Specification and Verification of High-Level Properties

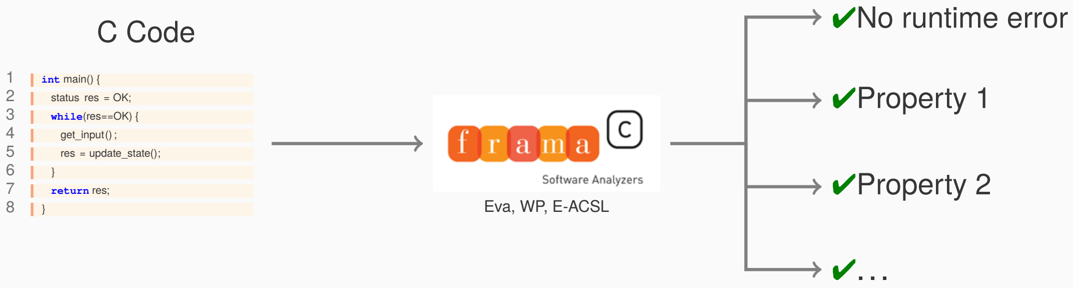
Frama-C Days

Virgile Prevosto

June 13th, 2024

Université Paris-Saclay, CEA, List

CONTEXT



C Code

```
int main() {
  status res = OK;
  while(res==OK) {
    get_input() ;
    res = update_state();
  }
  return res;
}
```

How do we specify properties?

ACSL

Position of the problem

- Frama-C verifies that the code is conforming to its specification.
- How confident can we be that annotations are really what we want to express?
- In particular if they are big/numerous
  - Example from Thales[1] (see next talk): ˜400kLoC of ACSL annotations (from ˜500 lines of MetAcsl).
- And/or if we need to encode part of the property with ghost code.

[1] Djoudi et al. Formal verification of a JavaCard virtual machine with Frama-C. FM 2021

Examples of properties beyond ACSL

What ACSL is good at

- Expressing a property that (should) hold at a specific program point
- or pre- and post- states for a contract
- or a few more within the same function with `\at`

More complex situations

- Check a property at a large number of program points: MetAcsl plug-in
- Compare several executions of a function/some functions: Rpp plug-in
- Check a sequence of events across the whole execution path: Aoraï plug-in

HILAREs: MetAcsl

- Developed by Virgile Robles during his PhD.
- HIgh-Level ACSL REquirement.
- Properties that must be verified at various program points (context).
- Meta-variables depending on the context.
- Each HILARE is instantiated as (many) ACSL annotations.

```
meta \prop,
\name(region_integrity_task),
\targets( \diff( \ALL, init ) ),
\context(\writing),
\forall integer i; 0 <= i < NUM_REGIONS
  \overlaps(\written,
         RegionStart[i] +
          (0 .. RegionSize[i] - 1))
  ==> RegionOwner[i] == CurTask;

meta \prop,
\name(region_owners_final),
\targets( \ALL ),
\context(\writing),
\separated(\written,
   RegionOwner + (0 .. NUM_REGIONS - 1));
```

Generation of HILAREs

```c
int x, y;
/*@ assigns x \from n; */
extern void set_x(int n);

/*@
meta \prop,\name(example),
\targets(\diff(\ALL,set_x)),
\context(\writing),
\separated(\written,&x);
*/

void test(int* p) {
  *p = 1;
  y = 2;
}
```

```
frama-c -meta meta_test.c
```

```c
void test(int *p)
{
  /*@ assert example:  meta:
       \separated(p, &x); */
  *p = 1;
  y = 2;
  return;
}
```

- Some (trivial) simplifications are done

Verification of HILAREs

Use standard analysis plug-ins

```
frama-c -meta meta_test.c -then-last -wp
```

- can also work with Eva or E-ACSL
- Wp may require additional annotations
- Some of which might come from other HILAREs

Use standard analysis plug-ins

```
frama-c -meta meta_test.c -then-last -wp
```

- can also work with Eva or E-ACSL
- Wp may require additional annotations
- Some of which might come from other HILAREs

Reasoning at Meta-level

- a few deduction patterns to prove an HILARE from other ones
- Why3 model
- Prolog deduction engine

Relational Properties: Rpp

- Developped by Lionel Blatter during his PhD
- Express relational properties between function calls
- Use self-composition to prove the property

Non-interference

```
extern int private, public;

//@ assigns public,private \from public,private;
void f() {
  if (public < 1000) public++;
  if (private > public) private-=public;
}

/*@ relational
    \callset(\call(f,c1),\call(f,c2)) ==>
    \at(public,Pre_c1) == \at(public,Pre_c2)
    ==>
    \at(public,Post_c1) == \at(public,Post_c2);
*/
```

```
frama-c -rpp rpp_test.c -then -wp
```

Instrumented code

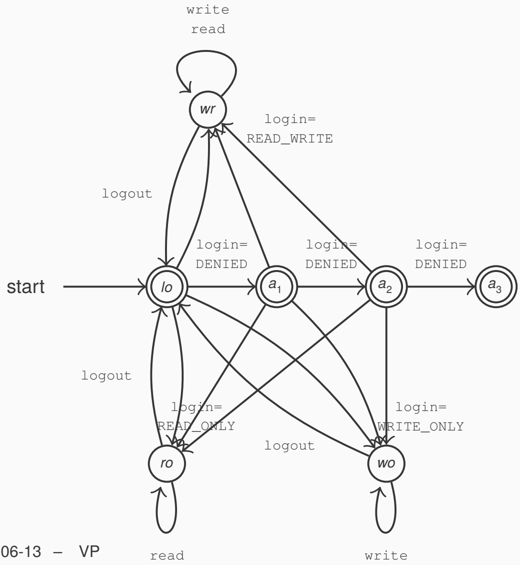
```
void relational_wrapper_2(void)
{
  if (public_c1_2 < 1000) public_c1_2 ++;
  if (private_c1_2 > public_c1_2) private_c1_2 -= public_c1_2;
  ;
  if (public_c2_2 < 1000) public_c2_2 ++;
  if (private_c2_2 > public_c2_2) private_c2_2 -= public_c2_2;
  ;
  /*@ check Rpp:
        \at(public_c1_2,Pre) == \at(public_c2_2,Pre) ==>
        \at(public_c1_2,Here) == \at(public_c2_2,Here);
  */
  return;
}
```

Beyond Self-composition

- Transformation becomes very complex in presence of pointers (and aliasing)
- More flexible solution using directly Wp primitives [2]
- Proved in Coq (on a subset of C)
- But not implemented in the Rpp plug-in itself

[2] Blatter et al., *An Efficient VCGen-Based Modular Verification of Relational Properties*. ISoLA'22, LNCS 13701

Automata: Aoraï



- Initial version by Nicolas Stouls (Inria/INSA Lyon).
- Automaton encoding admissible call sequences during an execution of the program
- Instrumented with ghost code, and possibly ACSL contracts

Automaton in Aoraï

```
logged_out: { login() {{ \result == -1 }} } -> attempt_1
          | { login() {{ \result == 0 }} } -> logged_ro
          | { login() {{ \result == 1 }} } -> logged_wo
          | { login() {{ \result == 2 }} } -> logged_rw;

attempt_1:  { login() {{ \result == -1 }} } -> attempt_2
          | { login() {{ \result == 0 }} } -> logged_ro
          | { login() {{ \result == 1 }} } -> logged_wo
          | { login() {{ \result == 2 }} } -> logged_rw;

attempt_2:  { login() {{ \result == -1 }} } -> too_many_attempts
          | { login() {{ \result == 0 }} } -> logged_ro
          | { login() {{ \result == 1 }} } -> logged_wo
          | { login() {{ \result == 2 }} } -> logged_rw;
...
```

Verification of an Aoraï instrumented code

```
frama-c -aorai-automata automaton.ya test_aorai.c -then-last -eva
```

Ghost functions advancing automaton

```
/*@ ghost void login_post_func(enum permission res) {
 /@ slevel full; @/
 ...
 if (aorai_intermediate_state_7 == 1)
  if (res == -1) too_many_attempts_tmp = 1; else too_many_attempts_tmp = 0;
 else too_many_attempts_tmp = 0;
  if (reject == 1) reject_tmp = 1; else reject_tmp = 0;
   if (aorai_intermediate_state_1 == 1) {
    if (res == 1) logged_wo_tmp = 1; else goto __aorai_label_16;
   }
   else
   ...
} */
```

Aoraï and Eva

- Aoraï's properties intrinsically a whole-program analysis
- Eva or E-ACSL are the primary candidates for verification
- Eva-related options and built-ins:
  - `Frama_C_show_aorai_state`
  - `-aorai-instrumentation-history`

Conclusion

- Various possibilities to tell Frama-C what you want to prove beyond mere function contracts.
- Experiment on industrial code (Aoraï).
- Used in production (MetAcsl).
- Takes full advantage of the modularity of the platform.
- More info in Chap. 10 of the book.

Perspectives

- Propose specialized analysis plug-ins for some kinds of properties.
- Abstraction and reasoning at model level (SecurEval project).
- New kinds of properties (e.g. CaRet, Typestates).
  - Sébastien Patte
- *Quis custodiet ipsos custodes?* Formalize ACSL semantics and the transformations
  - Louis Gauthier
  - Sébastien Patte