MGS

« encore un Modèle Général de Simulation »

and

Spatial Computing

Jean-Louis Giavitto, Antoine Spicher, Olivier Michel

giavitto@ircam.fr
michel@upec.fr
aspicher@upec.fr

Material

- The MGS home page (new home page for the end of the year)
  http://mgs.spatial-computing.org
- MGS download (new home page for the end of the year)
  http://www.spatial-computing.org/mgs
- This MGS tutorial
  http://www.spatial-computing.org/mgs:tutorial
- MGS publication
  http://mgs.spatial-computing.org/PUBLICATIONS (old)
  http://repmus.ircam.fr/giavitto/publications
  http://www.lacl.fr/~michel/doku.php?id=research:start
  http://www.lacl.fr/~michel/www/bib
  http://www.lacl.fr/~aspicher/publications.html

- The Spatial Computing home page
  http://www.spatial-computing.org

Lecture I (Tuesday, September 4)

A. Motivation & Application domains

1. Modelling Morphogenesis
2. $(DS)^2$: dynamical systems with a dynamical structure

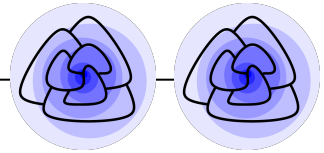
B. MGS

1. Collection
2. Transformation
3. A Functional Language

C. Applications

1. Algorithmic Examples
2. Monoidal Collections
3. GBF : Cellular and Lattice Gaz Automata
4. Delaunay : the growth of a meristem

Lecture II (Wednesday, September 5)



A. Motivation & Application domains

1. Spatial Computing
2. Interaction-Based Computing

B. MGS

1. Abstract Combinatorial Complex
2. Formalization

C. Applications

1. Remeshing
2. Growing a « T »
3. Self-Assembly of Musical Spaces
4. Growing an ontology with the Little Riding Hood
5. Growing an analogy with Paul Ricœur

Lecture III (Thursday, September 6)

B. MGS

1. Stochastic strategies

C. Applications (non physical spaces)

1. Self-Assembly of Musical Spaces
2. Growing an Ontology with the Little Riding Hood
3. Growing an Analogy with Paul Ricœur
4. Programming unconventional models

D. Implementation

1. Generic pattern matching
2. Incremental HOAS implementation of a DSL
3. Beyond algebraic data type : a Generic Advanced API for Containers

E. Current Frontiers

From Global to Local , Multi-level and partial processes, Hybrid Modeling, Time

Implementation

- pattern matching
- evaluation schema
- incremental implementation

A *polytypic* (generic) pattern matching algorithm

How to find the occurrence of a path pattern ?

- `x`
- `p/exp`
- `p,p`
- `p|p`
- `p*`

The notion of derivative of a regular expression

$\partial R/\partial a = \{w \mid aw \in L(R)\}$ (the set of word of R without a leading a)

$[R] = \{\varepsilon\}$ if $\varepsilon \in L(R)$ else $\varnothing$

Provided that R is a regular expression

- $\partial R/\partial a$ is a regular expression
- [R] is a regular expression

Checking word membership

$$\partial R/\partial a = \{ w \mid aw \in L(R) \} = R'$$

gives a new RE

reduce by one the number of elements in the word

iterate recursively until base case ε and then check [R']

Derivation rules

$$\frac{\partial \varepsilon}{\partial \alpha} = \emptyset$$

$$\frac{\partial \emptyset}{\partial \alpha} = \emptyset$$

$$\frac{\partial \alpha}{\partial \alpha} = \varepsilon$$

$$\frac{\partial \beta}{\partial \alpha} = \emptyset \quad \text{si } \alpha \neq \beta$$

$$\frac{\partial r.r'}{\partial \alpha} = \frac{\partial r}{\partial \alpha}.r' + [r].\frac{\partial r'}{\partial \alpha}$$

$$\frac{\partial r + r'}{\partial \alpha} = \frac{\partial r}{\partial \alpha} + \frac{\partial r'}{\partial \alpha}$$

$$\frac{\partial r^*}{\partial \alpha} = \frac{\partial r}{\partial \alpha}.r^*$$

$$\frac{\partial r^+}{\partial \alpha} = \frac{\partial r}{\partial \alpha}.r^*$$

$$\frac{\partial r?}{\partial \alpha} = \frac{\partial r}{\partial \alpha}$$

$$[\varepsilon] = \varepsilon$$

$$[\emptyset] = \emptyset$$

$$[\alpha] = \emptyset$$

$$[r.r'] = [r].[r']$$

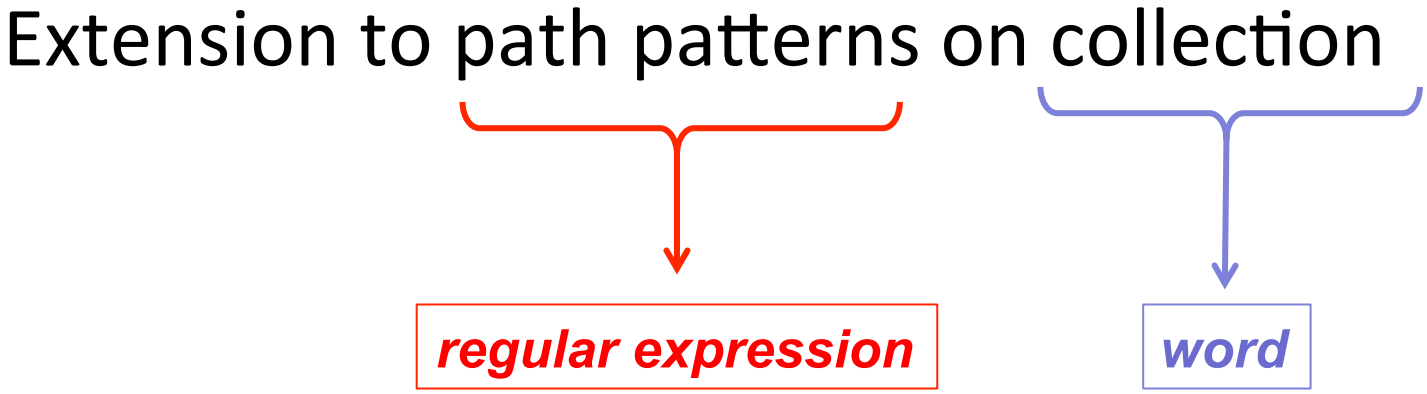
$$[r + r'] = [r] + [r']$$

$$[r^*] = \varepsilon$$

$$[r^+] = [r]$$

$$[r?] = \varepsilon$$

Extension to path patterns



Derivation of a path pattern

$\partial P/\partial \texttt{pos}$(Coll, Env, FreePos)

=

the set of paths in Coll

- begining at position `pos`
- matching the path pattern P
- passing only by elements of FreePos
- and variable binded in Env

The base case

Base case : the empty collection

the only pattern that can match a path in the empty collection must specify a zero-length path, hence

```
x*
```

all other pattern fails and gives an emty solution set

Derivation rules (by combinatorial analysis)

$$\partial(x, P)/\partial \mathtt{pos}(Coll, Env, FreePos)$$
$$= \mathtt{pos}$$
$$\otimes \{\ \partial P/\partial \mathtt{p'}\ (Coll, Env[x \rightarrow \mathtt{pos}], FreePos\text{-}\{\mathtt{p'}\}),$$
$$\mathtt{p'} \in neighbor_{Coll}(\mathtt{pos})\ \}$$

the set of path matching (x, P) and starting at pos is:

pos preppended to " the elements of the set of paths matching P and starting at a position belongings to a neigbhor of pos passing … "

The complete path pattern semantics

$$\frac{\partial \mathrm{dir}*}{\partial p}(G,E,\emptyset) = \{[]\} \quad (1)$$

$$\frac{\partial P}{\partial p}(G,E,\emptyset) = \emptyset \quad \text{provided that } P \neq \mathrm{dir}* \quad (2)$$

$$\frac{\partial \mathrm{id/expr}}{\partial p}(G,E,\Pi) = \texttt{if eval}(E+[\mathrm{id}\to p],G,\mathrm{expr})\ \texttt{then}\ \{[p]\}\ \texttt{else}\ \emptyset \quad (3)$$

$$\frac{\partial \mathrm{dir}*}{\partial p}(G,E,\Pi) = \{[]\} \cup \frac{\partial(\mathrm{id/true\ dir\ dir}*)}{\partial p}(G,E,\Pi) \quad \text{where id is a fresh variable} \quad (4)$$

$$\frac{\partial \mathrm{id/expr\ dir\ P}}{\partial p}(G,E,\Pi) = \texttt{let } E' = E+[\mathrm{id}\to p] \texttt{ and } \Pi' = \Pi - p \quad (5)$$
$$\texttt{in if eval}(E',G,\mathrm{expr})$$
$$\texttt{then } p \otimes \left(\bigcup_{p' \in \texttt{neighbor}(\Pi',\mathrm{dir},p)} \frac{\partial P}{\partial p'}(G,E',\Pi') \right)$$
$$\texttt{else } \emptyset$$

$$\frac{\partial \mathrm{dir}*\ \mathrm{dir}'\ P}{\partial p}(G,E,\Pi) = \bigcup_{p' \in \texttt{neighbor}(\Pi,\mathrm{dir}',p)} \left(\frac{\partial P}{\partial p'}(G,E,\Pi) \right) \quad \text{where id is a fresh variable} \quad (6)$$
$$\cup \quad \frac{\partial(\mathrm{id/true\ dir\ dir}*\ \mathrm{dir}'\ P)}{\partial p}(G,E,\Pi)$$

An HOAS evaluation schema

High Order Abstract Syntax

```
type expr =
| Const of value
| App of expr * expr

and value =
| Int of int
| Bool of bool
| Fun of (value -> value)
```

High Order Abstract Syntax [Pfenning & Elliot PLDI'88]

```
let rec eval = function
| Const v -> v
| App (e1, e2) ->
        (match (eval e1,eval e2) with
        | (Fun f , v) -> f v    (* caml application *)
        | _ -> raise TypeError
        )
```

Two step translations

1. non-strictness elimination
   transforming conditionals and imperative features into functions
2. variable elimination
   transforming λ-abstractions
   into host language functions
   S, **K**, **I** and $\mathbf{B}_{c}$, $\mathbf{C}_{c}$, $\mathbf{N}_{c,c'}$
   and *smart partial application*
   and multiple abstraction/application

Combinatorisation

$$\lambda x.U \rightarrow (\mathsf{K}\ U)$$
$$\lambda x.x \rightarrow \mathsf{I}$$
$$\lambda x.(M\ N) \rightarrow ((\mathsf{S}\ \lambda x.M)\ \lambda x.N)$$
$$\lambda x.c \rightarrow (\mathsf{K}\ c)$$

$$\mathsf{I} = \lambda x.x$$
$$\mathsf{K} = \lambda x.\lambda y.x$$
$$\mathsf{S} = \lambda x.\lambda y.\lambda z.((x\ z)\ (y\ z))$$

$$\lambda x.(U\ M) \rightarrow ((\mathsf{B}\ U)\ \lambda x.M) \quad (d)$$
$$\lambda x.(M\ U) \rightarrow ((\mathsf{C}\ \lambda x.M)\ U) \quad (e)$$

$$\lambda x.(c\ M) \rightarrow ((\mathsf{B}\ c)\ \lambda x.M) \quad (d_v)$$
$$\lambda x.(M\ c) \rightarrow ((\mathsf{C}\ \lambda x.M)\ c) \quad (e_v)$$
$$\lambda x.(c_1\ c_2) \rightarrow ((\mathsf{N}\ c_1)\ c_2) \quad (n_v)$$

$$\mathsf{B} = \lambda a.\lambda g.\lambda x.(a\ (g\ x))$$
$$\mathsf{C} = \lambda f.\lambda b.\lambda x.((f\ x)\ b)$$
$$\mathsf{N} = \lambda a.\lambda b.\lambda x.(a\ b)$$

Combinatorisation

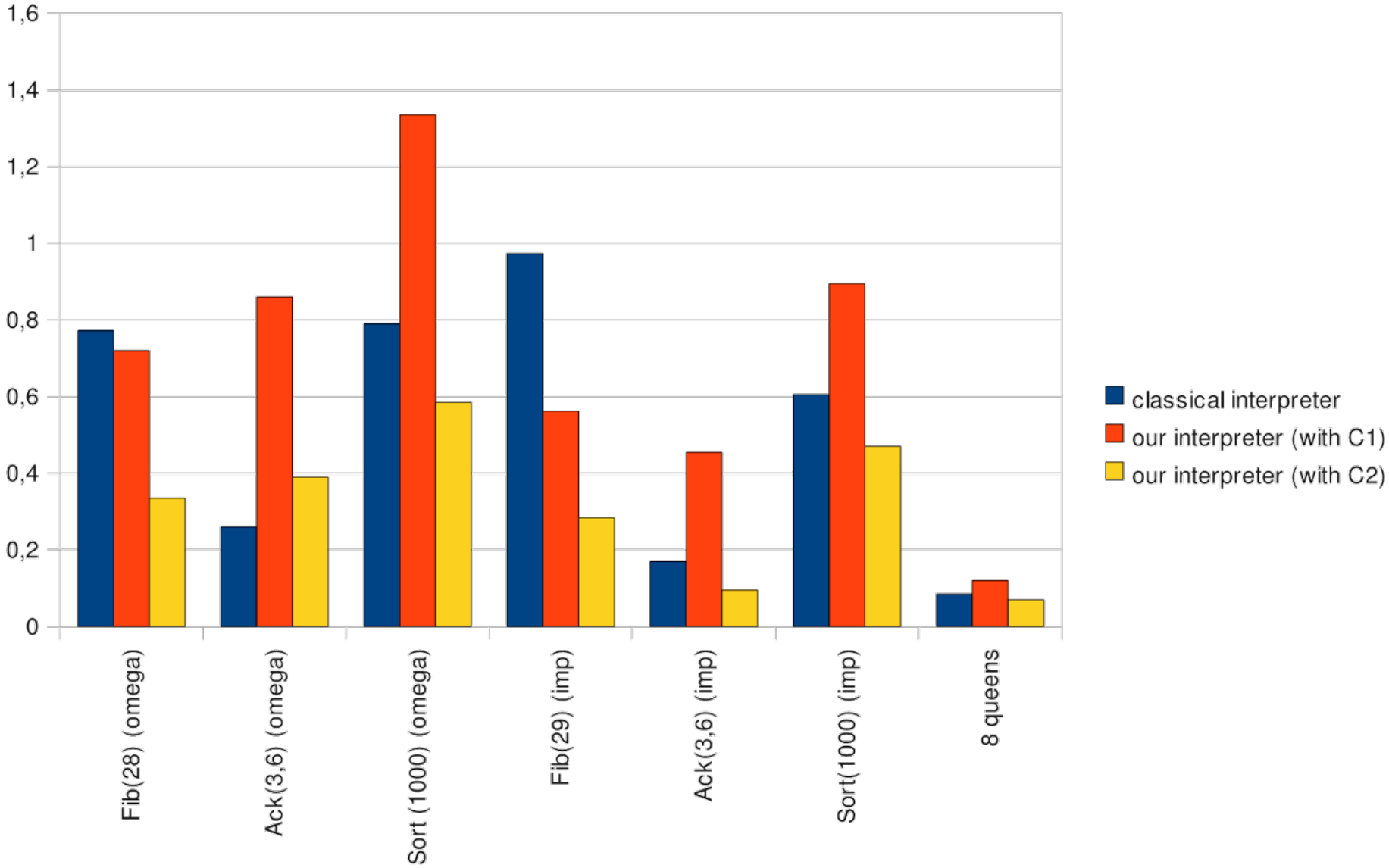
$$\lambda x.\lambda y.(e_1\ e_2) \rightarrow ((\mathsf{S}^2\ \lambda x.\lambda y.e_1)\ \lambda x.\lambda y.e_2) \qquad (s^2)$$
$$\lambda x.(e_1\ e_2\ e_3) \rightarrow (((\mathsf{S}_2\ \lambda x.e_1)\ \lambda x.e_2)\ \lambda x.e_3) \qquad (s_2)$$
$$\lambda x.\lambda y.(e_1\ e_2\ e_3) \rightarrow (((\mathsf{S}_2^2\ \lambda x.\lambda y.e_1)\ \lambda x.\lambda y.e_2)\ \lambda x.\lambda y.e_3 \qquad (s_2^2)$$
$$\lambda x.\lambda y.c \rightarrow (\mathsf{K}^2\ c) \qquad (k^2)$$
$$\lambda x.\lambda y.x \rightarrow \mathsf{I}^2 \qquad (i^2)$$

$$\mathsf{S}^2 = \lambda f.\lambda g.\lambda x.\lambda y.((f\ x\ y)\ (g\ x\ y))$$
$$\mathsf{S}_2 = \lambda f.\lambda g.\lambda h.\lambda x.((f\ x)\ (g\ x)\ (h\ x))$$
$$\mathsf{S}_2^2 = \lambda f.\lambda g.\lambda h.\lambda x.\lambda y.((f\ x\ y)\ (g\ x\ y)\ (h\ x\ y))$$
$$\mathsf{K}^2 = \lambda x.\lambda y.\lambda z.x$$
$$\mathsf{I}^2 = K$$

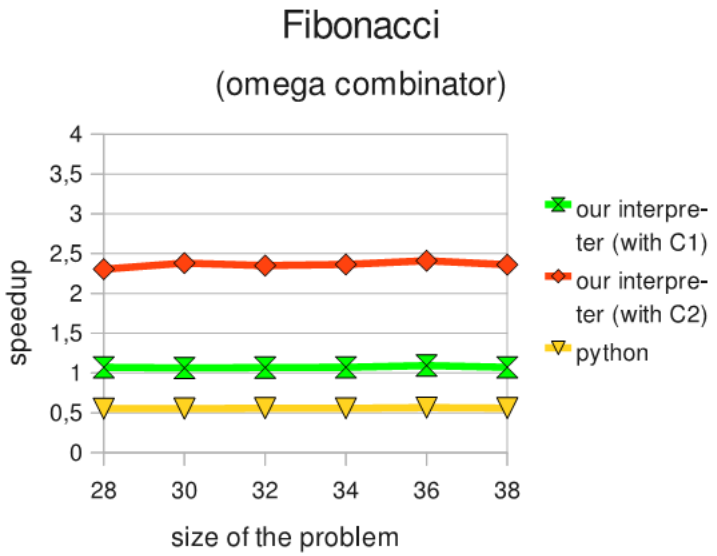
Advantages

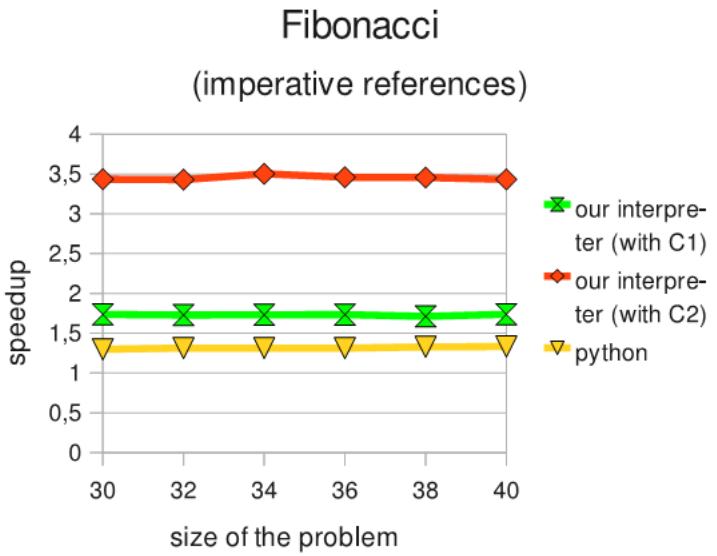
- includes higher-order functions, imperative features and non-strict features
- no difference between user and library functions (**seamless** mixing *including* high-order)
- very small
- host language can be any mainstream languages (done in Java)
- transformations proved correct (in Coq)
- effective…

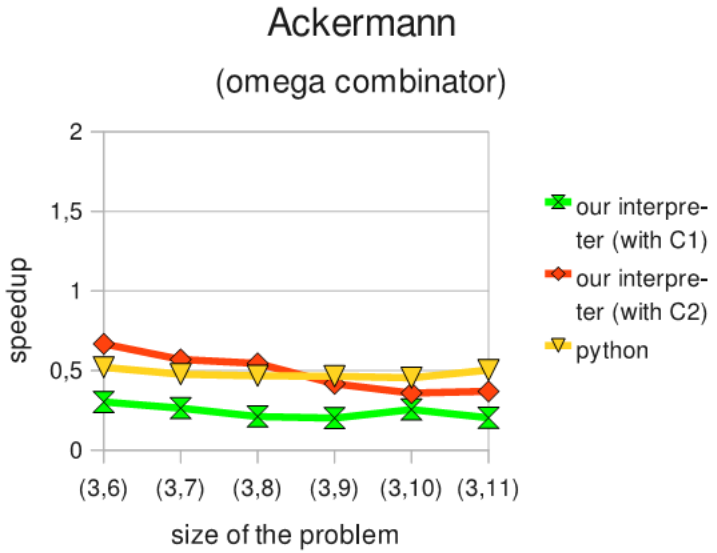
Performances

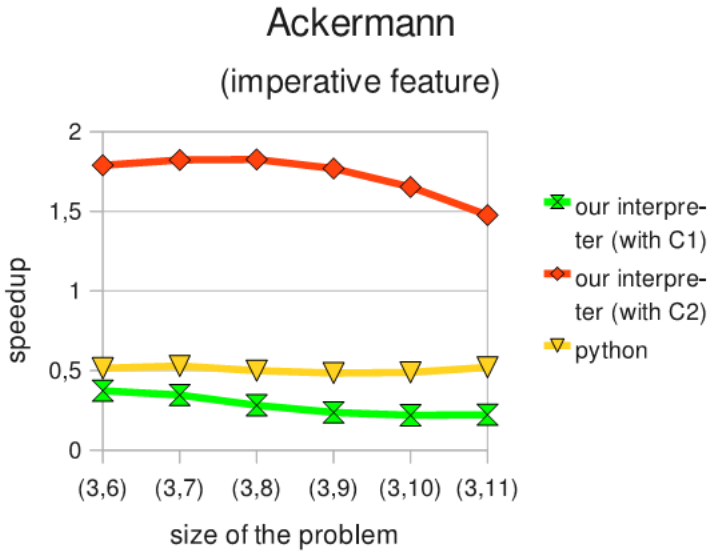


Performances









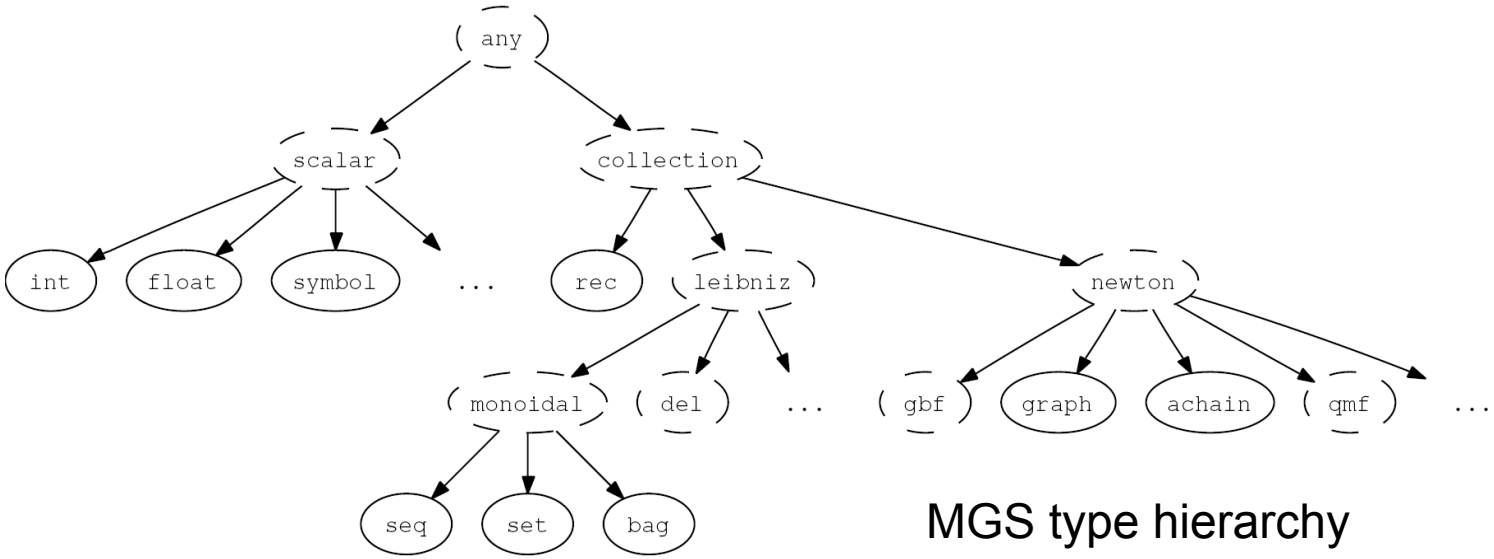
Performances

		Cl.	$\mathcal{C}_1$	$\mathcal{C}_2$	Python	Erlang
Fibonacci (omega)	28	0,77	0,72	0.34	1.39	341
Fibonacci (omega)	38	94	87,67	39,82	167,94	
Fibonacci (imp)	29	0.97	0.56	0.28	0,76	
Fibonacci (imp)	40	195,0	112,2	56,8	146,3	
Ackermann (omega)	(3,6)	0.34	0.86	0.39	0.50	146
Ackermann (omega)	(3,11)	357.9	1744	967	711	
Ackermann (imp)	(3,6)	0.17	0.46	0.10	0.33	
Ackermann (imp)	(3,11)	254	1139	172	488.0	
sort (omega)	1000	0,79	1.34	0.59	/	
sort (omega)	64 000	5603		14046	/	
sort (imp)	1000	0.61	0.90	0.47	/	
sort (imp)	64 000	4516	13570	10381	/	
queens problem (imp)	8	761	1105	563	/	

An incremental development

Interpreter Development

- In 2000: first version written in 1 year
  - 3 topological collection types
  - 9k lines of Ocaml, C and C++ code
- Now: third version written on a 3 years period
  - >12 topological collection types (and 12 scalar types) ⇒ **>24 data types**
  - 50k lines of Ocaml, and many libraries (> 100k lines)
  - Scattered over >100 files
  - 225 overloaded MGS functions



MGS type hierarchy

Incremental Development ?

Our goals

1. Allowing new target data structures to be added without modifying the already written implementation files of the interpreter,
2. Adding easily new target data structures and functions
3. Lifting Ocaml functions in MGS
4. Embedding MGS functions in Ocaml
5. Automating the definition of highly overloaded MGS functions
6. *Don't modify* the implementation language (Ocaml)
7. Do thing as much as possible statically

Our solution

1+2 ⇒ *AOP-like* approach

3+4 ⇒ Higher Order Abstract Syntax

5 ⇒ *Automagically* produce the dispatch code

6 ⇒ A simple external tool and simple syntactic conventions

7 ⇒ at compile-time

The Expression Problem or

How to Implement the Value *Data-Type*

- Goes back to [Wadler 98]

> To: **`java-genericity@...`**
>
> The Expression Problem
> Philip Wadler, 12 November 1998
>
> The Expression Problem is a new name for an old problem. The goal is to define a datatype by cases, where one can add new cases to the datatype and new functions over the datatype, without recompiling existing code, and while retaining static type safety (e.g., no casts). […]

The Expression Problem or

How to Implement the Value *Data-Type*

- Goes back to [Wadler 98]

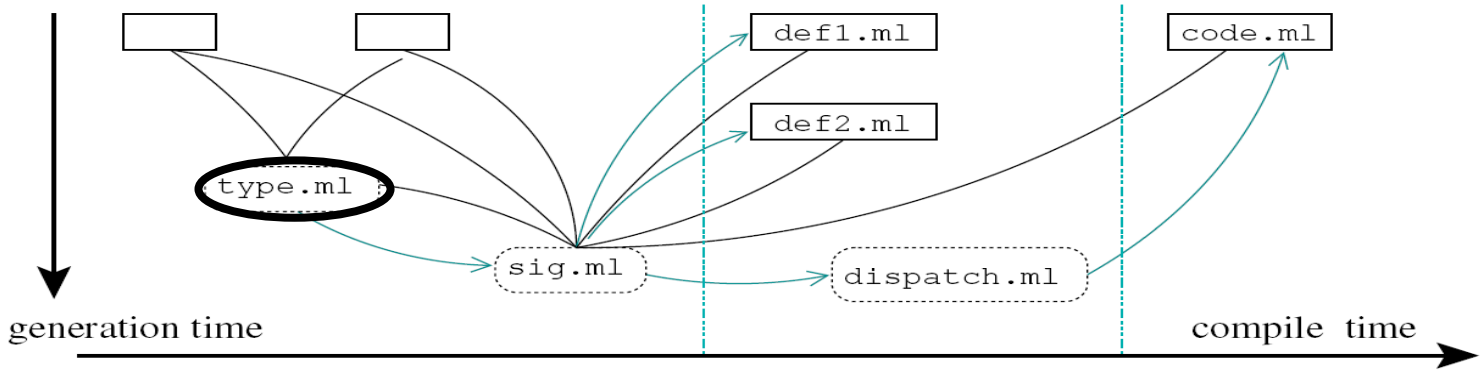
- A sum type
  - Painful incremental addition of new MGS collection types
  - Easy addition of new operators
  - Compatible with HOAS

- A product type
  - Easy incremental addition of new MGS collection types
  - Painful addition of new operators

An « aspect oriented » approach

- Language-level integrated approach
- Very light implementation
  - No sophisticated type/module/language strategy
  - Efficient
    - Minimal overhead at compile-time (files scanned instantaneously)
    - No run-time (except dispatch)

(1) Gathering all the cases of the sum type

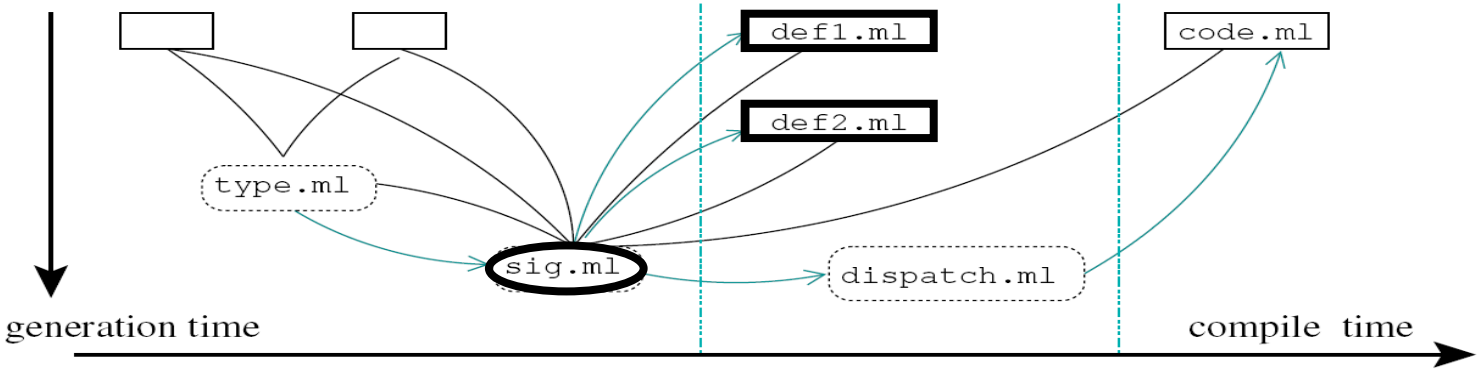


Access *all* **`.ml`** and **`.mli`** file to produce the **`value`** type

```
type value =
| Int of int
| Float of float
| …
```

types.ml

(2) Gathering all the cases of the overloaded functions



Access *all* .ml files and produce the signature file and set the forward mechanism

Note the use of the

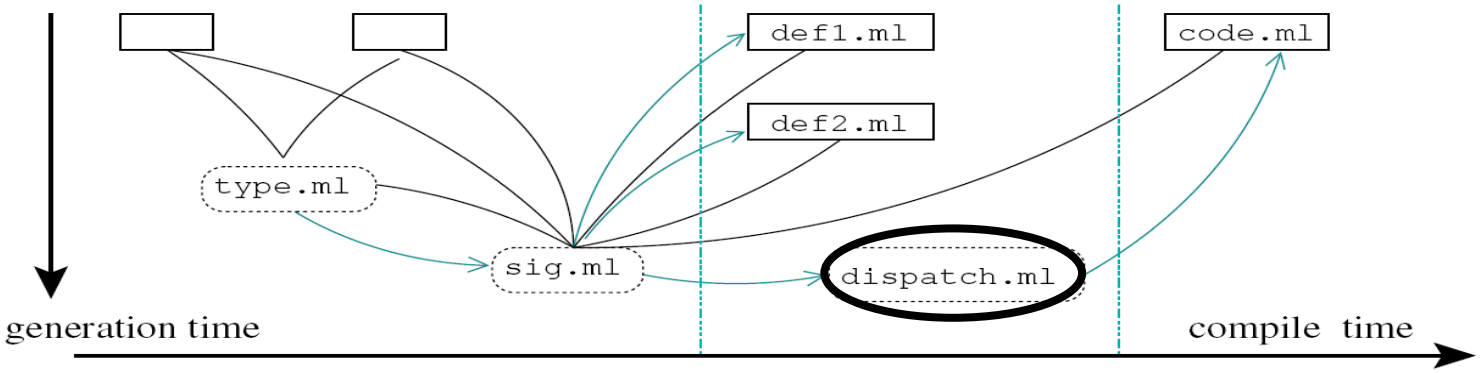
```
open Types;;

(* Signature declaration *)
let (add_forward : (value -> value -> value) ref) =
ref (function _ -> failwith "unitialized add")

(* Setting foward pointer *)
let add x y = !add_forward x y
```

def1.ml

def2.ml

sig.ml (generated)

(3) Generate the Overloaded Functions

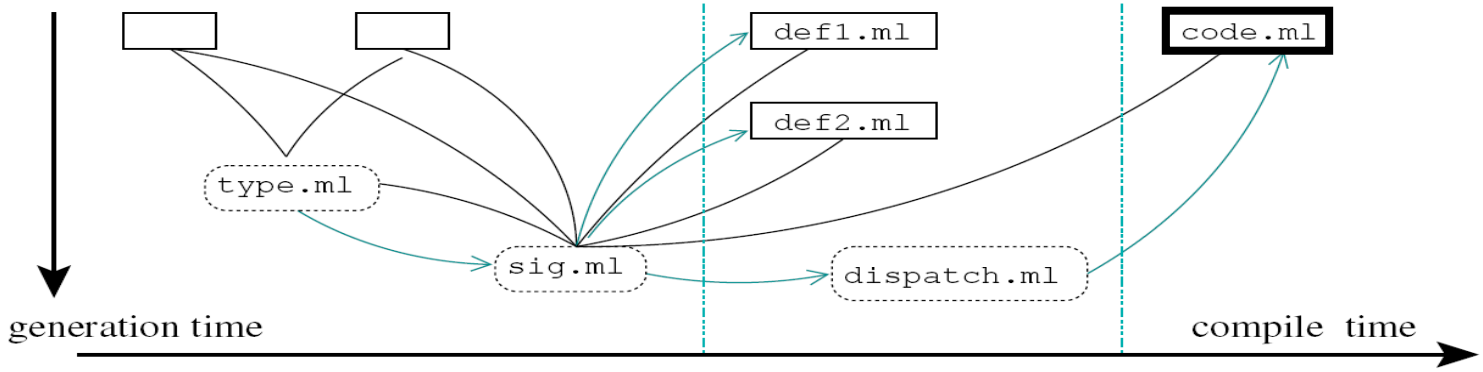


Computing the correct cases order and setting the correct link

```
open Types, Sig, Def1, Def2

let __add x y = match x, y with
| (Int x0), (Int x1)     -> _add_int_int x0 x1
| (Float x0), (Float x1) -> _add_float_float x0 x1
| (Int x0), (Float x1)   -> _add_int_float x0 x1
| (Float x0), (Int x1)   -> _add_float_int x0 x1
…
(* Setting the correct links *)

Sig.add_forward := __add
```

dispatch.ml (generated)

(4) Using the Dispatched Functions

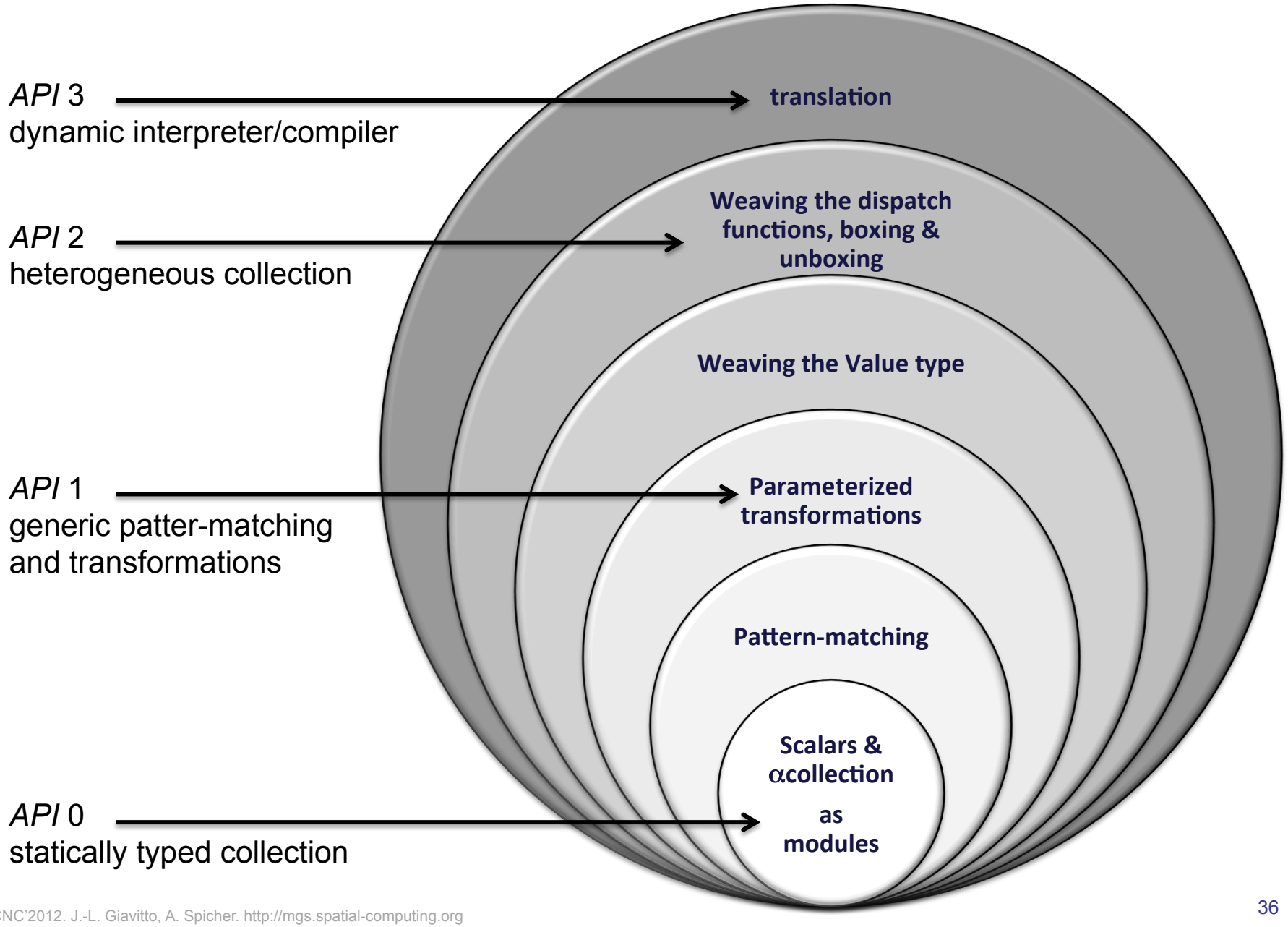


Voila!

```
open Types, Sig

print (add (Float 2.0) (Float 3.0))
print (add (Float 2.0) (Int 1))
print (add (Int 2) (Float 1.0))
print (add (Int 2) (Int 1))
```

using.ml

Towards a new evaluation/development architecture



Perspectives

Success

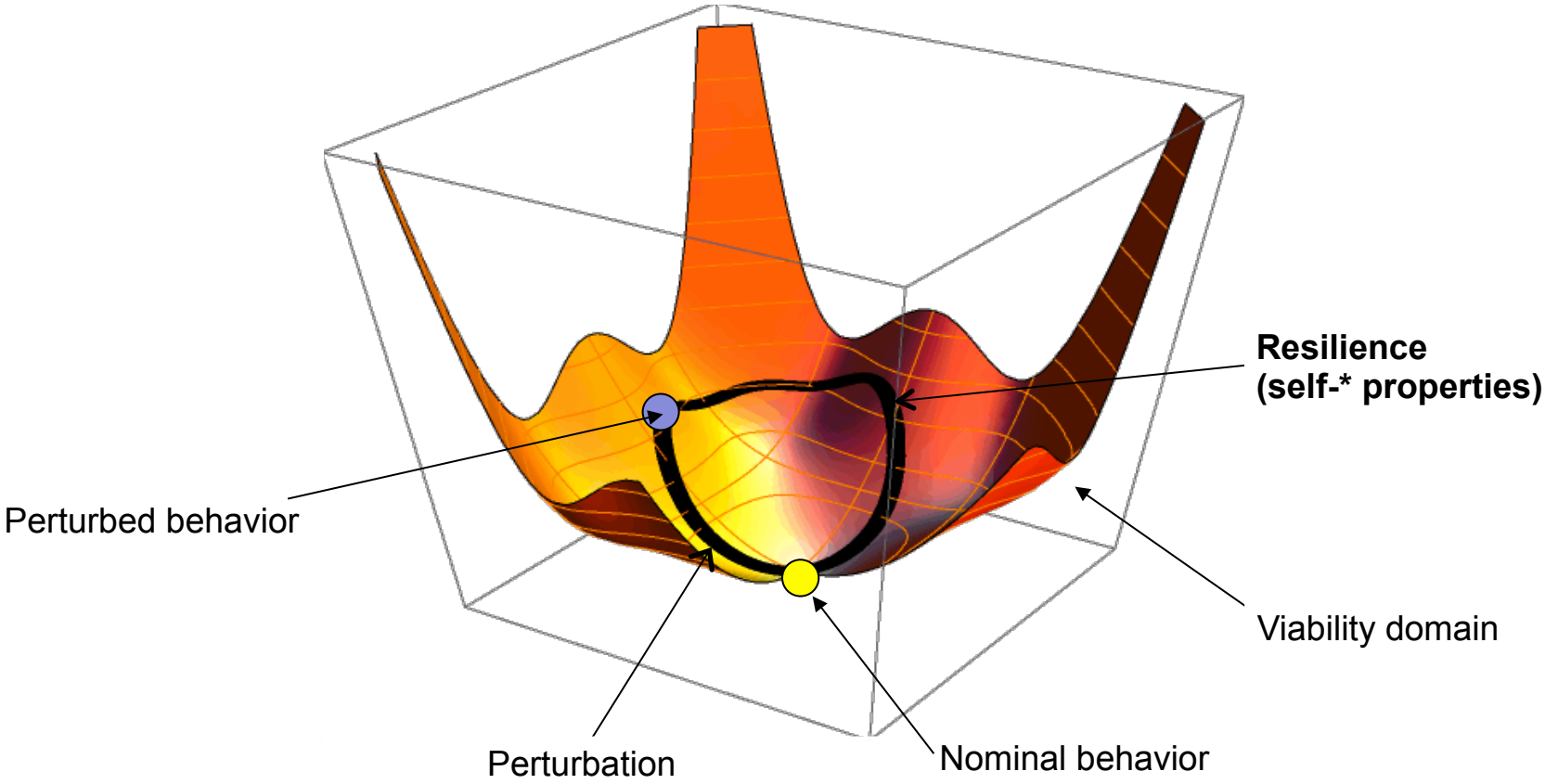
- Polytypisme is good
- Patterns/rules are expressive and usually concise
- Clean semantics
- Find the “good” topology for the problem at hand

To do

- Integration into a standard programming language (aspect, *à la* TOM,…)
- A logic for topological rewriting
- Efficiency: compilation *well…*
- ***Space-time !***
  - Amorphous/autonomic/spatial computing applications (including synthetic biology)
  - Musicology

Autonomic systems: *systemic software engineering?*

- Systems with self-* properties (self-healing, self-optimizing, …)
- Analogy with dynamical complex systems
  - Running programme: *dynamical* system
  - Nominal behavior: *equilibrium* state
  - Self-* properties: going *from* a *perturbation to* the *equilibrium*



Thanks

http://mgs.spatial-computing.org

- **Julien Cohen**
- **Antoine Spicher**
- **Olivier Michel**
- **PhD and other students**

  Julien, Antoine, P. Barbier de Reuille,

  E. Delsinne, V. Larue, F. Letierce, B. Calvez,

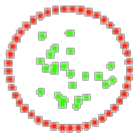
  F. Thonerieux, D. Boussié *and the others...*

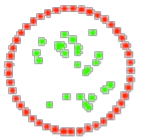
Julien

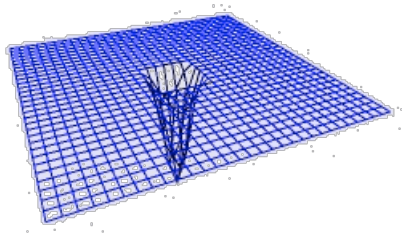
Antoine

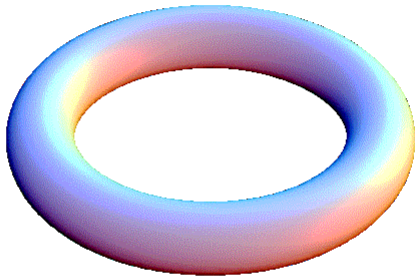
Olivier

- **Collaborations**
  - P. Fradet (INRIA Grenoble, programmation)
  - A. Lesne (IHES, stochastic simulation)
  - P. Prusinkiewicz (Calgary, declarative modeling)
  - P. Barbier de Reuille (meristeme model)
  - D. Pumain (Paris 1, géography)
  - René Doursat (ISC, morphogenetic engineering)
  - Moreno Andreatta (IRCAM, formalisation spatiale de la musique)
  - H. Berry (LRI, stochastic simulation)
  - C. Godin (CIRAD, biological modeling)
  - J.-P. Banâtre (IRISA, programming)
  - F. Gruau (U. PXI, language and hardware)
  - G. Malcolm (Liverpool, rewriting)
  - F. Delaplace (IBISC, synthetic biology)
  - P. Liehnard (Poitier, CAD, Gmap and quasi-manifold)









to be continued…

Acknowledgements

MGS: Antoine Spicher, Olivier Michel, Julien Cohen

S&S Bio : Hanna Klaudel, Franck Delaplace, Hugues Berry, Przemek Prusinkiewicz, Annick Lesne…

Spatial Computing: Jacob Beal, Fréderic Gruau, René Doursat…

Examples: Pierre Barbier de Reuille, Christophe Godin, Samuel Bottani, the Paris iGEM'07 team…

Some figures are borowed from Olivier Michel, Antoine Spicher, Pierre Barbier de Reuille, Franck Delaplace, Hugues Berry (INRIA), the iGEM Paris 2007 and many others.