Automatic Verification of Embedded Control Software with ASTRÉE and beyond

Patrick Cousot

Jerome C. Hunsaker Visiting Professor
Department of Aeronautics and Astronautics, MIT
cousot@mit.edu www.mit.edu/~cousot

École normale supérieure, Paris
cousot@ens.fr www.di.ens.fr/~cousot

Workshop on Critical Research Areas in Aerospace Software
Aero. Astro. Dept., MIT, August 9th, 2005

State of Practice

An example among many others (Matlab code)

```
> h=get(gca,'children');

apple.awt.EventQueueExceptionHandler Caught Throwable : java.lang.ArrayIndexOutOfBoundsException: 2 >= 2
java.lang.ArrayIndexOutOfBoundsException: 2 >= 2
at java.util.Vector.elementAt(Vector.java:431)
at com.mathworks.mde.help.IndexItem.getFilename(IndexItem.java:100)
at com.mathworks.mde.help.Index.getFilenameForLocation(Index.java:706)
at com.mathworks.mde.help.Index.access$3100(Index.java:29)
at com.mathworks.mde.help.Index$IndexMouseMotionAdapter.mouseMoved(Index.java:768)
at java.awt.AWTEventMulticaster.mouseMoved(AWTEventMulticaster.java:272)
at java.awt.AWTEventMulticaster.mouseMoved(AWTEventMulticaster.java:271)
at java.awt.Component.processMouseMotionEvent(Component.java:5211)
at javax.swing.JComponent.processMouseMotionEvent(JComponent.java:2779)
at com.mathworks.mwswing.MJTable.processMouseMotionEvent(MJTable.java:725)
at java.awt.Component.processEvent(Component.java:4967)
at java.awt.Container.processEvent(Container.java:1613)
at java.awt.Component.dispatchEventImpl(Component.java:3681)
at java.awt.Container.dispatchEventImpl(Container.java:1671)
at java.awt.Component.dispatchEvent(Component.java:3543)
at java.awt.LightweightDispatcher.retargetMouseEvent(Container.java:3527)
at java.awt.LightweightDispatcher.processMouseEvent(Container.java:3255)
at java.awt.LightweightDispatcher.dispatchEvent(Container.java:3172)
at java.awt.Container.dispatchEventImpl(Container.java:1657)
at java.awt.Window.dispatchEventImpl(Window.java:1606)
at java.awt.Component.dispatchEvent(Component.java:3543)
at java.awt.EventQueue.dispatchEvent(EventQueue.java:456)
at java.awt.EventDispatchThread.pumpOneEventForHierarchy(EventDispatchThread.java:234)
at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:184)
at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:178)
at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:170)
at java.awt.EventDispatchThread.run(EventDispatchThread.java:100)
>
```


The software challenge for next 10 years

– Present-day software engineering is almost exclusively manual, with very few automated tools;
– Trust and confidence in specifications and software can no longer be entirely based on the development process (e.g. DO178B);
– In complement, quality assurance must be ensured by new design, modeling, checking, verification and certification tools based on the product itself.

State of the Art in Automatic Static Program Analysis

Static analysis tools

– Determine automatically from the program text program properties of a certain class that do hold at runtime (e.g. absence of runtime error);
– Based on the automatic computation of machine representable abstractions [1] of all possible executions of the program in any possible environment;
– Scales up to hundreds of thousands lines;
– Undecidable whence false alarms are possible [2]

[1] sound but (in general) uncomplete approximations.
[2] cases when a question on the program runtime behavior cannot be answered automatically for sure

Degree of specialization

– Specialization for a class of runtime properties (e.g. absence of runtime errors)
– Specialization for a programming language (e.g. PolySpace Suite for Ada, C or C++)
– Specialization for a programming style (e.g. C Global Surveyor)
– Specialization for an application type (e.g. ASTRÉE for embedded real-time synchronous [3] autocodes)

) The more specialized, the less false alarms [4]!

[3] deterministic
[4] but the less specialized, the larger commercial market (and the less client satisfaction)!

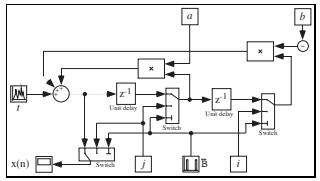
The ASTRÉE static analyzer

– ASTRÉE is a static program analyzer aiming at proving the absence of Run Time Errors (started Nov. 2001)
– C programs, no dynamic memory allocation and recursion
– Encompass many (automatically generated) synchronous, time-triggered, real-time, safety critical, embedded software
– automotive, energy and aerospace applications

) e.g. No false alarm on the electric flight control codes for the A340 (Nov. 2003) and A380 (Nov. 2004) generated from SAO/SCADE.

Ellipsoid Abstract Domain for Filters

2^d Order Digital Filter:

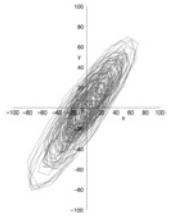


– Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$

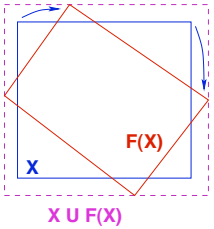
– The concrete computation is bounded, which must be proved in the abstract.

– There is no stable interval or octagon.

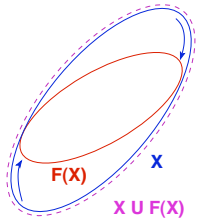
– The simplest stable surface is an ellipsoid.



execution trace



unstable interval



stable ellipsoid

Filter Example

```
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; }
}
```

Reference

see http://www.astree.ens.fr/

Arithmetic-geometric progressions

– Abstract domain: $(\mathbb{R}^+)^5$ [5]

– Concretization (any function bounded by the arithmetic-geometric progression):

$\gamma(\{(\alpha_0, \ldots, \alpha_4) \in (\mathbb{R}^+)^5 \mid \ldots\})(N, R)$

$\gamma(M; a; b; a^0; b^0) =$

$\{f \mid \forall k \in \mathbb{N} : |f(k)| \leq \left(-x \mapsto ax + b \circ (-x \mapsto a^0 x + b^0)^k\right)(M)\}$

Reference

see http://www.astree.ens.fr/

[5] here in $\mathbb{R}$

Arithmetic-Geometric Progressions (Example 1)

```
% cat count.c
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
volatile BOOLEAN I; int R; BOOLEAN T;
void main() {
  R = 0;
  while (TRUE) {
    __ASTREE_log_vars((R));
    if (I) { R = R + 1; }
    else { R = 0; }
    T = (R >= 100);
    __ASTREE_wait_for_clock(());
  }}
```

potential overflow!

```
% cat count.config
__ASTREE_volatile_input((I [0,1]));
__ASTREE_max_clock((3600000));
% astree –exec-fn main –config-sem count.config count.c|grep '|R|'
|R| <= 0. + clock *1. <= 3600001.
```

Arithmetic-geometric progressions (Example 2)

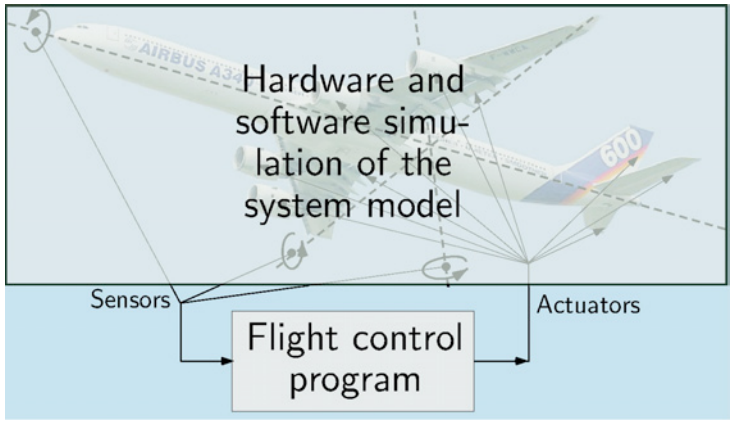
```
% cat retro.c
typedef enum {FALSE=0, TRUE=1} BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

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(());
  }}
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));
|P| <= (15.  + 5.87747175411e-39
/ 1.19209290217e-07) * (1
+ 1.19209290217e-07)^clock
- 5.87747175411e-39 /
1.19209290217e-07 <=
23.0393526881
```


Towards System Verification Tools

Computer controlled systems

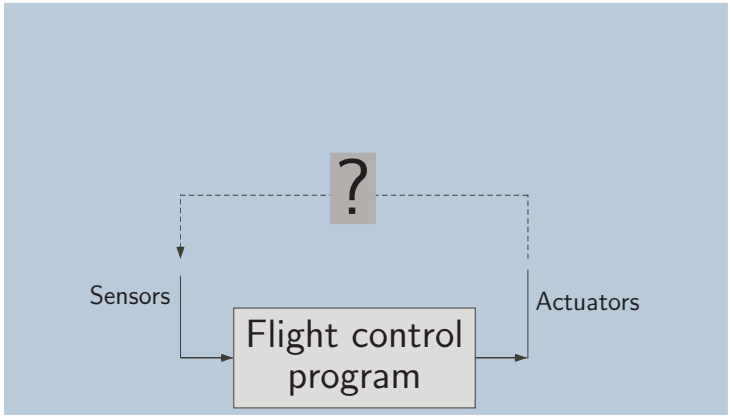
Software test



Abstractions: program ! none, system ! precise

– Very expensive
– Not exhaustive
– Extended during flight test period
– Late discovery of errors can delay the program by months (the whole software development process must be rechecked)

Software analysis & verification with ASTRÉE

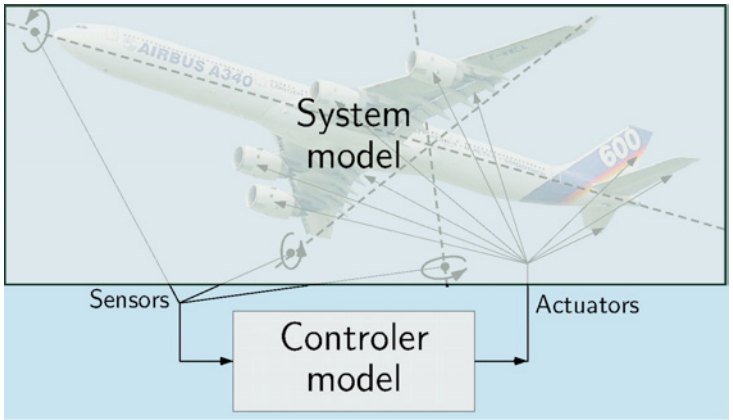


Abstractions: program ! precise, system ! coarse

– Exhaustive
– Can be made precise by specialization[6] to get no false alarm
– No specification of the controlled system (but for ranges of values of a few sensors)
– Impossible to prove essential properties of the controlled system (e.g. controlability, stability)

[6] To specific families of properties and programs

System analysis & verification by control engineers

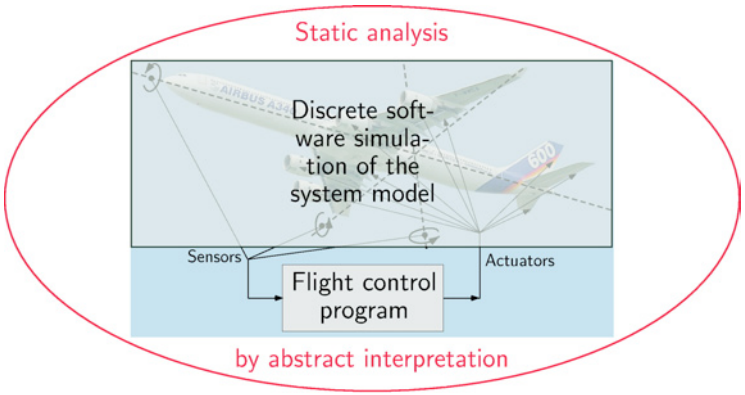


Abstractions: program ! imprecise, system ! precise

– The controler model is a rough abstraction of the control program:
  - Continuous, not discrete
  - Limited to control laws
  - Does not take into account fault-tolerance to failures and computer-related system dependability.
– In theory, SDP-based search of system invariants (Lyapunov-like functions) can be used to prove reachability and inevitability properties
– Problems to scale up (e.g. over long periods of time)
– In practice, the system/controler model is explored by discrete simulations (testing)

Exploring new avenues in static analysis

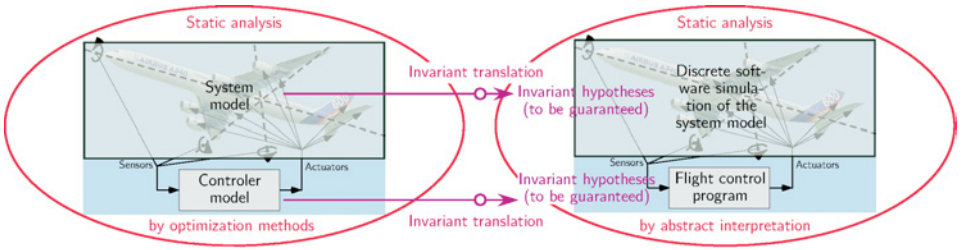
System analysis & verification, Avenue 1



Abstractions: program ! precise, system ! precise

– Exhaustive (contrary to current simulations)
– Traditional abstractions (e.g. polyhedral abstraction with widening) seem to be too imprecise
– Currently exploring new abstractions (issued from control theory like ellipsoidal calculus using SDP)
– Prototype implementation in construction!

System analysis & verification, Avenue 2

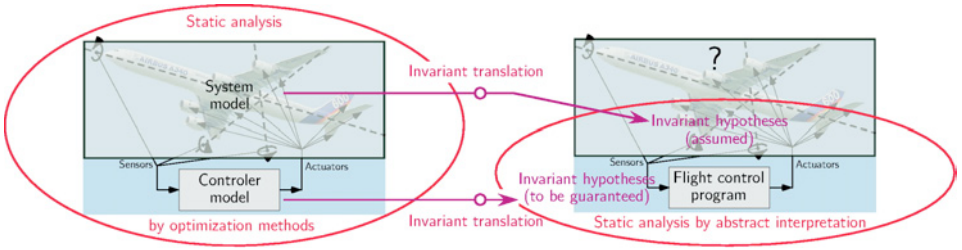


Abstractions: program ! precise, system ! precise

– Example of invariant translation: ellipsoidal ` ! polyhedral [7]
– The static analysis is easier on the system/controller model using continuous optimization methods
– The translated invariants can be checked for the system simulator/control program (easier than invariant discovery)
– Should scale up since these complex invariants are relevant to a small part of the control program only

[7] For which floating point computations can be taken into account

System analysis & verification, Avenue 3



Abstractions: program ! precise, system ! precise

– The invariant hypotheses on the controlled system are assumed to be true
– It remains to perform the control program analysis under these hypothesis
– The results can then be checked on the whole system (as in case 2, but now using refined invariants on the control program!)
– Iterating this process leads to static analysis by refinement of specifications

Conclusion

Scientific and technologic objective

To develop formal tools to answer questions about software:

– from control model design to software implementation,

– for a wide range of design and software properties,

which would be general enough to benefit all software-intensive industries, and can be adapted to specific application domains.

Research on software safety and security

– Investing $\frac{1}{10000}$ or even less of the software costs in research is far from sufficient

– A sustained effort of 1 to 3% would be more realistic and could significantly contribute to progress in the 10 forthcoming years.