Automatic verification of safety critical softwares

Xavier Rival

INRIA Paris Rocquencourt

Nov, 8th. 2012

Outline

- **Potential impact of bugs in safety critical softwares**:
  - **disastrous**, **not theoretical**
- **State of the art in industry**:
  - mostly **testing**, need for **better techniques**
- **Abstract interpretation based static analysis**:
  - **sound**, **automatic**
  - successful **verification** of synchronous softwares
- **towards the verification of wider families of softwares**

verification of programs
manipulating complex data-structures

The Ariane 501 flight failure (1996)

- **The failure**:
  - at $T_0 + 30\ s$, an **arithmetic overflow** (`float -> short int`) both Inertial Reference Systems to return **negative error codes**
  - the on-board computer **misinterprets** those as physical data
  - **loss of control** of the trajectory
- A **long list of design issues**:
  1. failure to assess the **range of inputs**: reuse of legacy code
  2. wrong settings of **hardware interruptions**: crash the system !
  3. the faulty computation was **useless** after takeoff...
  4. main and back-up systems running the **same** faulty software
- **A very expensive failure: more than $ 300 000 000 cost**

Issues in critical embedded softwares

Ariane 501 flight is not the only occurrence:

- **Patriot missile Dahran failure**:
  - **imprecisions in fixed-point computation** (0.1 not representable)
  - 28 fatalities
- **Loss of a Mars explorer vehicle**:
  - wrong use of **units**: no conversion between meters and yards
  - crash on the surface of Mars
- **Saab Grippen fighter jet**:
  - **unstability** issues in control sofwares
  - two crashes, due to "Pilot Induced Oscillations"
- Many others...

State of the art in industry

Defined per area, "good industrial practices":

- **DO 178 standards in avionics**:
  1. assess **level of criticality**

flight-by-wire	level A	highly critical
flight warning system	level C	medium
passenger IFE	level E	irrelevant

  2. address **qualification requirements** depending on criticality level
- **Examples of certification tasks**
  - **documentation**, traceability of software
  - **testing**, from unit testing to iron bird

- **Expensive** processes; e.g., test: about 90 % of the cost
- **No guarantee of safety**, test does not cover all executions

The undecidability barrier

Automatic verification is a very desirable goal

Cheaper, better guarantee on software...

- **Absence of runtime errors**
  e.g., no crashes on arithmetic or memory errors
- **Functional properties**
  e.g., the program transmits accurate orders to actuators

But interesting semantic properties are all undecidable

when onsidering Turing complete languages

- Proof **by reduction to the halting problem**

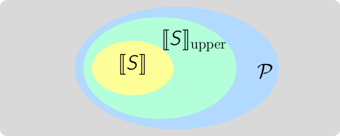
Static analysis and verification

Verification using abstraction

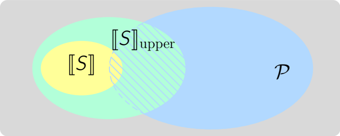
- Retain **only relevant properties** of the concrete semantics
  Derive a **computable, abstract semantics**
- **Sound**: forgets no concrete behavior
- **Generally incomplete**: may fail to capture desired properties

Example: attempt to verify that **semantics** $[\![S]\!]$ satisfies **property** $\mathcal{P}$
using over-approximate semantics $[\![S]\!]_{\mathrm{upper}}$

Successful verification:



Unsuccessful analysis:

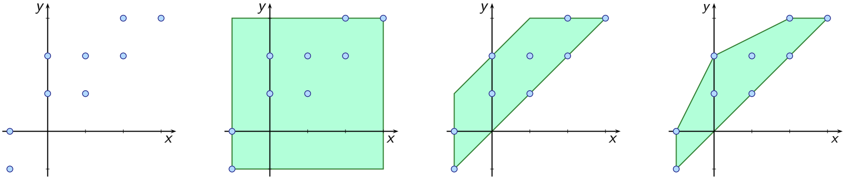


Abstraction of properties

Abstract domains

- Families of **abstract predicates** adapted to static analysis
- **Compact** and **efficient** representations
- **Operations** for the static analysis of concrete operations

Example: abstraction of **sets of pairs of integers**



concrete set | interval domain | octagon domain | polyedra domain

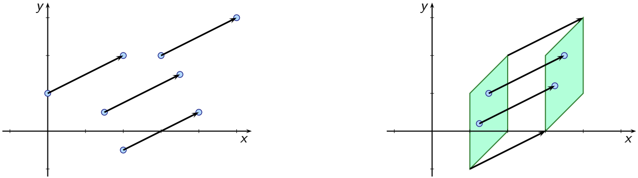
In static analyses: various cost / precision ratios

Abstraction of execution steps

Computing sound abstract transformer

- **Conservative analysis** of concrete execution steps in the abstract e.g., **assignments**, **condition tests**...
- May **lose precision**, will **never forget any behavior**
- Balance between **cost** and **precision**

Example: analysis of a **translation** with **octagons**



concrete transformation — abstract transformation

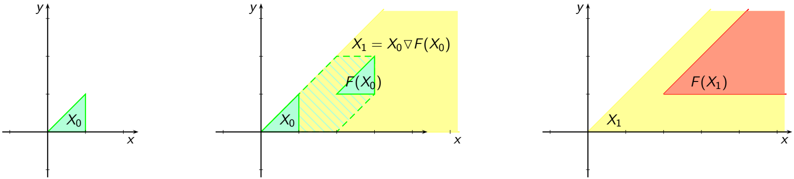
Soundness: all concrete behaviors are accounted for !

Abstraction of infinite computations

Computing invariants about infinite executions with widening ∇

- **Loops** may induce executions of **unbounded length**
- Analyses should compute **inductive invariants**
- **Widening** ∇ over-approximates $\cup$: **soundness guarantee**
- **Widening** ∇ guarantees the **termination of the analyses**

Example: iteration of the translation $(2, 1)$, with **octagons**



initial

iteration 1

iteration 2:
stable !

Soundness: all concrete behaviors are accounted for !

The Astrée analyzer

Goal: verify the absence of runtime errors in synchronous embedded softwares

- **Answer**: **domain specific** static analyzer
- **Group**:
  Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, Xavier Rival

```
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
```

Characteristics:

- **huge softwares**: around 1 MLOC
- **huge states**: $\approx$ 50 000 variables
- **complex algorithms**:
  boolean control, digital filtering, interpolations...
- **very hard to verify**

A numerical abstraction: octagons

An invariant **to prove** in the **analysis of a real system**:

```
assume(x ∈ [−10, 10])
if(x < 0)
    y = −x;
else
    y = x;
①if(y ≤ 5)
    ②assert(−5 ≤ x ≤ 5);
```

Relation between x, y needed

Relational numerical invariants

- **Convex polyedra**:

$$\bigvee_i \left(\sum_j \alpha_{ij} x_j \leq \beta_i \right)$$

  high computational cost
- **Octagons** (A. Miné):
  - two variables per inequality
  - $\alpha_{ij} \in \{-1, 0, 1\}$
  - **reasonable cost**

At ①: $\begin{cases} & 0 \leq \mathbf{y} - \mathbf{x} \leq 10 \\ \wedge & 0 \leq \mathbf{x} + \mathbf{y} \leq 20 \end{cases}$

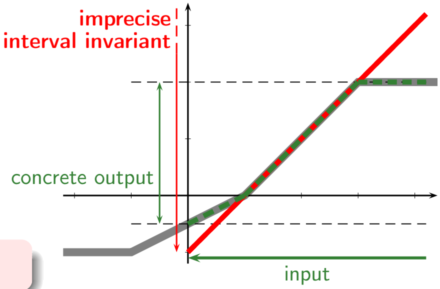
At ②: $\begin{cases} & 0 \leq \mathbf{y} - \mathbf{x} \leq 10 \\ \wedge & 0 \leq \mathbf{x} + \mathbf{y} \leq 20 \\ \text{thus} & -5 \leq \mathbf{x} \leq 5 \end{cases}$

A symbolic abstraction: trace partitioning

An **interpolation** routine to **analyze precisely**:

```
assume(x ≥ 0);
int i = 0;
while(i < n && t0[i] ≤ x)
    i = i + 1;
y = ((x − t0[i]) ⋆ t1[i] + t2[i]);
```

Disjunctions needed



Disjunctions in static analysis

- Can be **very costly**, if too many disjuncts
- **Trace partitioning**: **link states to control history** (L. Mauborgne, X. Rival)

- With no partitioning: $\mathbf{y} \geq -1$
- With partitioning: $\mathbf{y} \in [-0.5, 2]$

$$\begin{cases} 1 \text{ iter} \Rightarrow \mathbf{y} \in [-0.5, 0] \\ 2 \text{ iters} \Rightarrow \mathbf{y} \in [0, 2] \\ 3 \text{ iters} \Rightarrow \mathbf{y} \in [2, 2] \end{cases}$$

Results

Practical results

Proof of safety of industrial codes

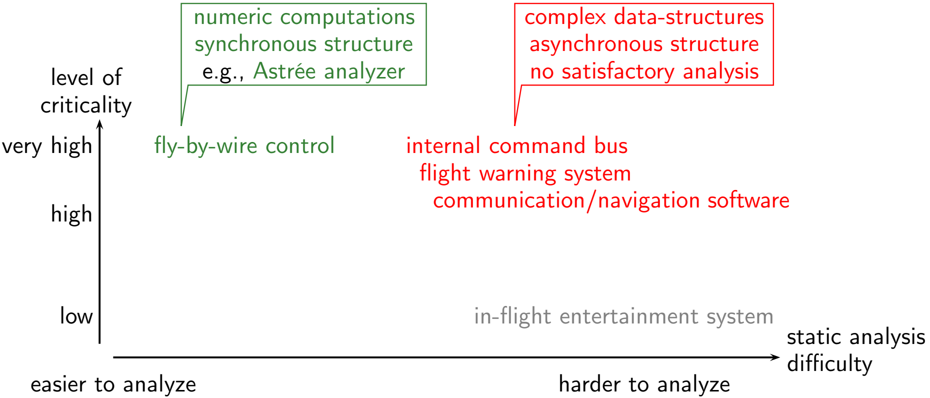
Airbus A 340 FBW	70 kLOC	1h30	400 Mb	**0 alarm**
Airbus A 380 FBW	700 kLOC	12h	2 Gb	**0 alarm**

Industrialized by AbsInt since 2009

- Customers in **avionics**, **automotive**, **embedded systems**
- **Continued research effort**, driven by industrial examples:
  - new abstract domains
  - new analysis techniques
  - ...

Theoretical results: better understanding of static analysis techniques, **combination of many abstract domains**

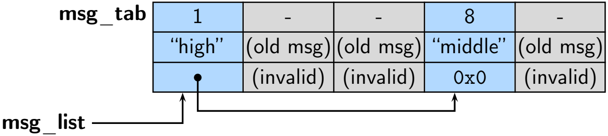
Towards the verification of wider families of softwares



- Many families of softwares **not addressed** by Astrée
- Significant issues to analyze them: **asynchrony**, **memory properties**

An example taken from a flight warning system

- **Cockpit application**, reports **aircraft systems status**
- **Static** message descriptors, **dynamically** linked at runtime




```
typedef struct msg{
    int prio;                  message priority
    char * txt;                warning content
    struct msg * next;         dynamic link
} msg;
msg[] msg_tab;                 statically allocated region
msg * msg;                     list of active messages
```

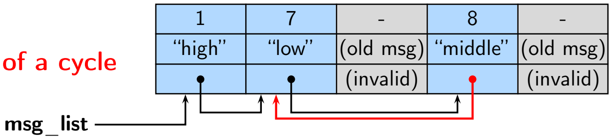
Possible sources of errors and consequences

Insertion of a message report (e.g., engine failure report):

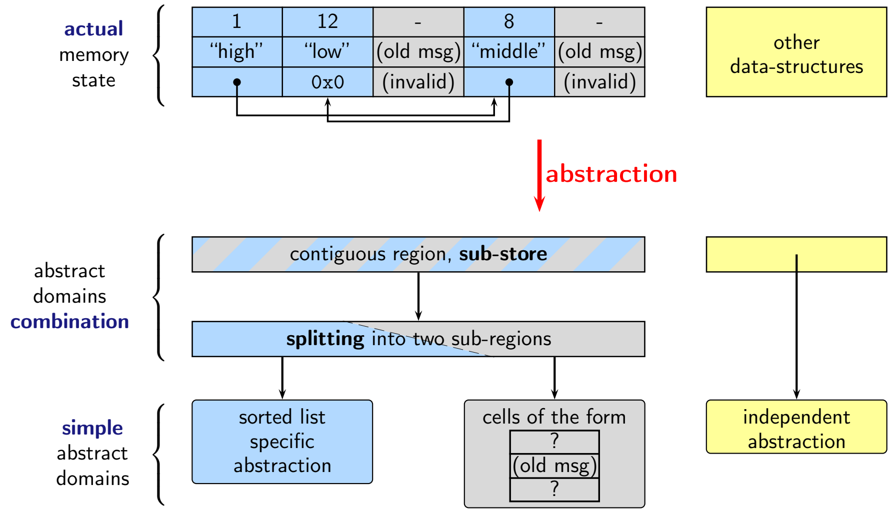
```
void insert(msg ⋆ m){
    msg ⋆ prev = search_pos(m);
    msg ⋆ e = find_empty_cell();          possible data corruption if not empty
    if(e ≠ NULL) update(m, e, prev);...          complex pointer operations
}

msg ⋆ search_pos(msg ⋆ m){
    msg ⋆ c = msg_list;
    while(c ≠ NULL && c → prio < m → prio)         non termination if cycle
        c = c → next;                          abrupt crash if dangling pointer
    return c;                                   improper order if not sorted
}
```

presence of a cycle



MemCAD ERC approach: design modular abstractions !

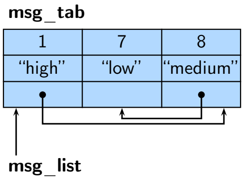


Hierarchical memory abstraction

Principle: use two memory abstractions (P. Sotin, X. Rival)

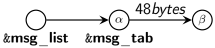
- **Main memory abstraction**: array contents = one value v
- **Sub-memory abstraction**: considers v a memory state

Concrete:

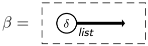


Abstract:

- **Main memory**:



- **Sub-memory**:



Analysis primitives:

- assignment
- test
- widening
- ...

$\Rightarrow$ modular as well !

Other **combination operators** and **domains**:

- **Predicates conjunctions**: **reduced product**
- **Array abstraction**

Open problems in program verification

Good results obtained despite undecidability
Real applications certified safe !

A lot of research still to be done:

- Verifying **complex data-structures manipulations**
- Taking into account **complex assumptions about the environment**
- Verifying **asynchronous softwares**
- Proving **functional properties**
- ...