Coming out of the niche?

David Monniaux

CNRS / VERIMAG
A joint laboratory of CNRS and Université Joseph Fourier (Grenoble).

15-16 November 2010

Yours truly

Situation

- Research associate at CNRS, VERIMAG laboratory, Grenoble
- VERIMAG is specialized in embedded systems, safe programming languages and verification techniques
- Part-time associate professor at École polytechnique, Paris

My interests

Former

Among other interests: co-designer and co-developer of the Astrée static analyzer.

Astrée proves the absence of runtime errors in critical C programs (e.g. Airbus fly-by-wire)

Current

- Decision procedures, quantifier elimination
- Use of floating-point and operation research techniques for exact results
- Alternatives to Kleene iteration for invariant inference (e.g. policy iteration)
- Modular static analysis

1 Short self-presentation

2 Static analysis a success?
- Different kinds of embedded systems
- Critical systems

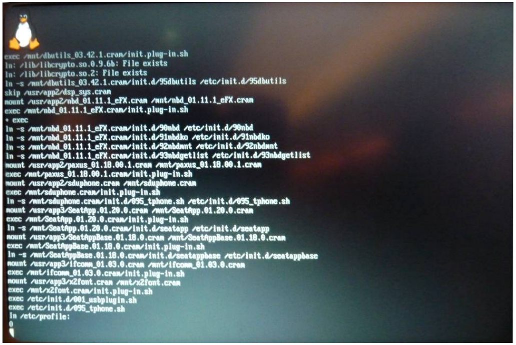
3 Static analysis on normal programs?

4 The End

Public phone blue screen

A public telephone running Microsoft Windows is experiencing a "blue screen of death".

Passenger entertainment system rebooting

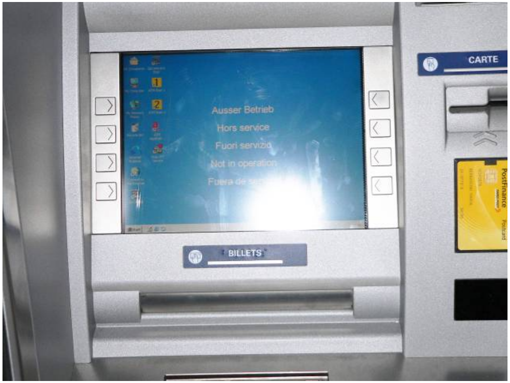


```
exec /mnt/dbutils_03.42.1.cram/init.plug-in.sh
ln: /lib/libcrypto.so.0.9.6b: File exists
ln: /lib/libcrypto.so.2: File exists
ln -s /mnt/dbutils_03.42.1.cram/init.d/95dbutils /etc/init.d/95dbutils
skip /usr/app2/dsp_sys.cram
mount /usr/app2/nbd_01.11.1_eFX.cram /mnt/nbd_01.11.1_eFX.cram
exec /mnt/nbd_01.11.1_eFX.cram/init.plug-in.sh
+ exec
ln -s /mnt/nbd_01.11.1_eFX.cram/init.d/90nbd /etc/init.d/90nbd
ln -s /mnt/nbd_01.11.1_eFX.cram/init.d/91nbdko /etc/init.d/91nbdko
ln -s /mnt/nbd_01.11.1_eFX.cram/init.d/92nbdmnt /etc/init.d/92nbdmnt
ln -s /mnt/nbd_01.11.1_eFX.cram/init.d/93nbdgetlist /etc/init.d/93nbdgetlist
mount /usr/app2/paxus_01.18.00.1.cram /mnt/paxus_01.18.00.1.cram
exec /mnt/paxus_01.18.00.1.cram/init.plug-in.sh
mount /usr/app2/sduphone.cram /mnt/sduphone.cram
exec /mnt/sduphone.cram/init.plug-in.sh
ln -s /mnt/sduphone.cram/init.d/095_tphone.sh /etc/init.d/095_tphone.sh
mount /usr/app3/SeatApp.01.20.0.cram /mnt/SeatApp.01.20.0.cram
exec /mnt/SeatApp.01.20.0.cram/init.plug-in.sh
ln -s /mnt/SeatApp.01.20.0.cram/init.d/seatapp /etc/init.d/seatapp
mount /usr/app3/SeatAppBase.01.18.0.cram /mnt/SeatAppBase.01.18.0.cram
exec /mnt/SeatAppBase.01.18.0.cram/init.plug-in.sh
ln -s /mnt/SeatAppBase.01.18.0.cram/init.d/seatappbase /etc/init.d/seatappbase
mount /usr/app3/ifcomm_01.03.0.cram /mnt/ifcomm_01.03.0.cram
exec /mnt/ifcomm_01.03.0.cram/init.plug-in.sh
mount /usr/app3/x2font.cram /mnt/x2font.cram
exec /mnt/x2font.cram/init.plug-in.sh
exec /etc/init.d/001_usbplugin.sh
exec /etc/init.d/095_tphone.sh
ln /etc/profile:
0
```

TAM Airbus A330 passenger entertainment runs Linux... if a bug occurs, as the stewardess to reboot it.

Linux reboot complete with syntax errors in scripts !

A crashed automatic teller machine



The automatic teller machine software has crashed, revealing a Windows desk.
What if in the middle of a transaction?

1996: Ariane 501

Before

After

This event raised great awareness of the consequences of bugs in critical systems.

MIM-104 Patriot

1991, Dhahran, Saudi Arabia:

A Patriot anti-missile missile, due to a floating-point roundoff bug, fails to intercept a Scud missile.

The Scud falls on barracks, 28 soldiers killed.

Fly-by-wire controls: Airbus A380

Exemplified by sidesticks: commands from pilots get sent to computers, which control the active surfaces.
Critical system

Fly-by-wire controls: Boeing 777-200ER

The control yoke is "fake": it is not mechanically tied to the active surfaces, but to a computer.

Critical system

1 Short self-presentation

2 Static analysis a success?
- Different kinds of embedded systems
- Critical systems

3 Static analysis on normal programs?

4 The End

A definition?

Critical systems, if they dysfunction, may cause

- significant harm to humans, even death
- significant economic losses, or social or environmental disruption

Examples:

- fly-by-wire controls in aircraft
- nuclear power plant safety systems
- surgical “robots”, radiation therapy machines
- in the future: brake-by-wire, steering-by-wire in cars?

Typical obligations

1. the system never produces runtime errors (division by zero, array access out of bounds)
2. the system implements all the functionality in the specification
3. the system does not implement more functionality (no Easter eggs)
4. the system answers within certain delay, even in the worst case

Point 1 handled by Astrée, point 4 handled by Absint aIT.

Points 2 and 3 not well handled but:

- Hoare-like proofs of individual functions instead of unit testing, using Caveat then Frama-C (CEA)
- Certified compilation (INRIA: Xavier Leroy)

Why it is hard

Control theory maths

Control systems contain floating-point code.
They implement complex digital filters.
Absence of overflows derives from mathematical stability of the filters.
So the analyzer has to know about filtering techniques.

Size

Sizes of 300–500 kLOC of C. Not your typical academic toy program.

Why it is easy

Code must be demonstrably safe. All "unsafe" programming practices are banned.

Coding rules make it easy for you

Because of certification requirement:

- No recursion
- No dynamic allocation
- No data structures beyond arrays
- Few function pointers
- Static scheduling, no multithreading
- Object code must be easily traceable to source code, thus very simple control-flow

To summarize

Current critical avionic code does not have all these annoying features of "normal" code:

- dynamic allocation
- virtual functions
- exceptions
- complicated data structures
- dynamic loading
- dynamic reconfiguration

A basic reason is that such code is absolutely not flexible: it is designed for one specific hardware, one specific usage, without possibility to add or subtract components.

Use of high-level languages

Control code is automatically generated from block-diagram specifications.

- Scade (industrial version of Lustre, from Verimag)
- Simulink

Computation primitives implemented by hand in C, boilerplate / data flow code around generated from such languages.
Analysis adapted to code generator idioms.

Difficulties ahead

Developers are already using:

- modern architectures (deep pipelines, complex caches)
- busmaster DMA, intelligent controllers

They may wish to use:

- multicore
- high-level “modern” programming languages (Java)
- object-oriented code, virtual methods
- artificial intelligence?

Abstract interpretation

- Works best when specialized on application domain (e.g. Astrée)
- Works less well in general, if sound (e.g. PolySpace)
- Is fully automatic
- When sound, does not deal well with complicated control flow or heap structures
- Generally limited to simple properties

Program proofs

- Tedious proofs
- Tediousness can be reduced by pre-analysis
- More powerful decision procedures can help
- General properties
- Again, difficulties with complicated control flow or heap
- Separation logic for heap?

The future

I've seen the future, baby, it is murder (Leonard Cohen)

Critical control code is very atypical.
Narrow user base, can't support *n* companies.
A niche concern (as per IEEE-754 anecdote).

Need to go to wider classes of code.

- Become unsound, esp. wrt pointers? (e.g. Clouzot, Coverity)
- Better analysis methods?

A plead

Work in static analysis is hampered by lack of common standards / code bases.
In order to run an experiment for a novel feature in static analysis, need a whole architecture.

- Parser / code loader
- Alias code analysis
- Typing

Microsoft has one, but the rest of us generally need to develop from scratch.
Most academic prototypes take toy languages as input.

How about some public, free static analyzer?

A shameless plug

Recruiting a post-doc on the Asopt project
(`http://asopt.inrialpes.fr/`)
Joint project with INRIA and CEA.

Static analysis advanced techniques (optimization from operation research, game-theoretic techniques, etc.)

Ask me for details.

@VERIMAG