LEVERAGING STATIC ANALYSIS IN INDUSTRIAL CONTEXTS
THE SEMANTICS OF SOFTWARE VERIFICATION

- DO-178B/C for aerospace safety
- EN 50128 for rail safety
- ISO 26262 for road vehicles
- CENELEC 61508
- Common Criteria EAL7 for security
APPLYING SOFTWARE VERIFICATION

- Simple “press button” rule enforcement
- Partial runtime error or security flaw detection
- Complete validation against a given specification
A SOURCE CODE ANALYSIS PLATFORM

- Extensible via plug-ins
- Collaborative
- Designed at CEA & Inria
- Semantic analyses
- ANSI / ISO C 99
THE FRAMA-C KERNEL

- Source code front-end
- Specification front-end
- Global state
- Analyzer services: journaling, messaging, printing
LINGUA FRANCA: THE ACSL SPECIFICATION LANGUAGE

- Specification front-end
- Contract-based
- Functional properties
- User-friendly
- Analysis-independent
unsigned int M;
/*@ 
requires \valid (p) && \valid (q);
ensures M == (*p + *q) / 2;
*/
void mean ( unsigned int* p, 
            unsigned int* q ) {
if (*p >= *q )
    M = (*p - *q ) / 2 + *q ;
else
    M = (*q - *p ) / 2 + *p ;
}

- Caller-callee contract
- Callee requires some pre-conditions from the caller
- Callee ensures some post-conditions hold when it returns
ENFORCING CODING STANDARDS WITH FRAMA-C

• **Result** Airbus and Atos have designed the Taster plugin on top of Frama-C to enforce coding standards.

• **Conclusion** Frama-C yields productivity gains and ensures code quality.

**Benefits:**
- Eases code review on syntactic or typing rules.
- Validation of semantic rules:
  - dataflow related rules on variables,
  - runtime errors requiring a value analysis.
SEMANTICAL ANALYSIS

- Automated process
- Integral & pointer ranges
- ACSL verifications
- Runtime-errors threats
- Side-effects & dependency analysis
- Program structure & transformations
- Confidentiality or integrity leaks
CHECKING INTRINSIC FAULTS IN SCADA SYSTEMS

• **Result** Researchers have demonstrated the absence of multiple fault families in safety-critical software.

• **In addition** derived analyses cover structural properties on memory separation and cyclic behaviors.

• **Conclusion** Frama-C enables highly-automated verification runs.

> 100+ kloc
> C source code
> Highest certification requirements
> 80% code coverage
> 200 alarms
DETECTION OF A SECURITY FLAW IN A COTS COMPRESSION LIBRARY

• **Result** CEA researchers identified a bug in the QuickLZ library. This bug was acknowledged by the designer and corrected in version beta 1.5.1.

• **Conclusion** Software analysis can be applied to general-purpose COTS, enabling their use in security-critical systems.

Fixed a condition where QLZ_MEMORY_SAFE could fail detecting corrupted data. Thanks to Pascal Cuoq and Kerstin Hartig who used Frama-C's value analysis!
ADVANCED VALIDATION OF A SET OF HYPERVISOR PROPERTIES

- **Result** Design-time security goals allow to specify and verify custom software properties.

- **Conclusion** A rigorous methodology can include the use of Frama-C to address strong software requirements.
RUNTIME MONITORING AND VERIFICATION

• **Result** Use program analysis and transformations to synthesize:
  - security monitors
  - fault injectors

• **Conclusion** Runtime code can be added to harden legacy software through hardware-enabled runtime verification.

```c
00: extern int a, b;
01: void f(int);

03: void g()
04: {
04:   if (b == 0) a = 1;
05:   else if(b == 1) a = 2;
06:   else return;
08:   assert((a == 1 && b == 0) || (a == 2 && b == 1))
09:   f(a);
10: }
```
% Conflict during interval [B,T]
\[\text{conflict}_2D?(s, v) : \text{bool} = \exists \ (t: \text{Lookahead}) : \text{sqv}(s + t*v) < \text{sq}(D)\]

% 2-D Conflict Detection (cd2d)
\[\text{cd2d}?(s, v) : \text{bool} = \text{horizontal}_\text{los}?(s + B*v) \text{ OR } \omega_v(s)(v) < 0\]

% THEOREM: cd2d is correct and complete
\[\text{cd2d} : \text{THEOREM} \quad \text{conflict}_2D?(s, v) \iff \text{cd2d}?(s, v)\]
Frama-C
A Software Analysis Perspective

Pascal Cuoq, Florent Kirchner, Nikos Karamutas, Vinglo Picronto,
Julien Signoles, and Bartek Yakubowski

with Patrick Baslin, Richard Boulton, Bernard Beguillot, Loїs Carrenon,
Zynteb Dugue, Philippe Hermetac, Benjamin Minet, Yanick Moi,
Anne Pacaud, Armand Pucetti, Muriel Roger, and Nicky William

CEA, LIST, Software Safety Laboratory, PC 174, 91191 Gif-sur-Yvette France

Abstract. Frama-C is a source code analysis platform that aims at conducting verification of industrial-size C programs. It provides its users with a collection of plugins that perform static analysis, deductive verification, and testing, for safety- and security-critical software. Collaboration of the verification across cooperating plugins is enabled by their integration on top of a shared kernel and datastructures, and their compliance to a common specification language. This foundational article presents a synthetic view of the platform, its main and composite analyses, and some of its industrial achievements.

Introduction

In past forty years we have seen much of the groundwork of formal software analysis being laid. Several angles and theoretical avenues have been explored, from deductive reasoning to abstract interpretation to program transformation to static analysis. While much remains to be done from an academic standpoint, one of the major advances in those fields are already being successfully implemented and met with growing industrial interest. The coming out for mainstream diffusion of software analysis techniques has several challenges. Chief among them are: their scalability, their interoperability, and the soundsness of their results.

Pointing predictably important from the point of view of adoptability, scaling a large problem is a prerequisite for the industrial diffusion of software analysis and verification techniques. It also represents a means to better understand how engage teams (e.g., pointers, unions, or dynamic memory allocation) influence and underlying architecture of large software developments. Overall, achieving scalability in the design of software analyzers for a wide range of software patterns remains a difficult question.

This work was partly supported by ANR UCAR and Viorits, and FUI HiLife projects
METHODS AND TOOLS FOR HCS&S

- Scientific roots and community
  - Formal proof
  - Model checking
  - Constraint solving
  - Simulation
  - Abstract interpretation
  - Test case generation
  - Architecture Exploration
  - Synchronous languages

- Prototyping and development of industrial-strength tools
  for academia
  for the industry

- Objectives
  - Raise the level of confidence in software
  - Lower the costs of verification
  - Enable the evaluation of academic prototypes on industrial problems

Technological strategy: creation of collaborative platforms
Scientific strategy: combination of approaches
Applicative strategy: cross-domain fertilization – aero, space, rail, energy, banking, defense

Guided by industrial requirements
Scaling & Performance
SOFTWARE DEVELOPERS

- Industrial support
- Commercial licenses
- Preinstalled workstations

SOFTWARE INTEGRATORS

Off-the-shelf validation kits for common open-source packages

www.trust-in-soft.com

SERVICE PROVIDERS

Dedicated affiliate programs