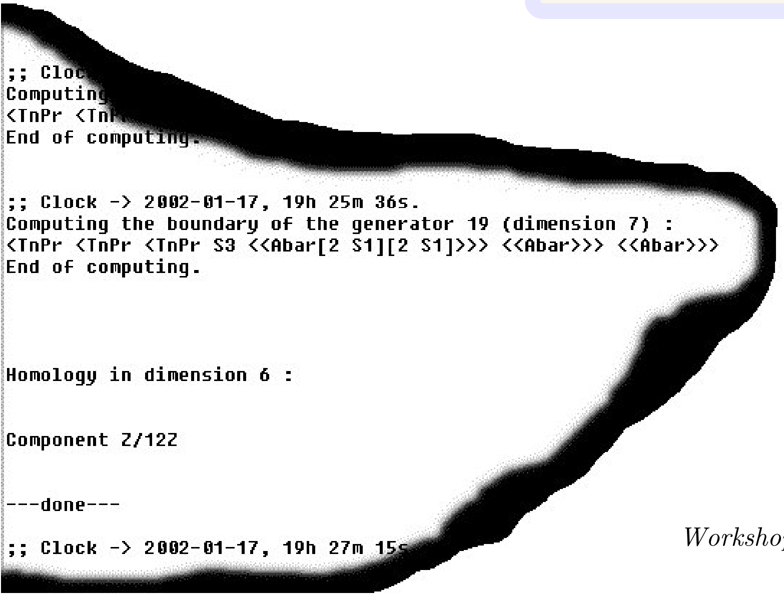
**Semantics of colours:**

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

# Functional Programming

# and

# Complexity



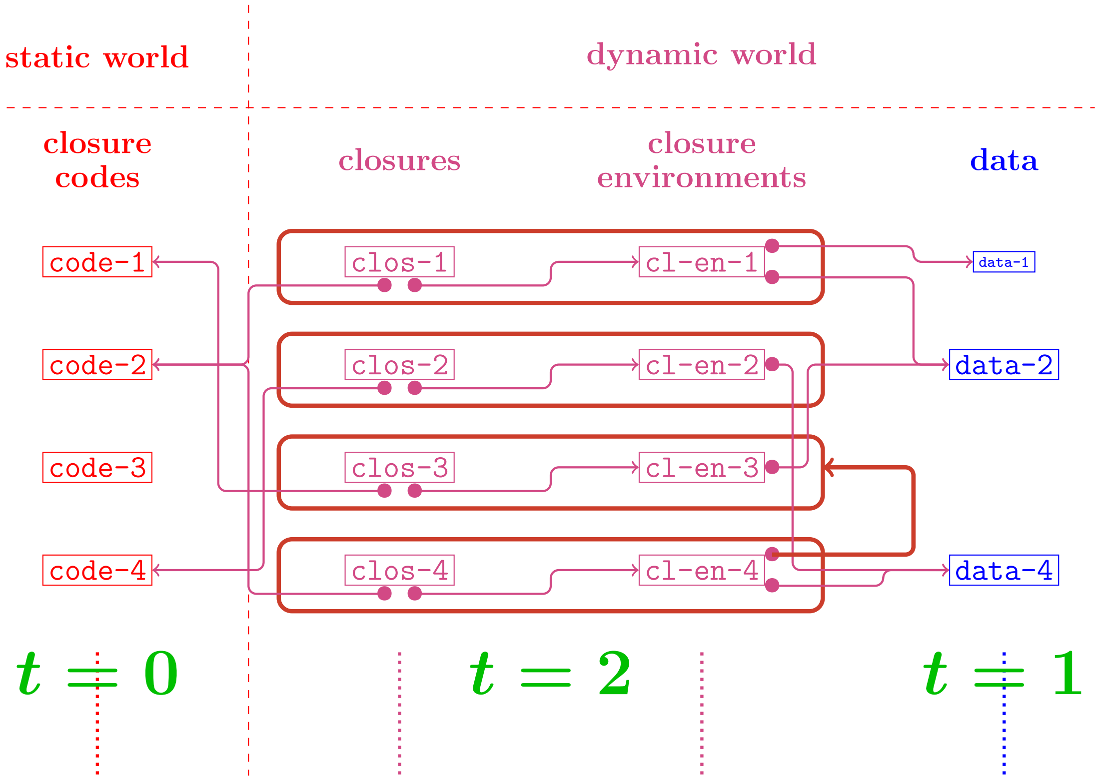


*Francis Sergeraert, Institut Fourier, Grenoble*


*Workshop on Algebra, Geometry and Proofs in Symbolic Computations*
*Fields Institute, Toronto, December 2015*

## Plan.

- **1. Introduction.**
- **2. Dynamic generation of functional objects is necessary.**
- **3. Technology of Closure Generation.**
- **4. Functional Programming and Polynomial Complexity.**
- **5. Particular case of the Kenzo program for the homology of iterated loop spaces.**
- **6. Challenges for Proof Assistants.**


static world
dynamic world
closure codes
closures
closure environments
data
code-1
code-2
code-3
code-4
clos-1
clos-2
clos-3
clos-4
cl-en-1
cl-en-2
cl-en-3
cl-en-4
data-1
data-2
data-4
$t = 0$
$t = 2$
$t = 1$

# 1. Introduction.

**Standard work in complexity study:**

1. **Write down a program $P : \mathcal{T}_1 \to \mathcal{T}_2$.**

2. **Determine the complexity of $P$ as**

   **a function $\chi : \mathbb{N} \to \mathbb{N}$ satisfying:**

$$\tau(P, \omega_1) \leq \chi(\sigma(\omega_1))$$

**with:** $\mathcal{T}_1$, $\mathcal{T}_2$ = **some data types;**

$\tau(P, \omega_1)$ = **computing time of $P$**

**working on the arbitrary input $\omega_1 \in \mathcal{T}_1$;**

$\sigma(\omega_1)$ = **size of the input $\omega_1$.**

**Subprogram technology.**

**Simple case: $P$ contains two subprograms $p_1$, and $p_2$.**

**A run of $P: \omega_1 \mapsto \omega_8$ could be:**

$$\omega_1 \overset{P'}{\mapsto} \omega_2 \overset{p_1}{\mapsto} \omega_3 \overset{P''}{\mapsto} \omega_4 \overset{p_2}{\mapsto} \omega_5 \overset{P'''}{\mapsto} \omega_6 \overset{p_1}{\mapsto} \omega_7 \overset{P''''}{\mapsto} \omega_8$$

**A subprogram $p$ may also call other subprograms $p'$, $p''$, $\ldots$**
**or recursively call itself, and so on.**

**The complexity of the subprograms**
**usually is a subproblem of the complexity of $P$,**
**but the nature of the problem is the same.**

**In "ordinary" programming,**

**the program and in particular all the subprograms**

**are written down $\boxed{\text{before}}$ execution,**

**left unchanged during the execution.**

**Functional programming is the art of writing programs**

**which $\boxed{\text{dynamically generate}}$ new functional objects**

**$\boxed{\text{during}}$ the execution.**

**Question: How to process this new context**

**when studying the complexity of such a program?**

# Main tool to study the problem:

**Notion of $\boxed{\text{LEXICAL CLOSURE}}$**

**Main results:**

**Most dynamic generations of functional objects are efficiently covered by (lexical) closures.**

**Studying the complexity of these objects is divided in three steps:**

- **Arbitrary segment of program preparing the closure generation.**
- **Actual generation of the closure (often free!).**
- **Execution of the closure body.**

**Example** : The Kenzo program contains

about 250 segments of Lisp code

devoted to closure generations.

Several thousands of closures are generated

for every meaningful use of Kenzo.

Proving polynomiality ??

Easy through the notion of polynomial configuration.

Corollary: The computation $X \xmapsto{\text{Kenzo}} \pi_n X$

is polynomial with respect to $X$ simply connected, $n$ fixed.

## 2. Dynamic generation of functional objects is necessary.

Notion of **simplicial set with effective homology** $X_{EH}$:

$$X_{EH} = (X, E_*, \varepsilon)$$

with: $X$ = one **functional object** implementing the **face operator** of $X$ (**X** often $\boxed{\text{non-finite}}$).

$E_*$ = **Free $\mathbb{Z}$-chain complex of finite type**.

$\varepsilon$ = **Strong homology equivalence** = .../...

$$\varepsilon \;=\; ((\widehat{C}_*, \widehat{d}), f, g, h, f', g', h')$$

$$\begin{tikzcd}
 & \widehat{C}_* \arrow[loop left, "h"] \arrow[loop right, "h'"] \arrow[loop below, "\widehat{d}"] \arrow[ddl, shift right, "f"'] \arrow[ddr, shift left, "f'"] & \\
\boxed{X} \arrow[d, mapsto] & & \\
C_*X \arrow[uur, shift right, "g"'] & & E_* \arrow[uul, shift left, "g'"]
\end{tikzcd}$$

**In general, all violet objects are $\boxed{\text{not of finite type}}$**

**necessarily functionally coded.**

**Implementation of a simplicial set $\boldsymbol{X}$ not of finite type:**

$$\boldsymbol{X} \; = (\mathcal{T}_X, \quad \partial_X : \mathcal{T}_X \tilde{\times} \mathbb{N} \rightarrow \mathcal{T}_X)$$

**with:**

$\mathcal{T}_X$ = **Type of the simplices of $\boldsymbol{X}$ ;**

$\partial_X$ = **Face operator of $\boldsymbol{X}$: $\partial_X(\boldsymbol{\sigma}, \boldsymbol{i})$ = $\boldsymbol{i}$-th face of $\boldsymbol{\sigma}$;**

**Whitehead tower's method (?!) computing homotopy groups.**

1. $X$ simply connected given.

2. Hurewicz $\Rightarrow \pi_2 X = H_2 X$ computable (?).

3. $X_3 = K(\pi_2, 1) \times_\tau X$ $(= X$ with $\pi_2$ killed$)$ is 2-connected.

4. Hurewicz $\Rightarrow \pi_3 X = \pi_3 X_3 = H_3 X$ computable (??).

5. $X_4 = K(\pi_3, 2) \times_\tau X_3$ $(= X_3$ with $\pi_3$ killed
$= X$ with $\pi_2$ and $\pi_3$ killed$)$ is 3-connected.

6. Hurewicz $\Rightarrow \pi_4 X = \pi_4 X_4 = H_4 X_4$ computable (???).

7. $X_5 = \ldots\ \ldots$

**Groups actually computable ???**

$\boxed{\textbf{Answer = Yes}}$ **if all the objects**

**are simplicial sets** $\boxed{\textbf{with effective homology}}$

$\Rightarrow$ **Actual Whitehead's algorithm.**

1. $[X, E_*^X, \varepsilon^X]$ **simply connected given.**

2. $E_*^X$ **of finite type** $\Rightarrow H_2 X = H_2 E_*^X$ **computable !**

3. **Effective homology theory** $\Rightarrow$

$$[X, E_*^X, \varepsilon^X] + [K(\pi_2, 1), E_*^{K(\pi_2,1)}, \varepsilon^{K(\pi_2,1)}] \mapsto [X_3, E_*^{X_3}, \varepsilon^{X_3}]$$

$=$ **version with effective homology of** $X_3$**.**

**3.** Effective homology theory $\Rightarrow$

$$[X, E_*^X, \varepsilon^X] + [K(\pi_2, 1), E_*^{K(\pi_2,1)}, \varepsilon^{K(\pi_2,1)}] \mapsto [X_3, E_*^{X_3}, \varepsilon^{X_3}]$$

= version with effective homology of $X_3$.

**4.** $E_*^{X_3}$ of finite type $\Rightarrow \pi_3 X = \pi_3 X_3 = H_3 X_3$ computable !

**5.** Effective homology theory $\Rightarrow \ldots \ldots$

Remark: Requires also versions with effective homology of the Eilenberg-MacLane spaces $K(\pi, n)$ for $\pi$ = Abelian group of finite type.

Effective homology theory $\Rightarrow [K(\pi, n), E_*^{K(\pi,n)}, \varepsilon^{K(\pi,n)}]$.

**Finally Whitehead's algorithm $X \mapsto \pi_n X$**

**must necessarily be decomposed:**

$$X \mapsto X_3 \mapsto \cdots \mapsto X_{n-1} \mapsto X_n \mapsto H_n X_n$$

**with auxiliary Eilenberg-MacLane spaces.**

**All the objects $X_i$ and $K(\pi, i)$ contain lots of components**

**that are functional objects.**

**$\Rightarrow$ A complexity study of the Whitehead's algorithm requires**

**a study of the cost**

**of the dynamic generation of all these functional objects.**

# 3. Technology of Closure Generation.

**Two facts:**

- **A machine can only execute program segments "imagined" by the programmer.**
- **A programmer remains the only "object" finally able to "imagine" from scratch program segments.**

**Functional object $\supset$ Program segment**

**$\Rightarrow$ A machine may not itself "imagine" such a segment.**

**$\Rightarrow$ A machine cannot create from scratch a functional object.**

**Facts:**

- **A machine can only create a functional object following a $\boxed{\text{pattern}}$ defined by the programmer.**
- **In particular, the code of the functional object must be defined by the programmer before execution.**

**Finally:**

- **A lexical closure is a $\boxed{\text{constant code}}$ combined with arbitrary $\boxed{\text{extra data}}$ constituting its own environment.**

# General organization of closures:

- Most problems of dynamic generation of functional objects

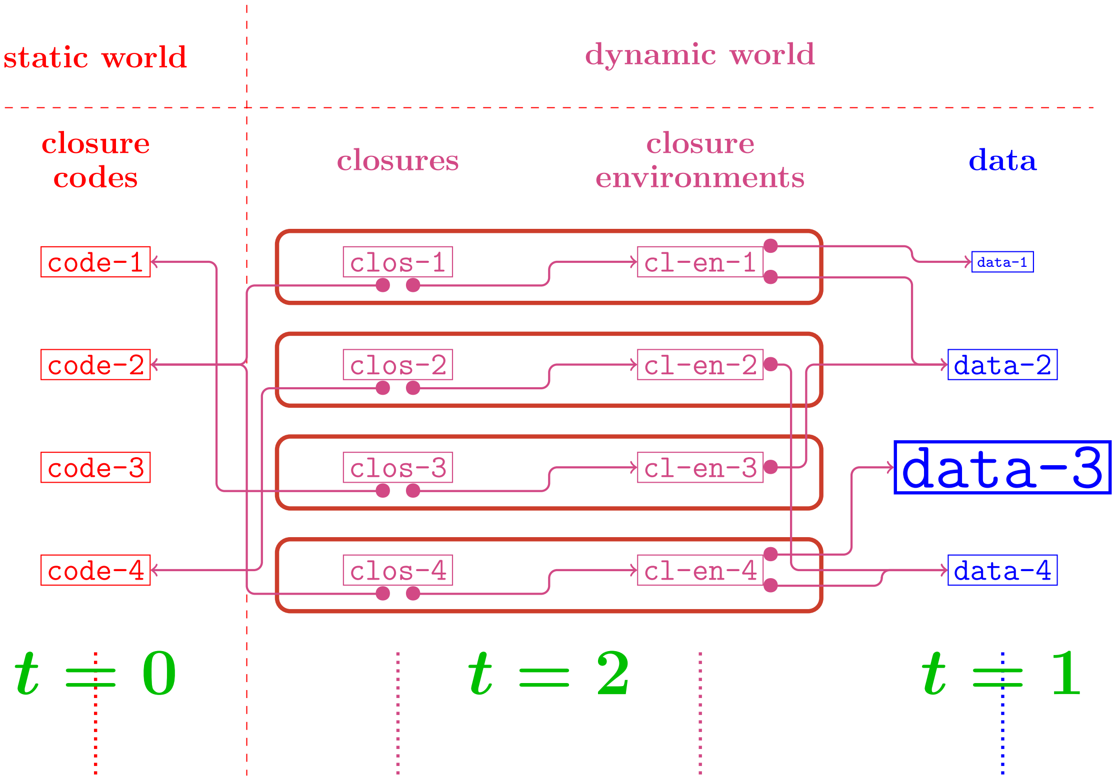
  can be solved through the notion of (lexical) closure.
- A closure is a pair [code + environment].
- The code of a closure

  must be defined by the programmer before execution.
- An arbitrary number of generated closures

  may share the same code.
- Only one copy of such a code is in the memory,

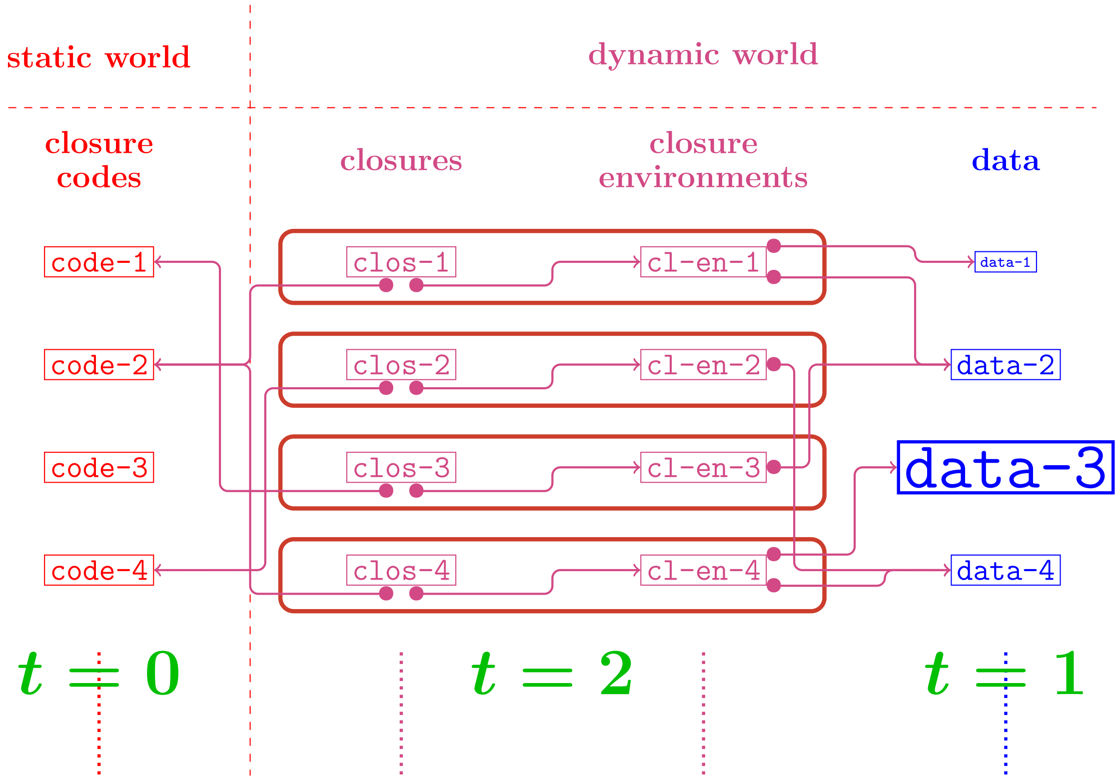
  present before execution.
- All the closures sharing this code can reach it for use

  when they are invoked.
- ······


static world
dynamic world
closure codes
closures
closure environments
data
code-1
code-2
code-3
code-4
clos-1
clos-2
clos-3
clos-4
cl-en-1
cl-en-2
cl-en-3
cl-en-4
data-1
data-2
data-3
data-4
$t = 0$
$t = 2$
$t = 1$

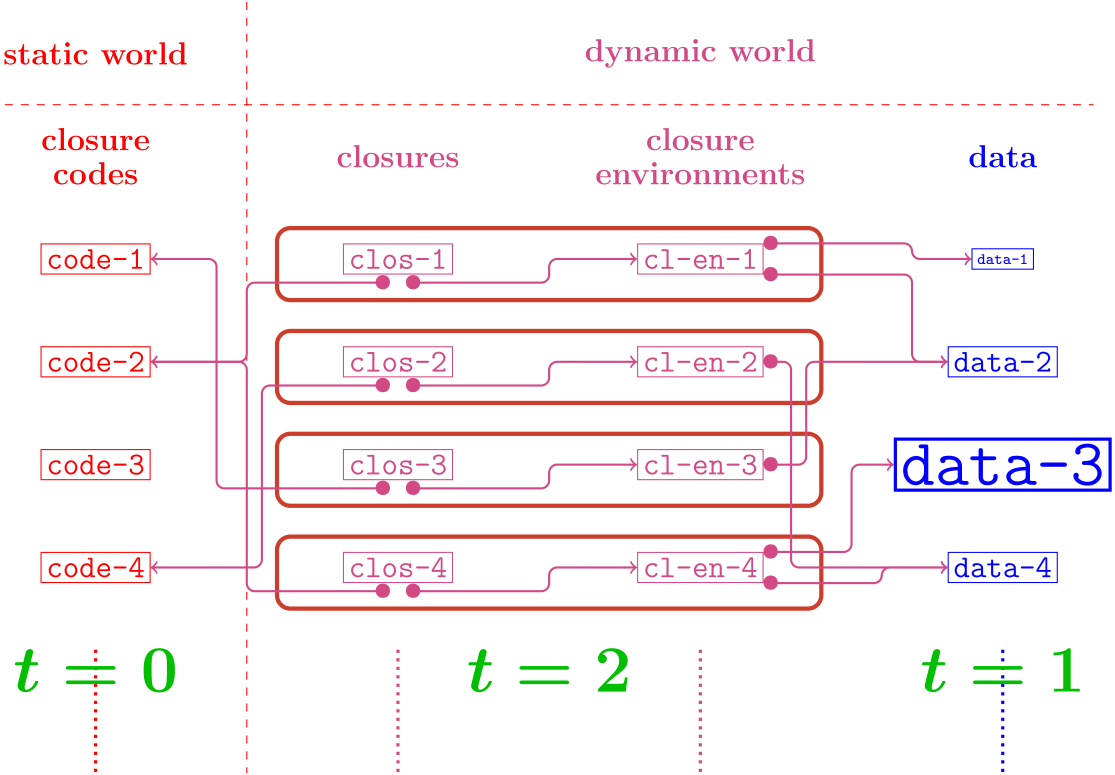
## General organization of closures (continued).

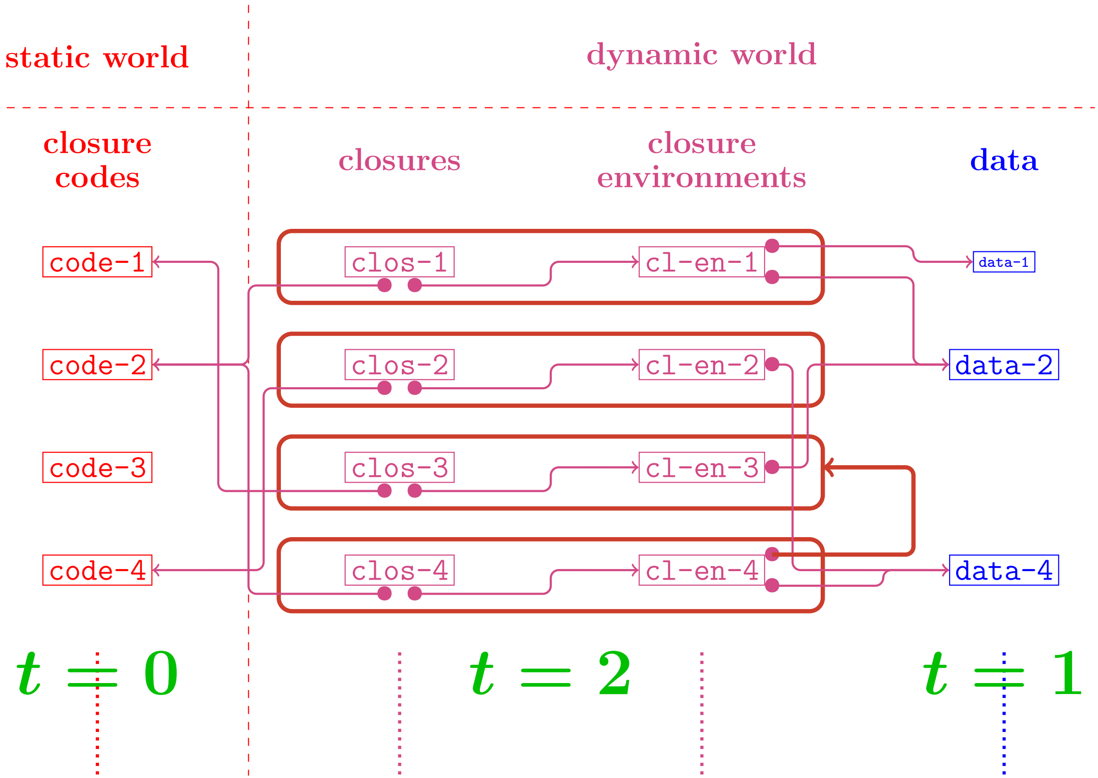
- ······
- All the closures sharing this code can reach it for use when they are invoked.
- The environment of a closure is dynamically generated when the closure is generated.
- The environment of a closure is a table of machine addresses.
- This table is made of the addresses of the objects constituting the environment of this particular copy of the closure.
- ······


static world
dynamic world
closure codes
closures
closure environments
data
code-1
code-2
code-3
code-4
clos-1
clos-2
clos-3
clos-4
cl-en-1
cl-en-2
cl-en-3
cl-en-4
data-1
data-2
data-3
data-4
$t = 0$
$t = 2$
$t = 1$

## General organization of closures (continued).

- ······
- The objects of this environment, not their addresses, may be arbitrarily modified during execution.
- The environments of the closures sharing the same code have the same format, in particular the $\boxed{\text{same size}}$,
- These environments and the corresponding closures are $\boxed{\text{generated in constant time}}$.
- $\Rightarrow$ With respect to complexity problems, most often, the $\boxed{\text{generation of a closure is } \boxed{\text{free}}}$.


static world
dynamic world
closure codes
closures
closure environments
data
code-1
code-2
code-3
code-4
clos-1
clos-2
clos-3
clos-4
cl-en-1
cl-en-2
cl-en-3
cl-en-4
data-1
data-2
data-3
data-4
$t = 0$
$t = 2$
$t = 1$


static world
dynamic world
closure codes
closures
closure environments
data
code-1
code-2
code-3
code-4
clos-1
clos-2
clos-3
clos-4
cl-en-1
cl-en-2
cl-en-3
cl-en-4
data-1
data-2
data-4
$t = 0$
$t = 2$
$t = 1$

## 4. Functional Programming and Polynomial Complexity.

**Assumed: Functional Programming is done**
**through (lexical) closures**
**and it is legal to ignore**
**the generation cost of the closures.**

$$\Leftrightarrow$$

**The number of generated closures is uniformly bounded.**

**Type reminder:**

**Atomic types: Numbers, booleans, characters, symbols, ...**

$\boldsymbol{a}$ **atomic object** $\Rightarrow$

**Obvious notion of size** $\boldsymbol{\sigma(a)}$.

**Decidable types: Atomic objects, lists, arrays, records, ...**

**made of decidable objects.**

$\boldsymbol{a}$ **decidable object** $\Rightarrow$

**Obvious notion of size** $\boldsymbol{\sigma(a)}$.

**Functional types**: $\mathcal{T}_1$ and $\mathcal{T}_2$ = types already defined.

$\mathcal{T}_1 \to \mathcal{T}_2 :=$ types of functional objects $\alpha$ satisfying

$$a \in \mathcal{T}_1 \quad \Rightarrow \quad \alpha(a) \text{ terminates and } \alpha(a) \in \mathcal{T}_2.$$

The functional types can in turn be used

to compose other arbitrary complex types.

Example: $A, \ldots, H$ decidable types. Then the type:

$$[(A \to B) \to (C \to D)] \to [(E \to F) \to (G \to H)]$$

is defined.

What about a $\boxed{\text{size function}}$ for the objects of this type?

 **In ordinary programming:**

$$\alpha : A \to B \quad \text{with } A \text{ and } B \text{ decidable.}$$

**Then:**

$$\sigma(\alpha(a)) \leq \tau(\alpha, a)$$

**<u>Proof</u>: Turing machine model.**

**In functional programming:**

$$\alpha : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$$

**can be a very small program $\alpha$ producing very quickly**

**a very small functional object $\alpha(a) : \mathbb{N} \to \mathbb{N}$**

**being a terrible Ackermann function.**

**Solution ???**

**Standard type equivalences via currying and uncurrying:**

$$A \to (B \to C) \underset{\text{uncurrying}}{\overset{\text{currying}}{\rightleftarrows}} (A \times B) \to C$$

$$\mathcal{T} := \boxed{\begin{array}{c} C \to D \\ \uparrow \\ A \to B \end{array}} \longrightarrow \boxed{\begin{array}{c} G \to H \\ \uparrow \\ E \to F \end{array}}$$

**Uncurrying $\Rightarrow$ $\mathcal{T}$ is equivalent to:**

$$\{[[(A \to B) \times C] \to D] \times [(E \to F) \times G]\} \longrightarrow H$$

**with all targets decidable if $A, \ldots, H$ are.**

**Definition:** A (polynomial) configuration for a type $\mathcal{T}$

is a map: arrow $\mapsto$ degree

defined on the arrows of a complete uncurrying of $\mathcal{T}$.

**Examples:** $\mathcal{T}$ decidable $\Rightarrow$ empty configuration.

$A \to B$ with $A$ and $B$ decidable.

A configuration is simply a degree $d$.

Means we intend to work only with

the functional objects $\alpha \in (A \to B)$

proved $\leq d$-polynomially complex:

$$\tau(\alpha, a) \leq k(1 + \sigma(a)^d)$$

**A configuration for :** $\mathcal{T} := \boxed{\begin{array}{c} C \to D \\ \uparrow \\ A \to B \end{array}} \longrightarrow \boxed{\begin{array}{c} G \to H \\ \uparrow \\ E \to F \end{array}}$

$$= \{[[(A \to B) \times C] \to D] \times [(E \to F) \times G]\} \longrightarrow H$$

**A configuration for :** $\mathcal{T} :=$ 
$$\begin{tikzcd} \boxed{\begin{matrix} C \to D \\ \uparrow \\ A \to B \end{matrix}} \arrow[r] & \boxed{\begin{matrix} G \to H \\ \uparrow \\ E \to F \end{matrix}} \end{tikzcd}$$

$$= \left\{ \left[ [(A \xrightarrow{d_B} B) \times C] \xrightarrow{d_D} D \right] \times [(E \xrightarrow{d_F} F) \times G] \right\} \xrightarrow{d_H} H$$

$$\chi = (d_B, d_D, d_F, d_H) \in \mathrm{Conf}_{\mathcal{T}}$$

$$\left\{\left[[(A \overset{d_B}{\to} B) \times C] \overset{d_D}{\to} D\right] \times [(E \overset{d_F}{\to} F) \times G]\right\} \xrightarrow{d_H} H$$

<u>**Definition**</u>: $(\mathcal{T}, \chi) := (\mathcal{T}', \chi') \overset{d_H}{\to} H$

with $H$ decidable and $\chi = (\chi', d_H)$.

$\alpha \in \mathcal{T} := (\mathcal{T}' \to H)$.

Then $$\boxed{\sigma_\chi(\alpha) := \sup_{a \in (\mathcal{T}', \chi')} \frac{\tau(\alpha, a)}{1 + \sigma_{\chi'}(a)^{d_H}}}$$

with $a \in (\mathcal{T}', \chi')$ and $\sigma_{\chi'}(a)$

assumed recursively already defined.

$\alpha \in (\mathcal{T}, \chi)$ iff $\sigma_\chi(\alpha) < +\infty$.

$$\left\{\left[[(A \xrightarrow{d_B} B) \times C] \xrightarrow{d_D} D\right] \times [(E \xrightarrow{d_F} F) \times G]\right\} \xrightarrow{d_H} H$$

$\boxed{\alpha : \mathcal{T}' \to A}$, $A$ decidable.

**Definition:** $\alpha$ is polynomial if,

$\boxed{\text{for every}}$ configuration $\chi' = (d_B, d_D, d_F) \in \mathrm{Conf}_{\mathcal{T}'}$,

$\boxed{\text{there exists}}$ $d_H < +\infty$ satisfying:

$$\alpha \in (\mathcal{T}, (\chi', d))$$

**Proposition:** Every composition diagram of polynomial functional objects is a polynomial functional object.

**Proof:** Obvious.

**Example:**

$$
\begin{tikzcd}
H_3(X) \arrow[d, mapsto] & & X_{EH} \arrow[ll, mapsto] \arrow[d, mapsto, bend left] \\
K(H_3(X),1)_{EH} \arrow[r, mapsto] & K(H_3(X),2)_{EH} & Y_{EH}
\end{tikzcd}
$$

5. Particular case of the Kenzo program

for the homology of iterated loop spaces.

Main algorithm: $\rho : \mathcal{SSEH} \to \mathcal{SSEH} : X_{EH} \mapsto (\Omega X)_{EH}$

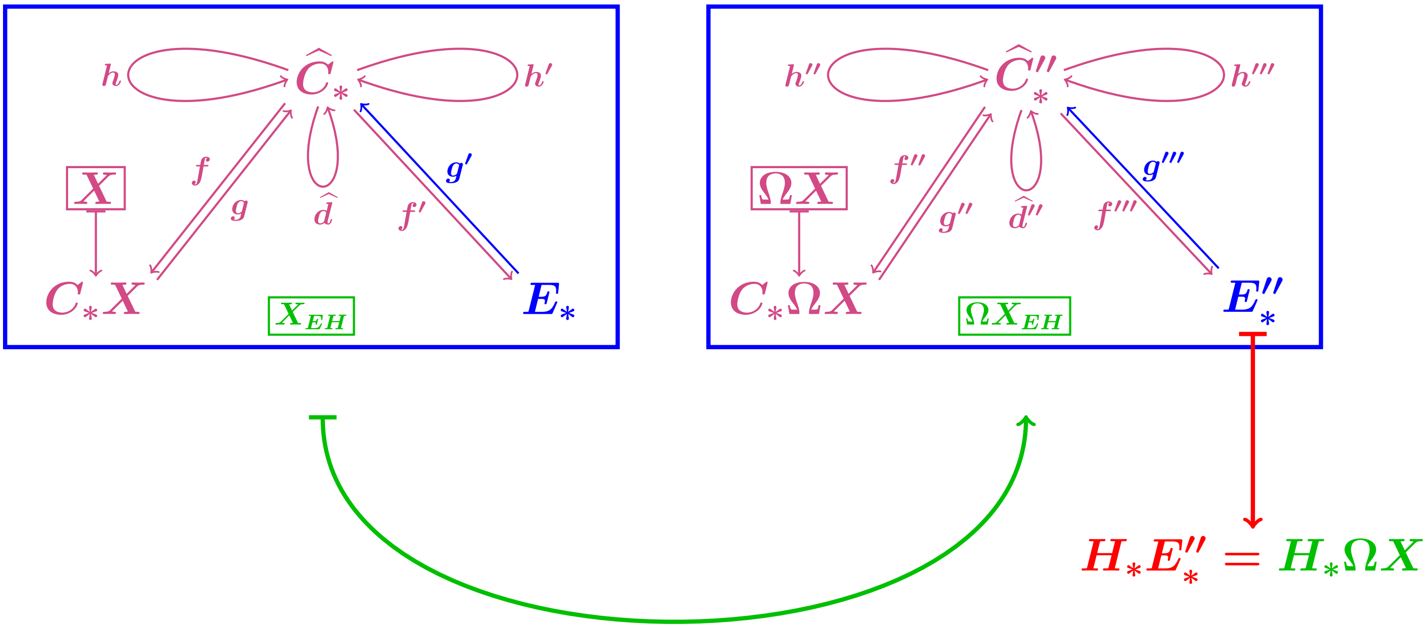
$\mathcal{SSEH}$ := type of Simplicial Sets with Effective Homology ;

$X_{EH}$ = some simplicial set with effective homology ;

$\Omega X$ := Loop space of $X$ := $\mathrm{Cont}(S^1, X)$.

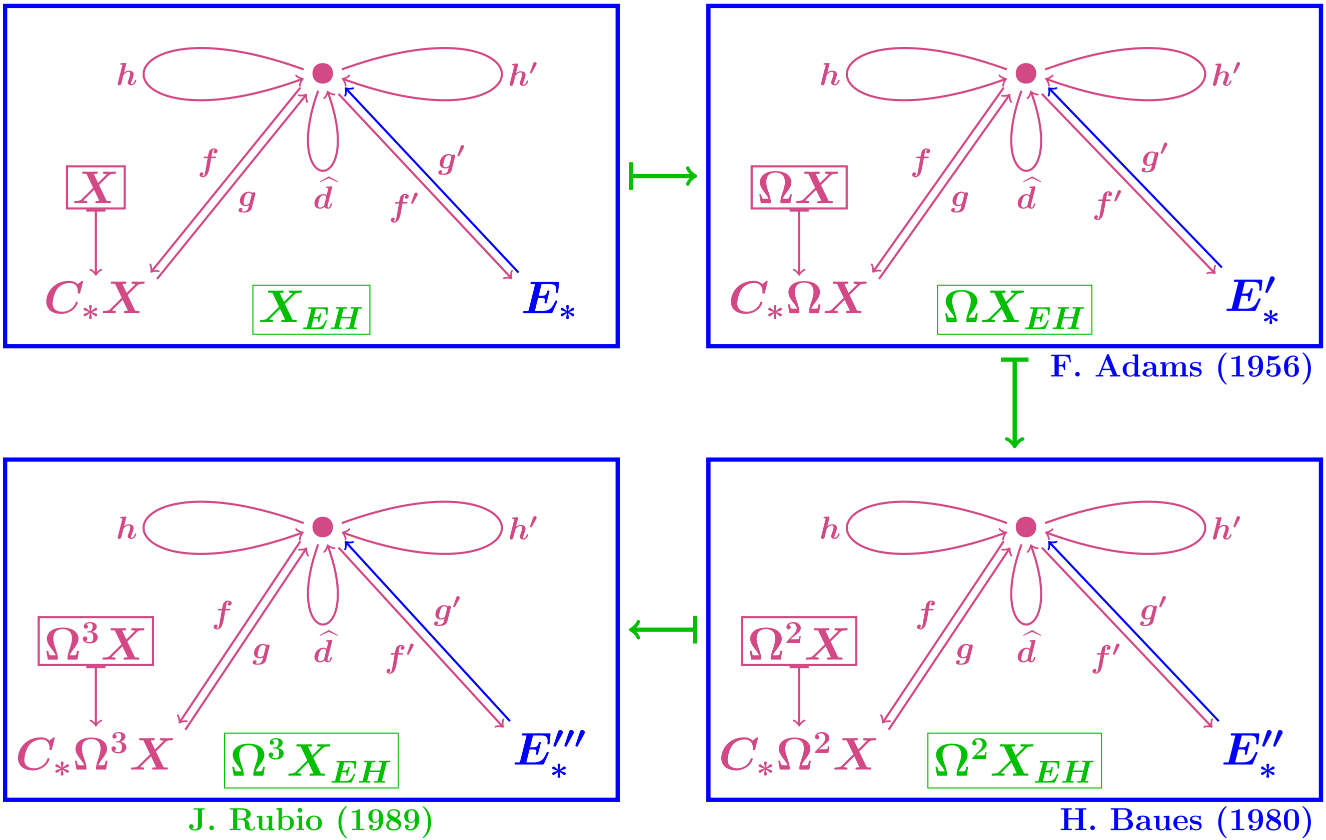
$(\Omega X)_{EH}$ = Version with effective homology of $\Omega X$.

$\Rightarrow$ $\boxed{\text{Trivial iteration !!}}$




$\rho =$ **J. Rubio's algorithm**

$C$


h
h′
f
g
d̂
f′
g′
X
C∗X
X_EH
E∗
ΩX
C∗ΩX
ΩX_EH
E′∗
F. Adams (1956)
Ω²X
C∗Ω²X
Ω²X_EH
E″∗
H. Baues (1980)
Ω³X
C∗Ω³X
Ω³X_EH
E‴∗
J. Rubio (1989)

**<u>Lemma</u>:** $\rho$ is polynomial [up to some fixed dimension].

**<u>Proof</u>:** Exercise.

**<u>Corollary</u>:** $\rho^n$ is polynomial [n fixed].

**<u>Corollary</u>$^2$ :** The computation of $H_p(\Omega^n X)$ is polynomial / $X$.

$$X_{EH} \mapsto (\Omega X)_{EH} \mapsto \cdots \mapsto (\Omega^n X)_{EH} \mapsto H_p \Omega^n X$$

**Typical example of computation :**

$$X = P^{\infty}\mathbb{R}/P^{3}\mathbb{R}$$

$$H_5(\Omega^3 X) = \quad ???$$

**Typical example of computation :**

$$\boldsymbol{X} = \boldsymbol{P}^{\infty}\mathbb{R}/\boldsymbol{P}^{3}\mathbb{R}$$

$$\boldsymbol{H}_5(\boldsymbol{\Omega}^3\boldsymbol{X}) = (\mathbb{Z}/\mathbf{2})^4 \oplus \mathbb{Z}/\mathbf{6} \oplus \mathbb{Z}$$

**Analogous technology for homotopy groups,**

**more complicated.**

**Example:** $\pi_6(\Omega S^3 \cup_2 D^3) = (\mathbb{Z}/2)^5 \oplus \mathbb{Z}$

## Requires:

- $EH$-version of the second Eilenberg-Moore SS (1×)
- $EH$-version of the first Eilenberg-Moore SS (6×)
- $EH$-version of the Serre SS (4×)
- 9 Eilenberg-MacLane spaces with effective homology:

$$K(\mathbb{Z}/2,1)_{EH} \quad K(\mathbb{Z}/2,1)_{EH}$$

$$K(\mathbb{Z}+\mathbb{Z}/4,1)_{EH} \quad K(\mathbb{Z}+\mathbb{Z}/4,2)_{EH} \quad K(\mathbb{Z}+\mathbb{Z}/4,3)_{EH}$$

$$K((\mathbb{Z}/2)^4,1)_{EH} \quad K((\mathbb{Z}/2)^4,1)_{EH} \quad K((\mathbb{Z}/2)^4,1)_{EH} \quad K((\mathbb{Z}/2)^4,1)_{EH}$$

+ 14 days of calculations on a good machine.

**Theorem**: The $EH$-algorithm:

$$(n, X) \longmapsto \pi_n(X)$$

is polynomial with respect to $X$.

**Theorem** (Anick, 1989): The polynomiality of $\pi_n(X)$

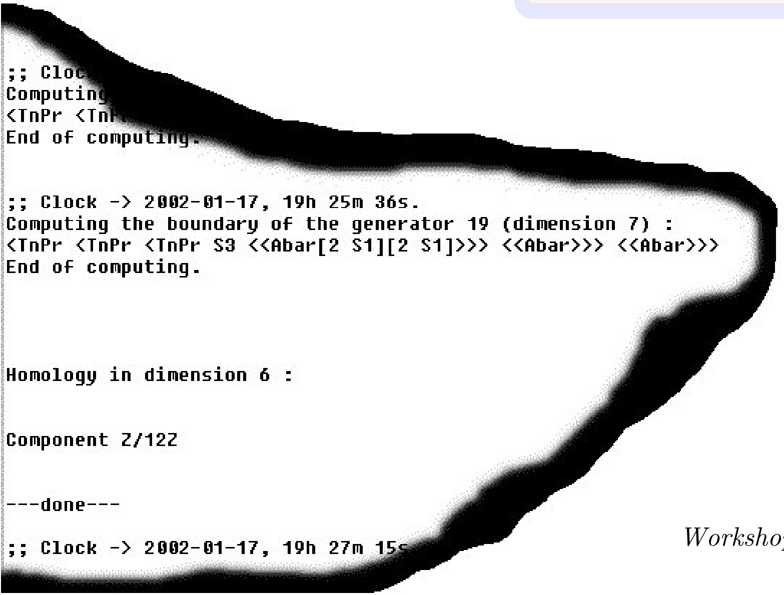
with respect to $n$

is as difficult as $P = NP$.

6. Challenges for Proof Assistants:

- Certified Proof for $H_n(\Omega^p X)$ polynomial / $X$.
- Certified Proof for $\pi_n X$ polynomial / $X$.
- Certified Proof for the $H_n(\Omega^p X)$ computed by Kenzo.
- Certified Proof for the $\pi_n X$ computed by Kenzo.

# The END




*Francis Sergeraert, Institut Fourier, Grenoble*
*Workshop on Algebra, Geometry and Proofs in Symbolic Computations*
*Fields Institute, Toronto, December 2015*