# Specification and proof of relational properties with Frama-C

Nikolai Kosmatov (CEA Tech, List)

joint work with Lionel Blatter, Virgile Prevosto, Pascale Le Gall

JML Workshop 2017
Orlando, December 18th, 2017

# Outline

Frama-C, a platform for analysis of C code

Function Contracts and WP Plugin

Motivation: Relational Properties

Specification and Proof of Relational Propserties with RPP

Demo with Relational Property Prover (RPP)

Conclusion

# Outline

Frama-C, a platform for analysis of C code

Function Contracts and WP Plugin

Motivation: Relational Properties

Specification and Proof of Relational Propserties with RPP

Demo with Relational Property Prover (RPP)

Conclusion

# A brief history

- 90's: CAVEAT, Hoare logic-based tool for C code at CEA
- 2000's: CAVEAT used by Airbus during certification process of the A380 (DO-178 level A qualification)
- 2008: First public release of Frama-C (Hydrogen)
- 2012: New Hoare-logic based plugin WP developed at CEA LIST
- Today: Frama-C v.16 Sulfur
  - Multiple projects around the platform
  - A growing community of users. . .
  - and of developers
- Used by, or in collaboration with, several industrial partners

AIRBUS
AtoS
eDF
AREVA
IRSN INSTITUT DE RADIOPROTECTION ET DE SÛRETÉ NUCLÉAIRE
BUREAU VERITAS 1828
TRUST IN SOFT
DASSAULT AVIATION
MITSUBISHI ELECTRIC

# Frama-C at a glance

Software Analyzers

- A Framework for Modular Analysis of C code
- Developed at CEA LIST
- Released under LGPL license
- Kernel based on CIL [Necula et al. (Berkeley), CC 2002]
- ACSL annotation language
- Extensible plugin oriented platform
  - Collaboration of analyses over same code
  - Inter plugin communication through ACSL formulas
  - Adding specialized plugins is easy
- `http://frama-c.com/` [Kirchner et al. FAC 2015]

# ACSL: ANSI/ISO C Specification Language

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

## Basic Components

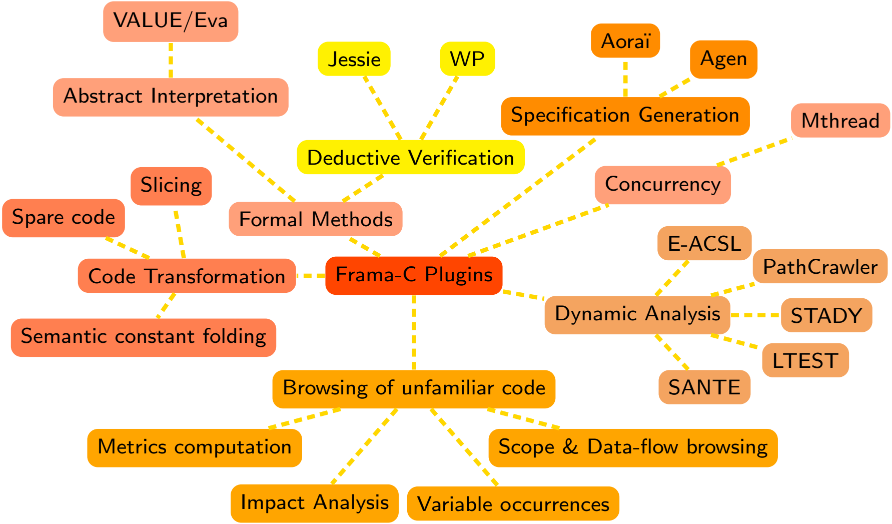
- First-order logic
- Pure C expressions
- C types + $\mathbb{Z}$ (integer) and $\mathbb{R}$ (real)
- Built-in predicates and logic functions particularly over pointers: `\valid(p) \valid(p+0..2), \separated(p+0..2,q+0..5), \block_length(p)`

# Example: a C program annotated in ACSL

```
/*@ requires n>=0 && \valid(t+(0..n-1));
    assigns \nothing;
    ensures \result != 0 <==>
      (\forall integer j; 0 <= j < n ==> t[j] == 0);
*/
int all_zeros(int t[], int n) {
  int k;
  /*@ loop invariant 0 <= k <= n;
      loop invariant \forall integer j; 0<=j<k ==> t[j]==0;
      loop assigns k;
      loop variant n-k;
  */
  for(k = 0; k < n; k++)
    if (t[k] != 0)
      return 0;
  return 1;
}
```

Can be proven in Frama-C/WP

# Main plugins



# Outline

Frama-C, a platform for analysis of C code

Function Contracts and WP Plugin

Motivation: Relational Properties

Specification and Proof of Relational Propserties with RPP

Demo with Relational Property Prover (RPP)

Conclusion

# Function contracts

$$\{P\}S\{Q\}$$

for all states $\sigma$ which satisfy P, if the execution of program S terminates in state $\sigma'$ then $\sigma'$ satisfies Q.

$$\{1000 > x\}result = x + 1\{result == x + 1\}$$

Pre-condition: assumed to be true on entry

```
/*@ requires 1000 > x;
@ ensures \result == x + 1;
@ assigns \nothing;*/
int f (int x){
  return x + 1;
}
```

# Function contracts

$$\{P\}S\{Q\}$$

for all states $\sigma$ which satisfy P, if the execution of program S terminates in state $\sigma'$ then $\sigma'$ satisfies Q.

$$\{1000 > x\}result = x + 1\{result == x + 1\}$$

Post-conditon:
to be
verified on exit

```
/*@ requires 1000 > x;
@ ensures \result == x + 1;
@ assigns \nothing;*/
int f (int x){
  return x + 1;
}
```

# Function contracts

$$\{P\}S\{Q\}$$

for all states $\sigma$ which satisfy P, if the execution of program S terminates in state $\sigma'$ then $\sigma'$ satisfies Q.

$$\{1000 > x\}\mathit{result} = x + 1\{\mathit{result} == x + 1\}$$

Frame rule:
to be
verified on exit

```
/*@ requires 1000 > x;
@ ensures \result == x + 1;
@ assigns \nothing;*/
int f (int x){
  return x + 1;
}
```

List of locations in the global memory that may have a different value before and after the call

# WP plugin

- Based on Weakest Precondition (WP) calculus
- Input: a function and its specification in ACSL
- Output: verification conditions (VCs)
- Relies on Automatic and Interactive Theorem Provers to discharge the VCs
  - Alt-Ergo, Simplify, Z3, Yices, CVC4 ...
- An Interactive Proof Assitant can be used to finish the proof

# Outline

Frama-C, a platform for analysis of C code

Function Contracts and WP Plugin

Motivation: Relational Properties

Specification and Proof of Relational Propserties with RPP

Demo with Relational Property Prover (RPP)

Conclusion

# Motivation: Relational Properties

$$\forall x1, x2 \in \mathbb{Z} : \; x1 < x2 \Rightarrow f(x1) < f(x2)$$

$$\forall \; Msg, Key; Decrypt(Encrypt(Msg, Key), Key) = Msg$$

$$(A + B)^T = (A^T + B^T).$$

# Motivation: Relational Properties

$$\forall x1, x2 \in \mathbb{Z} : \; x1 < x2 \Rightarrow f(x1) < f(x2)$$

$$\forall \; Msg, Key; Decrypt(Encrypt(Msg, Key), Key) = Msg$$

$$(A + B)^T = (A^T + B^T).$$

- ACSL is not sufficient to specify these properties
- WP is not sufficient to prove these properties

# Motivation: Relational Properties

Relational Properties (RPs): properties

- Invoking at least two function calls
- Invoking possibly dissimilar functions
- Invoking possible nested calls

# Motivation: Relational Properties

Relational Properties (RPs): properties

- Invoking at least two function calls
- Invoking possibly dissimilar functions
- Invoking possible nested calls

Goal:

- Specification of RPs in Frama-C
- Proof RPs in Frama-C
- Usage of RPs as hypotheses to prove other properties

# Outline

Frama-C, a platform for analysis of C code

Function Contracts and WP Plugin

Motivation: Relational Properties

Specification and Proof of Relational Propserties with RPP

Demo with Relational Property Prover (RPP)

Conclusion

# Proposal: specification of RPs

Extension of ACSL:

- New clause relational
- New built-in \callpure

Example with a pure function:

```
/*@ requires 1000 > x;
@ relational \forall int x1,x2; x1 < x2  ==>
@             \callpure(f,x1) < \callpure(f,x2);
@ assigns \nothing;*/
int f (int x){
  return x + 1;
}
```

Property: $\forall x1, x2 \in \mathbb{Z} : \ x1 < x2 \Rightarrow f(x1) < f(x2)$

# Proposal: Proof of RPs

- Inspired by Self-composition [Barthe et al (2011)]
- Inline involved function calls
- Express the RP as a standard ACSL assertion

```
void relational_wrapper_1(int x1, int x2){
  int return_1 = x1 + 1;
  int return_2 = x2 + 1;
  /*@ assert x1 < x2 ==> return_1 < return_2; */
  return;
}
```

Property: $\forall x1, x2 \in \mathbb{Z} :\ x1 < x2 \Rightarrow f(x1) < f(x2)$

# Proposal: Proof of RPs

- Inspired by Self-composition [Barthe et al (2011)]
- Inline involved function calls
- Express the RP as a standard ACSL assertion

```
void relational_wrapper_1(int x1, int x2){
  int return_1 = x1 + 1;
  int return_2 = x2 + 1;
  /*@ assert x1 < x2 ==                urn_2; */
  return;
}
```

Inlining of f(x1)

Property: $\forall x1, x2 \in \mathbb{Z}: \; x1 < x2 \Rightarrow f(x1) < f(x2)$

# Proposal: Proof of RPs

- Inspired by Self-composition [Barthe et al (2011)]
- Inline involved function calls
- Express the RP as a standard ACSL assertion

```
void relational_wrapper_1(int x1, int x2){
  int return_1 = x1 + 1;
  int return_2 = x2 + 1;
  /*@ assert x1 < x2 ==> return_1 < return_2; */
  return;
}
```

Inlining of f(x2)

Property: $\forall x1, x2 \in \mathbb{Z}: \; x1 < x2 \Rightarrow f(x1) < f(x2)$

# Proposal: Proof of RPs

- Inspired by Self-composition [Barthe et al (2011)]
- Inline involved function calls
- Express the RP as a standard ACSL assertion

```
void relational_wrapper_1(int x1, int x2){
  int return_1 = x1 + 1;
  int return_2 = x2 + 1;
  /*@ assert x1 < x2 ==> return_1 < return_2; */
  return;
}
```

Express the RP in ACSL

Property: $\forall x1, x2 \in \mathbb{Z}: \; x1 < x2 \Rightarrow f(x1) < f(x2)$

# Proposal: Using RPs as hypotheses

```
/*@ axiomatic RP_axiom {
@  logic int f_acsl(int x) ;
@
@ lemma RP_lemma:
@   \forall int x1, int x2; x1 < x2 ==>
@        f_acsl(x1) < f_acsl(x2);
}*/

/*@ requires 1000 > x;
@ assigns \nothing;
@ ensures \result == f_acsl(x);*/
int f(int x) {
  return x + 1;
}

/*@ relational \forall int x1 , x2; x1 < x2  ==>
@            \callpure(g,x1) < \callpure(g,x2);*/
int g(int x){
  return f(x) + 1;
}
```

# Proposal: Using RPs as hypotheses

Valid iff previous assertion is proved

```
/*@ axiomatic RP_axiom {
@  logic int f_acsl(int x) ;
@
@ lemma RP_lemma:
@   \forall int x1, int x2; x1 < x2 ==>
@         f_acsl(x1) < f_acsl(x2);
}*/

/*@ requires 1000 > x;
@ assigns \nothing;
@ ensures \result == f_acsl(x);*/
int f(int x) {
  return x + 1;
}

/*@ relational \forall int x1 , x2; x1 < x2  ==>
@            \callpure(g,x1) < \callpure(g,x2);*/
int g(int x){
  return f(x) + 1;
}
```

# Proposal: Using RPs as hypotheses

```
/*@ axiomatic RP_axiom {
@  logic int f_acsl(int x) ;
@
@ lemma RP_lemma:
@   \forall int x1, int x2; x1 < x2 ==>
@        f_acsl(x1) < f_acsl(x2);
}*/

/*@ requires 1000 > x;
@ assigns \nothing;
@ ensures \result == f_acsl(x);*/
int f(int x) {
  return x + 1;
}

/*@ relational \forall int x1 , x2; x1 < x2  ==>
@            \callpure(g,x1) < \callpure(g,x2);*/
int g(int x){
  return f(x) + 1;
}
```

Valid iff
previous assertion
is proved

Bridge between
f and f_acsl

# Proposal: Using RPs as hypotheses

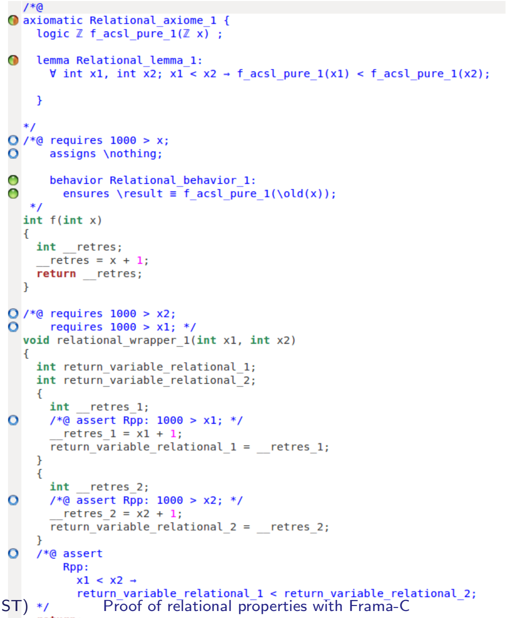
```
/*@ axiomatic RP_axiom {
@  logic int f_acsl(int x) ;
@
@ lemma RP_lemma:
@   \forall int x1, int x2; x1 < x2 ==>
@         f_acsl(x1) < f_acsl(x2);
}*/

/*@ requires 1000 > x;
@ assigns \nothing;
@ ensures \result == f_acsl(x);*/
int f(int x) {
  return x + 1;
}

/*@ relational \forall int x1 , x2; x1 < x2  ==>
@            \callpure(g,x1) < \callpure(g,x2);*/
int g(int x){
  return f(x) + 1;
}
```

Valid iff previous assertion is proved

Bridge between f and f_acsl

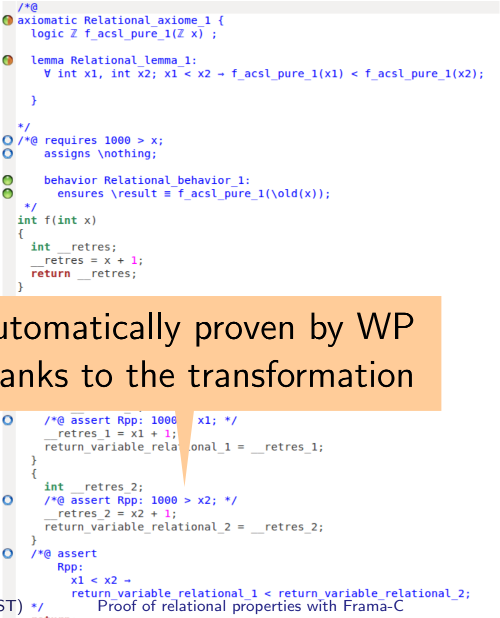
The RP can be used in another proof

# Relational properties with RPP



```
/*@
axiomatic Relational_axiome_1 {
  logic ℤ f_acsl_pure_1(ℤ x) ;

  lemma Relational_lemma_1:
    ∀ int x1, int x2; x1 < x2 → f_acsl_pure_1(x1) < f_acsl_pure_1(x2);

  }

*/
/*@ requires 1000 > x;
    assigns \nothing;

    behavior Relational_behavior_1:
      ensures \result ≡ f_acsl_pure_1(\old(x));
 */
int f(int x)
{
  int __retres;
  __retres = x + 1;
  return __retres;
}

/*@ requires 1000 > x2;
    requires 1000 > x1; */
void relational_wrapper_1(int x1, int x2)
{
  int return_variable_relational_1;
  int return_variable_relational_2;
  {
    int __retres_1;
    /*@ assert Rpp: 1000 > x1; */
    __retres_1 = x1 + 1;
    return_variable_relational_1 = __retres_1;
  }
  {
    int __retres_2;
    /*@ assert Rpp: 1000 > x2; */
    __retres_2 = x2 + 1;
    return_variable_relational_2 = __retres_2;
  }
  /*@ assert
      Rpp:
        x1 < x2 →
        return_variable_relational_1 < return_variable_relational_2;
  */
```

# Relational properties with RPP



```
/*@
axiomatic Relational_axiome_1 {
  logic ℤ f_acsl_pure_1(ℤ x) ;

  lemma Relational_lemma_1:
    ∀ int x1, int x2; x1 < x2 → f_acsl_pure_1(x1) < f_acsl_pure_1(x2);

  }

*/
/*@ requires 1000 > x;
    assigns \nothing;

    behavior Relational_behavior_1:
      ensures \result ≡ f_acsl_pure_1(\old(x));
 */
int f(int x)
{
  int __retres;
  __retres = x + 1;
  return __retres;
}
```

```
    /*@ assert Rpp: 1000    x1; */
    __retres_1 = x1 + 1;
    return_variable_rela  onal_1 = __retres_1;
  }
  {
    int __retres_2;
    /*@ assert Rpp: 1000 > x2; */
    __retres_2 = x2 + 1;
    return_variable_relational_2 = __retres_2;
  }
  /*@ assert
      Rpp:
        x1 < x2 →
        return_variable_relational_1 < return_variable_relational_2;
  */
```

Automatically proven by WP thanks to the transformation

# Outline

Frama-C, a platform for analysis of C code

Function Contracts and WP Plugin

Motivation: Relational Properties

Specification and Proof of Relational Propserties with RPP

Demo with Relational Property Prover (RPP)

Conclusion

# Outline

Frama-C, a platform for analysis of C code

Function Contracts and WP Plugin

Motivation: Relational Properties

Specification and Proof of Relational Propserties with RPP

Demo with Relational Property Prover (RPP)

Conclusion

# Conclusion

- Relational Properties:
  - relating several function calls

- Support of RPs in Frama-C
  - Extension of the ACSL spec. language for RPs
  - Code transformation for the proof of RPs
  - And using RPs as hypotheses for other proofs
  - Requires the RPP (Relational Property Prover) Plugin

- Ongoing and Future Work
  - RPs for functions with side effects
  - RPs for functions with pointers
  - RPs for recursive functions