

Developing Frama-C Plug-ins in OCaml

Julien Signoles
Software Safety Lab.



Ocaml Users Meeting
2011/04/15

(long na
[for 0 =>
C1); if (m
tmp2 =
se of the

tmp2[0] = 1 << (n0 - 1); else if (tmp1[0] >= 1) << (n0 - 1); tmp2[0] = (1 << (n0 - 1)) + tmp1[0];
tmp1[0] = 0; k = 0; k <= 5; k <=> tmp1[0] += mc2[0][k] * tmp2[k];
Final rounding: tmp2[0] is now represented on 9 bits: if (tmp1[0] < -256) m2[0] = -256; else if (tmp1[0] > 255) m2[0] = 255; else m2[0] = tmp1[0];



<http://frama-c.com>

- ▶ platform dedicated to **source code analysis** of **C programs**
- ▶ ANSI/ISO C 99 + a formal specification language **ACSL**
- ▶ developed by CEA LIST and Inria Proval since 2005
- ▶ **open source** and released under LGPL v2.1
- ▶ extensible platform through **plug-ins**
- ▶ **1 plug-in = 1 analyser**
- ▶ **collaboration** between analysers
- ▶ **several static analysers**
- ▶ both **academic** and **industrial** purposes

Several tools inside one platform



- ▶ fully written in **OCaml** (> 100 kloc)
- ▶ minimal number of **dependencies** (long-term support+easier installation) :
 - ▶ Fork of **CIL** (Necula and al, Berkeley)
 - ▶ **OCamlGraph**
 - ▶ **LablGtk2**
- ▶ big **API** for developing analysers
- ▶ **support** : API documentation, Plug-in Development Guide, mailing-list, BTS, wiki, ...

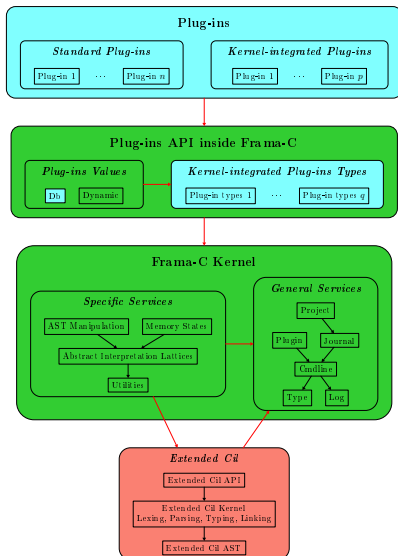
```

long n
for 0 <-> n do
  C1 := 0
  tmp2 := 0
  for 0 <-> n do
    tmp2[0] := 0
    for 0 <-> n do
      tmp2[k] := 0
    end
  end
end
  
```

```

tmp2[0] := 0
for 0 <-> n do
  tmp2[k] := 0
  for 0 <-> n do
    tmp2[k] := 0
  end
end
  
```





running example :

Loop counter

loop_counter.ml:

```
let main () =
  Format.printf "Counting number of loops...@."
```

(execute your plug-in among the others *)*

```
let () = Db.Main.extend main
```

\$ frama-c -load-script loop_counter



(register your plug-in to access basic services *)*

```
include Plugin.Register
```

```
(struct
```

```
  let name = "loop counter"
```

```
  let shortname = "loop"
```

```
  let help =
```

```
    "Count the number of loops in the program"
```

```
end)
```

- ▶ default **options** (-loop-help, -loop-debug, -loop-verbose)
- ▶ way to add **command line options**
- ▶ way to display **messages**



Format-like functions for user messages :

- ▶ feedback, result, warning, abort, error, fatal, ...
- ▶ consistent message taxonomy anywhere in Frama-C

```
let nb_loops = ref 0
let main () =
  feedback ~level:2 "Counting number of loops...";
  result "Program contains %d loops" !nb_loops
```

long na
for 0 <=
ct; if m
tmp2 =
of the

tmp2[0] = 1 << (n-1) else if tmp1[j] >= 1 << (n-1) - 1 then tmp2[j] = 1 else tmp2[j] = tmp1[j] / 2 Then the second part takes for the first one
tmp1[0] = 0; k = 5; k = k + 1; tmp1[0] = mc2[0][k] * tmp2[k][0] / 2 The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1
i = 1; tmp1[0] >= 1; Final operand tmp2[0] is now represented on 9 bits: if (tmp1[0] < 256) m2[0] = 256 else if (tmp1[0] > 255) m2[0] = 255 else m2[0] = 255



Functors for registering parameters

```

(* boolean option '-loop' initialized to [false] *)
module Enable =
  False(struct
    let option_name = "-loop"
    let help = "enable count of loops in the program"
    let kind = 'Correctness
  end)

let main () =
  (* compute iff the option is set *)
  if Enable.get () then begin
    feedback ~level:2 "Counting number of loops...";
    result "Program contains %d loops" !nb_loops
  end
  
```



Plug-in Makefile and configure

- ▶ may include a **generic Makefile**
- ▶ only **set few variables**
- ▶ may also write easily a **configure** from a generic one

Frama-C installation directories

```
FRAMAC_SHARE :=$(shell frama-c.byte -print-path)
FRAMAC_LIBDIR :=$(shell frama-c.byte -print-libpath)
PLUGIN_NAME  := Loop_counter
PLUGIN_CMO   := loop_counter
include $(FRAMAC_SHARE)/Makefile.dynamic
```

```
$ make && sudo make install
```

```
$ frama-c -loop
```



Counting loops while visiting

```

let nb_loops () =
  nb_loops := 0;
  let count_loop = object
    (* visit the AST in place by inheritance *)
    inherit Visitor.frama_c_inplace
    (* only implement what is required *)
    method vstmt s = match s.Cil_types.skind with
      | Cil_types.Loop _ -> incr nb_loops; Cil.DoChildren
      | _ -> Cil.DoChildren
  end
  in
    (* visit the AST with our custom visitor *)
    Visitor.visitFramacFile count_loop (Ast.get());
    !nb_loops
  
```



Provide access to your analyser to other plug-ins

- ▶ safe dynamic typing facilities provided by the Frama-C kernel
- ▶ the only way to define mutually-recursive plug-ins
- ▶ one interface, several implementations

```
let nb_loops =
```

```
  (* register function 'Loop_counter.nb_loops' *)
```

```
  (* of type 'unit -> int' in the API *)
```

```
  Dynamic.register
```

```
    ~journalize:false
```

```
    ~plugin:"loop_counter"
```

```
    "nb_loops"
```

```
    (Datatype.func Datatype.unit Datatype.int)
```

```
  nb_loops
```



- ▶ journal = OCaml script replaying user actions
- ▶ automatically generate on need/on user request
- ▶ help debugging
- ▶ kind of macro language
- ▶ quick plug-in prototyping

```
let nb_loops =
  Dynamic.register
    ~journalize:true
    ~plugin:"loop_counter"
    "main"
    (Datatype.func Datatype.unit Datatype.int)
  nb_loops
```



- ▶ project = one AST + associated global states
- ▶ as many projects as you want
- ▶ default project : no need to add extra parameters to analysers
- ▶ only require to register global states

(a reference to an int option by project *)*

```

module Nb_loops =
  State_builder.Option_ref
    (Datatype.Int)
  (struct
    let name = "Loop_counter.Nb_loops"
    let dependencies = [ Ast.self ]
    let kind = 'Correctness
  end)
  
```



possible to extend the Frama-C GUI in many ways

```

(* dedicated panel for our plug-in *)
let loop_counter_panel =
  let box = GPack.hbox () in
  ... (* adding widgets to the box *)
  let refresh () = ... in
  "Loop counter", box#coerce, Some refresh

let main main_ui =
  (* attach our new panel to the GUI *)
  main_ui#register_panel loop_counter_panel

(* run our main when creating the GUI *)
let () = Design.register_extension main
  
```



<http://frama-c.com>

- ▶ platform dedicated to source code analysis of C programs
- ▶ high-level powerful services
- ▶ handle all C constructs
- ▶ existing community of plug-in developers
- ▶ both academic and industrial purposes
- ▶ documentations and supports

Join the growing community

