E-ACSL and future Frama-C plug-in

Julien Signoles

CEA LIST

Software Safety Labs

Hi-Lite Annual Meeting

May 2011, the 10th

HI-LITE

Simplifying the use of formal methods

Part I

E-ACSL

Reminder of E-ACSL

Executable Ansi/ISO C Specification Language

What should be?

- 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 Reference Manual

- last version: 1.5-3
- Hi-Lite deliverable 3.4.1
- based on ACSL v1.5
- detailled syntax
- mainly point out differences with ACSL

E-ACSL Semantics

- similar to ACSL (e.g. mathematical integer arithmetic)
- 3-valued logic with undefinedness
  - `1/0 == 1/0` is undefined
  - `f(*p)` is undefined if p is invalid
  - tools must ensure that undefined terms are never evaluated
- lazy operators `&&`, `||`, `_?_:_`, `==>`
  - `\false && 1/0 == 1/0` is invalid
  - `1/0 == 1/0 && \false` is undefined
  - different but consistent semantics compared to ACSL
  - a valid (resp. invalid) E-ACSL predicate is valid (resp. invalid) in ACSL

Quantifications

- 3 different kinds of quantifications (only 1 in ACSL)
- unguarded quantification *à la* ACSL only allowed for boolean and char
- guarded integer quantification

```
\forall typ x1,...,xn;
  a1 <= x1 <= b1 ... && an <= xn <= bn
  ==> p
```

- guarded iterator quantification
  - from which element does the iteration begin?
  - how to access to the next elements?
  - which guards must be true to continue to iterate?
  - the only syntax extension from ACSL

```
struct btree {
  int val;
  struct btree *left, *right;
};

/*@ iterator access(_, struct btree *t):
  @   nexts t→left, t→right;
  @   guards \valid(t→left),
              \valid(t→right); */

/*@ predicate is_even(struct btree *t) =
  @   \forall struct btree *tt;
        access(tt, t) ==> tt→val % 2 == 0; */
```

Loop invariants

- lose their inductive nature
- a loop invariant I is equivalent to
  - put an assertion I just before entering the loop
  - put the same assertion at the very end of the loop body

Present:

- recursive logic definitions
- specification modules

Absent:

- lemmas and axiomatic (not computable)
- inductive predicate (not computable in general)
- polymorphism and higher order (still experimental in ACSL)
- concrete logic type (still experimental in ACSL)
- memory footprint (still experimental in ACSL)

Part II

Future Frama-C Plug-in

Future Frama-C plug-in

- new Frama-C plug-in called 'E-ACSL'
- takes a C program annotated with ACSL as input
- checks that annotations are part of E-ACSL
- roughly converts annotations

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

into C code

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

Example of Translation

- E-ACSL integers are mathematical integers
- heavy translation via GMP (could be optimized)

```
/*@ assert -3 == x; */ ;
// declare temp variables
mpz_t tmp1, tmp2, tmp3, int tmp 4;
mpz_init_set_si(tmp1,3);   // init tmp1 with 3
mpz_init(tmp2);            // init tmp2
mpz_neg(tmp2, tmp1);       // tmp2 = -tmp1 = -3
mpz_init_set_si(tmp3, x); // init tmp3 with x
// really check the assertion by comparing -3 to x
tmp4 = mpz_cmp(tmp2, tmp3);
if (tmp4 != 0) e_acsl_fail("(-3 == x)");
// deallocate temp variables
mpz_clear(tmp1); mpz_clear(tmp2); mpz_clear(tmp3);
```

What will be supported?

in Hi-Lite, only expect to handle a big-enough subset of E-ACSL

unsupported features:

- floats and reals
- constructs for memory management like `\valid` or `\at`

partially supported features:

- memory accesses: will not check that they are valid

What is planed?

- yet preliminary development
- first version planed to be released in September/October
- not possible to implement the whole stuff
- need your feedback to know what E-ACSL features you are going to use
- will implement required features before others

Any questions?