Specification and Verification of High-Level Properties with MetAcsl

Nikolai Kosmatov
Joint work with Virgile Robles, Virgile Prevosto,
Louis Rilling and Pascale Le Gall

Seminar at LIFO, March 15, 2021

Outline

- Frama-C and its specification language ACSL
- Motivation: Specification and verification of global properties
- Solution: High-Level ACSL Requirements (HILARE)
- Examples of Proof with MetAcsl and WP
- Conclusion

Frama-C at a Glance

- FRAmework for Modular Analysis of C programs
  - Various plugins: CFG, value analysis (abstract interpretation), impact analysis, dependency analysis, slicing, program proof, . . .
- Developed at CEA LIST and INRIA Saclay (Proval/Toccata team)
- Released under LGPL license
- Kernel based on CIL library [Necula et al. – Berkeley]
- Includes ACSL specification language
- Extensible platform
  - Adding specialized plugins is easy
  - Collaboration of analyses over the same code
  - Inter-plugin communication through ACSL formulas
- `http://frama-c.com/`

Frama-C and WP: A brief history

- 90's: CAVEAT, a Hoare logic-based tool for C programs
- 2000's: CAVEAT used by Airbus during certification of the A380
- 2002: Why tool and its C front-end Caduceus
- 2006: Joint project to write a successor to CAVEAT and Caduceus
- 2008: First public release of Frama-C (Hydrogen)
- 2009: Hoare-logic based Frama-C plugin Jessie developed at INRIA
- 2012: New Hoare-logic based plugin WP developed at CEA LIST
- Frama-C today:
  - Most recent release: Frama-C Titanium (v.22)
  - Multiple projects around the platform
  - A growing community of users

ACSL: ANSI/ISO C Specification Language

Presentation

- Based on the notion of contract, like in Eiffel
- Allows the users to specify functional properties of their programs
- Allows communication between various plugins
- Independent from a particular analysis
- ACSL manual at `http://frama-c.com/acsl`

Basic Components

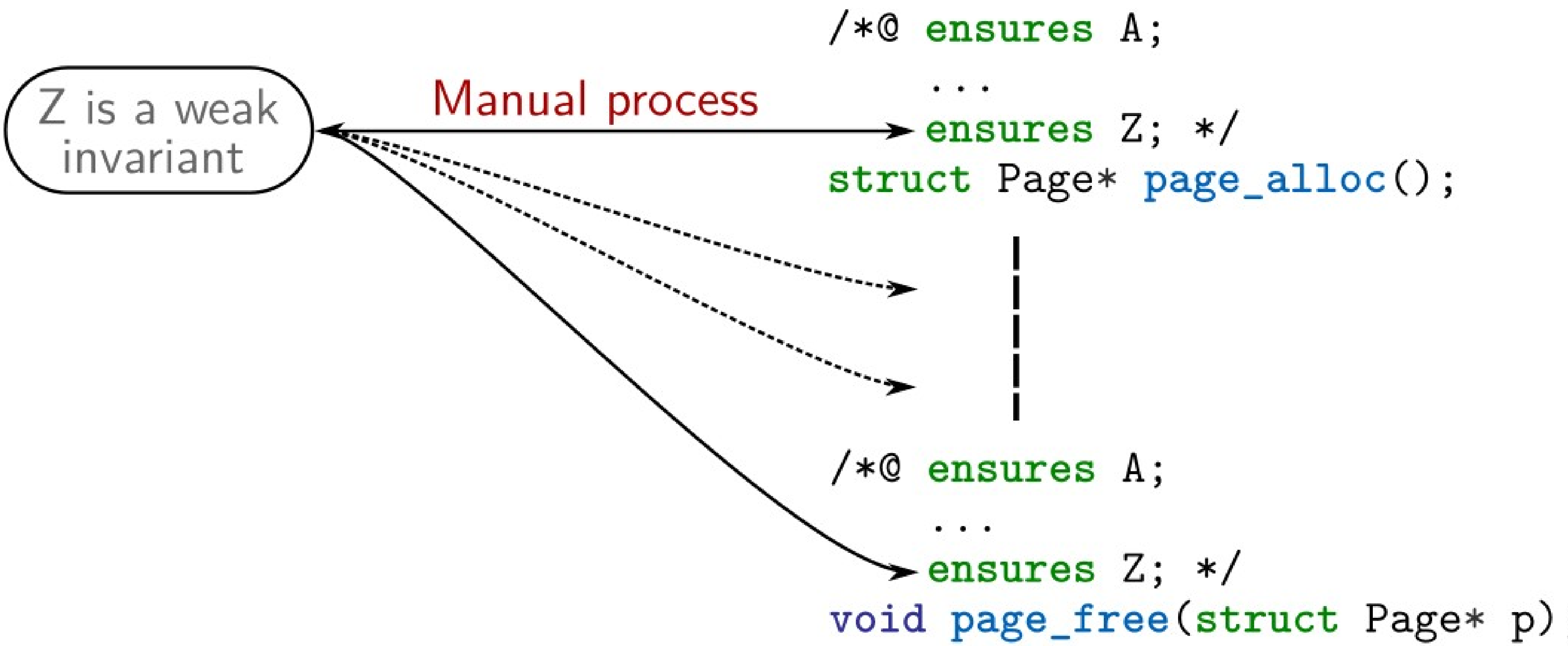
- First-order logic
- Pure C expressions
- C types + $\mathbb{Z}$ (integer) and $\mathbb{R}$ (real)
- Built-ins predicates and logic functions, particularly over pointers: `\valid(p)` `\valid(p+0..2)`, `\separated(p+0..2,q+0..5)`, `\block_length(p)`

Outline

- Frama-C and its specification language ACSL
- Motivation: Specification and verification of global properties
- Solution: High-Level ACSL Requirements (HILARE)
- Examples of Proof with MetAcsl and WP
- Conclusion

Motivation: Global Properties

Specifying global properties with contracts: manual and tedious. No explicit link between clauses.

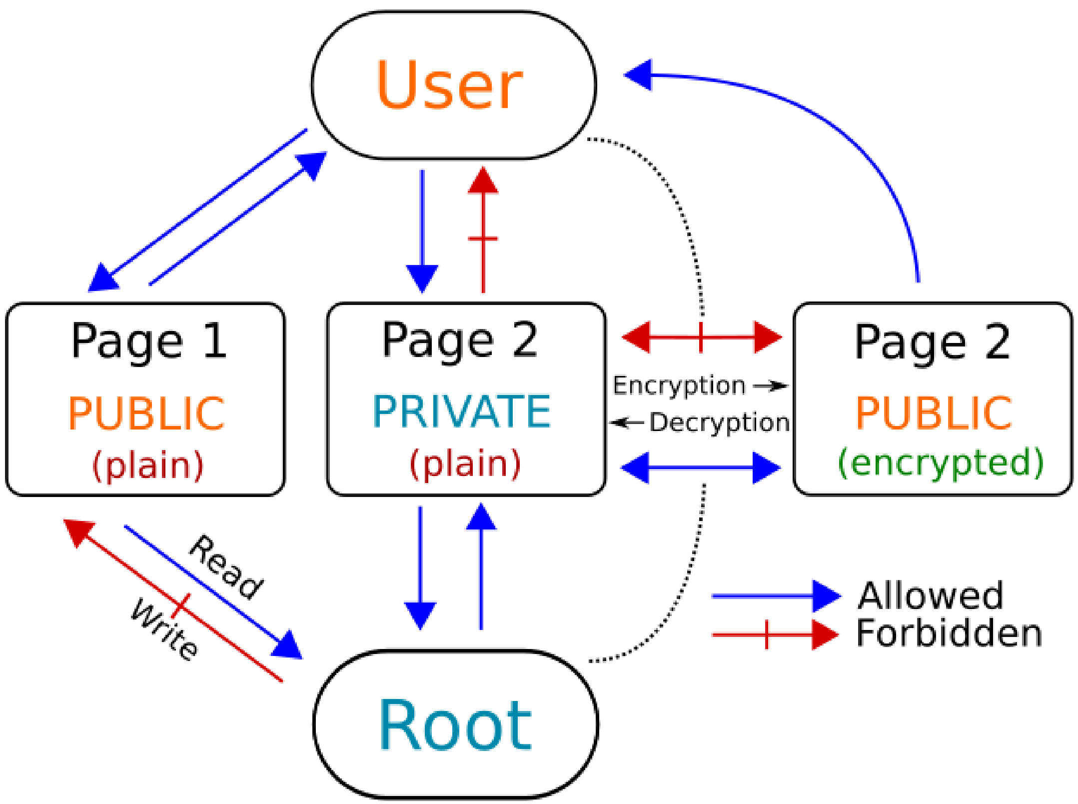


Assessing if contracts form a global property is difficult, especially after an update.

First Case Study

Confidentiality-oriented page management:

- Each page has a confidentiality level (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

Examples of High-Level Properties

- A non-privileged user never reads a privileged (private) data page
- A privileged user never writes to a non-privileged (public) page
- A non-privileged user cannot decrypt an encrypted data page
- The privilege level of a page cannot be changed unless...
- The privilege level of a user cannot be changed unless...
- A free page cannot be read or written, and must contain zeros

Such properties can be expressed as

- Constraints on reading / writing operations, calls to some functions,
- Strong or weak invariants

Second Case Study:

A Smart House

Room : **door lock** (locked/unlocked), window (open/closed), AC system (off, heating, cooling).
Notion of current user authentication. An alarm can be enabled.
Properties:

- P_1: only the unlocking function can open a **door lock**
- P_2: a **lock** can only be opened if the user has sufficient clearance or if the alarm is being enabled
- P_3: whenever the alarm is ringing, all **doors locks** must be open
- P_4: whenever a window is open, the AC in the room must be disabled

Outline

- Frama-C and its specification language ACSL
- Motivation: Specification and verification of global properties
- Solution: High-Level ACSL Requirements (HILARE)
- Examples of Proof with MetAcsl and WP
- Conclusion

Solution: Meta-properties, or HILARE (High-Level ACSL Requirements)

We introduce meta-properties, which are a combination of:

- **A set of targets functions**, on which the property must hold.
  foo {foo, bar} \ALL \diff(\ALL,{foo, bar})
- **A context**, which characterizes the situation in which the property must hold.
  \strong_invariant \writing \reading
- **An ACSL predicate**, expressed over the set of global variables.
  A < B *p == 0 \separated(\written, p)

```
meta \prop,
    \name(A < B everywhere in foo and bar),
    \targets({foo, bar}),
    \context(\strong_invariant),
    A < B;
```

Available Contexts

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

```
meta \prop, \name(X is only modified if null),
        \targets(\ALL), \context(\writing),
        !\separated(\written, &X) ⇒ X == 0;
```

- **Upon reading:** Similarly, when memory is read
- **Upon calling:** Similarly, when a function is called

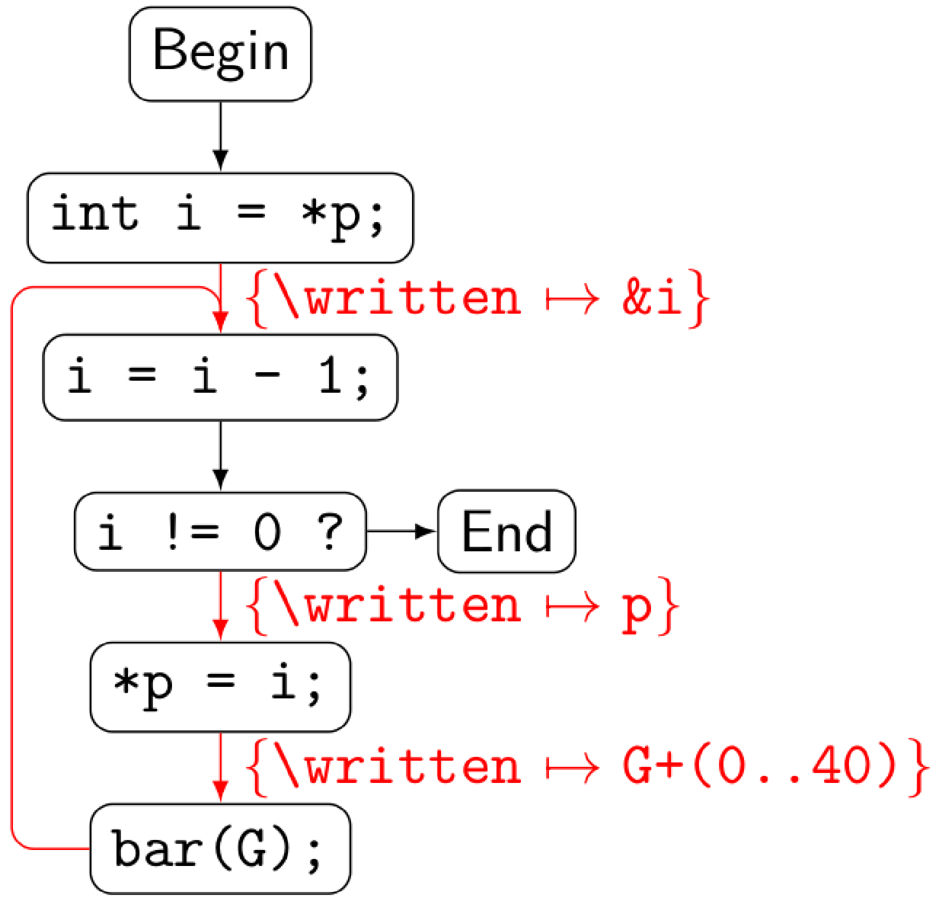
```
meta \prop, \name(foo can only be called from bar),
        \targets(\diff(\ALL, bar)),
        \context(\calling), \called ≠ &foo;
```

Illustration of \writing Context

```
char* G; int X;

/*@ assigns
    T[0 .. 40];
*/
void bar(char* T);
//bar is declared
//but not defined

void foo(int* p) {
    int i = *p;
    while(--i) {
        *p = i;
        bar(G);
    }
}
```



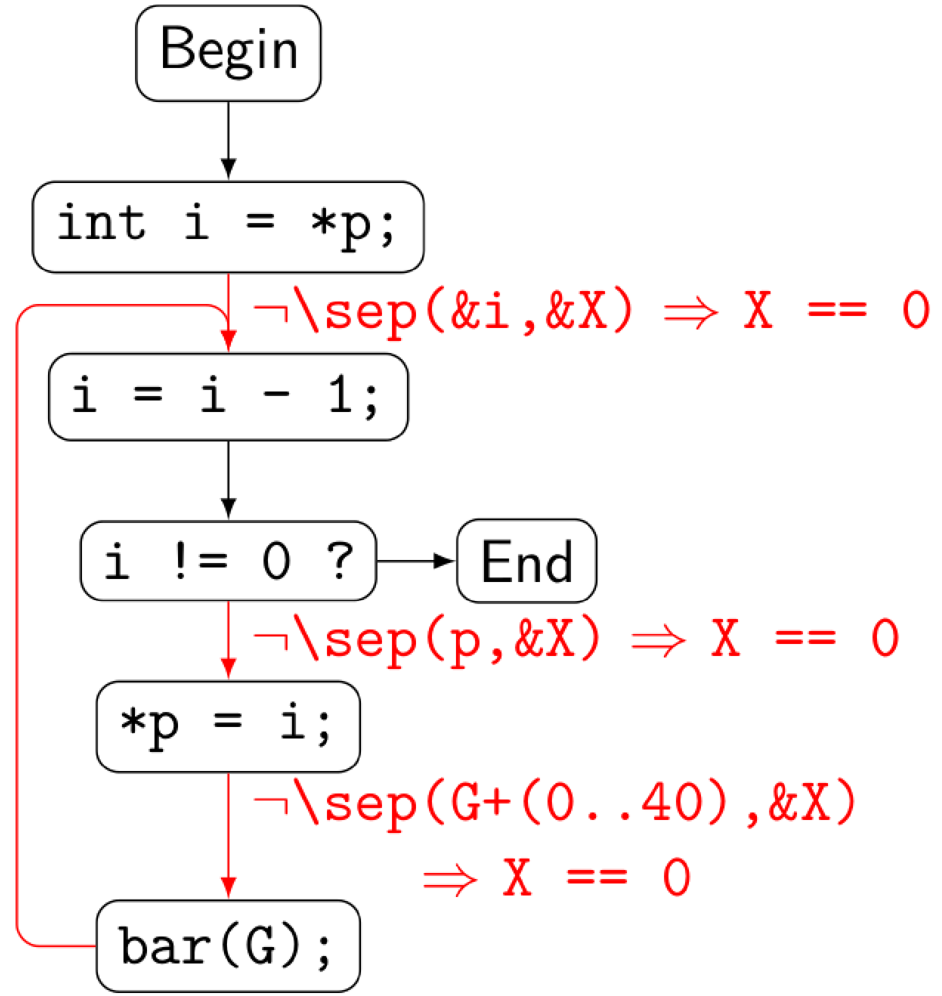
The \writing context maps a function to a set of edges which are labelled with the modified variable(s) in that edge.

HILARE Translation Example

Let's combine a context, a target and a predicate.

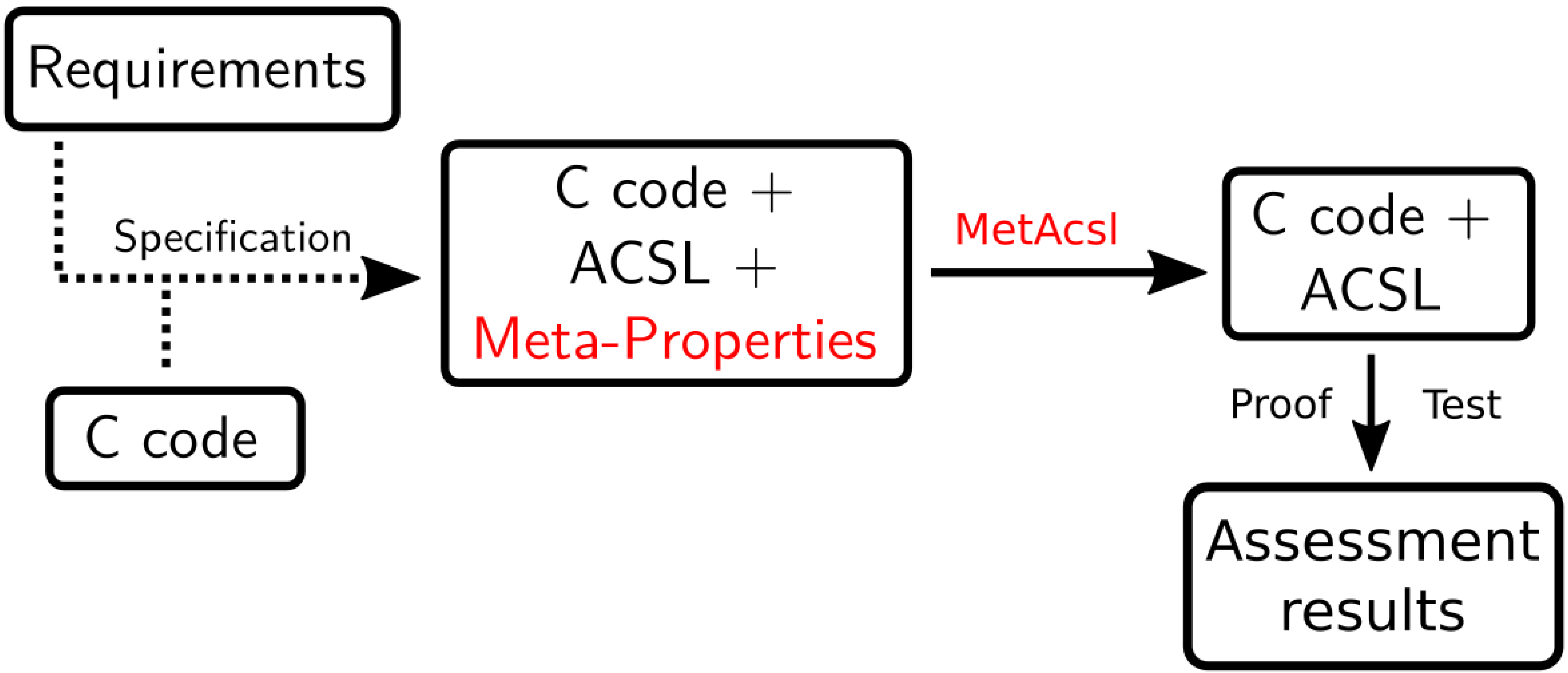
```
meta \prop,
     \name(X is only
          modified if null),
     \targets(\ALL),
     \context(\writing),
     !\separated(\written,&X)
        ⇒ X == 0;
```

Some edges are labelled with ACSL properties that must be true.



Automatic Assessment of HILARE's

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



Translation into Assertions

Mechanise the context → Plug in the property → Translate to assertions

```
void foo(int* p) {
    int i = *p;
    while(1) {
        i --;
        if (! i) break;
        *p = i;
        bar(G);
    }
}
```

(a) Previous program after normalization

```
void foo(int *p) {
  int i = *p;
  while (1) {
    //@ assert !\separated(&i, &X)
        ⇒ X == 0; //Trivial
    i --;
    if (! i) break;
    //@ assert !\separated(p, &X) ⇒
         X == 0;
    *p = i;
    //@ assert !\separated(G + (0
         .. 40), &X) ⇒ X == 0;
    bar(G);
  }
  return;
}
```

(b) After translation of the meta-property

Performance: discard trivial assertions to avoid overloading the prover

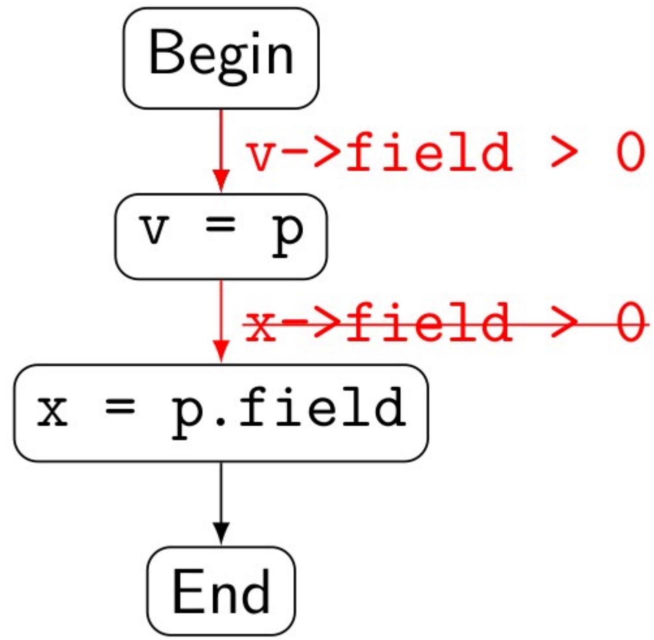
Typing Issue during Translation

```
struct S {
    int field;
};

/*@ meta \prop, \name(unsafe),
    \targets(\ALL),
    \context(\writing),
    \written->field > 0;
*/

struct S v; int x;

void foo(struct S p) {
    v = p;
    x = p.field;
}
```



We only know that \written, \read, etc. are (sets of) address(es).

Typing Issue: Possible Solution

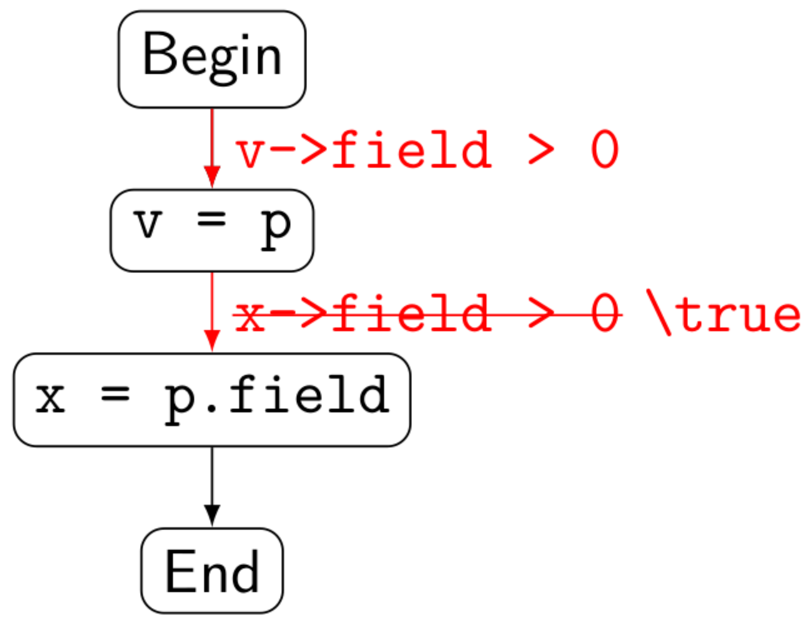
One solution : *guarding* predicates with default values.

```
struct S {
    int field;
};

/*@ meta \prop, \name(unsafe),
    \targets(\ALL),
    \context(\writing),
    \tguard(\written->field > 0);
*/

struct S v; int x;

void foo(struct S p) {
    v = p;
    x = p.field;
}
```



Hypotheses on the types of \written, \read, etc. *must* be guarded.

Relating Several States: Labels in HILARE

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

```
assert \at(x, Here) == \at(x, 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.

```
meta \prop, \name(door unlocked only if authorized),
    \targets(\ALL), \context(\writing),
      !\separated(\written, locked)
        ∧ \at(locked, Before) == 1 ∧ \at(locked, After) == 0
        ⇒ authorization_given;
```

Figure: We can express our smart house property

Examples for First Case Study

```
meta \prop, \name(Do not write to lower pages outside free),
  \targets(\diff(\ALL , {page_free})),
  \context( \writing ),

  \forall integer i; 0 <= i < MAX_PAGE_NB ==>
  \let p = pages + i;
  p->status == PAGE_ALLOCATED &&
  user_level > p->confidentiality_level ==>
  \separated(\written, p->data + (0.. PAGE_SIZE - 1));

meta \prop, \name(Free pages are never read),
  \targets(\ALL),
  \context( \reading ),

  \forall integer i; 0 <= i < MAX_PAGE_NB &&
  pages[i].status == PAGE_FREE ==>
  \separated(\read, pages[i].data + (0 .. PAGE_SIZE - 1));
```

Additional Proof Hints

- Specify a HILARE footprint
  - Identify all variables a HILARE relies on
  - Specify modification rules for them
- Prove the absence of runtime errors (undefined behavior)
  - Nothing is ensured if undefined behavior can occur
- For scalability, insert assertions as check's rather than assert's
  - In ACSL, asserts are proved and <u>kept</u> in proof context
- Lemma HILARE's: assist the prover
  - A HILARE can need assertions to be proved
  - Difficult to insert them all manually
  - Another HILARE (translated into asserts) can do that

Outline

- Frama-C and its specification language ACSL
- Motivation: Specification and verification of global properties
- Solution: High-Level ACSL Requirements (HILARE)
- Examples of Proof with MetAcsl and WP
- Conclusion

Install Frama-C and MetAcsl

Installation instructions using opam (v.2):

- opam update
- opam switch create 4.08.1_fc22 ocaml-base-compiler.4.08.1
- eval `opam env`
- opam install depext
- opam depext frama-c.22.0
- opam install alt-ergo.2.3.2 why3.1.3.3 frama-c.22.0 frama-c-metacsl.0.1
- why3 config –detect

Run MetAcsl to instantiate HILARE's and WP to prove the program: e.g.

- frama-c-gui myfile.c -meta -meta-checks -meta-no-simpl -meta-number-assertions -then-last -wp -wp-rte

Example: Strong Invariant

```
int A;
int B;
int C;
/*@ meta "A_B_eq_strong";
*/
/*@ check requires A_B_eq_strong: _1: meta: A ≡ B;
    requires A ≡ B;
    check ensures A_B_eq_strong: _1: meta: A ≡ B;
    ensures
      (C ≥ 0 ∧ A ≡ C ∧ B ≡ C) ∨
      (C < 0 ∧ A ≡ \old(A) ∧ B ≡ \old(B));
    assigns A, B;
 */
void foo(void)
{
  if (C >= 0) {
    A = C;
    /*@ check A_B_eq_strong: _3: meta: A ≡ B; */ ;
    B = C;
    /*@ check A_B_eq_strong: _4: meta: A ≡ B; */ ;
  }
  /*@ check A_B_eq_strong: _2: meta: A ≡ B; */ ;
  return;
  /*@ check A_B_eq_strong: _5: meta: A ≡ B; */ ;
}
```

test2.c

```
int A, B, C;
/*@
  meta \prop, \name(A_B_eq_strong),
    \targets(\ALL), \context(\strong_invariant),
    A == B;  // FAILS
*/
/*@
  requires A==B;
  assigns A,B;
  ensures C>=0 && A==C && B==C ||
    C<0 && A==\old(A) && B==\old(B); */
void foo(){
  if ( C >= 0 ){
    A = C;
    B = C;
  }
}
```

Example: Weak Invariant

```
int A;
int B;
int C;
/*@ meta "A_B_eq_weak";
*/
/*@ check requires A_B_eq_weak: _1: meta: A ≡ B;
    requires A ≡ B;
    check ensures A_B_eq_weak: _1: meta: A ≡ B;
    ensures
      (C ≥ 0 ∧ A ≡ C ∧ B ≡ C) ∨
      (C < 0 ∧ A ≡ \old(A) ∧ B ≡ \old(B));
    assigns A, B;
 */
void foo(void)
{
  if (C >= 0) {
    A = C;
    B = C;
  }
  return;
}
```

test3.c

```
int A, B, C;
/*@
  meta \prop, \name(A_B_eq_weak),
    \targets(\ALL), \context(\weak_invariant),
    A == B;
*/
/*@
  requires A==B;
  assigns A,B;
  ensures C>=0 && A==C && B==C ||
    C<0 && A==\old(A) && B==\old(B); */
void foo(){
  if ( C >= 0 ){
    A = C;
    B = C;
  }
}

```

Example: Reading Context

```
/*@ meta "A_not_read";
 */
/*@ requires A ≡ B;
    ensures
      (C ≥ 0 ∧ A ≡ C ∧ B ≡ C) ∨
      (C < 0 ∧ A ≡ \old(A) ∧ B ≡ \old(B));
    assigns A, B;
 */
void foo(void)
{
  /*@ check A_not_read: _1: meta: \separated(&C, &A); */
  if (C >= 0) {
    /*@ check A_not_read: _2: meta: \separated(&C, &A); */
    A = C;
    /*@ check A_not_read: _3: meta: \separated(&C, &A); */
    B = C;
  }
  return;
}
```

test4.c

```
int A, B, C;
/*@
  meta \prop, \name(A_not_read),
    \targets(\ALL), \context(\reading),
    \separated(\read, &A);
*/
/*@
  requires A==B;
  assigns A,B;
  ensures C>=0 && A==C && B==C ||
    C<0 && A==\old(A) && B==\old(B); */
void foo(){
  if ( C >= 0 ){
    A = C;
    B = C;
  }
}

```

Example: Writing Context

```
/*@ meta "A_unchanged_unless";
 */
/*@ requires A ≡ B;
    ensures
      (C ≥ 0 ∧ A ≡ C ∧ B ≡ C) ∨
      (C < 0 ∧ A ≡ \old(A) ∧ B ≡ \old(B));
    assigns A, B;
  */
void foo(void)
{
  if (C >= 0) {
    /*@ check A_unchanged_unless: _1: meta: C < 0 → \separated(&A, &A);
    A = C;
    /*@ check A_unchanged_unless: _2: meta: C < 0 → \separated(&B, &A);
    B = C;
  }
  return;
}
```

test5.c

```
int A, B, C;
/*@
  meta \prop, \name(A_unchanged_unless),
      \targets(\ALL), \context(\writing),
      C < 0 ==> \separated(\written, &A);
*/
/*@
  requires A==B;
  assigns A,B;
  ensures C>=0 && A==C && B==C ||
    C<0 && A==\old(A) && B==\old(B); */
void foo(){
  if ( C >= 0 ){
    A = C;
    B = C;
  }
}

```

Real-Life Case Study: Wookey

- Wookey: Secure open-source USB storage device
- Developed by ANSSI
- Wookey Bootloader: a highly critical module
- Security properties specified with HILARE's
- Fully verified with MetAcsl and WP
- For more detail, see [FormaliSE'21]

Outline

- Frama-C and its specification language ACSL
- Motivation: Specification and verification of global properties
- Solution: High-Level ACSL Requirements (HILARE)
- Examples of Proof with MetAcsl and WP
- Conclusion

Conclusion

- HILARE’s: a very convenient solution to specify global properties
  - Security properties (isolation, integrity, confidentiality)
  - Expressive thanks to ACSL (e.g. \separated, \forall) and several contexts
  - Explicit global view of what is proved (for verification engineers, evaluators...)
- MetAcsl enables HILARE proof/testing with WP, EACSL...
  - Proof without overloading proof context, scales very well (due to ACSL checks)
  - Lemma HILARE’s can help to prove other HILARE’s
- Used in large-scale verification projects

Future Work

- Reasoning on HILARE’s
- Automatic verification that no variable is left unspecified in a predicate footprint
- A more convenient view of HILARE’s (easier to specify? model-based view?)

References

- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
  “MetAcsl: Specification and Verification of High-Level Properties.”
  In Proc. of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019), pages 358-364, LNCS, vol. 11427. Springer.

- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
  “Tame your annotations with MetAcsl: Specifying, Testing and Proving High-Level Properties”.
  In Proc. of the 13th International Conference on Tests & Proofs (TAP 2019), pages 167-185. LNCS, vol. 11823. Springer.

- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
  “Methodology for Specification and Verifiation of High-Level Properties with MetAcsl”.
  In Proc. of the 9th IEEE/ACM International Conference on Formal Methods in Software Engineering (FormaliSE 2021), IEEE. To appear.