Specification and Verification of High-Level Properties

Virgile Robles
Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, Pascale Le Gall

CEA List, Software Safety and Security Laboratory

December 6th, 2018
A verification framework for C programs

- A specification language: ACSL
- A kernel to parse C and ACSL
- A large collection of collaborative plugins

ACSL is a contract-oriented specification language.

Example: The contract of a function testing if an array T of with size elements contains x

```c
/*@  
requires \valid_read(T + (0..(size - 1)));
ensures \result == 0 <=> \forall integer j; 
  0 <= j < size ==> T[j] != x;
  assigns \nothing \from size, x, *(T + 0 .. (size - 1));
*/
int is_member(int* T, unsigned size, int x) { ... }
```
Deductive Verification with WP

/*@
requires \valid_read(T + (0..(size - 1)));
ensures \result == 0 <==> \forall integer j;
  0 <= j < size ==> T[j] != x;
assigns \nothing;
*/

int is_member(int* T, unsigned size, int x) {
  int res = 0;
  /*@ loop invariant ... */
  for(unsigned i = 0 ; i < size ; ++i) {
    /*@ assert rte: mem_access: \valid_read(T + i); */
    if(T[i] == x)
      res = 1;
  }
  return res;
}

WP and deductive verification

- Brings formal guarantees when tests only increase trust
- Sound but incomplete
The Limits of ACSL: a Case Study

Confidentiality-oriented page management:

- Each page has a confidentiality level CL (PUBLIC or PRIVATE),
- Each process has a similar level,
- A process can read from (or write to) a page depending on their levels
- A process may encrypt/decrypt a page, thus changing its level

Function contracts are insufficient: need for more global properties
We introduce meta-properties, which are a combination of:

- **A classic property** $P$, expressed in ACSL.
- **A context**: The specific situation in which $P$ must hold inside a function.
- **Target functions**: The set of functions for which $P$ should hold in the given context.

\[
\text{meta } \text{strong\_invariant}\{\text{foo,bar}\}, A < B;
\]

"$A < B$" must hold everywhere in functions foo and bar

\[
\text{meta } \text{writing}(\text{ALL}), \ \text{written} != &X;
\]

No function can modify the global variable $X$

\[
\text{meta } \text{writing}(\text{ALL}), \ \text{written} == &X ==> X == 0;
\]

A function can only modify $X$ if it was previously null
Available Contexts for Meta-Properties

- **Strong invariant:** Everywhere in the function
- **Weak invariant:** Before and after the function
- **Upon writing:** Whenever the memory is modified. The property $P$ can use a special meta-variable `written`, referencing the address being written to at a particular point.

```
meta \text{writing}(\text{ALL}), \text{written} \neq &X;
```

No function can modify the global variable $X$

- **Upon reading:** Similarly, when memory is read
- **Upon calling:** Similarly, when a function is called
Use of Labels in Meta-Properties

In Frama-C, predicates can refer to the value of locations at different points (labels): Pre, Post, Here, C labels, etc.

\[\text{assert } \texttt{\at}(x, \text{Here}) == \texttt{\at}(x, \text{Pre});\]
x has the same value as when the function was called

Still true for meta-properties with two more labels: Before (resp. After), referring to state before (resp. after) any statement relevant to the context.

\[\text{meta } \texttt{\writing}(\text{main}), \texttt{\written} \neq \& X \mathbin{||} \texttt{\at}(X, \text{Before}) == 0 \mathbin{||} \texttt{\at}(X, \text{After}) \neq 0;\]
There is no statement changing X to 0 in main
Automatic Verification of Meta-properties (1/4)

Translation of meta-properties into native ACSL: leverage existing tools.

**Strong invariant** $P$: assert $P$ when truth may change

Before and after transformation for

```c
meta \strong_invariant(main), A == B;
A must remain equal to B at every point of main
```

```c
void main() {
  C = 42;
  A = C;
  B = C;
}

/*@ requires A == B; 
ensures A == B; */
void main() {
  C = 42;
  A = C;
  //@ assert A == B; //Failure
  B = C;
  //@ assert A == B;
}
```

**lenient delimiter:**

- Combines strong and weak invariant
- Allows to break the invariant locally
Upon writing: detect modification sites by syntactic analysis

Before and after transformation for

\[
\text{meta } \backslash \text{writing(main), } \backslash \text{written } \neq \& C; \\
\text{main cannot modify C}
\]

```c
void main() {
    C = 42;
    A = C;
    B = C;
}
```

Performance: discard any obvious assertion to avoid overloading the proof
After/Before labels: refer to local C labels

Before and after transformation for

\begin{verbatim}
meta writing(main), written != &X || 
   \at(X, Before) == 0 || \at(X, After) != 0;
\end{verbatim}

There is no statement changing \( X \) to 0 in \texttt{main}

\begin{verbatim}
void main() {
    _meta_1: X = X;
    /*@ assert \at(X, _meta_1) == 0 
        || \at(X, Here) != 0; */

    _meta_2: X = 4;
    /*@ assert \at(X, _meta_2) == 0 
        || \at(X, Here) != 0; */

    _meta_3: X = 0;
    /*@ assert \at(X, _meta_3) == 0 
        || \at(X, Here) != 0; */

    //Failure
}
\end{verbatim}
Specification-only functions: use *assigns* clause for *writing* context

```plaintext
/*@
behavior BA:
assumes PA(params);
assigns XA1, XA2;
behavior BB:
assumes PB(params);
assigns XB;
*/

extern void g(params);

void f() {
  g(act_params);
  /*@
  assert PA(act_parms) ⇒ &XA1 != &glob; */
  /*@
  assert PA(act_parms) ⇒ &XA2 != &glob; */
  /*@
  assert PB(act_parms) ⇒ &XB != &glob; */
}

/*@ meta \writing(f), \written != &glob; */
```
The confidentiality case study was:

- Implemented in C
- Partially specified with ACSL contracts
- Fully specified with meta-properties

Some of the meta-properties:

- Public allocated pages cannot be modified by private agents
- Confidentiality levels can only be modified by encryption/decryption
- Unallocated pages cannot be read from
- Only the allocation/deallocation functions can change the status of a page

**Verification results:**

- Transformation time: $< 5s$
- 290 proof obligations
- Automatically proved in $\approx 1m$ with Alt-Ergo
Conclusion

Contributions:
- More expressive power: see case study
- High-level view of properties established on a software module
- Ease development: automatically check if a property is maintained after an update (of the code or of a function contract)

Future Work:
- Tackle more realistic case studies
- Enrich meta-properties where needed
- Prove soundness of transformation
Conclusion

Contributions:

- More expressive power: see case study
- High-level view of properties established on a software module
- Ease development: automatically check if a property is maintained after an update (of the code or of a function contract)

For more details, see:

- [https://github.com/Firobe/metacsl_examples](https://github.com/Firobe/metacsl_examples)