The ASTRÉE Analyzer

Xavier RIVAL

rival@di.ens.fr
Certification of embedded softwares

- Safety critical applications
  - avionic softwares but also automotive, space...
  - synchronous

- Properties to prove to guarantee safety:
  - absence of runtime errors
    - no crash, no violation of application specific constraints
  - synchronous requirement, i.e., time constraint
    - critical sections should take a bounded amount of time
    - i.e., the software must be responsive
    - recursion is forbidden
  - resource usage
    - no dynamic memory allocation
    - stack usage

The ASTRÉE Analyzer – p.2/36
What ASTRÉE is?

- A static analyzer
  - Inputs a C program (some restrictions, see later)
  - User-defined assumptions about the input values (ranges)
  - Computes an over-approximation of the reachable states
  - Produces an alarm when an operation is not proved safe
    - **Sound**: detects all errors
    - **Incomplete**: false alarms are possible
      - Detecting all errors exactly: undecidable!

- Developed in the École Normale Supérieure (Paris, France)
  Joint work with B. BLANCHET, P. COUSOT, R. COUSOT, J. FERET, L. MAUBORGNE, A. MINÉ, D. MONNIAUX

- ASTRÉE addresses:
  - Runtime errors
  - Non specified behaviors
What ASTRÉE is not?

- **Testing:**
  - ASTRÉE covers all executions, hence is **sound**; testing is **unsound**
  - ASTRÉE does sound approximation, hence **incompleteness**
  - **Cost** considerations:
    - Weeks of heavy test processes vs. a few hours of computation

- **Model checking:**
  - ASTRÉE is designed to implement the **C semantics**
    - No separate model extraction phase
  - ASTRÉE uses a **quasi-infinite predicate set**
  - ASTRÉE lagging for **automatic refinement** (work in progress)

- **User-assisted theorem proving:**
  - ASTRÉE is automatic with **little to no user interaction**
    - Cost efficient!
  - But ASTRÉE needs to infer indecidable properties; hence is **incomplete**
Development of ASTRÉE

- **Fall 2001:** Request for a precise and fast analyzer (Airbus)
  ASTRÉE project started:
  - **Scalability** = main goal
    (data structures, algorithms)
  - **Simple non-relational domains:** intervals + first refinements
  - Analysis of a 10 kLOCs software, few alarms (2002)

- **From 2003,** analysis of industrial softwares:
  - **Inspection of alarms:**
    ⇒ True error ?
    ⇒ Imprecision in the analysis ? if yes, **find origin of imprecisions**
  - Improve precision, with new abstractions,
    solving imprecisions, but preserving scalability
  - **Successful** analysis of two families of industrial applications

- **Commercial diffusion,** since 2009, by Absint
A Specialized Analyzer

- Specialization with respect to some families of embedded softwares:
  Synchronous, real-time programs:

```
declare and initialize state variables;
loop forever
  read volatile input variables,
  compute output and state variables,
  write to volatile output variables;
wait for next clock tick (10 ms)
end loop
```

- Properties to establish: Absence of runtime errors; broad definition
  - No fatal error as defined by the semantics of the C language
    - e.g., division by 0, out of segment access
  - No overflow in integer or floating point computations
  - User defined properties: e.g., no NaN!
  - Architecture dependant properties (data-type sizes)
Specific Features of ASTRÉE

- **Simplifications:**
  - Not all C: no malloc, no recursion
  - Mostly static data; a few local variables

- **Issues:**
  - Size of programs to analyze: $> 100$ kLOC, $> 10,000$ variables
    - More typically 1 MLOC, $> 50,000$ variables
  - Floating point computation should be analyzed precisely
    - DSP filetering, non linear control, retroactions, interpolation functions
  - Intricate dependencies between variables:
    - Stability of computation should be established
    - Relations between numerical and boolean data to infer
    - Long sequences of dependences between inputs and outputs
    - e.g., slicing ineffective
Outline

- Context

✓ Structure of the Analyzer
  - Abstract Domain
  - Results Overview
Principle of the Analyzer

Computation of an over-approximation of reachable states

- **Model of C**: operational semantics
  \[ [P] = \text{set of executions (aka, traces) of } P \]
  - C-99 standard
  - IEEE 754-1985 norm (floating point computations)
  - Assumptions about the target architecture and the area of application:
    - size of integer data-types
    - initialization of static variables
    - ranges for inputs; duration of an execution

- **Abstraction** = approximation defined by an abstract domain

- **Systematic derivation of a sound and automatic analyzer**

- **Certification of a piece of code, in two automatic stages:**
  1. Computation of an approximation (99.9% of the work...)
  2. Checking of safety conditions
Abstraction

- **ASTRÉE computes** an invariant $I \in D^\#$ ($D^\#$: our abstract domain):
  - For each control point $\ell$
  - For each context $\kappa$ (e.g., calling stack)
  $\Rightarrow$ an approximation $I(\ell, \kappa) \in D^\#_M$ of a set of stores
  $D^\#_M$ expresses a (usually infinite) set of predicates

- **Soundness**:
  - $I$ should account for all executions in $P$
  - Meaning of $I$: $\gamma(I)$
  - Soundness statement: $[P] \subseteq \gamma(I)$
    where $[P]$ is a formal concrete semantics

- **Next question**: How to compute $I$?
  Definition of a generic interpretation scheme
Abstract Interpretation: Computing Invariants

Principle: run all computations in a unique abstract computation

- Analysis of an atomic statement $x = e$:
  - Use an abstract transfer function $\text{assign}(x = e) : D^\#_M \rightarrow D^\#_M$
  - $D^\#_M$ manages addition and removal of constraints
  - Soundness: this operation should over-approximate concrete executions

- Denotational form engine:
  - For each concrete, elementary step $F$, a sound approximation computed by an abstract store transformer $F^\#$:
    \[
    \forall \rho \in \mathbb{M}, \ d^\# \in D^\#_M, \ \rho \in \gamma(d^\#) \implies F(\rho) \subseteq \gamma(F^\#(d^\#))
    \]
  - $D^\#_M$ provides such sound transfer functions: $\text{guard}$, ...

- Control flow joins (after a conditional):
  - $D^\#_M$ provides a sound approximation $\sqcup$ of $\sqcup$
Analysis of Loops

- **Loops** should require *infinitely many iterations*
- **Solution**: use a **widening** operator
  - A sound approximation $\nabla$ of $\sqcup$;
  - Termination is enforced by the widening properties

```
while (...) { ... }
```

Program

Iterative invariant computation

- Memorized abstract invariants
- Propagated abstract invariants
Widening Operator

- Definition:
  - sound approximation of join: \( \forall x, y \in D^\#_M, \gamma(x) \cup \gamma(y) \subseteq x \nabla y \)
  - termination: for any increasing sequence \((y_n)_{n\in\mathbb{N}}\) of elements of \(D^\#_M\), the sequence \((x_n)_{n\in\mathbb{N}}\) of elements of \(D^\#_M\) defined by \(x_0 = y_0\) and \(\forall n \in \mathbb{N}, x_{n+1} = x_n \nabla y_n\) is not strictly increasing

- Widening:
  - Example, with intervals:
    \(I_0 : 0 \leq x \leq 10; \quad I_1 : 1 \leq x \leq 11; \quad I_0 \nabla I_1 : 0 \leq x\)
  - In practice: remove unstable constraints
    Convergence: ensured by the finiteness of the number of constraints at the first iteration

Though: analysis may still involve unbounded number of predicates
domain may still be infinite
widening chains are still unbounded
Widening Improvements

- “Unrolling” of the first iterations (better precision)
  - Idea: postpone widening to iteration 2 or 3
  - More precision in the first abstract join operations

- Thresholds:
  - Principle: when $x < 4$ is not stable, $x < 8$ may be stable
  - Threshold widening: ordered families of constraints $T$
    \[ \n\n\]
    \[ \n\n\]
    \[ \n\n\]
  - Implementation based on strategies such as:
    - if $x == 4$ appears in the code, automatically add step 4 for $x$ in $\n$

- Note:
  - Better precision $\Rightarrow$ smaller state space
  - $\Rightarrow$ shorter widening chains

  A precise analysis is NOT incompatible with efficiency

  Practical experience: imprecise analyses with many alarms are very slow
Outline

- Context
- Structure of the Analyzer

✓ Abstract Domain
✓ Generalities
  - A relational numerical domain: Octagons
  - A symbolic domain: Trace partitioning
  - Other domains
- Results Overview
Abstract Domain

- Abstractions of sets of states (e.g., stores)
- Usual transfer functions:
  - Guard: for conditions (if, while, assert, ...)
  - Assign
  - Variable creation, disposal...
- A widening operator, a lower upper bound
- Ordering: usually a sound approximation of the concrete ordering
  - If $\text{inf}^\#(x^\#, y^\#)$ returns TRUE, then $\gamma(x^\#) \subseteq \gamma(y^\#)$
  - But the test may fail! (decidability!)
- Support for communication with other abstract domains:
  - Dozens of domains implemented in ASTRÉE...
  - Need for information communication across domains
    Different domains typically establish complimentary properties
A Non-relational Abstraction: Intervals

• Simplification: contrived memory model:
  ♦ 1 abstract cell ≡ 1 or several concrete cells (\textit{smashed arrays})
  ♦ Information about pointers: points-to

• Interval-based approximation:
  ♦ Constraints $a \leq x \leq b$ ($x$: abstract cell)
  ♦ Not an expensive analysis
  ♦ Implementation: \textit{sound} approximation of floating point computations
    Should be ensured for \textit{all} numerical abstractions

• \textsc{Astrée}: started with an interval analysis
  ♦ Enough to express the absence of runtime errors
    Array bounds, overflows, division by 0
  ♦ Not expressive enough to \textit{infer/prove} precise invariants

• Next slides: imprecisions + new abstract domains
Outline

· Context

· Structure of the Analyzer

✓ **Abstract Domain**

· Generalities

✓ **A relational numerical domain: Octagons**

· A symbolic domain: Trace partitioning

· Other domains

· Results Overview
Octagons

\[
\text{assume}(x \in [-10, 10])
\]

\[
\text{if}(x < 0)\{y = -x; \}
\]

\[
\text{else}\{y = x; \}
\]

\[
\text{if}(y \leq 5)
\]

\[
\{\text{assert}(-5 \leq x \leq 5); \}
\]

- With an interval analysis:
  - At point ①: \( x \in [-10, 10]; \ y \in [0, 10] \)
  - At point ②: \( x \in [-10, 10]; \ y \in [0, 5] \)
  - Analyzer alarm (assert not proved)

- We need a relation between \( x \) and \( y \):
  \( \Rightarrow \) i.e., a relational abstraction: polyhedra?

- Octagons:
  - Express constraints of the form \( \pm x \pm y \leq c \).
    In the example:
    - At point ①, \( 0 \leq y - x \leq 20; \ 0 \leq y + x \leq 20 \)
    - At point ②, \( y \in [0, 5]; \ 0 \leq y - x \leq 20; \ 0 \leq y + x \leq 20, \)
      hence \( x \in [-5, 5] \)
  - More reasonable cost: \( \mathcal{O}(n^2) \) space; \( \mathcal{O}(n^3) \) time (still high)

- Several issues to solve to integrate octagons: cost, floating points...
Preserving Scalability with Octagons

- Still too costly:
  - $O(n^3)$ time complexity per operation, if $n$ variables
  - So if $n \equiv 10000$: will not work
- A remark: we will not need (or get) a relation between all pair $(x, y)$
- ⇒ Use several smaller octagons
  - Pack: small group of variables to relate
  - Strategy and heuristics used to choose packs
    - syntactic pre-analysis: variables “used together”
    - possibility to add user-defined packs (rarely needed)
  - Cost: linear in the number of packs
    - Size of packs: bounded by a fixed constant
    - Number of packs: linear in the size of the code
    - ⇒ linear cost
Soundness and Floating Point Computations

- Rounding errors in floating point concrete computations
  But the domain is defined with real numbers
  (same for all relational abstractions, such as polyhedra, linear equalities...)

- Approximation of expressions:
  bounded with linear combinations with ranges as coefficients
  Example:

  \[
  y \in [-10.5.] \\
  x := y \star z + c
  \]

  \[
  \implies x := [-10. - \epsilon_0, 5. + \epsilon_0] \star z + [c - \epsilon_2, c + \epsilon_2]
  \]

  - Linearized forms can be handled by an octagon transfer function
  - Interval constraints used to make the transformation

- Relational abstraction:
  - Semantics of octagons in terms of real numbers
  - Linearization bridges the gap with the floating point values
    Takes into account all possible rounding errors
Reduction

- **Intuition:** Use distinct predicates to improve precision
  - $D_M^\#, \gamma$ is reduced iff $\gamma(x^\#) = \gamma(y^\#) \Rightarrow x^\# = y^\#
    - most domains are not reduced;
    - this is source of imprecision
  - Reduction: should map $x^\#$ into a more precise $y^\#$
  - Non reduction may cause incompleteness of the ordering

- **Intra-domain reduction:**
  - Principle:
    - $x - y \leq a \land y - z \leq b \implies x - z \leq a + b$
  - Cost considerations: cubic cost, hence should be used sparsely
    - Equivalent to a graph shortest path problem (Floyd Warshall algorithm)

- **Extra-domain reduction:**
  - Use the more precise bounds found above, to refine interval constraints
  - Get more precise constraints from other domains (some described later)
Outline

· Context
· Structure of the Analyzer

✓ Abstract Domain
  · Generalities
  · A relational numerical domain: Octagons

✓ A symbolic domain: Trace partitioning
  · Other domains
· Results Overview
• Interval abstraction:
  ♦ At $l_5$, approximation: $sgn \in [-1, 1]$
  ♦ Consequence: the division is not proved safe (alarm at $l_5$)
  ♦ Clearly, $sgn \neq 0$ for any real execution (false alarm)

• If $D^\#$ is a domain such that $\forall x \in D^\#$, $x$ stands for a convex set of concrete values: same result
  Hence, octagons will not fix this!
Disjunction-based Refinement

• Solution: perform a case analysis on $x$
in order to avoid considering the fictitious case $sgn = 0$

• Refined analysis:
  ♦ Invariant at $l_5$:
    
    $(x < 0 \land sgn = -1) \lor (x \geq 0 \land sgn = 1)$

  ♦ Results:
    ▶ The division is safe
    ▶ Invariant for $y$ at $l_6$: $y \geq 0$

• Definition of the domain:
  ♦ We simply focus on the control history:
  ♦ Invariant at $l_5$:
    
    \[
    \begin{cases}
    \text{TRUE branch} & \implies x < 0 \land sgn = -1 \\
    \text{FALSE branch} & \implies x \geq 0 \land sgn = 1 \\
    \end{cases}
    \]
System Refinement

- **Refining the control structure:**

  We enrich control states $l_i$ with tokens $t_j$ (e.g., TRUE, FALSE)

  $l_0$ if($x < 0$)
  $l_1$ $sgn = -1$
  $l_2$ else
  $l_3$ $sgn = 1$
  $l_4$ 
  $l_5$ $y = x/sgn$
  $l_6$ ...

  $P_0$ $(l_0)$ $(l_1)$ $(l_2)$ $(l_3)$ $(l_4)$ $(l_5)$ $(l_6)$
  $P_1$ $(l_0)$ $(l_1)$ $(l_2)$ $(l_3)$ $(l_4)$ $(l_5)$ $(l_6)$

  - At the semantics level, we have a partition:

    $\llbracket P \rrbracket (l_0) = \llbracket P_0 \rrbracket (l_0, \text{TRUE}) \cup \llbracket P_0 \rrbracket (l_0, \text{FALSE})$

  - A hierarchy of refining control structures

    $P_0$ refines $P$  
    $P_1$ refines $P_0$
Construction of the Partitioning Domain

- For each refined control structure $P_i$: a domain $D[P_i] = \mathbb{L} \times T_i \to D$
  - Concrete level function mapping partitions into sets of traces
  - Abstract level function mapping partitions into abstract invariants

- Overall structure, for a given $D$:

- A trace partitioning domain value:
  - A partition $P_i$ + a semantic value $V \in D[P_i]$
Instantiation in ASTRÉE

- **Main criteria** for trace partitioning:
  - If statements: delayed abstract join
  - Loop unrolling: distinguish the $n$ first iterations; delay the abstract join
  - Variable value: distinguish all possible values of a variable
    - do not partition if too many values
    - possible values are determined by the analysis
      Partitioning is **dynamic**, i.e. known only at analysis time

- **Partitioning strategies**:
  - Exhaustive application of the criteria would **not scale up**
    - huge number of partitions available
    - most partitions are of no interest or may play against widening
  - Strategies: determine which partitions to consider
    - where to distinguish flow paths
    - where to merge distinguished flow paths
A Realistic Example: Linear Interpolation

\[ y = \begin{cases} 
-1 & \text{if } x \leq -1 \\
-0.5 + 0.5 \times x & \text{if } -1 \leq x \leq 1 \\
-1 + x & \text{if } 1 \leq x \leq 3 \\
2 & \text{if } 3 \leq x 
\end{cases} \]

- Without partitioning:
  - No relation between \( x \) and the slope (corresponding range \( i \))
  - Analysis, with input \( x > 0 \), output possibly unbounded (slope in blue)
- With partitioning of the loop: above issues are fixed, output in \([-1, 2]\)
- Strategy: partition loops computing variables used as array index after the loop exit
Outline

· Context
· Structure of the Analyzer
✓ Abstract Domain
  · Generalities
  · A relational numerical domain: Octagons
  · A symbolic domain: Trace partitioning
✓ Other domains
· Results Overview
Analyzing Digital Filters

Simplified 2nd order filter:

\[
X_n = \begin{cases} 
\alpha X_{n-1} + \beta X_{n-2} + Y_n \\
I_n 
\end{cases}
\]

• Computed sequence:

• Concrete computations are bounded
  Issue: how to infer an abstract bound?

• No stable octagon or interval

• A polyhedra with many faces?

• Most simple stable surface:
  an ellipsoid
(Un)Stability of Floating Point Computations

- Real numbers: $x = 1.0$ at ①

- Floating point numbers
  - Rounding errors (concrete semantics)

- Accumulation of rounding errors: may cause a (slow) divergence

- Solution: use arithmetico-geometric progressions to bound rounding errors with a function of the number of concrete iterates:
  - **Constraint** $|x| \leq A \cdot B^n + C$, where $A$, $B$, $C$ are constants and $n$ is the iteration number
  - **Number of iterations**: bounded by $N$: $|x| \leq A \cdot B^N + C$

- Ellipsoids, progressions:
  - **Domains using mathematical theorems** proved once for all (design of the analyzer)
  - **Beyond what automatic refinement can do!**
Binary Decision Trees

\[
\begin{align*}
bp &= x \leq 0.; \\
bn &= x \geq 0.; \\
\text{if}(bp \&\& bn) &\quad \Rightarrow \quad y = 0.0; \\
\text{else} &\quad y = 1.0/x;
\end{align*}
\]

- Non-relational analysis: Alarm at ② division by 0
- Relations needed at ①:
  - \( bp = \text{FALSE} \Rightarrow x \neq 0 \)
  - \( bn = \text{FALSE} \Rightarrow x \neq 0 \)
- Domain similar to BDDs:
  - Nodes labeled with boolean variables
  - Leaves: values in a numerical domain e.g., intervals in this example
- Scalability issues:
  - Same as in the case of octagons;
  - Also addressed with a variable packing strategy
Outline

· Context
· Structure of the Analyzer
· Abstract Domain
✓ Results Overview
Benchmarks

- 2 families of synchronous embedded programs
  A340 and A380 Airbus Aircraft fly-by-wire systems
- 2.2 GHz Bi-opteron, 1 processor used (64-bits arch)

<table>
<thead>
<tr>
<th>LOC</th>
<th>70 000</th>
<th>226 000</th>
<th>400 000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>32</td>
<td>51</td>
<td>88</td>
</tr>
<tr>
<td>Memory used (Gb)</td>
<td>0.6</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>Time</td>
<td>46mn</td>
<td>3h57mn</td>
<td>11h48mn</td>
</tr>
<tr>
<td>False alarms</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Conclusion:

- Few or no false alarms: can be used to certify critical code
- Memory and time requirements are reasonable
- Due to the specialization of the analyzer
Recent and Future Extensions

- Extensions:
  - Memory model, to analyze low level features
  - Asynchronous softwares (ongoing)
  - Automatize alarm diagnostics...

- For More Information: