Bismon
a static source code analysis framework using *some* symbolic artificial intelligence techniques.

Basile Starynkevitch - starynkevitch.net/Basile
basile.starynkevitch@cea.fr and basile@starynkevitch.net

CEA/LIST (DILS) - laboratoire de Sûreté des Logiciels -

November 21st, 2019

BISMON
a static source code analysis framework using some symbolic artificial intelligence techniques.

B.Starynkevitch

Introduction
Data and persistence
Metaprogramming approach
future, questions & related project

BISMON is funded by two Horizon 2020 research and innnovation actions:

- CHARIOT, under Grant Agreement 780075.
- DECODER, under Grant Agreement 824231.

So BISMON is European.

(100% funded by the European Commission)

.

Opinions are only mines (not from CEA or E.C.)

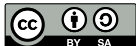
Work in progress !

Introduction
AI and other inspirations

- D.Lenat work on RLL -1 then EURISKO
- J.Pitrat† (1934 - october 2019) pionneering work on CAIA
- my PhD work (1985 - 1990)
- my past GCC MELT work (2008 - 2016)
- FRAMA-C, its ACSL, OCAML runtime, and *non-relational* databases
- GCC > 10MLOC of *C++* (bootstrapped with `g++ -O2 -flto`) and a dozen of DSLs

For references, see the BISMON **draft** report on my home page

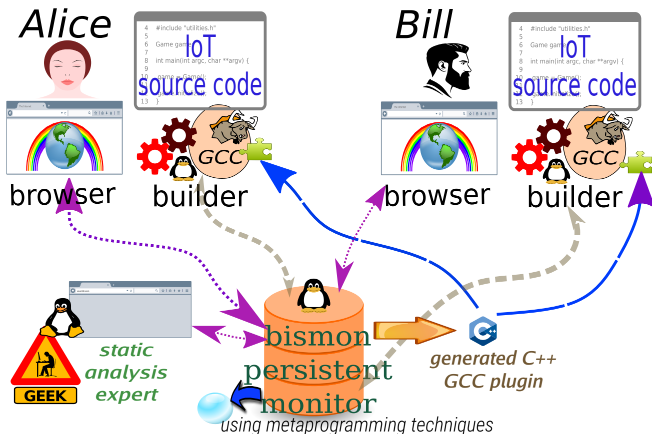
Some GPLv3+ code for LINUX/x86-64 desktop is available
github.com/bstarynk/bismon

These slides are under 

(CC-BY-SA-4) and available from
starynkevitch.net/Basile/Bismon-Starynkevitch-Lamsade-21nov2019.pdf

Introduction
motivations

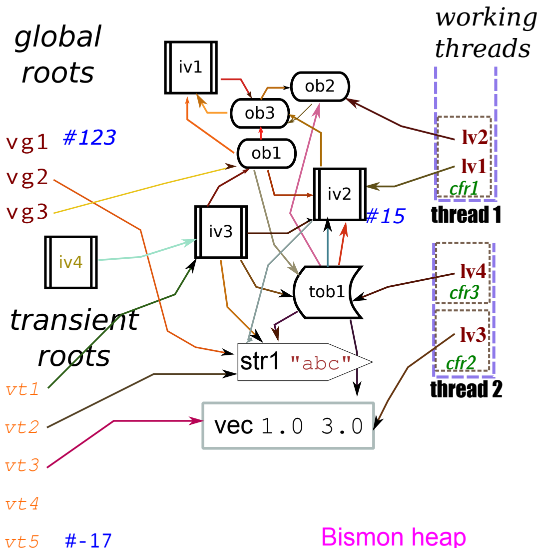
Help small teams of mostly *junior* software developers (e.g. IoT) using LINUX thru a collaborative *Web assistant* tool



BISMON a static source code analysis framework using some symbolic artificial intelligence techniques.
B.Starynkevitch
Introduction
Data and persistence
Metaprogramming approach
future, questions & related project

Data and persistence

the *Bismon* heap **greatly simplified** (my figure borrowed from `refpersys.org`)



BISMON
a static source code analysis framework using some symbolic artificial intelligence techniques.
B.Starynkevitch
Introduction
Data and persistence
Metaprogramming approach
future, questions & related project

- **immutable values** (often with flexible array members in *C* code) :
  - scalar values: tagged 63 bits **integers**, boxed **strings**, boxed **doubles** (not `NaN`, since it is uncomparable) ...
  - composite values :
    - ordered **sets** of objects with $O(\log n)$ membership test
    - sequential **tuples** of objects (same layout in memory as sets)
    - **nodes** with an object connective and zero or more son values.
    - **closures** with their code represented by an object, and zero or more closed values (same data layout as nodes)
- **mutable and lockable** so "heavy" **objects** (each having its *unique objid* e.g. `_5t7pTgRckFK_7hOY5yvx8v3`)

The NIL pointer denotes a lack of value. Every value has its class (an object).

In addition, our GC manages **quasi-values** as an *implementation detail*. A quasi-value is simply a GC-ed memory zone which could belong to some value or object.

Bismon a static source code analysis framework using some symbolic artificial intelligence techniques.

B.Starynkevitch

Introduction

Data and persistence

Metaprogramming approach

future, questions & related project

Garbage collection is:

- well understood in theory but difficult in practice
- dealing with **whole-program properties** of running processes
- related to **virtual memory** and **virtual address space**
- **crucially important for performance** :
  - can be very efficient, at least with single-threaded mostly immutable values (see Ocaml, Haskell, Sbcl or some Jvm implementations)
  - very brittle (a GC bug usually crashes your program)
- practically very dependent of both hardware (CPU cache) and operating system.
- still an **art of delicate trade-offs** (finalizers, decaying or weak pointers, tuning parameters, size of generations, frequency of major GCs, ...)

BISMON a static source code analysis framework using some symbolic artificial intelligence techniques.
B.Starynkevitch
Introduction
Data and persistence
Metaprogramming approach
future, questions & related project

Garbage collection is:

- requiring boring coding conventions and calling conventions at runtime
- needing cooperation from the compiler or code generator
- depending upon compiler optimizations
- **difficult to code**, notably with multi-threading
- **difficult to debug and test** (Heisenbugs)
- wanting **metaprogramming** (the code of GC support routines should be easily generated, since very regular and)
- using algorithms (in copying GCs) close to persistence, since traversing the entire heap graph.

Data and persistence

garbage collection in Bismon (3/3)

Today, in november 2019, it is :

- slow, bad but easy naive, precise, mark-and-sweep GC algorithm
- does not scale yet to large 50Gbytes heaps
- should be generated by metaprogramming
- multi-thread "friendly" ☺ : stop the world variety (joke!)
- a **design bug** in Bismon commit `6b26b802b8c0f4dee3053` : Gtk recursive event loop breaking our GC invariants.
- should become a copying generational GC for immutable values, and a tri-color marking onne for mutable objects
- then the write-barrier has to be implemented by changing the metaprogram (i.e. our C code generator)
- **not `dlclose`-ing** generated plugins (still science-fiction but should be theoretically done for garbage collection of generated code).

Every **object** has :

- its **globally unique**, *randomly generated*, **constant objid** like `_8dgEp1oxLMz_5iGP2Eq1wn7` ($\approx 128 bits$)
- its `pthread(7)` **mutex** lock for synchronization
- its space number
- its modification time
- its atomic **class** - itself an object
- its **attributes**, associating key *objects* to non-nil values
- its optional **routine** pointer, with ...
- an optional object describing the **signature** of that routine.
- its **components**, a vector of values
- some optional **payload** which is a data owned by the object.

BISMON
a static source code analysis framework using some symbolic artificial intelligence techniques.
B.Starynkevitch
Introduction
Data and persistence
Metaprogramming approach
future, questions & related project

Of course, the class, the attributes, the components, the payload may change, usually under protection of the mutex. The routine and signature change thru `dlsym(3)`.

Examples of payloads include :

- mutable vector of values
- mutable class information (vector of superclasses, dictionnary of methods, ...)
- mutable hashed set of objects
- Web sessions or user information
- dictionnaries associating strings to objects
- etc ...

So BISMON **objects are very versatile** (similar to those of EURISKO or CAIA, more general that JAVASCRIPT objects, with OBJVLISP model ...).

Data and persistence

Orthogonal persistence (1 / 2)

Most of the Bismon **heap** (but not the call stacks) is persisted in **textual files** (so `git` friendly). For example in our `store2.bmon` file :

```
«_6dKbq51BdxU_7XmAmI0XbBR objid
± 1544448437.314 modtime
 ∈ _4GJJnvyrLyW_5mhopCYvh8h |=basiclo_cexpansion| class
↦ _01h86SAf0fg_1q2oMegGRwW |=comment| attribute key
"emit int v_comp"   corresponding attribute value
↦ _0jFqaPPHgYH_5Jpj0PxQ67p |=arguments|
* _0jFqaPPHgYH_5Jpj0PxQ67p ( _41F1rKwGbaA_30OJWKsqNWy
    * _5MLPTLuT4ey_0YKIUpvXybX ( _0ZL8gaI6sH8_7UPhmAQcwMe))
↦ _9OzBvYbDWm8_3XA4wkArOmo |=expander|
_6cFSE2rDxvF_99QhDhtBeS4
»_6dKbq51BdxU_7XmAmI0XbBR end objid
```

In november 2019, we have nearly 3370 persisted objects.

Data and persistence

Orthogonal persistence (2 / 2)

Most values are persisted. Some values are not, they are **transient**. Obviously the user interface is reified as transient values.

At startup, the BISMON process **loads** the persisted heap. Before exiting, the BISMON process **dumps** the persisted heap. Then it usually would be `git commit`-ed.

Conceptually, **the** BISMON **heap never dies** : process `./bismon` would be started at morning and dumps its updated state at evening.

The persistence machinery starts the dump from a few **predefined objects** in space 1, notably **`the_system`** i.e. `€_4ggW2XwfXdp`.... It uses a depth-first approach with an hashed set of dumped objects, and a FIFO queue of objects to be scanned then dumped. Generated *C* code is re-emitted at dump time.

BISMON a static source code analysis framework using some symbolic artificial intelligence techniques.

B.Starynkevitch

Introduction

Data and persistence

Metaprogramming approach

future, questions & related project

GCC MELT was a LISP-like bootstrapped domain specific language translated (or "transpiled") to C++ plugin code of GCC

```
;;; citerator on lists
(defciterator foreach_pair_component_in_list
  (lis) ;start formals
  eachlist ;state
  (curpair curcomp) ;local formals
  :doc #{The $FOREACH_PAIR_COMPONENT_IN_LIST iterator goes within a
  list, given by the start formal $LIS …}#
  #{/* start foreach_pair_component_in_list $EACHLIST */
  for ($CURPAIR = melt_list_first( (melt_ptr_t)$LIS);
    melt_magic_discr((melt_ptr_t) $curpair) == MELTOBMAG_PAIR;
    $CURPAIR = melt_pair_tail((melt_ptr_t) $CURPAIR)) {
    $CURCOMP = melt_pair_head((melt_ptr_t) $CURPAIR); }#
  #{ } /* end foreach_pair_component_in_list $EACHLIST */
  $CURPAIR = NULL; $CURCOMP = NULL; }# )
```

So *C* or *C++* code with holes or metavariables in **macrostrings**

BISMON
a static source code analysis framework using some symbolic artificial intelligence techniques.

B.Starynkevitch

Introduction

Data and persistence

Metaprogramming approach

future, questions & related project

Typical usage was

```
(cond ( (is_closure f)
  (foreach_pair_component_in_list
  (lis)
  (curpair curcomp)
  (multiple_put_nth tup count (f curcomp))
  (setq count (+i count 1)))
  )
etc...
```

Metaprogramming approach

Metaprogramming in BISMON(1 / 2)

Principles:

- “light” things are *often* (but not always!) represented by immutable nodes (small expressions)
- “heavy” things are usually objects (variables, blocks, statements, etc...)
- a hierarachy of metaprogramming classes exist.

Some constant-related metaprogram `BM_makeconst`is needed for `BMK_0saT3fDy8bt_1R3vTikLuIx` in hand-written C code, representing the object `_0saT3fDy8bt_1R3vTikLuIx`

Bismon a static source code analysis framework using some symbolic artificial intelligence techniques.

B.Starynkevitch

Introduction

Data and persistence

Metaprogramming approach

future, questions & related project

The C code generator routines are partly hand-written, partly generated.

Ours Bismon domain specific language, represented by persistent **objects** (not textual representation!) has:

- functions
- lambda-s
- object and value variadic creation primitives
- machinery similar to Gcc Melt code chunks
- `switch` on objects
- conditional statements
- locking statements
- function application
- message sending

Bismon
a static source code analysis framework using some symbolic artificial intelligence techniques.
B.Starynkevitch
Introduction
Data and persistence
Metaprogramming approach
future, questions & related project

- generate both JavaScript and HTML5 (mostly done)
- complete a Web interface (this could make Bismon practically usable)
- add higher level constructs
- add rule machinery
- improve, or even generate, the GC
- generate GCC plugins (or perhaps interact with other interpreters)

Bismon
a static source code analysis framework using some symbolic artificial intelligence techniques.

B.Starynkevitch

Introduction
Data and persistence
Metaprogramming approach
future, questions & related project

Demo ?
Questions ? Thanks !

refpersys.org

hobby AGI GPLv3+ LINUX-only project, with enthusiastic partners -more wanted-, embryonic, risky, preparing my retirement, unrelated to BISMON or to static analysis