Bacterial gene expression in the pi calculus

Céline Kuttler

Interdisciplinary Research Institute (CNRS)
Lille, France

Trento, May 4th, 2006
Understanding gene expression

which genes are used, where, how, and why?

- **importance:**
  - development and evolution,
  - health and disease.

- **cases:**
  - bacteria: observe fundamental phenomena,
  - higher organisms: upscale in complexity.

- **dynamics:**
  - results from a number of components,
  - and their interactions over time,
  - complex dependencies.
Objectives

- **bacterial** gene expression:
  - expression of all gene by the same base machinery
  - expression of each gene varies in regulation

- stochastic simulation:
  - reproduce temporal and quantitative patterns of the dynamics of gene expression,
  - discrete event based approach,
  - modeling as concurrent programming.

- extensible model of transcription & translation,
  - such that more detail can be incorporated later,
  - how to support extensibility in the $\pi$-calculus approach?

[CMSB 2005, journal submission 2006, PhD thesis in progress]
Related work

Calculi:
- $\pi$-calculus milner et al
- stochastic $\pi$ (Priami), PEPA (Hillston)
- Danos, Cardelli, ...

stochastic simulation of coupled chemical reactions:
- Gillespie 1976: standard algorithm,
- Priami, Regev, Shapiro, Silverman 2001: $\pi$ and Gillespie

discrete event models of gene expression:
- pioneered 1977 on ribosomal movement, Heijne et. al
- since late 1990s, reviewed by Gilman & Arkins

simulating genetic networks in stochastic $\pi$:
- Kuttler, Niehren (BioConcur 2004, TCSB 2006) – regulation of transcription *initiation* at the $\lambda$ switch
- Cardelli et al. (TCSB 2006) – expression of a gene represented as atomic (summarized in one step)
- here: transcription and translation. no details of regulation.
Outline

1. Bacterial gene expression
2. $\pi$ calculus
3. Modeling with objects
   - objects: lists
   - inheritance: degradable lists
   - multi-profile objects: queueing lists
4. Biological modeling
5. Simulation

C. Kuttler
IRI, Lille
Gene expression: central dogma

\[
\text{DNA} \quad \rightarrow \quad \text{mRNA} \quad \rightarrow \quad \text{proteins}
\]

*transcription*  \quad *translation*

[basic scheme, need more details for regulation and higher organisms.]
Transcription: DNA $\rightarrow$ (m)RNA

- performed by RNAP in phases:
  - initiation,
  - elongation,
  - termination.

- each can be regulated.

$\text{R\text{NAP} + P} \rightarrow k_{\text{on}} (\text{R\text{NAP} \cdot P})_{\text{closed}}$  
$\rightarrow k_{\text{off}} (\text{R\text{NAP} \cdot P})_{\text{closed}}$  
$\rightarrow k_{\text{init}} (\text{R\text{NAP} \cdot P})_{\text{open}}$  
$\rightarrow k_{\text{elong}} \text{R\text{NAP} \cdot D\text{NA}}_{n}$  
$\rightarrow k_{\text{elong}} \text{R\text{NAP} \cdot D\text{NA}_{\text{terminator}}}$

[core initiation scheme à la McClure, Annual Review Biochemistry (1985),
elongation & termination à la McAdams & Arkin, PNAS 1997]
Transcription: DNA → (m)RNA

- performed by RNAP in phases:
  - initiation,
  - elongation,
  - termination.

- each can be regulated.

[Alberts et al, *Molecular biology of the cell*]
Transcription: DNA $\rightarrow$ (m)RNA

**evidence:** RNAP elongates mRNA in steps of 1 nucleotide, with exponential waiting times

- performed by RNAP in phases:
  - initiation,
  - elongation,
  - termination.
- each can be regulated.

[Abbondanzieri et. al, nature 11/2005]
Translation: mRNA $\rightarrow$ protein

- mRNA is processed while being produced,
- translation is performed by ribosome in phases:
  - initiation,
  - elongation,
  - termination,
- translation competes with degradation,

[Alberts et al, Molecular biology of the cell]
Speed of reactions

- each reaction step takes a certain time
- costly to determine rates by experiment
- can vary over orders of magnitude from case to case:
  - transcription initiation: 1 per hour – 1 per second
  - translation efficiency: averages of 1 – 100 proteins per mRNA
- reaction speeds enter models as rates
- are decisive for the accuracy of quantitative predictions

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

<table>
<thead>
<tr>
<th>Strain</th>
<th>Ribosome binding site</th>
<th>Initiation codon</th>
<th>Translational efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERT25</td>
<td>GGG AAA AGG AGG TGA ACT ACT</td>
<td>ATG</td>
<td>1.00</td>
</tr>
<tr>
<td>ERT27</td>
<td>GGG AAA AGG AGG TGA ACT ACT</td>
<td>ATG</td>
<td>0.87</td>
</tr>
<tr>
<td>ERT3</td>
<td>GGG AAA AGG AGG TGA ACT ACT</td>
<td>ATG</td>
<td>0.84</td>
</tr>
<tr>
<td>ERT29</td>
<td>GGG AAA AGG AGG TGA ACT ACT</td>
<td>ATG</td>
<td>0.66</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Strain</th>
<th>-10 regulatory region</th>
<th>+1</th>
<th>Transcriptional efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERT57</td>
<td>CAT AAT GTC TGC AAT</td>
<td></td>
<td>6.63</td>
</tr>
<tr>
<td>ERT25</td>
<td>CAT AAT GTC TGC AAT</td>
<td></td>
<td>1.00</td>
</tr>
<tr>
<td>ERT53</td>
<td>CAT AAT GTC TGC AAT</td>
<td></td>
<td>0.79</td>
</tr>
<tr>
<td>ERT51</td>
<td>CAT AAT GTC TGC AAT</td>
<td></td>
<td>0.76</td>
</tr>
<tr>
<td>ERT55</td>
<td>CAT AAT GTC TAA AAT</td>
<td></td>
<td>0.76</td>
</tr>
</tbody>
</table>

Dynamics in gene expression

- many surprises in experiments
  - can not explained by available knowledge
  - can not be reproduced
- example: genetically identical cells can widely differ,
- strong variability in a protein’s level

---

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

<table>
<thead>
<tr>
<th>Strain</th>
<th>Ribosome binding site</th>
<th>Initiation codon</th>
<th>Translational efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERT25</td>
<td>GGG AAA AGG TGG TGA ACT ACT ATG</td>
<td>ATG</td>
<td>1.00</td>
</tr>
<tr>
<td>ERT27</td>
<td>GGG AAA AGG TGG TGA ACT ACT ATG</td>
<td>ATG</td>
<td>0.87</td>
</tr>
<tr>
<td>ERT3</td>
<td>GGG AAA AGG TGG TGA ACT ACT ATG</td>
<td>ATG</td>
<td>0.84</td>
</tr>
<tr>
<td>ERT29</td>
<td>GGG AAA AGG TGG TGA ACT ACT ATG</td>
<td>ATG</td>
<td>0.66</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Strain</th>
<th>−10 regulatory region</th>
<th>+1</th>
<th>Transcriptional efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERT57</td>
<td>CAT AAT GGG TGG AAT</td>
<td>6.63</td>
<td></td>
</tr>
<tr>
<td>ERT25</td>
<td>CAT AAT GGG TGG AAT</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>ERT53</td>
<td>CAT AAT GGG TGG AAT</td>
<td>0.79</td>
<td></td>
</tr>
<tr>
<td>ERT51</td>
<td>CAT AAT GGG TGG AAT</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>ERT55</td>
<td>CAT AAT GGG TGG AAT</td>
<td>0.76</td>
<td></td>
</tr>
</tbody>
</table>

Dynamics in gene expression

- many **surprises** in experiments
  - can not explained by available knowledge
  - can not be reproduced
- example: genetically identical cells can widely differ,
- strong variability in a protein’s level

**translational bursting**

key contributor to **stochasticity in bacterial gene expression**
Quantitative control in expression of a single gene

- **alternative combinations** of initiation efficiencies:
  1. slow transcription + fast translation (**typical in bacteria**),
  2. fast transcription + slow translation (**very rare exception**).

- **observed effects**:
  1. strong variability in protein level, **translational bursting**,
  2. same *average* protein level, weak fluctuations.

---

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

<table>
<thead>
<tr>
<th>Strain</th>
<th>Ribosome binding site</th>
<th>Initiation codon</th>
<th>Translational efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERT25</td>
<td>GGG AAA AGG AGG TGA ACT ACT ATG</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>ERT27</td>
<td>GGG AAA AGG AGG TGA ACT ACT TGG</td>
<td>0.87</td>
<td></td>
</tr>
<tr>
<td>ERT3</td>
<td>GGG AAA AGG TGG TGA ACT ACT ATG</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>ERT29</td>
<td>GGG AAA AGG AGG TGA ACT ACT TGG</td>
<td>0.66</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2** • Transcriptional mutants: point mutations in the P<sub>Pac</sub> promoter

<table>
<thead>
<tr>
<th>Strain</th>
<th>-10 regulatory region</th>
<th>+1</th>
<th>Transcriptional efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERT57</td>
<td>CAT AAT GGG TCG AAT</td>
<td></td>
<td>6.63</td>
</tr>
<tr>
<td>ERT25</td>
<td>CAT AAT GGG TCG AAT</td>
<td></td>
<td>1.00</td>
</tr>
<tr>
<td>ERT53</td>
<td>CAT AAT GGG TGG AAT</td>
<td></td>
<td>0.79</td>
</tr>
<tr>
<td>ERT51</td>
<td>CAT AAT GGG TGG AAT</td>
<td></td>
<td>0.76</td>
</tr>
<tr>
<td>ERT55</td>
<td>CAT AAT GGG TGG AAT</td>
<td></td>
<td>0.76</td>
</tr>
</tbody>
</table>

Translational bursting

1. **strong fluctuations.**
   - around average of 50 proteins,
   - rare transcripts, \( b = 10 \) proteins per transcript in average
   - strong fluctuations in protein crops due to translation-degradation race for mRNA (geometric distribution)
   - **translational bursting.**

2. **protein level fluctuates only mildly.** Frequent transcription, \( b = 1 \) protein per transcript in average

Stochastic π-calculus

Priami, Regev et al.'s biochemical stochastic π (2001) offers

- formal modeling language for systems biology,
- concurrency,
- stochastic rates control interaction speeds,
- supported by simulation engines.
Some limitations of stochastic $\pi$

lacks support for usual programming abstractions:
- object orientation,
- module system,
- ...

difficult to:
- re-use and extend models,
- represent statefulness of molecules,
- represent interfaces of molecules.
Suggest modified modeling framework

- extension of stochastic $\pi$ calculus by pattern guarded inputs,
- allows to express objects,

new:

- can express multi-profile objects [with D. Duchier, MTCoord2005 workshop],
- adapted stochastic semantics,
- adapted notion of inheritance,
- supports extensibility.
Stochastic π with pattern guarded inputs: syntax

**Vocabulary**
- channel *names* $x, y$,
- function *symbols* $f$,
- stochastic rate definitions $\rho$.

**Processes**

\[ P ::= P_1 | P_2 \]
\[ | \text{new } x(\rho).P \]
\[ | A(x) \]
\[ | C_1 + \cdots + C_n \]

**Definitions**

\[ D ::= A(y) \triangleq P \]

**Choices**

\[ C ::= x!f(y).P \]
\[ | x?f(y).P \]

tuple output, pattern input.
Stochastic $\pi$ with pattern guarded inputs: syntax

**Vocabulary**
- channel *names* $x, y$,
- function *symbols* $f$,
- stochastic rate definitions $\rho$.

**Processes**

$$P ::= P_1 | P_2$$

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

$$| A(x)$$

$$| C_1 + \cdots + C_n$$

**Definitions**

$$D ::= A(y) \triangleq P$$

**Choices**

$$C ::= x!f(y).P$$

$$| x?f(y).P$$

*tuple output, pattern input.*
Semaphore

Semaphore_free(me) ≜ me?bind().Semaphore_bound(me)
Semaphore_bound(me) ≜ me?free().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) ≜ Semaphore_free(me)
    Semaphore_free(me) ≜ me?bind().Semaphore_bound(me)
    Semaphore_bound(me) ≜ me?free().Semaphore_free(me)
### Operational semantics

**Communication, choice, pattern matching:**

\[ x!f(y).P_1 + \ldots \mid x?f(z).P_2 + \ldots \rightarrow P_1 \mid P_2[z \leftrightarrow y] \]

if \( z \) free for \( y \) in \( P_2 \)

**Unfolding of parametric processes:**

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

**With respect to**

structural congruence, closure rules, \( \alpha \)-conversion

---

C. Kuttler  
IRI, Lille

Bacterial gene expression in the pi calculus
Operational semantics

Communication, choice, pattern matching:

\[ x!f(y).P_1 + \ldots \mid x?f(z).P_2 + \ldots \rightarrow P_1 \parallel P_2[z \mapsto y] \]

if \( z \) free for \( y \) in \( P_2 \)

Unfolding of parametric processes:

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

With respect to

structural congruence, closure rules, \( \alpha \)-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:

Semaphore_free(me) ≜
    me?bind().Semaphore_bound(me)

Semaphore_bound(me) ≜
    me?free().Semaphore_free(me)
Communication: blocking of unexpected messages

A (Send)

\[ m e \! f(y) \cdot A_{\text{cont}} \]

B (Receive)

\[ m e ? f(z) \cdot B_{\text{cont}} \]
\[ + m e ? g(v) \cdot B'_{\text{cont}} \]

Communicating processes must agree in both channel name and function symbol.
Communication: blocking of unexpected messages

A (Send)

\[ \text{me} ! f(y) \cdot A_{\text{cont}} \]

B (Receive)

\[ \text{me} ? f(z) \cdot B_{\text{cont}} \]
\[ + \text{me} ? g(v) \cdot B'_{\text{cont}} \]

C (Send)

\[ \text{me} ! h(y) \cdot C_{\text{cont}} \]

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

Running example: lists

- readable,
- readable + degradable,
- readable + queueing,
- readable + queueing + degradable,
- basis for modeling DNA and mRNA.
### Objects

$$
\text{Obj}(\text{me}, \overline{z}) \triangleq \me?f_1(x_1).P_1 + \ldots + \me?f_n(x_n).P_n
$$

<table>
<thead>
<tr>
<th>class</th>
<th>definition of parametric process</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>instance of class, identified by 'me'</td>
</tr>
<tr>
<td>Obj's function f</td>
<td>f-guarded input over 'me'</td>
</tr>
<tr>
<td>interface of Obj</td>
<td>collection of functions, including arities</td>
</tr>
</tbody>
</table>

[passive, single-profile object à la Vasconcelos 1993.]
Objects

\[ \text{Obj}(me, \overline{z}) \triangleq me?f_1(x_1).P_1 + \ldots + me?f_n(x_n).P_n \]

<table>
<thead>
<tr>
<th>class</th>
<th>definition of parametric process</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>instance of class, identified by 'me'</td>
</tr>
<tr>
<td>Obj</td>
<td>f-guarded input over 'me'</td>
</tr>
<tr>
<td>Obj’s function f</td>
<td></td>
</tr>
<tr>
<td>interface of Obj</td>
<td></td>
</tr>
<tr>
<td></td>
<td>collection of functions, including arities</td>
</tr>
</tbody>
</table>

[passive, single-profile object à la Vasconcelos 1993.]
Persistent lists

module 'persistent list'

export

Node with getNext/1, getValue/1, isNil/1
Nil with isNil/1

define

Node(me, next, val) ≜
    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) ≜ me?isNil(c).c!true().Nil(me)
Walking over a list

Walker

\[ \text{Walker}(\text{node}) \triangleq \text{new } b. \text{node}! \text{isNil}(b). \]
\[ + \quad b? \text{true}().0 \]
\[ + \quad b? \text{false}().\text{new } c. \text{node}! \text{getNext}(c).c? \text{next}. \text{Walker}(\text{next}) \]
Inheritance

Next:

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

\[
\text{Obj2} \text{ extends } \text{Obj} \text{ by } \\
\text{Obj2} (\text{me}, \overline{z}) \text{ extended by } \\
C_{n+1} + \ldots + C_m
\]

Can be spelled out as:

\[
\text{Obj2} (\text{me}, \overline{z}) \triangleq \\
C_1 + \ldots + C_m [\text{Obj} \mapsto \text{Obj2}]
\]
 inheritance: degradable lists

Inheritance

Next:

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

Obj2 extends Obj by ...

Obj2 (me, \(\overline{z}\)) extended by \(C_{n+1} + \ldots + C_m\)

Can be spelled out as:

Obj2 (me, \(\overline{z}\)) \(\triangleq\) \(C_1 + \ldots + C_m\) [Obj \(\mapsto\) Obj2]
Inheritance

Next:

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

$\text{Obj2} \ extends \ \text{Obj} \ by \ ...$

$\text{Obj2} (\text{me}, \overline{z}) \ extended \ by \ C_{n+1} + \ldots + C_m$

Can be *spelled out as*:

$\text{Obj2} (\text{me}, \overline{z}) \ \triangleq \ C_1 + \ldots + C_m \ [ \text{Obj} \ \leftrightarrow \ \text{Obj2} ]$
inheritance: degradable lists

Degradable list

module 'degradable list'

import

    List(Node, Nil) from 'persistent list'

export

    Node extends List.Node by kill/0
    Nil extends List.Nil by kill/0

define

    Node(me, val, next) extended by
        me ? kill().0

    Nil(me) extended by me ? kill().0
Degradable list

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

Killer(node) ≜ new b. node! isNil(b).
  b?true() . node! kill().0
+ b?false(). new c. node! getNext(c).
  c?next. node! kill(). Killer(next)
```
Prohibit overtaking

- each list may have **several visitors** at a time,
- visitors may overtake each other.
- Explicitly no overtaking in our biological application:
  - RNAP transcribing the same gene **queue**,
  - ribosomes translating the same mRNA,
  - degradation does not affect initiated translation!
- Combine semaphore and list nodes:
- nodes may not accept more than one visitor.
a multi-profile object is a collection of objects,
defined in same module,
profiles correspond to states of an entity,
each profile offers different interface (set of functions),
profiles are mutually recursive.

**Example:**

\[
\text{Semaphore\_free}(\text{me}) \triangleq \text{me}\?\text{bind}.\text{Semaphore\_bound}(\text{me})
\]
\[
\text{Semaphore\_bound}(\text{me}) \triangleq \text{me}\?\text{free}.\text{Semaphore\_free}(\text{me})
\]
Multi-profile object

- A multi-profile object is a **collection** of objects,
- Defined in the same module,
- Profiles correspond to **states** of an entity,
- Each profile offers a different **interface** (set of functions),
- Profiles are **mutually recursive**.

**Naming convention:**

\[
\text{Obj}_p^1(me, \overline{z_1}) \triangleq C_1^1 + \ldots + C_{n_1}^1,
\]

\[
\ldots
\]

\[
\text{Obj}_p^n(me, \overline{z_n}) \triangleq C_1^n + \ldots + C_{n_n}^n
\]
Queueing list

Two profiles for nodes:

- **bound**: serving a visitor, blocking arrival of another,
- **free**: willing to accept new visitor, thus offering `bind()` input.
- **switching is synchronized with neighbours.**
multi-profile objects: queueing lists

Queueing list

```
module 'persistent queueing list'

export
  Node with getNext/1, getValue/1, isNil/1
  Nil with isNil/1

define
  Node(me, next, val) ≜ Node_free(me, next, val)

  Node_free(me, next, val) ≜
    me?bind().Node_bound(me, next, val)

  Node_bound(me, next, val) ≜
    me?isNil(c).c!false().Node_bound(me, next, val)
    + me?getValue(c).c!val.Node_bound(me, next, val)
    + me?getNext(c).
    next!bind().
    c!next.
    Node_free(me, next, val) // switch to free

  Nil(me) ≜
    me?isNil(c).c!true().Nil(me)
    + me?bind().Nil(me)
```

[Degradable extension: straightforward.]

C. Kuttler

IRI, Lille

Bacterial gene expression in the pi calculus
Biological modeling

Refine the toolbox:

- DNA (specialized queueing list)
- RNAP (specialized copier)
- mRNA (specialized queueing degradable list)
- ribosome (specialized copier)
- degradation machinery (killer)
DNA module

- **P**: transcription initiates on promoter,
- connection with RNAP is established over global ’rnap’ channel
- **N**: transcription is propagated over nucleotides, queueing node `++`
- mRNA is assembled,
- **T**: transcription ends on terminator, queueing node `++`
[mRNA assembly on DNA, and is catalyzed by RNAP.]
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++, release of full protein.
- reproduces: transcription/translation race, and translation delay.
mRNA module

\[
\begin{align*}
\text{RBS\_free}(\text{me, next}) & \triangleq \\
& \text{ribosome}\?\text{bind}(c).c!\text{me}.\text{RBS\_bound}(\text{me, next}) \\
& + \text{degradosome}\?\text{bind}(c).\text{next!bind}().c!\text{next}.0 \\
\text{RBS\_bound}(\text{me, next}) & \triangleq \\
& \text{me}\?\text{init}(c).\text{next!bind}().c!\text{next}.\text{RBS\_free}(\text{me, next}) \\
& + \text{me}\?\text{unbind}().\text{RBS\_free}(\text{me, next})
\end{align*}
\]

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

```plaintext
module 'mRNA'
channel
  ribosome with bind/1
  degradosome with bind/1
import
  List(Node, Nil) from 'degradable 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(\(\rho_{\text{protein}}\)). Terminator_bound(me, next, val) | Protein(it)
RBS(me, next) ≜ RBS_free(me, next)
RBS_free(me, next) ≜ 
  ribosome?bind(c).c!me.RBS_bound(me, next)
  + degradosome?bind(c).next!bind().c!next.0
RBS_bound(me, next) ≜ 
  me?init(c).next!bind().c!next.RBS_free(me, next)
  + me?unbind().RBS_free(me, next)
```
Simulate expression of single gene

- no tool yet of the suggested $\pi$ variant,
- implemented model within BioSpi,
- manually down-compiled the code to the BioSpi format,
- spelled out inheritance,
- introduced handshake protocols for connection establishment (extrusion of private channels for interaction with list nodes).
Simulate expression of single gene

1. translational bursting, large variability across population,
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 [own previous work] + termination [to do],
  - regulation of elongating RNAP [to do],
  - colliding traffic on double-stranded DNA [to do],

- **RNA and translation/degradation:**
  - polycistronic mRNA,
  - alternative 5’ ends, . . . ,
  - details of regulation [to do],
  - more details in translation and degradation [to do],

- **integration** of these components into larger models.
Outlook

On the language level

- trade-off expressiveness (language) / analyzability (calculi)
- how to integrate these ideas into a powerful language?
- elaborate on inheritance mechanisms, module system, type system

Biological coverage

- study more phenomena at the example of bacteria
- towards better understanding of gene expression in higher organisms
Technical questions to solve in the future

- no implementation yet,
- emulation of function calls in $\pi$ blur up models,
- an object can not call own functions while in a function call (re-entrant locking problem),
- inheritance so far:
  - needs to be down-compiled,
  - no multiple inheritance,
  - not defined for active objects (Walker, Killer, ...)
- nasty in $\pi$: bi-directional channel use.
Bacterial gene expression in the pi calculus

Credits

- D. Duchier,
- C. Lhoussaine,
- J. Niehren,
- B. Vandenbunder.
Thank you

for your comments and questions?