Simulating bacterial transcription and translation in a stochastic pi-calculus

Celine Kuttler

Simulating Bacterial Transcription and Translation in a Stochastic Pi Calculus

Céline Kuttler

Interdisciplinary Research Institute* and LIFL, Lille, France

Abstract. Stochastic simulation of genetic networks based on models in the stochastic π-calculus is a promising recent approach. This paper contributes an extensible model of the central mechanisms of gene expression i.e. transcription and translation, at the prototypical instance of bacteria. We reach extensibility through object-oriented abstractions, that are expressible in a stochastic π-calculus with pattern guarded inputs. We illustrate our generic model by simulating the effect of *translational bursting* in bacterial gene expression.

1 Introduction

Gene expression is essential to all forms of life. In order to maintain their vital functions, cells selectively activate subsets of their genetic material, which is stored in the form of DNA. Genes are segments of this linear macromolecule, they specify molecules that are functional components of the cell. Their expression proceeds through two phases: *transcription* of static DNA-encoded information into a short-lived mRNA molecule, followed by *translation* of the latter into proteins. Expression is subject to rigorous control and modulation, referred to as *gene regulation* [54], that complicate its understanding.

As a result of regulation, the phases of gene expression are not strictly independent, as they were initially deemed. The first such case was found in bacteria: transcription of certain genes aborts prematurely, unless the nascent mRNA is translated efficiently [72]. In higher organisms the basic two-phase scheme of gene expression is extended by additional phases, and couplings may occur between virtually all levels [37, 44]. This work concentrates on the common fundamental mechanisms, as observed in bacteria.

Modeling and simulation seek to contribute to better understanding of the dynamics of gene expression. We follow the *discrete event modeling* approach [61], to be distinguished from more established continuous deterministic frameworks [69]. It is appropriate for gene expression and regulation, since decisive events between individual molecules are inherently discrete. Based on individual interactions, we describe the evolution of molecular networks in gene expression over time. Discrete event models are typically executed through *stochastic simulation* [22], which resolves the nondeterministic choice between alternative

* FRE 2963 of CNRS.

events, and introduces a stochastic scheduling. The probability distributions from which waiting times between events are drawn can lead to significant variability between different executions of a model.

In 2001, Regev et al. proposed the *stochastic π-calculus* as foundation for discrete modeling and stochastic simulation in systems biology [52, 59], which is a stochastic refinement of the π-calculus [40, 42]. The latter was invented by Milner et al. as a minimal formal language for modeling mobile systems, while abstracting from the many details of distributed computation, parallel programs, or multi-threading operating systems. Components of mobile systems dynamically acquire new interaction possibilities as a result of interaction. Such behavior is reminiscent of intra cellular dynamics. Cells are densely populated by a variety of molecules that perpetually interact. The observed interaction patterns evolve, molecules acquire new patterns through modification by others.

The stochastic π-calculus [50] augments the π-calculus with stochastic parameters, which control the speed of nondeterministic interactions. These parameters impose waiting times drawn from exponential distributions. The biochemical stochastic π-calculus [52] specializes the stochastic scheduling to comply with Gillespie's algorithm from 1976, which constitutes a rigorous framework for the simulation of the dynamics of chemical reactions in the regime of small numbers [22].

The stochastic π-calculus has recently been applied to a number of biological simulation case studies, performed in the BioSPI or [52] or SPiM [48] systems. Kuttler and Niehren [34] proposed to simulate gene regulation in the stochastic π-calculus. In a first case study they showed how to simulate the molecular-level dynamics in the control of transcription initiation at the λ switch [53]. Cardelli et al. followed in presenting a complementary view based on networks of genes in the stochastic π-calculus, hereby assuming gene expression as atomic [6]. Both contributions left the modeling of transcription and translation to future research.

Contributions. In this article, we present a generic model of the general machinery of bacterial transcription and translation in a stochastic π-calculus, to the best of our knowledge for the first time. This machinery subsumes the following aspects, including their quantitative control:

Transcription: promoter binding and unbinding of RNA polymerase, initiation of transcription, stepwise elongation of RNA, simultaneous transcription of a gene by multiple RNA polymerases working on disjoint portions of it, co-transcriptional processing of nascent mRNA molecules, intrinsic termination.

Translation: binding and unbinding of ribosomes to mRNA, initiation of translation, elongation, simultaneous elongation by multiple ribosomes operating on disjoint mRNA subsequences, termination, release of the protein.

Degradation of mRNA competing with translation.

Our model is generic in two ways. First, the stochastic parameters can be flexibly set when using our model components. Second, and more importantly, our model can be extended to cover points that arise for particular genes. Since

the consideration of specific cases is essential to biology, it is highly desirable to provide models covering basic mechanisms, that can later be refined to integrate further detail. For instance, our model can be extended to account for genes grouped into operons, as well as alternative promoters for a transcriptional unit. Both are reflected by the resulting mRna. Such detailed aspects matter when engineering regulatory networks [29], and to the quantitative dynamics of gene expression [67, 43], however remain poorly supported in current stochastic simulation packages that tend to emphasize large-scale simulation [55].

From the modeling perspective, the minimality of the π-calculus is sometimes unfortunate. Besides other programming abstractions, the π-calculus lacks object-oriented features and pattern matching. We use object-oriented programming abstractions in order to create models of Dna and mRna that become sufficiently extendable. Objects help specifying the interfaces of concurrent actors. We extend models by inheritance, i.e. we add new functionality to concurrent actors, while keeping consistent with their previously defined interface. This idea is new to π-calculus based simulation approaches in systems biology. As we will see, it applies to both examples mentioned above, operons and tandem promoters.

As modeling language, we rely on the stochastic π-calculus with pattern guarded inputs [35], which has been developed in parallel with the present article. Pattern guarded inputs are the key to express concurrent objects in the π-calculus, as already noticed by Vasconcelos in 1993 [47, 68]. We propose a notion of inheritance for objects in the stochastic π-calculus. We keep this notion on a meta level, rather than defining it inside the π-calculus. It only serves for creating π-calculus programs, and is compiled away before program execution.

Last not least, we illustrate the power of the presented modeling techniques in a case study. We concentrate on the effect of *translational bursting*, that has been identified as a major source of stochasticity in bacterial gene expression [30, 56]. It arises from variations in the quantitative control of transcription and translation, and is thus not captured by the atomic representation of gene expression in [6].

Related work. This paper builds on previous work on stochastic simulation of bacterial gene expression. Heijne and co-authors suggested the the first stochastic treatment of ribosome movement during translation almost 30 years ago [70]. Carrier and Keasling elucidated the relation between molecular actors in translation and mRna decay in 1997 [11]. In the same year McAdams and Arkin attracted wide attention with a scheme for stochastic simulation of gene expression based on Gillespie's algorithm [38]. It combines a continuous representation of transcription initiation in the tradition of [63] with a stochastic account of transcript elongation and subsequent processing of mRna. This schema was successfully applied to bacteriophage lambda [3]. However, this latter work neglected important differences in stochastic fluctuations between genes due to distinct translational efficiencies. This aspect was systematically investigated by Kierzek, Zaim, and Zielenkiewitz [32]. They simulated bacterial gene expression while systematically varying translation and transcription initiation frequencies.

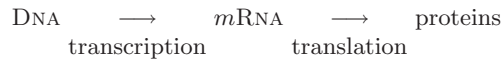


Fig. 1. The first phase of gene expression is *transcription* of static DNA encoded information into RNA molecules, of which several types exist. Short-lived *messenger* RNA, or mRNA, acts as information carrier. It serves as template for *translation* into proteins.

Their predictions were confirmed experimentally by Ozbudak and co-authors [45]. The coupling between transcription and translation in the control of tryptophan expression [72] was recently investigated by Elf and Ehrenberg [19].

Gillespie style simulations have for long been executed by programs crafted ad hoc, for one time use. In recent years several dedicated tools for stochastic simulation of genetic and molecular networks have been suggested [2, 15, 31, 55]. While some of these packages provide templates for gene expression, many models keep being hand-crafted for a single use.

Several dedicated formal languages for biological modeling have been suggested in recent years, each with different objectives [9, 12–14, 51]. They are reviewed in [8, 49].

Outline. We first provide biological background, highlighting stochastic and concurrent features in gene expression. Section 3 introduces the stochastic π-calculus variant with pattern guarded choices that constitutes our modeling language. Sec. 4 introduces the expression of multi-profile objects, and a notion of inheritance for these. We then present dynamical models of transcription and translation, validate our approach with a set of selected simulations, and conclude.

2 Bacterial transcription and translation

We overview the main activities during transcription and translation, contemporaneous events, and couplings between the phases of gene expression. It follows a presentation of details of transcription and translation at the level of molecular interactions, which provides the basis for later discrete event modeling. We then review stochastic aspects of bacterial gene expression, putting an emphasis on the quantitative parameters that control transcription, translation and mRNA decay - they are indispensable for stochastic simulation. We finally present particular cases of transcriptional organization in bacteria, that can have interesting impact on the quantitative patterns of gene expression.

2.1 Overview of genetic actors and gene expression

Each cell of an organism contains its complete genetic information, which is passed on from one generation to the next. It is encoded in a linear, double-stranded DNA macromolecule that winds up to a helix. Each DNA strand con-

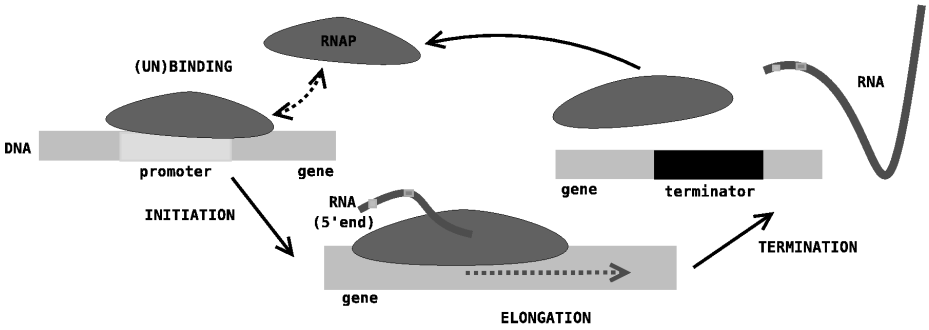


Fig. 2. DNA processing by RNA polymerase (RNAP): promoter binding and initiation, transcript elongation, termination with release of RNA

tains a sequence over the alphabet of nucleotides $\{A, C, G, T\}$. A *gene* is a segment of one strand of DNA, with explicit begin and end delimiters. Its information content can be transcribed into a single-stranded RNA molecule. DNA also comprises regions that are never transcribed, but contain regulatory information. Figure 1 summarizes the two phase of gene expression.

Transcription of a gene is carried out by RNA *polymerase*. RNAP assembles RNA molecules, that reflect the information content of the template DNA strand. Certain categories of transcripts have an immediate functional role in the cell. *Messenger* RNA (mRNA) acts as an information carrier, and is subject to two competing subsequent processing phases: translation into proteins and degradation.

Translation of mRNA into proteins is performed by *ribosomes*, the largest macromolecular complexes in the cell. Ribosomes read out the genetic code from mRNA in three-letter words, mapped into growing sequences of amino acids, which fold into three-dimensional proteins.

Both transcription and translation follow a similar scheme of three phases. Figure 2 illustrates it for transcription, summarizes as follows:

Initiation. RNAP localizes its start point on DNA, a dedicated *promoter* sequence, where it reversibly binds. Upon successful initiation it opens the double-stranded DNA, making its information content accessible. RNAP reads out the first portion of the template DNA strand, assembles the $5'$ end of a new RNA molecule, and continues into elongation.

Elongation. RNAP translocates over DNA in discrete steps of one nucleotide, and for each adds a complementary nucleotide to the growing transcript. Throughout elongation, RNAP maintains a tight contact to the growing extremity of the nascent RNA, as well as the template DNA strand.

Termination. RNAP unbinds from DNA and releases the transcript when it recognizes a *terminator* sequence.

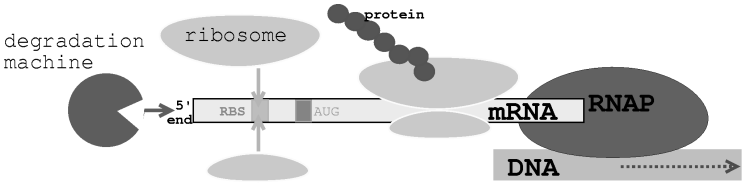
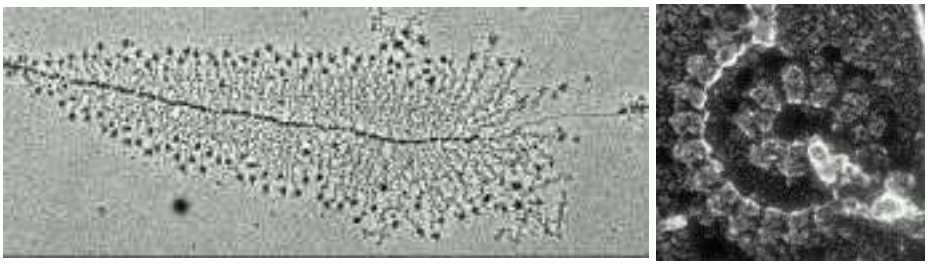


Fig. 3. mRNA is subject to competing translation and transcription. Degradation is initiated at the 5′ end, while the ribosome assembles on the nearby ribosomal binding site Rbs. The actual translation initiates at the start codon, here 'AUG'.



(a) ***DNA*** **(b)** ***mRNA***

Fig. 4. Simultaneous processing of Dna and mRna [20]

*m*Rna *decay.* Instability is a second decisive property of mRna molecules, that undergo degradation after fractions of a minute to half an hour. When mRna was discovered, instability was its defining feature [7, 25]. Degradation is performed by the *degradosome*, which comprises several enzymes and their respective actions [10, 26]. The decisive step is the initial access to the 5′ end of mRna, which competes with translation initiation as Fig. 3 illustrates.

Proteins are the most prominent active constituents of a cell. In brief, proteins carry out instructions that are hard-wired in Dna. They can be enzymes that catalyze reactions, receptors sitting on the cell's outer membrane and conferring information about the environment to the inside, signaling molecules that carry on information within the cell, transcription factors that control gene expression through binding to Dna, or others. All proteins are subject to degradation, their half-lives usually exceed those of mRna.

Concurrent features of gene expression. Several features of gene expression have a flavor of concurrency. The first is simultaneous processing of the same macromolecule by a number of molecular actors. The second are interdependencies or *couplings* between different phases of gene expression, that are not yet visible in the simple scheme of Fig. 1. The third is immediate competition for a resource, as the race for mRna by ribosomes and the degradosome.

$$\text{RNAP} + P \rightarrow_{k_{on}} (\text{RNAP} \cdot P)_{\text{closed}} \quad (2.1)$$

$$(\text{RNAP} \cdot P)_{\text{closed}} \rightarrow_{k_{off}} \text{RNAP} + P \quad (2.2)$$

$$(\text{RNAP} \cdot P)_{\text{closed}} \rightarrow_{k_{init}} (\text{RNAP} \cdot P)_{\text{open}} \quad (2.3)$$

$$(\text{RNAP} \cdot P)_{\text{open}} \rightarrow_{k_{elong}} \text{RNAP} \cdot \text{DNA}_1 \quad (2.4)$$

$$\text{RNAP} \cdot \text{DNA}_n \rightarrow_{k_{elong}} \text{RNAP} \cdot \text{DNA}_{n+1} \quad (2.5)$$

$$\text{RNAP} \cdot \text{DNA}_{\text{terminator}} \rightarrow_{k_{elong}} \text{RNAP} + \text{DNA}_{\text{terminator}} + m\text{RNA} \quad (2.6)$$

Table 1. Equations for transcription of DNA

The macromolecules DNA and mRNA are *typically processed by multiple actors at the same time*. Bacterial genomes contain several thousand genes, many of which can undergo transcription at any instant. In addition, each gene can simultaneously be transcribed by several RNAP. This is visible in Fig. 4(a), which shows a structure reminiscent of a comb. Its backbone is formed by a stretch of DNA encoding one gene. We can not discern the RNAP themselves, but their products. The comb's teeth are formed by these nascent RNA transcripts. Transcription initiates at the left, elongation has a left-to-right orientation, as indicated by the increasing lengths of transcripts. Its end point is easy to recognize: the non-coding stretch of DNA remains naked. Note that transcription of this gene initiates with high efficiency, thus the RNAP densely follow each other. Figure 4(b) shows the analogous phenomenon in translation. While the mRNA itself can not be seen, the visible blobs are ribosomes.

Coupling between phases of gene expression. Unlike in eukaryotes where they are separated in time and space, transcription and translation are contemporaneous in bacteria. While one end of mRNA molecule is being elongated by RNAP, ribosomes start accessing the other end. The coupling between transcription and translation can become very tight, and fulfill specific goals [24]. Transcriptional attenuation is a regulatory mechanism in which transcription stops in the case of low efficiency in translation of the growing mRNA [72]. The complexity of couplings further increases in higher organisms [36, 37].

2.2 DNA and transcription

The summary of discrete interactions between DNA and RNAP in Table 1 constitutes the basis for our later π-calculus representation [38, 39]. It also comprises the parameters for quantitative control necessary to reproduce the dynamics of transcription in stochastic simulation.

Equations (2.1) to (2.3) represent the three essential steps of transcription initiation [39], omitting intermediary ones that are not fully characterized: initial reversible binding of RNAP to promoter DNA (P), followed by transition to the open complex. The parameter k_{on} in equation (2.1) indicates the speed at which RNAP scans the DNA, recognizes and binds to arbitrary promoters. The *closed*

complex formed hereby is reversible, its stability is reflected by the promoter specific k_{off} in (2.2). In successful initiation RNAP unwinds the duplex DNA locally, and reaches an *open* complex (2.3). This requires a promoter specific effort reflected by the parameter k_{init}. We will later report parameter ranges.

Regulation of initiation.[1] Bacteria apply various strategies to use their genetic material with great effectiveness, in the correct amount and at the appropriate time [5]. Transcription initiation is controlled by DNA binding proteins. Repressors exclude RNAP from promoters by stable binding to overlapping sequences. Activators conversely attach in the vicinity of the promoter, and favor initiation by increasing the transition rate to the open complex k_{init}, or stabilize RNAP by lowering k_{off}.

Elongation: After a successful transition to the open complex (2.3), RNAP starts to transcribe information content from DNA into RNA, at a first coding nucleotide (2.4). In *elongation* (2.5), it continues the synthesis of RNA complementary to the template DNA strand. Only in 2005 experiments provided evidence for an assumption that had for long remained under debate [1]: RNAP elongates RNA, while it advances over individual nucleotides by discrete steps, with an exponential distribution of waiting times determined by by parameter k_{elong}. Elongating RNAP can stall, slow down and pause on certain sequences [71]. Another detail is that the promoter becomes available for further binding only after RNAP has cleared the length of its own footprint (a few tens of nucleotides). This *promoter clearance* delay becomes rate-limiting at highly efficient promoters [28], such as the one from Figure 4(a).

Termination: Following a common view, which omits intermediary steps, RNAP dissociates from DNA as it recognizes a *terminator* sequence on the template, and releases the completed transcript. Equation (2.6) summarizes this *intrinsic* termination. The simplifying notation in Tab. 1 does not explicitly track the growth of the nascent RNA molecule.

A detail not considered here is that under certain circumstances, small molecules can load on elongating RNAP, and cause it to overrun intrinsic terminators. This is referred to as *anti-termination*, and can be explained by a more detailed model of intrinsic termination. An alternative mechanism is so called *rho-dependent* termination [4, 27], where a small protein slides along the transcript starting from its $5'$ end, reaches RNAP and causes it to terminate. We will not cover this mechanism either.

2.3 mRNA, translation and degradation

The flow of mRNA encoded information into proteins is again organized in three phases, and summarized by Table 2. Equation (2.7) represents the initial step of ribosome binding and assembly on a dedicated mRNA sequence, the *r*ibosomal *b*inding *s*ite. As depicted in Fig. 3, the RBS is located close to the mRNA's $5'$

[1] For completeness we sketch principles of regulation, which we do not cover in this work. In a previous case study [34] we elaborated on two well characterized bacterial promoters in the stochastic π-calculus [52].

$$m\text{RNA}_{RBS} + Ribosome \rightarrow_{k_{on}} m\text{RNA}_{RBS} \cdot Ribosome \quad (2.7)$$

$$m\text{RNA}_{RBS} \cdot Ribosome \rightarrow_{k_{off}} m\text{RNA}_{RBS} + Ribosome \quad (2.8)$$

$$m\text{RNA}_{RBS} \cdot Ribosome \rightarrow_{k_{init}} m\text{RNA}_1 \cdot Ribosome \quad (2.9)$$

$$m\text{RNA}_n \cdot Ribosome \rightarrow_{k_{elong}} m\text{RNA}_{n+1} \cdot Ribosome \quad (2.10)$$

$$m\text{RNA}_{\text{terminator}} \cdot Ribosome \rightarrow_{k_{elong}} m\text{RNA}_{\text{terminator}} + Ribosome + Protein \quad (2.11)$$

$$m\text{RNA}_{RBS} + Degradosome \rightarrow_{k_d} m\text{RNA}_{RBS} \cdot Degradosome \quad (2.12)$$

Table 2. Equations for translation and and degradation of mRNA

end. The ribosome may dissociate readily (2.8). Its stability k_{off} depends on the agreement with an ideal sequence.

The abbreviation RNA_{RBS} in Tab. 2 refers to the 5′ end of mRNA, including both the RBS and the start signal where translation initiates with an efficiency k_{init} in (2.9), that depends on the actual start signal. The ribosome then slides over mRNA (2.10), reads out information content and assembles a growing chain of amino acids. Unlike transcription that maps individual nucleotides, translation proceeds in three letter words over mRNA (*codons*), which each determine one amino acid. Illustrative comics are widespread in the biological literature [20], nevertheless the detailed internal functioning of ribosomes is only partially understood [21]. Elongation ends as the ribosome reaches a dedicated *terminator* signal on mRNA (2.11).

Degradation: Table 2 covers the initial step of degradation by (2.12). After this, decay proceeds with the same net orientation as translation, and does not affect ribosomes that have already initiated. This scheme approximates the net outcome of multiple degradation pathways in a phenomenological manner [11, 26, 66]. For long, the detailed understanding of mRNA degradation lagged behind that of other steps in gene expression; this is now changing rapidly [10, 60].

2.4 Quantitative control of gene expression

Part of the variability in gene expression originates from the inherently stochastic nature of the biochemical reactions involved, combined with low numbers of molecules in regulatory events [65]. Other effects are due to the specific quantitative control for a given gene of interest. In Tables 1 and 2 we met its parameters as reaction labels k_{on}, k_{off}, k_{init} and k_{elong}. We consider ranges of values in Table 3, and discuss their impact.

The quantitative properties of promoters vary greatly. On some RNAP falls off the closed complex within fractions of seconds, while on such with favorable k_{off} parameter, it may remain stably bound for minutes. Transition to the open complex (2.3) occurs within a second at strong promoters, at weak ones after minutes, depending on their k_{init} parameter. Thus, the frequency of transcription per gene varies from one per second (ribosomal RNA) to one per cell

parameter	value	comment
Transcription of DNA		
k_{on}	0.1 sec^{-1}	binding: equally fast for all promoters
$\frac{k_{on}}{k_{off}}$	10^6 to 10^9	unbinding: promoter specific
k_{init}	10^{-3} to 10^{-1} sec^{-1}	initiation: promoter specific
k_{elong}	$\frac{1}{30}$ sec^{-1}	elongation speed: 30 nucleotides/sec
Translation and degradation of mRNA		
$\frac{k_{on}}{k_d}$	1 to 100	gene specific *mean* protein crop per transcript
k_{elong}	$\frac{1}{100}$ sec^{-1}	elongation speed: 100 nucleotides/sec
mRNA lifetime	few sec to 30 min	

Table 3. Quantitative control of gene expression

generation (certain regulatory proteins). RNAP elongates transcripts at around 30 nucleotides per second. This combines to an average transcript elongation delay of roughly 30 seconds, with an average gene length of 1000 nucleotides.

For completeness, we recall that RNAP's access to promoters can be hindered by repressor proteins, bound to DNA. A repressor protein can stick to highly specific sequences for several bacterial generations of each 30 min – 1 hour, while falling off less specific sequences after a few seconds.

Translation proceeds faster than transcription, such that the average time required for the translation of a protein from a mRNA is in the order of 10 seconds. Note that degradation can start before a first protein has been completed from an mRNA, and that a ribosome bound to the RBS protects mRNA from decay until it either unbinds or dissociates.

2.5 Translational bursting

The *average* number of proteins produced from a single mRNA is gene specific, typical ranges are between 1 and 100. Nevertheless there are important *fluctuations* of protein crops, even for transcripts of the same gene, determined by the race between degradation and translation. When translation initiates efficiently, and the crop for the transcript is high, ribosomes queue on mRNA, all proteins are released soon after transcript completion. With long spacings between transcriptions, high crops result in *translational bursts* in which the number of proteins rapidly increases. After this for a while very few proteins are made, until the next burst occurs.

Details. Let p be the probability that translation succeeds in one round, over degradation that has a probability of $(1-p)$. Considering several rounds of this race, the probability to produce x proteins from one transcript before it is degraded is given by $p^x(1-p)$: with a probability of p, translation succeeds for each of n rounds, and then degradation wins with the complementary probability of $(1-p)$. This is a geometric distribution function, which is characterized by asymmetry and many large values. Figure 5 illustrates the complementary

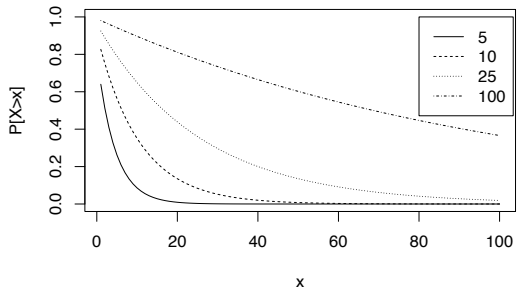


Fig. 5. A *geometric distribution* characterizes the fluctuations around the mean crop of proteins per mRNA. $P[X > x]$ for different mean values.

cumulative distribution function for geometric distributions with different mean values. It indicates the probability to obtain more than x transcripts from one transcript, $P[X > x]$. For example, if the mean crop per transcript is 10, 9% of the transcripts yield each over 25 proteins.

Translational bursting is frequent. Only a minority of bacterial genes yield averages of fewer than 5 proteins per transcript, a value of 20 is rather normal, and burst sizes increase with these means. Combined with the fact that most genes are only transcribed occasionally, translational bursting becomes prevalent; and significantly contributes to stochasticity in gene expression, which has attracted much attention in recent years [30, 56]. It explains why two cells with identical genetic material, under the same conditions, can exhibit significantly variable individual behavior. The effects can propagate up to the level of population of cells, which are partitioned into sub-populations with externally distinct characteristics. While these consequences have been known for long, the origins have have only become observable recently through real time courses of levels in proteins and mRNA [23, 33].

2.6 Transcriptional organization in bacteria

We now sketch specific cases in the arrangement of genes and promoters in bacterial genomes. They have important impact on expression patterns, and are difficult to explicitly represent in previous modeling approaches. We will discuss them within ours.

Operons: In bacteria, sequences of several genes are typically co-transcribed in one go from a common promoter. Figure 6(a) presents such an *operon*; operons yield *polycistronic* mRNA molecules in which each *cistron* codes for a different protein, bears its own ribosomal binding site and translation start signal. The translational efficiency can vary up to a factor of 1000 across cistrons on the

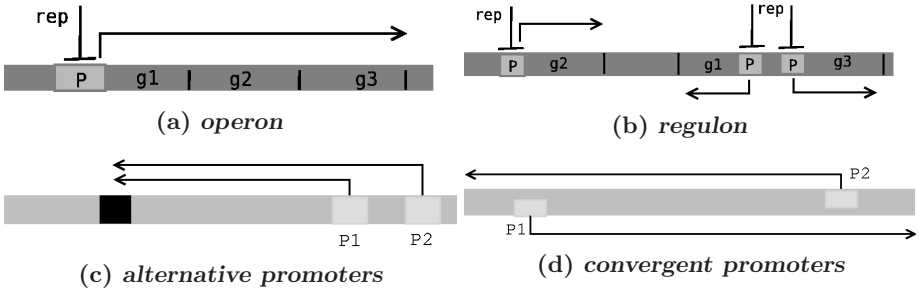


Fig. 6. Particular cases of promoter arrangements

same mRNA [58]. Operons eliminate the need for multiple promoters subject to the same regulatory signal, called *regulon* and illustrated in Fig. 6(b). The proteins encoded by the operon are made available at the same time, even if in different quantities.

Alternative promoters for one gene sketched in Fig. 6(c) offer two interesting regulatory strategies in bacteria. Alternative promoter can be activated independently, depending on different environmental signals. Second, alternative transcripts of the same gene bear different $5'$ ends, where both translation and degradation initiate. The longer transcript is likely to contain a second ribosomal binding site, or translation start signal, and be stabler due to secondary structures into which its $5'$ end folds. Both factors allow to further tune protein crops. Alternative promoters can best be observed for ribosomal RNA genes in bacteria [16, 46], which account for 90% of transcription in rapidly growing cells. The aspect of subsequent tunable translation control is relevant for viruses that infect bacteria, taking the decision to either enter their dormant state, or start multiplying at the expense of their host cell [62].

Convergent promoters. Two transcriptional units on opposing strands sometimes overlap within a segment of DNA. In this case transcription starts from *converging promoters*. As illustrated in Fig. 6(d), two RNAP can then proceed over the two strands with converging orientations. However transcriptional traffic over DNA occurs on a *single lane, two way street* [64]. Head-on collisions between two RNAP causes at least one participant to fall off DNA, releasing a truncated transcript. This suppressive influence is known as *transcriptional interference* [64].

3 Stochastic Pi Calculus with Pattern Guarded Inputs

We recall the stochastic π-calculus with pattern guarded inputs [35]. Pattern guarded inputs allow to express concurrent objects in the π-calculus, as already noticed by Vasconcelos [47, 68]. The stochastic semantics is induced by Gillespie's algorithm.

Processes	$P ::= P_1 \mid P_2$	parallel composition
	$\mid$ **new** $x(\rho).P$	channel creation
	$\mid C_1 + \ldots + C_n$	choice ($n \geq 0$)
	$\mid A(\overline{x})$	process application
Guarded processes	$C ::= x?f(\overline{y}).P$	pattern guarded input
	$\mid x!f(\overline{y}).P$	tuple guarded output
Definitions	$D ::= A(\overline{y}) \triangleq P$	

Table 4. Syntax of the stochastic π-calculus with pattern guarded inputs

3.1 Process expressions and reduction semantics

We construct π-calculus expressions starting from a vocabulary, that consists of an infinite set of *channel names* $\mathscr{N} = \{x, y, z, \ldots\}$, an infinite set of *process names* A, and an infinite set of *function symbols* $f \in \mathscr{F}$. The vocabulary fixes arities, i.e. numbers of parameters for every process name A and for every function symbol f. Our vocabulary additionally comprises functions $\rho : \mathscr{F} \to]0, \infty]$ which define collections of *stochastic rates* associated to channels. If ρ is assigned to channel x and $f \in \mathscr{F}$ is a function symbol, then $\rho(f)$ is the rate of the pair (x, f).

The syntax of our π-calculus is defined in Table 4. We write $\overline{x}$ for finite, possibly empty sequences of channels $x_1, \ldots, x_n$ where $n \geq 0$. Whenever we use tuples $f(\overline{x})$ or terms $A(\overline{x})$ we assume that the number of arguments (the length of $\overline{x}$) is equal to the respective arity of f or A. Process expressions are ranged over by P. Let us define the *free channel names* of all processes P and guarded processes C by induction over the structure of such expressions:

$$fv(x?f(\overline{y}).P) = \{x\} \cup (fv(P) - \{\overline{y}\}) \qquad fv(P_1 \mid P_2) = fv(P_1) \cup fv(P_2)$$
$$fv(x!f(\overline{y}).P) = \{x\} \cup fv(P) \cup \{\overline{y}\} \qquad fv(\textbf{new } x(\rho).P) = fv(P) - \{x\}$$
$$fv(C_1 + \ldots + C_n) = fv(C_1) \cup \ldots \cup fv(C_n)$$

The only atomic expression (that cannot be decomposed into others) is the guarded choice of length $n = 0$, that we write as $\mathbf{0}$. The expression $P_1|P_2$ denotes the parallel composition of processes P_1 and P_2. A term **new** $x(\rho).P$ describes the introduction of a new channel x taking scope over P; the rate function ρ fixes stochastic rates $\rho(f)$ for all pairs (x, f) where $f \in \mathscr{F}$. We can omit rate functions ρ in the declaration of a channel x if all reactions on x are instantaneous, i.e. $\rho(f) = \infty$ for all $f \in \mathscr{F}$. An expression $A(\overline{x})$ applies the definition of a parametric process A with actual parameters $\overline{x}$.

A guarded choice $C_1 + \ldots + C_n$ offers a choice between $n \geq 0$ communication alternatives $C_1, \ldots, C_n$. A guarded input $x?f(\overline{y})$ describes a communication act, ready to receive a tuple that is constructed by f over x. The channels $\overline{y}$ in input guards serve as pattern variables; these are bound variables that are replaced by the channels received as input. An output guarded process $x!f(\overline{y}).P$ describes a

$(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3) \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P \mid \mathbf{0} \equiv P$
$\ldots + C_1 + C_2 + \ldots \equiv \ldots + C_2 + C_1 + \ldots$
$\mathbf{new}\ x_1(\rho_1).\mathbf{new}\ x_1(\rho_2).P \equiv \mathbf{new}\ x_2(\rho_2).\mathbf{new}\ x_1(\rho_1).P$
$\mathbf{new}\ x(\rho).(P_1 \mid P_2) \equiv P_1 \mid \mathbf{new}\ x(\rho).P_2,$ if $x \notin fv(P_1)$
capture free renaming of bound variables (α-conversion)

Table 5. Axioms of the structural congruence

communication act willing to send tuple $f(\overline{y})$ over channel x and continue as P. Here, the channels $\overline{y}$ are data values, i.e. free.

A definition of a parametric process has the form $A(\overline{x}) \triangleq P$, where A is a process name, and $\overline{x}$ is a sequence of (universally bound) channels that we call the formal parameters of A. We assume that definitions do not contain free variables. This means for all definitions $A(\overline{x}) \triangleq P$ that $fv(P) \subseteq \{\overline{x}\}$[2].

We define the operational semantics of the π-calculus in terms of a binary relation over expressions, called (one step) reduction. The definition relies on the usual structural congruence between expressions. Reduction steps on an expression can be performed on arbitrary congruent expressions.

Structural congruence is the smallest relation induced by the axioms in Table 5. It identifies expressions modulo associativity and commutativity of parallel composition, i.e. the order in $P_1 | \ldots | P_n$ does not matter, order independence of alternatives in choices, scope extrusion. We also assume α-conversion. Unless this captures free variables, we can rename variables that are bound by a pattern input or **new**.

Table 6 defines the *reduction relation*. The first axiom tells how to interpret choices; it comprises channel communication and pattern matching. It applies to two complementary matching alternatives in parallel choices, an output alternative $x!f(\overline{y}).P_1$ willing to send a term $f(\overline{y})$ and an input pattern $x?f(\overline{z}).P_2$ on the same channel x, of which the pattern matches in that it is built using the same function symbol f. The reduction cancels all other alternatives, substitutes the pattern's variables $\overline{z}$ by the received channels $\overline{y}$ in the continuation P_2 of the input, and reduces the result in parallel with the continuation of the output P_1. Note that *only matching tuples can be received over a channel.* Other sending attempts have to suspend until a suitable input pattern becomes available. This fact will prove extremely useful for concurrent modeling. Upon reception, tuples are immediately decomposed.

The unfolding axiom applies one of the definitions of the parametric processes in a given set Δ. An application $A(\overline{y})$ reduces in one step to definition P, in which the formal parameters $\overline{y}$ have been replaced by the actual parameters $\overline{x}$. Note

[2] In modeling practice, global parameters are quite useful, but not essential. They correspond to free variables in definitions that are introduced by the reduction context, while fixing their stochastic rates.

Communication, choice, pattern matching:

$$x!f(\overline{y}).P_1 + \ldots \mid\ x?f(\overline{z}).P_2 + \ldots \quad \rightarrow \quad P_1 \mid P_2[\overline{z} \mapsto \overline{y}] \qquad \text{if } \overline{z} \text{ free for } \overline{y} \text{ in } P_2$$

Unfolding of parametric processes:

$$A(\overline{x}) \quad \rightarrow \quad P[\overline{y} \mapsto \overline{x}] \qquad \text{if } A(\overline{y}) \triangleq P \text{ in } \Delta, \text{ and } \overline{y} \text{ free for } \overline{x} \text{ in } P$$

Closure rules:

$$\frac{P \equiv P' \quad P' \rightarrow Q' \quad Q \equiv Q'}{P \rightarrow Q} \qquad \frac{P \rightarrow P'}{\mathbf{new}\ c(\rho).P \rightarrow \mathbf{new}\ c(\rho).P'} \qquad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$$

Table 6. Reduction relation for a finite set of definitions Δ

that parametric definitions can be recursive, and that another call of A may be contained in P. The usual closure rules state that reduction can be applied in arbitrary contexts, but not below choices or in definitions.

As in Milner's polyadic π-calculus, we can communicate sequences of names over a channel. It is sufficient to fix n-ary functions symbols f for all $n \geq 0$, in order to wrap name sequences of length n. When defining $x?\overline{y}.P =_{\text{def}} x?f(\overline{y}).P$ and $x!\overline{y}.P =_{\text{def}} x!f(\overline{y}).P$ we obtain the usual reduction step of the polyadic π-calculus:

$$x!\overline{y}.P_1 \mid x?\overline{z}.P_2 \quad \rightarrow \quad P_1 \mid P_2[\overline{z} \mapsto \overline{y}]$$

3.2 Example: semaphore

Semaphores control the access to shared resources in concurrent systems [17]. They are widespread in programming languages, operating systems, or distributed databases. Their purpose lies in restricting the access to some resource, to a single user at a time. We will apply them to grant exclusive access to molecular binding sites. We consider simple semaphores with two states - free and bound. Importantly, any binding attempt on a bound semaphore has to wait until the semaphore has become free again.

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

Consider the reduction sequence of the following process expression, of a bound semaphore, located at site s, in parallel with a bind request and a free request.

```
    Semaphore_bound(s) | s!bind().0 | s!free().0
→   s?free().Semaphore_free(s) | s!bind().0 | s!free().0
→   Semaphore_free(s) | s!bind().0
→   s?bind().Semaphore_bound(s) | s!bind().0
→   Semaphore_bound(s)
```

In a first step, the Semaphore_bound at s unfolds its definition. This creates an input offer on s to receive a free message. Other messages cannot be received over s in this state, in particular no bind requests. Hence, the site s cannot get

```
1 module 'semaphore'
2 export
3    Semaphore with bind/0, free/0
4 define
5    Semaphore(me) ≙ Semaphore_free(me)
6    Semaphore_free(me) ≙ me?bind().Semaphore_bound(me)
7    Semaphore_bound(me) ≙ me?free().Semaphore_free(me)
```

Fig. 7. Semaphore

bound a second time. Only once the free message got received, s was able to accept the next bind request, while becoming bound again.

3.3 Stochastic scheduling

In order to apply the Gillespie algorithm to the π-calculus with pattern guarded inputs, we have considered a π-calculus expression with a chemical solution, and all instances of the reduction rules of the π-calculus as chemical reaction rules [35].

An abstract chemical expression is a multiset of molecules. In the set of the π-calculus, a molecule will be either a choice $C_1 + \ldots + C_n$ or an application $A(\overline{y})$, or more precisely, a congruence class of such an expression with respect to the structural congruence. A parallel composition without *new* binder modulo structural congruence is most naturally identified with a chemical solution, i.e. a multiset of such molecules.

The *new*-binder assigns a rate function ρ_x to all channels x in a closed expression and can be ignored otherwise. By appropriate renaming, this assignment can be defined globally for all channel names. The chemical reduction rules are instances of the reduction rules of the π-calculus relative to some set of definitions Δ. A communication on channel x with pattern function f is assigned the rate $\rho_x(f)$. Applications of definitions are immediate, i.e. have rate ∞.

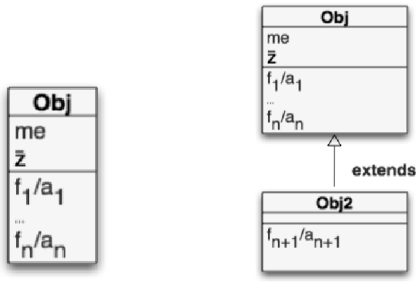
$$x!f(\overline{y}).P_1 + \ldots \mid \; x?f(\overline{z}).P_2 + \ldots \quad \rightarrow^{\rho_x(f)} \quad P_1 \mid P_2[\overline{z} \mapsto \overline{y}]$$

$$A(\overline{x}) \quad \rightarrow^{\infty} \quad P[\overline{y} \mapsto \overline{x}] \qquad \text{if } A(\overline{y}) \triangleq P \text{ in } \Delta$$

The Gillespie algorithm [22] thus defines a scheduling for our π-calculus, including the delays for all steps.

3.4 Modules

We use a simple module system on an informal level, inspired by that of SML. Each module contains a set of definitions and declarations of process names and function symbols. Modules may export some, but not necessarily all names of defined processes, and import definitions from other modules.



(a) ***object*** **(b)** ***inheritance***

Fig. 8. Obj2 extends Obj with the a_{n+1}-ary function f_{n+1}

The module 'semaphore' (Fig. 7) exports a single process named Semaphore, which may receive tuples built from two 0-ary function symbols bind and free. The process names Semaphore_free and Semaphore_bound are not exported for external usage, they remain local to the implementation inside the module. Note that the rates of function calls for each Semaphore are determined by the ρ of the channel it is instantiated at.

4 Concurrent objects

In this section, we introduce the expression of multi-profile objects in the stochastic π-calculus with pattern guarded choices. These object-oriented abstractions accompanied by a notion of inheritance are one of the central reasons for the extensibility of the model of transcription and translation presented in the current work. Note that previous π-calculus based approaches to biomolecular modeling do not use object-orientation.

Objects with multiple profiles are a recent notion [18]. Multiple profiles enable objects to change their interface, i.e. the set of functions offered. Based on multi-profile objects, we show how to model persistent and degradable lists, with or without queueing discipline. In Sec. 5 we will refine such lists to models of DNA and RNA. More traditional concurrent objects can be expressed in a closely related variant of the π-calculus, as already noticed by Vasconcelos in the beginning of the nineties [68].

4.1 Single-profile objects

A concurrent object resembles a server that offers a set of functions to its clients. The *interface* of such an object is specified by a finite set of function names. The *class* of an object defines functions for each of the function names in the interface.

In the π-calculus with pattern guarded choices, we represent objects as follows. The names of object functions correspond to the function names of the

π-calculus; the class of the object is a parametric process name, say Obj. Every object of this class is represented by an application Obj(me,$\bar{z}$) of the class to a channel me identifying the object and a sequence of parameters $\bar{z}$. Its interface is defined by a choice of input offers on channel me, each guarded by one of the functions of the object:

$$\begin{aligned} &\mathsf{Obj}(\mathsf{me},\bar{z}) \triangleq \\ &\quad \mathsf{me}?\,\mathsf{f}_1(\overline{x_1})\,.\,\mathsf{P}_1 \\ &+ \ \ldots \\ &+ \ \mathsf{me}?\,\mathsf{f}_n(\overline{x_n})\,.\,\mathsf{P}_n \end{aligned}$$

Upon reception of some message matching $\mathsf{f_i}(\overline{x_i})$, the object replaces the formal parameters $\overline{x_i}$ in $\mathsf{P_i}$ by the actual parameters, and continues as the resulting process. Note that $\mathsf{P_i}$ may further depend on the parameters $\bar{z}$ of the object, and on global names left free in the object's definition. In order to continue their service, most objects go into recursion; sometimes they terminate as the idle process 0.

We will frequently extend object classes by new functions in order to define new refined objects, i.e. a newly defined class inherits the functions of an existing one. Figure 8(b) illustrates. Class definitions by inheritance are always compiled into regular π-calculus definitions before execution. Let class Obj be defined by the following choice:

$$\mathsf{Obj}(\mathsf{me},\bar{z}) \triangleq \mathsf{C}_1 + \ldots + \mathsf{C}_n$$

Inheritance refines this class to Obj2 by adding further choices:

$$\mathsf{Obj2}\ \textbf{extends}\ \mathsf{Obj}$$

$$\mathsf{Obj2}(\mathsf{me},\bar{z})\ \textbf{extended by}\ \mathsf{C}_{n+1} + \ldots + \mathsf{C}_m$$

This definition by inheritance can be resolved as follows:

$$\mathsf{Obj2}(\mathsf{me},\bar{z}) \triangleq \mathsf{C}_1 + \ldots + \mathsf{C}_m\ [\mathsf{Obj} \mapsto \mathsf{Obj2}]$$

The latter substitution means that all recursive calls to some Obj are renamed into recursive calls to Obj2. Note that according to our definition, only if the added choice is an input offer over channel me, the resulting Obj2 is in turn an object.

4.2 Examples: persistent and degradable lists

A persistent *list* consists of a sequence of nodes, each having a successor next and a value val. It is defined in Fig. 9 by concurrent objects of the class Node with three parameters: me, next, and val. The successor of a list's last Node is represented by a Nil object. Each Node object has three unary functions getNext, getValue, and isNil. The Nil object provides the unary function isNil only.

Consider a list [a,b] of length 2. It is represented by the parametric process, using the module 'persistent list'.

```
1  module 'persistent list'
2  export
3     Node with getNext/1,getValue/1,isNil/1
4     Nil  with isNil/1
5  define
6     Node(me,next,val) ≜
7        me?getNext(c) .c!next.Node(me,next,val)
8      + me?getValue(c).c!val.Node(me,next,val)
9      + me?isNil(c)    .c!false().Node(me,next,Cal)
10    Nil(me) ≜ me?isNil(c).c!true().Nil(me)
```

Fig. 9. Persistent list

```
module 'persistent list [a,b]'
import Node Nil from 'persistent list'
export List
define
   List(n1) ≜ new n2(ρ).new nil.
      Node(n1,n2,a) | Node(n2,nil,b) | Nil(nil)
```

The rate function ρ determines the temporal behavior of the second node of the list, located at n2. We fix it by setting ρ(getNext)=30. We illustrate list processing with a Walker. It traverses the list by querying each node for it successor via getNext, and stops after identifying Nil by its positive response to isNil.

```
Walker(node) ≜ new c1.node!isNil(c1).
     c1?true().0
+    c1?false().new c2.node!getNext(c2).c2?next.Walker(next)
```

The attentive reader may have noticed a detail of our Walker process. It is actively sending requests to objects, which keep *waiting* for input over their respective me channels. The same holds for all devices processing representatives of macromolecules (i.e., data structures) in the remainder of this paper: the RNAP and ribosome abstractions proceed by *calling* functions[3] that are *offered* by DNA and mRNA representatives.

After importing module 'persistent list [a,b]', we let the Walker run over our example list [a,b]:

```
List(n1) | Walker(n1)
→ new n2(ρ).new nil. Walker(n1) | Nodes
     where Nodes = Node(n1,n2,a) | Node(n2,nil,b) | Nil(nil)
→* new n2(ρ).new nil.Walker(n2) | Nodes
→* new n2(ρ).new nil.Walker(nil) | Nodes
→* new n2(ρ).new nil.Nodes
=  List(n1)
```

[3] When emulating a function call in the π-calculus, we pass a fresh private channel on which the result comes back [41], see the Walker's c_1 and c_2.

```
1 module 'degradable list'
2 import Plist(Node, Nil) from 'persistent list'
3 export
4    Node extends Plist.Node by kill/0
5    Nil  extends Plist.Nil  by kill/0
6 define
7    Node(me, val, next) extended by me?kill().0
8    Nil(me) extended by me?kill().0
```

Fig. 10. Degradable list

Suppose the first node n1 was introduced with rates ρ as well. All calls to getNext functions are then associated with a stochastic rate of 30. Given that this is the single parameter determining an exponential distribution of waiting times [4], our Walker traverses lists at an average speed of 30 nodes per second. Besides merely running down the list, the Walker does not perform any further action. We leave it to the reader to extend the Walker into a Copier such that:

$$\mathsf{List(n1)} \mid \mathsf{Copier(n1,n2)} \rightarrow^* \mathsf{List(n1)} \mid \mathsf{List(n2)}$$

Degradable lists. The distinguishing feature between a degradable and persistent list is that the former can be destroyed. We define degradable lists by inheritance from the persistent in Fig. 10. The import statement in line 2 imports the specifications of Node and Nil from the module 'persistent lists', which it refers to as Plist. With this, Plist.Node and Plist.Nil denote the respective objects of persistent lists, clearly distinguished from the corresponding objects in non-persistent lists. The export statement tells that this module provides definitions for Node and Nil. These objects provide the same functions as their analogs from the 'persistent list' module, and additionally kill of arity zero.

A non-persistent list [a,b] can now be built by the same definition as a persistent list [a,b]. The only difference is that we have to import module 'non-persistent list' instead of 'persistent list'. Destructing a non-persistent list is easy. A Killer proceeds like a Walker, except that it kills a Node before continuing with the next:

```
Killer(node) ≜ new c1.node!isNil(c1).
    c1?true()  .node!kill().0
 +  c1?false().new c2.node!getNext(c2). c2?next.node!kill().
     Killer(next)
```

It is worthwhile observing that Walkers are able to traverse non-persistent lists, without changing their code. This is one of the main advantages of the object-oriented approach proposed in this work. Objects of non-persistent lists specialize those of persistent lists, so we can always replace the latter by the former. This would not hold for our model encoded in the biochemical stochastic π-calculus [52].

[4] The inverse of its *mean*, in units of seconds.

4.3 Multi-profile objects

A multi-profile object is a collection of objects, that may recursively depend on another [18]. In this paper, we need the concept of multi-profile objects with an appropriate notion of inheritance (a topic left open by previous work). We group objects into multi-profile objects by a naming convention. We assume object names Obj and profile names p such that composed names Obj_p belong to the set of parametric process names. The class of multi-profile object Obj with profiles p_1, ..., p_n is defined by a collection of classes Obj_p_1, ..., Obj_p_n:

$$\begin{aligned} Obj\_p_1(me, \overline{z_1}) &\triangleq C^1_1 + \ldots + C^1_{n_1} \\ &\ldots \\ Obj\_p_n(me, \overline{z_n}) &\triangleq C^n_1 + \ldots + C^n_{n_n} \end{aligned}$$

We have already seen an example of a multi-profile object, the semaphore from Sec. 3.4 with profiles free and bound. We can extend a class Obj into another Obj2 by adding new functions to some or all profiles:

$$\begin{aligned} &Obj2 \ \textbf{extends} \ Obj \\ &Obj2\_p_1(me, \overline{z_1}) \ \textbf{extended by} \ C^1_{n_1+1} + \ldots + C^1_{m_1} \\ &\quad \ldots \\ &Obj2\_p_n(me, \overline{z_n}) \ \textbf{extended by} \ C^n_{n_n+1} + \ldots + C^n_{m_n} \end{aligned}$$

This definition by inheritance can be resolved into the following π-calculus definition:

$$\begin{aligned} Obj2\_p_1(me, \overline{z_1}) &\triangleq C^1_1 + \ldots + C^1_{m_1} \ [Obj \mapsto Obj2] \\ &\ldots \\ Obj2\_p_n(me, \overline{z_n}) &\triangleq C^n_1 + \ldots + C^n_{m_n} \ [Obj \mapsto Obj2] \end{aligned}$$

The latter substitution renames all recursive calls to some profile Obj_p_i into recursive calls to $Obj2\_p_i$ for $1 \leq i \leq n$.

4.4 Examples: persistent and degradable queueing lists

The persistent and degradable lists presented so far can be traversed by several visitors at the same time, e.g. by a Walker and by a Reader. Each visitor proceeds over the nodes, and hereby draws waiting times independently of the others. This permits overtaking of one visitor by another, which is however not possible in transcription of DNA, nor in translation of mRNA or its degradation.

We impose *queueing* on visitors of a *persistent* list by incorporating a semaphore style behavior, distinguishing two profiles of Nodes (Fig. 11): free and bound. The functions exported as the interface can only be used as the Node is in profile bound. It becomes impossible to bind a node twice. Function bind is not exported outside the module, it can only be called on a node by its predecessor. This is implemented by re-defining the getNext function, compared to our previous list module. Upon a getNext request a Node binds its successor - if necessary it waits - before passing over the reference next (line 10).

Note that before the Walker can operate on a persistent queueing lists, we need to bind its first node:

```
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) ≜ Node_free(me,next,val)
7     Node_free(me,next,val) ≜
8        me?bind().Node_bound(me,next,val)
9     Node_bound(me,next,val) ≜
10       me?getNext(c).next!bind().c!next.
11            Node_free(me,next,val)
12    + me?getValue(c).c!val.Node_bound(me,next,val)
13    + me?isNil(c).c!false().Node_bound(me,next,val)
14    Nil(me) ≜
15       me?bind().Nil(me)
16    +  me?isNil(c).c!true().Nil(me)
```

Fig. 11. Persistent queueing list

```
1  module 'degradable queueing list'
2  import
3     Plist(Node,Nil) from 'persistent queueing list'
4  export
5     Node extends Plist.Node by kill/0
6     Nil  extends Plist.Nil  by kill/0
7  define
8     Node_bound(me,next,val) extended by me?kill().0
9     Nil(me) extended by me?kill().0
```

Fig. 12. Degradable queueing list

```
   List(head) | head!bind().0 | Walker(head)
→* new n1.new n2. Node_free(head,n1,a) | Node_bound(n1,n2,
     b) | Nil(n2) | Walker(n1)
→* ...
```

A *degradable* queueing list can easily be expressed by inheritance (Fig. 12). Its members are equipped with a kill function for stepwise destruction. As for the Walker, the Killer remains functional starting on a bound first node.

5 Modeling transcription and translation

We now introduce models of bacterial transcription and translation in the stochastic π-calculus with pattern guarded choices. Hereby we make use of modules, inheritance and the modeling techniques introduced in the previous sections.

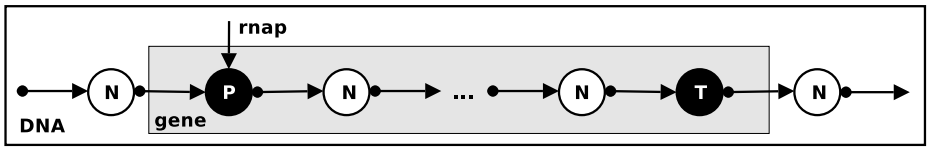


Fig. 13. We abstract DNA as list, extending on the module 'persistent queueing list'. Each list member has the private knowledge of its successor. **P**romoters control the access of RNAP to coding DNA, they can be contacted over the globally visible channel rnap. They abstract the behavior of the promoter DNA sequence as a whole, similarly as **T**erminators, but unlike regular **N**ucleotides.

Objects with multiple profiles are essential to our modeling. We represent multi-step interactions between pairs of molecular actors, as observed in transcription initiation, by synchronized transitions of two interacting multi-profile objects.

5.1 DNA and transcription

Our model covers the following events from Table 1: interactions between RNAP and promoter sequences on DNA during initiation as sketched in equations (2.1) to (2.4), transcript *elongation* in steps of one DNA nucleotide (2.5), and *termination* (2.6) on terminator DNA sequences. Accordingly, we abstract promoter and terminator sequences as one object, with a common interface. In contrast we represent individual nucleotides within the coding region, building on Node from the 'persistent queueing list' module (Fig. fig:persistentQList). Figure 13 distinguishes these abstraction levels by coloring.

Let us precede the discussion of the components with a comment on a design choice. We want to represent the growth of the transcript, which itself is not included in Table 1. Recall that RNA is assembled *on* DNA *by* RNAP. In the π-calculus model, the transcript representative must be spawned by *one*. We attribute the task to the DNA representative, which by its sequence determines the information content of the transcript, and thus its behavior. Due to this choice, our representative of RNAP is simpler to explain than those of DNA.

Rnap has four profiles, listed in Fig. 14. Three deal with transcription initiation – they have corresponding Promoter profiles. The fourth covers elongation, and roughly resembles our previous Walker. We first consider formation of the closed promoter complex, summarized by equation (2.1): Rnap_free invocates bind over the global channel rnap (line 6). It waits for satisfaction by a Promoter, that is nondeterministically selected among several available in profile free, and extrudes its me channel. As the bind interaction succeeds, Rnap and Promoter switch to their closed profiles. They now jointly represent the closed promoter complex. As such they can interact over Promoter's shared me channel, by the competing functions unbind and initiate. The race between these is controlled by the rates k_{init} (2.2) and k_{off} (2.3), which enter the model over the ρ function that quantifies the Promoter's channel me. Unbinding without transcription initiation

```
1  module 'rnap'
2  channel rnap
3  export Rnap
4  define
5   Rnap ≜ Rnap_free
6   Rnap_free ≜ new c.rnap!bind(c).c?prom.Rnap_closed(prom)
7   Rnap_closed(prom) ≜
8      prom!unbind().Rnap_free
9    + prom!initiate().Rnap_open(prom)
10  Rnap_open(prom)≜ new c1.new c2.
11       prom!startTranscript(c1).c1?rna.
12       prom!getNext(c2).c2?dna.
13       Rnap_elongating(dna,rna)
14  Rnap_elongating(dna,rna) ≜ new c1.dna!isTerm(c1).
15       c1?true().dna!elongate(rna).Rnap_free
16    + c1?false().new c2.dna!elongate(rna,c2).c2?rna_nxt.
17          new c3.dna!getNext(c3).c3?dna_nxt.
18           Rnap_elongating(dna_nxt,rna_nxt)
```

Fig. 14. RNAP module

is straightforward. It causes transitions from bound to free (line 8). If conversely initiate succeeds, both switch to their open profile (line 9). Transcription subsequently launches upon a startTranscript call, reflecting (2.4). Promoter_open creates the first transcript segment, and returns a reference to its growing end rna. Rnap_open continues as elongating, using rna and the second parameter dna, that it obtains by a getNext call.

Rnap_elongating traverses the DNA representative, calling elongate on each Nucleotide over its extruded me channel. After reaching the Terminator it returns to Rnap_free (line 15), unlike the previous Walker that terminates as 0 at the end of a list. This behavior corresponds to equations (2.5) and (2.6).

DNA. Our module 'DNA' includes the complementary specification of Promoter, Nucleotide, and Terminator (Fig. 15). Promoter implements the explicit access control for genes. It has three profiles analogous to those of Rnap, with which it synchronizes transitions: Promoter_free is dual to Rnap_free.[5]

Transcription ensues once both Rnap and Promoter are open. As a result of a startTranscript call, Promoter_open spawns a RBS as first chunk of the transcript, and returns its growing end rna to Rnap_open. The transition to Promoter_free is caused by Rnap_open's following call to getNext, inherited from Node.

[5] Note that Promoter_free is not subsumed by our object definition. We obtain it by extension of Node, using not its me channel but the global rnap. While individual objects are addressable over their me, interactions with (some arbitrary) Promoter_free are nondeterministically initiated over rnap, on which *all* such processes listen. This resembles *distributed objects* [57] in TyCO [47, 68], which share a common input channel.

```
1  module 'DNA'
2  channel
3     rnap with bind/1
4  import
5     List(Node, Nil) from 'persistent queueing list'
6     Mrna(Node, Terminator, RBS) from 'mRNA'
7  export
8    Promoter extends List.Node by unbind/0,
9              initiate/0, startTranscript/1
10   Nucleotide extends List.Node by isTerm/1, elongate/2
11   Terminator extends List.Node by isTerm/1, elongate/1
12 define
13   Promoter(me, next) ≜ Promoter_free(me, next)
14   Promoter_free(me, next)  extended by
15      rnap?bind(c).c!me.Promoter_closed(me, next)
16   Promoter_closed(me, next) ≜
17      me?unbind().Promoter_free(me, next)
18    + me?initiate().Promoter_open(me, next)
19   Promoter_open(me, next)  extended List.Node by
20      me?startTranscript(c).
21         new him(ρrbs).new rna(ρrna).c!rna.
22         Promoter_open(me, next) | Mrna.RBS(him, rna)
23   Nucleotide_bound(me, next, v)  extended by
24      me?isTerm(c).c!false().Nucleotide_bound(me, next, v)
25    + me?elongate(rna, c).new rna_nxt(ρrna). c!rna_nxt.
26      Nucleotide_bound(me, next, v)
27      | Mrna.Nucleotide(rna, rna_nxt, v')
28   Terminator_bound(me, next, v)  extended by
29      me?isTerm(c).c!true().Terminator_bound(me, next, v)
30    + me?elongate(rna).new last(ρrna).
31      Terminator_bound(me, next, v)
32      | Mrna.Terminator(rna, last, v') | List.Nil(last)
```

Fig. 15. DNA module

Both Nucleotide and Terminator extend Node with two functions. Queries via isTerm determine whether elongation should stop: Nucleotide returns false, whereas Terminator returns true. With the second function elongate, Rnap sends a reference rna to the transcript's growing end that the DNA representative elongates. Nucleotide appends an individual Nucleotide of complementary content indicated by v', imported from the 'mRNA' module, and returns the new growing extremity rna_nxt. The hybrid Terminator completes the nascent transcript with an mRNA.Terminator, followed by a Nil.

Parameterization. Table 7 gives sample values of the ρ function that attributes rates to on object's functions. Functions not associated with a rate are instantaneous. Recall that each object is identified by its me channel, over which

name	ρ_{name}(function)	quantifies
rnap	ρ_{rnap}(bind)=0.1	RNAP access to promoters over global channel
ribosome	ρ_{ribosome}(bind)=1	... ribosome to mRNA
degradosome	$\rho_{\text{degradosome}}$(bind)=0.1	... degradosome to mRNA
prom	ρ_{prom}(initiate)=0.1	interaction with individual Promoter
	ρ_{prom}(unbind)=0.1	
dna	ρ_{dna}(getNext)=30	interaction with individual Nucleotide and Terminator of DNA
rbs	ρ_{rbs}(init)=0.5	interaction with a RBS
	ρ_{rbs}(unbind)=2.25	
rna	ρ_{rna}(getNext) = 100	interaction with mRNA Nucleotide and Terminator
protein	ρ_{protein}(kill)=0.002	protein degradation

Table 7. Examples of ρ definitions, which fix rates of functions calls over channels.

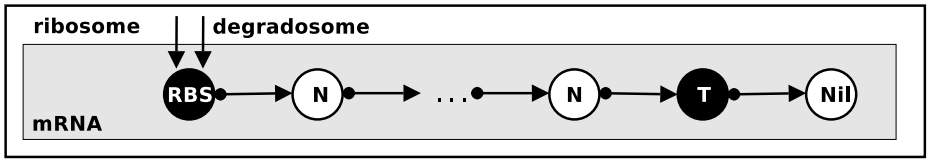


Fig. 16. RNA is based on the module 'degradable queueing list'.

its function are invocated. This allows to associate method calls on different objects of the same class with distinct rates, as useful for initiation rates of promoters or ribosomal binding sites.

5.2 mRNA, translation and degradation

Our model of mRNA is based on 'degradable queueing list' (Fig. 12). By this we implicitly account for the molecule's unstable character by inheritance, and impose queueing on the ribosomes and degradosomes processing it. Similarly as we did for DNA, we assemble mRNA from three components at different levels of abstraction illustrated in Fig. 16. The module's definition follows in Fig. 17.

The two-profile RBS represents the $5'$ end of mRNA, including the ribosomal binding site and the translation start signal. It implements the co-transcriptional race between translation and degradation. Decay initiates over the global channel degradosome in the free profile: after RBS has passed the reference to its bound successor, it becomes the inert process **0** (line 15). If alternatively a Ribosome binds over the global channel ribosome, it causes a switch from RBS_free to RBS_bound. Similarly to the unstable intermediate on promoters, two interactions unbind and initiate become possible, reflecting equations (2.8) and (2.9) in Table 2. Their quantitative behaviour is fixed by the ρ function associated with the RBS's me channel.

```
module 'mRNA'
channel
   ribosome with bind/1
   degradosome with bind/1
import
   List(Node, Nil) from 'degradable queueing list'
   Protein         from 'favorite protein'
export
   RBS extends List.Node by init/1,unbind/0
   Nucleotide extends List.Node by isTerm/1,elongate/0
   Terminator extends List.Node by isTerm/1,elongate/0
define
   RBS(me,next) ≜ RBS_free(me,next)
   RBS_free(me,next) ≜ extended by
      degradosome?bind(c).next!bind().c!next.0
    + ribosome?bind(c).c!me.RBS_bound(me,next)
   RBS_bound(me,next) ≜ extended by
      me?init(c).next!bind().c!next.RBS_free(me,next)
    + me?unbind().RBS_free(me,next)
   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?().
       Terminator_bound(me,next,val) | Protein
```

Fig. 17. mRNA module

Nucleotide propagates translation and degradation. A Degradome decays a Nucleotide using its inherited function kill. The Ribosome uses function getNext to step through the mRNA representative in translation; it stops on the Terminator due to its positive response to isTerm. The last function elongate that Ribosome calls remains without effect in this basic Nucleotide model, but spawns a fresh protein when called on a Terminator, that marks the endo of a protein coding region.

Ribosome. The module 'ribosome' declares the co-actions for translation of mRNA into proteins, see Fig. 18. Note how the simplicity in the representation of translation becomes apparent in the elongate function of a 'mRNA' Nucleotide, compared to that of DNA.

Degradosome. The degradosome specification is straightforward as well (Fig. 19). After gaining access to the RBS, it stepwise destructs the whole mRNA, calling kill and getNext on each of its Nucleotides. Degradation stops at the end of the transcript - on Nil, *not* on the Terminator as does translation. This distinction will prove useful when dealing with polycistronic mRNA.

```
1  module 'ribosome'
2  channel ribosome
3  export  Ribosome
4  define
5     Ribosome() ≜ Ribosome_free()
6     Ribosome_free() ≜ ribosome!bind(c).c?rna.
          Ribosome_bound(rna)
7     Ribosome_bound(rna) ≜ new c.
8        rna!init(c).c?next.Ribosome_elongating(next)
9      + rna!unbind().Ribosome_free
10    Ribosome_elongating(rna) ≜ new c1.rna!isTerm(c1)
11       c1?true().rna!elongate().Ribosome_free()
12     +  c1?false().rna!elongate().
13           new  c2.rna!getNext(c2). c2?next.
14           Ribosome_elongating(next)
```

Fig. 18. Ribosome module

```
1  module 'degradosome'
2  channel degradosome
3  export Degradosome
4  define
5     Degradosome ≜ Degradosome_free()
6     Degradosome_free ≜ new c.
7        degradosome!bind(c).c?rna.Degradosome_working(rna)
8     Degradosome_working(rna) ≜ new b.rna!isNil(b).
9        b?true()  .node!kill().Degradosome_free
10     + b?false().new c.node!getNext(c).c?next.rna!kill().
11          Degradosome_working(next)
```

Fig. 19. Degradosome module

Proteins. Proteins have many functions in the cell, that are not covered in this work. The only point we mention beyond their expression is their limited lifetime, reflected by a kill function for the degradation of Proteins of a certain type identified by prot:

```
Protein ≜ prot?kill().0
```

5.3 Extensions

We extend our components to cover details introduced in Sec. 2.6.

Operons and polycistronic mRNA. We devote a module to operons, and the polycistronic mRNA transcribed from them (Fig. 20). It defines the processes OperonLinker and InternalRBS. OperonLinker connects two genes within

```
1  module 'operon'
2  channel ribosome
3  import
4     List(Node) from 'persistent queueing list'
5     Mrna(Nucleotide,Terminator,RBS) from 'mRNA'
6  export
7     OperonLinker extends List.Node by isTerm/1,elongate/2
8     InternalRBS  extends Mrna.Nucleotide
9  define
10    OperonLinker_free(me,next,v) ≜ List.Node(me,next,v)
11    OperonLinker_bound(me,next,v)  extended by
12       me?isTerm(c).c!false().OperonLinker(me,next,v)
13     + me?elongate(rna,c).
14          new rna_nxt(ρ_rna). new_rna_nxt2(ρ_rna).c!rna_nxt2.
15          OperonLinker_bound(me,next,v)
16          | Mrna.Terminator(rna,rna_nxt)
17          | InternalRBS(rna_nxt,rna_nxt2)
18    InternalRBS(me,next,v) ≜ InternalRBS_free(me,next,v)
19    InternalRBS_free(me,next,v)  extended by
20     + ribosome?bind(c).c!me.InternalRBS_bound(me,next,v)
21    InternalRBS_bound(me,next,v)≜Mrna.RBS_bound(me,next,v)
```

Fig. 20. Operon module

an operon. It spawns a transcript that comprises a mRNA Terminator, on which translation of the first protein stops, followed by an InternalRBS, on which translation of the second protein initiates – but not degradation. Initial binding of the Ribosome occurs as on a regular RBS, same for unbinding and translation initiation, while decay propagation is inherited from a regular mRNA Nucleotide.

We can assemble an Operon after importing the previous modules. For better legibility we omit channel creations and parameterization.

```
Operon ≜
      Dna.Promoter | Dna.Nucleotide | ... | Dna.Nucleotide
    | OperonLinker | Dna.Nucleotide | ... | Dna.Nucleotide
    | Dna.Terminator
```

The transcription of our operon yields polycistronic mRNA coding for two different proteins[6], which are translated with distinct efficiencies (flexibly set by the ρ function of RBS's me).

```
PolycistronicMrna ≜ Mrna.RBS
    | Mrna.Nucleotide | ... | Mrna.Nucleotide
    | Mrna.Terminator | InternalRBS
```

[6] This suggests to make our import statement for modules parametric. The idea would be to import a specific protein into the module 'mRNA', rather than the generic 'favorite protein', with its specific ρ_{protein}.

```
1  module 'tandem promoter'
2  import
3     P(Promoter) from 'DNA'
4     InternalRBS from 'operon'
5  export
6     Promoter extends P.Promoter by elongate/2
7  define
8     Promoter_free(me,n) ≜ extended by
9       + me?elongate(rna,c).new rna_nxt(ρ_rna).c!rna_nxt.
10        Promoter_free(me,n) | InternalRBS_free(rna,rna_nxt)
```

Fig. 21. Tandem promoter module

```
    | Mrna.Nucleotide | ... | Mrna.Nucleotide
    | Mrna.Terminator | Node.Nil
```

We obtain the following reduction sequence, at the end of which the transcript has yielded distinct numbers of A and B proteins, and been degraded:

```
Operon | Rnap | Ribosome | Degradosome
→* Operon | Rnap | Ribosome | Degradosome |
   | PolycistronicMrna
→* Operon | Rnap | Ribosome | Degradosome
   | ProteinA | ... | ProteinA
   | ProteinB | ... | ProteinB
```

Tandem promoter. Let us now consider promoters in a tandem, illustrated in Fig. 6(c). The question is how to represent the internal promoter, over which transcription proceeds after it has initiated at the outer. The transcript is then appended a new element which offers translation initiation, but only propagation of decay - we previously designed an element of this functionality for transcripts resulting from operons. Our specialized promoter representative listed in Fig. 21 extends from the regular one by an elongate function that appends an Internal-RBS_free imported from module 'operon' to the existing transcript.

Promoter clearance. One of our simplifications so far is that a Promoter becomes available for new Rnap_free immediately upon after initiation, when switching to free. In reality RNAP clears its footprint stepwise, inducing a possibly limiting delay for highly efficient promoters, as those depicted in Figure 4(a). The model can be extended with an additional profile to reflect this synchronization, delaying the return to profile free until Rnap has moved far enough. We included this is in our implementation, used for the simulations in Section 6.

5.4 Challenging cases

The integration of more biological detail can become increasingly challenging. This becomes clearer on closer inspection of the biological matter, as points

related to secondary structures of mRNA (intrinsic termination of transcription, transcriptional attenuation), and two-way traffic on DNA and mRNA.

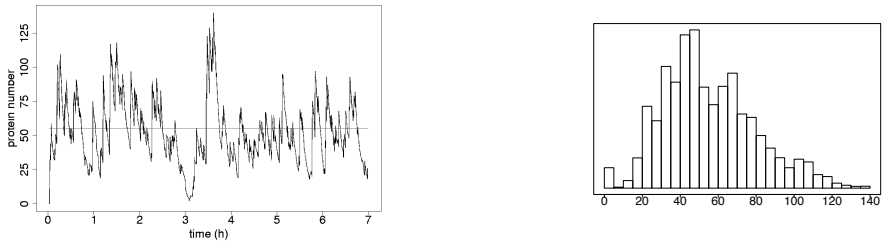
An example is *transcriptional attenuation*, in which transcription termination is determined by the efficiency of translation of the nascent transcript. The crucial detail is that intrinsic termination depends on more than mere recognition of a terminator sequence which actually comprises two portions with different effects. The first codes for an mRNA sequence that quickly forms a stable secondary structure, called hairpin. It is followed by a DNA sequence on which RNAP stalls. Both factors contribute to destabilize elongating RNAP, allowing the transcript below its footprint to partly dissociate from the template strand. RNAP eventually falls off DNA. In *transcriptional attenuation*, the formation of the mRNA secondary structured is impaired by ribosomes that translate the mRNA portion forming the hairpin with sufficient efficiency. RNAP recovers from its speed loss and continues transcription over the terminator. Capturing this would required more elaborate mechanisms of concurrent control than so far, specifically regarding RNAP's dependency on the last chunk of mRNA assembled.

Antitermination of transcription on intrinsic terminators is related to attenuation. One could satisfyingly deal with it, while covering less detail than required for attenuation. The simplest strategy would be to introduce an additional profile antiterminated for RNAP, which continues over terminator signals. Transition to this profile would be triggered by interaction of Rnap_elongating with regulatory proteins.

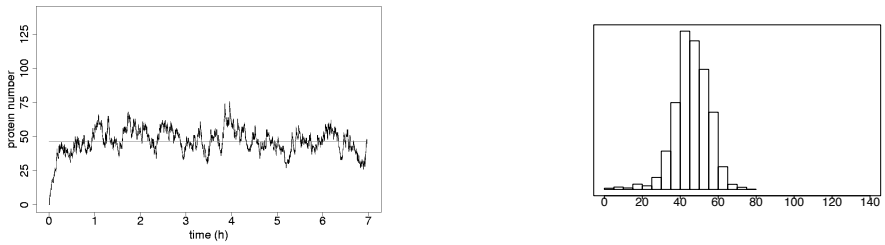
Two-way traffic occurs both on DNA and mRNA. The concurrent control supported so far can not yet account for traffic problems on double-stranded DNA (one of two RNAP falls off after a head-on collision), nor for details of mRNA decay. So far we only realized queueing control on single stranded macromolecules, which are processed in one direction. Our model of mRNA decay is phenomenological, a detailed one would cover the initial step of decay, in which the transcript is cleaved by one member of the degradosome proceeding with the same orientation as transcription, and subsequent decomposition of isolated mRNA chunks by another enzyme in opposite direction.

6 Simulation

In this section we present simulations obtained from our model components with the BioSPI tool [52], after encoding pattern guarded inputs within the stochastic π-calculus [35]. Our focus lies on the effect of *translational bursting* introduced in Sec. 2.5. We hence compare the dynamics of unregulated expression of a single gene under variation of two crucial parameters. The underlying model is that of a single gene comprising 1000 nucleotides, its transcription into mRNA, and subsequent translation and degradation. As experimentally confirmed by Ozbudak et. al [45], the variation of transcription of translation initiation parameters leads to significant differences in expression patterns. These would not appear in continuous deterministic simulation, which nevertheless yield comparable *av-*



(a) *bursty gene expression*



(b) *smooth gene expression*

Fig. 22. Basal expression of a single gene under parameter variation. Left: time course of protein numbers, right: histogram of protein number over the simulated period.

erage expression levels, nor can they be reproduced by one step models of gene expression as [6]. Our parameter combinations are the following:

(a) In the setting we refer to as *bursty*, the promoter yields rare transcription initiations (k_{init} of 0.01). This is combined with efficient translation (k_{init} of 1).
(b) The *smooth* setting inverts the parameters: transcription initiates more frequently (k_{init} of 0.1), while translation initiation is rarer (k_{init} of 0.1).

The mRNA and protein decay rates are the same for both settings, 0.1 and 0.002 respectively, these are taken from [45] as the above paramters, and the number of RNAP, ribosomes and degradosomes are fixed. We executed both combinations for 7 hours of simulated time.

Figure 22 reports a sample run for each of the two settings. The left curve in Fig. 22(a) displays the evolution of the protein level over time for an execution of the *bursty* combination. The protein level fluctuates strongly around an average of 55, marked by a horizontal line. The fourth hour exhibits the strongest variability: after it has almost emptied, the protein pool replenishes rapidly to a maximal level around 140. We provide a summary of the protein levels observed over the whole simulation period by the histogram to the right of Fig. 22(a). Here the simulated time is divided into equally long intervals. The bars indicate by their heigth how often a given number of proteins (as labeled on horizontal axis) is observed.

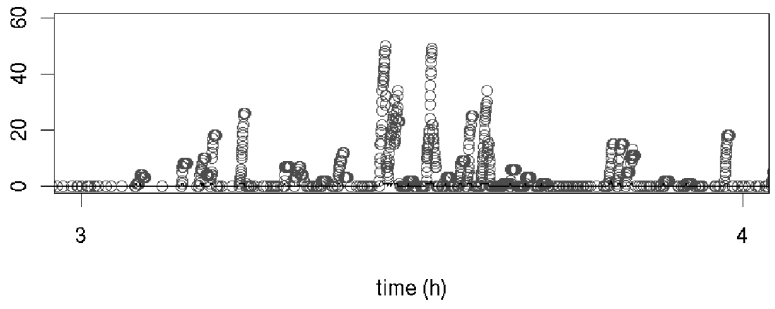


Fig. 23. Concurrent translation of mRNA by multiple ribosomes.

Simulations based on the *smooth* setting have a clearly distinct behavior, as shows Fig. 22(b) where the protein level only weakly fluctuates around an averate of 46. The histogram confirms that the distribution is pronouncedly narrowed compared with the previous setting.

An alternative interpretation of the histograms is as *population snapshots*, with respect to the expression level of a given protein. In this view the height of the bars indicate the fraction of the population with a certain protein level (horizontal axis). This shows how for the setting *bursty*, the variability propagates from the time course within an individual cell, up to population heterogeneity.

In the following table we adopt another perspective on the same data. The second column reports the mean protein crop per transcript, which averages to 10 for setting *bursty*. New transcripts appear about every 100 seconds (3rd column). This explains the drops observed in Fig. 22(a) over periods in which the effect of protein degradation surpasses that of expression. Setting *smooth* behaves differently. It yields approximately one protein per transcript, fresh transcripts become available every 10 seconds, and both together result in weak fluctuations. Note that while the total number of transcriptions for setting *smooth* almost tenfold exceeds that of *bursty* (4th column), the total number of protein produced differ far less across the two settings (5th column):

setting	proteins per transcript	avg. spacing between transcript initiations	total transcriptions	total translations
bursty	≈ 10	≈ 100 sec	250	2725
smooth	≈ 1	≈ 10 sec	2318	2338

Translational bursting. We could not yet observe the origin of the strong bursts in protein numbers in Fig. 22(a). This motivates further inspection of setting *bursty*'s simulation. Figure 23 displays the numbers of translating ribosomes

(circles) within the fourth hour of the simulation period, in which the protein pool empties, and then rapidly replenishes to the maximum. While the number of full mRNA molecules never exceeds two (data not shown), these are simultaneously processed by up to 50 ribosomes. As discussed in Sec. 2.5 the strongest bursts in protein levels occur when for a mRNA, the number of translations by far exceeds the average. The circles forming the bottom line may first appear peculiar; they can be explained as follows. For setting *bursty* new transcriptions are *completed* every 100 seconds. However, recall that nascent transcripts are translated co-transcriptionally. This means that whenever some RNAP is producing a transcript (wich takes around 100 seconds), one or more ribosomes closely follow it on the nascent mRNA. Hence the bottom line reflects that there is virtually always *some* coupled transcription and concomitant translation going on. The column-reminiscent peaks mark transcripts yielding exceptionally high protein crops; this is in agreement with a geometric distribution.

7 Conclusion

The experience gained in this work motivates further improvements of modeling languages based on the stochastic π-calculus [48, 52]. We extend the stochastic π-calculus [52] by input guarded patterns [68], this allows to express multi-profile objects [18] for which we introduce a simple notion of inheritance. We propose a simple module system, which is useful for model building. A stochastic semantics for our calculus, and an encoding into the stochastic π-calculus are presented in a companion paper [35]. We contribute an extensible model of transcription and translation, the two phases of gene expression. It represents discrete interactions between the molecular actors involved, and thus allows predictions at a high level of detail. We underline the expressiveness of our approach by a simulation case study, focusing on stochasticity in gene expression. Hereby we concentrate on translational bursting, which has been identified a prime origin of stochasticity in bacterial gene expression [30, 56].

The models suggested here may be extended to cover regulatory mechanisms in bacterial gene expression. They may constitute a starting point to deal with higher organisms, which refine the fundamental mechanism observed in bacterial gene expression.

Acknowledgments The author acknowledges insightful discussions with Denys Duchier, Cédric Lhoussaine, Joachim Niehren, and Bernard Vandenbunder. Many thanks also to David Ardell and Michael Baldamus for their feed-back during a research visit to the *Linnaeus Centre for Bioinformatics* in Uppsala, where part of this work was carried out (European Commission grant HPRI-CT-2001-00153). The author was funded by a PhD grant from the Conseil Régional Nord-Pas de Calais.

References

1. Elio A. Abbondanzieri, William J. Greenleaf, Joshua W. Shaevitz, Robert Landick, and Steven M. Block. Direct observation of base-pair stepping by RNA polymerase. *Nature*, 438:460–465, 2005.
2. David Adalsteinsson, David McMillen, and Timothy Elston. Biochemical Network Stochastic Simulator (BioNetS): software for stochastic modeling of biochemical networks. *BMC Bioinformatics*, 5(1):24, 2004.
3. Adam Arkin, John Ross, and Harley H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected Escherichia coli cells. *Genetics*, 149:1633–1648, 1998.
4. S Banerjee, J Chalissery, I Bandey, and RJ Sen. Rho-dependent transcription termination: More questions than answers. *Journal of Microbiology*, 44(1):11–22, 2006.
5. Anne Barnard, Alan Wolfe, and Stephen Busby. Regulation at complex bacterial promoters: how bacteria use different promoter organizations to produce different regulatory outcomes. *Current Opinion in Microbiology*, 7:102–108, 2004.
6. Ralf Blossey, Luca Cardelli, and Andrew Phillips. A compositional approach to the stochastic dynamics of gene networks. *Transactions on Computational Systems Biology*, IV:99–122, 2006. LNCS vol 3939.
7. S Brenner, F Jacob, and M Meselson. An unstable intermediate carrying information from genes to ribosomes for protein synthesis. *Nature*, 190:576–581, 1961.
8. Luca Cardelli. Abstract machines of systems biology. *Transactions on Computational Systems Biology*, III:145–168, 2005. LNCS vol 3737.
9. Luca Cardelli. Brane calculi: interactions of biological membranes. In *Proceedings of CMSB 2004*, volume 3082 of *Lecture Notes in Bioinformatics*, pages 257–278, 2005.
10. A. J. Carpousis. The Escherichia coli RNAdegradosome: structure, function and relationship to other ribonucleolytic multienzyme complexes. *Biochemical Society Transactions*, 30(2):150–154, 2002.
11. Trent A. Carrier and J. D. Keasling. Mechanistic modeling of mRNA decay. *Journal of Theoretical Biology*, 189:195–209, 1997.
12. Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, François Fages, , and Vincent Schächter. Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 325(1):24–44, 2004.
13. Nathalie Chabrier-Rivier, Francois Fages, and Sylvain Soliman. The biochemical abstract machine BioCham. In *Proceedings of CMSB 2004*, volume 3082 of *Lecture Notes in Bioinformatics*, pages 172–191, 2005.
14. Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.
15. Madhukar S. Dasika, Anshuman Gupta, and Costas D. Maranas. DEMSIM: a discrete event based mechanistic simulation platform for gene expression and regulation dynamics. *Journal of Theoretical Biology*, 232(1):55–69, 2005.
16. Patrick P. Dennis, Mans Ehrenberg, and Hans Bremer. Control of rRNA synthesis in Escherichia coli: a systems biology approach. *Microbiology and Molecular Biology Reviews*, 68(4):639–668, 2004.
17. Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138, 1971.
18. Denys Duchier and Céline Kuttler. Biomolecular agents as multi-behavioural concurrent objects. In *Proceedings of the First International Workshop on Methods*

and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord 2005), volume 150 of *Electronic Notes in Theoretical Computer Science*, pages 31–49, 2006.
19. Johan Elf and Mans Ehrenberg. What makes ribosome-mediated trascriptional attenuation sensitive to amino acid limitation? *PLoS Computational Biology*, 1(1):14–23, 2005.
20. B. Alberts et al. *Molecular Biology of the Cell*. Garland Science, 2002.
21. Joachim Frank and Rajendra Kumar Agrawal. A ratchet-like inter-subunit reorganization of the ribosome during translocation. *Nature*, 406:318–322, 2000.
22. Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976.
23. I Golding, J Paulsson, SM Zawilski, and EC Cox. Real-time kinetics of gene activity in individual bacteria. *Cell*, 123(6):1025–1036, 2005.
24. J. Gowrishankar and R. Harinarayanan. Why is transcription coupled to translation in bacteria? *Molecular Microbiology*, 54(3):598–603, 2004.
25. F Gros, H Hiatt, W Gilbert, CG Kurland, RW Risebrough, and JD Watson. Unstable ribonucleic acid revealed by pulse labeling of Escherichia coli. *Nature*, 190:581–585, 1961.
26. Marianne Grunberg-Manago. Messenger RNA stability and its role in control of gene expression in bacteria and phages. *Annual Reviews Genetics*, pages 193–227, 1999.
27. Tina M Henkin. Transcription termination control in bacteria. *Current Opinion in Microbiology*, 3(2):149–153, 2000.
28. Lilian M. Hsu. Promoter clearance and escape in prokaryotes. *Biochimica et Biophysica Acta*, 1577:191–207, 2002.
29. Mads Kaern, William J. Blake, and J. J. Collins. The engineering of gene regulatory networks. *Annual Review Biomedical Engineering*, 5:179–206, 2003.
30. Mads Kaern, Timothy Elston, William Blake, and James Collins. Stochasticity in gene expression: from theories to phenotypes. *Nature Reviews Genetics*, 6(6):451–467, 2005.
31. Andrzej M. Kierzek. STOCKS: STOChastic Kinetic Simulations of biochemical systems with Gillespie algorithm. *Bioinformatics*, 18(3):470–481, 2002.
32. Andrzej M. Kierzek, Jolanta Zaim, and P. Zielenkiewicz. The effect of transcription and translation initiation frequencies on the stochastic fluctuations in prokaryotic gene expression. *Journal of Biological Chemistry*, 276:8165–8172, 2001.
33. Oren Kobiler, Assaf Rokney, Nir Friedman, Donald L. Court, Joel Stavans, and Amos B. Oppenheim. Quantitative kinetic analysis of the bacteriophage λ genetic network. *Proceedings of the National Academy of Sciences USA*, 102(12):4470–4475, 2005.
34. Céline Kuttler and Joachim Niehren. Gene regulation in the pi calculus: Simulating cooperativity at the lambda switch. *Transactions on Computational Systems Biology*, 2006. Special issue of BioConcur 2004. In the press.
35. Céline Kuttler, Cédric Lhoussaine, and Joachim Niehren. A stochastic pi calculus for concurrent objects. Technical report, INRIA, 2006.
36. Karolina Maciag, Steven J Altschuler, Michael D Slack, Nevan J Krogan, Andrew Emili, Jack F Greenblatt, Tom Maniatis, and Lani F Wu. Systems-level analyses identify extensive coupling among gene expression machines. *Molecular Systems Biology*, 2006.
37. Tom Maniatis and Robin Need. An extensive network of coupling among gene expression machines. *Nature*, 416:499–506, 2002.

38. Harley H. McAdams and Adam Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences USA*, 94:814–819, 1997.
39. William R. McClure. Mechanism and control of transcription initiation in prokaryotes. *Annual Review Biochemistry*, 54:171–204, 1985.
40. Robin Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.
41. Robin Milner. *Computer Systems: Theory, Technology, and Applications. A tribute to Roger Needham*, chapter What's in a name?, pages 205–211. Monographs in Computer Science. Springer, 2004.
42. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100:1–77, 1992.
43. Petra J. Neufing, Keith E. Shearwin, and J. Barry Egan. Establishing lysogenic transcription in the temperate coliphage 186. *Journal of Bacteriology*, 183(7):2376–2379, 2001.
44. George Orphanides and Danny Reinberg. A unified theory of gene expression. *Cell*, 108:439–451, 2002.
45. Ertugrul M. Ozbudak, M Thattai, I Kurtser, A Grossman, and A.D. van Oudenaarden. Regulation of noise in the expression of a single gene. *Nature Genetics*, 31:69–73, 2002.
46. Brian J. Paul, Wilma Ross, Tamas Gaal, and Richard L. Gourse. rRNA transcription in E. Coli. *Annual Review Genetics*, 38:749–770, 2004.
47. Hervé Paulino, Pedro Marques, Luis Lopes, Vasco T. Vasconcelos, and Fernando Silva. A multi-threaded asynchronous language. In *7th International Conference on Parallel Computing Technologies*, volume 2763 of *Lecture Notes in Computer Science*, pages 316–323. Springer, 2003.
48. Andrew Phillips and Luca Cardelli. A correct abstract machine for the stochastic pi-calculus. *Transactions on Computational Systems Biology*, 2006. Special issue of BioConcur 2004. In the press.
49. D. Prandi, C. Priami, and P. Quaglia. Process calculi in a biological context. *Bulletin of the EATCS*, 85:53–69, 2005.
50. Corrado Priami. Stochastic π-calculus. *Computer Journal*, 6:578–589, 1995.
51. Corrado Priami and Paola Quaglia. Beta binders for biological interactions. In *Proceedings of CMSB 2004*, volume 3082 of *Lecture Notes in Bioinformatics*, pages 20–33, 2005.
52. Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80:25–31, 2001.
53. Mark Ptashne. *A Genetic Switch: Phage Lambda Revisited*. Cold Spring Harbor Laboratory Press, 3rd edition, 2004.
54. Mark Ptashne and Alexander Gann. *Genes and Signals*. Cold Spring Harbor Laboratory Press, 2002.
55. Stephen Ramsey, David Orrell, and Hamid Bolouri. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *Journal of Bioinformatics and Computational Biology*, 3(2):415–436, 2005.
56. Jonathan M. Raser and Erin K. O'Shea. Noise in Gene Expression: Origins, Consequences, and Control. *Science*, 309(5743):2010–2013, 2005.
57. António Ravara and Vasco T. Vasconcelos. Typing non-uniform concurrent objects. In *CONCUR'00*, volume 1877 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2000.

58. PN Ray and ML Pearson. Evidence for post-transcriptional control of the morphogenetic genes of bacteriophage lambda. *Journal Molecular Biology*, 85(1):163–175, 1974.
59. Aviv Regev and Ehud Shapiro. Cells as computation. *Nature*, 419:343, 2002.
60. Maria Elena Regonesi, Marta Del Favero, Fabrizio Basilico, Federica Briani, Louise Benazzi, Paolo Tortora, Pierluigi Mauri, and Gianni Deho. Analysis of the Escherichia coli RNA degradosome composition by a proteomic approach. *Biochimie*, 88(2):151–161, 2006.
61. Hessam S. Sarjoughian and Francois E. Cellier, editors. *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-based Theories and Methodologies*. Springer Verlag New York Inc., 2001.
62. Ursula Schmeissner, Donald Court, Hiroyuki Shimatake, and Martin Rosenberg. Promoter for the establishment of repressor synthesis in bacteriophage lambda. *Proceedings of the National Academy of Sciences USA*, 77(6):3191–3195, 1980.
63. Madeline Shea and Gary K. Ackers. The O_R control system of bacteriophage lambda: A physical-chemical model for gene regulation. *Molecular Biology*, 181:211–230, 1985.
64. KW Shearwin, BP Callen, and JB Egan. Transcriptional interference - a crash course. *Trends in Genetics*, 21:339–345, 2005.
65. Kim Sneppen and Giovanni Zocchi. *Physics in Molecular Biology*. Cambridge University Press, 2005.
66. D. A. Steege. Emerging features of mRNA decay in bacteria. *RNA*, 6(8):1079–1090, 2000.
67. Peter S. Swain. Efficient attenuation of stochasticity in gene expression through post-transcriptional control. *Journal of Molecular Biology*, 344:965–976, 2004.
68. Vasco T. Vasconcelos and Mario Tokoro. A typing system for a calculus of objects. In *1st International Symposium on Object Technologies for Advanced Software*, volume 472 of *Lecture Notes in Computer Science*, pages 460–474. Springer, 1993.
69. E.O. Voit. *Computational Analysis of Biochemical Systems: A Practical Guide for Biochemists and Molecular Biologists*. Cambridge University Press, 2000.
70. G von Heijne, L Nilsson, and C Blomberg. Translation and messenger RNA secondary structure. *Journal of Theoretical Biology*, 68:321–329, 1977.
71. Rolf Wagner. *Transcription Regulation in Prokaryotes*. Oxford University Press, 2000.
72. Charles Yanofsky. Trancription attenuation. *Journal Biological Chemistry*, pages 609–612, 1988.