Using Frama-C

Ming-Hsien Tsai, 2022/11/23

Frama-C

- A suite of tools for the analysis of source code written in C
  - A modified version of CIL (C Intermediate Language) as the kernel
  - Static and dynamic analysis techniques
  - Extensible architecture
  - Collaborations across analyzers
  - Bug free versus bug finding

A Simple Program

```
int abs(int x) {
  if (x < 0) return -x;
  else return x;
}
```

Is this program correct?

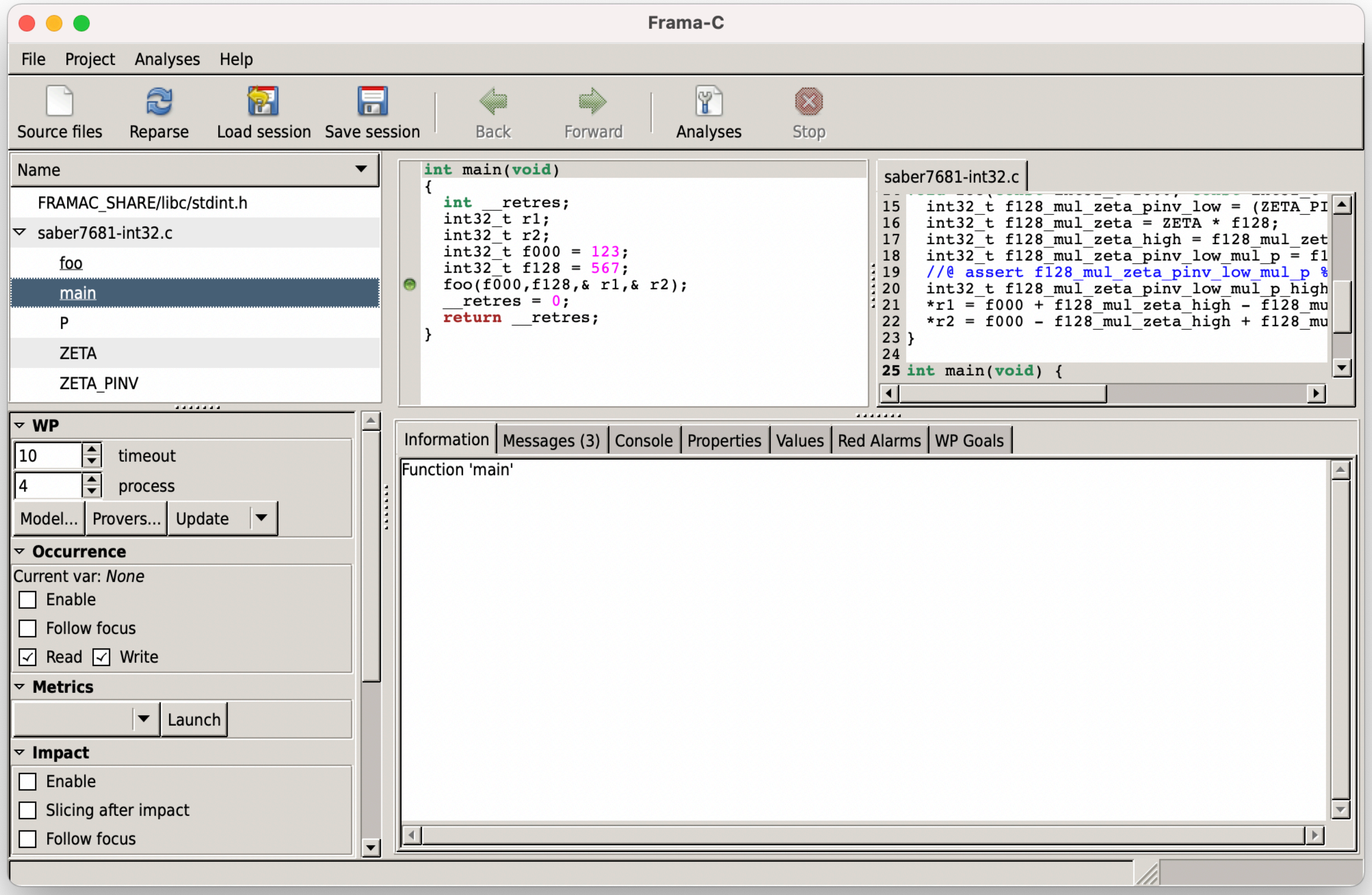
Installation

- Installation instructions: https://frama-c.com/html/get-frama-c.html
- It is recommended to install Frama-C via opam (https://opam.ocaml.org)
  - frama-c
  - why3
  - why3-coq
  - coq
  - coqide
  - alt-ergo

Basic Usage

$ frama-c -PLUGIN -OPTION$_1$ -OPTION$_2$ … file.c -OPTION$_i$ …

$ frama-c-gui -PLUGIN -OPTION$_1$ -OPTION$_2$ … file.c -OPTION$_i$ …



Action Order

- Actions are applied in order according to -then.
  - $ frama-c ARGS-ACT-1 -then ARGS-ACT-2 -then ARGS-ACT-3 ...
- The action after -then-on PROJECT is applied after PROJECT
- The action specified after -then-last is applied on the last project created by a program transformer

Value Analysis via EVA

- Based on abstract interpretation
- Compute variation domains for variables
- Can detect overflow problems
- Recursive calls are not supported

EVA Example 1

```
int dbl(int n) {
  return n * 2;
}

int main(void) {
  int n, m = 0;
  printf("Enter an integer: ");
  scanf("%d", &n);
  if (0 <= n && n <= 3)
    m = dbl(n);
  return 0;
}
```

```
[eva:final-states] Values at end of function main:
  n ∈ [--..--]
  m ∈ {0; 2; 4; 6}
  __retres ∈ {0}
  S___fc_stdin[0..1] ∈ [--..--]
  S___fc_stdout[0..1] ∈ [--..--]
```

[--..--]: the set of all integers that fit within the type of the variable or expression

$ frama-c -eva dbl-1.c

EVA Example 1

-Wider Range-

```c
int dbl(int n) {
  return n * 2;
}

int main(void) {
  int n, m = 0;
  printf("Enter an integer: ");
  scanf("%d", &n);
  if (0 <= n && n <= 9)
    m = dbl(n);
  return 0;
}
```

```
[eva:final-states] Values at end of function main:
  n ∈ [--..--]
  m ∈ [0..18],0%2
  __retres ∈ {0}
  S___fc_stdin[0..1] ∈ [--..--]
  S___fc_stdout[0..1] ∈ [--..--]
```

[L..H]: { n | L ≤ n ≤ H }
[L..H],r%m: { n | L ≤ n ≤ H, and n % m = r }

-eva-ilevel <n>: controls the maximal number of integers that should be precisely represented as a set

EVA Example 2

-Loops-

```c
int main(void) {
  int x = 0, y = 1;
  for (int i = 0; i < 10; i++) {
    int tmp = x;
    x = y;
    y = tmp + 2 * y;
  }
  int a = x;
  int b = y;
  return 0;
}
```

[eva:final-states] Values at end of function main:
  x ∈ [0..2147483647]
  y ∈ [1..2147483647]
  a ∈ [0..2147483647]
  b ∈ [1..2147483647]
  __retres ∈ {0}

EVA Example 2

-Precision Improvement-

```c
int main(void) {
  int x = 0, y = 1;
  //@ loop unroll 10;
  for (int i = 0; i < 10; i++) {
    int tmp = x;
    x = y;
    y = tmp + 2 * y;
  }
  int a = x;
  int b = y;
  return 0;
}
```

[eva:final-states] Values at end of function main:
  x ∈ {2378}
  y ∈ {5741}
  a ∈ {2378}
  b ∈ {5741}
  __retres ∈ {0}

-eva-auto-loop-unroll <n>: loops with less than <n> iterations will be completely unrolled
-eva-min-loop-unroll <n>: specify the number of iterations to unroll in each loop

Catch Overflow Bugs

```
int abs(int x) {
  if (x < 0) return -x;
  else return x;
}
```

```
$ frama-c -eva abs.c
...
[eva:alarm] abs.c:5: Warning: signed overflow. assert -x ≤ 2147483647;
[eva] done for function main
[eva] abs.c:5: assertion 'Eva,signed_overflow' got final status invalid.
[eva] ====== VALUES COMPUTED ======
...
[eva:summary] ====== ANALYSIS SUMMARY ======
  ----------------------------------------------------------------------------
  2 functions analyzed (out of 2): 100% coverage.
  In these functions, 4 statements reached (out of 12): 33% coverage.
  ----------------------------------------------------------------------------
  No errors or warnings raised during the analysis.
  ----------------------------------------------------------------------------
  1 alarm generated by the analysis:
       1 integer overflow
  1 of them is a sure alarm (invalid status).
...
```

Runtime Assertions via E-ACSL

- Translate an annotated C program into another program with runtime assertions
  - Both programs have the same behavior if no annotation is violated
- Possible usage:
  - Detect undefined behaviors (+RTE)
  - Verification of linear temporal properties (+Aoraï)
  - Verification of security properties (+SecureFlow)

E-ACSL Example 1

```c
/*@
  @ ensures x <= \result && y <= \result;
  @ ensures \result == x || \result == y;
  @*/
int max(int x, int y) {
  if (x < y) return y;
  else return x;
}

int main(void) {
  int x, y, z;
  z = max(x, y);
  return 0;
}
```

$ frama-c -e-acsl max.c -then-last -print

E-ACSL Example 1

```
int __gen_e_acsl_max(int x, int y)
{
  int __gen_e_acsl_at_4;
  int __gen_e_acsl_at_3;
  int __gen_e_acsl_at_2;
  int __gen_e_acsl_at;
  int __retres;
  __gen_e_acsl_at_4 = y;
  __gen_e_acsl_at_3 = x;
  __gen_e_acsl_at_2 = y;
  __gen_e_acsl_at = x;
  __retres = max(x,y);
  {
    …
 }
}
```

```
 {
    int __gen_e_acsl_and;
    int __gen_e_acsl_or;
    if (__gen_e_acsl_at <= __retres) __gen_e_acsl_and =
__gen_e_acsl_at_2 <= __retres;
    else __gen_e_acsl_and = 0;
    __e_acsl_assert(__gen_e_acsl_and,1,"Postcondition","max",
                    "\\old(x) <= \\result && \\old(y) <= \\result",
                    "e-acsl-1.c",3);
    if (__retres == __gen_e_acsl_at_3) __gen_e_acsl_or = 1;
    else __gen_e_acsl_or = __retres == __gen_e_acsl_at_4;
    __e_acsl_assert(__gen_e_acsl_or,1,"Postcondition","max",
                    "\\result == \\old(x) || \\result == \\old(y)",
                    "e-acsl-1.c",4);
    return __retres;
  }
}
```

Every call to max is replaced by a call to __gen_e_acsl_max.

E-ACSL Example 2

-With RTE-

```c
int main(void) {
  int x = 0xffff;
  int y = 0xfff;
  int z = x + y;
  return 0;
}
```

$ frama-c -rte eacsl.c -then -print

```c
int main(void)
{
  int __retres;
  int x = 0xffff;
  int y = 0xfff;
  /*@ assert rte: signed_overflow: -2147483648 ≤ x + y; */
  /*@ assert rte: signed_overflow: x + y ≤ 2147483647; */
  int z = x + y;
  __retres = 0;
  return __retres;
}
```

E-ACSL Example 2

-With RTE+E-ACSL-

```
int main(void) {
  int x = 0xffff;
  int y = 0xfff;
  int z = x + y;
  return 0;
}
```

$ frama-c -rte eacsl.c -then -e-acsl -then-last -print

```
int main(void)
{
  int __retres;
  int x = 0xffff;
  int y = 0xfff;
  /*@ assert rte: signed_overflow: -9223372036854775808 ≤ x + (long)y; */
  /*@ assert rte: signed_overflow: x + (long)y ≤ 9223372036854775807; */
  __e_acsl_assert(x + (long)y <= 2147483647L,1,"Assertion","main",
                  "rte: signed_overflow: x + y <= 2147483647","e-acsl-2.c",4);
  /*@ assert rte: signed_overflow: -9223372036854775808 ≤ x + (long)y; */
  /*@ assert rte: signed_overflow: x + (long)y ≤ 9223372036854775807; */
  __e_acsl_assert(-2147483648L <= x + (long)y,1,"Assertion","main",
                  "rte: signed_overflow: -2147483648 <= x + y","e-acsl-2.c",
                  4);
  /*@ assert rte: signed_overflow: -2147483648 ≤ x + y; */
  /*@ assert rte: signed_overflow: x + y ≤ 2147483647; */
  int z = x + y;
  __retres = 0;
  return __retres;
}
```

Limitations of E-ACSL

- Uninitialized values
  - Runtime error may not occur depending on the compiler
- Incomplete programs
- Recursive functions
- Variadic functions
- Function pointers

```
int main(void) {
  int x;
  /*@ assert x == 0; */
  return 0;
}
```

Test Cases Generation via PathCrawler

- Generate test inputs
- Cover all feasible execution paths
- Based on constraint resolution
- Try it online at http://pathcrawler-online.com:8080/

Program Slicing

- Program slicing computes a subset of program statements that may affect a given set of values called slicing criterion
  - control dependency
  - data dependency

```
…
if (i <= j)
  x = y * 2;
else
  y = y + 3;
return x;
```

```
…
if (i <= j)
  x = y * 2;


return x;
```

slicing criterion: x at the end of the program

Program Slicing

-Example-

```
int max(int m, int n) {
  if (m <= n) return n;
  else return m;
}

int dbl(int m) {
  return m * 2;
}

int f(void) {
  int m = 3, n = 5;
  int a = max(m, n);
  int b = dbl(m);
  /*@ assert b <= 10; */
  return a + b;
}

void main(void) { … }
```

```
/* Generated by Frama-C */
int dbl_slice_1(int m)
{
  int __retres;
  __retres = m * 2;
  return __retres;
}

void f_slice_1(void)
{
  int m = 3;
  int b = dbl_slice_1(m);
  /*@ assert b ≤ 10; */ ;
  return;
}

void main(void) { … }
```

$ frama-c slicing.c -slice-assert f -then-last -print

Deductive Verification via WP

- Based on weakest-precondition calculus
- Relies on external automated provers and proof assistants
- Provers are invoked via Why3 (http://why3.lri.fr)
    - Alt-Ergo
    - CVC4
    - Gappa
    - Princess
    - Vampire
    - Z3
    - Coq
    - PVS
    - Isabelle/HOL

After installation of why3 and external provers, run command `why3 config detect` to detect available provers.

WP Example 1

```
/*@
  @ ensures \result == x + y;
  @ assigns \nothing;
  */
int add(int x, int y) {
  return x + y;
}
```

$ frama-c -wp add.c -then -report

```
[wp] Warning: Missing RTE guards
[wp] 2 goals scheduled
[wp] [Cache] not used
[wp] Proved goals:    2 / 2
  Qed:             2  (0.21ms-0.88ms)
[report] Computing properties status...

--------------------------------------------------------------------------------
--- Properties of Function 'add'
--------------------------------------------------------------------------------

[  Valid  ] Post-condition (file add-1.c, line 2)
            by Wp.typed.
[  Valid  ] Assigns nothing
            by Wp.typed.
[  Valid  ] Default behavior
            by Frama-C kernel.
...
```

WP Example 1
-With RTE-

```
/*@
  @ ensures \result == x + y;
  @ assigns \nothing;
  */
int add(int x, int y) {
  return x + y;
}
```

$ frama-c -wp -wp-rte add.c -then -report

Refine the specification such that the absence of runtime errors can be proven

```
[kernel] Parsing add-1.c (with preprocessing)
[rte] annotating function add
[wp] 4 goals scheduled
[wp] [Alt-Ergo 2.4.1] Goal typed_add_assert_rte_signed_overflow_2 :
Timeout (Qed:0.64ms) (10s)
[wp] [Alt-Ergo 2.4.1] Goal typed_add_assert_rte_signed_overflow :
Timeout (Qed:0.84ms) (10s)
[wp] [Cache] updated:2
[wp] Proved goals:    2 / 4
  Qed:               2  (0.83ms-1ms)
  Alt-Ergo 2.4.1:    0  (interrupted: 2)
[report] Computing properties status...

--------------------------------------------------------------------------------
--- Properties of Function 'add'
--------------------------------------------------------------------------------

[ Partial ] Post-condition (file add-1.c, line 2)
            By Wp.typed, with pending:
             - Assertion 'rte,signed_overflow' (file add-1.c, line 6)
             - Assertion 'rte,signed_overflow' (file add-1.c, line 6)
...
```

WP Example 2

```
/*@ requires \valid(a) && \valid(b);
  @ ensures *a == \old(*b) && *b == \old(*a);
  @ assigns *a, *b;
  @*/
void swap(int *a, int *b)
{
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

void order3(int *a, int *b, int *c) {
  if (*a > *b) swap(a, b);
  if (*a > *c) swap(a, c);
  if (*b > *c) swap(b, c);
}
```

Write a specification for order3

Source: A. Blanchard. Introduction to C program proof with Frama-C and its WP plugin, Creative Commons, 2020.

WP Example 2

-Additional Assertions-

```c
/*@ requires \valid(a) && \valid(b);
  @ ensures *a == \old(*b) && *b == \old(*a);
  @ assigns *a, *b;
  @*/
void swap(int *a, int *b)
{
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

void order3(int *a, int *b, int *c) {
  if (*a > *b) swap(a, b);
  if (*a > *c) swap(a, c);
  if (*b > *c) swap(b, c);
}
```

```c
void test() {
  int a1 = 5, b1 = 3, c1 = 4;

  order3(&a1, &b1, &c1);
  //@ assert a1 == 3 && b1 == 4 && c1 == 5;

  int a2 = 2, b2 = 2, c2 = 2;
  order3(&a2, &b2, &c2);
  //@ assert a2 == 2 && b2 == 2 && c2 == 2;

  int a3 = 4, b3 = 3, c3 = 4;
  order3(&a3, &b3, &c3);
  //@ assert a3 == 3 && b3 == 4 && c3 == 4;

  int a4 = 4, b4 = 5, c4 = 4;
  order3(&a4, &b4, &c4);
  //@ assert a4 == 4 && b4 == 4 && c4 == 5;
}
```

Write a specification for order3 such that all assertions are verified

Deductive Verification with Interactive Prover

- For proof obligations that cannot be discharged by automatic provers, the interactive prover Coq can be used
- We will show how to use Frama-C with Coq by some examples

Field Operations

-Annotated C Code-

```
const int32_t P = 7681;
const int32_t ZETA = 3777;
const int32_t ZETA_PINV = 28865;

/*@
  @ requires \valid(r1) && \valid(r2);
  @ requires \separated(r1, r2, &P, &ZETA, &ZETA_PINV);
  @ requires (-4096 < f000 < 4096);
  @ requires (-4096 < f128 < 4096);
  @ assigns *r1, *r2;
  @*/
void foo(const int32_t f000, const int32_t f128, int32_t *r1, int32_t *r2) {
  int32_t f128_mul_zeta_pinv_low = (ZETA_PINV * f128) % (1 << 16);
  int32_t f128_mul_zeta = ZETA * f128;
  int32_t f128_mul_zeta_high = f128_mul_zeta >> 16;
  int32_t f128_mul_zeta_pinv_low_mul_p = f128_mul_zeta_pinv_low * P;
  //@ assert f128_mul_zeta_pinv_low_mul_p % (1 << 16) == f128_mul_zeta % (1 << 16);
  int32_t f128_mul_zeta_pinv_low_mul_p_high = f128_mul_zeta_pinv_low_mul_p >> 16;
  *r1 = f000 + f128_mul_zeta_high - f128_mul_zeta_pinv_low_mul_p_high;
  *r2 = f000 - f128_mul_zeta_high + f128_mul_zeta_pinv_low_mul_p_high;
}
```

alt-ergo fails to prove the assertion

Field Operations

-Proof Obligations-

```
/*@
  @ requires \valid(r1) && \valid(r2);
  @ requires \separated(r1, r2, &P, &ZETA, &ZETA_PINV);
  @ requires (-4096 < f000 < 4096);
  @ requires (-4096 < f128 < 4096);
  @ assigns *r1, *r2;
  @*/
void foo(const int32_t f000, const int32_t f128, int32_t *r1, int32_t *r2) {
  // (((ZETA_PINV * f128) % (1 << 16)) * P) % (1 << 16) == (ZETA * f128) % (1 << 16)
  int32_t f128_mul_zeta_pinv_low = (ZETA_PINV * f128) % (1 << 16);
  // (f128_mul_zeta_pinv_low * P) % (1 << 16) == (ZETA * f128) % (1 << 16)
  int32_t f128_mul_zeta = ZETA * f128;
  // (f128_mul_zeta_pinv_low * P) % (1 << 16) == f128_mul_zeta % (1 << 16)
  int32_t f128_mul_zeta_high = f128_mul_zeta >> 16;
  // (f128_mul_zeta_pinv_low * P) % (1 << 16) == f128_mul_zeta % (1 << 16)
  int32_t f128_mul_zeta_pinv_low_mul_p = f128_mul_zeta_pinv_low * P;
  //@ assert f128_mul_zeta_pinv_low_mul_p % (1 << 16) == f128_mul_zeta % (1 << 16);
  …
}
```

\valid(r1) && \valid(r2) -> \separated(r1, r2, &P, &ZETA, &ZETA_PINV) -> (-4096 < f000 < 4096) -> (-4096 < f128 < 4096) -> (((ZETA_PINV * f128) % (1 << 16)) * P) % (1 << 16) == (ZETA * f128) % (1 << 16)

Field Operations

-Prove by Hand-

```
  (((ZETA_PINV * f128) % (1 << 16)) * P) % (1 << 16)
= (((28865 * f128) % 65536) * 7681) % 65536
= ((28865 * f128) * 7681) % 65536                # ((a%n) * b)%n = (a * b)%n
= (7681 * (28865 * f128)) % 65536                # a * b = b * a
= ((7681 * 28865) * f128) % 65536                # a * (b * c) = (a * b) * c
= ((7681 * 28865) % 65536 * f128) % 65536        # ((a%n) * b)%n = (a * b)%n
= (3777 * f128) % 65536
= (ZETA * f128) % (1 << 16)
```

```
const int32_t P = 7681;
const int32_t ZETA = 3777;
const int32_t ZETA_PINV = 28865;
```

Field Operations

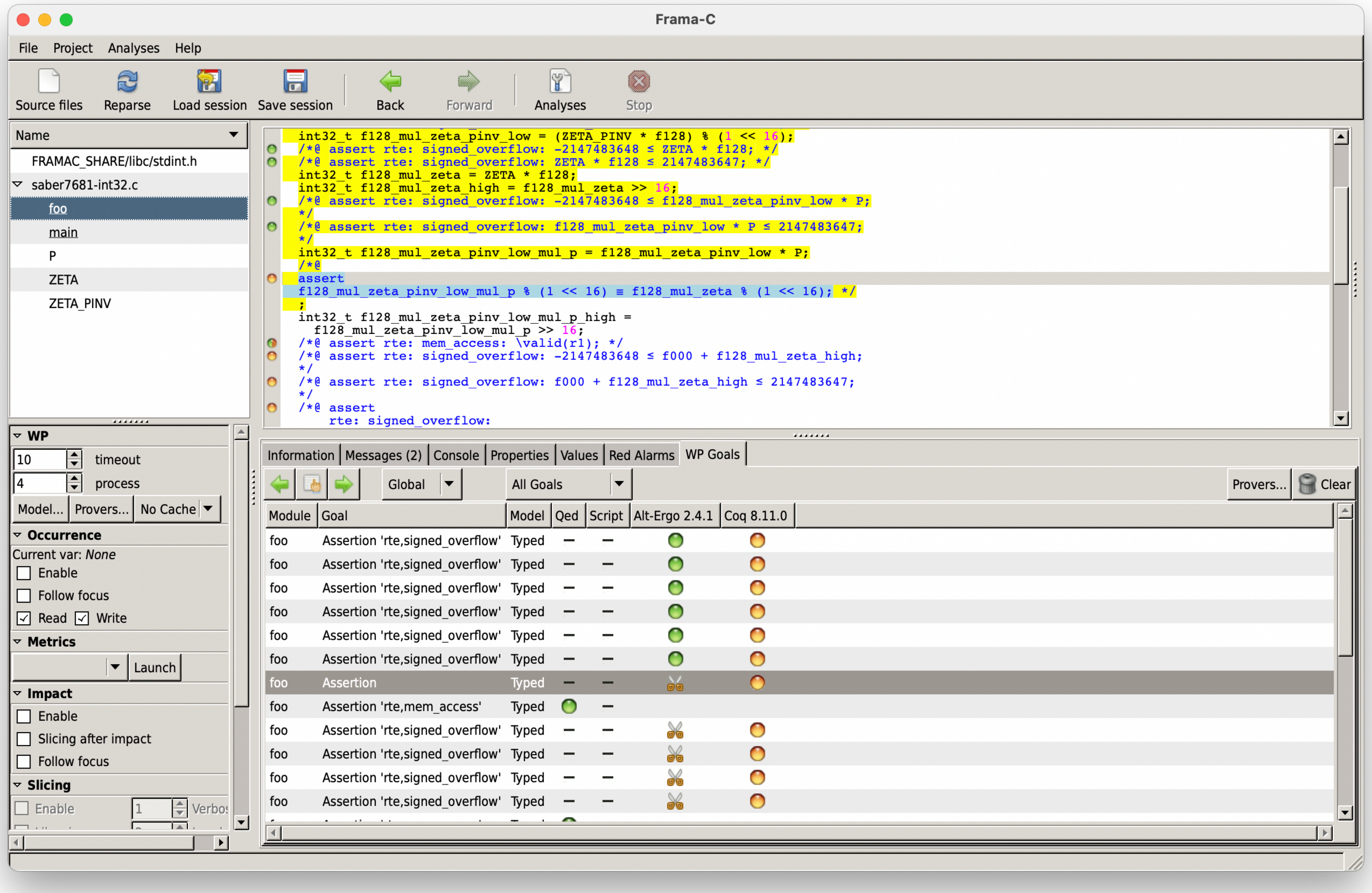
-Prove by Frama-C/Coq-

- Run the following command to invoke Frama-C

  $ frama-c-gui -wp -wp-rte -wp-prover alt-ergo,coq saber7681-int32.c

- Double click the orange circle of the assertion on the Coq column to edit the Coq proof script

- unknown
- surely valid
- surely invalid
- valid under hypothesis
- invalid under hypothesis



Multiplication by Addition

-Annotated C Code-

```
/*@
  @ requires INT_MIN <= x * y <= INT_MAX;
  @ ensures \result == x * y;
  @*/
int mul(int x, int y) {
  int r = 0;
  /*@
    @ loop assigns r, y;
    @ loop invariant r + x * y == \at(x, Pre) * \at(y, Pre);
    @ loop variant \abs(y);
    @*/
  while (y != 0) {
    if (0 < y) { r += x; y -= 1; }
    else { r -= x; y += 1; }
  }
  return r;
}
```

Multiplication by Addition

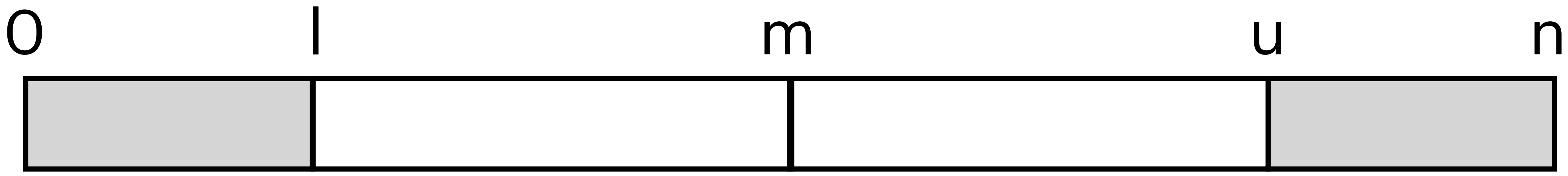
-Prove Goals using Coq-

- Invariant (preserved)
  - .frama-c/wp/interactive/mul_loop_invariant_preserved.v
- Loop invariant at loop (decrease)
  - .frama-c/wp/interactive/mul_loop_variant_decrease.v

Binary Search

-C Code-

```c
int binary_search(long t[], int n, long v) {
  int l = 0, u = n - 1;
  while (l <= u) {
    int m = (l + u) / 2;
    if (t[m] < v) l = m + 1;
    else if (t[m] > v) u = m - 1;
    else return m;
  }
  return -1;
}
```



Source: http://proval.lri.fr/gallery/BinarySearchACSL.en.html

Binary Search

-Function Contract-

```
/*@ requires 0 <= n <= (INT_MAX / 2) && \valid(t + (0..n-1));
  @ ensures -1 <= \result < n;
  @ assigns \nothing;
  @ behavior success:
  @   ensures \result >= 0 ==> t[\result] == v;
  @ behavior failure:
  @   assumes sorted(t,0,n-1);
  @   ensures \result == -1 ==>
  @     \forall integer k; 0 <= k < n ==> t[k] != v;
  @*/
```

```
int binary_search(long t[], int n, long v) {
  int l = 0, u = n - 1;
  while (l <= u) {
    int m = (l + u) / 2;
    if (t[m] < v) l = m + 1;
    else if (t[m] > v) u = m - 1;
    else return m;
  }
  return -1;
}
```

Binary Search

-Loop Annotations-

```
/*@ loop invariant 0 <= l <= u + 1 <= n;
  @ loop assigns l, u;
  @ for failure:
  @   loop invariant
  @   \forall integer k;
  @     0 <= k < n && t[k] == v ==> l <= k <= u;
  @ loop variant u-l;
  @*/
```

```
int binary_search(long t[], int n, long v) {
  int l = 0, u = n - 1;
  while (l <= u) {
    int m = (l + u) / 2;
    if (t[m] < v) l = m + 1;
    else if (t[m] > v) u = m - 1;
    else return m;
  }
  return -1;
}
```

Binary Search

-Prove Goals using Coq-

- Invariant (preserved)
- Loop variant at loop (decrease)
- Post-condition
- Post-condition for `failure’
- Invariant for `failure’ (preserved)

Nistonacci Numbers

-Annotated C Code-

$$\text{nist}(n) = \begin{cases} n & \text{if } n < 2 \\ \text{nist}(n-2) + 2 * \text{nist}(n-1) & \text{otherwise} \end{cases}$$

```
/*@
  @ requires 0 <= n;
  @ ensures n <= \result;
  @ assigns \nothing;
  @*/
int nist_impl(int n) {
  int x = 0, y = 1, i = 0;
  /*@
    @ loop invariant 0 <= i <= n;
    @ loop invariant x == nist(i);
    @ loop invariant y == nist(i + 1);
    @ loop assigns x, y, i;
    @*/
  for (i = 0; i < n; i++) {
    int tmp = x;
    x = y;
    y = tmp + 2 * y;
  }
  return x;
}
```

Source: http://toccata.lri.fr/gallery/nistonacci.fr.html

Nistonacci Numbers

-Prove Goals using Coq-

- Invariant (preserved)
- Post-condition

Summary

- Frama-C is a powerful and flexible tool for deductive program verification
- There are still the following challenges:
  - Writing a correct specification
  - Writing a strong enough loop invariant
  - Analysis of proof failures

Reference:Nikolai Kosmatov, Virgile Prevosto, and Julien Signoles. A Lesson on Proof of Programs with Frama-C. Invited Tutorial Paper. International Conference on Tests and Proofs. 2013.