frama C
Software Analyzers

Frama-C in a Nutshell

Virgile Prevosto

October 17th, 2013

Frama-C at a glance

- A framework for modular analysis of C code.
- `http://frama-c.com/`
- Developed at CEA LIST and INRIA Saclay (Proval, now Toccata team).
- Released under LGPL license (Fluorine version 3 days ago)
- Kernel based on CIL (Necula et al. – Berkeley).
- ACSL annotation language.
- Extensible platform
  - Collaboration of analysis over same code
  - Inter plug-in communication through ACSL formulas.
  - Adding specialized plug-in is easy

Installation

On Linux

- On Debian, Ubuntu, Fedora, Gentoo, OpenSuse, Linux Mint, ...
- Compile from sources using OCaml package managers:
  - Godi (`http://godi.camlcity.org/godi/index.html`)
  - Opam (`http://opam.ocamlpro.com/`)

On Windows

- Godi
- Wodi (`http://wodi.forge.ocamlcore.org/`)

On Mac OS X

- Binary package available
- Source compilation through homebrew.

Documentation

Manuals

- `http://frama-c.com/support.html`
- In directory
  `$(frama-c -print-share-path)/manuals`

Support

- `frama-c-discuss@gforge.inria.fr`
- tag `frama-c` on `http://stackoverflow.com`

Tutorials

- `http://bts.frama-c.com/dokuwiki/doku.php?id=mantis:frama-c:tutorial`
- Material from presentation in Munich on the sharepoint

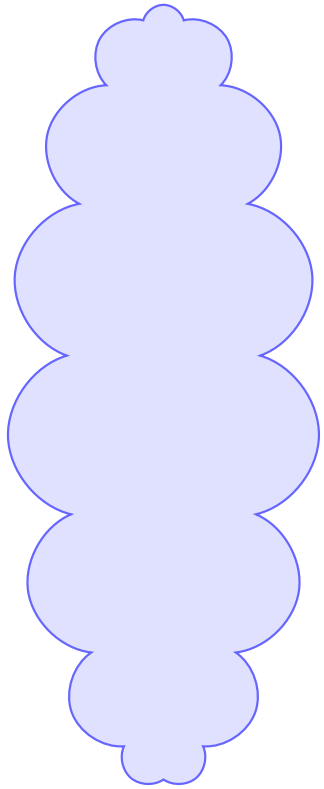
Possible usage in OpenETCS

Code verification

- Absence of run-time error
- Correct with respect to functional specification written in ACSL
- Flow information (input/output variables, functional dependencies)

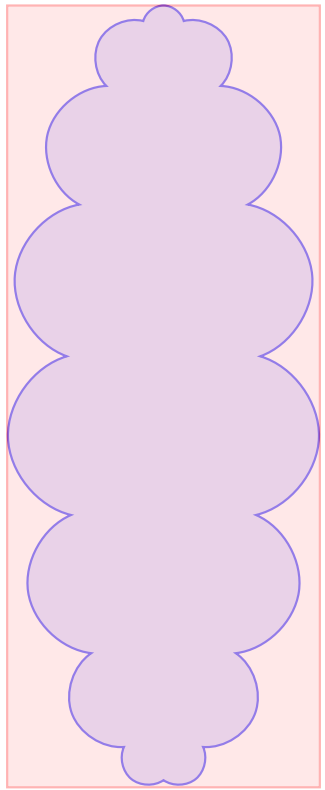
Over-approximations

- Correct analysis: All concrete values are included in the abstraction
  - ✔ don't miss any concrete behavior
  - ✘ Might lead to spurious warning (over-approximation, false alarm)
- Simulate all possible concrete executions...
- ... by propagating abstract values.
- Take the union of values from all possible paths.
- Loops: compute until a fixpoint is reached...
- ... using broader over-approximations (widening) to ensure termination.



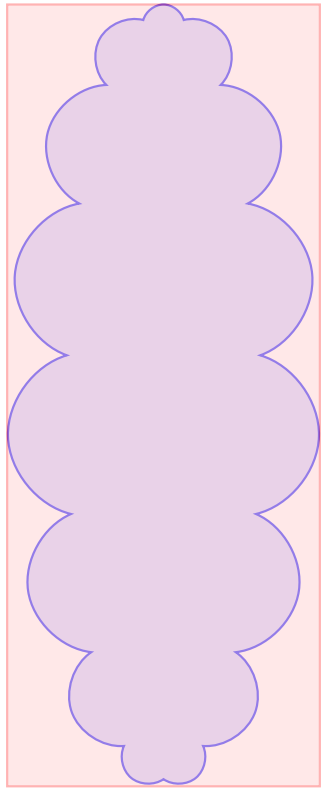
Over-approximations

- Correct analysis: All concrete values are included in the abstraction
  - ✔ don't miss any concrete behavior
  - ✘ Might lead to spurious warning (over-approximation, false alarm)
- Simulate all possible concrete executions...
- ... by propagating abstract values.
- Take the union of values from all possible paths.
- Loops: compute until a fixpoint is reached...
- ... using broader over-approximations (widening) to ensure termination.



Over-approximations

- Correct analysis: All concrete values are included in the abstraction
  - ✔ don't miss any concrete behavior
  - ✘ Might lead to spurious warning (over-approximation, false alarm)
- Simulate all possible concrete executions...
- ... by propagating abstract values.
- Take the union of values from all possible paths.
- Loops: compute until a fixpoint is reached...
- ... using broader over-approximations (widening) to ensure termination.



Value Analysis plug-in

Find the domains of the variables of a program

- based on abstract interpretation
- alarms on operations that may be invalid
- alarms on the specifications that may be invalid
- Correct: if no alarm is raised, no runtime error can occur
- Mostly automated, at least on simple code

Enhance precision

Loops

- option `-ulevel`: syntactic loop unrolling
- option `-slevel`: allows Value to explore *n* separated paths before joining them
- option `-wlevel`: number of loop steps before performing widening (default is 3, use with caution)

Driving Value through Annotations

- ACSL assertions can be used to restrict propagated domains
- but only if Value can interpret it

```
/*@ assert x % 2 == 0; */
// potentially useful
/*@ assert \exists integer y; x == 2 * y; */
// useless
```

- Case analysis using disjunctions

Hoare Logic

```
/*@ requires R;
    ensures E; */
int f(int* x) {


S_1;


S_2;


}
```

- Hoare Triples:

$$\{P\}S\{Q\}$$

- Weakest Preconditions:

$$\forall P, (P \Rightarrow wp(S, Q)) \Rightarrow \{P\}S\{Q\}$$

- Proof Obligation (PO):

$$\mathtt{R} \Rightarrow wp(\mathtt{Body}, \mathtt{E})$$

Hoare Logic

```
/*@ requires R;
    ensures E; */
int f(int* x) {


S_1;


S_2;

/*@assert E; */
}
```

- Hoare Triples:

$$\{P\}S\{Q\}$$

- Weakest Preconditions:

$$\forall P, (P \Rightarrow wp(S, Q)) \Rightarrow \{P\}S\{Q\}$$

- Proof Obligation (PO):

$$\mathtt{R} \Rightarrow wp(\mathtt{Body}, \mathtt{E})$$

Hoare Logic

```
/*@ requires R;
    ensures E; */
int f(int* x) {


S_1;

/*@assert wp(S_2,E); */
S_2;

/*@assert E; */
}
```

- Hoare Triples:

$$\{P\}S\{Q\}$$

- Weakest Preconditions:

$$\forall P,(P \Rightarrow wp(S,Q)) \Rightarrow \{P\}S\{Q\}$$

- Proof Obligation (PO):

$$\mathtt{R} \Rightarrow wp(\mathtt{Body},\mathtt{E})$$

Hoare Logic

```
/*@ requires R;
    ensures E; */
int f(int* x) {

/*@assert
   wp(S_1,wp(S_2,E)); */
S_1;

/*@assert wp(S_2,E); */
S_2;

/*@assert E; */
}
```

- Hoare Triples:

$$\{P\}S\{Q\}$$

- Weakest Preconditions:

$$\forall P,(P \Rightarrow wp(S,Q)) \Rightarrow \{P\}S\{Q\}$$

- Proof Obligation (PO):

$$\mathtt{R} \Rightarrow wp(\mathtt{Body},\mathtt{E})$$

WP plug-in

Main ingredients

- provide precise enough specifications
- require an appropriate context
- explain side-effects of the function

WP commands

- `frama-c -wp -wp-rte file.c`
- WP tab on the GUI
- Inspect (failed) proof obligation
- can be interfaced with automated provers and Coq proof assistant

Current status within OpenETCS

- Formal specification and automated verification of railway software with Frama-C, presented at INDIN'13
- on-going verification of Bitwalker from Siemens (FOKUS with support from CEA LIST)
- analysis of part of ERSA's simulator (FOKUS/CEA LIST, to be started)
- possible development: translation from SysML to ACSL contracts