Conception des systèmes critiques
CAP'TRONIC - Pôle PEGASE

An overview of the ASTRÉE Analyzer

Jérôme Feret
DIENS
(INRIA/ÉNS/CNRS)
http://www.di.ens.fr/∼feret

Joint work with:
B. BLANCHET, P. COUSOT, R. COUSOT, L. MAUBORGNE,
A. MINÉ, D. MONNIAUX, X. RIVAL

1 Oct 2013

Certification of Critical Software

Goal of ASTRÉE:
Prove the absence of runtime errors in embedded, C programs

- Situation: bugs are no longer acceptable in such systems
  - May cause human casualties: transportation, energy
  - Prohibitive financial cost: Ariane 501 failure
- Automatic certification of the code is required
  - Soundness: absolutely essential
  - Strong efficiency requirements:
    - Keep analysis time low:
      analyses and fixes possible during development
    - Precision:
      number of alarms should be reasonable (e.g. < 10)
  - No alarm = full certification (If possible)

Specialization of the Analyzer

- Specialization to a family of embedded programs: Synchronous, real-time code:

```
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
end loop
```

- Properties of interest: Absence of run-time errors
  - No Run-Time Error
    e.g. division by 0, NaN, out-of-bound array access
  - No integer / floating point overflow
  - User-defined properties, e.g. architecture dependent

Specialization of the Analyzer

- Simplifying aspects:
  - Not full C: no malloc
  - No recursion, no backward goto
  - Data mostly static
- Challenging aspects:
  - Size: > 100 kLOC, > 10 000 variables
  - Floating point computations
    including filtering, non linear control with feed-back, interpolations...
  - Interdependencies among variables:
    * Stability of computations should be established
    * Complex relations should be inferred among numerical, boolean data
    * Very long data paths from input to outputs

Principle

Compute an over-approximation of the reachable states

- Model of the language: semantics
  $[\![P]\!]$ = set of runs (i.e. traces) of P
  - C 99 norm
  - IEEE 754-1985 Floating point norm
  - User/architecture defined assumptions:
    * Size of integer data types
    * Initialization of statically allocated variables
    * Range of inputs; maximum program run-time
- Abstraction = approximation relying on abstract domains
- Derivation of an automatic, sound static analyzer
- Certification of a program:
  - Fully automatic computation of an invariant
  - Verification of the absence of runtime errors

Overview

1. The ASTRÉE Abstract Interpreter
2. Abstract Domains
3. Practical Use and Benchmarks
4. Conclusions and Perspectives

Development of ASTRÉE

- Fall 2001: demand for a high precision, fast analyzer
  ASTRÉE project started:
  - Scalability ensured first (algorithms and data-structures)
  - Simple, non relational domains
  - First refinements
  - Analysis of 10 kLOCs, low number of alarms

- Then, real applications considered:
  - Investigation of an alarm:
    $\Rightarrow$ Source of imprecision
    $\Rightarrow$ true alarm or need for a refinement?
  - Implementation of new domains,
    solve imprecisions and preserve scalability
  - Analysis of several families of real-world, large applications

Abstractions and Abstract Domains

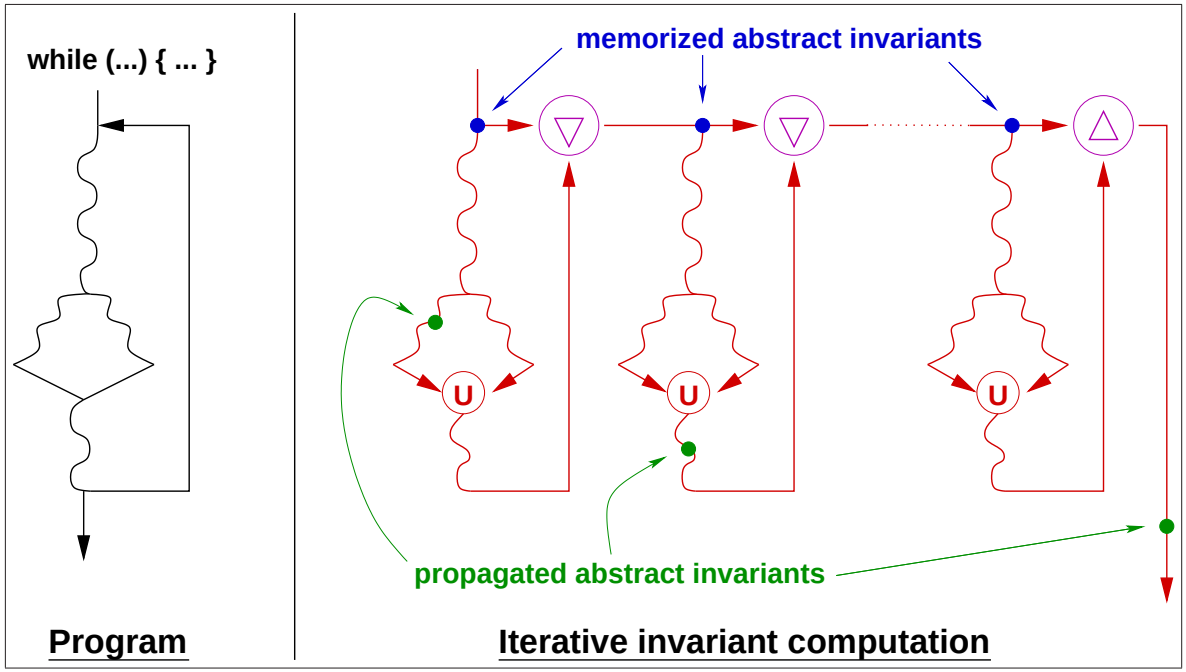
- What ASTRÉE computes:
  Invariant $I \in D^\sharp$: approximation for the set of traces $[\![P]\!]$
- Structure of the abstraction $I \in D^\sharp$:
  - For each control point l
  - For each execution context κ (e.g. calling stack)

  $\Rightarrow$ an approximation $I(l, \kappa) \in D^\sharp_{\mathtt{M}}$ for a set of memory states
- Layout of $D^\sharp_{\mathtt{M}}$:
  - Reduced product of a collection of abstract domains
  - Each domain:
    - Expresses a (generally infinite) family of predicates
    - Transfer functions: $assign, guard$, ... operators
    - Approximations for $\cup$: $\sqcup$ and ∇ (convergence acceleration)

The Abstract Interpreter

Principle: play all executions in a single, abstract computation

- Analysis of a basic statement $x = e$:
  - Transfer function $\mathit{assign}(x = e) : D^\sharp_{\mathtt{M}} \to D^\sharp_{\mathtt{M}}$
  - $D^\sharp_{\mathtt{M}}$: accounts for the new/removed constraints
- Analyzing compound programs, e.g. loops:



Overview

1. The ASTRÉE Abstract Interpreter
2. Abstract Domains
3. Practical Use and Benchmarks
4. Conclusions and Perspectives

A Simple Domain: Intervals

- Resolution of concrete cells:
  - 1 abstract cell $\equiv$ 1 or more concrete cells (smashed arrays)
  - Simple points-to information
- Interval abstraction:
  - Constraints $a \leq x \leq b$ (x abstract cell)
  - Very common abstraction
  - Implementation: sound approximation for floating point (in all domains)
- Development of ASTRÉE: structure + intervals was the base:
  - Enough to express the absence of runtime-errors (array bounds, overflows)
  - Not enough to express its proof

Octagons

- Interval analysis:
  - At ①, $\mathtt{x} \in [-10, 10];\ \mathtt{y} \in [0, 10]$
  - At ②, $\mathtt{x} \in [-10, 10];\ \mathtt{y} \in [0, 5]$
  - Alarm (assert not proved)

assume$(\mathtt{x} \in [-10, 10])$
if$(\mathtt{x} < 0)\{\mathtt{y} = -\mathtt{x};\}$
else$\{\mathtt{y} = \mathtt{x};\}$
①**if**$(y \leq 5)$
$\{$②**assert**$(-5 \leq x \leq 5);\}$

- A relation between $\mathtt{x}$ and $\mathtt{y}$ is required:
  $\Rightarrow$ We need a relational abstraction
- Octagons:
  - Express constraints of the form $\pm x \pm y \leq c$.
    Above example:
    - At ①, $0 \leq \mathtt{y} - \mathtt{x} \leq 20;\ 0 \leq \mathtt{y} + \mathtt{x} \leq 20$
    - At ②, $\mathtt{y} \in [0, 5];\ 0 \leq \mathtt{y} - \mathtt{x} \leq 20;\ 0 \leq \mathtt{y} + \mathtt{x} \leq 20,$
      so we derive $\mathtt{x} \in [-5, 5]$
  - Reasonable cost: $\mathcal{O}(n^2)$ memory and $\mathcal{O}(n^3)$ time complexity

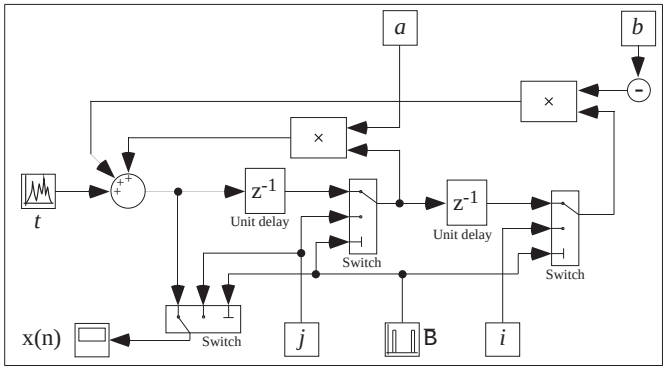
Using Octagons

Several issues should be addressed:

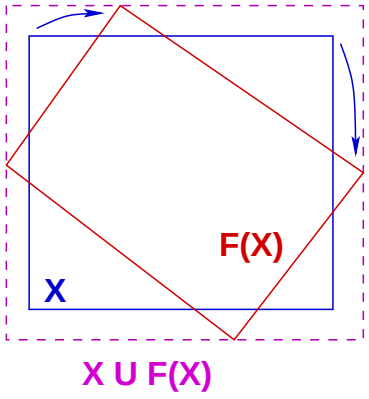
- Scalability: $\mathcal{O}(n^3)$ time, $n \equiv 10\,000$ will not scale:
  $\Rightarrow$ Use many small octagons instead of a big one
  - Packs: small group of variables relations are required for
  - Strategy: determines packs, required relations represented
  - Complexity: linear in the number of packs
    Size of packs: bounded by a constant
    Number of packs: linear in the size of the code
    $\Rightarrow$ Linear complexity
- Floating point rounding errors in the concrete computations
  solved by linearization of expressions:
  - Expressions approximated with real interval linear forms
  - Relational domain: semantics in terms of real numbers
  - Rounding errors in concrete computations accounted for at linearization time

Analyzing Filters

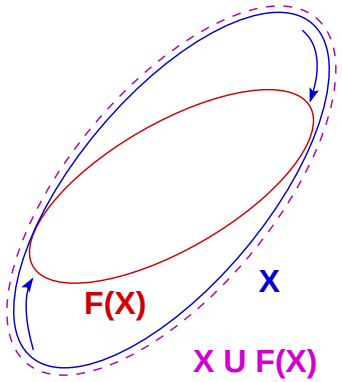
Simplified 2nd Order Filter:



- Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is bounded, which must be proved in the abstract
- No stable interval or octagon
- Simplest stable surface is an ellipse



Unstable intervals



Stable ellipse

Bounding Slow Divergences

- With real numbers: $x = 1.0$ at ①
- With floating point numbers:
  $\Rightarrow$ rounding errors in the concrete;
  $\Rightarrow$ these errors accumulate;
  $\Rightarrow$ possible cause of divergence

```
x = 1.0;
while(TRUE){①
  x = x/3.0;
  x = x * 3.0;
}
```

Solution: an arithmetico-geometric progression domain, aimed at bounding the rounding errors using the number of iterations:

- Relation $\mid x \mid \leq A \cdot B^n + C$,
  where A, B, C are constants, n is the iteration number
- Number of iterations: bounded by N: $\Rightarrow \mid x \mid \leq A \cdot B^N + C$

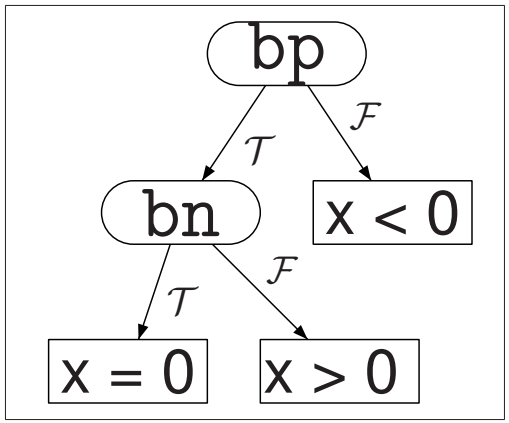
Decision Trees

- Non relational analysis: alarm at ② (div. 0)
- Relations needed at ①:
  - $\texttt{bp} = \texttt{FALSE} \Rightarrow \texttt{x} \neq 0$
  - $\texttt{bn} = \texttt{FALSE} \Rightarrow \texttt{x} \neq 0$

```
bp = x ≤ 0.;
bn = x ≥ 0.;
①if(bp && bn)
   ②y = 0.0;
else y = 1.0/x;
```

Solution: Domain similar to BDDs:

- Nodes labeled with booleans variables
- Leaves: values in an underlying domain e.g. interval in the example



Scalability problems:

- Packing, similar to octagons

Trace Partitioning

- No partitioning, interval analysis, at ①:
  - $\texttt{i} \in [0, 2]$
  - $(\texttt{t}[\texttt{i}+1] - \texttt{t}[\texttt{i}]) \in [-10, 20]$
  - Alarm at ① (divide by 0)

```
assume(x ∈ [0, 15]);
float t = {0, 10, 20, 30};
int i = 0;
while(i < 3 && x < t[i + 1])
  {i++;}
①y = 1.0/(t[i + 1] − t[i]);
```

- We need to do case-by-case analysis:
  - Relate $\texttt{x}$ and $\texttt{i}$
  - Relate $\texttt{x}$ and the number of iterations in the loop
- General, control-based partitioning domain
- Partitioning strategy (choice of partitions)

Overview

1. The ASTRÉE Abstract Interpreter
2. Abstract Domains
3. Practical Use and Benchmarks
4. Conclusions and Perspectives

Use of ASTRÉE

- The analyzer:
  - Full automatic mode:
    Should do well for the families of programs ASTRÉE is designed for
  - $\simeq 150$ options, so as to set
    - The input (one or many files...)
    - The iteration strategy, the packing strategy
    - The domains to enable or disable, domain parameters
    - The export of invariants to disk
  - Standard output: alarms, invariants
  - Can be run in parallel mode
- A graphical interface:
  Navigation through invariants (saved invariants)

Industrial applications

In Nov. 2003, ASTRÉE certified the absence of any RTE in the primary flight control software of the Airbus A340.

In Apr. 2005, ASTRÉE certified electric flight control codes then in development and test for the A380 series.

In Apr. 2008, ASTRÉE was able to prove the absence of any RTE in a C version of the automatic docking software of the Jules Vernes ATV (for the International Space Station).

ASTRÉE is commercially available from AbsInt Angewandte Informatik.

Benchmarks

Program number	Nb of lines	Nb of iterations	Time
1	70k	49	1h35
2	154k	218	11h36
3	173k	199	14h30
1	122k	27	41mn
2	502k	69	7h34
3	544k	181	22h37
4	594k	123	23h
5	780k	144	32h
6	705k	141	28h

On a 2.6 GHZ single core 64-bit Intel under Linux, using between 2GB and 8GB of memory.

Overview

1. The ASTRÉE Abstract Interpreter
2. Abstract Domains
3. Practical Use and Benchmarks
4. Conclusions and Perspectives

Main Project Results

- The proof of strong safety properties is amenable to static analysis methods:
  - Very few or no false alarms
  - Reasonable resource usage
  - Thanks to a specialized abstract interpreter
- Many practical and theoretical advances:
  - Relational numerical domains and floating point
  - Packing, linearization and relational domains
  - Development of new, specialized domains
  - Implementation of symbolic domains, e.g. partitioning, symbolic...

Ongoing works

- Analyze asynchronous programs

  $\Rightarrow$ abstract interleavings while preserving the accuracy of the analyzer.
  Encouraging early results: alarms successfully diagnosed

- Analyze more complex data-structures

  $\Rightarrow$ Design scalable shape analysis for recursive data-structures.