

# The Flatac Framac front-end

Radu Iosif, Florent Garnier

May 10, 2012

# Outline

# What is flatac ?

- Part of a toolchain that aims at proving that C programs don't generate memory faults and don't violates assertions.
- A front end that generates NTS based models of C programs.
- Coded as a Frama-C plugin.

# Typical memory faults :

- Memory access outside and allocated memory zone of the heap
- Access to an array outside of its bounds
- Memory access using a non aligned address
- Double free
- Freeing an allocated segment using an pointer that does not points at the begining of the segment.
- Memory leaks

# Two subkinds of properties :

- Properties concerning the memory shape (Simple Separation Logic):
  - Relation between pointer variables (Stack) and location variables (heaps).
  - Memory allocation.
  - Allocated Segment separation.
- Arithmetic properties :
  - Memory segment access within its bounds.
  - Memory address alignment (Congruence).

# Tracked property

This front end aims at proving that C programs :

- Have no execution run that lead to memory fault.
- Have no execution that violates some assertion expressed using arithmetic constraints.

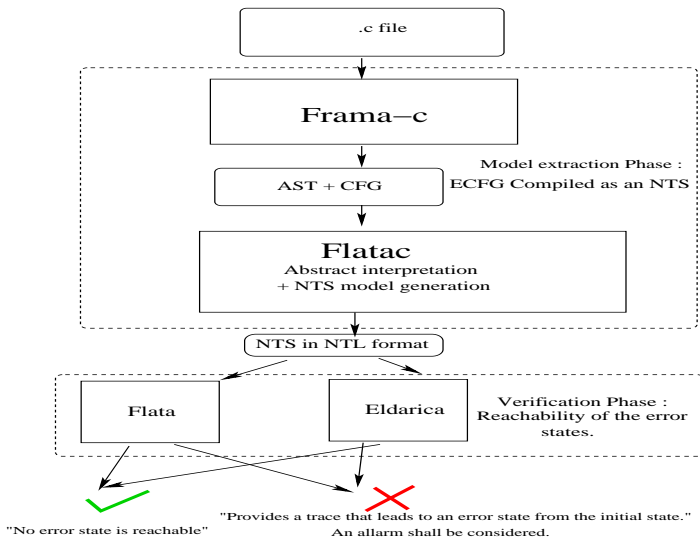
Flatac plugin: Front end of NTS error state reachability analysis

- Extracts models of C Programs using Abstract Interpretation Techniques.
- Adds **Numerical Transitions Systems** informations on the model for **a posteri Verification Phase**.

# How to do that ?

- 1 Extracting an extended cfg from Framac-c cfg (Cil statements  $\times$  SSL memory abstractions)<sup>2</sup>
- 2 Labelling the Ecfg transitions with Numerical Transition System expression –Guards, counter affectation and Function Calls.
- 3 If a SSL Abs value of a state is  $\perp$ , define this state as an error state.
- 4 Export the labelled Ecfg into Nts Format.
- 5 Ask an analysis tool –Flata, Eldarica, to check whether some error state is reachable from the entry point (main function).

# Flatac in the tool-chain:





# Abstract interpretation preliminary part

Simple Separation Logic formulae : Abstract domain.

$\phi$	$:= \pi \updownarrow \sigma \mid \exists l. \phi$	Formulae
$\pi$	$:= x \mapsto l \mid x \mapsto \text{nil} \mid (l_1 = l_2) \mid \pi_1 \wedge \pi_2$	Pure part
$\sigma$	$:= \text{Emp} \mid \text{alloc}(l) \mid \sigma_1 * \sigma_2 \perp$	Spatial part

# Properties of SSL

The problem that follows are decidable :

- Satisfiability (Valid configuration)
- Entailment, Equivalence.
- Memory leaks

Those problems are solved using rewriting techniques.

## Example of SSL formulae

- $x \mapsto l_1 \wedge y \mapsto l \wedge z \mapsto \text{nil} \updownarrow \text{Emp}$
- $x \mapsto l_1 \wedge y \mapsto l \wedge z \mapsto \text{nil} \updownarrow \text{alloc}(l_1)$
- $x \mapsto l_1 \wedge y \mapsto l \wedge z \mapsto \text{nil} \updownarrow \text{alloc}(l_1) * \text{alloc}(l)$
- $x \mapsto l_1 \wedge y \mapsto l \wedge z \mapsto \text{nil} \updownarrow \text{alloc}(l_1) * \text{alloc}(l)$
- $x = y \wedge x \mapsto l_1 \wedge y \mapsto l \updownarrow \text{alloc}(l_1) * \text{alloc}(l) \text{ (Unsat)}$
- $x = y \wedge x \mapsto l_1 \wedge y \mapsto \text{nil} \updownarrow \text{alloc}(l_1) \text{ (Unsat)}$
- $\text{true} \updownarrow \text{alloc}(l) \text{ (Leak)}$

# Flata-c Memory model

A memory model that associates counters to SSL variables :

SSL Variable	NTS counter	
$x \in PVar$	<code>x_offset</code>	offset
$l \in LVar$	<code>l_size</code>	segment size

In order to :

- Associate to segment their size.
- Associate to pointer their offset.
- To express guards on memory access.

# Example

C "statement"	SSL formula	NTS transition
<code>int *x;</code>	$\exists l x \mapsto l \Downarrow \text{Emp}$	<code>offset_x'=0</code>
<code>x=malloc(10);</code>	$\exists l x \mapsto l \Downarrow \text{alloc}(l)$	<code>size_l=10</code>
<code>x++;</code>	$\exists l x \mapsto l \Downarrow \text{alloc}(l)$	<code>offset_x'+=sizeof(int)</code>
<code>int y =*x;</code>	$\exists l x \mapsto l \Downarrow \text{alloc}(l)$	<code>offset_x' &lt; size_l</code>
	$\exists .l x \mapsto l \Downarrow \text{alloc}(l)$	<code>havoc(y)</code>

# Example

C "statement"	SSL formula	NTS transition
<code>int *x;</code>	$\exists l x \mapsto l \Downarrow \text{Emp}$	<code>offset_x'=0</code>
<code>x=malloc(10);</code>	$\exists l x \mapsto l \Downarrow \text{alloc}(l)$	<code>size_l=10</code>
<code>x++;</code>	$\exists l x \mapsto l \Downarrow \text{alloc}(l)$	<code>offset_x'+=sizeof(int)</code>
<code>int y =*x;</code>	$\exists l x \mapsto l \Downarrow \text{alloc}(l)$	<code>offset_x' &lt; size_l</code>
	$\exists .l x \mapsto l \Downarrow \text{alloc}(l)$	<code>havoc(y)</code>
<code>x+=10;</code>	$\exists l x \mapsto l \Downarrow \text{alloc}(l)$	<code>offset_x'+=10*sizeof(int)</code>

# Example

C "statement"	SSL formula	NTS transition
<code>int *x;</code>	$\exists l.x \mapsto l \Downarrow \text{Emp}$	<code>offset_x'=0</code>
<code>x=malloc(10);</code>	$\exists l.x \mapsto l \Downarrow \text{alloc}(l)$	<code>size_l=10</code>
<code>x++;</code>	$\exists l.x \mapsto l \Downarrow \text{alloc}(l)$	<code>offset_x'+=sizeof(int)</code>
<code>int y =*x;</code>	$\exists l.x \mapsto l \Downarrow \text{alloc}(l)$	<code>offset_x' &lt; size_l</code>
	$\exists .l.x \mapsto l \Downarrow \text{alloc}(l)$	<code>havoc(y)</code>
<code>x+=10;</code>	$\exists l.x \mapsto l \Downarrow \text{alloc}(l)$	<code>offset_x'+=10*sizeof(int)</code>
<code>*x=42;</code>	$\perp$	<code>offset_x <math>\geq</math> size_l</code>

Access to `*x` is out of bounds of allocated segment at 1.

# Cil representation of C programs

Cil provides an AST and CFG info from C files.

## Most relevant information

For each function of the AST :

- Locals and formal variables.
- Statements of the Control flow graph.
- Expressions.



# Control flow statement v.s. basic instructions

## Control flow statement

- `if(expr,blockif,blockelse)`
- `switch(exp,case_list)`
- `while(exp)`

## Instruction statement

- `lval=expr`
- `lval=funcall(name,exp list)`

# Model extraction :

Input : Cil AST and Control flow graph

Generated Model : Extended CFG,

$(S_i, S_f, S_{err}, S, \rightarrow \in (S \times R \times S))$

where :

- $S \in (\text{Cil\_types.stmt} \times \text{Abs})$ ,
- $\text{Abs} = \text{Set of SSL formula } \bigcup \perp$ ,
- $R$  is a set of possibly guarded NTS transitions.

# Memory access rules

Correct access:

$$\boxed{\{\phi\} \xrightarrow[\substack{0 \leq [x]_{\phi} + \text{sizeof}(\tau) \leq l_{\text{size}} \\ \wedge ([P]_{\phi} + \text{base}_{\phi}(P)) \equiv 0[\text{sizeof}(\tau)]}]{\text{access}(P)} \{\phi\} \quad \text{alloc}(l) \in SP(\phi) \quad \text{where} \begin{cases} P : \tau*, \\ l \equiv \text{base}_{\phi}(P). \end{cases}}$$

Access to an unallocated address :

$$\boxed{\{\exists l. \phi\} \xrightarrow{\text{access}(P)} \{\perp\} \quad \text{alloc}(l) \notin SP(\phi), l \in \mathcal{F}\text{Vars}(\phi), \text{ where } l \equiv \text{base}_{\phi}(P)}$$

Access outside of an allocated zone, or with an unaligned address :

$$\boxed{\{\phi\} \xrightarrow[\substack{[P]_{\phi} + \text{sizeof}(x) > l_{\text{size}} \\ \vee [P]_{\phi} \not\equiv 0[\text{sizeof}(\tau)]}]{\text{access}(P)} \{\perp\} \quad \text{alloc}(l) \in SP(\phi) \begin{cases} \text{where } P : \tau*, \\ l \equiv \text{base}_{\phi}(P), \end{cases}}$$

## Among other things

- Validity of integer values : Initialization, difference between two pointers, (Valid, Not Valid, Don't Know)
- Transitions not generated when guards can be statically proved false.

## Verification Phase : Reachability Analysis

- Exporting the Ecfg Hierarchical Numerical Transition System.
- Reachability analysis of the error states by FLATA and/or ELDARICA
- If some error state is reachable : An alarm is raised.
- If no error states is reachable : The program is free of the memory fault we consider.