Flata-C

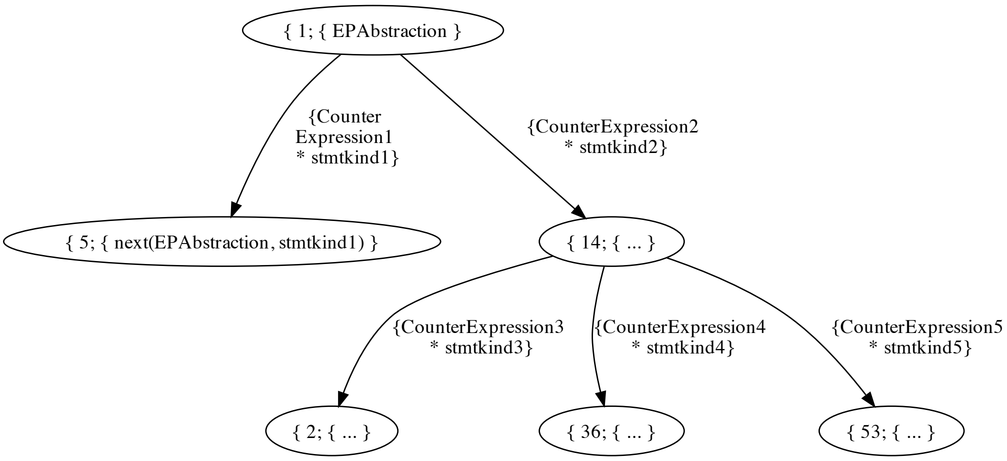
Florent GARNIER & *Maxime* GAUDIN

juin 2011

1 eCFG

The Extended Call Function Graph (eCFG) is the internal way for *Flata-C* to represent the *Frama-C* Call Function Graph (CFG) plus our own datas such as :

- Statements and their abstraction,
- each transition counter automata guard,
- value analysis (still todo).


{ 1; { EPAbstraction }
{Counter Expression1 * stmtkind1}
{CounterExpression2 * stmtkind2}
{ 5; { next(EPAbstraction, stmtkind1) }
{ 14; { ... }
{CounterExpression3 * stmtkind3}
{CounterExpression4 * stmtkind4}
{CounterExpression5 * stmtkind5}
{ 2; { ... }
{ 36; { ... }
{ 53; { ... }

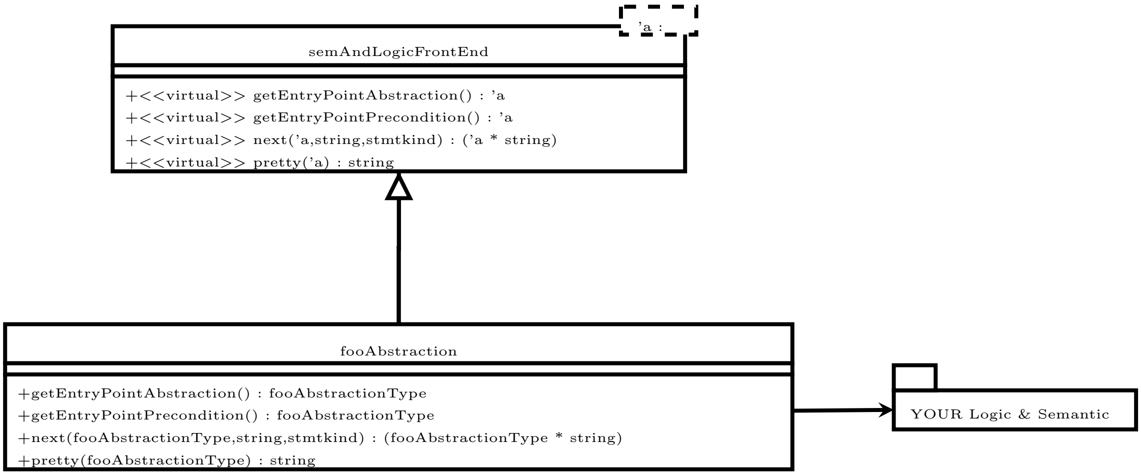
In principle, you **do not have to access directly to the eCFG**. Indeed, the eCFG module encapsulate every algorithms and data structures to hide complexity. That's why the (only) thing to do is to define a new logic & semantic by coding a *front-end* as you'll see in next section.

2 How to . . . ?

2.1 How to : Plug your own logic and semantic to Flata-C ?

The architecture of *Flata-C* was made to be as generic as possible. One of our goals was that, whatever the complexity of your *logic & semantic*, plugging it into *Flata-C* has to be easy and efficient.

To comply with these aims, we designed a fairly simple architecture :


'a :
semAndLogicFrontEnd
+<<virtual>> getEntryPointAbstraction() : 'a
+<<virtual>> getEntryPointPrecondition() : 'a
+<<virtual>> next('a,string,stmtkind) : ('a * string)
+<<virtual>> pretty('a) : string
fooAbstraction
+getEntryPointAbstraction() : fooAbstractionType
+getEntryPointPrecondition() : fooAbstractionType
+next(fooAbstractionType,string,stmtkind) : (fooAbstractionType * string)
+pretty(fooAbstractionType) : string
YOUR Logic & Semantic

The *front-end* acts as an interface between the eCFG algorithms and data structures, this way, even if the methods of the *front-end* are constrained, you can choose the way to implement your own *logic & semantic.*

Sample : The TrueLogic front end

```

open Cil_types
open SemAndLogicFrontEnd

class trueLogicFrontEnd =
object
inherit [bool] semAndLogicFrontEnd
  method getEntryPointAbstraction () = true
  method getEntryPointPrecondition () = ""
  method isErrorState state = (state = false)
  method next currentAbstraction _ _ = (currentAbstraction, "NONE" )
  method pretty abs = if abs then "TRUE" else "FALSE"
end
```

This logic return the opposite of the previous node.

For instance, a simple “while loop” code produce, with this logic & semantic, the following eCFG :

Glossaire

LOGIC & SEMANTIC

Describe the abstraction of the code and all the operations on this abstraction. [Page(s) 1, 2]

Table des matières