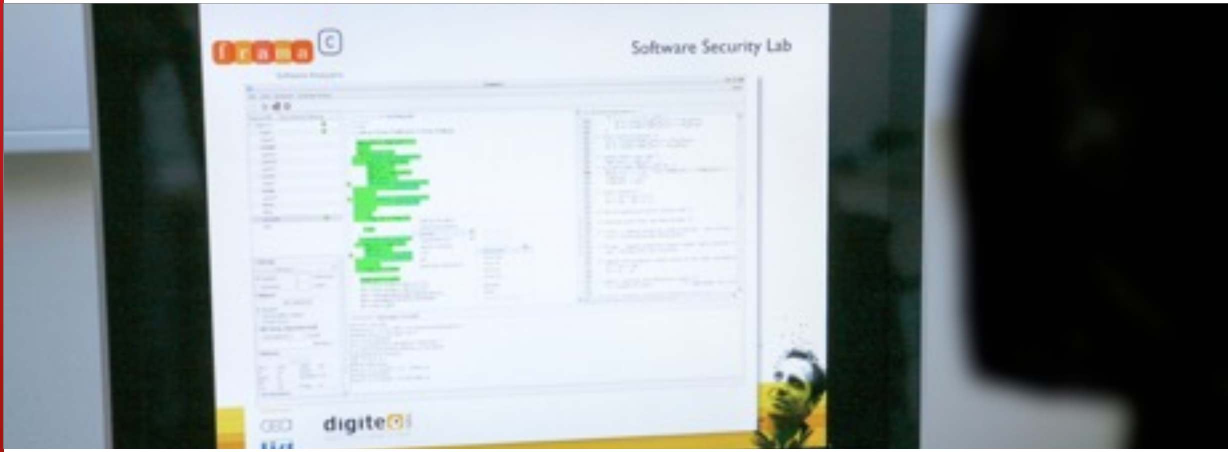
DE LA RECHERCHE À L'INDUSTRIE

Proving a C-Code with GATeL

L. Correnson



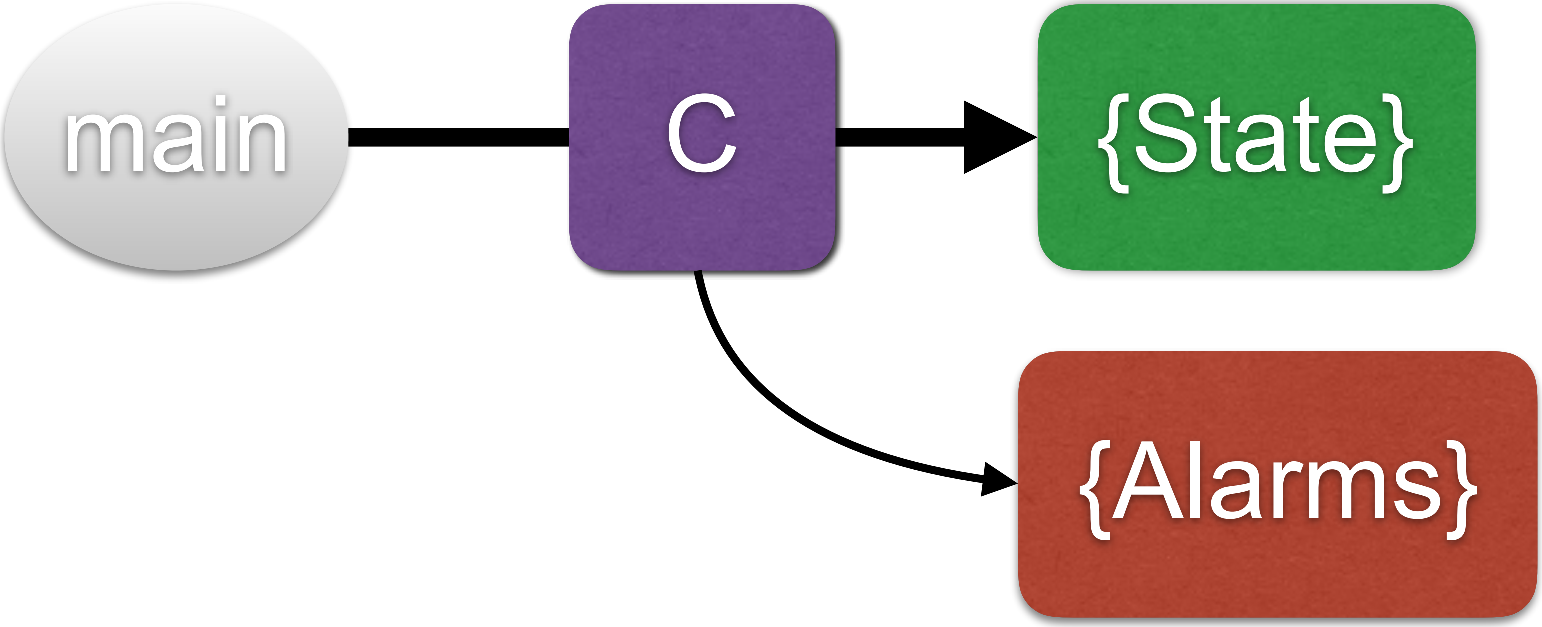
www.cea.fr

FRAMA-C DAY

WHAT'S THIS ABOUT ?

Value

WHAT'S THIS ABOUT ?

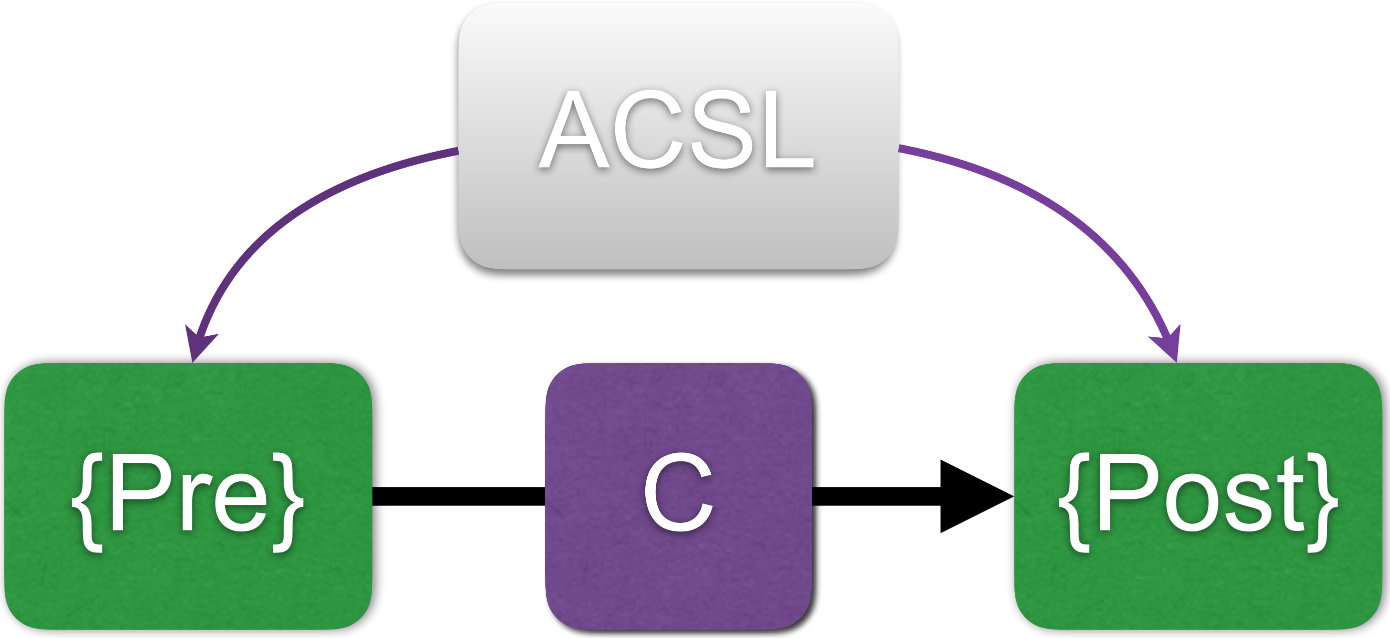


/ Value

WHAT'S THIS ABOUT ?

WP

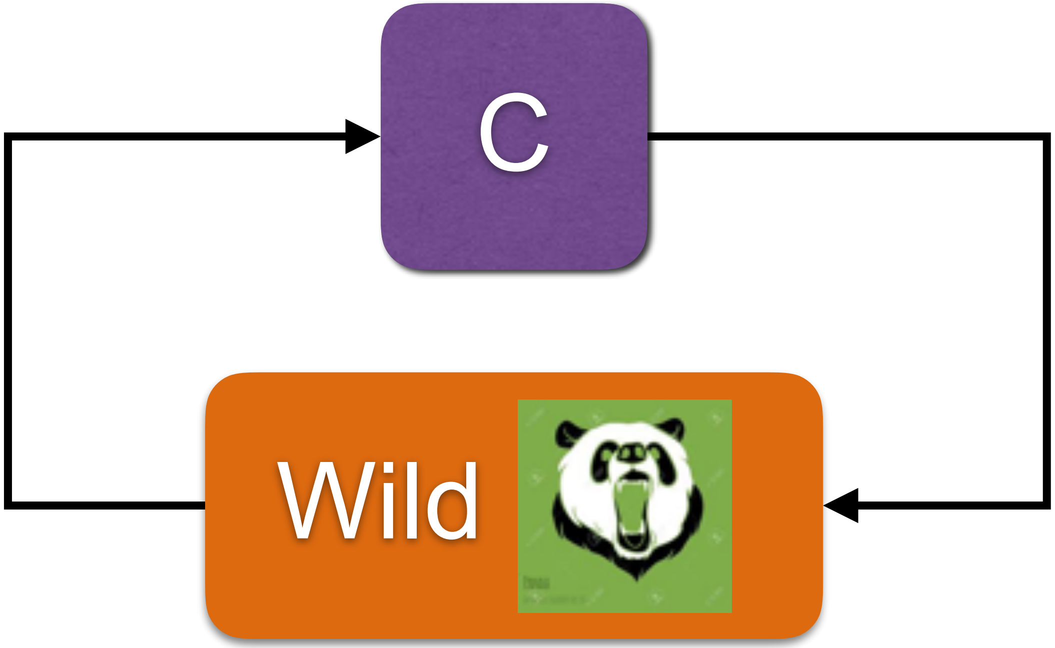
WHAT'S THIS ABOUT ?



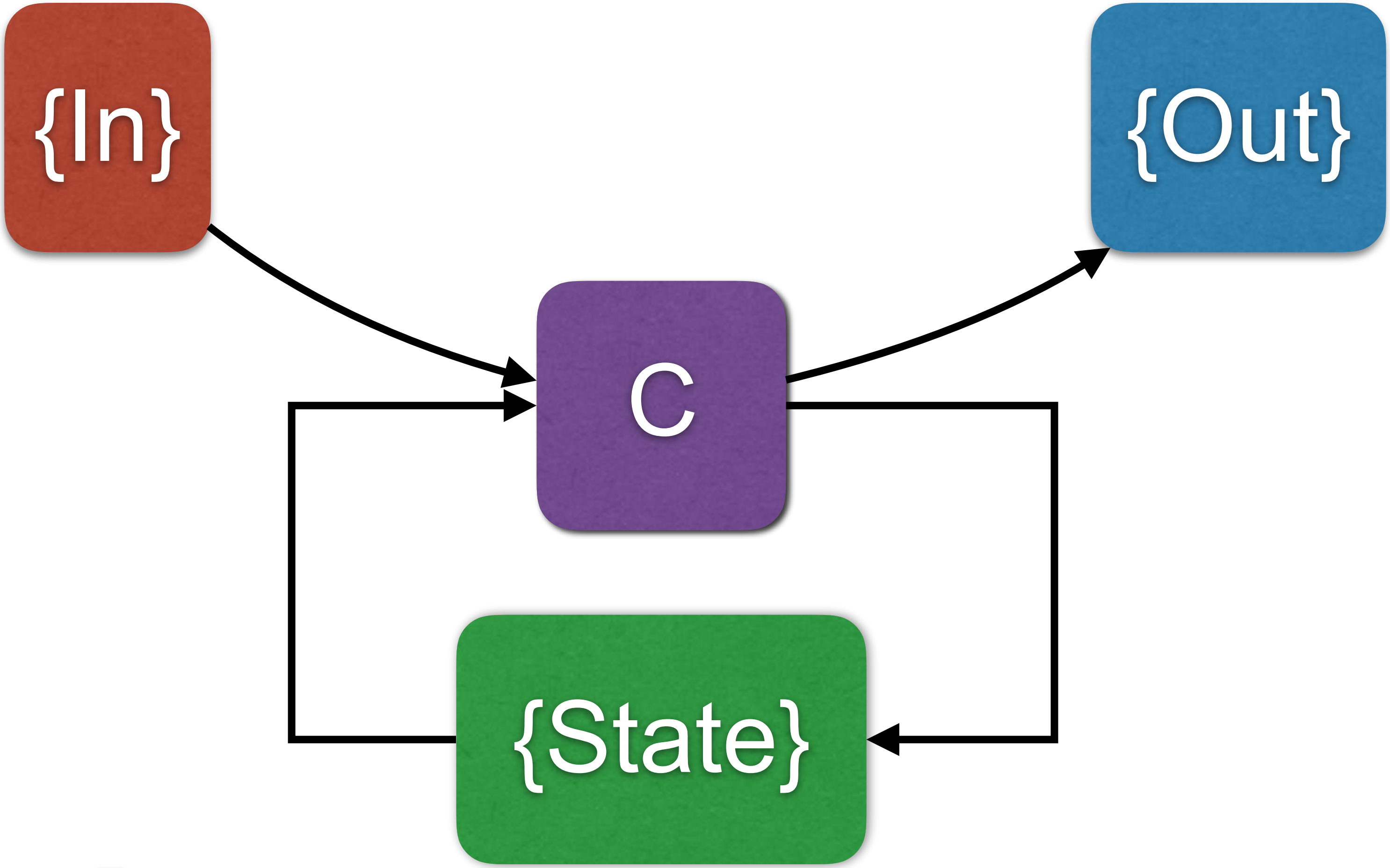
WHAT'S THIS ABOUT ?

Now ?

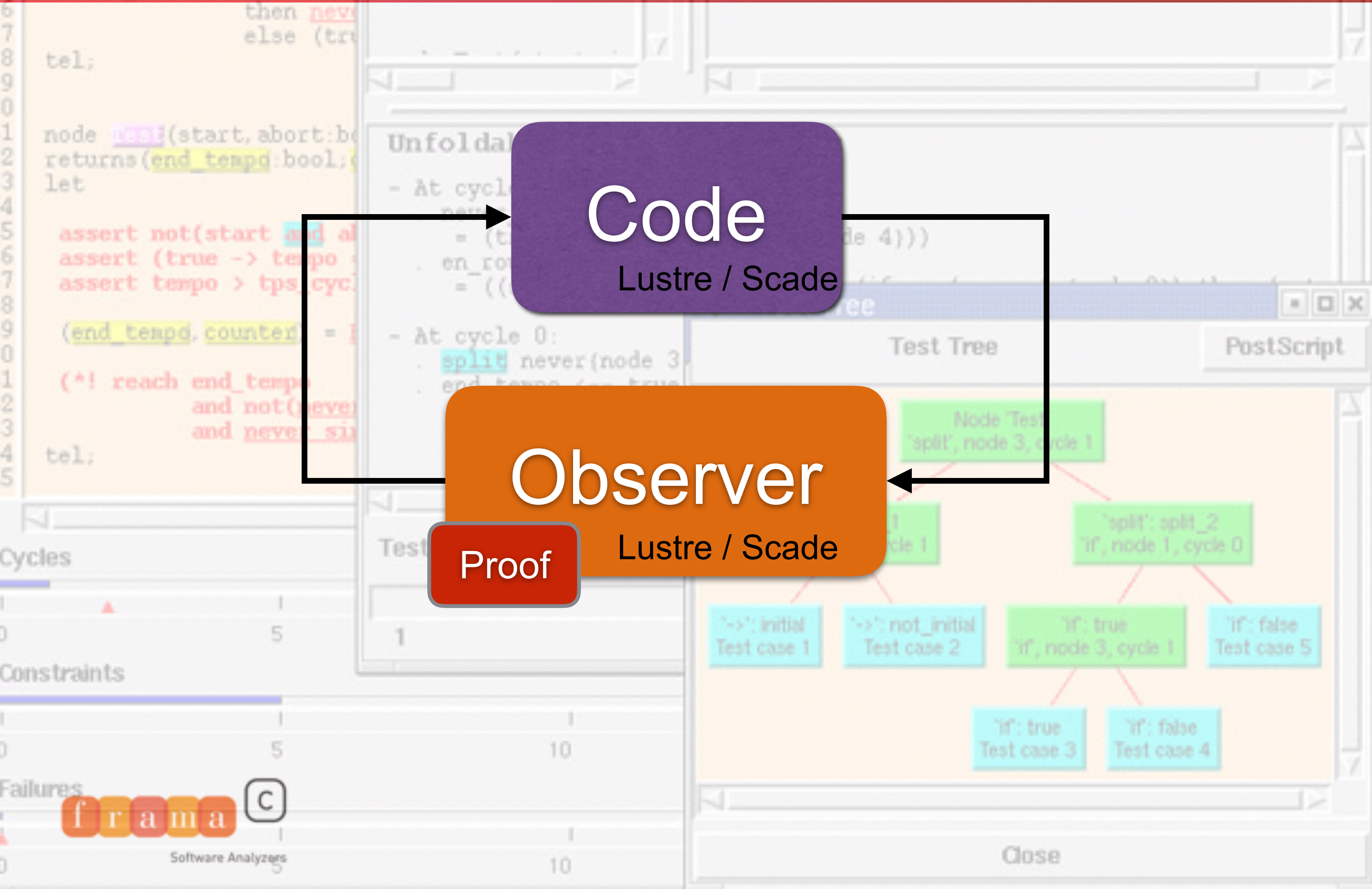
WHAT’S THIS ABOUT ?



CONTROL COMMAND LOOP



GATEL



Code

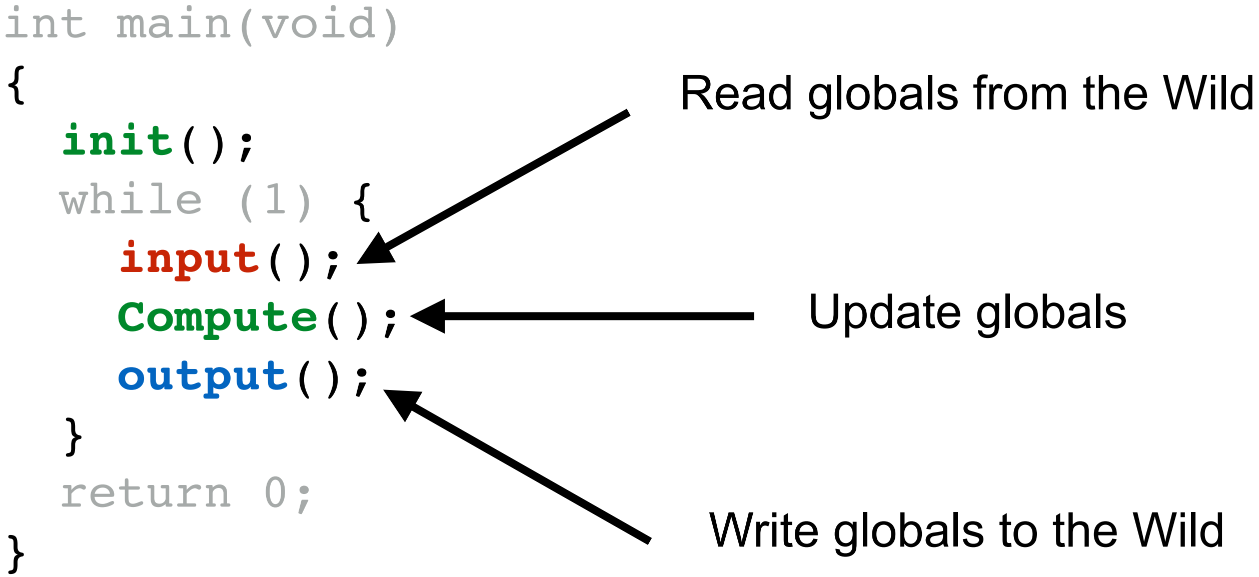
Lustre / Scade

Observer

Lustre / Scade

Proof

CONTROL COMMAND LOOP



```
int main(void)
{
  init();
  while (1) {
    input();
    Compute();
    output();
  }
  return 0;
}
```

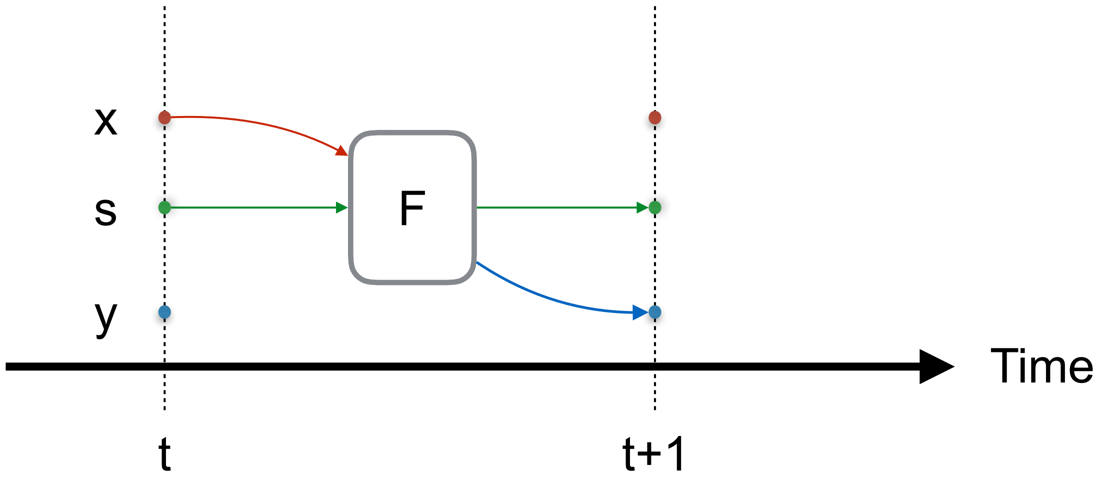
Read globals from the Wild

Update globals

Write globals to the Wild

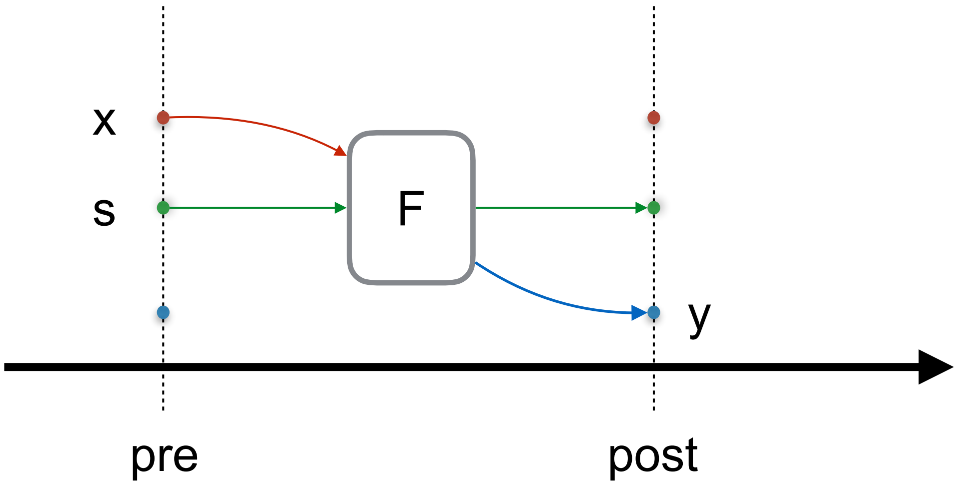
THE « SYNCHRONE » PLUG-IN

```
init();

/*@ input x,… ;
    probe y,… ;
*/
Compute();
```



A KIND OF WP ?

```
init();

/*@ ensures s = F_s(x,\old(s)) ;
    ensures y = F_y(x,\old(s)) ;
*/
Compute();
```

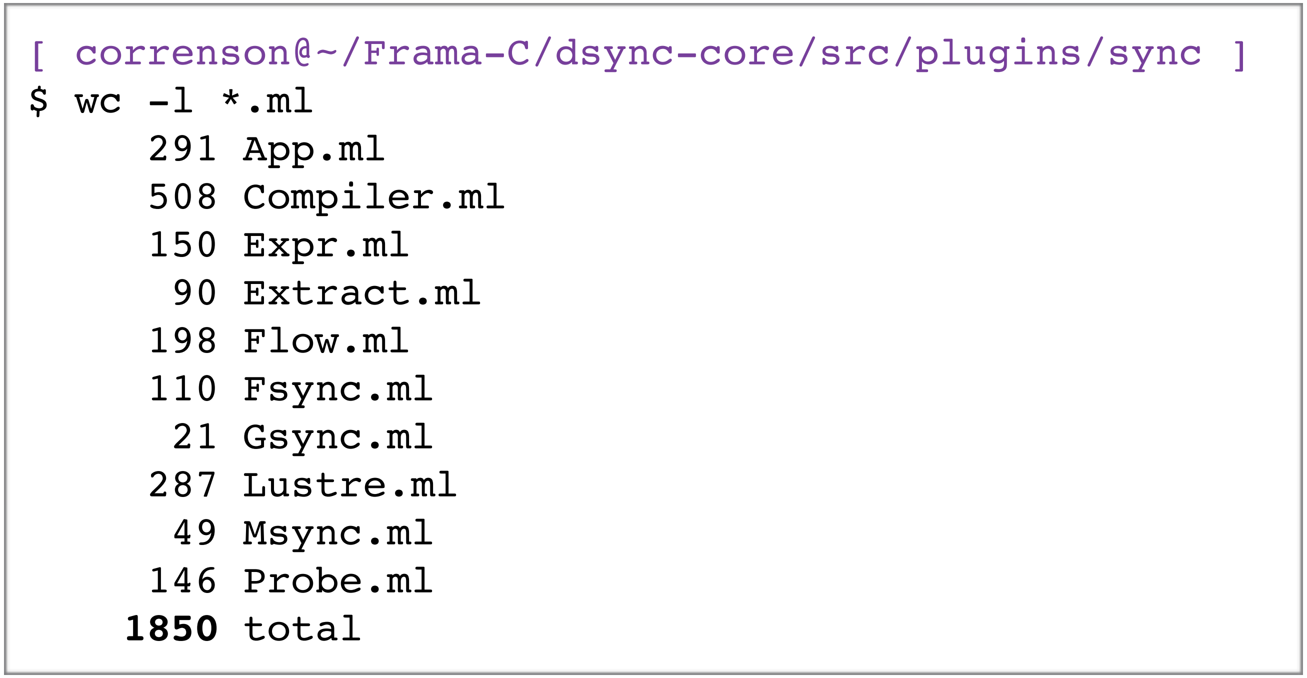


THE CLAIM

Just combine

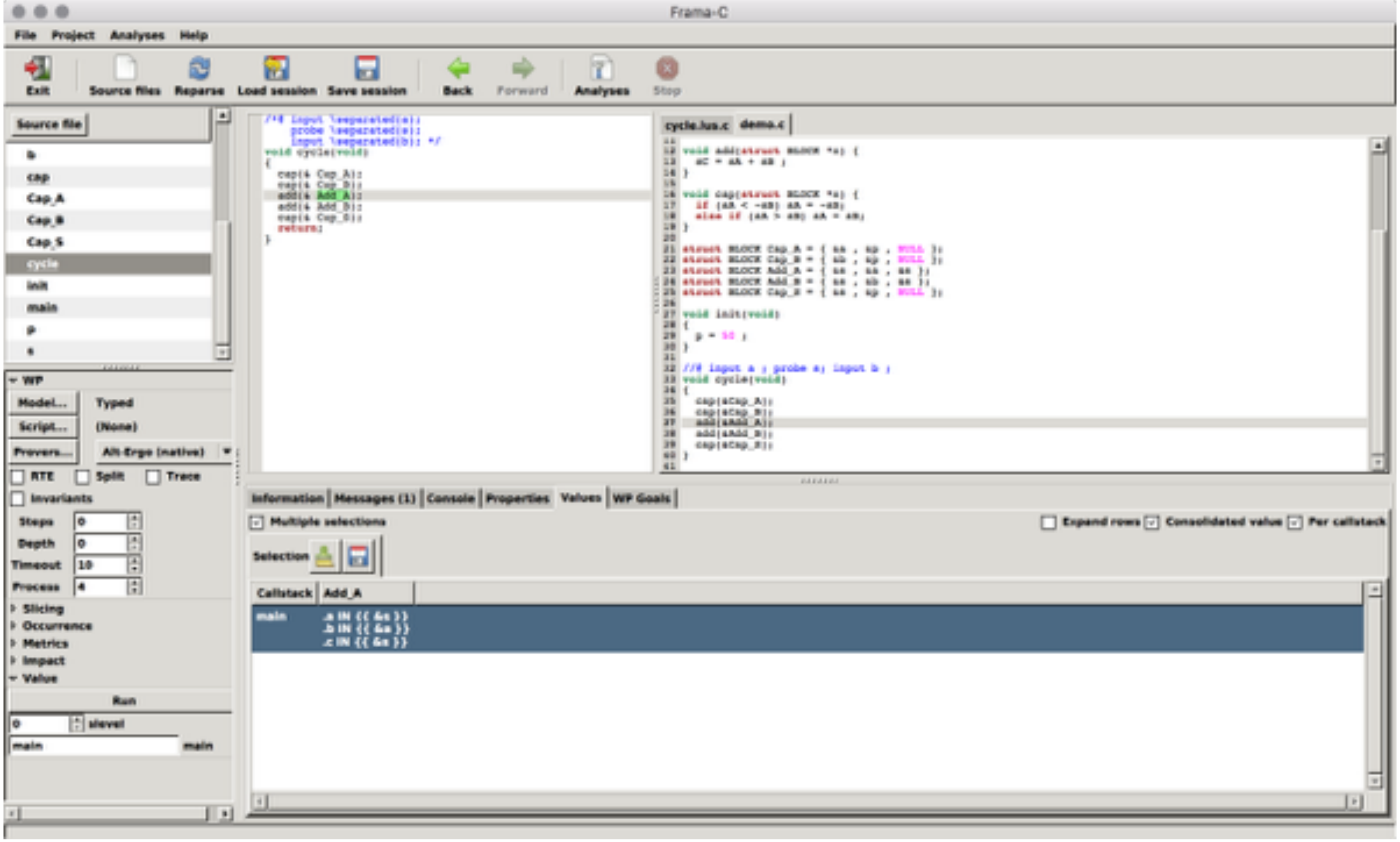
WP & Value APIs

THE PROOF



```
[ correnson@~/Frama-C/dsync-core/src/plugins/sync ]
$ wc -l *.ml
     291 App.ml
     508 Compiler.ml
     150 Expr.ml
      90 Extract.ml
     198 Flow.ml
     110 Fsync.ml
      21 Gsync.ml
     287 Lustre.ml
      49 Msync.ml
     146 Probe.ml
    1850 total
```

DEMO



```
/*@ input \separated(a);
    probe \separated(s);
    input \separated(b); */
void cycle(void)
{
  cap(& Cap_A);
  cap(& Cap_B);
  add(& Add_A);
  add(& Add_B);
  cap(& Cap_S);
  return;
}
```

```
void add(struct BLOCK *s) {
  sC = sA + sB ;
}

void cap(struct BLOCK *s) {
  if (sA < -sB) sA = -sB;
  else if (sA > sB) sA = sB;
}

struct BLOCK Cap_A = { &a , &p , NULL };
struct BLOCK Cap_B = { &b , &p , NULL };
struct BLOCK Add_A = { &a , &a , &s };
struct BLOCK Add_B = { &s , &b , &s };
struct BLOCK Cap_S = { &s , &p , NULL };

void init(void)
{
  p = 50 ;
}

//@ input a ; probe s; input b ;
void cycle(void)
{
  cap(&Cap_A);
  cap(&Cap_B);
  add(&Add_A);
  add(&Add_B);
  cap(&Cap_S);
}
```

Callstack	Add_A

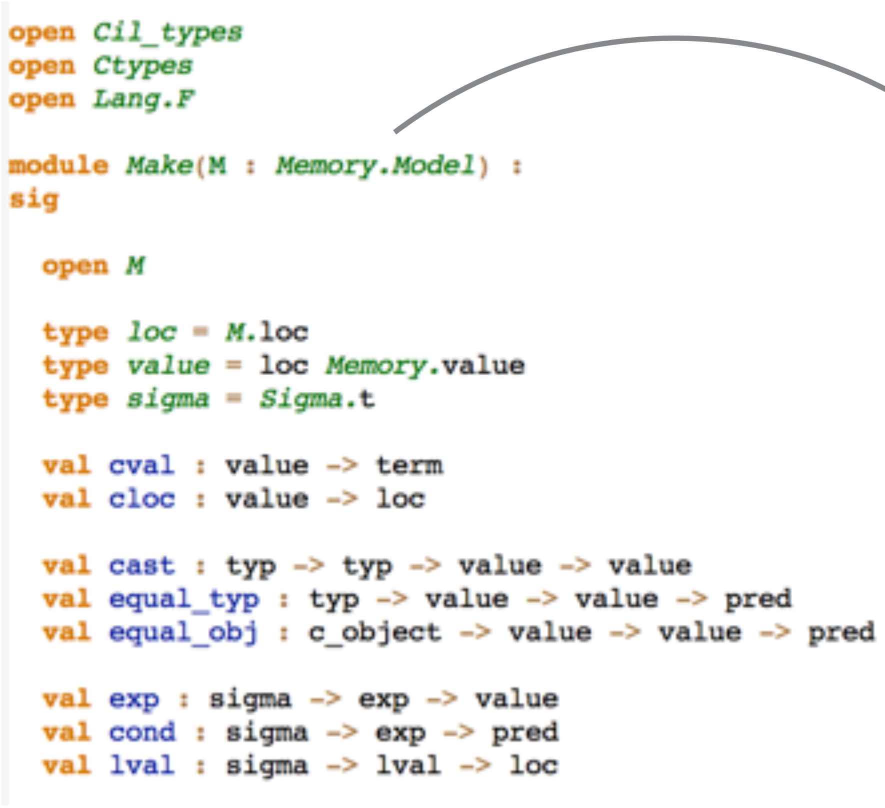
main	.a IN {{ &a }}
	.b IN {{ &a }}
	.c IN {{ &s }}

Software Analyzers

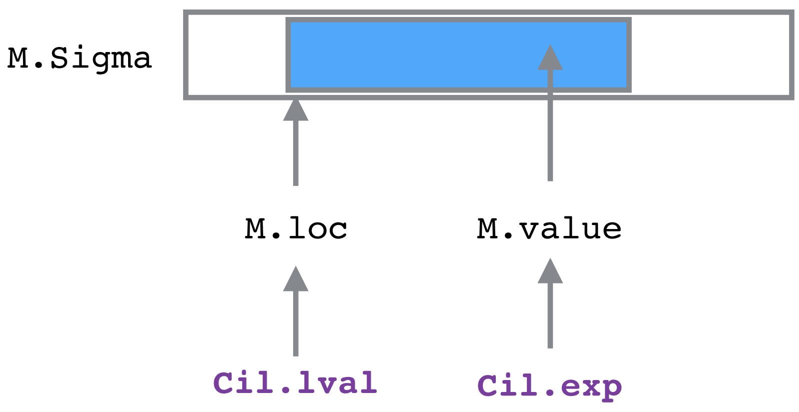
WP API

Qed	First Order Logic + Theories
Lang	Qed Instance with C/ACSL Definitions
Model	Vector of Logic Variables describing the C-Memory
CodeSemantics(Model)	Compiler of C-Code
LogicSemantics(Model)	Compiler of ACSL
Conditions	Formula Simplifier

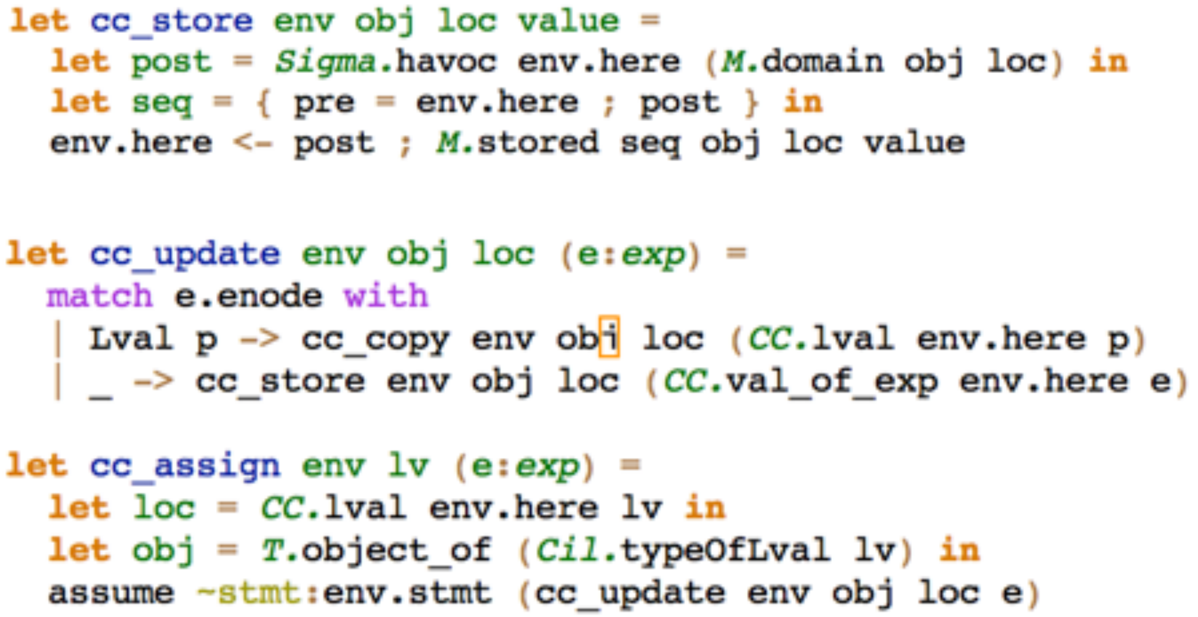
WP API



```
open Cil_types
open Ctypes
open Lang.F

module Make(M : Memory.Model) :
sig

  open M

  type loc = M.loc
  type value = loc Memory.value
  type sigma = Sigma.t

  val cval : value -> term
  val cloc : value -> loc

  val cast : typ -> typ -> value -> value
  val equal_typ : typ -> value -> value -> pred
  val equal_obj : c_object -> value -> value -> pred

  val exp : sigma -> exp -> value
  val cond : sigma -> exp -> pred
  val lval : sigma -> lval -> loc
```

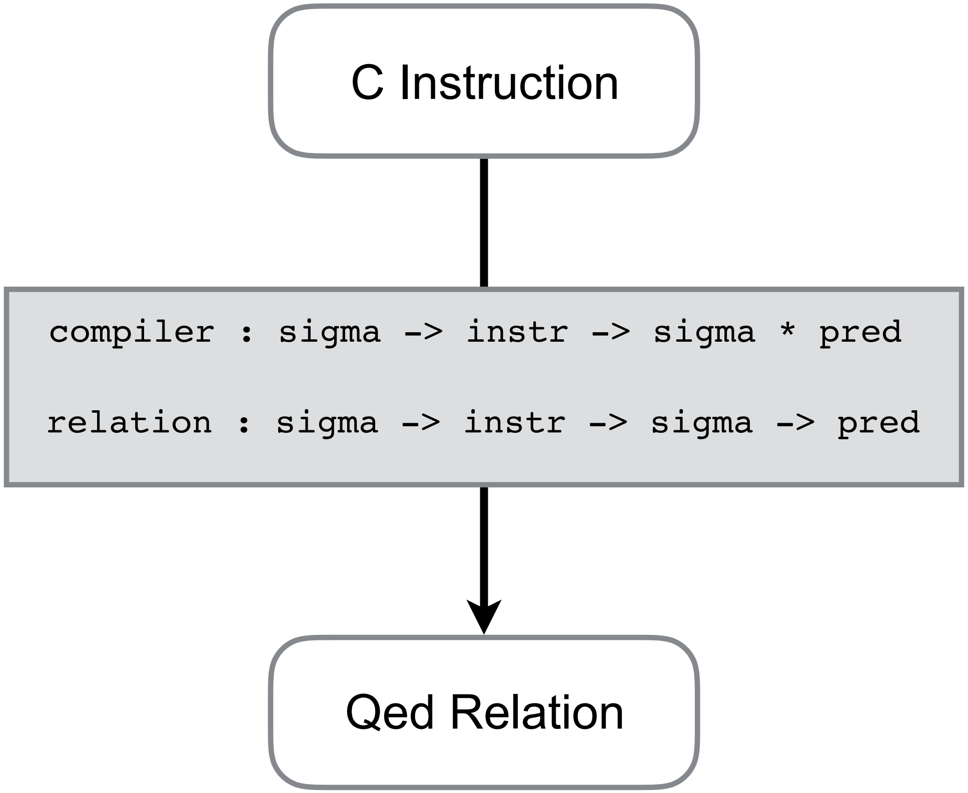


WP API



```
let cc_store env obj loc value =
  let post = Sigma.havoc env.here (M.domain obj loc) in
  let seq = { pre = env.here ; post } in
  env.here <- post ; M.stored seq obj loc value


let cc_update env obj loc (e:exp) =
  match e.enode with
  | Lval p -> cc_copy env obj loc (CC.lval env.here p)
  | _ -> cc_store env obj loc (CC.val_of_exp env.here e)

let cc_assign env lv (e:exp) =
  let loc = CC.lval env.here lv in
  let obj = T.object_of (Cil.typeOfLval lv) in
  assume ~stmt:env.stmt (cc_update env obj loc e)
```

WP API : A SYMBOLIC EVALUATION ENGINE



C Instruction

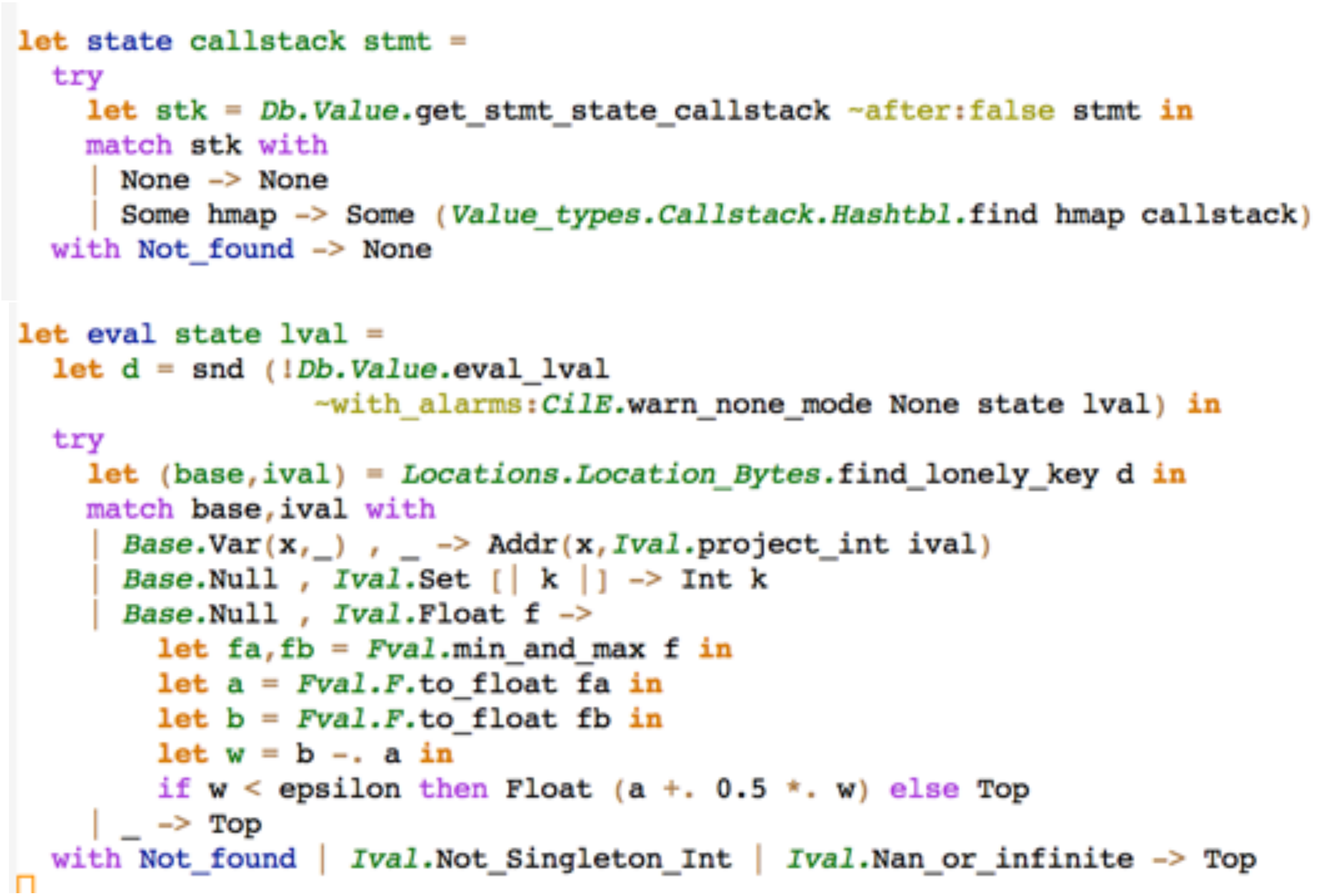
```
compiler : sigma -> instr -> sigma * pred

relation : sigma -> instr -> sigma -> pred
```

Qed Relation

VALUE API

Integer	Domain for Set of Integers
Value	Domain for Values : float, integer, address (Base x Integer), …
OffsetMaps	Domain for Memory (Base x Offset => Integer)
Call Stacks	Context of the Analyzer
Value State	Call Stack => Lvalue => Value

VALUE API



```
let state callstack stmt =
  try
    let stk = Db.Value.get_stmt_state_callstack ~after:false stmt in
    match stk with
    | None -> None
    | Some hmap -> Some (Value_types.Callstack.Hashtbl.find hmap callstack)
  with Not_found -> None

let eval state lval =
  let d = snd (!Db.Value.eval_lval
                 ~with_alarms:CilE.warn_none_mode None state lval) in
  try
    let (base,ival) = Locations.Location_Bytes.find_lonely_key d in
    match base,ival with
    | Base.Var(x,_) , _ -> Addr(x,Ival.project_int ival)
    | Base.Null , Ival.Set [| k |] -> Int k
    | Base.Null , Ival.Float f ->
        let fa,fb = Fval.min_and_max f in
        let a = Fval.F.to_float fa in
        let b = Fval.F.to_float fb in
        let w = b -. a in
        if w < epsilon then Float (a +. 0.5 *. w) else Top
    | _ -> Top
  with Not_found | Ival.Not_Singleton_Int | Ival.Nan_or_infinite -> Top
```

Software Analyzers

SYNCHRONE IMPLEMENTATION

Main Loop	Pack init, inputs and compute in a loop
Run Value	Invariants on l-values, per callstack
Compile	Symbolic Evaluation of each instruction - inline function calls - value’s invariants on each callstack - simplify the equations

Lustre Expression	Another Qed Instance
Probe Extraction	Extract Output from Input and State
Memory Projection	Extract State from State

A SIMPLIFIED EXAMPLE

```
int c=1, g=0, *p=&g;

//@ input x ; probe y ;
void Compute(void) {
    *p += c ;
    y = *p + c ;
    *p += x ;
}
```

Value

```
C = 1 ; P = &G
```

WP

```
M1 = M0 [ P -> M0[P]+C ];
Y = M1[ P ];
M2 = M1 [ P -> M1[P]+X ];
```

Simplification

```
Y = M0[&G]+1;
M2 = M0[&G -> X+Y];
```

Projection

```
S0 => M0[&G]
S2 => M2[&G]
```

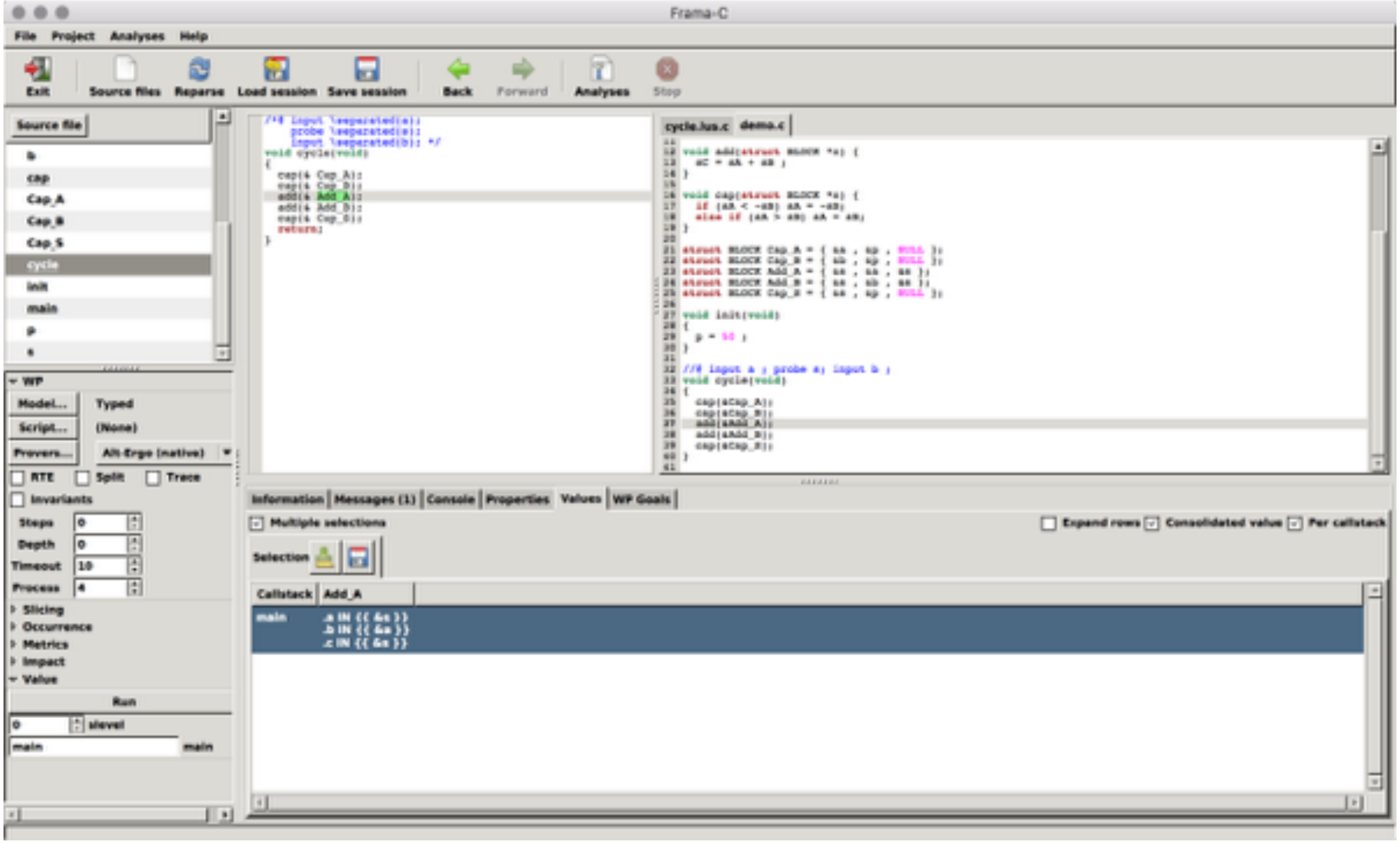
Extraction

```
Y = S0 + 1;
S2 = S0 + X + 1 ;
```

Lustre

```
Y = S + 1;
S = 0 -> pre (S + X + 1) ;
```

DEMO



Typed
(None)
Alt-Ergo (native)

□ RTE □ Split □ Trace

□ Invariants

Steps 0

Depth 0

Timeout 10

Process 4

0 slevel

main main

```
/*@ input \separated(a);
    probe \separated(a);
    input \separated(b); */
void cycle(void)
{
  cap(& Cap_A);
  cap(& Cap_B);
  add(& Add_A);
  add(& Add_B);
  cap(& Cap_S);
  return;
}
```

```
void add(struct BLOCK *s) {
  sC = sA + sB ;
}

void cap(struct BLOCK *s) {
  if (sA < -sB) sA = -sB;
  else if (sA > sB) sA = sB;
}

struct BLOCK Cap_A = { &a , &p , NULL };
struct BLOCK Cap_B = { &b , &p , NULL };
struct BLOCK Add_A = { &a , &a , &s };
struct BLOCK Add_B = { &s , &b , &s };
struct BLOCK Cap_S = { &s , &p , NULL };

void init(void)
{
  p = 50 ;
}

//@ input a ; probe s; input b ;
void cycle(void)
{
  cap(&Cap_A);
  cap(&Cap_B);
  add(&Add_A);
  add(&Add_B);
  cap(&Cap_S);
}
```

☐ Multiple selections

□ Expand rows ☑ Consolidated value ☑ Per callstack

Callstack	Add_A

main	.a IN {{ &s }}
	.b IN {{ &a }}
	.c IN {{ &s }}

CONCLUSION

(just) combine

Frama-C APIs & other Tools

WP

Value

GATeL

FUTURE WORK

Qed	Soon on GitHub & Opam
WP	API for Symbolic Evaluation (helpers)
Value	Everything works (yes, I know, …)
Doc	Oh, yes, of course …
Synchrone	Maturation of the prototype (loops, arrays, bitwise operators, modular extraction, …)