




# Cellular Meta-Programming

**Gabriel Ciobanu**

Romanian Academy, Institute of Computer Science, Iași

`gabriel@iit.tuiasi.ro`



# Outline



- What can we learn from the cell (behaviour)?
- On Meta-Programming (structural, behavioural)
- Briefly on P Systems
- More about Maude
- Reflection in Rewriting Logic
- Cellular Meta-Programming over Membranes
- Membranes Specification
- Example, Software Experiments
- Conclusion



# What can we learn from the cell?



- the distribution of the tasks among the cell compartments
- the high adaptability and flexibility of the cell behaviour

The adaptability of cells to the changing environment requires sophisticated processing mediated by interacting genes and proteins. A cell is able to adapt its execution according to various developmental and environmental stimuli, causing corresponding changes in its behaviour. We refer to this adaptability in terms of meta-programming.

We try to put together meta-programming and cell compartments in a formal computational framework, describing

adaptable executions of membranes specifications.



# On Meta-Programming



- adaptable executions are generated in computer science by meta-programming;
- meta-programming is the act of writing programs able to manipulate themselves as their data, allowing execution modification;
- reification and reflection;
- reification is the mechanism of encoding execution states as data; at run-time a program is used as a representation and made available (to the program) as ordinary data.
- cell is able to modify its activity and to change its own code at run-time → this behaviour is more than reification.



# Software Reflection



- in programming, reflection is defined as the ability of a program to manipulate the encoding of the state of the program during its own execution;
- the mechanisms are both structural and behavioural;
- structural reflection is the ability to work with the structures and processes of a programming system within the programming system itself (easier to implement - Lisp, Smalltalk, Java have structural reflection mechanisms);
- the behavioural reflection allows a program to modify, even at run-time, its own code;
- cell adaptability is close to the behaviour reflection.



# Reflection over specifications



- We provide executable specifications in Maude of the abstract model of the cell compartments (P systems);
- Maude is a rather mathematical language, and a software system supporting reflection; it has meta-logical axioms for reflection, as well as for computational strategies in rewriting logic;
- Membrane systems represent a new formal model of parallel and distributed computing inspired by cell compartments; the membranes determine regions where objects and evolution rules can be placed; the objects evolve according to the rules of each region, and the regions cooperate in order to maintain the proper behaviour of the whole system.



# Briefly on P Systems

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_0)$$

- (i)  $O$  is an alphabet of **objects**;
- (ii)  $\mu$  is a **membrane structure** consisting of labelled membranes;
- (iii)  $w_i$  are multisets over  $O$  associated with the **regions** defined by  $\mu$ ;
- (iv)  $R_i$  are finite sets of **evolution rules** of the form  $ab \rightarrow a(c, in_2)(c, out)$ ;
- (v)  $i_0$  specifies the **output membrane** of  $\Pi$  or the outer region.



# Membranes

**Elementary membrane**  $M = (R_M, w_M)$

A computation step transition is defined as a rewriting rule

$$\frac{x_1 \rightarrow y_1, \dots, x_n \rightarrow y_n \in R_M, z \text{ is } R_M\text{-irreducible}}{x_1 \dots x_n \Rightarrow y_1 \dots y_n z} \quad (1)$$

$z$  is  $R_M$ -irreducible whenever there does not exist rules in  $R_M$  applicable to  $z$

**Composite membrane**  $(M_1, \dots, M_k, R_M, init)$ , where  $M_i (1 \leq i \leq k)$  is an elementary or composite membrane, and  $init$  is its initial multiset of form  $(w, (w_1, \dots, w_k))$ .

$$\frac{w \Rightarrow w', w_1 \Rightarrow w'_1, \dots, w_n \Rightarrow w'_n}{(w, (w_1, \dots, w_k)) \Rightarrow (w', (w'_1, \dots, w'_k))} \quad (2)$$





# Membranes Computation



- objects of the membranes are the subject of local evolution rules that evolve simultaneously;
- a sequence of computation steps represents a computation; a computation is successful if this sequence is finite, namely there is no rule applicable to the objects present in the last configuration;
- in a final configuration, the **result** of a successful computation is the total number of objects present in the skin membrane.



# More about Maude



- Maude is a **specification language** with a strong mathematical foundation (OBJ family)
- There is a software system able to execute Maude specifications <http://maude.cs.uiuc.edu>
- Basic programming statements: **equations**, **membership assertions**, and **rewriting rules**.
- **Functional module**: only equations and membership assertions
- **System module** = Functional module + (multiset) rewriting rules specifying local transitions in a possibly concurrent system  
(there is no assumption that all rewriting sequences will lead to the same final result, and for some systems there may not be any final states).



# Specification of the P Systems



- each P system  $\Pi$  is represented as a collection of Maude modules such that each membrane is represented by a corresponding Maude system module; sort `Obj` is for object names, and its subsort `Output` is for results; a sort `Soup` is for the multisets of objects, and a sort `Config` for the states of a P system;
- an expression of the form  $\langle M \mid S \rangle$  represents a configuration corresponding to an elementary membrane  $M$  with its multiset  $S$ , and a configuration  $\langle M \mid S; C_1, \dots, C_n \rangle$  corresponds to a composite membrane  $M$  in state  $S$  and with the component  $i$  having the configuration  $C_i$ .



# Objects and Configurations in Maude

```
fmod OBJ is
  sorts Obj Output .
  subsort Output < Obj .
  ops a b c d : -> Obj .
  mb b : Output .
  mb c : Output .
endfm
```

```
fmod CONFIG is
  inc OBJ .
  inc QID .
  sorts Soup Config .
  subsort Obj < Soup .
  op empty : -> Soup .
  op ___ : Soup Soup -> Soup [assoc comm id: empty] .
  op <_|_> : Qid Soup -> Config .
  op <_|_i_> : Qid Soup Config -> Config .
  op _',_ : Config Config -> Config [assoc comm] .
endfm
```

# Reflection in Rewriting Logic

Rewriting logic is **reflective**:  
there is a **universal rewriting specification**  $\mathcal{U}$  such that

$$M \vdash t \rightarrow t' \text{ iff } \mathcal{U} \vdash \langle \overline{M}, \overline{t} \rangle \rightarrow \langle \overline{M}, \overline{t'} \rangle,$$

A **reflective tower**:

$$M \vdash t \rightarrow t' \text{ iff } \mathcal{U} \vdash \langle \overline{M}, \overline{t} \rangle \rightarrow \langle \overline{M}, \overline{t'} \rangle \text{ iff } \mathcal{U} \vdash \langle \overline{\mathcal{U}}, \overline{\langle \overline{M}, \overline{t} \rangle} \rangle \rightarrow \langle \overline{\mathcal{U}}, \overline{\langle \overline{M}, \overline{t'} \rangle} \rangle \dots$$

This concept is supported in Maude through a built-in module called **META-LEVEL**.

This module has functions like `metaReduce( $\overline{SP}, \overline{t}$ )` returning the representation of the reduced form of a term  $t$  using the equations in the module  $SP$ .





# Evaluation Strategies and META-LEVEL

- Evaluation strategies control the positions in which equations can be applied, giving the user the possibility of indicating which arguments to evaluate before simplifying a given operator (using the equations);
- Reflection allows a complete control of the rewriting (execution) using the rewriting rules in the theory; reflective computations allow the link between meta-level and the object level, whenever possible;
- META-LEVEL module can be extended by the user to specify strategies of controlling the rewriting process; we use META-LEVEL in order to provide a clear (algorithmic) description of the “maximal parallel rewriting” strategy given by `maxParRew`.



# Cellular Meta-Programming



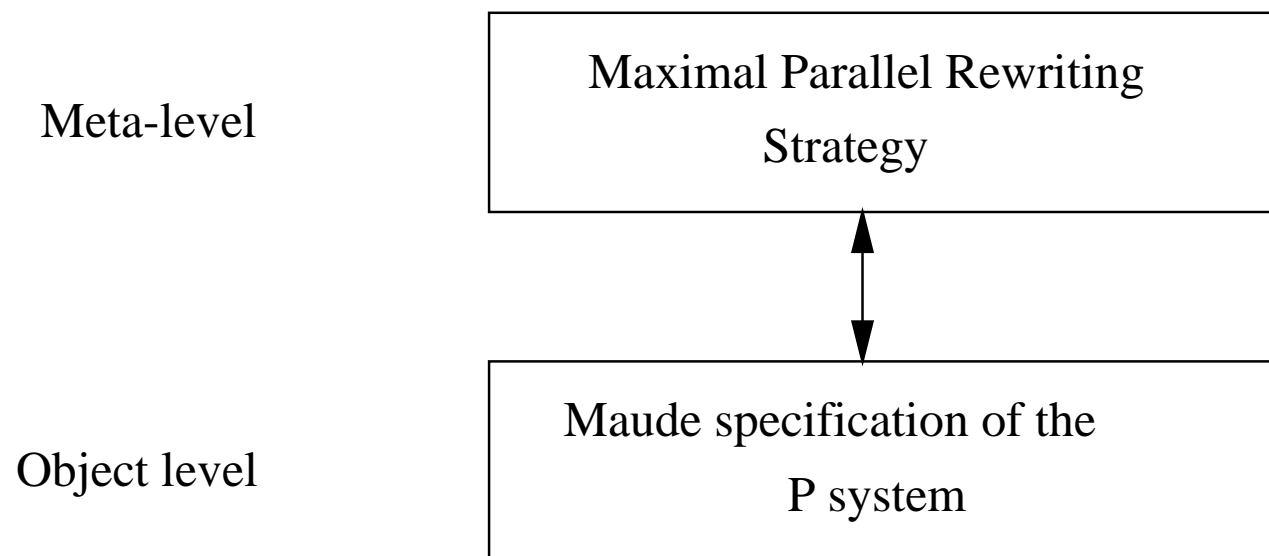
- using `maxParRew` as a transition step between meta-level configurations, we then provide an operational semantics of the membrane systems;
- using the power given by the reflection tower in Maude, we define operations over modules and strategies to guide the deduction process;
- finally we can use a meta-metalevel to analyze and verify the properties of the membrane systems.

This description of the membrane systems based on capabilities given by reflection is called **cellular meta-programming**; it could become a useful paradigm for further investigations (in system biology and UPP).



# Maximal Parallel Rewriting

Maude semantics of a module  $M$  is not the same with the  $P$  system semantics  $\rightarrow$  we must associate with  $M$  the new semantics based on the maximal parallel rewrite relation. We use `META-LEVEL` to define the “maximal parallel rewriting” strategy, defining the  $P$  system semantics at the meta-level.





# P System Semantics

For the elementary membranes, a computation step is defined as

$$\frac{S \Rightarrow S'}{\langle M \mid S \rangle \Rightarrow \langle M \mid S' \rangle} \quad (3)$$

where  $S \Rightarrow S'$  is defined in (1).  $S \Rightarrow S'$  is not the ordinary rewriting defined by  $M$ , but they are related:

$$S \Rightarrow S' \text{ iff } S \xrightarrow{+}_{R_M} S' \text{ s.t. } \mathit{maxParCons}(R_M, S, S')$$

where  $\xrightarrow{+}_{R_M}$  is the ordinary rewriting defined by  $R_M$ , and  $\mathit{maxParCons}(R_M, S, S')$  represents the constraints defining the maximal parallel rewriting strategy over  $R_M$ .

# P System Semantics

1. if  $S = S'$ , then  $\text{maxParCons}(R_M, S, S)$  holds iff  $S$  is  $R_M$ -irreducible;
2. if  $S \neq S'$ , then  $\text{maxParCons}(R_M, S, S')$  holds iff there exists  $S_1, S'_1, \ell \rightarrow r \in R_M$  such that  $S = \ell S_1$ ,  $S' = r S'_1$ , and  $\text{maxParCons}(R_M, S_1, S'_1)$ .

Since  $\text{maxParCons}$  has the set of rules of the module  $M$  as parameter, it follows that it can be decided **only at the meta-level**.

The transition between configurations for composite membrane is defined as:

$$\frac{S \Rightarrow S', C_1 \Rightarrow C'_1, \dots, C_k \Rightarrow C'_k}{\langle M \mid S; C_1, \dots, C_k \rangle \Rightarrow \langle M \mid S'; C'_1, \dots, C'_k \rangle} \quad (4)$$

# Dynamic Adaptive Processes



- the maximal parallel rewriting is a rewriting strategy depending on the topology + rules of the P system
- it is an **dynamic adaptive process**,
- and it can be defined only at meta-level;
- Maude is reflective, so the meta-levels are inside the system and allow dynamic strategies
- therefore Maude is an appropriate meta-programming platform for implementing the membrane systems.



# Example, Software Experiments



We consider a simple example of membrane system, and then describe and execute its Maude specification. We consider a P system generating symbols  $b$  and  $c$  with the properties that the number of  $c$ 's is double of the number of  $b$ 's, and the total number of  $b$ 's and  $c$ 's is a multiple of 6.

We present only some important steps of the specification, and the results of executing this specification using Maude.



# Example, Software Experiments

```
(mod SKIN is
  inc CONFIG .
  op init : -> Soup .
  eq init = a a .
  rl ['SKIN] : a => a b c c .
  rl ['SKIN] : a a => empty .
endm)
```

The structure of the system is specified in the initial configuration. The module describing  $\Pi_1$  is:

```
(mod PSYS is
  inc SKIN .
  op initConf : -> Config .
  eq initConf = < 'SKIN | init > .
endm)
```





# Software Experiments

The `rew` command with a limited number of steps, is not useful because the number of rewriting steps in Maude is not the same with the number of computation steps of P systems. Therefore, the rewriting process is restricted by the configuration size:

```
cr1 rwf(X) => rwf(maxParRew(X))
           if (X :: Term) /\ (#(X) < 20) .
cr1 rwf(X) => X if #(X) >= 20 .
```







# Software Experiments

The command `down` and the function `up` are used to move between two successive levels of the reflection tower. For instance, `down COMPS` : interprets the result returned by `red` in the module `COMPS`. The function call `up(PSYS, initConf)` returns the representation at the meta-level of the term `initConf` defined in `PSYS`. If we wish to investigate the properties of the result, then we may proceed as follows:

```
(mod PROOF is
  inc METACOMPS .
  op ql : -> QidList .
  eq ql = out(...getTerm(metaRewrite(up(COMPS),
    up(COMPS, getTerm(metaReduce(up(PSYS), up(PSYS, initConf))))), 100)))
endm)
```



# Software Experiments

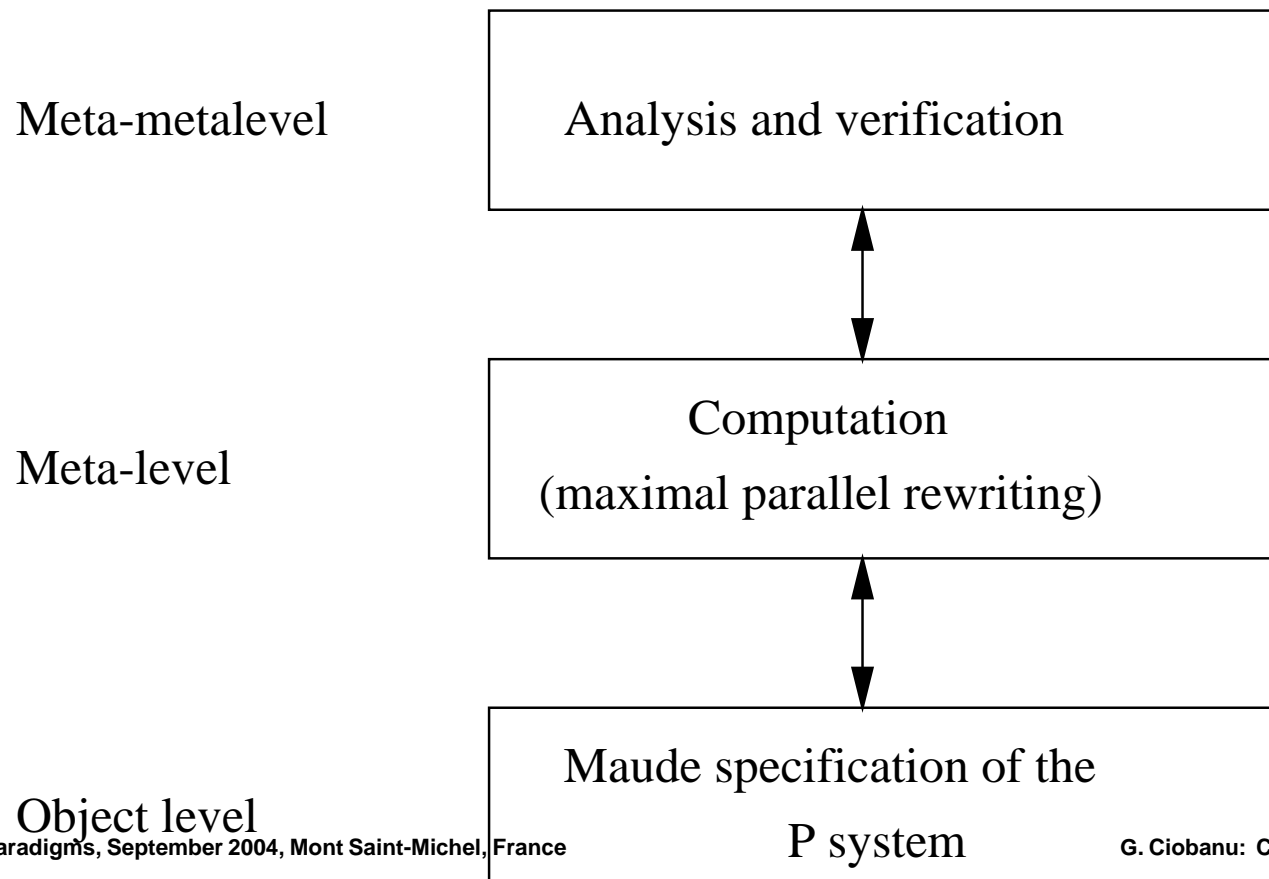
We can check now that the number of objects  $c$  is double of the number of objects  $b$ , and the total number of  $b$ 's and  $c$ 's is a multiple of 6:

```
Maude> (red #(q1, ''c) == 2 * #(q1, ''b) .)
rewrites: 322 in 230ms cpu (230ms real) (1400 rewrites/second)
reduce in PROOF :
  #(q1, ''c) == 2 * #(q1, ''b)
result Bool :
  true
Maude> (red ((#(q1, ''c) + #(q1, ''b)) rem 6) .)
rewrites: 326 in 10ms cpu (10ms real) (32600 rewrites/second)
reduce in PROOF :
  ((#(q1, ''c) + #(q1, ''b)) rem 6)
result Zero :
  0
```



# Analysis and Verification

Maude has a collection of formal tools supporting different forms of logical reasoning to verify program properties, including a model checker to verify temporal properties of finite-state system modules.



Object level

Maude specification of the  
P system

# Conclusion



- We provide executable specifications of P systems.
- Starting from the cells ability to react and change their behaviour at run-time, we translate this adaptability in a meta-programming feature of these executable specifications.
- Cellular meta-programming of the membrane systems and a reflective specification language based on rewriting.
- The approach exploits the reflection property of the rewriting logic.
- Availability of an automated verification tool for the P systems.



# Research Directions



At least two possible further research lines:

1. The mechanism of reflection could describe the biological entities ability to react and change their behaviour according to various developmental and environmental stimuli.
2. Further research will investigate how to integrate directly the P systems aspects with the meta-programming paradigm, trying to harmonize a nice theory and a powerful software technique.

