

Bacterial gene expression in (yet) a(nother) π -calculus

Céline Kuttler

Interdisciplinary Research Institute (CNRS)
Lille, France

April 2006

Modelling transcription and translation in π

Biological goal:

- Simulate temporal expression patterns.
- Observe effects of modifications.
- Not just *one* system of transcription and translation,
- Here: **basic machinery of gene expression**,
- such that specific details can be added.

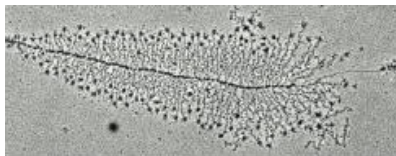
Modeling framework:

- polyadic π calculus + stochastics + **pattern guarded inputs**,
- allows to express **objects**,
- useful biological extensibility via inheritance (and flexible parametrization).

Outline

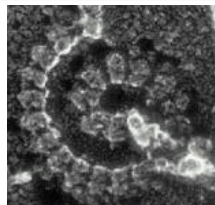
- 1 Biology
- 2 Pi
- 3 Modeling with objects
 - objects: lists
 - inheritance: degradable lists
 - multi-profile objects: queueing lists
- 4 Biological modeling
- 5 Simulation

Gene expression

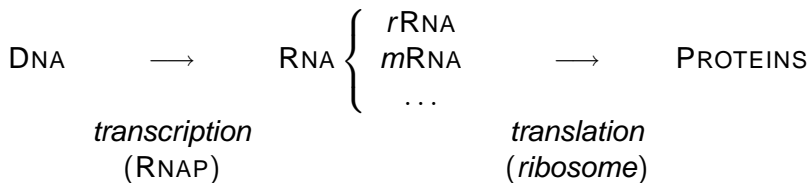


transcription

[electrone microscopy images from Alberts et al, *Molecular biology of the cell*]

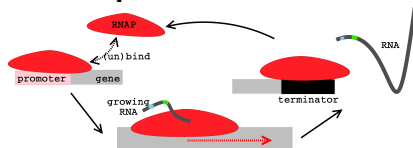


translation

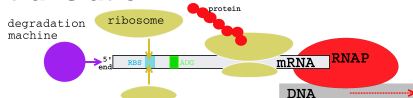


Gene expression

transcription



translation



- both: initiation, elongation, termination,
- mRNA is subject to competing translation and **degradation**,
- both co-transcriptional,
- *regulation* at all stages.

Parameters for transcription

	range	biological significance
RNAP: $\frac{k_{on}}{k_{off}}$	10^6 to 10^9	stability of closed RNAP-promoter complex
RNAP: k_{init}	$10^{-3} \frac{1}{\text{sec}}$ to $10^{-1} \frac{1}{\text{sec}}$	transition closed \rightarrow open complex
elongation delay	30 nt/sec	avg of exponential distribution
gene length	1000 nt	transcription delay: 30 sec
repressors: $\frac{k_{on}}{k_{off}}$	10^7 to 10^{11}	

Parameters for translation

	range	biological significance
ratio degradosome vs ribosome access to mRNA	1 to 100	typical ranges of <i>mean</i> protein number translated from one transcript
translation speed	100 nt/sec	average of exponential distribution of individual steps
mRNA length	1000 nt	average translation delay: 10 seconds

What can people measure?

[Abbondanzieri et. al, nature 11/2005]

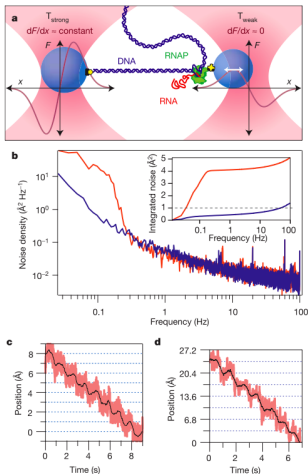


Figure 1 | Experimental set-up, passive force clamp and sensitivity of the RNAP dumbbell array. a. Cartoon of the dumbbell geometry with

[Baek et al, 2003, Journal Mol Biol]

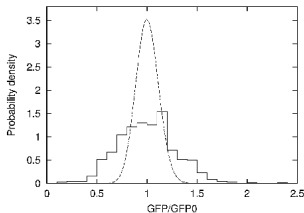


Figure 4. Distribution of GFP. The Figure shows the measured fluorescence of KB126 (continuous line) after growth in YT broth and the results of a simulation of GFP production (broken line). The fluorescence of KB126 was determined on 659 cells. The X axis shows

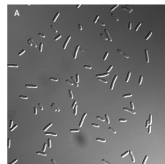


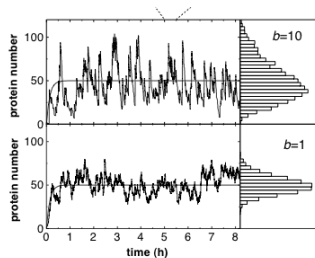
Figure 5. Confocal microscopy images. The strain shown in the negative image of the fluorescence.

Quantitative fine-tuning matters

- (unregulated) expression of a single gene
- vary efficiencies of transcription versus translation initiation.
- Case 1: transcription low + translation high, case 2: high + low. Same *average* protein level *but*.
- Case 1: **translational bursting**. Results in large variability at population level.
- main contributor to **stochasticity in bacterial gene expression**
- theory 2001 [Kierzek], experiment 2002 [Ozbudak], review [Kaern et al, Nature Reviews Genetics, 2005, 6(6),451–467]



Translational bursting



[Ozbudak et. al, Nature Genetics
2002, vol 31,pp 69–73]

- both *average* protein level of 50
- case 1: **translational bursting**, with mean $b = 10$ proteins per transcript
- case 2: rather stable protein level, mean $b = 1$ protein per transcript
- strong fluctuations around mean due to translation-degradation race for mRNA

Table 1 • Translational mutants: point mutations in the RBS and initiation codon of *gfp*

Strain	Ribosome binding site	Initiation codon	Translational efficiency
ERT25	GGG AAA AGG AGG TGA ACT ACT	ATG	1.00
ERT27	GGG AAA AGG AGG TGA ACT ACT	T TG	0.87
ERT3	GGG AAA AGG T GG TGA ACT ACT	ATG	0.84
ERT29	GGG AAA AGG AGG TGA ACT ACT	G TG	0.66

Table 2 • Transcriptional mutants: point mutations in the P_{SPAC} promoter

Strain	-10 regulatory region -10	+1	Transcriptional efficiency
ERT57	CAT AAT GTG TGT	AAT	6.63
ERT25	CAT AAT GTG TGG	AAT	1.00
ERT53	CAT AAT GTG TGC	AAT	0.79
ERT51	CAT AAT GTG TGA	AAT	0.76
ERT55	CAT AAT GTG TAA	AAT	0.76



(Yet) a(nother) π calculus dialect

- content of communication: tuples of names,
- but: synchronization over pairs of names + function symbols,
- stateful objects with changing interfaces.

Draws from:

- Priami, Regev et. al (2001) bio-stochastic π ,
- π with data terms (cryptographic protocols),
- Vasconcelos calculus of objects (1993).

Stochastic π with pattern guarded inputs

Vocabulary

channel *names* x, y ,
function *symbols* f ,
stochastic rate definitions ρ

Processes

$$P ::= P_1 \mid P_2$$

$$\quad \mid \text{new } x(\rho).P$$

$$\quad \mid A(\bar{x})$$

$$\quad \mid C_1 + \dots + C_n$$

Definitions

$$D ::= A(\bar{y}) \triangleq P$$

Choices

$$C ::= x!f(\bar{y}).P$$

$$\quad \mid x?f(\bar{y}).P$$

tuple output, pattern input.

Stochastic π with pattern guarded inputs

Vocabulary

channel *names* x, y ,
function *symbols* f ,
stochastic rate definitions ρ

Processes

$$P ::= P_1 \mid P_2$$

$$\quad \mid \text{new } x(\rho).P$$

$$\quad \mid A(\bar{x})$$

$$\quad \mid C_1 + \dots + C_n$$

Definitions

$$D ::= A(\bar{y}) \triangleq P$$

Choices

$$C ::= x!f(\bar{y}).P$$

$$\quad \mid x?f(\bar{y}).P$$

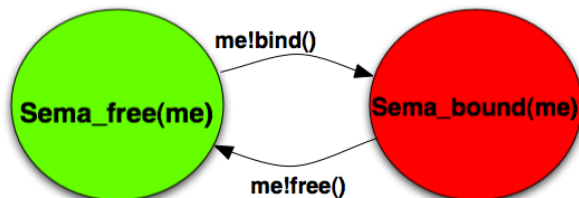
tuple output, pattern input.

Semaphore

$$\text{Semaphore_free}(me) \triangleq$$

$$me?bind().\text{Semaphore_bound}(me)$$

$$\text{Semaphore_bound}(me) \triangleq$$

$$me?free().\text{Semaphore_free}(me)$$


Semaphores can only be bound **once**.
Once bound, don't accept another *bind()*.

Module notation

module 'semaphore'

export

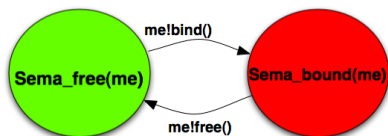
Semaphore **with** bind / 0 , free / 0

define

Semaphore(me) \triangleq Semaphore_free(me)

Semaphore_free(me) \triangleq me?bind(). Semaphore_bound(me)

Semaphore_bound(me) \triangleq me?free(). Semaphore_free(me)



Operational semantics

Communication, choice, pattern matching:

$$x!f(\bar{y}).P_1 + \dots \mid x?f(\bar{z}).P_2 + \dots \rightarrow P_1 \mid P_2[\bar{z} \mapsto \bar{y}]$$

if \bar{z} free for \bar{y} in P_2

Unfolding of parametric processes:

$$A(\bar{y}) \rightarrow P[\bar{x} \mapsto \bar{y}] \quad \text{if } A(\bar{x}) \triangleq P \text{ is a valid definition}$$

With respect to

structural congruence, closure rules, α -conversion

Operational semantics

Communication, choice, pattern matching:

$$x!f(\bar{y}).P_1 + \dots \mid x?f(\bar{z}).P_2 + \dots \rightarrow P_1 \mid P_2[\bar{z} \mapsto \bar{y}]$$

if \bar{z} free for \bar{y} in P_2

Unfolding of parametric processes:

$$A(\bar{y}) \rightarrow P[\bar{x} \mapsto \bar{y}] \quad \text{if } A(\bar{x}) \triangleq P \text{ is a valid definition}$$

With respect to

structural congruence, closure rules, α -conversion

Semaphore in action

```

Semaphore_bound(s) | s!bind() | s!free()
  → s?free().Semaphore_free(s) | s!bind() | s!free()
  → Semaphore_free(s) | s!bind()
  → s?bind().Semaphore_bound(s) | s!bind()
  → Semaphore_bound(s)

```

Definition:

$$\text{Semaphore_free}(me) \triangleq me?bind().\text{Semaphore_bound}(me)$$

$$\text{Semaphore_bound}(me) \triangleq me?free().\text{Semaphore_free}(me)$$

Communication: blocking of unexpected messages

A (Send)

$$me !f(\bar{y}) . A_{cont}$$

B (Receive)

$$me ? f(\bar{z}) . B_{cont} \\ + me ? g(\bar{v}) . B'_{cont}$$

Communicating processes must agree in *both* channel name and function symbol.

Communication: blocking of unexpected messages

A (Send)

$$me !f(\bar{y}) . A_{cont}$$

B (Receive)

$$me ? f(\bar{z}) . B_{cont} \\ + me ? g(\bar{v}) . B'_{cont}$$

C (Send)

$$me !h(\bar{y}) . C_{cont}$$

C can not communicate with B.
No input offer matches h.

Modeling with objects

Lists and lists:

- readable,
- readable + degradable,
- readable + queueing,
- readable + queueing + degradable,
- **biological modeling: all these ++**

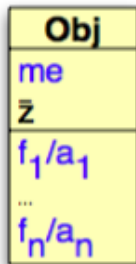
objects: lists

Objects

$$\text{Obj}(me, \bar{z}) \triangleq$$

$$me?f_1(\bar{x}_1).P_1$$

$$+\dots$$

$$+me?f_n(\bar{x}_n).P_n$$


class	definition of parametric process
object Obj	instance of class, identified by 'me'
Obj's function f	f-guarded input over 'me'
interface of Obj	collection of functions, including arities

[passive, single-profile object]

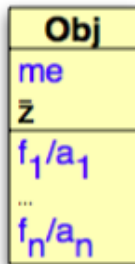
objects: lists

Objects

$$Obj(me, \bar{z}) \triangleq$$

$$me?f_1(\bar{x}_1).P_1$$

$$+\dots$$

$$+me?f_n(\bar{x}_n).P_n$$


class	definition of parametric process
object Obj	instance of class, identified by ' me '
Obj's function f	f-guarded input over 'me'
interface of Obj	collection of functions, including arities

[passive, single-profile object]

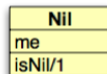
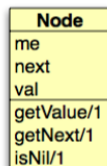
objects: lists

Persistent lists

```

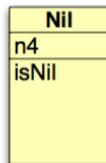
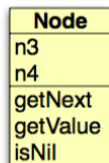
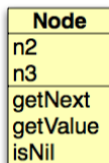
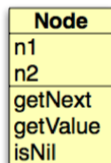
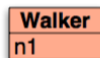
module 'persistent list '
export
  Node with getNext/1,getValue/1, isNil/1
  Nil  with isNil/1
define
  Node(me, next , val)  $\triangleq$ 
    me?getValue(c) .c!val.Node(me, next , val)
    + me?getNext(c) .c!next.Node(me, next , val)
    + me?isNil(c) .c!false().Node(me, next ,
      val)
  Nil(me)  $\triangleq$  me?isNil(c) .c>true().Nil(me)

```



objects: lists

Walking over a list



```
Walker(node)  $\triangleq$  new b.node!isNil(b).
  b?true().0
+ b?false().new c.node!getNext(c).c?next.Walker(next)
```

[Walker: slightly different than our 'passive objects']

inheritance: degradable lists

Inheritance

Next:

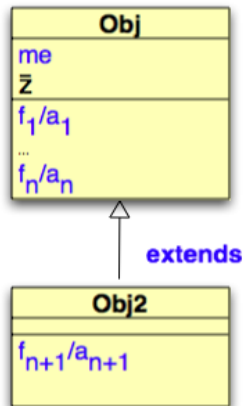
- re-use existing list specification,
- create lists that can be destructed node by node,
- can still be walked over by Walker.

Obj2 **extends** Obj

Obj2(me, \bar{z}) **extended by**
 $C_{n+1} + \dots + C_m$

Resolved as:

Obj2(me, \bar{z}) \triangleq
 $C_1 + \dots + C_m$ [Obj \mapsto Obj2]



inheritance: degradable lists

Degradable list

```

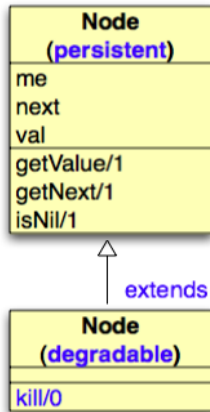
module 'degradable list '
import
  Plist(Node, Nil) from 'persistent list '
export
  Node extends Plist.Node by kill/0
  Nil extends Plist.Nil by kill/0
define
  Node(me, val, next) extended by
    me ? kill().0
  Nil(me) extended by me ? kill().0

```

```

Killer(node)  $\triangleq$  new b.node!isNil(b).
  b?true().node!kill().0
+ b?false().new c.node!getNext(c).
  c?next.node!kill().Killer(next)

```



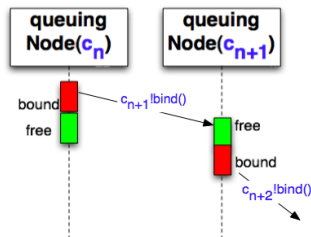
multi-profile objects: queueing lists

Prohibit overtaking

- lists can have *several* visitors at the same time,
- visitors may overtake each other.
- **Explicitly no overtaking in transcription/translation!**
- Howto prohibit? Combine semaphore and list nodes!

multi-profile objects: queuing lists

Queueing list



Nodes with multiple profiles:

- Nodes synchronize with their neighbors,
- in profile **bound** a Node serves its visitor, while blocking arrival of another,
- in profile **free** they signalize availability for new visitor, by accepting **bind()** input.

multi-profile objects: queueing lists

Multi-profile object

- a multi-profile object is a **collection** of objects,
- defined in same module,
- one per discrete **state** of an entity,
- offer different interfaces per profile,
- profiles are mutually recursive.

Notation:

$$Obj_{p_1}(me, \bar{z}_1) \triangleq C_1^1 + \dots + C_{n_1}^1$$

...

$$Obj_{p_n}(me, \bar{z}_n) \triangleq C_1^n + \dots + C_{n_n}^n$$

Multi-profile object

- a multi-profile object is a **collection** of objects,
- defined in same module,
- one per discrete **state** of an entity,
- offer different interfaces per profile,
- profiles are mutually recursive.

Example:

$$\text{Semaphore_free}(me) \triangleq me?.bind().\text{Semaphore_bound}(me)$$
$$\text{Semaphore_bound}(me) \triangleq me?.free().\text{Semaphore_free}(me)$$

multi-profile objects: queueing lists

Queueing list module

```
1  module 'persistent queueing list '  
2  export  
3      Node with getNext/1 , getValue/1 , isNil/1  
4      Nil  with isNil/1  
5  define  
6      Node(me, next , val)  $\triangleq$  Node_free (me, next , val)  
7      Node_free (me, next , val)  $\triangleq$   
8          me?bind () . Node_bound (me, next , val)  
9      Node_bound (me, next , val)  $\triangleq$   
10         me?isNil (c) . c! false () . Node_bound (me, next , val)  
11         + me?getValue (c) . c! val . Node_bound (me, next , val)  
12         + me?getNext (c) . next ! bind () . c! next .  
13         Node_free (me, next , val)  
14  
15     Nil (me)  $\triangleq$   
16         me?isNil (c) . c! true () . Nil (me)  
17         + me?bind () . Nil (me)
```



Biological modeling

- DNA
- RNAP (assembles or *polymerizes* RNA)
- mRNA
- ribosome + degradation machinery

DNA module



- P: transcription initiates on promoter,
- connection with RNAP is established over global 'rnap' channel
- N: transcription is propagated over (private channels between) nucleotides, **queueing node ++**
- mRNA is assembled stepwise,
- T: transcription ends on terminator, **queueing node ++**

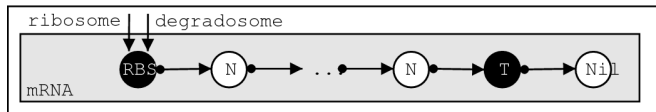
DNA module

```

1  module 'DNA'
2  import   List(Node, Nil) from 'persistent queueing list'
3     Mrna(Node, Terminator) from 'mRNA'
4  export
5     Nucleotide extends List.Node by isTerm/1, elongate/2
6     Terminator extends List.Node by isTerm/1, elongate/2
7  define
8     Nucleotide_bound(me, next, v) extended by
9         me?isTerm(c) . c! false () . Nucleotide_bound(me, next, v)
10        + me?elongate(rna, c) . new rna_nxt( $\rho_{rna}$ ) . c! rna_nxt .
11        Nucleotide_bound(me, next, v) | Mrna.Nucleotide(rna,
12            rna_nxt, v')
13    Terminator_bound(me, next, v) extended by
14        me?isTerm(c) . c! true () . Terminator_bound(me, next, v)
15        + me?elongate(rna, c) . new last( $\rho_{rna}$ ) . c! last .
16        Terminator_bound(me, next, v) | Mrna.Terminator(rna,
17            last, v') | Nil(last)

```

mRNA module



- RBS: interfaces for translation and degradation machineries,
- N: propagation of degradation and translation, **degradable queueing list node++**
- T: termination of translation, **degradable queueing list node++**.

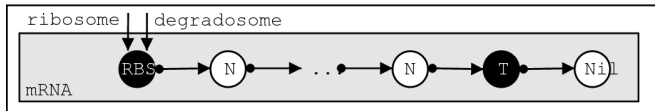
mRNA module

```

module 'mRNA'
channel
  ribosome with bind/1
  degradosome with bind/1
import
  List(Node, Nil) from 'non-persistent queueing list'
  Protein from 'your favorite protein'
export
  Nucleotide extends List.Node by isTerm/1, elongate/0
  Terminator extends List.Node by isTerm/1, elongate/0
  RBS with init/1, unbind/0
define
  Nucleotide_bound(me, next, val) extended by
    me?isTerm(c) . c! false () . Nucleotide_bound(me, next, v)
    + me?elongate () . Nucleotide_bound(me, next, v)
  Terminator_bound(me, next, val) extended by
    me?isTerm(c) . c! true () . Terminator(me)
    + me?elongate ?() . new_it (c, next) . Terminator_bound(me

```

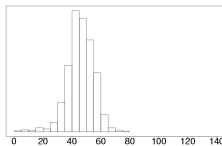
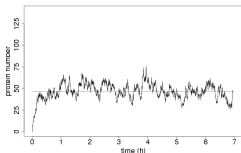
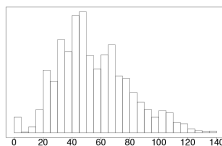
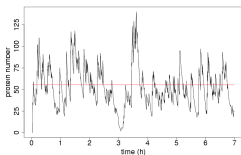
mRNA module



$$\begin{aligned} \text{RBS_free}(me, next) &\triangleq \\ &\text{ribosome?bind}(c) . c!me.\text{RBS_bound}(me, next) \\ &+ \text{degradosome?bind}(c) . next!bind() . c!next.0 \\ \text{RBS_bound}(me, next) &\triangleq \\ &me?init(c) . next!bind() . c!next . \text{RBS_free}(me, next) \\ &+ me?unbind() . \text{RBS_free}(me, next) \end{aligned}$$

[RBS_free: communication over global channels. RBS_bound: passive object]

Stochasticity in expression of single gene: translational bursting



- case 1:
translational
bursting, large
variability
across
population,
- case 2: steady
protein
production, little
variability
across
population.

Covering more biology

A number of cases can be covered by extensions:

- DNA and transcription:
 - alternative promoters for one gene,
 - multiple genes for one promoter (operon),
 - detailed control of initiation [previous work] + termination,
 - modification of RNAP,
 - colliding traffic on double-stranded DNA,
- RNA and translation/degradation:
 - polycistronic mRNA,
 - alternative 5' ends, . . . ,
 - details of regulation,
 - more about translation,
- integration of these components into larger models.

Credits

- C. Lhoussaine.
- B. Vandenbunder,
- J. Niehren.