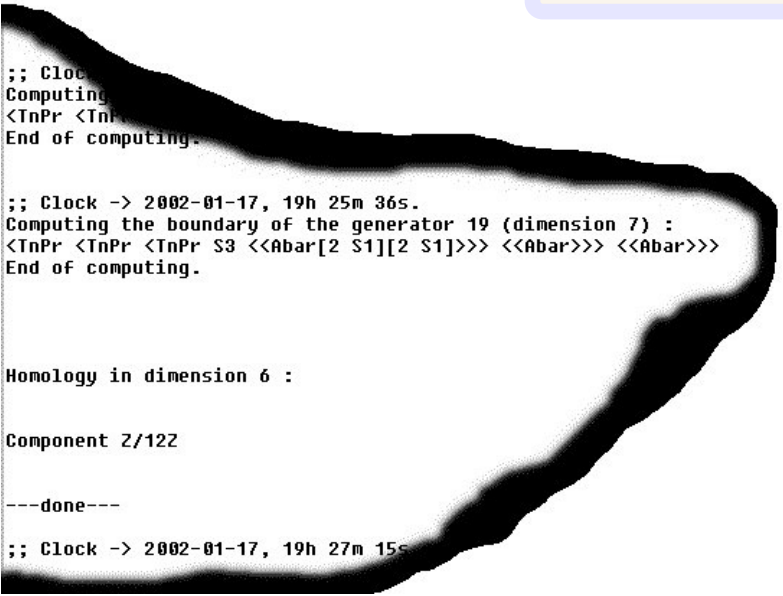
The Sphynx kernel



Francis Sergeraert, Univ. of Grenoble and Lorient

Logroño, May 2023

Semantics of colours:

Blue	=	**"Standard" Mathematics**
Red	=	**Constructive, effective, algorithm, machine object, ...**
Violet	=	**Problem, difficulty, obstacle, disadvantage, ...**
Green	=	**Solution, essential point, mathematicians, ...**

Plan.

- **1. Introduction.**
- **2. The general CLOS environment.**
- **3. The toy example of implementing magmas through the sphynx-class /MAGMA/.**
- **4. How it works.**

1. Introduction.

Goal of the Sphynx program:

Designing a computer algebra system
convenient to program computations
in algebraic topology,
more precisely in constructive algebraic topology.

Must satisfy: Code production is natural and easy,
Code modification and extension are natural and easy,
Code is easily readable,
Complete, safe and efficient typing,
Efficient solution for functional programming,
Easy switch efficient development / efficient production.

Must be able to define

rich sets of complex mathematical classes,

with a complex hierarchy,

taking account of this hierarchy

to optimize the structure of this program

and also its use.

Warning: Using this kernel shows the goal

is essentially satisfied

But the kernel itself is quite complex.

The most important: Safe **and** Efficient Typing.

Safe = Strict and arbitrary sophisticated typing
of the elementary objects.

Example: $x \in K(\mathbb{Z}, 2)_n \Leftrightarrow x$ is a list of integer lists
of lengths $(n-1, n-2, \ldots, 0)$.

`((7 6 -6) (0 5) (-10) ())` $\in K(\mathbb{Z}, 2)_4$

`((7 6 -6 ) (0 5 1) (-10) ())` $\notin K(\mathbb{Z}, 2)_4$

Safe = During development
the type of any low level object is checked,
regardless of its sophisticated type.

Efficient = When the development is finished,
when the program is used
for actual computations,
these type checkings are no longer used.

Brief "history" of typing:

Fortran-1970: `i...,j...,...,n...` = **integers**
other **identifiers** = **float numbers**

Lisp-1.5 (**McCarthy**): **No typing at all**.

PL/I, **Pascal**, **C**, **C++**, **Caml**, **Common Lisp** ...: **Various weak typings**.

Coq, **HoTT**, **Lean**, ...: **Dependent types** and **strict typing**.
Programs completely proved.

Brief "history" of typing:

Fortran-1970: `i...,j...,...,n...` = integers
other identifiers = float numbers

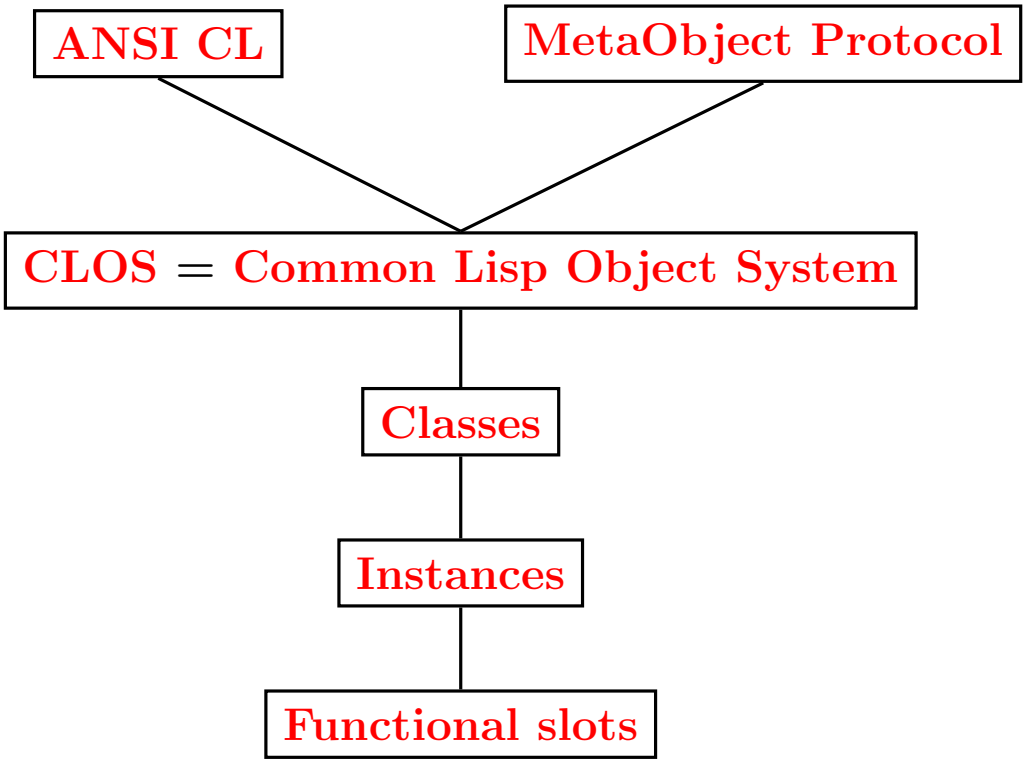
Lisp-1.5 (McCarthy): No typing at all.

PL/I, **Pascal**, **C**, **C++**, **Caml**, **Common Lisp** ...: Various weak typings.

Sphynx kernel: Dependent types, strict typing and efficiency.
Programs reasonably reliable, but not completely proved.

Coq, **HoTT**, **Lean**, ...: Dependent types and strict typing.
Programs completely proved.
Efficiency?

2.The general CLOS environment.



Rough history of CL OOP:

CL-1984 = First Steele's CL definition:

CLTL-1 (No OOP at all).

First OOP systems added to CL:

LOOPS, FLAVORS, OBJECT LISP, ...

1984-1990: MetaObject Protocol (Kiczales, Sussman, ...)

$\Rightarrow$ **CLOS = Common Lisp Object System**

CL-1990 = Final Steele's CL Definition $\simeq$ ANSI-CL:

$$\text{CLOS} \subset \text{ANSI-CL}$$

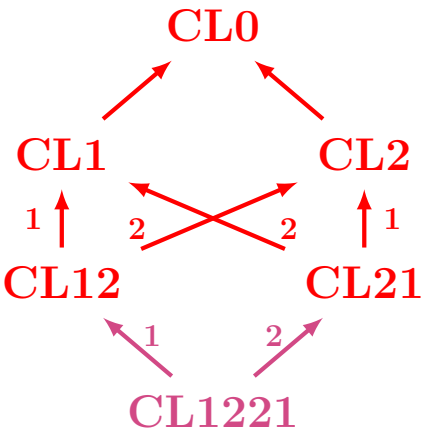
$$\text{CLOS} \subset \text{MetaObject Protocol}$$

~~$\text{MetaObject Protocol} \subset \text{ANSI-CL}$~~

Main components of CLOS:

- Classes.
  - Multi-inheritance.
  - Possible redefinition.
  - Elements = Instances = Collections of slots.
  - Powerful initialization of slots.
  - Complex hierarchies processed by `class-precedence-list`.
- Generic functions and methods.
  - Uses multi-inheritance.
  - Possible qualifiers `:before`, `:after` and `:around`
  - $\Rightarrow$ Arbitrary complex combinations of the methods of a generic function.

Simplest diagrams of inheritance:



3. The toy example of Magmas.

Definition. A magma is a set

with a binary internal composition law $*$.

Example: $M = \mathbb{N}$ with the composition law:

$$* : (x, y) \longmapsto x + 2y + xy$$

Sphynx implementation of a set.

An instance of the Sphynx class /SET/ has four essential slots:

1. DFNT (Definition): A list used only for documentation.
2. MMBR (Membership): A predicate strictly defining the type of the elements.
3. CMPR (Comparison): A total order for the elements of the set.
4. CMPR= : A predicate deciding the equality of two elements.

Example: the set *NNI* of the non-negative integers.

Apparent Sphynx implementation of a Magma:

1. DFNT (as before).
2. MMBR (as before).
3. CMPR (as before).
4. CMPR= (as before).
5. BNRL : The BiNaRy composition Law of this magma.

$$\text{MAGMA} := \text{SET} + \text{BNRL}$$

Actual Sphynx implementation of a Magma:

1. DFNT (as before).
2. MMBR (as before).
3. CMPR (as before).
4. CMPR= (as before).
5. BNRL: SOURCE: (This-Magma This-Magma)
   TARGET: This-Magma
   IMPL: The actual Lisp functional object
   Other technical slots

$$\text{MAGMA} := \text{SET} + \text{BNRL}$$

4. Technicalities of the SPHYNX Kernel.

SPHYNX Classes $\in$ Graph under the top class /SPHYNX-OBJECT/.

Remark: Graph $\neq$ Tree.

Any SPHYNX-CLASS is defined by SPHYNX-CLASS.

Any SPHYNX CLASS is a SubClass of /SPHYNX-OBJECT/.

A class hierarchy is defined by SPHYNX-CLASS and
computed by the MOP function:
COMPUTE-CLASS-PRECEDENCE-LIST

Defining a new SPHINX-CLASS /CL/.

Recipe:

- Use a SPHYNX-CLASS statement
  to define the immediate super-classes.
  and the direct slots.
- Use a FUNCTIONAL-SLOT(S) statement
  to define the possible functional slots.
- Use a BIND-SLOTS statement to define
  the initialization process
  of an instance of this SPHYNX-CLASS.

A BIND-SLOTS method can be quite complex.

How a /SPHYNX-OBJECT/ of class /CL/ is constructed:

- Allocate a fresh instance of class /CL/.
- Determine the class-precedence-list for /CL/.
- For every item of the class-precedence-list,
  call the correponding BIND-SLOTS method.

Essential:

The order of BIND-SLOTS invocations is :

- From the most general classes
- to the most specific classes.

according to the class-precedence-list.

Implies in rare cases a BIND-SLOTS method
must consider by anticipation
a more specific class.

Typical example: the class of constructive sets = /SET/C.

Example: The set of the integers 3, 5 and 7.

Switching between

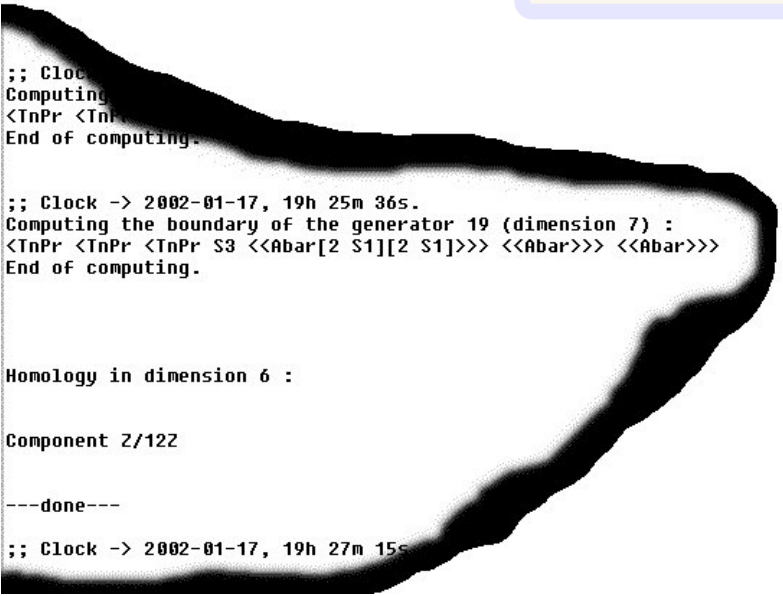
DEVELOPMENT and PRODUCTION

thanks to

the parameter SYSDEF::*CHECK-MEMBERSHIP-P*.

To be used mainly for the definitive compiled version.

The END



Francis Sergeraert, Univ. of Grenoble and Lorient

Logroño, May 2023