Abstract-Interpretation-based Static Analysis of Safety-Critical Embedded Software

Patrick Cousot

Patrick.Cousot@ens.fr www.di.ens.fr/~cousot

Workshop Airbus/partners on formal verification tools strategy

Airbus St Martin, M86, Auditorium Dubaï, Dec. 4–5, 2008

Abstract

Static software analysis has known brilliant successes in the small, by proving complex program properties of programs of a few dozen or hundreds of lines, either by systematic exploration of the state space or by interactive deductive methods. To scale up is a definite problem. Very few static analyzers are able to scale up to millions of lines without sacrificing automation and/or soundness and/or precision. Unsound static analysis may be useful for bug finding but is less useless in safety critical applications where the absence of bugs, at least of some categories of common bugs, should be formally verified.

After recalling the basic principles of abstract interpretation including the notions of abstraction, approximation, soundness, completeness, false alarm, etc., we introduce the domain-specific static analyzer ASTRÉE (`www.astree.ens.fr`) for proving the absence of runtime errors in safety critical real time embedded synchronous software in the large.

The talk emphasizes soundness (no runtime error is ever omitted), parametrization (the ability to refine abstractions by options and analysis directives), extensibility (the easy incorporation of new abstractions to refine the approximation), precision (few or no false alarms for programs in the considered application domain) and scalability (the analyzer scales to millions of lines).

In conclusion, present-day software engineering methodology, which is based on the control of the design, coding and testing processes should evolve in the near future, to incorporate a systematic control of final software product thanks to domain-specific analyzers that scale up.

1. Motivation: bugs are everywhere

Example of bug report

Department
Infrastructure Australia
Bureau of Infrastructure, Transport and Regional Economics

Minister | Help | Feedback | Contact Us | Search keyword GO

14 October 2008 - Qantas Airbus A330 accident Media Conference

- Transport
  - Transport Safety
  - About the ATSB
  - Newsroom
  - Aviation Safety
  - Marine Safety
  - Rail Safety
  - Road Safety
  - Publications and Investigation Reports
  - Accident and Incident Notification Forms
  - Aviation
  - Roads
  - Rail
  - Maritime
- Regional
- Local Government
- The Department

Adjust font size:

MEDIA RELEASE

2008/43

Qantas Airbus A330 accident Media Conference

14 October 2008

The Australian Transport Safety Bureau's investigation into the accident involving an Airbus A330-300 aircraft operating as Qantas flight 72 on a flight from Singapore to Perth on 7 October 2008 is progressing well. The ATSB has scheduled the media conference this evening to coincide with the release of an Operators Information Telex/Flight Operations Telex, which is being sent by Airbus to operators of all Airbus aircraft. The aim of that telex is to:

- update operators on the factors identified to date that led to the accident involving QF72,
- provide operational recommendations to mitigate risk in the event of a reoccurrence of the situation which occurred on QF72.

To assist in understanding the following information, I would just like to refer you quickly to the diagrams projected on the screen specifically, the term angle of attack which refers to the difference in angle between the aircraft and its control surfaces, and the air stream as the aircraft moves through the air.

The next diagram is a simple representation of the aircraft and the components relevant to this explanation, which include the angle of attack sensors located on the outside of the aircraft, the Air Data Inertial Reference Units (ADIRUs), of which there are three, located in the avionics compartment inside the aircraft, the Flight Control Primary Computers of which there are also three located in the avionics compartment, and the elevators, located on the aircrafts horizontal stabiliser. In the context of this occurrence, the angle of attack sensors send raw data to the ADIRUs, which provide processed angle of attack information to the Flight Control Primary Computers, which in turn command the elevator position.

Returning to the circumstances of the 7 October flight, preliminary analysis of the Flight Data Recorder data, Post Flight Report data and Built-in Test Equipment data has enabled the investigation to establish a preliminary sequence of events this information is also contained in the Airbus telex.

The aircraft was flying at FL 370 or 37, 000 feet with Autopilot and Auto-thrust system engaged, when an Inertial Reference System fault occurred within the Number-1 Air Data Inertial Reference Unit (ADIRU 1), which resulted in the Autopilot automatically disconnecting. From this moment, the crew flew the aircraft manually to the end of the flight, except for a short duration of a few seconds, when the Autopilot was reengaged. However, it is important to note that in fly by wire aircraft such as the Airbus, even when being flown with the Autopilot off, in normal operation, the aircrafts flight control computers will still command control surfaces to protect the aircraft from unsafe conditions such as a stall.

The faulty Air Data Inertial Reference Unit continued to feed erroneous and spike values for various aircraft parameters to the aircrafts Flight Control Primary Computers which led to several consequences including:

- false stall and overspeed warnings
- loss of attitude information on the Captain's Primary Flight Display
- several Electronic Centralised Aircraft Monitoring system warnings.

About 2 minutes after the initial fault, ADIRU 1 generated very high, random and incorrect values for the aircrafts angle of attack.

These very high, random and incorrect values of the angle attack led to:

- the flight control computers commanding a nose-down aircraft movement, which resulted in the aircraft pitching down to a maximum of about 8.5 degrees,
- the triggering of a Flight Control Primary Computer pitch fault.

The crew's timely response led to the recovery of the aircraft trajectory within seconds. During the recovery the maximum altitude loss was 650 ft.

The Digital Flight Data Recorder data show that ADIRU 1 continued to generate random spikes and a second nose-down aircraft movement was encountered later on, but with less significant values in terms of aircraft's trajectory.

At this stage of the investigation, the analysis of available data indicates that the ADIRU 1 abnormal behaviour is likely as the origin of the event.

The aircraft contains very sophisticated and highly reliable systems. As far as we can understand, this appears to be a unique event and Airbus has advised that it is not aware of any similar event over the many years of operation of the Airbus.

Airbus has this evening, Australian time, issued an Operators Information Telex reflecting the above information. The telex also foreshadows the issue of Operational Engineering Bulletins and provides information relating to operational recommendations to operators of A330 and A340 aircraft fitted with the type of ADIRU fitted to the accident aircraft. Those recommended practices are aimed at minimising risk in the unlikely event of a similar occurrence. That includes guidance and checklists for crew response in the event of an Inertial Reference System failure.

Meanwhile, the ATSB's investigation is ongoing and will include:

- Download of data from the aircraft's three ADIRUs and detailed examination and analysis of that data. Arrangements are currently being made for the units to be sent to the component manufacturer's facilities in the US as soon as possible and for ATSB investigators to attend and help with that testing, along with representatives from the US National Transportation Safety Board, The French Bureau dEnquêtes et dAnalyses (BEA) and Airbus.
- In addition, investigators have been conducting a detailed review of the aircraft's maintenance history, including checking on compliance with relevant Airworthiness Directives, although initial indications are that the aircraft met the relevant airworthiness requirements.
- Work is also ongoing to progress interviews, which will include with injured passengers to understand what occurred in the aircraft cabin. The ATSB plans to distribute a survey to all passengers.

There has been close and frequent communication between the ATSB, Qantas, Airbus, the BEA, and CASA. That close communication will continue as the investigation progresses to ensure that any additional safety action can be instigated as soon as possible should critical safety factors be identified. The ATSB expects to publish a Preliminary Factual report in about 30 days from the date of the accident.

Media Contact: David Hope 1800 020 616

Related Documents: | Audio file of media conference, 14 October 2008 (18 MB) |

Print
Last Updated: 14 October, 2008

Privacy | Copyright | Disclaimer | Linking to the ATSB website | Sitemap

"The Australian Transport Safety Bureau (ATSB) found that the main probable cause of this incident was a *latent software error* which allowed the ADIRU to use data from a failed accelerometer"

http://www.atsb.gov.au/newsroom/2008/release/2008_43.aspx,
http://en.wikipedia.org/wiki/Qantas_Flight_72

2. Varieties of Static Analyses

Static Analysis

– In general static analysis means "the *fully automatic* verification of properties of program executions using the program text only" (excluding running programs)

– But for trivial cases, it is undecidable

– Alternatives to impossible total verification:
  - under-verification (testing, bounded model-checking, bug pattern mining, etc): bug finding, misses bugs, never ends
  - over-verification (typing, dataflow analysis,etc): no bug missed but false alarms

– Challenge: total verification for a given category of properties and a given family of programs (no bug missed, no false alarm but not for all possible properties of all programs)

Example: Sparrow versus Astrée

– **Original program:**
  - Astrée: signals all errors
  - Sparrow: forgets one!

– **Corrected program:**
  - Astrée: no alarm
  - Sparrow: still errors!

```c
int any() {
    int x;
    __ASTREE_known_fact(((0<=x)&&(x<=32767)));
    return x;
}

typedef struct _S {
    int arr[10];
    int n;
    int *p;
} S1;

void alias ()
{
//    struct _S *s;
//    int  p,*q;
//    s = (struct _S*)malloc(100);
//    p = (int*)malloc(100*sizeof(int));
//    q = p; // alias
//    s->p = q; // alias
//    *(s->p+100) = 1;    // buffer overrun : alias
}

// safe usage of library functions
void library_calls()
{
     char buf1[100];
     char buf2[200];
//    FILE *fp = NULL;

     int id = 5;
     S1 s;
     s.n = 10;
//     memmove (s.arr + id + 1, s.arr + id, sizeof(int) * (s.n - id)); // buffer overrun: LIB2

//     strcpy(buf1, buf2); // buffer overrun: LIB3
}

// correct condition check for function parameters
void param_check (int k)
{
     int arr[10];
     if (k > 10) return;
     arr[k] = k;                  // buffer overrun: PARAM1
}

void  loop_test()
{
     int i, j, k, pp, datal, datar;
     int arr[10][20];
     int x[100];
     int tmp[64];
     int triple[3][4][5];
     int *yptr;

     // nested loops
     for (i=0;i<10;i++)
         for (j=0;j<=20;j++)
             arr[i][j] = 0;      // buffer overrun: LOOP1

     // stride
     for (i=0;i<100;i+=2)
     {
         x[i] = datal;
         x[i+2] = datar;          // buffer overrun: LOOP2
     }
}

int atkbd(int *dev, unsigned int type)
{
    const short period[32] =
            { 33, 37, 42, 46, 50, 54, 58, 63, 67, 75, 83, 92, 100, 109, 116, 125,
             133, 149, 167, 182, 200, 217, 232, 250, 270, 303, 333, 370, 400, 435, 470, 500 }
    const short delay[4] =
            { 250, 500, 750, 1000 };
    unsigned char param[2];
    int i, j;

    switch (type) {
        case 0x09: return 0;
        case 0x11:
        case 0x14:
        i = j = 0;
        while (i < 32 && period[i] < *dev) i++;
        while (j < 4 && delay[j] < *dev) j++;
        *dev = period[i];         // buffer overrun: AFTERLOOP2
        *dev = delay[j];          // buffer overrun: AFTERLOOP3
        param[0] = i | (j << 5);
        return 0;
    }
    return -1;
}

void bo_test()
{
     atkbd((int*)any(), any());
     library_calls ();
     loop_test();
     param_check(any());
}

int main()
{
     bo_test();
     return 1;
}
```

Sparrow The Early Bird

http://www.spa-arrow.com/

3. Abstract Interpretation

Example of static analysis

Example after invariant abstraction:

$\{y \geqslant 0\}$ $\leftarrow$ hypothesis

```
x := y
```

$\{I(x, y)\}$ $\leftarrow$ loop invariant

```
while (x > 0) do
  x := x - 1;
od
```

Abstract fixpoint equation:

$$I(x, y) = x \geqslant 0 \wedge (x = y \vee I(x + 1, y)) \qquad (i.e.\ I = \mathbf{F}^\sharp(I)^{(1)})$$

Equivalent Floyd-Naur-Hoare verification conditions:

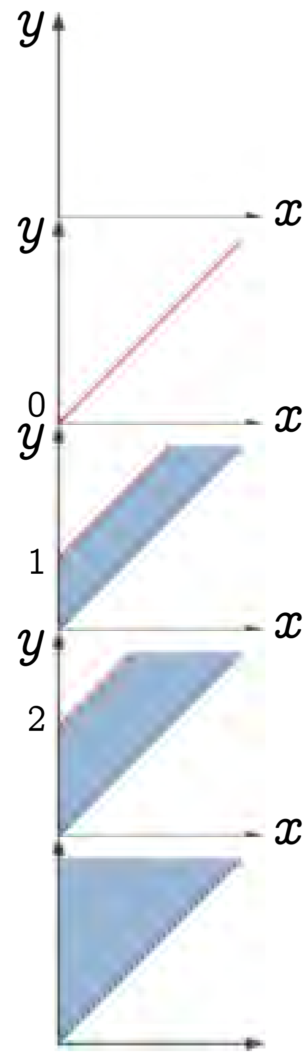
$$(y \geqslant 0 \wedge x = y) \Longrightarrow I(x, y) \qquad \textit{initialisation}$$

$$(I(x, y) \wedge x > 0 \wedge x' = x - 1) \Longrightarrow I(x', y) \qquad \textit{iteration}$$

(1) We look for the most precise invariant I, implying all others, that is $\mathsf{lfp}^{\Longrightarrow}\ \mathbf{F}^\sharp$.

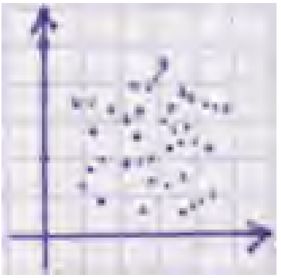
Accelerated Iterates $I = \bigsqcup_{n\to\infty} \mathbf{F}^{\sharp^n}(\text{false})$

$$
\begin{aligned}
I^0(x,y) &= \text{false}\\
I^1(x,y) &= x \geqslant 0 \wedge (x = y \vee I^0(x+1,y))\\
&= 0 \leqslant x = y\\
I^2(x,y) &= x \geqslant 0 \wedge (x = y \vee I^1(x+1,y))\\
&= 0 \leqslant x \leqslant y \leqslant x+1\\
I^3(x,y) &= x \geqslant 0 \wedge (x = y \vee I^2(x+1,y))\\
&= 0 \leqslant x \leqslant y \leqslant x+2\\
I^4(x,y) &= I^2(x,y) \mathbin{\nabla} I^3(x,y) \leftarrow \text{widening}\\
&= 0 \leqslant x \leqslant y\\
I^5(x,y) &= x \geqslant 0 \wedge (x = y \vee I^4(x+1,y))\\
&= I^4(x,y) \quad \text{fixed point!}
\end{aligned}
$$

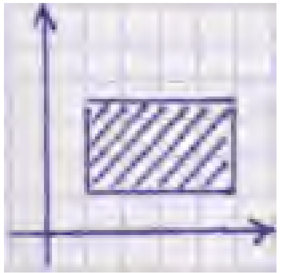


The invariants are computer representable with octagons!

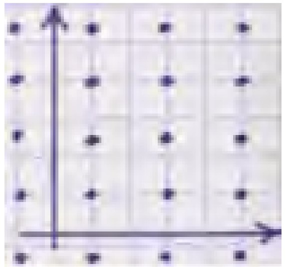
Examples of abstractions used by ASTRÉE



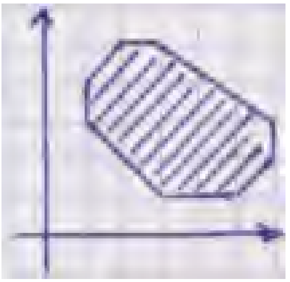
semantics
set of points



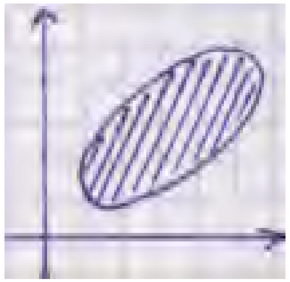
intervals
$x \in [a, b]$



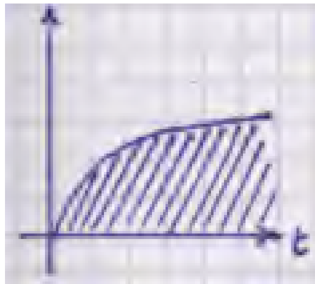
simple congruences
$x \equiv a[b]$



octagons
$\pm x \pm y \leqslant a$



ellipsoids
$ax^2 + by^2 + cxy \leqslant d$



exponentials
$x(t) \leqslant a^{bt}$

4. Scaling up

The difficulty of scaling up

– The abstraction must be coarse enough to be effectively computable with reasonable resources

– The abstraction must be precise enough to avoid false alarms

– Abstractions to *infinite domains with widenings* are more expressive than abstractions to *finite domains* (when considering the analysis of a programming language) [CC92]

– Abstractions are ultimately incomplete (even intrinsically for some semantics and specifications [CC00])

5. Astrée

Abstraction/refinement by tuning the cost/precision ratio in ASTRÉE

– Approximate reduced product of a choice of coarsenable/refinable abstractions

– Tune their precision/cost ratio by

  - Globally by parametrization
  - Locally by (automatic) analysis directives

  so that the overall abstraction is <u>not</u> uniform.

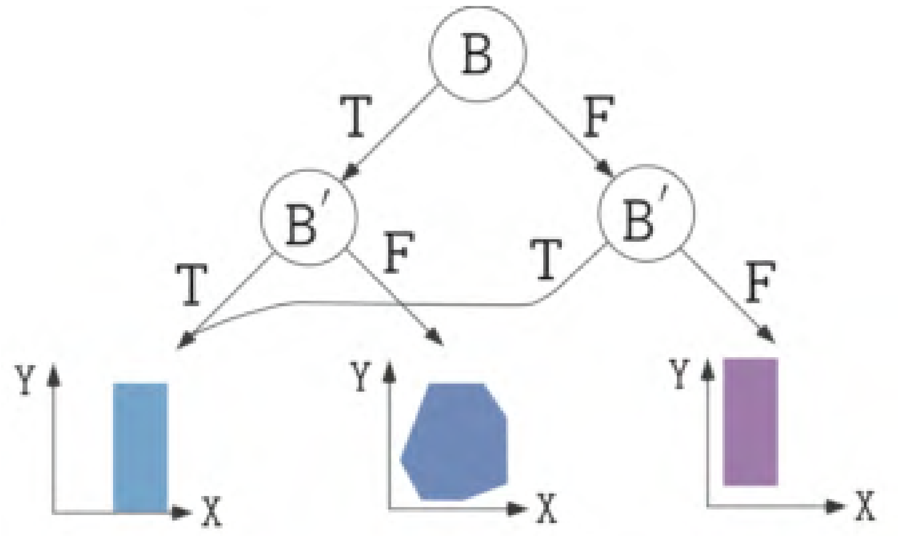
Example of abstract domain choice in Astrée

```
/* Launching the forward abstract interpreter */
/* Domains:  Guard domain, and Boolean packs (based on Absolute
value equality relations, and Symbolic constant propagation
(max_depth=20), and Linearization, and Integer intervals, and
congruences, and bitfields, and finite integer sets, and Float
intervals), and Octagons, and High_passband_domain(10), and
Second_order_filter_domain (with real roots)(10), and
Second_order_filter_domain (with complex roots)(10), and
Arithmetico-geometric series, and new clock, and Dependencies
(static), and Equality relations, and Modulo relations, and
Symbolic constant propagation (max_depth=20), and Linearization,
and Integer intervals, and congruences, and bitfields, and
finite integer sets, and Float intervals.  */
```

Example of abstract domain functor in ASTRÉE: decision trees

– Code Sample:

```
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ...
    B = (X == 0);
    ...
    if (!B) {
      Y = 1 / X;
    }
    ...
  }
}
```



The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leafs

Reduction [CC79, CCF+08]

Example: reduction of intervals by simple congruences

```
% cat -n congruence.c
     1  /* congruence.c */
     2  int main()
     3  { int X;
     4    X = 0;
     5    while (X <= 128)
     6      { X = X + 4; };
     7    __ASTREE_log_vars((X));
     8  }
% astree congruence.c -no-relational -exec-fn main |& egrep "(WARN)|(X in)"
direct = <integers (intv+cong+bitfield+set): X in {132} >
```

Intervals : $X \in [129, 132]$ + congruences : $X = 0 \mod 4 \Longrightarrow X \in \{132\}$.

Parameterized abstractions

– Parameterize the cost / precision ratio of abstractions in the static analyzer

– Examples:

  - array smashing: `--smash-threshold` n (400 by default) $\rightarrow$ smash elements of arrays of size $> n$, otherwise individualize array elements (each handled as a simple variable).

  - packing in octogons: (to determine which groups of variables are related by octagons and where)
    - `--fewer-oct`: no packs at the function level,
    - `--max-array-size-in-octagons` n: unsmashed array elements of size $> n$ don't go to octagons packs

Parameterized widenings

- Parameterize the rate and level of precision of widenings in the static analyzer
- Examples:
  - delayed widenings: `--forced-union-iterations-at-beginning` n (2 by default)
  - thresholds for widening (e.g. for integers):

```
let widening_sequence =
 [ of_int 0; of_int 1; of_int 2; of_int 3; of_int 4; of_int 5;
   of_int 32767; of_int 32768; of_int 65535; of_int 65536;
   of_string "2147483647"; of_string "2147483648";
   of_string "4294967295" ]
```

Analysis directives

– Require a local refinement of an abstract domain

– Example:

```
% cat repeat1.c
typedef enum {FALSE=0,TRUE=1} BOOL;
int main () {
  int x = 100; BOOL b = TRUE;

  while (b) {
    x = x - 1;
    b = (x > 0);
  }
}
% astree -exec-fn main repeat1.c |& egrep "WARN"
repeat1.c:5.8-13::[call#main@2:loop@4>=4:]: WARN: signed int arithmetic
range [-2147483649, 2147483646] not included in [-2147483648, 2147483647]
%
```

Example of directive (cont'd)

```
% cat repeat2.c
typedef enum {FALSE=0,TRUE=1} BOOL;
int main () {
  int x = 100; BOOL b = TRUE;
 __ASTREE_boolean_pack((b,x));
  while (b) {
    x = x - 1;
    b = (x > 0);
  }
}
```

```
% astree -exec-fn main repeat2.c |& egrep "WARN"
%
```

The insertion of this directive could be automated in ASTRÉE (if the considered family of programs has "repeat" loops).

Automatic analysis directives

– The directives can be inserted automatically by static analysis

– Example:

```
% cat p.c
int clip(int x, int max, int min) {
 if (max >= min) {
  if (x <= max) {
   max = x;
  }
  if (x < min) {
   max = min;
  }
 }
 return max;
}
void main() {
 int m = 0; int M = 512; int x, y;
 y = clip(x, M, m);
  __ASTREE_assert(((m<=y) && (y<=M)));
}
% astree -exec-fn main p.c |& grep WARN
%
```

```
% astree -exec-fn main p.c -dump-partition
...
int (clip)(int x, int max, int min)
{
  if ((max >= min))
  {  __ASTREE_partition_control((0))
    if ((x <= max))
    {
      max = x;
    }
    if ((x < min))
    {
      max = min;
    }
    __ASTREE_partition_merge_last(());
  }
  return max;
}
...
%
```

Adding new abstract domains

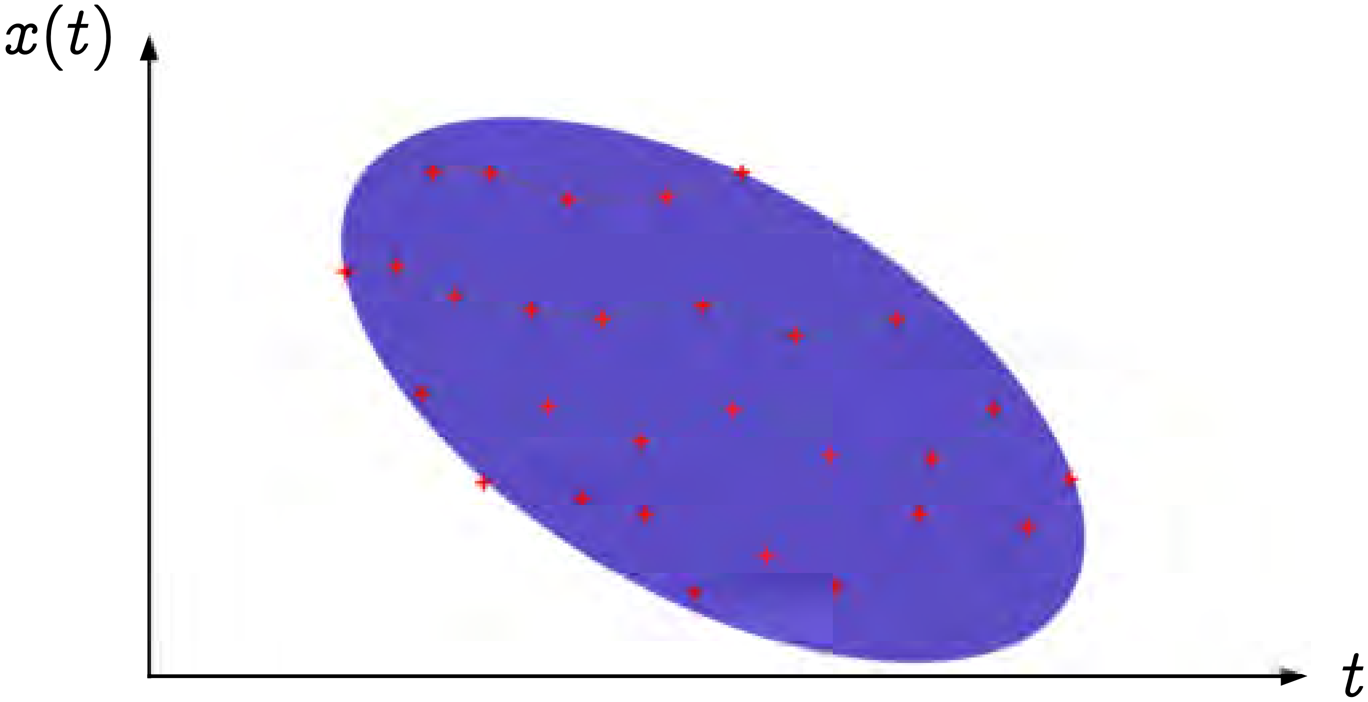
– The weakest invariant to prove the specification may not be expressible with the current refined abstractions ⇒ false alarms cannot be solved

– No solution, but adding a new abstract domain:

  - representation of the abstract properties
  - abstract property transformers for language primitives
  - widening
  - reduction with other abstractions

– Examples : ellipsoids for filters, exponentials for accumulation of small rounding errors, quaternions, ...

Abstraction by ellipsoid for filters

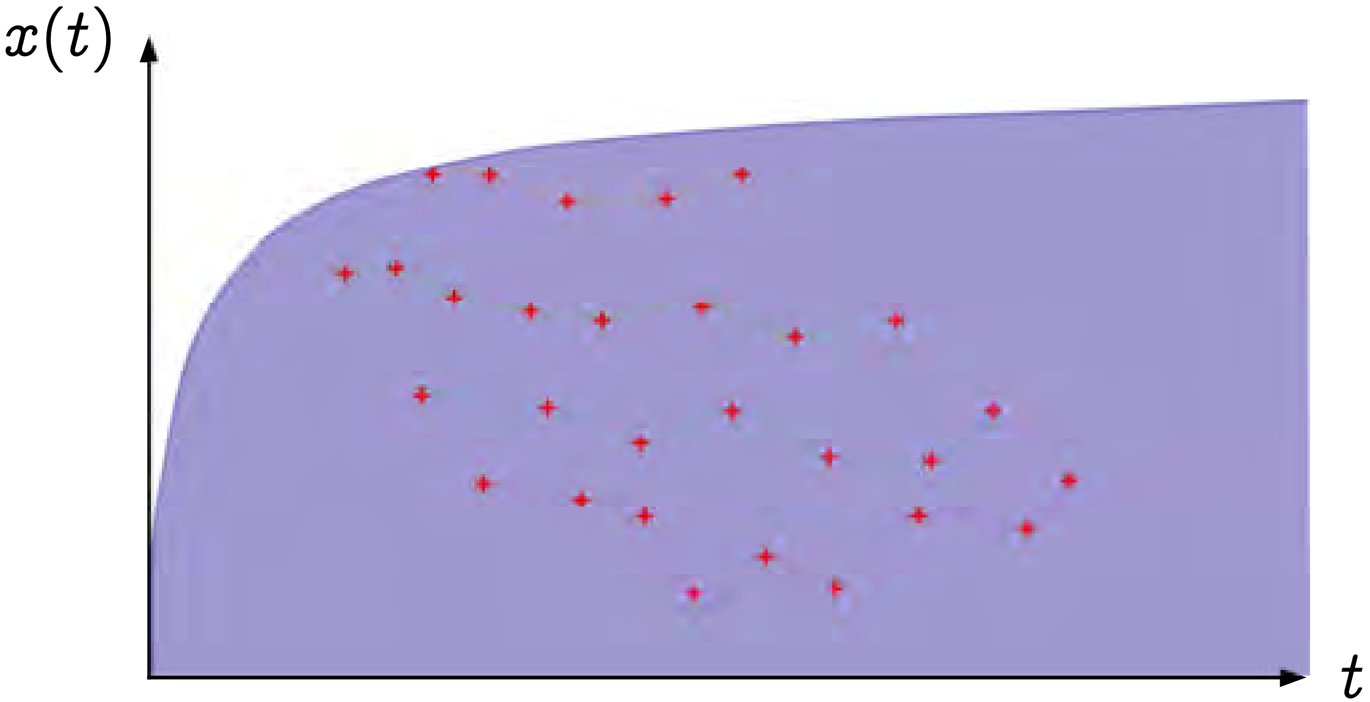


Ellipsoids $(x-a)^2 + (y-b)^2 \leq c$

Example of analysis by ASTRÉE

```
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;
void filter () {
  static float E[2], S[2];
  if (INIT) { S[0] = X; P = X; E[0] = X; }
  else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
             + (S[0] * 1.5)) - (S[1] * 0.7)); }
  E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}
void main () { X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}
```

Abstraction by exponentials for accumulation of small rounding errors



Exponentials $a^x \leq y$

Example of analysis by ASTRÉE

```
% 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)
                 * 5.0e-03)); };
  B = A;
  if (SWITCH) {A = P;}
  else {A = X;}
}
```

```
void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock(());
  }}
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));
```

```
astree -exec-fn main -config-sem retro.config
retro.c |& grep "|P|" | tail -n 1
|P| <=1.0000002*((15.  +
5.8774718e-39/(1.0000002-1))*(1.0000002)ĉlock -
5.8774718e-39/(1.0000002-1)) + 5.8774718e-39 <=
23.039353
```

6. Industrial Application of ASTRÉE

Industrial results obtained with ASTRÉE

- Automatic proofs of absence of runtime errors in Electric Flight Control Software:
  - A340/600: 132.000 lines of C, 40mn on a PC 2.8 GHz, 300 Mb (Nov. 2003)
  - A380: 1.000.000 lines of C, 34h, 8 Gb (Nov. 2005)

  no false alarm, **World premières !**

- Automatic proofs of absence of runtime errors in the ATV software [(2)]:
  - C version of the automatic docking software: 102.000 lines of C, 23s on a Quad-Core AMD Opteron™ processor, 16 Gb (Apr. 2008)

---

(2) the Jules Vernes Automated Transfer Vehicle (ATV) enabling ESA to transport payloads to the International Space Station.

7. Other Prototypes Developped by the Abstraction project/team

Lcertify

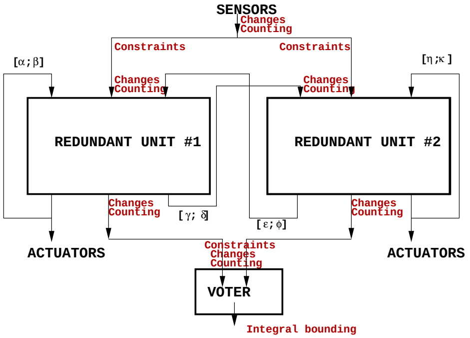
- ASTRÉE proves the absence of runtimes errors on C code (with semantics tailored for a given Intel/PPC 32/64 bits machine)
- In absence of runtime errors, LCERTIFY proves the semantic equivalence of the C and machine code

$\Rightarrow$ the certification is on the flying code.

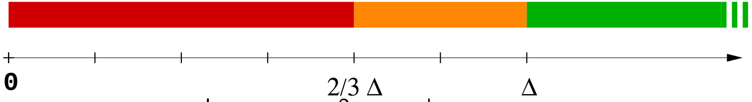
Macro generation

- Most of the control/command code (75%?) is automatically generated from SCADE/SAO, but for the primitives (handcoded by macros)
- Developed a formal specification language to specify the primitive semantics (close to the gascon specification)
- Developped tools
  - to automatically generate C macro code from the specification
  - to verify manual/automatic code with respect to the specification

Static analysis of communicating imperfectly clocked redundant units



Specification : no alarm raised with a normal input



input stability $< \Delta$: counter-example	Between $\frac{2}{3} \times \Delta$ and Δ : ?	input stability $> \Delta$: the analyzer proves the specification

8. Projects

Static Analysis of Parallel Code

– Some applications make use of parallel code

– Extremely hard to verify, whichever method is chosen!

– Work on static analysis of parallel code is in progress.

Conclusion

– Vision: to understand the numerical world, different levels of abstraction must be considered

– Theory: abstract interpretation ensures the coherence between abstractions and offers effective approximation techniques to cope with infinite systems

– Applications: the choice of effective abstraction which are coarse enough to be *computable* and precise enough to be *avoid false alarms* is central to master undecidability and complexity in model and program verification

– Software engineering : Manual validation by control of the software design process will ultimately be complemented by the verification of the final product

THE END

Thank you for your attention

9. Bibliography

Short bibliography

[BCC+03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proceedings of the ACM SIGPLAN '2003 Conference on Programming Language Design and Implementation (PLDI)*, pages 196–207, San Diego, California, United States, 7–14 June 2003. ACM Press, New York, New York, United States.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, New York, United States.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, New York, United States.

[CC92] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the Fourth International Symposium on Programming Language Implementation and Logic Programming, PLILP '92*, Leuven, Belgium, 26–28 August 1992, Lecture Notes in Computer Science 631, pages 269–295. Springer, Berlin, Germany, 1992.

[CC00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Massachusetts, United States, January 2000. ACM Press, New York, New York, United States.

[CCF+07] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Varieties of static analyzers: A comparison with ASTRÉE, invited paper. In M. Hinchey, He Jifeng, and J. Sanders, editors, *Proceedings of the First IEEE & IFIP International Symposium on Theoretical Aspects of Software Engineering, TASE '07*, pages 3–17, Shanghai, China, 6–8 June 2007. IEEE Computer Society Press, Los Alamitos, California, United States.

[CCF+08] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the ASTRÉE static analyzer. In M. Okada and I. Satoh, editors, *Eleventh Annual Asian Computing Science Conference, ASIAN 06*, pages 272–300, Tokyo, Japan, 6–8 December 2006, 2008. Lecture Notes in Computer Science 4435, Springer, Berlin, Germany.

[DS07] D. Delmas and J. Souyris. ASTRÉE: from research to industry. In G. Filé and H. Riis-Nielson, editors, *Proceedings of the Fourteenth International Symposium on Static Analysis, SAS '07*, Kongens Lyngby, Denmark, Lecture Notes in Computer Science 4634, pages 437–451. Springer, Berlin, Germany, 22–24 August 2007.