

# Using Framac-C

**Framac-C 27.1**

**Coq 8.15.2**

**Alt-ergo 2.4.2**

# 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;  
}
```

abs.c

Is this program correct?

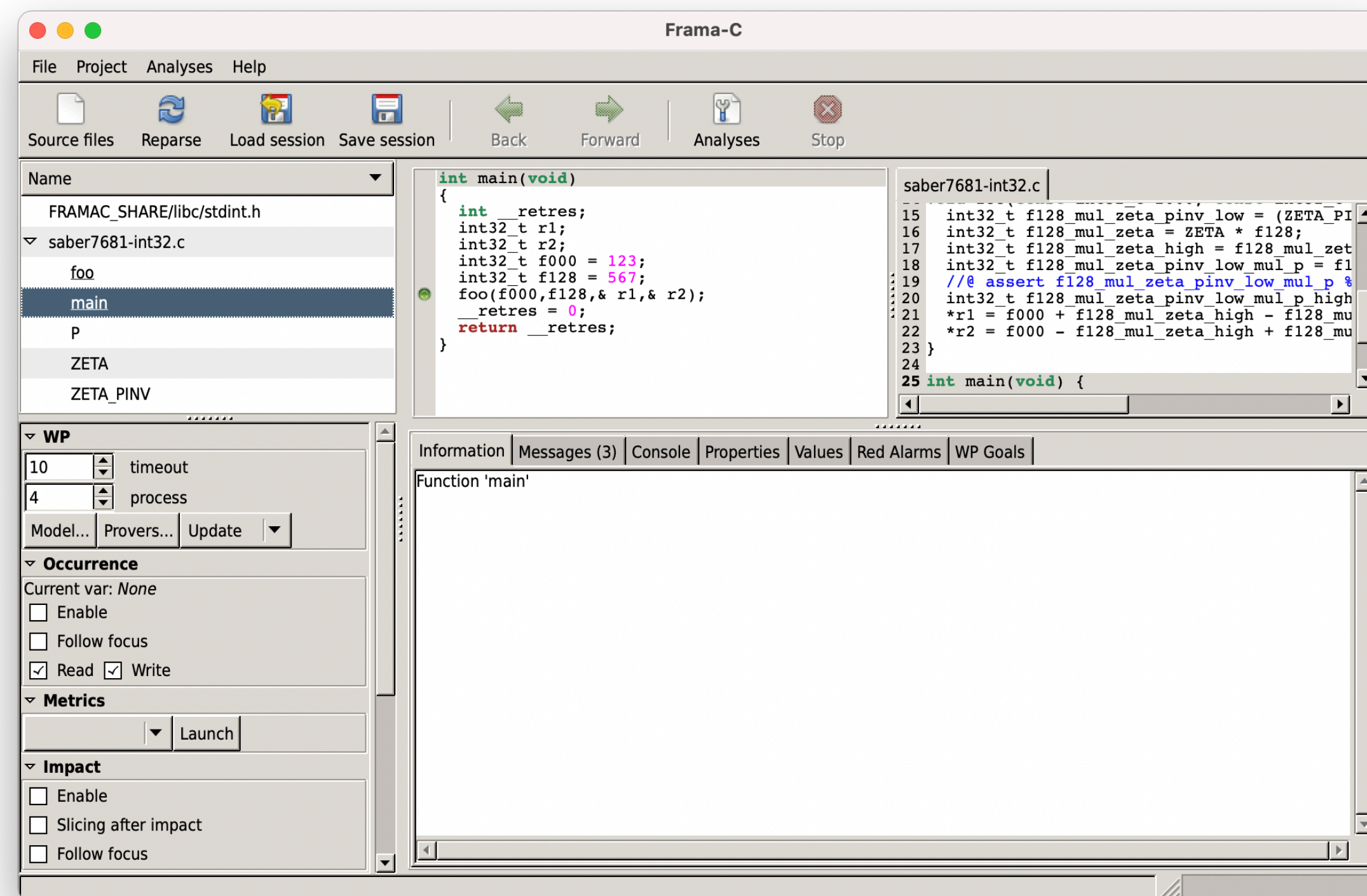
# Installation

- 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<sub>1</sub> -OPTION<sub>2</sub> ... file.c -OPTION<sub>i</sub> ...

\$ frama-c-gui -PLUGIN -OPTION<sub>1</sub> -OPTION<sub>2</sub> ... file.c -OPTION<sub>i</sub> ...



# 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;  
}
```

dbl-1.c

\$ frama-c -eva dbl-1.c

[eva:final-states] Values at end of function main:

$n \in [---...---]$

$m \in \{0; 2; 4; 6\}$

$\_retres \in \{0\}$

$S\_fc\_stdin[0..1] \in [---...---]$

$S\_fc\_stdout[0..1] \in [---...---]$

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



# EVA Example 2

## -Wider Range-

```
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;
}
```

dbl-2.c

[eva:final-states] Values at end of function main:

$n \in [---]$

$m \in [0..18], 0\%2$

$\_retres \in \{0\}$

$S\_fc\_stdin[0..1] \in [---]$

$S\_fc\_stdout[0..1] \in [---]$

**[L..H]:**  $\{ n \mid L \leq n \leq H \}$

**[L..H],r%m:**  $\{ n \mid L \leq n \leq H, \text{ and } n \% m = r \}$

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

# EVA Example 3

## -Loops-

```
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;  
}
```

dbl-3.c

[eva:final-states] Values at end of function main:

$x \in [0..2147483647]$

$y \in [1..2147483647]$

$a \in [0..2147483647]$

$b \in [1..2147483647]$

$\_retres \in \{0\}$

# EVA Example 4

## -Precision Improvement-

```
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;
}
```

dbl-4.c

[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;  
}
```

abs.c

```
$ frama-c -eva -main abs abs.c
```

```
[eva:alarm] dbl-3.c:6: Warning: signed overflow. assert 2 * y ≤ 2147483647;
```

```
[eva:alarm] dbl-3.c:6: Warning:
```

```
signed overflow. assert tmp + (int)(2 * y) ≤ 2147483647;
```

```
[eva] done for function main
```

```
[eva] ===== VALUES COMPUTED =====
```

```
[eva:final-states] Values at end of function main:
```

```
x ∈ [0..2147483647]
```

```
y ∈ [1..2147483647]
```

```
a ∈ [0..2147483647]
```

```
b ∈ [1..2147483647]
```

```
_retres ∈ {0}
```

```
[eva:summary] ===== ANALYSIS SUMMARY =====
```

```
...
```

# 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

```
/*@
  @ 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;
}
```

max.c

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

# E-ACSL Example 1

```
/*@
int __gen_e_acsl_max(int x, int y) result;
{
  int __gen_e_acsl_at_2;
  int __gen_e_acsl_at;
  int __retres;
  __gen_e_acsl_at = x;
  __gen_e_acsl_at_2 = y;
  __retres = max(x,y);
  {
    ..
  }
}
return 0;
}
```

max.c

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

Every call to max is replaced by a call to \_\_gen\_e\_acsl\_max.

# E-ACSL Example 1

```
/*@
int __gen_e_acsl_max(int
{
  int __gen_e_acsl_at_2;
  int __gen_e_acsl_at;
  int __retres;
  __gen_e_acsl_at = x;
  __gen_e_acsl_at_2 = y;
  __retres = max(x,y);
  {
  ...
  }
}

return 0;
}
```

```
int __gen_e_acsl_and;
int __gen_e_acsl_or;
__e_acsl_assert_data_t __gen_e_acsl_assert_data = {.values = (void *)0};
__e_acsl_assert_register_int(& __gen_e_acsl_assert_data, "\\old(x)", 0,
                             __gen_e_acsl_at);
__e_acsl_assert_register_int(& __gen_e_acsl_assert_data, "\\result", 0,
                             __retres);
if (__gen_e_acsl_at <= __retres) {
  __e_acsl_assert_register_int(& __gen_e_acsl_assert_data, "\\old(y)", 0,
                              __gen_e_acsl_at_2);
  __e_acsl_assert_register_int(& __gen_e_acsl_assert_data, "\\result", 0,
                              __retres);
  __gen_e_acsl_and = __gen_e_acsl_at_2 <= __retres;
}
else __gen_e_acsl_and = 0;
...
```

**\_\_gen\_e\_acsl\_and == 1 iff x <= \result && y <= \result**

max.c

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

Every call to max is replaced by a call to \_\_gen\_e\_acsl\_max.



# E-ACSL Example 2

## -With RTE-

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

eacsl.c

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

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

# E-ACSL Example 2

## -With RTE+E-ACSL-

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

eacsl.c

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

```
int main(void)  
{  
    int __retres;  
    __e_acsl_memory_init((int *)0,(char **)0,8UL);  
    int x = 0xffff;  
    int y = 0xffff;  
    {  
        ...  
        /*@ assert rte: signed_overflow: -9223372036854775808 ≤ x + (long)y; */  
        /*@ assert rte: signed_overflow: x + (long)y ≤ 9223372036854775807; */  
        __e_acsl_assert(x + (long)y ≤ 2147483647L,& __gen_e_acsl_assert_data);  
        ...  
        /*@ assert rte: signed_overflow: -9223372036854775808 ≤ x + (long)y; */  
        /*@ assert rte: signed_overflow: x + (long)y ≤ 9223372036854775807; */  
        __e_acsl_assert(-2147483648L ≤ x + (long)y,& __gen_e_acsl_assert_data_2);  
        __e_acsl_assert_clean(& __gen_e_acsl_assert_data_2);  
    }  
    /*@ assert rte: signed_overflow: -2147483648 ≤ x + y; */  
    /*@ assert rte: signed_overflow: x + y ≤ 2147483647; */  
    int z = x + y;  
    __retres = 0;  
    __e_acsl_memory_clean();  
    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;  
}
```

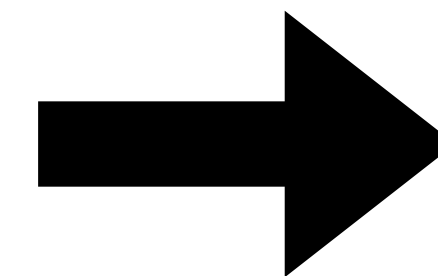
# Test Cases Generation via PathCrawler

- 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

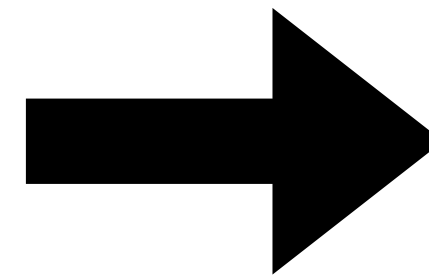
## -Example 1-

```
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; */
    int c = b + 100;
    return c;
}

void main(void) { f(); ... }
```



```
/* 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) { f_slice_1(); ... }
```

slicing-1.c

\$ frama-c slicing-1.c -slice-assert f -then-last -print

# Slicing Criteria

## -Code Observation-

- -slice-calls f1,...,fn: calls to functions f1,...,fn
- -slice-return f1,...,fn: returned values of functions f1,...,fn
- -slice-value v1,...,vn: left-values at the end of the entry function (specified by -main)
- -slice-wr v1,...,vn: write accesses to left-values
- -slice-rd v1,...,vn: read accesses to left-values
- -slice-pragma f1,...,fn: slicing pragmas in the code of functions f1,...,fn

Ref: <https://frama-c.com/fc-plugins/slicing.html>

# Slicing Criteria

## -Proving Properties-

- -slice-assert  $f_1, \dots, f_n$ : assertions of functions  $f_1, \dots, f_n$
- -slice-loop-inv  $f_1, \dots, f_n$ : loop invariants of functions  $f_1, \dots, f_n$
- -slice-loop-var  $f_1, \dots, f_n$ : loop variants of functions  $f_1, \dots, f_n$
- -slice-threat  $f_1, \dots, f_n$ : threats (emitted by Eva) of functions  $f_1, \dots, f_n$



# Program Slicing

## -Example 2: Case 1 of -slice-rd-

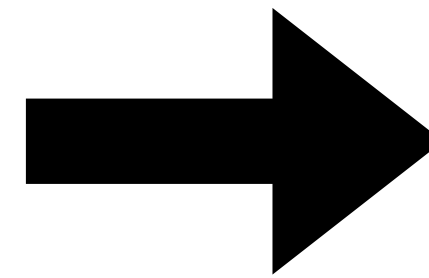
```
int max(int m, int n) { ... }

int dbl(int m) { ... }

int f(void) {
    int m = 3, n = 5;
    int a = max(m, n);
    int b = dbl(m);
    /*@ assert b <= 10; */
    int c = b + 100;
    return c;
}

void main(void) { f(); ... }
```

slicing-1.c



```
/* Generated by Frama-C */
void f(void)
{
    return;
}
```

\$ frama-c slicing-1.c -slice-rd c -main f -then-last -print

# Program Slicing

## -Example 3: Case 2 of -slice-rd-

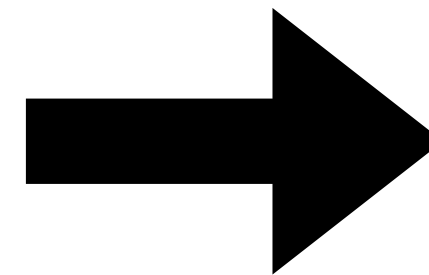
```
int max(int m, int n) { ... }

int dbl(int m) { ... }

int f(void) {
    int m = 3, n = 5;
    int a = max(m, n);
    int b = dbl(m);
    /*@ assert b <= 10; */
    int c = b + 100;
    return c + a;
}

void main(void) { f(); ... }
```

slicing-2.c



```
$ frama-c slicing-2.c -slice-rd c -main f -then-last -print
```

# Program Slicing

## -Example 3: Case 2 of -slice-rd-

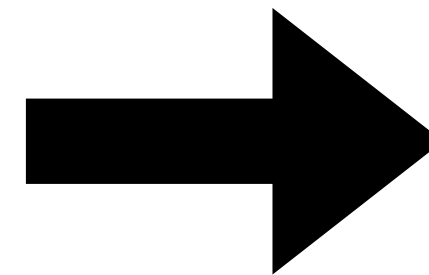
```
int max(int m, int n) { ... }

int dbl(int m) { ... }

int f(void) {
    int m = 3, n = 5;
    int a = max(m, n);
    int b = dbl(m);
    /*@ assert b <= 10; */
    int c = b + 100;
    return c + a;
}

void main(void) { f(); ... }
```

slicing-2.c



```
/* Generated by Frama-C */
int max_slice_1(int n)
{ ... }

int dbl_slice_1(int m)
{ ... }

void f(void)
{
    int __retres;
    int m = 3;
    int n = 5;
    int a = max_slice_1(n);
    int b = dbl_slice_1(m);
    /*@ assert b ≤ 10; */ ;
    int c = b + 100;
    __retres = c + a;
    return;
}
```

\$ frama-c slicing-2.c -slice-rd c -main f -then-last -print

# Program Slicing

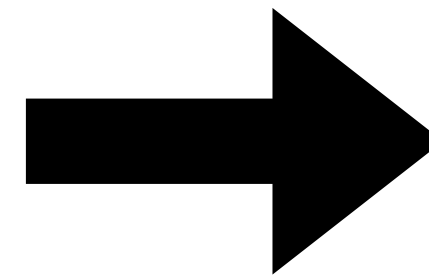
## -Example 4: -slice-pragma-

```
int max(int m, int n) { ... }

int dbl(int m) { ... }

int f(void) {
    int m = 3, n = 5;
    int a = max(m, n);
    /*@ slice pragma expr a; */
    /*@ slice pragma stmt; */
    int b = dbl(m);
    /*@ assert b <= 10; */
    int c = b + 100;
    return c + a;
}

void main(void) { f(); ... }
```



slicing-3.c

\$ frama-c slicing-3.c -slice-pragma f -main f -then-last -print

# Program Slicing

## -Example 4: -slice-pragma-

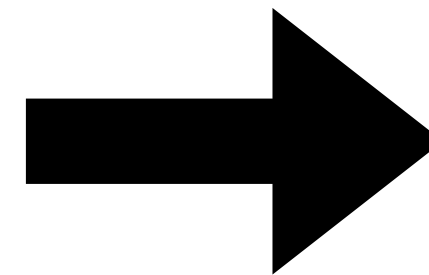
```
int max(int m, int n) { ... }

int dbl(int m) { ... }

int f(void) {
    int m = 3, n = 5;
    int a = max(m, n);
    /*@ slice pragma expr a; */
    /*@ slice pragma stmt; */
    int b = dbl(m);
    /*@ assert b <= 10; */
    int c = b + 100;
    return c + a;
}

void main(void) { f(); ... }
```

slicing-3.c



```
/* Generated by Frama-C */
int max_slice_1(int n)
{ ... }

int dbl_slice_1(int m)
{ ... }

void f(void)
{
    int __retres;
    int m = 3;
    int n = 5;
    int a = max_slice_1(n);
    /*@ slice pragma expr a; */ ;
    /*@ slice pragma stmt; */
    int b = dbl_slice_1(m);
    /*@ assert b ≤ 10; */ ;
    return;
}
```

\$ frama-c slicing-3.c -slice-pragma f -main 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;
}
```

add.c

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

```
[kernel] Parsing add.c (with preprocessing)
[wp] Warning: Missing RTE guards
[wp] 2 goals scheduled
[wp] [Cache] not used
[wp] Proved goals:  2 / 2
  Qed:          2
[report] Computing properties status...
```

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

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

...

# WP Example 2

## -With RTE-

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

add.c

```
$ 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.c (with preprocessing)
[rte:annot] annotating function add
[wp] 4 goals scheduled
[wp] [Timeout] typed_add_assert_rte_signed_overflow_2 (Alt-Ergo) (Cached)
[wp] [Timeout] typed_add_assert_rte_signed_overflow (Alt-Ergo) (Cached)
[wp] [Cache] updated:2
[wp] Proved goals: 2 / 4
Qed: 2
Timeout: 2
[report] Computing properties status...
...
[ Partial ] Post-condition (file add.c, line 2)
By Wp.typed, with pending:
- Assertion 'rte,signed_overflow' (file add.c, line 6)
- Assertion 'rte,signed_overflow' (file add.c, line 6)
...
[ - ] Assertion 'rte,signed_overflow' (file add.c, line 6)
      tried with Wp.typed.
[ - ] Assertion 'rte,signed_overflow' (file add.c, line 6)
      tried with Wp.typed.
...
28
```



# WP Example 3

```
/*@ 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);
}
```

order3.c

Write a specification for order3

# WP Example 3

```
/*@ requires \valid(a) && \valid(b);  
@ ensures *a == \old(*b) && *b == \old(*a);  
@ assigns *a,  
@*/
```

```
void swap(int *  
{  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

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

```
/*@  
@ requires \valid(a) && \valid(b) && \valid(c) && \separated(a, b, c);  
@ ensures *a <= *b <= *c;  
@ ensures { *a, *b, *c } == { \old(*a), \old(*b), \old(*c) };  
@ assigns *a, *b, *c;  
@*/
```

```
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);  
}
```

sol/order3-annotated-1.c

order3.c

# WP Example 3

## -Additional Assertions-

```
/*@ 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);
}
```

sol/order3-annotated-2.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;
}
```

# WP Example 3

## -Additional Assertions-

```
/*@ 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);
}
```

With the previous annotation

sol/order3-annotated-2.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

# WP Example 3

## -Refined Annotations I-

```
/*@
  @ requires \valid(a) && \valid(b) && \valid(c) && \separated(a, b, c);
  @ ensures *a <= *b <= *c;
  @ ensures { *a, *b, *c } == { \old(*a), \old(*b), \old(*c) };
  @ ensures (\old(*a) == \old(*b)) ==>
             (*a == *b == \old(*a) || *b == *c == \old(*a) || *c == *a == \old(*a));
  @ ensures (\old(*b) == \old(*c)) ==>
             (*a == *b == \old(*b) || *b == *c == \old(*b) || *c == *a == \old(*b));
  @ ensures (\old(*c) == \old(*a)) ==>
             (*a == *b == \old(*c) || *b == *c == \old(*c) || *c == *a == \old(*c));
  @ assigns *a, *b, *c;
  */
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);
}
```

sol/order3-annotated-3.c

# WP Example 3

## -Refined Annotations II-

```
/*@
@ requires \valid(a) && \valid(b) && \valid(c) && \separated(a, b, c);
@ ensures *a <= *b <= *c;
@ ensures { *a, *b, *c } == { \old(*a), \old(*b), \old(*c) };
@ ensures \forall int* x, int* y;
        \subset({ x, y }, { a, b, c }) && \separated(x, y) && \old(*x) == \old(*y) ==>
        \exists int* u, int* v;
        \subset({ u, v }, { a, b, c }) && \separated(u, v) && *u == *v == \old(*x);
@ assigns *a, *b, *c;
@*/
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);
}
```

sol/order3-annotated-5.c

# WP Exercise

```
#include <limits.h>

/*@
  @ requires 0 <= x <= INT_MAX / 2;
  @ assigns \nothing;
  @ ensures \result == 2 * x;
  @*/
int times2 (int x) {
  int r = 0 ;
  /*@
    @ loop invariant ...;
    @ loop assigns ...;
    @ loop variant ...;
    @*/
  while (x > 0) {
    r += 2;
    x --;
  }
  return r;
}
```

# Deductive Verification with Interactive Prover

- For proof obligations that cannot be discharged by automatic provers, an interactive prover such as Coq can be used
- We show how to use interactive provers in the following examples



# Prover: Script

The screenshot shows the Frama-C GUI. The top menu bar includes 'Project', 'Analyses', 'File', and 'Help'. Below the menu is a toolbar with various icons. On the left, there is a sidebar with a 'Name' dropdown showing 'ex\_script.c'. Below this are sections for 'WP' (with a 'time' field set to 2 and a 'proc' field set to 4), 'Slicing' (with 'Enable' and 'Libraries' checkboxes), 'Occurrence' (with 'Current var: None' and checkboxes for 'Enable', 'Follow focus', 'Read', and 'Write'), and 'Metrics'. The main editor area contains two lemmas:

```
/*@  
lemma Zmul_le_mono_nonneg_l:  
  ∀ ℤ x, ℤ y, ℤ z; 0 ≤ x ⇒ y ≤ z ⇒ x * y ≤ x * z;  
*/  
/*@  
lemma Zadd_le_mul: ∀ ℤ x, ℤ y; 2 ≤ x ⇒ 2 ≤ y ⇒ x + y ≤ x * y;*/
```

Below the editor is a tabbed interface with 'Information', 'Messages (1)', 'Console', 'Properties', 'Values', 'Red Alarms', and 'WP Goals'. The 'WP Goals' tab is active, showing a table with columns: 'Module', 'Goal', 'Model', 'Qed', 'Script', and 'Alt-Ergo 2.4.2'. The table contains two rows, both of which are highlighted in blue:

Module	Goal	Model	Qed	Script	Alt-Ergo 2.4.2
Axiomatics	Lemma 'Zadd_le_mul'	Typed	-	-	
Axiomatics	Lemma 'Zmul_le_mono_nonneg_l'	Typed	-	-	

A red rectangular box highlights the 'Script' column for both rows in the table.

# Prover: Script

The screenshot displays the Frama-C interface with two overlapping windows. The background window shows the 'Project Analyses File Help' menu and a sidebar with a file list containing 'ex\_script.c'. The foreground window shows the same interface but with the script editor open, displaying the following code:

```
/*@  
lemma Zmul_le_mono_nonneg_l:  
  ∀ ℤ x, ℤ y, ℤ z; 0 ≤ x ⇒ y ≤ z ⇒ x * y ≤ x * z;  
*/  
/*@  
lemma Zadd_le_mul: ∀ ℤ x, ℤ y; 2 ≤ x ⇒ 2 ≤ y ⇒ x + y ≤ x * y;  
*/
```

Below the code, the 'WP Goals' panel is visible. It includes tabs for 'Information', 'Messages (1)', 'Console', 'Properties', 'Values', 'Red Alarms', and 'WP Goals'. The 'WP Goals' tab is active, showing a 'Current goal' for the lemma 'Zmul\_le\_mono\_nonneg\_l'. The goal text is:

```
script '.frama-c/wp/script/lemma_Zmul_le_mono_nonneg_l.json' (not created)  
-----  
No Script  
-----  
Lemma Lemma 'Zmul_le_mono_nonneg_l':  
Assume { Have: 0 <= x_0. Have: y_0 <= z_0. }  
Prove: (x_0 * y_0) <= (x_0 * z_0).  
-----  
Goal id: typed_lemma_Zmul_le_mono_nonneg_l  
Short id: lemma_Zmul_le_mono_nonneg_l  
-----  
Driver Alt-Ergo 2.4.2: Timeout (2s) (cached)
```

To the right of the goal text is a 'Tactics' panel with a list of available tactics:

- Alt-Ergo 2.4.2 (?)
- Strategies
- Cut
- Filter

Red boxes highlight the 'Current goal' text and the 'Tactics' list.

# Script Prover

## -Example-

```
/*@
 @ lemma Zmul_le_mono_nonneg_l :
 @     \forall integer x, y, z;
 @     0 <= x ==> y <= z ==> x * y <= x * z;
 @
 @ lemma Zadd_le_mul :
 @     \forall integer x, y;
 @     2 <= x ==> 2 <= y ==> x + y <= x * y;
 @*/

/*@
 @ requires 2 <= x;
 @ requires 2 <= y;
 @ requires x * (x * y) <= INT_MAX;
 @ ensures x * (x + y) <= x * (x * y);
 @ assigns \nothing;
 @*/
void foo(int x, int y) {
}
```

ex\_script.c

# Script Prover

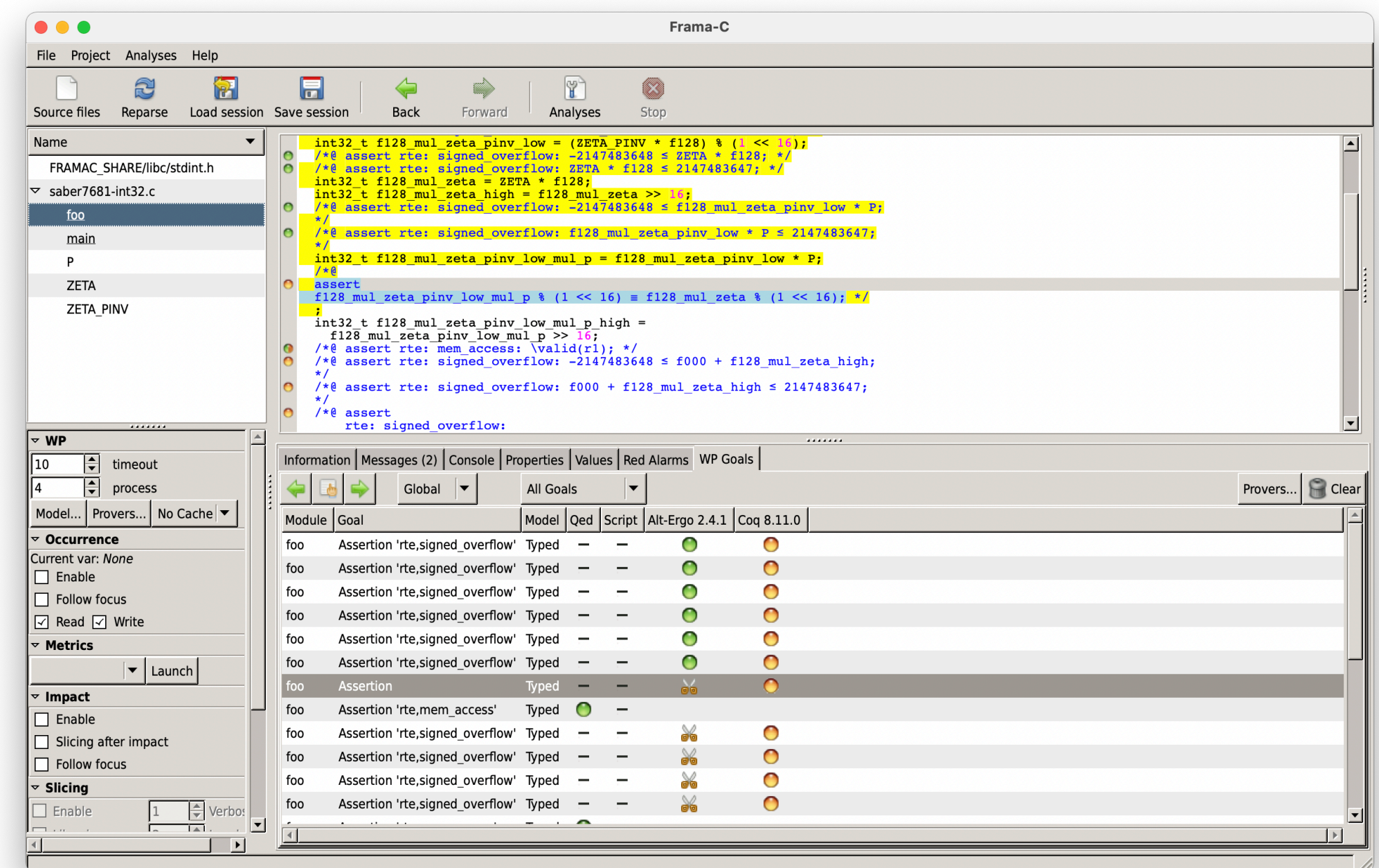
## -Applying Tactics-

- Zmul\_le\_mono\_nonneg\_l:
  - Apply induction on x
- Zadd\_le\_mul
  - Apply induction on x
  - Case analysis on  $\leq$
- Scripts can be saved and replayed with `tip` added to `-wp-prover`

# Prover: Coq

- Run the following command to invoke Frama-C  
`$ frama-c-gui -wp -wp-rte -wp-prover alt-ergo,coq EXAMPLE.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



# 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;
}
```

saber7681-int32.c

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);
  ...
}
```

$\text{\valid}(r1) \ \&\& \ \text{\valid}(r2) \ \rightarrow \ \text{\separated}(r1, r2, \&P, \&ZETA, \&ZETA\_PINV) \ \rightarrow \ (-4096 < f000 < 4096) \ \rightarrow \ (-4096 < f128 < 4096) \ \rightarrow$   
 $((ZETA\_PINV * f128) \% (1 \ll 16)) * P \% (1 \ll 16) == (ZETA * f128) \% (1 \ll 16)$

# 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 Goals using Coq-

- Assertion
  - `.frama-c/wp/interactive/foo_assert.v`
  - Print definitions
  - Search for lemmas
  - Basic tactics

# 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;
}
```

mul\_by\_add.c

# Multiplication by Addition

## -Prove Goals using Coq-

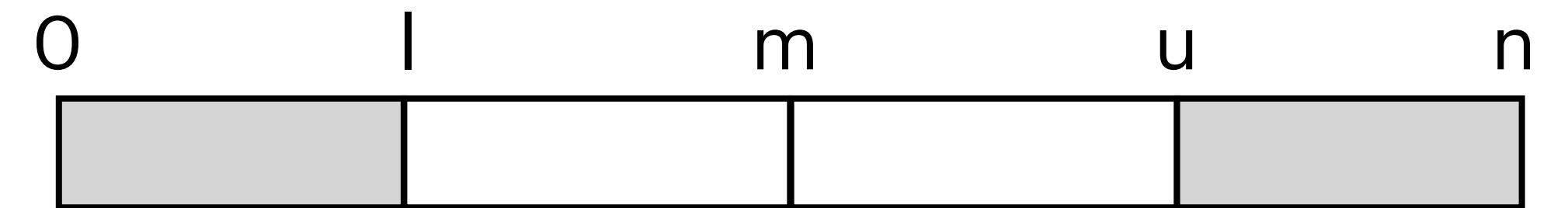
- Invariant (preserved)
  - `.frama-c/wp/interactive/mul_loop_invariant_preserved.v`
  - Basic tactics
  - Proof automation
- Loop variant at loop (decrease)
  - `.frama-c/wp/interactive/mul_loop_variant_decrease.v`
  - Apply lemmas

# Binary Search

## -C Code-

```
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.c



Source: <http://proval.lri.fr/gallery/BinarySearchACSL.en.html> (invalid now)

# 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)
  - Contradiction
  - Focus mode
- Loop variant at loop (decrease)
- Post-condition
- Post-condition for `failure`
- Invariant for `failure` (preserved)

# Nistonacci Numbers

## -Axiomatic Definition-

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

```
/*@  
@ axiomatic Nist {  
@   logic integer nist(integer n);  
@   axiom nist1 : \forall integer n; n < 2 ==> nist(n) == n;  
@   axiom nist2 : \forall integer n; !(n < 2) ==> nist(n) == nist(n - 2) + 2 * nist(n - 1);  
@ }  
@*/
```

nistonacci.c

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



# Nistonacci Numbers

-Goal-

```
/*@  
  @ requires 0 <= n < m;  
  @ ensures n < nist(m);  
  @ assigns \nothing;  
  @*/  
void foo(int n, int m) {  
  ...  
}
```

nistonacci.c

alt-ergo fails to prove the postcondition

# Nistonacci Numbers

## -Lemma-

```
//@ lemma nist_geN : \forall integer n; 0 <= n ==> n <= nist(n);

/*@
  @ requires 0 <= n < m;
  @ ensures n < nist(m);
  @ assigns \nothing;
  @*/
void foo(int n, int m) {
  ...
}
```

nistonacci-lemma.c

alt-ergo proves the postcondition but fails to prove the lemma

# Nistonacci Numbers

## -Implement nist-

```
/*@
 @ 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;
}
```

nistonacci-impl.c

# Nistonacci Numbers

-Prove Goals using Coq-

- Invariant (preserved) ( $y = \text{nist}(n + 1)$ )
  - Order of operands
  - Replace terms
- Post-condition
  - Induction

# Nistonacci Numbers

## -Prove Postcondition by Alt-ergo-

- What can be added to the annotations so that alt-ergo can prove the postcondition ( $n \leq \text{result}$ )?

```
/*@
 @ 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;
}
```

nistonacci-impl.c

# Nistonacci Numbers

## -Lemma Function-

```
/*@ ghost
/@
@ requires 0 <= n;
@ ensures n <= \result;
@ assigns \nothing;
@/
int nist_impl(int n) {
    ...
}
@*/

/*@
@ requires 0 <= n < m;
@ ensures n < nist(m);
@ assigns \nothing;
@*/
void foo(int n, int m) {
    //@ ghost nist_impl(m);
}
```

nistonacci-ghost.c

# Nistonacci Numbers

## -Simpler Lemma Function-

```
/*@ ghost
/@
@ requires 0 <= n;
@ ensures n <= nist(n);
@ assigns \nothing;
@/
void nist_geN(int n) {
    if (n >= 2) {
        nist_geN(n-1);
        nist_geN(n-2);
    }
    return;
}
@*/
```

nistonacci-rec.c

# Inconsistent Annotations

```
/*@ logic integer last0(integer n) = n%10 == 0 ? 1 + last0(n / 10) : 0; */

/*@
  ensures \false;
  assigns \nothing;
*/
int foo(void) {
  /*@ assert last0(0) == 1; */
  return 0;
}
```

Output:

```
...
[wp] Proved goals: 3 / 3
Qed: 1
Alt-Ergo 2.4.2: 2 (5ms)
```

inconsistent1.c



# Summary

- Frama-C is a powerful and flexible tool for deductive program verification
- There are still the following challenges:
  - Write a correct specification
  - Write a strong enough loop invariant
  - Analyze 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.