frama C
Software Analyzers

Towards combining static and dynamic analyses within Frama-C

Julien Signoles

CEA LIST

Software Safety Labs

Hi-Lite Meeting

May 2012, the 29th

cea list

HI-LITE

Simplifying the use of formal methods

frama C
Software Analyzers

Outline

1. what's new in E-ACSL
2. property statuses for combining analyses

Executable Ansi/ISO C Specification Language

http://frama-c.com/eacsl

What is it?

- executable subset of ACSL
- preserve ACSL semantics as much as possible
- compatible with ALFA as much as possible

Which goals?

- runtime assertion checking
- usable by dynamic analyses tools
- usable by static verification tools like Frama-C plug-ins
- verification of mixed ADA/C programs

E-ACSL Plug-in

Reminder

- takes an annotated C program as input
- checks that each annotation belongs to E-ACSL
- returns a new C program
- equivalent to the input
- each annotation is converted into new C statements
- including (at least) one guard
- which fails at runtime if the annotation is wrong

E-ACSL

Highligth

- new release of E-ACSL Reference Manual (version 1.5-4)
- first release of the E-ACSL plug-in (version 0.1)
  - compatible with Frama-C Nitrogen
  - compatible with E-ACSL 1.5-4
  - open source (LGPL v2.1)
  - webpage: http://frama-c.com/eacsl
  - what's currently implemented detailed in `frama-c.com/download/e-acsl/e-acsl-implementation.pdf`
- submitted publication (short paper):
  M. Delahaye, N. Kosmatov and J. Signoles. Towards a Common Specification Language for Static and Dynamic Analysis of C Programs.

Plug-in Current Status

Logic Types

implemented

- C types
- integer
- boolean
- implicit coercions

not yet implemented

- real

No change since the last meeting

Plug-in Current Status

Terms

implemented

- integer constants
- C left values
- arithmetic operators
- casts
- address &
- sizeof
- alignof
- `\null`
- `\at`
- `\result`
- boolean operators
- conditionals _?_:_

under implementation

- `\base_addr`
- `\offset`
- `\block_length`

not yet implemented

- `\true` and `\false`
- bitwise operators
- let binding
- typeof
- t-sets

Plug-in Current Status

Predicates

implemented

- `\true` and `\false`
- relations (==, <=, ...)
- lazy conjunction &&
- lazy disjunction ||
- lazy implication ==>
- negation !
- equivalence <==>
- quantifications over integral types
- `\at`
- conditionals _?_:_

under implementation

- `\valid` *et al.*
- `\initialized`

not yet implemented

- other quantifications
- exclusive or ^^
- let bindings

Plug-in Current Status

Annotations

implemented

- assertions
- function contracts
- statement contracts
- loop invariants

not yet implemented

- behavior-specific annotations
- loop variants
- loop assigns
- global annotations

Plug-in Current Status

Behavior Clauses

implemented

- assumes
- requires
- ensures

not yet implemented

- assigns
- decreases
- abrupt termination
- complete behaviors
- disjoint behaviors

No change since the last meeting

E-ACSL Type System

Main issue of E-ACSL 0.1

- E-ACSL 0.1 uses GMP integers to correctly tranlate integers
- heavy encoding

```
/*@ assert y != 0; */
mpz_t __e_acsl_y;  // declare additional variables
mpz_t __e_acsl;
int __e_acsl_ne;
mpz_init_set_si(__e_acsl_y, y); // allocate GMP's
mpz_init_set_si(__e_acsl, 0);
// do the comparison
__e_acsl_ne = mpz_cmp(__e_acsl_y, __e_acsl);
e_acsl_assert(__e_acsl_ne != 0,
              "Assertion", "(y != 0)", 2);
// deallocate GMP's
mpz_clear(__e_acsl_y);
mpz_clear(__e_acsl);
```

E-ACSL Type System

Goal and Design

- for each integer n, a type system infers an interval $[a; b]$ containing n as small as possible
- if there are C types containing $[a; b]$, use the smallest one.
- otherwise use GMP's as before
- the previous example becomes simple

```
/*@ assert y != 0; */
e_acsl_assert(y != 0, "Assertion", "(y != 0)", 2);
```

- in practice, no more GMP's on the E-ACSL's test suite
- will be part of E-ACSL 0.2 (July? September?)

Combining Analyses within Frama-C

Demo

- how to ensure the safety of an annotated program
- by using several customizable analyzers
- based on different techniques?
- a "consolidation algorithm" merges all the results coming from the different analyzers with their different configurations

Demo!

Combining Analyses within Frama-C

Theory

- the consolidation algorihm is correct

  if each analyzer is correct, then the algorithm returns “Valid” (resp. “Invalid”) for a valid (resp. invalid) property. It returns “Inconsistent” if there are both a proof of validity and invalidity.

- the consolidation algorithm is complete

  if each analyzer is correct and indicates the right hypotheses, and if one analyzer does not indicate “Dont know” under recursively valid hypotheses, then the computed status is either “Valid” or “Invalid”.

Combining Analyses within Frama-C

Possible Statuses

- unknown
- valid
- invalid
- never tried
- considered valid (e.g. precondition of main)
- valid under hypotheses
- invalid under hypotheses
- unknown but dead
- valid but dead
- invalid but dead
- inconsistent

Combining Analyses within Frama-C

Papers

- 2 submitted articles:
  - L. Correnson and J. Signoles. Combining Analyses for C Program Verification.
  - P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles and B. Yakobowski. Frama-C: A Software Analysis Perspective.

Planning

- release of Frama-C Oxygen in July
- release of E-ACSL v0.2 in July or September.
  - will include the new type system
- integrating and testing the executable memory model
- handling runtime errors in annotations
- going further into collaboration between static and dynamic analyses
- writting a full paper on E-ACSL
- what about mix C/ADA programs?

Questions

Any questions?