Towards an Executable ACSL

Virgile Prevosto

CEA List

July 13th, 2010
Introduction
  Context
  A first ACSL specification

Basic expressions

Code Annotations
  Function Contracts
  Loops

Logic Definitions
Introduction

Context
A first ACSL specification

Basic expressions

Code Annotations
Function Contracts
Loops

Logic Definitions
Frama-C in a nutshell

- [http://frama-c.com](http://frama-c.com)
- Developed at CEA List and INRIA Saclay-Île de France
- Framework for multiple analyses of C
- a kernel providing basic functionalities, based on CIL
- a set of plug-ins performing various static analyses

Some existing verification plug-ins

- Value analysis
- Jessie plug-in
- Wp
- Sante and link to the PathCrawler tool
Frama-C in a nutshell
- [http://frama-c.com](http://frama-c.com)
- Developed at CEA List and INRIA Saclay-Île de France
- Framework for multiple analyses of C
- a kernel providing basic functionalities, based on CIL
- a set of plug-ins performing various static analyses

Some existing verification plug-ins
- Value analysis
- Jessie plug-in
- Wp
- Sante and link to the PathCrawler tool
The ACSL Language

- ANSI/ISO C Specification Language
- Formal specification language dedicated to C programs
- Based on function contracts and code assertions

Main objectives

- Syntax and concepts close to C
- Independence from a particular analysis
- Allow communication between analyses
The ACSL Language

- ANSI/ISO C Specification Language
- Formal specification language dedicated to C programs
- Based on function contracts and code assertions

Main objectives

- Syntax and concepts close to C
- Independence from a particular analysis
- Allow communication between analyses
A first contact with ACSL

▶ Presentation of ACSL
▶ Simple specification examples
▶ Covering most constructions of the language

Executable subset of ACSL

▶ Identifying constructions that are relevant to run-time assertion checking
▶ Discussing how they could be implemented/compiled in C.
A first contact with ACSL
▶ Presentation of ACSL
▶ Simple specification examples
▶ Covering most constructions of the language

Executable subset of ACSL
▶ Identifying constructions that are relevant to run-time assertion checking
▶ Discussing how they could be implemented/compiled in C.
/**
 * requires \valid(root) && finite_list(root);
 * assigns \nothing;
 * ensures mem(\result,root);
 * ensures \forall integer n; mem(n,root) ==> \result >= n;
 */

int max_list(list* root);
int max_list(list* root) {
    int max = root->element;
    while(root->next) {
        root = root->next;
        if (root->element > max)
            max = root->element;
    }
    return max;
}
Introduction - A first ACSL specification

Loop Invariants

/*@ loop invariant \valid(root) && reachable(at(root,Pre),root) &&
mem(max,at(root,Pre)) &&
\forall int n;
  mem_sub(n, at(root,Pre),root) ==> max >= n;
*/
while(root->next) {
  ...
}
Introduction

Context
A first ACSL specification

Basic expressions

Code Annotations
Function Contracts
Loops

Logic Definitions
Description
▶ All pure C expressions are valid ACSL expressions
▶ Integer operations are performed in $\mathbb{Z}$, real in $\mathbb{R}$.

Executable Setting
▶ Translation is trivial...
▶ … Modulo the representation of mathematical datatypes
  ▶ For integers, we could use some BigNum library (GMP)
  ▶ For reals: ?
Basic expressions

Logic Operators

Description

▶ Propositional connectors
▶ Quantifiers

Executable Setting

▶ Propositional level can be mapped directly to booleans
▶ Possible to support some limited form of quantification over a finite set, e.g.

\[ \forall \text{ integer } x; 0 \leq x \leq 10 \implies a[x] \leq a[10] \]
Basic expressions

Built-in constructions

Description

- Predicates and logic functions about pointers (\texttt{valid, baseaddr,}...).
- Evaluation of an expression at a given program point (\texttt{at, old}).

Executable Setting

- Would require a dedicated memory management (allocation table).
- Store needed results in intermediate variables as required.
Pre/Post conditions

Description

- **requires** supposed to hold when entering the function.
- **ensures** supposed to hold at function exit.
- Notion of **behavior** with **assumes** clause.

Executable Setting

- Use assertions (when the underlying predicate can be evaluated at run time).
Assigns clause

Description

- **assigns** describes the locations that may be modified during one run of the function.

- In other words, every location not in an **assigns** clause must retain the same value between pre and post state.

Executable Setting

- Using allocation table and type information, might be possible to check the exact property.

- Other possibility: Decorate each assignment with an assertion that the lval is in **assigns** (overapproximation but might be easier to check).
Inductive invariant

Description

▶ Must be true before entering the loop.
▶ If true when entering, must stay true at the end of a loop step.

Executable Setting

▶ Use plain assertions (property a bit weaker).
Termination

Description

- **loop variant**: integer expression which:
  - is always non negative;
  - is strictly decreasing between two loop steps

Executable Setting

- Use assertions
- Store the preceding value of the variant
/*@ predicate */
hd_max{L}(list* root, integer m) = \
valid(root) && root->element <= m;
*/

Executable setting
- Inline the definition.
- Use C function.
Axiomatic definition

/*@ inductive
reducible(L)(list* root, list* node) {
  case reducible_hd{L}:
    \forall list* l1; reducible(l1,l1);
  case reducible_next{L}:
    \forall list* l1, *l2;
    \valid(l1) =>
    reducible(l1->next,l2) =>
    reducible(l1,l2);
}*/

Executable setting

- Not translatable in the general case
- Might identify some patterns in axioms?