Frama-C, un analyseur statique de code source : concepts et exemples d'utilisation

Journée CAP'TRONIC

Nikolai Kosmatov

Cergy, November 29th, 2017

Outline

Outline

Frama-C – Historical Context

- 90's: CAVEAT, Hoare logic-based tool for C code at CEA
- 2000's: CAVEAT used by Airbus during certification process of the A380 (DO-178 level A qualification)
- 2002: Why and its C front-end Caduceus (at INRIA)
- 2004: start of Frama-C project as a successor to CAVEAT and Caduceus
- 2008: First public release of Frama-C (Hydrogen)
- 2012: WP: Weakest-precondition based plugin
- 2012: E-ACSL: Runtime Verification plugin
- 2013: CEA Spin-off TrustInSoft
- 2016: Eva: Evolved Value Analysis
- 2016: Frama-Clang: C++ extension
- Today: Frama-C Sulfur (v.16)

Frama-C – Open Source Distribution

Framework for analyses of source code written in ISO 99 C

[Kirchner et al, FAC'15]

- offers ACSL, an ISO/ANSI C Specification Language
- mostly open source (LGPL 2.1)

http://frama-c.com

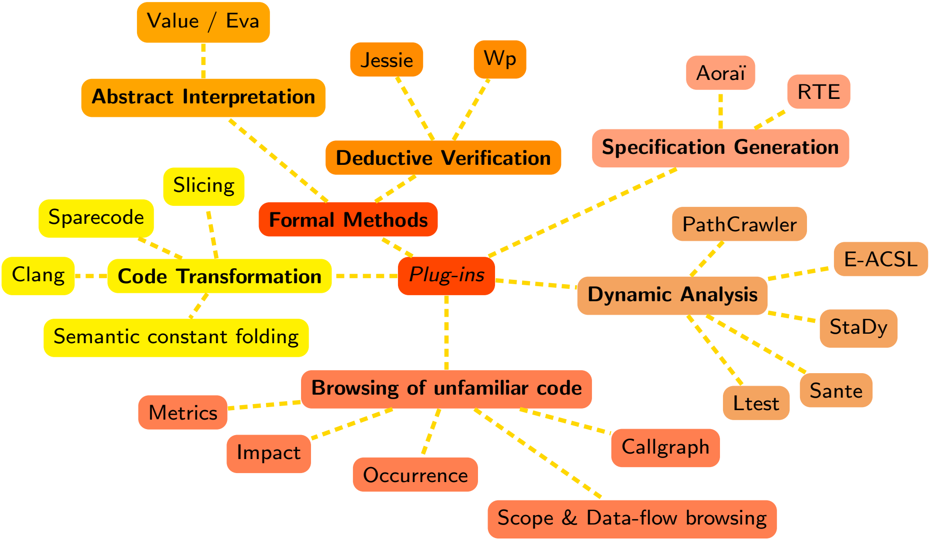
- also proprietary extensions and distributions
- targets both academic and industrial usage

Frama-C – a Collection of Tools

Several tools inside a single platform

- plug-in architecture *à la* Eclipse
- tools provided as plug-ins
  - over 20 plug-ins in the open-source distribution
  - close-source plug-ins, either at CEA (about 20) or outside
- plug-ins connected to a kernel
  - provides an uniform setting
  - provides general services
  - synthesizes useful information

Plug-in Gallery (a selection)



Frama-C – a Development Platform

- developed in OCaml (≈ 180 kloc in the open source distribution, ≈ 300 kloc with proprietary extensions)
- offers a library to develop
  - dedicated plug-ins for specific task (e.g. verifying your coding rules)
  - dedicated plug-ins for fine-grain parameterization
  - extension of existing analyzers

Outline

Objectives of Deductive Verification

Rigorous, mathematical proof of semantic properties of a program

- functional properties
- safety:
  - all memory accesses are valid,
  - no arithmetic overflow,
  - no division by zero, . . .
- termination

ACSL: ANSI/ISO C Specification Language

Presentation

- Based on the notion of contract, like in Eiffel, JML
- Allows users to specify functional properties of programs
- Allows communication between various plugins
- Independent from a particular analysis
- Manual at `http://frama-c.com/acsl`

Basic Components

- Typed first-order logic
- Pure C expressions
- C types + $\mathbb{Z}$ (integer) and $\mathbb{R}$ (real)
- Built-ins predicates and logic functions, particularly over pointers: `\valid(p)`, `\valid(p+0..2)`, `\separated(p+0..2,q+0..5)`, `\block_length(p)`

WP plugin

- Hoare-logic based plugin, developed at CEA List
- Proof of semantic properties of the program
- Modular verification (function by function)
- Input: a program and its specification in ACSL
- Relies on Automatic Theorem Provers
  - Alt-Ergo, Simplify, Z3, Yices, CVC3, CVC4 ...
- WP manual at `http://frama-c.com/wp.html`
- If all properties are proved, the program respects the given specification

Example: a C program annotated in ACSL

```
/*@ requires n>=0 && \valid(t+(0..n-1));
    assigns \nothing;
    ensures \result != 0 <==>
      (\forall integer j; 0 <= j < n ==> t[j] == 0);
*/
int all_zeros(int t[], int n) {
  int k;
  /*@ loop invariant 0 <= k <= n;
      loop invariant \forall integer j; 0<=j<k ==> t[j]==0;
      loop assigns k;
      loop variant n-k;
  */
  for(k = 0; k < n; k++)
    if (t[k] != 0)
      return 0;
  return 1;
}
```

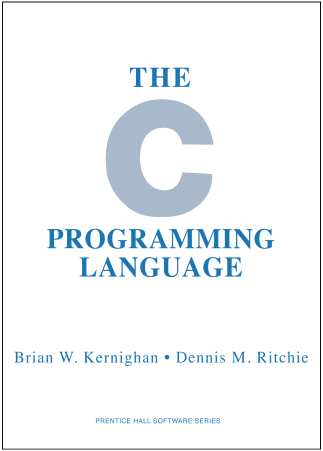
Can be proven with Frama-C/WP

The C language is risky!

- Low-level operations
- Widely used for critical software
- Lack of security mechanisms

Runtime errors are common:

- Division by 0
- Invalid array index
- Invalid pointer
- Non initialized variable
- Out-of-bounds shifting
- Arithmetical overflow
- ...



Safety warnings: arithmetic overflows

Absence of arithmetic overflows can be important to check

- A sad example: crash of Ariane 5 in 1996

WP can automatically check the absence of runtime errors

- Use the command `frama-c-gui -wp -wp-rte file.c`

A Use Case: Verification of memb Module of Contiki OS

Contiki OS at a glance:

- An Open Source OS for the Internet of Things, created in 2003
- More and more commercial products
- Supports many embedded platforms
- http://www.contiki-os.org/

- Continuous integration system does not include formal verification

Overview of the `memb` Module

- No dynamic allocation in Contiki
  - to avoid fragmentation of memory in long-lasting systems
- Memory is pre-allocated (in arrays of blocks) and attributed on demand
- The management of such blocks is realized by the `memb` module

The `memb` module API allows the user to

- initialize a `memb` store (i.e. pre-allocate an array of blocks),
- allocate or free a block,
- check if a pointer refers to a block inside the store
- count the number of allocated blocks

Verification of memb Module

- The `memb` module specified and formally verified with Frama-C/WP
- A few client functions proven as expected
  - Proof fails for out-of-bound access attempts
- A potentially harmful situation detected
  - `count--;` used instead of `count=0;`

Formal verification should be more systematically applied to IoT software to guarantee safety and security.

[Mangano et al, CRISIS 2016]

Outline

Value Analysis Overview

Principle: compute the domains of program variables

- abstract interpretation
- automatic analysis
- correct over-approximation
- alarms for potential invalid operations
- alarms for potential invalid ACSL annotations
- ensures the absence of runtime errors

Value Analysis Parameterization

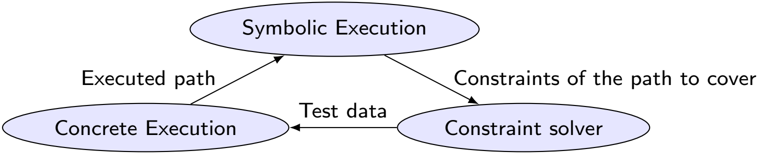
- Value analysis is automatic
- but requires fine-tuned parameterization to be more precise/efficient
- trade-off between time efficiency *vs* memory efficiency *vs* precision

Derived analyses

- results from Value/Eva are useful for other plug-ins
  - domains of values
  - aliasing information
  - dependency information
- program dependency graph (PDG)
  - slicing
  - impact analysis
- domain specific analysis
  - information flow analysis [Assaf et al, SEC'13]
  - concurrency analysis

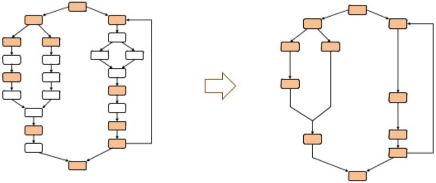
Outline

Plugin PathCrawler for test generation



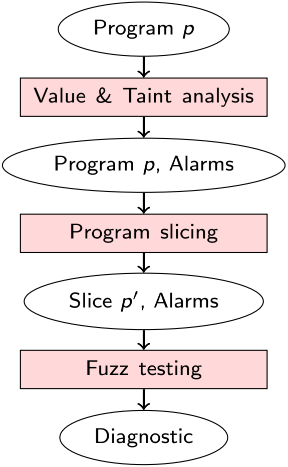
- Performs Dynamic Symbolic Execution (DSE)
- Automatically creates test data to cover program paths [Botella et al. AST 2009]
- Uses code instrumentation, concrete and symbolic execution, constraint solving
- Online version: `pathcrawler-online.com`

Plugin Slicing



- Simplifies the program using control and data dependencies
- Preserves the executions reaching a point of interest (*slicing criterion*) with the same behavior
- Example of slicing criteria: instructions, annotations (alarms), function calls and returns, read and write accesses to selected variables. . .

A Combined Analysis Applied to Security



- Used in EU FP7 project STANCE (CEA LIST, Dassault, Search Lab, FOKUS,...)
- Value analysis to detect alarms
- Taint analysis to identify most security-relevant alarms
- Slicing to reduce the program
- Fuzz testing for efficient detection of vulnerabilities
- Applied to the recent Heartbleed security flaw (2014) in OpenSSL

[Kiss et al., HVC 2015]

Outline

Conclusion

We have presented an overview of :

- the Frama-C toolset
- specification and proof of programs with WP
- verification for absence of runtime errors with EVA
- test generation with PathCrawler
- examples of use cases

All of these and much more inside Frama-C

Frama-C can be used for:

- industrial applications
- teaching
- academic prototyping

http://frama-c.com