonical abstraction of TVLA)

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

**Efficient:** always terminate (**≠** counterexample-driven automatic abstraction refinement BLAST, SLAM)
Characteristics of the **ASTRÉE** Analyzer (Cont’d)

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

**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)

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

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

---

**The ASTRÉE Abstract Interpreter**

---

**Abstract Semantics**

- **Reachable states** for the concrete trace operational semantics (with partial history)
- **Volatile environment** is specified by a trusted configuration file.

**Requirements:**

- **Soundness:** absolutely essential
- **Precision:** few or no false alarm \(^{15}\) (full certification)
- **Efficiency:** rapid analyses and fixes during development

\(^{15}\) Potential runtime error signaled by the analyzer due to overapproximation but impossible in any actual program run.
**ASTRÉE's Architecture**

- C-preprocessor
- C99 parser
- Link editor
- Intermediate code generation and typing
- Constant propagation and simplification
- Local and global dependence analysis
- Abstract Interpreter

**The Iterator**

- Flow through all possible program executions, following the program syntactic structure
- Example: loops

**The Abstract Interpreter**

- Iterator
- Trace partitionning
- Memory model and aliases
- Reduced product of numerical abstract domains
- Intervals Octagons Decision trees...

**Handling Functions**

- No recursion ⇒ functions can be handled without any abstraction
- we get a flow and context sensitive static analysis using an abstract stack for control/parameters/local variables (isomorphic to the concrete execution stack)
  ⇒ the analysis is extremely precise.
Handling Simple Variables

- In a given context, the abstraction of variable properties $\rho(\mathcal{X} \mapsto \mathcal{V})$ for fixed variables in $\mathcal{X}$ is given
- everywhere by non-relational abstract domains (abstracting $\rho(\mathcal{V})$ by bitstring, set of values, simple congruence, interval, etc);
- in chosen contexts, as a component of a relational abstract domains (octagons, etc).
and subject to widening/narrowing and interdomain reductions.
$\mathcal{X} \mapsto a(\rho(\mathcal{V}))$ is represented by a balanced tree.
⇒ the parametrization of the analysis allows for a fine tuning of the cost/precision balance.

Handling Arrays

The array $T$ of size $n$ can be handled
- as a collection of separate variables ($T[0], T[1], \ldots, T[n-1] \in \mathcal{X}$) handled individually as simple variables;
- as a single smashed variable $T \in \mathcal{X}$ which concrete value include all possible values of $T[0], T[1], \ldots, \text{ and } T[n-1]$;
⇒ the parameterized analysis can be extremely precise when needed.

Handling Pointers

- No dynamic memory allocation ⇒ the heap and aliases can be handled without any abstraction (using an abstract heap isomorphic to the concrete heap);
- A pointer is a basis plus an integer offset (abstracted separately by a set of bases $\subseteq \mathcal{X}$ and an auxiliary integer variable in $\mathcal{X}$ for the offset).
⇒ the analysis is extremely precise (but maybe for pointers to smashed array elements).

Memory Model

The union type, pointer arithmetics and pointer transtyping is handled by allowing aliasing at the byte level [32]:

```
union {
    struct { uint8 a[l, ah, b[l, bh]; } b;
    struct { uint16 ax, bx; } w;
} r;
```

- A box (auxiliary variable) in $\mathcal{X}$ for each offset and each scalar type
- intersection semantics for overlapping boxes

Reference

Iteration Refinement: Loop Unrolling

Principle:
- Semantically equivalent to:
  \[
  \text{while (B) \{ C \}} \quad \Rightarrow \quad \text{if (B) \{ C \}}; \text{ while (B) \{ C \}}
  \]
- More precise in the abstract: less concrete execution paths are merged in the abstract.

Application:
- Isolate the initialization phase in a loop (e.g. first iteration).

---

Control Partitioning for Case Analysis

Principle:

- 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 \(-100 \leq x \leq 100\) ...
    while ((i < 3) && (x >= t[i+1])) {
        i = i + 1;
    }
    r = (x - t[i]) * c[i] + d[i];
}
```

---

Control Partitioning:

Control point partitionning:

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

---

Iteration Refinement: Trace Partitioning

Principle:
- Semantically equivalent to:
  \[
  \text{if (B) \{ C1 \} else \{ C2 \}; C3}
  \]
  \[
  \Downarrow
  \]
  \[
  \text{if (B) \{ C1; C3 \} else \{ C2; C3 \}};
  \]
- More precise in the abstract: concrete execution paths are merged later.

Application:
- Isolate the initialization phase in a loop (e.g. first iteration).

---

Convergence Accelerator: Widening

Principle:

- Brute-force widening:

- Widening with thresholds:

Examples:
- 1., 10., 100., 1000., etc. for floating-point variables;
- maximal values of data types;
- syntactic program constants, etc.
Problem: Fixpoint Stabilization for Floating-point
- Mathematically, we look for an abstract invariant inv such that $F(\text{inv}) \subseteq \text{inv}$.
- Unfortunately, abstract computation uses floating-point and incurs rounding: maybe $F_\varepsilon(\text{inv}) \not\subseteq \text{inv}$!

Solution:
- Widen inv to $\text{inv}_{\varepsilon'}$ with the hope to jump into a stable zone of $F_\varepsilon$.
- Works if $F$ has some attractiveness property that fights against rounding errors (otherwise iteration goes on).
- $\varepsilon'$ is an analysis parameter.

Examples of Abstractions in ASTRÉE

General-Purpose Abstract Domains: Intervals and Octagons

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

Octagons
- $1 \leq x \leq 9$
- $x + y \leq 77$
- $1 \leq y \leq 20$
- $x - y \leq 04$

Difficulties: many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [CC77a, Min01, Min04a]

Floating-point linearization [Min04a, Min04b]
- Approximate arbitrary expressions in the form $[a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)$
- Example:
  $Z = X - (0.25 \times X)$ is linearized as $Z = ([0.749 \ldots, 0.750 \ldots] \times x) + (2.35 \times 10^{-38} \times [-1, 1])$
- Allows simplification even in the interval domain
  - If $X \in [-1, 1]$, we get $|Z| \leq 0.750 \ldots$ instead of $|Z| \leq 1.25 \ldots$
- Allows using a relational abstract domain (octagons)
- Example of good compromise between cost and precision
Symbolic abstract domain \cite{Min04a, Min04b}

- Interval analysis: if \(x \in [a, b]\) and \(y \in [c, d]\) then \(x - y \in [a - d, b - c]\) 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);

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 (1B) {
            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 leaves.

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.

Filter Example \cite{Fer04}
Arithmetic-geometric progressions\textsuperscript{56} [Fer05]

- Abstract domain: \((\mathbb{R}^+)^5\)
- Concretization:
  \[
  \gamma \in (\mathbb{R}^+)^5 \mapsto \varphi(N \mapsto R)
  \]

\[
\gamma(M, a, b, a', b') = \{ f \mid \forall k \in \mathbb{N} : |f(k)| \leq (\lambda x . a x + b \circ (\lambda x . a' x + b')^k) (M) \}
\]

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

\textsuperscript{56} here in R

---

(Automatic) Parameterization

- All abstract domains of ASTR\`EE are parameterized, e.g.
  - variable packing for octagones and decision trees,
  - partition/merge program points,
  - loop unrollings,
  - thresholds in widenings, \ldots;
- 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).

---

Example of Abstract Domain in ASTR\`EE: The Arithmetic-Geometric Progression Abstract Domain

Arithmetic-Geometric Progressions: Motivating Example

```c
% 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();
  }
}
```

---

Running example (in \( \mathbb{R} \))

1: \( X := 0; k := 0; \)
2: \( \text{while } (k < 1000) \{ \)
3: \( \text{if } (?) \{ X \in [-10; 10]; \}
4: \( X := X/3; \)
5: \( X := 3 \times X; \)
6: \( k := k + 1; \)
7: \}

---

Objective

- In automatically generated programs using floating point arithmetics, some computations may diverge because of rounding errors.
- To prove the absence of floating point number overflows, we use non polynomial constraints:
  - we bound rounding errors at each loop iteration by a linear combination of the loop inputs;
  - we get bounds on the values depending exponentially on the program execution time.
- The abstract domain is both precise (no false alarm) and efficient (linear in memory / \( \pi n(n) \) in time).

---

Interval analysis: first loop iteration

1: \( X := 0; k := 0; \)
2: \( \text{while } (k < 1000) \{ \)
3: \( \text{if } (?) \{ X \in [-10; 10]; \}
4: \( X := X/3; \)
5: \( X := 3 \times X; \)
6: \( k := k + 1; \)
7: \}

---
Interval analysis: Invariant

1: \( X := 0; k := 0; \)
\( X = 0 \)
2: while \( (k < 1000) \) {
   \( |X| \leq 10 \)
3: \( \) if (?) \{ \( X \in [-10; 10] \) \};
   \( |X| \leq 10 \)
4: \( X := X/3; \)
   \( |X| \leq \frac{10}{3} \)
5: \( X := 3 \times X; \)
   \( |X| \leq 10 \)
6: \( k := k + 1; \)
7: \} \( |X| \leq 10 \)

Including rounding errors [Miné–ESOP’04]

1: \( X := 0; k := 0; \)
2: while \( (k < 1000) \) {
3: \( \) if (?) \{ \( X \in [-10; 10] \) \};
4: \( X := X/3 + [-\varepsilon_1; \varepsilon_1] \times X + [-\varepsilon_2; \varepsilon_2]; \)
5: \( X := 3 \times X + [-\varepsilon_3; \varepsilon_3] \times X + [-\varepsilon_4; \varepsilon_4]; \)
6: \( k := k + 1; \)
7: \} \( X = 0, k = 0 \)

The constants \( \varepsilon_1, \varepsilon_2, \varepsilon_3 \), and \( \varepsilon_4 \geq 0 \) are computed by other domains.

Interval analysis

Let \( M \geq 0 \) be a bound:

1: \( X := 0; k := 0; \)
\( X = 0 \)
2: while \( (k < 1000) \) {
   \( |X| \leq M \)
3: \( \) if (?) \{ \( X \in [-10; 10] \) \};
   \( |X| \leq \max(M, 10) \)
4: \( X := X/3 + [-\varepsilon_1; \varepsilon_1] \times X + [-\varepsilon_2; \varepsilon_2]; \)
   \( |X| \leq (\varepsilon_1 + \frac{1}{3}) \times \max(M, 10) + \varepsilon_2 \)
5: \( X := 3 \times X + [-\varepsilon_3; \varepsilon_3] \times X + [-\varepsilon_4; \varepsilon_4]; \)
   \( |X| \leq (1 + a) \times \max(M, 10) + b \)
6: \( k := k + 1; \)
7: \}

Ari.-geo. analysis: first iteration

1: \( X := 0; k := 0; \)
\( X = 0, k = 0 \)
2: while \( (k < 1000) \) {
3: \( \) if (?) \{ \( X \in [-10; 10] \) \};
   \( |X| \leq 10 \)
4: \( X := X/3 + [-\varepsilon_1; \varepsilon_1] \times X + [-\varepsilon_2; \varepsilon_2]; \)
   \( |X| \leq \left\{ v \mapsto \left( \frac{1}{3} + \varepsilon_2 \right) \times v + \varepsilon_2 \right\}(10) \)
5: \( X := 3 \times X + [-\varepsilon_3; \varepsilon_3] \times X + [-\varepsilon_4; \varepsilon_4]; \)
   \( |X| \leq f^{(1)}(10) \)
6: \( k := k + 1; \)
7: \}
\( |X| \leq f^{(k)}(10), k = 1 \)

with \( f = \left\{ v \mapsto \left( 1 + 3 \times \varepsilon_1 + \frac{\varepsilon_3}{3} + \varepsilon_1 \times \varepsilon_2 \right) \times v + \varepsilon_2 \times (3 + \varepsilon_3) + \varepsilon_4 \right\}. \)
Ari.-geo. analysis: Invariant

1: \( X := 0; k := 0 \)
\[ X = 0, k = 0 \]

2: \( \text{while } (k < 1000) \{ \)
\[ X = 0, \]
\[ k = 0 \]

3: \( \text{if } (?) \{ X \in [-10;10]; \}
\[ |X| \leq f^{(b)}(10) \]

4: \( X := X/3 + [-\varepsilon_1;\varepsilon_1].X + [-\varepsilon_2;\varepsilon_2]; \)
\[ |X| \leq (\frac{1}{3} + \varepsilon_1) \times (f^{(b)}(10)) + \varepsilon_2 \]

5: \( X := 3 \times X + [-\varepsilon_3;\varepsilon_3].X + [-\varepsilon_4;\varepsilon_4]; \)
\[ |X| \leq f\left(f^{(b)}(10)\right) \]

6: \( k := k + 1; \)
\[ |X| \leq f^{(k)}(10) \]

7: \( \} \)
\[ |X| \leq f^{(1000)}(10) \]

with \( f = [v \rightarrow (1 + 3 \times \varepsilon_1 + \frac{\varepsilon_2}{3} + \varepsilon_1 \times \varepsilon_3) \times v + \varepsilon_2 \times (3 + \varepsilon_3) + \varepsilon_4]. \)

Arithmetic-geometric progressions (in \( \mathbb{R} \))

An arithmetic-geometric progression is a 5-tuple in \( (\mathbb{R}^+)^5 \).

An arithmetic-geometric progression denotes a function in \( \mathbb{N} \rightarrow \mathbb{R}^+ \):

\[ \beta_{\mathbb{R}}(M, a, b, a', b')(k) \triangleq [v \mapsto a \times v + b] \left( [v \mapsto a' \times v + b']^{(k)}(M) \right) \]

Thus,

\( k \) is the loop counter;
\( M \) is an initial value;
\( [v \mapsto a \times v + b] \) describes the current iteration;
\( [v \mapsto a' \times v + b']^{(k)} \) describes the first \( k \) iterations.

A concretization \( \gamma_{\mathbb{R}} \) maps each element \( d \in (\mathbb{R}^+)^5 \) to a set \( \gamma_{\mathbb{R}}(d) \subseteq (\mathbb{N} \rightarrow \mathbb{R}^+) \) defined as:

\[ \{ f \mid \forall k \in \mathbb{N}, |f(k)| \leq \beta_{\mathbb{R}}(d)(k) \} \]

Monotonicity

Let \( d = (M, a, b, a', b') \) and \( d = (M, a, b, a', b') \) be two arithmetic-geometric progressions.

If:

\( - M \leq M, \)
\( - a \leq a, a' \leq a', \)
\( - b \leq b, b' \leq b'. \)

Then:

\( \forall k \in \mathbb{N}, \beta_{\mathbb{R}}(d)(k) \leq \beta_{\mathbb{R}}(d')(k). \)
Disjunction
Let \( d = (M, a, b, a', b') \) and \( d' = (M, a, b, a', b') \) be two arithmetic-geometric progressions.
We define:
\[
d \sqcup d' \triangleq (M, a, b, a', b')
\]
where:
\[
M \triangleq \max(M, M), \\
a \triangleq \max(a, a), \\
a' \triangleq \max(a', a'), \\
b \triangleq \max(b, b), \\
b' \triangleq \max(b', b'),
\]
For any \( k \in \mathbb{N} \), \( \beta_R(d \sqcup d')(k) \geq \max(\beta_R(d(k)), \beta_R(d')(k)) \).

Assignment
We have:
\[
\beta_R(M, a, b, a', b')(k) = a \times (M + b' \times k) + b \quad \text{when } a' = 1 \\
\beta_R(M, a, b, a', b')(k) = a \times (a')^k \times \left(M - \frac{b'}{1-a'}\right) + b \quad \text{when } a' \neq 1.
\]
Thus:
1. for any \( a, a', M, b, b', \lambda \in \mathbb{R}^+ \),
\[
\lambda \times (\beta_R(M, a, b, a', b')(k)) = \beta_R(\lambda \times M, a, b, M \times a', a, \lambda \times b')(k);
\]
2. for any \( a, a', M, b, b', M, b, b' \in \mathbb{R}^+ \), for any \( k \in \mathbb{N} \),
\[
\beta_R(M, a, b, a', b')(k) + \beta_R(M, a, b, a', b')(k) = \beta_R(M + M, a + b, a', b')(k).
\]

Conjunction
Let \( d \) and \( d' \) be two arithmetic-geometric progressions.

1. If \( d \) and \( d' \) are comparable (component-wise), we take the smaller one:
\[
d \sqcap d' \triangleq \inf_{\leq} \{d; d'\}.
\]
2. Otherwise, we use a parametric strategy:
\[
d \sqcap d' \in \{d; d'\}.
\]
For any \( k \in \mathbb{N} \), \( \beta_R(d \sqcap d')(k) \geq \min(\beta_R(d(k)), \beta_R(d')(k)) \).

Projection I/II
\[
\beta_R(M, a, b, a', b')(k) = a \times (M + b' \times k) + b \quad \text{when } a' = 1 \\
\beta_R(M, a, b, a', b')(k) = a \times (a')^k \times \left(M - \frac{b'}{1-a'}\right) + b \quad \text{when } a' \neq 1.
\]
Thus, for any \( d \in (\mathbb{R}^+)^5 \), the function \( [k \mapsto \beta_R(d)(k)] \) is:
- either monotonic,
- or anti-monotonic.
Projection II/II

Let \( d \in (\mathbb{R}^+)^5 \) and \( k_{\text{max}} \in \mathbb{N} \).

\[ \text{bound}(d, k_{\text{max}}) \triangleq \max(\beta_{\mathbb{R}}(d)(0), \beta_{\mathbb{R}}(d)(k_{\text{max}})) \]

For any \( k \in \mathbb{N} \) such that \( 0 \leq k \leq k_{\text{max}} \)

\[ \beta(d)(k) \leq \text{bound}(d, k_{\text{max}}). \]

About floating point numbers

Floating point numbers occur:

1. in the concrete semantics:
   Floating point expressions are translated into real expressions with interval coefficients [Miné—ESOP’04].
   So the abstract domains, can handle real numbers.

2. in the abstract domain implementation:
   For efficiency purpose, each real primitive is implemented in floating point arithmetics: each real is safely approximated by an interval with floating point number bounds.

Incrementing the loop counter

We integrate the current iteration into the first \( k \) iterations:
- the first \( k + 1 \) iterations are chosen as the worst case among the first \( k \) iterations and the current iteration;
- the current iteration is reset.

Thus:

\[ \text{next}_{\mathbb{R}}(M, a, b, a', b') \triangleq (M, 1, 0, \max(a, a'), \max(b, b')). \]

For any \( k \in \mathbb{N}, d \in (\mathbb{R}^+)^5 \), \( \beta_{\mathbb{R}}(d)(k) \leq \beta_{\mathbb{R}}(\text{next}_{\mathbb{R}}(d))(k + 1). \)

Using ASTRÉE
Example application

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

- C program, automatically generated from a proprietary high-level specification (à la Simulink/SCADE)
- A340 family: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays, now × 2
- A380: × 3/7

Digital Fly-by-Wire Avionics

- V1\textsuperscript{58}, 132,000 lines, 75,000 LOCs after preprocessing
- **Comparative results** (commercial software):
  - 4,200 (false?) alarms, 3.5 days;
- **Our results**:
  - 0 alarms,
  - 40mn on 2.8 GHz PC, 300 Megabytes
  - → A world première in Nov. 2003!

\textsuperscript{57} The electrical flight control system is placed between the pilot's controls (sidesticks, rudder pedals) and the control surfaces of the aircraft, whose movement they control and monitor.

\textsuperscript{58} “Flight Control and Guidance Unit” (FCGU) running on the “Flight Control Primary Computers” (PCPC). The three primary computers (PCPC) and two secondary computers (PSCC) which form the A340 and A380 electrical flight control system are placed between the pilot's controls (sidesticks, rudder pedals) and the control surfaces of the aircraft, whose movement they control and monitor.
(Airbus A380 Primary Flight Control Software)

- Now at 1,000,000 lines!
- 0 alarms (Nov. 2004), after some additional parametrization and simple abstract domains developments
  34h,
  8 Gigabyte
  → A world grand première!

The main loop invariant for the A340 V1
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.

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 \(^{59}\);
- Inexpressivity i.e. indispensable local inductive invariant are inexpressible in the abstract → add a new abstract domain to the reduced product (e.g. filters).

\(^{59}\) 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.

7. 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 [34].

References


8. Bibliography


