E-ACSL Frama-C plug-in

Julien Signoles
CEA LIST
Software Safety Labs

Hi-Lite Meeting
November 2011, the 29th
Executable ANSI/ISO C Specification Language


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

- 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
Simplified Compilation Scheme

▶ input:

```c
int div(int x, int y) {
    /*@ assert y != 0; */
    return x / y;
}
```

▶ output:

```c
int div(int x, int y) {
    /*@ assert y != 0; */
    if (y == 0) e_acsl_fail();
    return x / y;
}
```

▶ a correct translation is much more complicated
Standard Compilation Scheme

- use **GMP integers** when required
- **keep the annotation** for documentation and further uses
- usually **one block of statements by annotation** (not always possible, e.g. \at)
- inserted at the **right code location**
- declares **temporary variables**
  - memoization to reduce memory usage
  - at function/global level when required
- **allocates and deallocates** them when required
- contains a **guard** if (! guard) e_acsl_fail(msg);
- may contain additional guards to **prevent execution of undefined values** (or at least a warning right now)
/*@ assert y != 0; */ z = x / y;

1. push a new environment \texttt{env} to translate the annotation

2. translate term \texttt{y} of type \texttt{int} to the \texttt{int} expression \texttt{y}

3. coerce \texttt{y} to an integer
   3.1 generate a fresh \texttt{mpz_t} variable \texttt{e_acsl\_1} corresponding to \texttt{y}
   3.2 add its declaration to \texttt{env}
   3.3 add its initialisation to \texttt{env}
      3.3.1 as the type of \texttt{y} is signed and smaller than \texttt{long}, generate \texttt{mpz_init_set_si(e_acsl\_1, y)};
   3.4 add its deallocation to \texttt{env}
      3.4.1 generate \texttt{mpz_clear(e_acsl\_1)};
   3.5 translate \texttt{y} to \texttt{e_acsl\_1}

4. translate term \texttt{0} of type \texttt{integer} to a fresh \texttt{mpz_t} variable \texttt{e_acsl\_2}
5. as its operands are integers, translate $\neq$ by using mpiz_cmp
   5.1 generate a fresh int variable e_acsl_3
   5.2 add its declaration to env
   5.3 add its initialisation to env
   5.3.1 generate e_acsl_3 = mpiz_cmp(e_acsl_1, e_acsl_2);
   5.4 no deallocation of e_acsl_3 required
   5.5 translate $y \neq 0$ to e_acsl_3 != 0

6. add the guard checking the assertion to env
   6.1 e_acsl_3 != 0 already gets type int: right!
   6.2 add the statement if (! (e_acsl_3 != 0)) then
    
    e_acsl_fail("y != 0"); to env

7. extend /*@ assert y != 0; */ z = x / y; with a new block computed from env and z = x / y;

8. pop env
E-ACSL in practice

- option `-e-acsl` to run the plug-in
- resulting code put in a new Frama-C project "e-acsl"
- new code linkable against GMP
- new code analysable by other analysers
- use standard Frama-C options on these projects
- option `-e-acsl-project` to set the resulting project name

Demo!
Plug-in Current Status

Typing

- C types
- integer
- boolean
- implicit coercions

implemented

not yet implemented

- real
Plug-in Current Status

Terms

**implemented**

- integer constants
- C left values
- arithmetic operators
- casts
- address &
- sizeof
- alignof
- `\null` (as `(void *)0`)
- `\at` (extra restriction)
- `\result`

**not yet implemented**

- `\true` and `\false`
- bitwise operators
- boolean operators
- conditional
- let binding
- `\typeof`
- `t-sets`
- `\base_addr`, `\offset` and `\block_length`
Plug-in Current Status

Predicates

implemented

- \texttt{true} and \texttt{false}
- relations ($==$, $<=$, ...)
- lazy conjunction $\&\&$
- lazy disjunction $||$
- lazy implication $\Rightarrow$
- negation $!$

not yet implemented

- equivalence $\iff$
- exclusive or $\wedge$
- conditionals
- let bindings
- quantifications
- $\texttt{at}$
- $\texttt{valid\ et\ al.}$
- $\texttt{initialized}$
implemented

▶ assertions
▶ function contracts
▶ statement contracts

not yet implemented

▶ behavior-specific annotations
▶ loop annotations
▶ global annotations
Plug-in Current Status

Behavior Clauses

implemented
- assumes
- requires
- ensures

not yet implemented
- assigns
- decreases
- abrupt termination
- complete behaviors
- disjoint behaviors
Planning

- release of first prototype planned for January 2012
  - based on Frama-C Nitrogen-20111001
  - implement some missing useful features
    - quantifiers over integers
    - what else?
  - plug-in packaging and documentation
  - stronger testing
- new release of E-ACSL reference manual at the same time
- use case during 2012
- implement other missing useful features during 2012
- better handling of E-ACSL undefined terms
  - will require Frama-C Oxygen
- improve customizability on need
- internship proposal: executable C memory model
Any questions?