N° d'ordre : 3949

Université des Sciences et Technologies de Lille

Thèse de doctorat en informatique préparée au
Laboratoire d'informatique fondamentale de Lille

presentée par
Céline KUTTLER

Modélisation de l'expression génétique bactérienne dans un pi-calcul stochastique à objets concurrents

Soutenance le 24 janvier 2007 devant le jury composé de:

Bernard VANDENBUNDER	IRI, FRE 2963 CNRS	Directeur de thèse
Cédric LHOUSSAINE	LIFL	Co-directeur de thèse
Hidde DE JONG	INRIA Rhone-Alpes	Rapporteur
Corrado PRIAMI	University of Trento	Rapporteur
François FAGES	INRIA Rocquencourt	Membre
Sophie TISON	LIFL	Membre
Hélène TOUZET	CNRS	Prédidente

Résumé

La biologie systémique cherche à comprendre la dynamique cellulaire qui émerge des interactions des constituantes cellulaires au cours du temps. La modélisation et la simulation sont des méthodes fondamentales de ce domaine.
Nous nous inspirons de la proposition de Regev et Shapiro (2002) consistant à appliquer le pi-calcul stochastique comme langage formel de représentation de connaissances biomoléculaires. Nos études portent sur la modélisation à l´échelle moléculaire de l'expression génétique bactérienne et s'avèrent pertinentes pour des organismes supérieurs. Nos points de départs sont des études de cas concrets de la régulation de la bascule génétique du phage lambda, de la transcription et de la traduction. Ces études révèlent l'utilité de concepts de programmation tels que les objets concurrents et motivent une extension du pi-calcul stochastique avec des motifs de réception. Nous présentons une sémantique pour ce langage qui attribue des chaînes Markoviennes en temps continu aux programmes. Nous validons nos modèles par des simulations stochastiques.

Résumé long

La biologie systémique cherche à comprendre la dynamique cellulaire qui émerge des interactions des constituants cellulaires au cours du temps. La modélisation et la simulation sont des méthodes fondamentales de ce domaine. Idéalement, les modèles informels peuvent être affinés en modèles formels, permettant ainsi leur simulation. Ceci permet de valider la cohérence des connaissances que le modèle intègre, d'obtenir des prédictions, et éventuellement d'approfondir la compréhension du comportement cellulaire.

Dans cette thèse, nous étudions la modélisation et la simulation de l'expression génétique. Nous nous concentrons sur l'étude des bactéries, qui ont été l'objet de recherches approfondies au cours des dernières decénnies en biologie. La modélisation des systèmes d'expression génétique bactérienne s'en trouve d'autant facilitée au niveau de précision nécessaire à la simulation. Quoique plus simples, les phénomènes associés à l'expression génétique bactérienne sont également pertinents pour l'étude d'organismes supérieurs.

Nous nous inspirons de la proposition de Regev et Shapiro (2002) consistant à appliquer le π-calcul stochastique comme langage formel de représentation de connaissances biomoléculaires. Le π-calcul est un langage fondamental pour décrire le comportement dynamique d'acteurs concurrents en interaction. Suivant la métaphore chimique, les acteurs sont des molécules et les interactions des réactions chimiques. Milner, Parrow et Walker (1992) ont conçu les opérateurs du π-calcul afin d'être fortement expressifs, quoique abstraits et minimalistes. La sémantique stochastique du π-calcul par Priami (1995) ajoute une notion de vitesse de réaction, déterminant les distributions des instants de réactions au cours du temps.

Cette thèse a pour amorce des d'études de modélisation de cas concrets. Celles-ci nous mènent à considérer des concepts de programmation tels que les objets concurrents, motivant une extension de notre langage de modélisation : le π-calcul.

Étude de modélisation de la bascule du phage lambda. Nous étudions la régulation transcriptionelle de la bascule du phage lambda (Ptashne, 2004). Depuis des décennies, cet exemple bactérien prototype reste central au déchiffrement des principes de la régulation génétique. La régulation transcriptionelle est un phénomène clé.

Les modèles quantitatifs précédents du contrôle de la transcription de la bascule du phage lambda reposent majoritairement sur l'analyse de Shea et Ackers (1985). Les approches de ce type mènent à des prédictions exactes de l'évolution temporelle à l'échelle de populations bactériennes. Néanmoins, elles laissent de côté les interactions individuelles entre molécules, interac-

tions qui déterminent la dynamique du contrôle de la transcription pour un gène unique. Par conséquent, elles ne peuvent prédire la dynamique de la population moléculaire au sein d'une unique cellule bactérienne.

Les techniques expérimentales pour l'évaluation quantitative de l'expression génétique de cellules individuelles ont considérablement progressé ces dernières années. De nouvelles données expérimentales suggèrent la réflexion par de nouvelles études de modélisation et simulation, et tenant compte des interactions précises entre molécules individuelles, permettant ainsi de prédire leur dynamique. Notre étude de l'initiation de la transcription de la bascule du phage lambda représente un premier modèle de ce type, et permet des simulations correspondantes.

Outre l'identification des acteurs et de leurs interactions précises dans la littérature, l'une des difficultés de ce travail est la quantification stochastique précise de toutes les réactions.

Langage et concepts de modélisation. Nous considérons le contrôle de l'initiation de la transcription de la bascule du phage lambda au niveau moléculaire en tant que contrôle concurrent dans le π-calcul stochastique. Des sémaphores correspondent à des sites opératoires de l'ADN, auxquels précisément une protéine peut être liée, en exclusion mutuelle de l'accès de l'Arn polymérase aux promoteurs chevauchants. Nous montrons comment formuler le renforcement coopératif du taux de réactions chimiques en tant que modulation de taux de fonctions dans le π-calcul.

La transcription et la traduction procèdent itérativement le long des macromolécules linéaires d'ADN et d'ARNm, qui sont considérés commes listes. Il est possible de modéliser les éléments de telles listes comme des objets concurrents, afin que l'héritage entre objets permette de déduire des modèles d'ADN et d'ARNm d'un modèle commun de liste élémentaire.

Néanmoins, le π-calcul stochastique ne facilite guère l'expression de tels objets concurrents. Vasconcelos *et al.* (1993) résolvent ce problème pour le π-calcul asynchrone — qui sous-tend le langage de programmation distribué TyCO— en proposant une extension avec des motifs de réception.

Dans cette thèse, nous présentons une extension du π-calcul stochastique avec des motifs de réception, permettant d'exprimer des objets concurrents et qui s'avère utile en tant que langage de modélisation pour la biologie systémique. En sus d'introduire et motiver l'usage des motifs de réception pour le π-calcul stochastique, une difficulté de ce travail réside dans la définition d'une sémantique stochastique en termes de chaînes Markoviennes en temps continu. L'algorithme de Gillespie (1976) spécifie l'exécution de telles chaînes de Markov, c'est à dire montre comment calculer des traces de simulation à partir de programmes en π-calcul stochastique. Nous montrons comment encoder les motifs de réception dans le π-calcul considéré sans ces dernières, et prouvons que cette opération préserve la sémantique stochastique.

Simulation stochastique. Nous faisons usage du système BioSpi afin d'implanter et d'exécuter nos modèles dans le π-calcul stochastique sans motifs de réception. Ceci est possible grâce à notre codage. Nous menons des simulations stochastiques exhaustives du système contrôlant la bascule du phage λ, ainsi que de la transcription et traduction bactrérienne. Nos résultats sont en accord quantitatif avec les connaissances expérimentales, ce qui souligne la pertinence de notre approche.

University of Lille 1

Doctoral thesis in computer science

Céline Kuttler

Modeling Bacterial Gene Expression in a Stochastic Pi-Calculus with Concurrent Objects

Defended Jan 24th, 2007 with the following comittee:

Bernard Vandenbunder	Interdisciplinary Research Institute (CNRS)
Cédric Lhoussaine	LIFL, University of Lille 1
Hidde de Jong	INRIA Rhônes-Alpes
Corrado Priami	Microsoft Research - University of Trento Centre for Computational and Systems Biology
François Fages	INRIA Rocquencourt
Sophie Tison	LIFL, University of Lille 1
Hélène Touzet	CNRS, University of Lille 1

Abstract

Systems biology seeks to understand the cellular-level dynamics arising from the interaction of cellular constituents over time. Modeling and simulation are essential techniques of the field. We follow Regev and Shapiro (2002) in using the stochastic pi-calculus as a formal representation language for biomolecular knowledge. We elaborate molecular level modeling studies for bacterial gene expression, which is relevant to higher organisms as well. Our starting point are concrete case studies on regulation at the lambda switch, transcription and translation. These reveal the usefulness of programming concepts such as concurrent objects, and motivate an extension of the stochastic pi-calculus with input patterns. We present a semantics for this language that maps programs to continuous time Markov chains. We validate our models through exhaustive stochastic simulation.

Extended abstract

Systems biology seeks to understand the cellular-level dynamics arising from the interaction of cellular constituents over time. Modeling and simulation are essential techniques of the field. Ideally, informal models can be refined into formal models, that can be executed to yield simulations, which allow to validate the coherence of the knowledge incorporated in the model, obtain predictions, and eventually deepen the understanding of the cell's behavior.

In this thesis, we study the modeling and simulation of gene expression. We restrict ourselves to bacteria, since these have been investigated more thoroughly over the last decades from the biological perspective. This facilitates modeling of systems of bacterial gene expression at the necessary precision for simulation. Even though simpler, the phenomena arising in bacterial gene regulation are relevant to higher organisms as well.

We follow the approach of Regev and Shapiro (2002) in using the stochastic π-calculus as a formal modeling language for biomolecular knowledge. The π-calculus is a fundamental language for the investigation of the behavior of concurrent actors, and the dynamics of their interactions. In the chemical metaphor, actors can be molecules and interactions chemical reactions. Milner, Parrow, and Walker (1992) designed the operators of the π-calculus to be highly expressive, even though abstract and minimalistic. The stochastic semantics of the π-calculus by Priami (1995) adds a notion of reaction speed, determining the temporal distribution of reaction time points.

In this thesis project, we start with concrete modeling case studies. This leads us to programming concepts such as concurrent objects, that motivate extensions of the stochastic π-calculus, our modeling language.

Lambda switch modeling study. We study transcriptional regulation at the λ switch (Ptashne, 2004). This prototypical bacterial example has for decades remained central to unraveling the principles of gene expression. Transcriptional regulation is a key aspect.

Previous quantitative models of transcription control at the λ switch are based on Shea and Ackers (1985). This class of approaches yields accurate predictions of the temporal evolution at the level of bacterial populations. They however disregard interactions between individual molecules, which determine the temporal dynamics of transcription control from a single gene. As a consequence, they lack the ability to predict the dynamic evolution of the molecular population within an individual bacterial cell.

In recent years, the experimental techniques for quantitative assessment of gene expression in single cells considerably advanced. New experimental results stimulate reflection by new modeling and simulation studies, taking

into account the precise interactions of individual molecules, so that their temporal dynamics can be predicted. Our case study on transcription initiation at the lambda switch provides a first such model and the corresponding simulations. Besides distilling the actors and their interactions from the literature, the most demanding aspect consists in the accurate stochastic quantification of all reactions.

Modeling language and concepts. We capture the molecular-level control of transcription initiation at the λ switch as concurrent control in the stochastic π-calculus. Semaphores correspond to operator sites on DNA, where precisely one protein may bind, in mutual exclusion with RNAP's access to overlapping promoters. We show how to express cooperative enhancement of chemical reaction rates as modulation of function rates in the π-calculus.

Transcription and translation consist in iterated processing of the linear macromolecules DNA and mRNA, which can be seen as lists. It is convenient to model the elements of such lists as concurrent objects, so that object inheritance permits to infer models of DNA and mRNA from a common basic list model. Unfortunately however, it is not easy to express concurrent objects in the stochastic π-calculus. Vasconcelos and Tokoro (1993) solve this problem for the asynchronous π-calculus underlying the distributed programming language TyCO by proposing an extension by input patterns.

In this thesis, we present an extension of the stochastic π-calculus by input patterns so that it can express concurrent objects, and use it as modeling language for systems biology. Besides discovering the usefulness of input patterns for the synchronous π-calculus, the difficult part is to define a stochastic semantics in terms of continuous time Markov chains. The algorithm of Gillespie (1976) specifies the execution of such Markov chains, i.e. how to compute simulation traces for programs of the stochastic π-calculus. We show how to encode input patterns in the π-calculus without input patterns, and prove that the stochastic semantics is preserved.

Stochastic simulation. We use the BioSpi system for implementing and executing our models in the stochastic π-calculus with input patterns. This becomes possible due to our above encoding of input patterns. We conduct exhaustive simulations of both the λ switch control system, and transcription and translation. Our results are in quantitative agreement with experimental knowledge, and thus supports the appropriateness of the approach.

Acknowledgements

Among my collaborators, I am most deeply indepted to Joachim Niehren and Cédric Lhoussaine. Their backgrounds in theory and pratice of concurrency greatly helped to mature my work. Denys Duchier and Gil Payet were my daily discussion partners when the first ideas on modeling with concurrent objects with multiple profiles christallized. Because both needed to pursue their professional pathes towards other destinations, the line of work continued with Cédric and Joachim. I thank Adelinde Uhrmacher for her support and advice, and Bernard Vandenbunder who assisted with his broader perspective on biology, and provided feed-back on my work in numerous discussions.

Different sources contributed funding for my work. The Conseil Régional Nord-Pas de Calais granted my salary. A grant by the University of Lille to Cédric Lhoussaine provided our hardware, and some travel expenses. The EU commission enabled my research visits to Jan Komorowski's lab at the Linnaeus Centre for Bioinformatics in Uppsala through the grant HPRI-CT-2001-00153. The IRI provided travel funding in the initial phase of my thesis, and the recruitment cost for my thesis commitee.

Several institutes in Lille supported me through their infrastructure. I am indepted to Sophie Tison for integrating me within her team at the LIFL, the University of Lille's computer science department. Michel Petitot and Hélène Touzet gave me a generous share of their office space, and Bernard Vandenbunder institutional support through the Interdisciplinary Research Institute.

My work benefited from discussions with many people from other places. Many thanks for their patience and expertise that helped clarifying questions to: David Ardell, Erik Aurell, Michael Baldamus, Ralf Bundschuh, Adrian Custer, Ingemar Ernberg, Ido Golding, Isabella Moll, Mark Ptashne, Keith Shearwin, and Kim Sneppen. My apologies to those I may have forgotten.

Next, I need to thank Anna Ingolfsdottír and Gordon Plotkin. They appreciated my work in early stages, and helped it to grow along the path to journal publication. I also owe gratitude to all anonymous reviewers, even if it took me a while to accept certain opinions.

I keep best memories of my ever chaning office mates, who cheerfully supported me throughout my thesis, namely Martin Brinkmann, Enrico Carlon, Ngalla Djitte, and friendly people next door at the LIFL. Chloé Degoulet and Sureyya Dogan kindly distracted me from my worries during lunchbreaks in the critical phase of writing down this thesis. Last not least, let me mention those cherished milongueros who brightened my rare late hours.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

The young field of *systems biology* seeks to aid the quantitative understanding of cellular biology. The term was coined by Ideker et al. (2001) and is meanwhile well established[1]. Systems biology complements research in the established area of *molecular biology*, which through wet-lab experimentation yields detailed descriptions of biomolecules and their interactions in living cells (Alberts et al., 2002; Lodish, 2003). Systems biology aims to contribute to better understanding of the overall dynamics of living matter, that arises from the interaction of many components, and is only partially understood.

Modeling and simulation is an important branch of systems biology, and the domain to which this thesis subscribes. It is concerned with the formal *modeling* of the behavior of cellular systems, in terms of quantities and time. Such models should be executable, so that they yield *simulations* by which to validate the coherence of the integrated knowledge, and eventually predict the system's behavior, and its response to perturbations. A complementary branch of systems biology deals with the systematic acquisition of biological data through techniques that are high-throughput, quantitative, and large-scale. While these reveal less detail than more traditional manual experimentation, they enable rapid knowledge discovery.

Gene expression is the biological phenomenon addressed in this thesis (Lewin, 2003). Gene expression is the sophisticated multi-step transfor-

[1]Systems biology is presented in recent textbooks by Alon (2006) and Palsson (2006), and article collections of Szallasi et al. (2006), Kitano (2001), and Bower and Bolouri (2001), where the latter not yet adopts this term.

mation of *hereditary information*, statically encoded in DNA, into those biomolecules that carry out the cell's vital functions: *proteins*. Many aspects of the exploitation of genetic material remain intricate to elucidate. Numerous interdependent components contribute to gene expression, and virtually any intermediate of the activities they perform can be modulated. The dynamics arising from this *control* or *regulation* of gene expression (Davidson, 2006; Ptashne and Gann, 2002) rapidly surpasses intuitive understanding.

Bacteria play a fundamental role in biological research (Trun and Trempy, 2003). The knowledge gained through the study of bacterial systems often yields insights relevant to other kinds of cells, which refine and complement the observed mechanisms. This namely holds for gene expression and regulation. Their central mechanisms were discovered within the bacterium Escherichia coli (Jacob and Monod, 1961), and their principles understood through two prototypical systems: the *lac* operon and the λ *switch* (Müller-Hill, 1996; Ptashne, 2004). Gene regulation at this latter has for decades remained a fruitful research field (Gottesman, 1999), and keeps providing important new insights (Gottesman and Weisberg, 2004; Dodd et al., 2005; Oppenheim et al., 2005). The λ switch frequently serves for benchmarking simulation methods (Hasty et al., 2001).

Regev and Shapiro (2002) suggested to abstract biomolecular knowledge in terms of concurrent computation in the π-*calculus* of Milner, Parrow, and Walker (1992), and base quantitative simulations on it. For the purpose of simulation, Priami, Regev, Shapiro, and Silverman (2001) adapted the stochastic variant of the π-calculus of Priami (1995) to the well-established simulation algorithm of Gillespie (1976). This latter accurately gives rise to the temporal and quantitative dynamics of biochemical systems at the level of molecular interactions.

This proposal raised numerous questions. While cellular networks were shaped by evolution and exhibit properties whose origins are difficult to unravel, the situation in engineering is complementary. Complex computational systems are incrementally assembled from fully specified components in order to reach specific goals. Methodologies and strategies play an important role in the *design* of software systems. The π-calculus however does not ease systematic and structured model creation. Its means of expression are indeed restrained to the very essence of concurrent computation, for the sake of semantic precision. It should be considered as an assembler or machine code, as opposed to higher languages with built-in support for actual programing techniques. From a methodological perspective, this thesis seeks to remedy drawbacks in the development, understanding, extension and sharing of biological models based on the π-calculus approach.

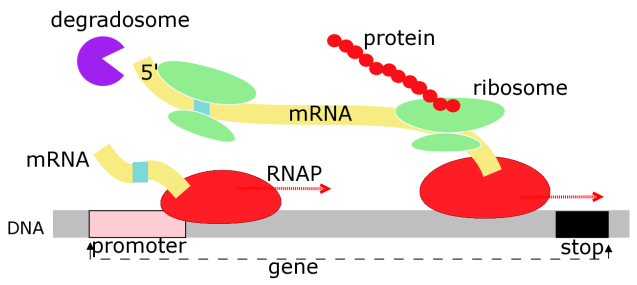


Figure 1.1: Overview of bacterial gene expression: decoding genes into proteins

1.1 Bacterial gene expression

Diverse activities co-exist in living cells. They ensure its vital functions, response to changes in the environment, allow the cell to fulfill its role in multi cellular organisms, and transmit its hereditary material from one generation to the next.

Gene expression is a major cellular activity. Figure 1.1 sketches this transfer of hereditary information from the sequence encoded in the macro molecule DNA (grey strip) via mRNA (yellow strip) to proteins. Numerous enzymes contribute. In the first step of gene expression, RNA polymerase (short: RNAP [2]) locates a *promoter* site on DNA, which indicates the start of a *gene*. Several RNAP may proceed along the same gene at the same time, all *transcribing* the DNA nucleotide sequence into mRNA. While transcription continues, the mRNA becomes accessible to ribosomes that bind at a dedicated start site (blue), in order to build proteins from its information content. This is called *translation*. More of it can start as long as the degradosome has not yet accessed the mRNA's $5'$ end in order to degrade the transcript.

Summarizing, our overview of gene expression illustrated first aspects of coordination, competition, and simultaneity between the various actors. These soon give rise to intricate dependencies.

Regulation of gene expression. Cells use their genetic material to produce appropriate *quantities* of products at the right time. The principles of regulation were understood in bacterial systems (Müller-Hill, 1996; Ptashne, 2004), which still offer many insights. Gene expression is controlled within all phases, it is most effective however to regulate it early – before transcription starts. Many discrete molecular events contribute to this *control of*

[2] RNAP polymerizes RNA, hence its name

transcription initiation that occurs at promoters, i.e. the DNA areas marking the start of a gene, colored in pink in Figure 1.1. Proteins find *operator sites* on DNA, located within or in the vicinity of promoters. The Velcro rule [3] summarizes transcription control by protein that *bind* to operators, remain fixed there for a while (from seconds to hours), and eventually fall off. Protein presence affects RNAP's interaction with the promoter.

The repertoire of molecular mechanisms for *repression* and *activation* of transcription initiation through DNA binding proteins is well identified in bacteria. In the simplest case, transcription is repressed by protein binding to an operator within the promoter. This excludes RNAP from promoter access; see Figure 1.6 on page 16. Activation conversely occurs through operator binding nearby, but outside the promoter. In this case the protein is beneficial to interactions between RNAP and the promoter, i.e. stability of binding and efficiency of transcription initiation. Both activation and repression in bacteria necessitate only a couple of regulatory proteins, while in higher organisms several dozens are involved. In this thesis, we will investigate fundamental mechanisms of repression and activation in bacteria, at the example of two promoters at the λ switch.

Regulatory patterns depend on protein quantities. Transcriptional regulation in bacteria depends on comparatively few components, the interaction between these may nevertheless challenge intuition. Consider an E.coli bacterium infected by the λ virus. The viral DNA can either be integrated in the hosts's genome, and remain dormant, or hijack the host cell machinery to produce a new crop of viruses. This fundamental lifestyle choice depends on two viral genes cI and cro, that code for regulatory proteins Rep (λ *rep*ressor) and Cro. Both control their own transcription and the other's through DNA binding. The protein binding patterns to DNA operator sites, and thus the regulatory outcome, heavily depend on protein quantities.

Figure 1.2 visualizes the regulatory influences by a diagrammatic notation, in which black horizontal bars represent genes. Assuming gene names A and B, a directed arrow from A to B represents the regulatory influence of gene A's protein on the transcription of gene B. The kind of arrowhead indicates repression (T-shaped) and activation (triangle). Note auto-regulatory genes, and expression without explicit activation as visualized by an activating arrow starting in the void.

[3]Central contribution of M. Ptashne, Lasker laureate of 1997. Velcro is the trademark of *a fastener for clothes or other items, consisting of two strips of thin plastic sheet, one covered with tiny loops and the other with tiny flexible hooks, which adhere when pressed together and can be separate when pulled apart deliberately* (McKean, 2005).

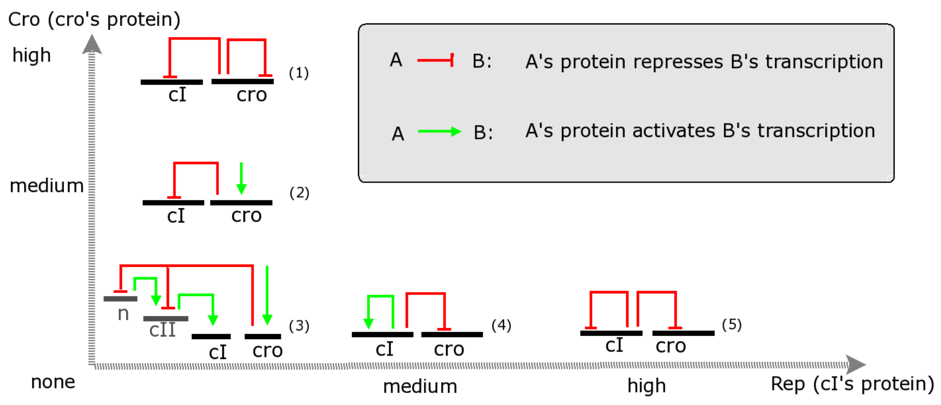

Cro (cro's protein)
high
medium
none
cI
cro
(1)
(2)
n
cII
(3)
(4)
(5)
medium
high
Rep (cI's protein)
A B: A's protein represses B's transcription
A B: A's protein activates B's transcription

Figure 1.2: Regulatory patterns between the λ switch genes cI and cro depend on protein abundance.

While Figure 1.2 distinguishes the regulatory patterns due to alternative abundances of Cro and Rep, the *alternative promoters* (see also Section 2.3) for cI remain implicit. The *p*romoter for *r*epressor *e*stablishment $\mathrm{P_{RE}}$ contributes to the viral lifestyle choice following infection. Transcription of cI in immunized cells initiates from the *p*romoter for *r*epressor *m*aintenance. $\mathrm{P_{RM}}$ neighbors cro's promoter $\mathrm{P_R}$. Both $\mathrm{P_{RM}}$ and $\mathrm{P_R}$ are controlled from the same *three operators sites* on DNA, that remain implicit in the illustration. Protein binding to either operator affects RNAP-promoter interactions in a distinct manner. It is important to underline the different *specificity* for each combination of operator, Cro and Rep binding. The stability of protein operator complexes scales with specificity; fewer proteins suffice to saturate more specific operators.

Item (3) summarizes the initial decision upon infection of the host cell by the λ virus. As long as neither Cro nor Rep are available, basal transcription of cro is possible. In the same setting cI's transcription depends on the protein products of the genes cII and N, which in turn are repressed by Cro. The impact of environmental conditions on this initial decision is still not fully understood. As Cro wins the race, it represses the expression of Rep (item 2). This enables the production of new λ viruses that eventually kill the host cell. After reaching a higher level, Cro represses itself (item 1).

If conversely Rep wins the initial race, it rapidly shuts off Cro, and up to intermediary concentrations activates transcription from its own promoter $\mathrm{P_{RM}}$ (item 4). Rep immunizes the host cell and its off-spring against further infections, and eventually suppresses its own expression at higher concentrations (item 5). The molecular mechanism for auto-inibition was only recently

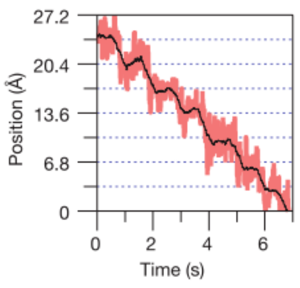


Figure 1.3: Observation of Rnap stepping over Dna

identified by Dodd et al. (2001): under physiological concentrations it necessitates a complex formed between multiple Rep bound to the Dna operator sites mentioned so far, and distal ones on λ Dna.

Certain environmental conditions lead to massive Rep decay, which efficiently re-activates the λ virus. This phenomenon called *induction* is not covered by our illustration. The traditional view of Cro's contribution to induction has been revised by Svenningsen et al. (2005), who reconsidered the influence of minor to moderate Cro amounts on immunized cells at typical Rep concentrations.

Observing gene expression in real time. We report recent quantitative assessment of the temporal dynamics of gene expression in a single cell, that are of interest to the modeling work in this thesis.

Abbondanzieri et al. (2005) provide evidence that transcribing Rnap advances over the coding region of Dna in *discrete steps* of a length equivalent to that of a single base of Dna (3.7 Å). The one-step advancement hypothesis had remained under debate since proposed by von Heijne et al. (1977). It is well understood that each advance over a single nucleotide subsumes intermediary steps, in which Rnap reads the Dna sequence, and takes up the suitable material from the environment to reflect the appropriate nucleotide in the nascent mRna. Figure 1.3 is also important to understand the *timing* of Rnap advancement. It visibly clarifies that in each step, a unique internal reaction limits Rnap in taking the next step. Assuming each forward step has constant probability to occur per unit time, Rnap's movement corresponds to a Poisson process, implying an *exponential distribution* of waiting times between steps (Manabe, 1981; Kampen, 2001).

Golding et al. (2005) tracked the dynamics of mRna production from a single gene, transcribed from the strongly repressible bacterial promoter $P_{lac/ara}$. This revealed that transcription can occur in quantal bursts, i.e.

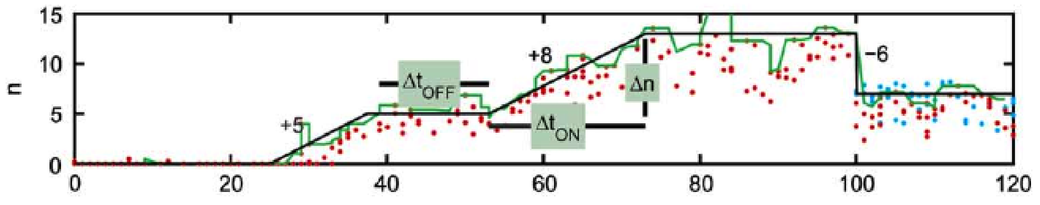


Figure 1.4: mRNA numbers (red points) produced from a single gene over time (in minutes)

meaning that long phases of repression take turns with such in which transcription occurs. The former are annotated as Δt_{OFF}, the latter as Δt_{ON} in Figure 1.4. The durations of the periods of inactivity are characterized by an exponential distribution. In each activity period, varying numbers of transcriptions occur, annotated as Δn.

Based on a wealth of recent experimental studies in single cells, Kou et al. (2005) discuss the kinetics of enzymatic reactions at the level of individual reactions. This embraces activities of the molecular machines of gene expression. The authors argue that the appropriate representation is in terms of *probabilities* for the enzyme to be in one among multiple *states* in its pathway. This is in contrast to the usual bulk reactions of biochemistry, where kinetics is expressed in terms of concentrations.

1.2 Concurrent modeling languages

Concurrent computation extends sequential computation by notions of concurrent actors, that may progress independently. Concurrent actors exchange information and resources by communication, and synchronize until enough resources and information are available. Actors may know each other or belong to private groups unknown to the external world.

Research on concurrent computation is traditionally motivated by applications within distributed networks, multi-tasking operating systems, parallel computers, and concurrent programming languages. Nondeterminism is unavoidable in concurrent computation. It arises once two actors compete for a resource, that is is only attributed to one. The omnipresence of nondeterminism renders concurrent systems notoriously difficult to analyze, model, and understand.

A number of formal languages were proposed to model concurrent systems. The earliest are Petri nets (Petri, 1962), a graphical language with states and actions reminiscent of finite automata. State charts are a more recent graphical formalism (Harel, 1987). Both found applications in biological modeling (Goss and Peccoud, 1998; Fisher et al., 2005). The calculus of communication systems (CCS) is a process algebra (Milner, 1980) preceding the π-calculus. It corresponds to the fragment of the π-calculus where all messages are void.

The π-calculus by Milner, Parrow, and Walker (1992) is a more recent language providing a purely syntactical notation for modeling concurrent computation on a high level of abstraction. Its syntax is built from a minimalistic set of operators that can be freely composed with each other. These operators include parallel composition, channel creation, expressions for channel communication, and possibly choice operators [4].

Even though minimalistic, the π-calculus is very expressive. Besides of many different aspects of concurrent control, it permits to express sequential computation in the λ-calculus, a Turing complete foundation of functional computation[5]. Two major variants of the π-calculus are to be distinguished following Palamidessi (2003). The *asynchronous* π-calculus is frequently studied in the context of distributed programming languages, where centralized control is neither available nor expressible. The more expressive *synchronous* π-calculus is advantageous for modeling in systems biology, where the stochastic scheduler needs to be implemented by a centralized controller anyway.

Banâtre and Metayer (1986) proposed to base concurrent computation on the chemical metaphor, where molecules interact according to chemical reaction rules[6]. Berry and Boudol (1990) followed this idea in presenting a so called chemical abstract machine as alternative formalization of the π-calculus semantics. What this model lacks however, is a notion of reaction speed.

The stochastic π-calculus of Priami (1995) extends the synchronous π-calculus by this lacking notion of *time*. It associates stochastic parameters to all channels, which defines the speed of interaction on this channel. The

[4]Choice operators are often omitted in *asynchronous* π-calculus dialects, since they are dispensable to nondeterminism, and often undesired in the context of concurrent programming languages. Choice operators are essential ingredients of the synchronous π-calculus though. Indeed, they are convenient constructs for modeling applications in systems biology.

[5]Beyond its enormous fundamental importance, the λ-calculus has also been used for modeling in biology (Fontana and Buss, 1996).

[6]This work has led to the development of the Gamma programming language (Banâtre and Métayer, 1993; Banâtre et al., 2001, 2005). In the latter, a minimalistic model for chemical programming called γ_0 was proposed concomitantly with a comparison to the chemical abstract machine.

$$\begin{aligned} \mathsf{deg}: &\quad \mathrm{D} + \mathrm{P} \xrightarrow{k_1} \mathrm{D} \\ \mathsf{dim}: &\quad \mathrm{P} + \mathrm{P} \xrightarrow{k_2} \mathrm{P}_2 \end{aligned}$$

(a) ***Chemical reactions***

```
public deg(k1), public dim(k2)
```

$$\begin{aligned} \mathsf{P} &\triangleq \mathsf{deg?()}.\mathbf{0} + \mathsf{dim!()}.\mathsf{P}_2 + \mathsf{dim?()}.\mathbf{0} \\ \mathsf{D} &\triangleq \mathsf{deg!()}.\mathsf{D} \end{aligned}$$

(b) ***π-calculus definitions***

Figure 1.5: Defining chemical rules in the π-calculus: shifting the perspective from reaction rules to objects

stochastic parameter defines an exponential distribution of delays, until reactions happen. This idea dates back to the stochastic process algebra PEPA by Hillston (1995), that defines semantics in terms of continuous time Markov chains (CTMCs).

Modeling biochemical reactions in the stochastic π-calculus. Regev and Shapiro (2002) propose to apply formal languages of concurrency to model molecular networks in cellular biology. This approach is inverse to that of Berry and Boudol, i.e. to use the chemical metaphor to formalize languages for the description of concurrent computation.

The stochastic π-calculus is the language chosen by Regev and Shapiro. It offers the advantage that processes defined it can be executed by the simulation algorithm of Gillespie (1976). Due to its suitably defined stochastic semantics, the stochastic π-calculus of Priami, Regev, Shapiro, and Silverman (2001) becomes a formal modeling language yielding quantitative simulations.

An underlying idea in using the π-calculus for descriptions of chemical reactions is the *shifted perspective from rules to objects*, i.e. objects represent molecules of different species, nucleotides or whole segments of DNA and RNA. In chemistry's rule-based view, one enumerates the reaction rules for all objects present in a chemical solution. Taking the alternative perspective of the π-calculus, one independently states each object's reaction capabilities.

Let us illustrate the perspective shift at the example of Figure 1.5. We consider two chemical reaction rules. The first is named deg and describes the *degradation* of a protein of species P by an enzyme of species D. The reaction speed is defined as rate k_1 which is some positive real number. In terms of mesoscopic chemical kinetics, it characterizes an exponential distribution of waiting times between reactions specified by the rule deg. The second rule is named dim. It represents the complexation of two P proteins to a heterodimer of type P_2. This reaction proceeds at rate k_2.

The π-calculus specification in Figure 1.5(b) first introduces two public channel names deg and dim, assigned the stochastic rates k_1 and k_2, respectively. Objects may use channel names to communicate, as walkie-talkie[7] users must agree. The speed of communication acts is defined by the stochastic rates of the channel in use.

P objects may play three different roles, that correspond to the three occurrences of P as reactants in the rules of Figure 1.5(a). The π-calculus definition of process type P in Figure 1.5(b) renders these three roles as alternatives of a choice $+$:

deg?().0: as second participant of rule deg, a P object may receive the dummy message over channel deg. As a consequence, it becomes the inert process 0;

dim!().P_2: as first participant of rule dim, a P object may send a dummy message () over channel dim, and then continue as P_2;

dim?().0: as second participant of rule dim, a P object may receive the dummy message over channel dim and disappear.

D objects may play only a single role, given there is a single occurrence of D as reactant of the chemical rules. This role is expressed by sending the dummy message on channel deg, once some P object is willing to receive it. In this case, the D object is re-incarnated as such, while the P object vanishes.

Two design decisions underly this encoding. Similarly to the order of participants in chemical reaction rules, they are somehow arbitrary:

1. First participants in chemical reactions always send, while second participants receive. Given that no true message is exchanged (so far), this design decision does not matter and could be inverted.

2. First participants always survive and continue as the product of the reaction, while second participants disappear. This design decision is arbitrary too, and could be inverted as well.

Chemical solutions are seen as multisets of objects. In the π-calculus, such multisets are expressed by parallel compositions of objects. The solution $P \mid P \mid D$ for instance may either reduce to $P_2 \mid D$ in one step, or to $P \mid D$, depending on which reaction rule is used.

[7]or: two-way radio

Discussion. Regev formulates the above ideas in terms of modeling guidelines for systems biology. These guidelines are sufficient for modeling systems of chemical reactions, but leave a broad design space for cases in which rule application depends on fine grained control. As a consequence, models may rapidly become obfuscated. Model legibility often suffers from lacking discipline in nomenclature, e.g. for a set of processes representing the same biological entity in various scenarios. This limits the scalability of this modeling approach, and indeed only fairly small modeling tasks have been approached in practice so far.

To conclude the situation as this thesis started in 2003, one can say that the π-calculus approach to modeling systems biology remained impaired by lack of programming language concepts and programming techniques, both with respect to the design and the understandability of models. These problem quickly appeared with the case studies of this thesis, which therefore contributes to remedy the situation.

1.3 Contributions

We first present an extension of the stochastic π-calculus that supports a richer notion objects, with features known from object-oriented programming languages. In particular it explicits notions of functions, interfaces, and inheritance. We then discuss modeling and simulation studies within this π-calculus extension.

1.3.1 Modeling language and concepts

The objects seen so far lack notions of persistent object identities and functions. This makes it impossible to distinguish different objects offering each the same functions, such as the many different building blocks of Dna. Let us consider a simpler example, with two promoters regulated in the same manner for different genes on the same Dna. Assume these promoters can be bound by Rna polymerase (Rnap) according to the following two chemical rules:

$$\begin{array}{ll} \mathsf{bind}_1: & \mathsf{PR\_free}_1 + \mathsf{RNAP} \rightarrow \mathsf{PR\_bound}_1 \\ \mathsf{bind}_2: & \mathsf{PR\_free}_2 + \mathsf{RNAP} \rightarrow \mathsf{PR\_bound}_2 \end{array}$$

As before, the activities of the two promoters can be expressed by two definitions in the π-calculus:

$$\begin{array}{l} \mathsf{PR\_free}_1 \triangleq \mathsf{bind}_1?().\mathsf{PR\_bound}_1 + \ldots \\ \mathsf{PR\_free}_2 \triangleq \mathsf{bind}_2?().\mathsf{PR\_bound}_2 + \ldots \end{array}$$

What is less nice here, is that both definitions are alike except for the identity, since both promoters do the same but at different places. Hence one might prefer having a unique definition, that is parameterized by the promoter's identity. It could be of the following form, where the channel me is universally quantified:

$$\mathsf{PR\_free(me)} \triangleq \mathsf{me?bind().PR\_bound(me)} + \ldots$$

Promoter identities are now distinguished by their channel names me, along which individual promoters receive *function calls.* All promoters offer a function bind that is addressed by the promoter's identity. Hence they provide the same interface, i.e. the same set of function names. Inheritance can be defined as the addition of new functions to objects.

Unfortunately however, function names are not available in the stochastic π-calculus (Priami et al., 2001), which motivates an extension. The idea of function names as input patterns in the π-calculus is not new, though. It was proposed by Vasconcelos and Tokoro (1993) in order to extend the asynchronous π-calculus with objects, and lays the foundation of the distributed programming language TyCO of Paulino et al. (2003). Note that objects can receive only such inputs, for which they provide a matching pattern. The test whether some pattern matches is closely tied to communication.

We present the *stochastic π-calculus with input patterns* that extends the synchronous stochastic π-calculus by input pattern with function names. The parametric promoter above is a first definition in that calculus. Besides discovering the usefulness of input patterns for the synchronous case, the difficult part is to define the *stochastic semantics* of this π-calculus.

Stochastic rates are now assigned to pairs of channel and function names. From the semantical perspective, programs in our stochastic π-calculus define continuous time Markov chains (CTMCs). The stochastic semantics for the π-calculus with input patterns that we propose defines an assignment of CTMCs to π-calculus programs. Our semantics copes with *instantaneous* reactions, i.e. calls to functions with infinite rates. These are necessary to deal with concurrent control between more than a pair of concurrent actors. The problem is to appropriately define the Markov chain of a process, under the assumption that there are no instantaneous loops. Previous work does either not define Markov chains explicitly (Priami et al., 2001; Phillips and Cardelli, 2004) so that correctness propositions can only be stated partially, or differently (Priami, 1995) which is less relevant to application in systems biology.

The essential benefit that our language offers to *model building* is the expression of *concurrent objects with multiple profiles*, i.e. multiple states with possibly different interfaces. We show how to define object extension by inheritance on a meta level. This representation style permits to accommodate precise information on quantitative and concurrent control, while retaining a maximal ease of notation, legibility, maintenance and extensibility.

We also show how to encode the stochastic π-calculus with input patterns back into the π-calculus without. We prove that our translation is correct, in that it preserves the Markov chains. Hence, existing simulation machines can be used to execute models formulated in our π-calculus with input patterns.

1.3.2 Lambda switch modeling studies

We investigate two cases in bacterial gene expression and regulation. First, we show how to model the control of transcription initiation at the λ switch promoters P_{RM} and P_R. Second, we present an general model of bacterial transcription and translation, that may be refined to numerous specific cases.

We express molecular actors in terms of concurrent objects in our stochastic π-calculus with input patterns, and regulatory mechanisms as concurrent control. It should be noted that semaphores appear all over. Each operator and promoter side forms a semaphore, since it can be bound by at most one protein or Rna polymerase. Another point is that we model coding regions of Dna and Rna as lists of nucleotides. Each nucleotide is an object, that can e.g. be rendered degradable by object inheritance, and leaves space for future refinement.

Our λ switch model substantially differs from previous work. We are more fine grained in that we explicitly render the discrete events in transcriptional regulation. These are abstracted away in kinetic models of gene regulation following the tradition of Shea and Ackers (1985), namely the major other models of the λ switch by Aurell et al. (2002) and Aurell and Sneppen (2002), as well as that of Arkin, Ross, and McAdams (1998). It is worthwhile emphasizing that this latter model only stochastically deals with gene expression *beyond* the point of initiation control, to which it applies the framework of Shea and Ackers.

More fine grained models may capture the level of observations enabled by recent experimental advances. Thus new methods of computational modeling and simulation are increasingly seeked. Independently of the present thesis, Saiz and Vilar (2006) propose a model that yields CTMC descriptions of events in transcription regulation at the λ switch.

Quantification of the models. We model the quantitative behavior by stochastic rates that determine the speed of all interactions in the system. Due to the high level of detail of our models, the literature provided only partial information, notably regarding the discrete events in transcription control at the λ switch, such as dimerization, binding, and quantitative control of interdependent events[8].

One of the problems was to differentiate between forward and backward directions of *reversible reactions*, as DNA binding and dimerization of proteins. These are disregarded by continuous models, as that by Shea and Ackers of the λ switch. Albeit valid for general biochemistry, it is increasingly emphasized that this simplification reaches its limits in genetic networks (Fedoroff and Fontana, 2002). With respect to transcription control in an individual cell, it means to disregard essential intermediary stages, that can persist over minutes (Bryant and Ptashne, 2003; Halford et al., 2004), while typical time scales of biochemical systems range around fractions of seconds.

Cooperative enhancement. DNA-bound proteins can increase each other's specificity by adhesive contacts. Positive control of RNAP in transcription initiation boils down to modulate the speeds at which RNAP's possible interactions occur. We present a modeling technique for the stochastic π-calculus to modulate the externally observable rate of communication. This technique applied to all other cases of biomolecular interactions we encountered, in which concurrent control affects reaction rates.

1.3.3 Stochastic simulation

We encoded our models in a format amenable to execution with the stochastic π-calculus simulator BioSpi, and conducted exhaustive simulations herein. This is enabled through our encoding of input patterns, since neither BioSpi nor SPiM support them as primitives[9]. For analysis of simulation traces and visualization of data, we developed tools in Perl and R (R Development Core Team, 2006).

In a first simulation study, we validate our models of transcription initiation control at the λ switch by reproducing well-established results from

[8] Useful new sources became available after finalization of our work in fall 2005 (Kuttler and Niehren, 2006): The highly valuable textbook by Sneppen and Zocchi (2005) covers the biophysical and quantitative aspects of gene regulation, including a chapter on the λ switch. Saiz and Vilar (2006) propose a full stochastic model of discrete events in transcriptional regulation at the λ switch, that is compatible with ours in that it gives rise to CTMCs (its scope is wider than ours, since it covers longe-range regulatory interactions on DNA). Golding et al. (2005) monitor mRNA production from the promoter P_{RM} at the λ switch in real time, and Kobiler et al. (2005) real-time protein levels in E. coli cells, during the decision between lysogeny and lysis upon infection by the λ virus.

[9] Our early models did neither use concurrent objects nor input patterns. Instead they relied on sophisticated protocols, that made these models difficult to understand, parametrize, and extend. Note also that the SPiM system was not available at that time.

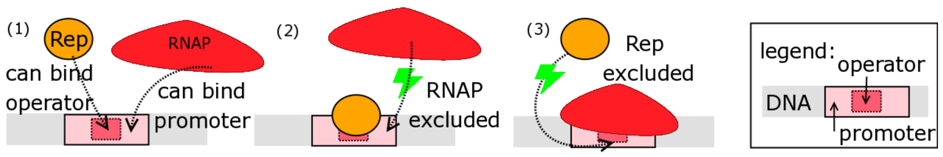


Figure 1.6: States of repressible promoter:(1) free, (2) blocked, (3) bound

experiments and deterministic simulation. For this we summarize longitudinal observations of repressor binding to operators. Furthermore, at a greater level of detail, we report the activity of the promoter P_{RM}, varying the control by dimerization, quantities of repressor protein, and cooperative binding. Herein we illustrate the variability of results for series of simulation runs under equal conditions. Again, our results well reproduce the effects attributed to the various regulatory mechanisms.

In a second study, we simulate combined dynamics of bacterial transcription and translation. Here, we concentrate on the effect of *translational bursting*, that has been identified as a key to stochasticity in bacterial gene expression (Kaern et al., 2005; Raser and O'Shea, 2005). It arises from variations in the quantitative control of transcription initiation versus translation initiation. We consider it worthwhile mentioning that the representation of gene expression in the π-calculus approach of Blossey, Cardelli, and Phillips (2006) lacks the means to capture translational bursting. This is due to an atomic representation of gene expression, which disregards those details of quantitative control that our model emphasizes.

1.3.4 Modeling example

Figure 1.6 illustrates the concurrent control of a promoter by an overlapping operator. This DNA sites can respectively be bound by RNAP and the repressor protein Rep. Importantly, these bindings are mutually exclusive. In setting (1) the promoter is not involved in any interaction, nor is the operator, hence we consider both as free. In (2) Rep's binding to overlapping operator site results in the promoter being blocked, while in (3) the promoter is bound by RNAP. We can similarly define states of the operator, RNAP and Rep.

Notably, the states free, blocked and bound determine the promoter's next *possible interactions*. In setting (1), binding can occur. In setting (2), Rep can unbind the operator, while the promoter may not accept RNAP's binding. In setting (3) the bound RNAP may either fall off DNA, or initiate

```
 1  module 'repressible promoter'
 2  public rnap with bind/1
 3  export Promoter with initiate/0,unbind/0,block/0,free/0
 4  define
 5  Promoter(me,op) ≜ Promoter_free(me,op)
 6
 7  Promoter_free(me,op) ≜
 8      rnap?bind(c).c!(me).Promoter_bound(me,op)
 9   + me?block().Promoter_blocked(me,op)
10   + op!unblock().Promoter_free(me,op)
11
12  Promoter_bound(me,op) ≜
13     me?unbind().Promoter_free(me,op)
14   + me?initiate().Promoter_free(me,op)
15   + op!block().Promoter_bound(me,op)
16
17  Promoter_blocked(me,op) ≜
18     me?unblock().Promoter_free(me,op)
```

Figure 1.7: Repressible promoter as π-calculus object

transcription along it, while Rep's access to the operator is suspended.

Figure 1.7 lists the module 'repressible promoter'. It formally specifies this narrated behavior in our π-calculus, defining a concurrent object Promoter with three profiles. The export statement in line 3 declares the Promoter object delivered by the module. The keyword with documents that each Promoter provides the functions initiate and unbind for interaction with RNAP, and block versus unblock to keep the operator synchronized with the operator. Note we also declare the arities of functions. The module's core follows the keyword define. It provides four parametric process definition.

The first definition in line 5 defines how a Promoter object is created.

The remaining three definitions define the *profiles* of Promoter, each corresponding to one of the previously discussed states. Note that all profiles take two parameters. The dedicated channel me represents the object's constant *identity*, while the other parameter op grants access to the object representing the operator site.

Each profile is defined as *choice* between pattern inputs over me, and communication over other channels. Pattern input over me correspond to function invocations on the object at this channel. The acceptance of pattern input is determined by the object's current profile. Our π-calculus dialect refuses function calls on an object, aka pattern input over me, that do not match the current profile. This works by requiring communication not only to agree the names of channels, but also in the function symbol.

For instance, Promoter_blocked ignores unbind requests. It is only able to respond to a unblock invocation, see line 18.

As Promoter is bound, the abstraction of the visiting RNAP can call either function unbind or initiate, see lines 13 and 14. Promoter_bound also causes a transition of the operator to profile blocked, by calling the block on op, see line 15.

We last consider the profile free, in which a Promoter is created. It offers three interactions. Before explaining the first, we need to comment on line 2 of the module. It declares a public channel rnap, and associates it with pattern input of the unary bind. Channel rnap is used in the binding of a RNAP representative to Promoter_free, see line 8. Using this public channel allows initial RNAP-promoter binding to occur between any possible pair of molecular actors, in a many-to-many fashion known from concurrent scenarios. Upon binding, the Promoter extrudes its identifier channel me to the RNAP representative that contacted it. In doing so, Promoter uses the dedicated channel c it was passed in the preceding step [10].

Promoter_free has to other ways to interact, which both regard synchronizations with its sibling Operator at channel op. In line 9, the Operator causes Promoter's transition to the blocked profile. This occurs when we are in scenario (2) from Figure 1.6, i.e. a repressor is bound to the operator, and the promoter representative must adjust to this state. The communication in line 10 is realized after Promoter's transition from profile bound to free. It corresponds to the adjustment of the operator representative, that accordingly switches from blocked to free.

1.4 Organization of this document

Part I introduces the biological material dealt with in this thesis: While Chapter 2 presents the overall mechanisms of transcription and translation, Chapter 3 details on transcriptional regulation at the λ switch.

Part II presents our methodological contribution. Its core consists in the stochastic π-calculus for concurrent objects (core SPICO) from Chapter 4. We map it into CTMCs by its stochastic semantics, and we show an encoding allowing to rewrite programs such that they become executable with previously proposed stochastic π-calculus engines.
Chapter 5 introduces our notion of objects and a simple module system that

[10]Throughout this work we emulate function-call-and-return in the π-calculus as sketched in Milner (2004).

is useful for later model building. These notions are illustrated at the example of various kinds of list, that constitute the basis for our later models of DNA and mRNA.
Chapter 6 proposes modeling techniques for concurrent objects expressed in our π-calculus, that may be useful beyond our own work.

Part III presents our modeling and simulation studies, based on the previously introduced concepts and biological background. Chapter 7 is devoted to the general mechanisms of transcription and translation, Chapter 8 to λ switch regulation.

Chapter 9 concludes.

1.5 Bibliographic note

Two thirds of the technical contributions in this thesis are published in international journals (Articles 1 and 2), the remaining third so far as a conference paper (Article 3). Two further publications report early ideas, and survey related simulation frameworks.

Article 1. Céline Kuttler. Simulating bacterial transcription and translation in a stochastic pi calculus. In *Transactions on Computational Systems Biology VI*, volume 4220 of *LNCS*, pages 113–149. Springer 2006. Special issue of CMSB 2005.

This journal article presents the modeling study on bacterial transcription and translation in the stochastic π-calculus with input patterns. It is covered by three chapters in this thesis: Chapter 2 presents the biological background. Chapter 5 elaborates on the notions of objects, inheritance, and modules; it also includes the discussion of lists as examples. Chapter 7 covers the π-calculus models of transcription and translation, and highlights some simulation results.

Article 2. Céline Kuttler and Joachim Niehren. Gene regulation in the pi calculus: Simulating cooperativity at the lambda switch. In *Transactions on Computational Systems Biology VII*, volume 4230 of *LNCS*, pages 24-55. Springer, 2006. Special issue of BioConcur 2004.

This journal publication presents the λ switch model in the original stochastic π-calculus. In Chapter 8 of this thesis, we simplify this model by reformulating it in the π-calculus with concurrent objects. Chapter 6 distills generic modeling techniques that are of major use to our λ switch model. Chapter 3 discusses the λ switch biology, basically following this paper. It complements with a rule-based representation, and relates its stochastic parameterization to that of our model in the π-calculus with objects.

Article 3. Céline Kuttler, Cédric Lhoussaine, and Joachim Niehren. A stochastic pi calculus for concurrent objects. In *Proceedings of the Second International Conference on Algebraic Biology*, volume 4545 of *LNCS*, pages 232-246, 2007.

This article corresponds to Chapter 4, that furthermore includes full proofs from the INRIA technical report 6076, 2006.

Article 4. 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*, volume 150 of *ENTCS*, pages 31–49, 2006.

This workshop paper contains early ideas on a new programming language with multi-profile objects and argues their advantage for modeling in systems biology. A formalization of this modeling language is still lacking, but much of the ideas can be expressed in the stochastic π-calculus with input patterns.

Article 5. Adelinde M. Uhrmacher and Céline Kuttler. Multi-level modeling in systems biology by discrete event approaches. *it – Information Technology*, 48(3):148–153, 2006.

This journal article compares the π-calculus with two discrete event modeling frameworks, currently used in systems biology: state charts of Harel (1987) and DEVS proposed by Zeigler (1984).

1.6 Related work

In this section, we first discuss related work in the areas of formal languages inspired by the chemical metaphor and dedicated to biological modeling. We then sketch actual modeling and simulation studies carried out in π-calculi and some close relatives.

Further related work follows later, after setting up some biological background: studies in stochastic simulation of bacterial gene expression are discussed in Section 2.5, and on the the λ switch in Section 3.3.

1.6.1 Modeling languages

A number of alternative modeling languages have been proposed for systems biology.

BioCham. BioCham (Chabrier-Rivier et al., 2004, 2005) is a rule-based language for the representation of biomolecular systems, with a notation reminiscent of that of chemistry. It is well-suited to consider dynamics by distinguishing three levels of semantics: a boolean, one at concentrations level, and stochastic that allows for simulation. BioCham offers querying of model properties by formulae in the temporal logic CTL (Emerson, 1990), and model input via SBML (Hucka et al., 2003).

Computing with membranes. Membranes allow to structure space into distinct compartments, and are another basic notion that fits naturally with the chemical metaphor, and indeed, they are basic to the minimalistic chemical programming model γ_0 of Banâtre et al. (2005). Computing with membranes was promoted by Păun (2000), whose P-systems consist of nested membranes in which molecules interact. The ambient calculus of Cardelli and Gordon (2000) is another instance inspired by the π-calculus. Motivated by modeling needs in biology, it was succeeded by the bio-ambient calculus of Regev et al. (2004) and the brane calculus of Cardelli (2005).

Kappa calculus. The κ-calculus of Danos and Laneve (2004) aims to express the combinatorics between proteins. Its conceptional starting point is reminiscent of StochSim's by Morton-Firth and Bray (1998) (see page 42). In this view, multi-state molecules are sets of binary flags. Proteins in the κ-calculus are represented as a set of *sites*. The evolvable configurations of sites are termed *interfaces*; individual sites can be either free, hidden, or bound (by an edge to another protein's site).

Beta binders. The β-binders formalism of Priami and Quaglia (2005) is a biologically motivated variation of the π-calculus. A leading desire is to eliminate the strict channel complementarity for interaction. Communication is enabled upon agreeing types, rather than names. It appears that composite β-binders are related to the interfaces of objects expressed in our π-calculus with pattern input, and that the operations of hiding and unhiding of β-binders correspond to profile transitions in multi-profile objects. In contrast to a driving point in the development of our calculus, β-binders do not allow to address a specific individual of a given species. A thorough discussion of the β-binders and our calculus with respect to biological modeling is planned for future work.

Pathway modeling language. Chang (2007) proposed the draft of a Pathway Modeling Language (PML), a high-level language that can be down-compiled to the π-calculus, suggesting it could yield models of more evident structure and modularization, that would be safer to compose.

Other languages. Ciocchetta and Priami (2006) propose a stochastic π-calculus with transactions, that permits atomic expressions of multi-participant interactions.
Danos and Krivine (2007) propose CCS-R, a variant of the CCS calculus by Milner (1980). It explicitly supports reversible reactions frequently encountered in biomolecular networks. The language is further investigated in Danos and Krivine (2004) and Danos and Krivine (2005).

1.6.2 Modeling studies

A number of modeling case studies have been elaborated in the stochastic π-calculus or in stochastic process algebras.

Lymphocyte recruitment Lecca et al. (2004) examine lymphocyte recruitment in inflamed brain vessels with the stochastic π-calculus engine BioSpi. Their results are the first based on this approach to faithfully agree with experimental knowledge. Their model doesn't include use of the choice operator.

Metabolism Curti et al. (2005) and Chiarugi et al. (2006) examine the metabolic pathways of a hypothetic minimal bacterium, that they modeled in a fragment of the stochastic π-calculus, with synchronizations but devoid of message-passing and recursion/replication.

The observation of internal states was crucial to this study, which lead the authors to develop their own abstract π-calculus machine, based on SPiM that does not directly support this possibility.

FGF signaling pathways Tymchyshyn et al. (2006) examine the quantitative properties of the FGF signaling pathway through simulation in the stochastic π-calculus of Priami et al. (2001), making comprehensive use of its linguistic features. Independently of the π-calculus specification, they elaborate a CTMC description of the same model. It is analyzed by Heath et al.

(2006) through CTMC probabilistic model checking, using the PRISM tool of Kwiatkowska et al. (2004). Kwiatkowska et al. (2006) thoroughly review the methodological toolset and related formalisms.

ERK signaling pathway Calder, Gilmore, and Hillston (2006) differ to the previous modeling studies in the view they adapt on the ERK signaling pathway: in addition to a molecule-centric model, they propose to make a correspondence between a subpathway and a process. They express this the process algebra PEPA , for which Hillston (1995) developed a mapping into CTMCs.

Part I

Bacterial Gene Expression

CHAPTER 2

Transcription and Translation

In this chapter, we first overview the main activities during transcription and translation, contemporaneous events in gene expression, and interdependencies between its phases. Section 2.2 follows with a presentation of the details of transcription and translation at the level of molecular interactions. This constitutes the foundation for our later discrete event modeling. Section 2.3 presents particular cases of transcriptional organization in bacteria, that can have interesting impact on the quantitative patterns of gene expression. Section 2.4 discusses the quantitative parameters that control transcription, translation and mRNA decay - they are indispensable for stochastic simulation. We put an emphasis on parameter combinations leading to translational bursting, one of the major intrinsic origins of stochasticity in bacterial gene expression. Finally, in Section 2.5 we review modeling and simulation studies that influenced our own work.

2.1 Overview of genetic actors and activities

Each cell contains the complete hereditary information of an organism, that is transmitted from one cellular generation to the next. This information is encoded in a linear, double-stranded DNA macromolecule, that winds up to a helix. Each of the two strands of DNA contains a sequence over the four-letter alphabet of nucleotides $\{A, C, G, T\}$. The sequences on both strands are complementary, A faces T, while C faces G. A *gene* is a segment of one strand of DNA, with explicit begin and end delimiters. Its information content can

$$\text{DNA} \xrightarrow[\text{transcription}]{} m\text{RNA} \xrightarrow[\text{translation}]{} \text{proteins}$$

Figure 2.1: The *central dogma of molecular biology* summarizes bacterial gene expression.

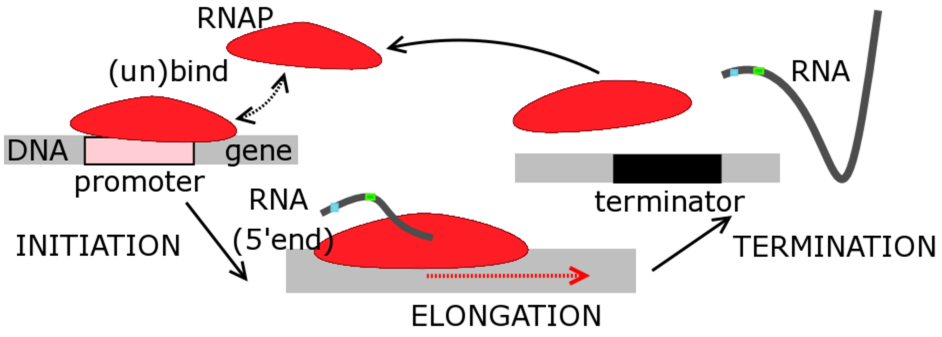


Figure 2.2: DNA processing by RNA polymerase (RNAP): promoter binding and initiation, transcript elongation, termination with release of RNA

be *transcribed* into a single-stranded RNA molecule, and *translated* for such RNA encoding proteins. Figure 2.1 summarizes these two phases of bacterial 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. It is subject to two competing subsequent processing phases: translation into proteins and degradation.

Both transcription and translation follow a similar scheme of *three phases*, illustrated in Figure 2.2 for transcription:

Initiation. RNAP localizes its start point on DNA, a *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.

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 (called *codons*), corresponding to amino acids, assemble these into growing sequences of amino acids, that subsequently fold into three-dimensional proteins.

mRNA decay. Besides carrying the code for proteins, a second decisive property of mRNA is its rapid *degradation*. Instability was indeed the defining feature as mRNA was discovered (Brenner et al., 1961; Gros et al., 1961). Degradation is accomplished by the *degradosome*, an ensemble encompassing several enzymes and their respective actions. With respect to translation, the decisive step is the degradosome's initial access to the $5'$ end of mRNA. Figure 2.3 illustrates its competion with ribosomal access for translation initiation. The reviews of Carpousis (2002) and Grunberg-Manago (1999) account for the full details of degradation, that may be covered by refinement of the formal model presented in this thesis.

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, with half-lives exceeding those of mRNA by far.

Concurrent features of gene expression. Many aspects of gene expression has a flavor of concurrency. The first is simultaneous processing of the same macromolecule by several molecular actors. The second are interdependencies, or *couplings*, between the different phases of gene expression, that are not yet visible in the simple scheme of Figure 2.1. The third is immediate competition for a resource, as the race for mRNA between ribosomes and the degradosome.

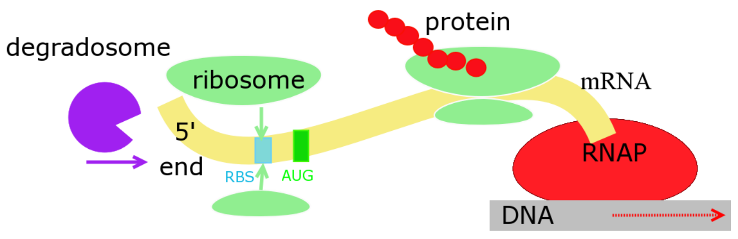


Figure 2.3: mRNA is subject to competing translation and degradation. The latter initiates at the 5′ end, while the ribosome assembles on the nearby ribosomal binding site RBS. Actual translation begins with the start codon, here 'AUG'.

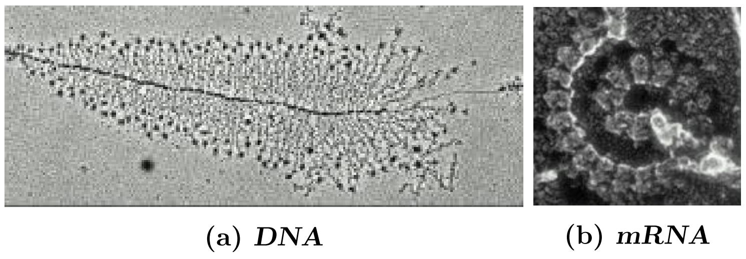


Figure 2.4: Simultaneous processing of DNA and mRNA (Alberts et al., 2002)

The macromolecules DNA and mRNA are *typically processed by multiple actors at the same time*. Bacterial genomes contain several thousand genes, many of which may undergo transcription at any instant. In addition, each gene can simultaneously be transcribed by several RNAP. Consider the structure reminiscent of a comb in Figure 2.4(a). The backbone is a DNA region encoding one gene. The comb's teeth are formed by nascent RNA transcripts. Althoughh we can not discern the RNAP themselves, the increasing lengths of transcripts indicates that transcription initiates at the left, and that elongation proceeds in left-to-right orientation [1]. The end point of transcription is easy to recognize: the non-coding stretch of DNA remains naked. Note that because transcription of this gene initiates with high efficiency, RNAP densely follow each other. Figure 2.4(b) shows the analogous phenomenon of queueing in translation. While the mRNA itself can not be seen, the visible blobs are ribosomes that rapidly follow each other.

[1]These transcripts are not protein coding. Starting from their respective 5′ ends, they gradually fold into three dimensional structures, which become ribosome constituents.

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 an mRNA molecule is being elongated by RNAP, ribosomes start accessing the other. The coupling between transcription and translation can become very tight, and fulfill specific goals (discussed in Gowrishankar and Harinarayanan (2004)). Transcriptional *attenuation* is a regulatory mechanism in which transcription stops unless the growing mRNA is efficiently translated, as discovered by Yanofsky (1981).

The complexity of couplings dramatically increases in *eukaryotic cells*, as Orphanides and Reinberg (2002) and Maniatis and Need (2002) review. Recent system-level analysis by high-throughput methods reveal surprising aspects (Maciag et al., 2006).

2.2 Molecular events

In this section, we represent the molecular events in transcription, translation and mRNA decay in terms of chemical reactions. This representation is as general as possible. It does not intend to reflect any particular gene, but emphasizes the *core mechanisms* by which any gene is transcribed by RNAP, any mRNA processed by ribosomes in translation, or degraded.

The level of resolution with respect to the elongation mechanisms is that commonly adopted by stochastic models: each elongation step is considered as atomic, accounting for the unique rate-limiting step in the advance of the macromolecular machines RNAP and ribosome. It does not consider the internals of actually appending new elements to a nascent mRNA or protein. Quantitative descriptions of these are recent, e.g. Greive and von Hippel (2005).

2.2.1 DNA and transcription

Table 2.1 summarizes the discrete interactions between DNA and RNAP following the scheme of McClure (1985) for initiation, and McAdams and Arkin (1997) for elongation. It also comprises the parameters of quantitative control, necessary to reproduce the dynamics of transcription in stochastic simulation.

Reactions (2.2.1) to (2.2.3) represent the essential steps of *transcription initiation*: initial reversible binding of RNAP to the promoter (*Prom*), followed by transition to the open complex. The parameter k_{on} for binding to an arbitrary promoter in the binding reaction (2.2.1) subsumes the time

$$
\begin{aligned}
\text{RNAP} + \text{Prom} &\rightarrow_{k_{on}} (\text{RNAP} \cdot \text{Prom})_{\text{closed}} && (2.2.1)\\
(\text{RNAP} \cdot \text{Prom})_{\text{closed}} &\rightarrow_{k_{off}} \text{RNAP} + \text{Prom} && (2.2.2)\\
(\text{RNAP} \cdot \text{Prom})_{\text{closed}} &\rightarrow_{k_{init}} (\text{RNAP} \cdot \text{Prom})_{\text{open}} && (2.2.3)\\
(\text{RNAP} \cdot \text{Prom})_{\text{open}} &\rightarrow_{k_{elong}} \text{RNAP} \cdot \text{DNA}_1 && (2.2.4)\\
\text{RNAP} \cdot \text{DNA}_n &\rightarrow_{k_{elong}} \text{RNAP} \cdot \text{DNA}_{n+1} && (2.2.5)\\
\text{RNAP} \cdot \text{DNA}_{\text{terminator}} &\rightarrow_{k_{elong}} \text{RNAP} + \text{DNA}_{\text{terminator}} + m\text{RNA} && (2.2.6)
\end{aligned}
$$

Table 2.1: Reaction rules for transcription

RNAP spends scanning DNA, until it recognizes and binds. At this point, promoter and RNAP form a *closed* complex, in which the two strands of DNA are firmly associated. RNAP may *unbind* without any further effect, see reaction (2.2.2). The stability of the closed complex is reflected by the promoter specific parameter k_{off}. In successful *initiation* RNAP unwinds the duplex DNA locally – we then reach the *open* complex (reaction (2.2.3)). The parameter k_{init} reflects the promoter specific efficiency of transcription initiation.

Elongation: After a successful transition to the open complex, RNAP starts to transcribe information content from DNA into RNA, at a first coding nucleotide (2.2.4). During *elongation* (2.2.5), it continues the synthesis of RNA complementary to the template DNA strand. The nucleotide wise advance of RNAP was experimentally confirmed by Abbondanzieri et al. (2005). These steps are separated by waiting times following an exponential distribution, determined by the parameter k_{elong} [2].

Interesting details in elongation are that RNAP may stall, slow down and pause on certain sequences (Wagner, 2000). Also, 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* becomes rate-limiting at highly efficient promoters such as the one from Figure 2.4(a) [3].

Termination can be summarized as follows. RNAP dissociates from DNA when recognizing a *terminator* sequence on the template strand. It then

[2] Note the reaction-base notation disregards the nascent mRNA.

[3] This aspect is incorporated in our executable models, yet not technically discussed in this thesis. For details, see (Hsu, 2002).

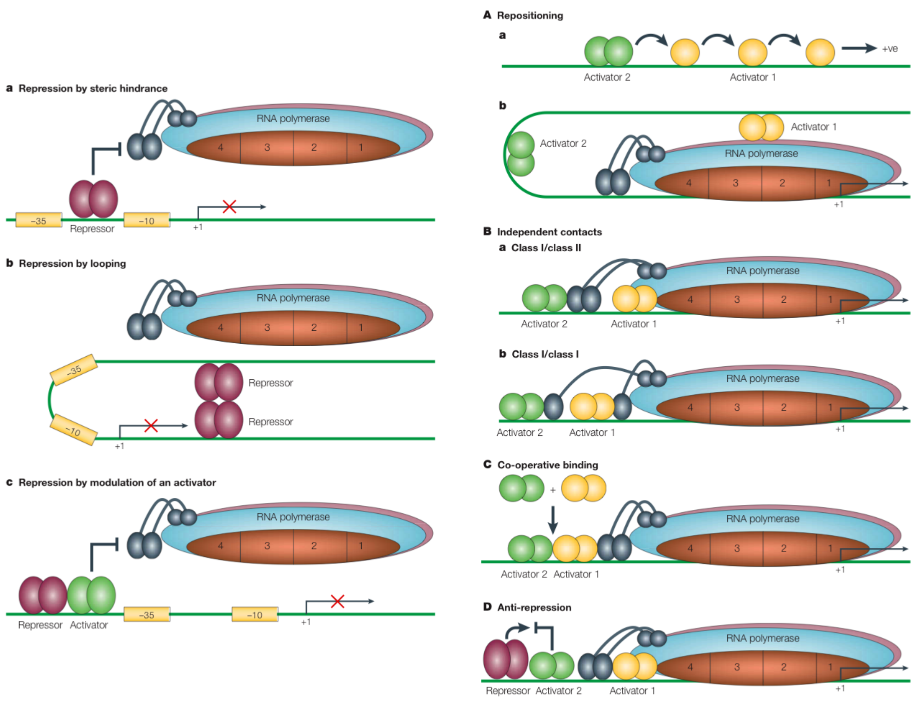


Figure 2.5: **Control of bacterial promoters.** Regulatory proteins apply as small catalog of molecular mechanisms when binding DNA for repression (left) and activation (right) of transcription initiation. From Browning and Busby (2004).

releases the completed transcript. Reaction (2.2.6) summarizes this *intrinsic* termination.

A regulatory detail not considered in this thesis 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 termination mechanism is called *rho-dependent*. In it, 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, for details see Banerjee et al. (2006) and Henkin (2000),

Regulation of initiation: Bacteria apply various strategies to use their genetic material with great effectiveness, in the correct amount and at the appropriate time. Transcription initiation is controlled by DNA binding

$$
\begin{aligned}
\text{Ribosome} + \text{mRNA}_{\text{RBS}} &\rightarrow_{k_{on}} \text{Ribosome} \cdot \text{mRNA}_{\text{RBS}} && (2.2.7)\\
\text{Ribosome} \cdot \text{mRNA}_{\text{RBS}} &\rightarrow_{k_{off}} \text{Ribosome} + \text{mRNA}_{\text{RBS}} && (2.2.8)\\
\text{Ribosome} \cdot \text{mRNA}_{\text{RBS}} &\rightarrow_{k_{init}} \text{Ribosome} \cdot \text{mRNA}_{1} && (2.2.9)\\
\text{Ribosome} \cdot \text{mRNA}_{\text{n}} &\rightarrow_{k_{elong}} \text{Ribosome} \cdot \text{mRNA}_{\text{n}+1} && (2.2.10)\\
\text{Ribosome} \cdot \text{mRNA}_{\text{terminator}} &\rightarrow_{k_{elong}} \text{Ribosome} + \text{Protein} && (2.2.11)\\
& \qquad +\text{mRNA}_{\text{terminator}} &&\\
\text{Degradosome} + \text{mRNA}_{\text{RBS}} &\rightarrow_{k_{d}} \text{Degradosome} \cdot \text{mRNA}_{\text{RBS}} && (2.2.12)
\end{aligned}
$$

Table 2.2: Reaction rules for translation and mRNA decay

proteins[4]. 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}.

The repertoire of regulatory interactions between proteins and bacterial DNA is actually small. Figure 2.5 illustrates essential molecular mechanisms of repression (left) and activation (right), depending on no more than two proteins.

In Chapter 3, we will detail on promoter repression by *steric hindrance*, depicted uppermost in the left column. It lead to the actual discovery [5] of gene regulation Jacob and Monod (1961), is the simplest mechanism of repression, and the most frequent. Steric hindrance is encountered at λ switch promoters, for which we develop π-calculus models in Chapter 8. These models refine the sketch from Section 1.3.4.

Among the mechanisms of activation, we will focus on cooperative binding of regulatory proteins, depicted as case (C) in the right column of Figure 2.5, and their positive control of RNAP. Cooperativity activates the λ switch promoter P_{RM}, and it is the key to transcription control in higher organisms (Müller-Hill, 2006; Ptashne, 2005).

2.2.2 mRNA, translation and degradation

Translation, the flow of mRNA encoded information into proteins, is again subdivided into initiation, (protein) elongation, and termination. Table 2.2 summarizes.

[4]Reviewed by Barnard et al. (2004) and Browning and Busby (2004).
[5]Awarded a Nobel prize in 1965.

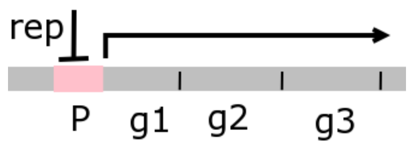
Initiation. Reaction (2.2.7) represents the initial step of ribosome binding and assembly on a dedicated mRNA sequence, the *r*ibosomal *b*inding *s*ite. As depicted in Figure 2.3 on page 29, the RBS is located nearby the mRNA's $5'$ end. The ribosome may dissociate readily. The parameter k_{off} in reaction (2.2.8) reflects its stability of binding, that depends on the agreement with an ideal sequence mRNA. The abbreviation mRNARna$_{RBS}$ in Table 2.2 refers to the $5'$ end of mRNA, including both the RBS and the *start sequence*, e.g. 'AUG' in Figure 2.3. The actual combination of RBS and start sequence determines the efficiency of translation initiation, see parameter k_{init} in reaction (2.2.9).

In *elongation*, the ribosome slides over mRNA, reads out information content and assembles a growing chain of amino acids. Unlike transcription that maps individual nucleotides, translation proceeds over mRNA in three letter words (*codons*), that each code for one amino acid. For simplification, it is however common practice in mathematical modeling to consider only the *average* elongation delay per mRNA nucleotide. It is accounted for by the parameter k_{elong} in equation (2.2.10). While illustrative comics of elongation are widespread in the biological literature as Alberts et al. (2002), the detailed internal functioning of ribosomes is only partially understood, see Frank and Agrawal (2000). Elongation ends as the ribosome reaches a dedicated *terminator* signal on mRNA (2.2.11). The protein is then released.

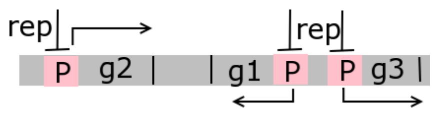
Degradation: Table 2.2 covers the initial step of degradation as reaction (2.2.12). After this, decay to proceeds with the same net orientation as translation. It hence does not affect translation that already initiatied. Note that this scheme approximates the *net* outcome of multiple degradation pathways in a phenomenological manner. For details, see Carrier and Keasling (1997), Grunberg-Manago (1999), and Steege (2000). For long, the detailed understanding of mRNA degradation lagged far behind that of other steps in gene expression, recent discoveries are reported by Carpousis (2002) and Regonesi et al. (2006).

2.3 Particular promoter arrangements

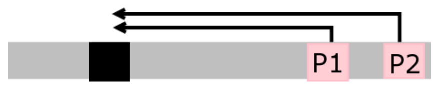
In this section we sketch specific cases in the arrangement of genes and promoters in bacterial genomes. They have important impact on expression patterns, and are difficult to explicitly represent in previous modeling approaches.



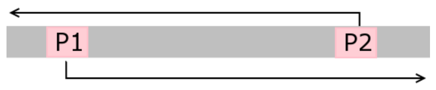
(a) ***operon***



(b) ***regulon***



(c) ***tandem promoters***



(d) ***convergent promoters***

Figure 2.6: Particular promoter arrangements. Horizontal arrows indicate the orientation of transcription.

Operons are sequences of several genes, co-transcribed in one go from a common promoter. Figure 2.6(a) presents an example. Operons yield *polycistronic* mRNA molecules. Each mRNA segment called *cistron* corresponds to one gene, hence codes for a different protein. While decay initiates at the common 5′ end as it does for monocistronic mRNA, translation is distinctly controlled across cistrons, each bearing their own ribosomal binding site and translation start signal. The translational efficiency across cistrons on the same mRNA can vary up to a factor of 1000 across cistrons (Ray and Pearson, 1974). Section 7.4.1 deals with modeling operons and polycistronic mRNA in our π-calculus with concurrent objects.

Regulons illustrated in Figure 2.6(b) are sets of genes, each transcribed from its individual promoter. The point that matters is that these promoters depend on the same regulatory signal. Assuming the signal reaches all promoters in this *regulon*, the genes are transcribed under the same circumstances.

In our illustration, the regulatory signal consists in repression by the same protein. Operons eliminate the need for multiple promoters. It appears interesting to compare the sensitivity of operons and regulons to a common regulatory signal, however we do not elaborate on this aspect in this thesis.

Tandem promoters are sketched in Figure 2.6(c). They lie close by on the same DNA strand, and control the transcription of the same gene(s). Tandem promoters offer two interesting aspects to fine tune expression levels. First, they can be activated independently, in response to different environmental signals. Second, the mRNA initiated from $P2$ is longer and thus contains more information, compared to that from $P1$. The different $5'$ ends can have important consequences, since both translation and degradation initiate there. The longer transcript may contain a second ribosomal binding site, or translation start signal. It may be more resistant to decay due to secondary structures, into which its longer $5'$ end folds.

The fine-tuning of translation efficiency by transcripts of the same gene, initiated at alternative promoters is an interesting regulatory strategy. It is commonly applied by bacterial *viruses* as λ, that after infecting a bacterial cell, must take a decision to either enter their dormant state, or start multiplying at the expense of their host cell (Schmeissner et al., 1980). The most prominent example of alternative promoters however is *ribosomal* RNA *genes* in bacteria. These account for 90% of transcription in rapidly growing cells (Dennis et al., 2004; Paul et al., 2004). Beyond bacteria, alternative promoters are also relevant to gene regulation in *higher organisms.*

In *humans*, an interesting example is the control of alternative promoters for the cancer-inducing Epstein-Barr virus, see Thompson and Kurzrock (2004) and Young and Rickinson (2004). This virus infects over 90% of the world population. Its switch from the dormant to the cancer-inducing mode depends on the control of alternative promoters.

We will discuss the modeling of a tandem promoter in the π-calculus in Section 7.4.2, however not enter the details of parameterization and simulation.

Convergent promoters are the fourth promoter arrangement illustrated in Figure 2.6(d). The crucial point about convergent promoters arranged on opposing strands of DNA is that RNAP starting from them proceed over the two strands with converging orientations. However transcriptional traffic over double stranded DNA occurs on a *single lane, two way street.* 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*, and can be used as a regulatory mechanism (Shearwin et al., 2005). This thesis only deals with single stranded DNA, leaving the treatment of the concurrent control of transcriptional interference in the π-calculus to future work.

2.4 Quantitative control

Cell-to-cell variability contronts biologists with intriguing questions. A frequent observation is that cells from a genetically identical population exhibit widely differing behavior, although being exposed to the same, precisely con-

parameter (reaction)	value	comment
Transcription of DNA		
k_{on} (in 2.2.1)	0.1 sec^{-1}	binding: equally fast for all promoters
$\frac{k_{on}}{k_{off}}$ (in 2.2.1 and 2.2.2)	10^6 to 10^9	unbinding: promoter specific
k_{init} (in 2.2.3)	10^{-3} to 10^{-1} sec^{-1}	initiation: promoter specific
k_{elong} (in 2.2.5)	$\frac{1}{30}$ sec^{-1}	elongation speed: 30 nucleotides/sec
Repression		
repressors: $\frac{k_{on}}{k_{off}}$	10^7 to 10^{11}	attachment to operators ranges from seconds to several bacterial generations
Translation and degradation of mRNA		
	1 to 100	gene specific *mean* protein crop per transcript, depending on k_{off}, k_{init} and k_d
k_{elong} (in 2.2.10)	$\frac{1}{100}$ sec^{-1}	elongation speed: 100 nucleotides/sec

Table 2.3: Quantitative control of gene expression. Reaction numbers refer to Tables 2.1 and 2.2.

trolled environment. Such differences may lead to visibly distinct behavior or appearance of the cells. In quantitative terms, they correspond to fluctuation of protein amounts, within a cell over time, as well as across cells in a population snapshot.

Part of the variability in gene expression originates from the inherently stochastic nature of the biochemical reactions, combined with low numbers of molecules in regulatory events (Sneppen and Zocchi, 2005). Other effects are due to the specific quantitative control for a given gene of interest, i.e. the efficiencies of transcription and translation initiation. Tables 2.1 and 2.2 listed the main parameters as reaction labels: k_{on}, k_{off}, k_{init}, k_{elong}, and k_d.

In this section we first consider value ranges for these parameters. We then discuss the impact of actual parameter combinations on the overall dynamics of gene expression, that can be considerable.

2.4.1 Parameter ranges

Table 2.3 reports some ranges of relevant values. The quantitative properties of *promoters* vary greatly. On some RNAP falls off the closed complex within fractions of seconds, while on others with favorable k_{off} parameter, it may remain stably bound for minutes. Transition to the open complex depends on the parameter k_{init} in reaction (2.2.3), at strong promoters it occurs within a second, while at weak ones after minutes. As a consequence, the frequency of transcription per gene varies from one per second (ribosomal RNA) to one per cell generation (certain regulatory proteins). RNAP elongates transcripts with an average rate of 30 nucleotides per second.[6] With an average gene length of 1000 nucleotides, this combines to an average transcript elongation delay around 30 seconds.

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 operator sequences for several bacterial generations of each 30 min – 1 hour; while at less specific sequences it falls off within 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. As a consequence of degradation, average mRNA lifetimes range from few seconds to 30 minutes (Grunberg-Manago, 1999).

2.4.2 Translational bursting

The *average* number of proteins produced from a single mRNA is gene specific, typically ranges are between 1 and 100. It is important to consider the *fluctuations* of protein crops around these averages, for transcripts of the same gene. These depend on the outcome of the race between degradation and translation. When translation initiates efficiently, and the crop for the transcript is high, ribosomes queue on mRNA, and all proteins are released soon after transcript completion. With long spacings between transcriptions, this combines to *translational bursts*, i.e. rapid release of comparatively high numbers of proteins.

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

[6]This is a considerable simplification of the quantitative details of individual steps in mRNA elongation. Greive and von Hippel (2005) review current methods of quantitative assessment.

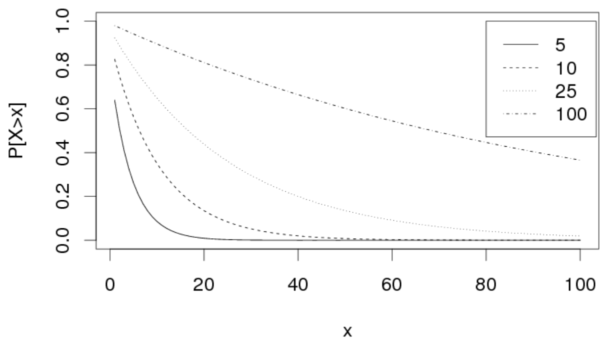


Figure 2.7: A *geometric distribution* characterizes the fluctuations around the mean crop of proteins per mRNA. $P[X > x]$ for different mean values.

these means. Combined with the fact that most genes are only transcribed occasionally, translational bursting becomes prevalent. Figure 2.8 (left) on page 41 illustrates it. It contrasts the resulting dynamics at protein level for average crops per transcript of 10 proteins (up) with one of 1 (low). Importantly, in both cases the combination with transcriptional efficiencies yields the same *average* protein level.

Translational bursting significantly contributes to stochasticity in gene expression, which has attracted much attention in recent years, reviewed by Kaern et al. (2005), Paulsson (2005), and Raser and O'Shea (2005). 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, as in the experiments of Kobiler et al. (2005) and Golding et al. (2005).

Geometric distribution. We consider the geometric distribution, that illuminates the increase of the stochastic variability tied to increasing average protein crops per transcript. 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 x 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 2.7 illustrates the complementary 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.

2.5 Related models

We report on studies of bacterial gene expression that our work draws from, dealing with modeling, simulation and experimental investigation. After this, we briefly sketch software packages for stochastic simulation, and comment on continuous modeling approaches.

Carrier and Keasling (1997) elucidated the relation between molecular actors in mRNA processing. Based on the prevalent narrative theories of prokaryotic mRNA, they developed a mechanistic model of discrete interactions between the various sets of enzymes contributing to *degradation*, and the initiation of *translation* by ribosomes. The authors performed simulations of the alternative models and tested their ability to reproduce dynamic effects known from wet lab experiments. We will follow their conceptional scheme in mRNA decay, that is rarely explicitly rendered by other models in the field.

McAdams and Arkin (1997) attracted wide attention with a scheme for simulation of gene expression. It combines a continuous model of transcription initiation in the tradition of Shea and Ackers (1985) with a stochastic account of transcript elongation and subsequent processing of mRNA (based on Gillespie's algorithm). We adhere to this conceptional scheme for the race between the initiation of translation versus degradation. Arkin, Ross, and McAdams (1998) applied this generic model to the initial decision between the lysogenic and lytic pathways bacteriophage lambda. We discuss points that we deem critical in their parameterization on page 52. Namely, they neglected the importance of distinct translational efficiencies, giving rise to differences in stochastic fluctuations in protein levels.

Kierzek, Zaim, and Zielenkiewicz (2001) systematically addressed this aspect, for bacterial gene expression in general. They clarified the contribution of translational efficiencies to noise at the protein level in stochastic simulation while systematically varying translation and transcription initiation frequencies.

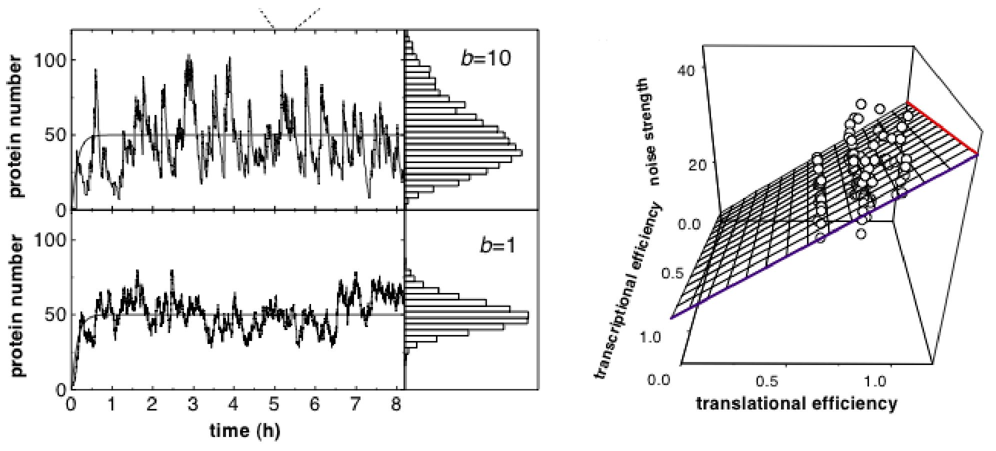


Figure 2.8: **Translational bursting.** The experiments of Ozbudak et al. (2002) confirmed the hypothesis consolidated through modeling and simulation, that efficiently translated transcripts together with rare transcription initiations are a major source of stochasticity in bacterial gene expression. (left) simulation data, (right) data points .

Ozbudak et al. (2002) confirmed Kierzek et al.'s predictions experimentally. They systematically varied single bases in the sequences of Dna determining transcription initiation efficiencies, and those transcribed into the translation initiation sequence (Rbs) on mRna. For these modified sequences, they quantified the resulting initiation efficiencies. In addition, they monitored the resulting dynamics at protein level for different combinations of promoter and Rbs sequences. Their experimental observations support the proposal of Kierzek et al. (2001) that stochastic noise increases with the combination of high translational efficiency and rare transcription initiation. Figure 2.8 summarizes the observations of protein levels in silico (left) and vitro (right).

Software packages for stochastic simulation of gene expression While most of the above studies were performed with ad hoc implementations of algorithms for stochastic simulation, in recent years several dedicated software packages have been proposed.

STOCKS. Kierzek (2002) contributed a tool used for his own simulation studies, that executes chemical reaction rules specified in a reaction based syntax according to Gillespie's algorithm.

Dizzy. [7] This tool by Ramsey et al. (2005) and Orrell et al. (2005) implements current variants of Gillespie's algorithm of improved efficiency, for large-scale simulation of genetic networks. It includes explicit support for the multi-step reactions in transcription and translation, and reusable templates for the creation of large-scale models of regulatory dependencies between genes. Models are specified in a rule-based language via a graphical interface. The paper by Ramsey et al. (2005) includes a comprehensive overview of software packages for stochastic simulation.

STOCHSIM. Multi-state molecules are one of the central ideas realized in StochSim, a general purpose biochemical simulator described in Morton-Firth (1998) and Morton-Firth and Bray (1998). StochSim represents individual molecules interacting stochastically, however with time steps at even pace. Multi-state molecules possess a set of binary flags, indicating possible modifications as the presence of a smaller molecule.

BioNet. [8] For completeness we list the stochastic ODE solver of Adalsteinsson et al. (2004). It proved insightful through the recent simulation studies of Guido et al. (2006), dealing with the construction of a synthetic bacterial promoter.

Deterministic models. Continuous deterministic models have a decade-long tradition in chemistry, biochemistry, or physics. They are mostly represented in the unifying framework of differential reactions (Voit, 2000). By their design deterministic models abstract from the nondeterministic behavior of individual molecules. Their aim is to reproduce average behavior of systems with huge numbers of components, where discrete interactions may be disregarded. Although their predictive power with respect to detailed level as consider in our work is limited, such approaches have indeed proven insightful to gene expression, see e.g. Leloup and Goldbeter (1999), and von Dassow et al. (2000).

[7] `http://magnet.systemsbiology.net/software/Dizzy`
[8] `http://x.amath.unc.edu/BioNetS`

CHAPTER 3

Transcription Regulation at the Lambda Switch

Bacteriophage λ is a virus which infects the bacterium *Escherichia coli.* Injecting its genome into the bacterial cell, two developmental pathways are possible, as illustrated in Figure 3.1. In *lytic growth* the viral genome uses the molecular machinery of the bacterial cell to produce new viruses and eventually burst the host. Alternatively, the viral genome gets integrated into the bacterial genome. Note the highlighted segment within the bacterial genome in Figure 3.1. The only viral protein expressed after the decision for lysogeny is the λ repressor. It represses the expression of all other viral genes. The host cell is now immune against further infections. Both the viral genome, and the immunity, are subsequently transmitted in a passive way. This state called *lysogeny* is extremely stable, and usually maintained for many bacterial generations. Spontaneous transitions from lysogeny to the state of lytic growth would occur about once every 5000 years for a single bacterial cell (Dodd et al., 2005). Taking into account that the bacterium divides into two daughter cells within less than an hour, the lysogenic state is extremely stable.

But surprisingly, upon an environmental signal the phage genome can efficiently become re-activated – this is called *induction.* Now, the bacterium switches from lysogeny into the phase of lytic growth. The viral genome is extracted from the host's, and uses the cell machinery to produce a fresh crop of viruses. This unavoidably leads to the *lysis*, or destruction of the host cell.

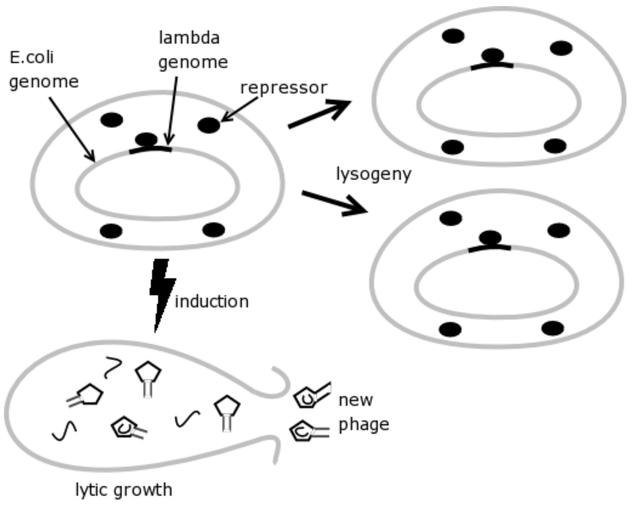


Figure 3.1: Two pathways of λ infected E.coli bacterium: lysogeny and lytic growth.

3.1 Molecular events

What happens during induction, as well as the maintenance of lysogeny, crucially depends on the control of transcription initiation within O_R, the *r*ight *o*perator region of phage λ's genome. O_R is commonly referred to as *the* λ switch.

The control of transcription initiation at the λ switch illustrates phenomena of *cooperativity*, which are increasingly important for gene regulation in higher forms of life (Ptashne and Gann, 2002). Cooperative enhancement of a reaction between two molecular actors means that its strength is enhanced by a third, otherwise independent actor. We will see two instances of cooperative enhancement at the λ switch: *positive control* and *cooperative binding.*

Genes and promoters: By the λ switch we refer to a segment of phage λ's genome, that comprises the control regions for the genes cI and cro, illustrated Figure 3.2. Their transcription initiates at the promoters $\mathrm{P_{RM}}$ and $\mathrm{P_R}$ respectively, the transcripts are subsequently translated to the *proteins* Rep and Cro.

Cro and Rep proteins appear in two forms, as *dimers* and *monomers* which can be distinguished in Figure 3.4. When expressed they first appear as monomers. They can only bind to DNA after having associated pairwise. Dimers are unstable, unless bound to DNA they soon dissociate back to

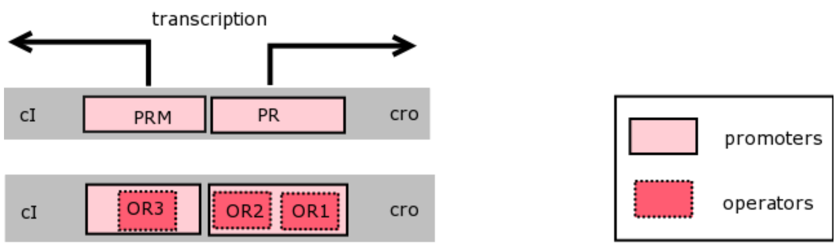


Figure 3.2: A spatial view of the λ switch, a segment of phage lambda's genome.

gene	**protein**	**promoter**
cI	Rep	P_{RM}: *p*romoter for synthesis of *r*epressor during *m*aintenance of lysogeny
cro	Cro	P_R: *r*ight *p*romoter

Figure 3.3: Genes, proteins, and promoters at the λ switch

monomers. The higher the protein concentration in the cell, the higher is the degree of dimerization.

In lysogeny, the regulatory network is characterized by a high number of Rep and negligible amount of Cro proteins; these frequencies are inverted during lytic growth. The environmental signal upon induction leads to a massive destruction of Rep proteins. P_R then becomes activated automatically, while transcription from P_{RM} ceases. These are consequences of the network controlling transcription initiation.

Repression of promoters by steric hindrance: The regulatory proteins Rep and Cro bind to the three neighboring *operator* regions O_{R1}, O_{R2}, and O_{R3}. By doing so, they control RNAP access to the promoters. As Figure 3.2 indicates O_{R1} and O_{R2} both overlap the promoter P_R, while O_{R3} lies within P_{RM}. A protein bound within a promoter blocks recognition of the promoter by RNAP. This principle is called *steric hindrance.* The typical constellations are sketched in Figure 3.4. Note that all bindings are *reversible*, i.e. the proteins dissociate from the DNA strand after some time. Similarly, RNAP frequently falls off a promoter without initiating transcription.

The maintenance of lysogeny depends on the presence of a sufficient amount of repressor, that is predominantly bound at O_{R1} and O_{R2}. This impedes RNAP binding to P_R. As a consequence, Cro and all other viral

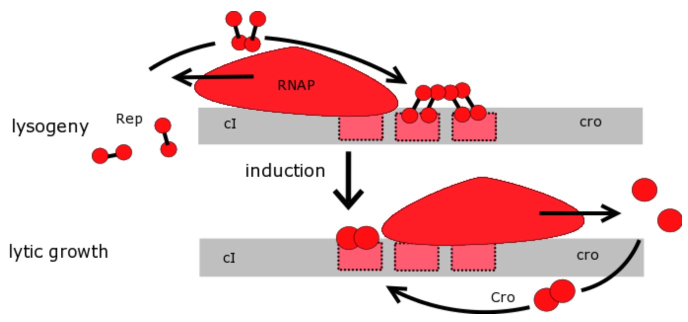


Figure 3.4: Network states during lysogeny and lytic growth. In lysogeny, Rep attached to either or both of the binding sites O_{R1} and O_{R2} blocks recognition of the promoter P_R by RNAP, and thus prevents transcription of the gene *cro*. At the same time, interactions between Rep at O_{R2} and RNAP at P_{RM} stimulate transcription of the gene `cI`, which allows for the production of new Rep.

genes are repressed.

Cooperative enhancement of repressor binding at O_{R2} : The intrinsic binding affinity of Rep for O_{R1} is tenfold higher than for O_{R2} and O_{R3}. Thus, Rep is more likely to be found at O_{R1}. Furthermore, Rep at O_{R1} significantly favors binding of another Rep to O_{R2} – this is what we call *cooperative binding*. One could say that the λ repressor at O_{R1} *recruits* another to O_{R2} (Ptashne and Gann, 2002).

Positive control of transcription initiation is needed for virtually all genes. It refers to the fact that RNAP bound to a promoter needs the help of regulatory proteins in order to successfully initiate transcription. At P_{RM}, RNAP's initiation frequency increases through a physical contact with Rep bound at O_{R2}. This second instance of cooperative enhancement, called *positive control*, is decisive for maintenance of the lysogenic state. Without it RNAP would rather fall off the inherently weak promoter P_{RM} than start to transcribe.

The production of Rep ceases once the available quantities fill the operator site site O_{R3}, in addition to O_{R1} and O_{R2}. At this point Rep inhibits its own production by steric hindrance of P_{RM} in a negative feedback loop.

Upon *induction*, the number of repressors rapidly decreases due to an external signal, so that O_{R1} and O_{R2} become more and more likely to remain vacant. Now polymerases find frequent opportunities to bind to P_R. As P_R

$$
\begin{array}{rcll}
TF + \mathrm{O_{Ri}} & {}_{k_{off}}\rightleftharpoons^{k_{on}} & TF \cdot \mathrm{O_{Ri}} & (3.1.1) \\
\mathrm{RNAP} + \mathrm{P_{RM}} & {}_{k_{off}}\rightleftharpoons^{k_{on}} & (\mathrm{RNAP} \cdot \mathrm{P_{RM}})_{\mathrm{closed}} & (3.1.2) \\
(\mathrm{RNAP} \cdot \mathrm{P_{RM}})_{\mathrm{closed}} & \rightarrow_{k_{init}} & (\mathrm{RNAP} \cdot \mathrm{P_{RM}})_{\mathrm{open}} & (3.1.3) \\
\mathrm{RNAP} + \mathrm{P_{R}} & {}_{k_{off}}\rightleftharpoons^{k_{on}} & (\mathrm{RNAP} \cdot \mathrm{P_{R}})_{\mathrm{closed}} & (3.1.4) \\
(\mathrm{RNAP} \cdot \mathrm{P_{R}})_{\mathrm{closed}} & \rightarrow_{k_{init}} & (\mathrm{RNAP} \cdot \mathrm{P_{R}})_{\mathrm{open}} & (3.1.5)
\end{array}
$$

Table 3.1: Reaction rules for protein-DNA interactions at the λ switch

is inherently a *strong promoter*, these bindings rapidly ensue transcription, followed by the production of Cro proteins.

3.2 Quantitative control

The stochastic π-calculus assumes rates that determine the speed of reactions. In this section, we discuss how to distill such rates from the literature. The resulting parameters are summarized in Table 3.2.

Table 3.1 summarizes the reactions for RNAP-promoter interactions, and protein binding to operators and at the λ switch, i.e. $TF \in \{Rep, Cro\}$ and $i \in \{1, 2, 3\}$. It is however not straightforward to include the mutual affections of these reactions. Table 3.2 explicits all side conditions.

In our system reversible binding reactions are frequent, such as by λ repressor to the operator $\mathrm{O_{R1}}$:

$$Rep + \mathrm{O_{R1}} \; {}_{k_{off}}\rightleftharpoons^{k_{on}} Rep \cdot \mathrm{O_{R1}} \qquad (3.2.1)$$

This bidirectional reaction converges to an equilibrium, in which the number of reactants on both sides remains constant. The *association* constant k_{on} determines the speed of the association reaction. It measures the number of Rep-$\mathrm{O_{R1}}$-pairs that form complexes per mol and second. For the case of regulatory proteins the *association* rate constant k_{on} has been experimentally determined – see Berg et al. (1981); Wagner (2000); Sneppen and Zocchi (2005). It is given by the net rate with which a protein locates its target site on DNA:

$$k_{on} = \frac{10^8}{\text{mol sec}} \qquad (3.2.2)$$

We assume this value for all combinations of proteins and operator sites [1].

[1]This constant exceeds three dimensional diffusion by two orders of magnitude, and subsumes a number of mechanisms of target site location by proteins. In its search process

reaction	rate	protein	site	condition	value	(channel, function)	reference
	k_{on}	any	any	overlapping promoter vacant	0.098	pro,bind	(Berg et al., 1981; Wagner, 2000)
		Rep	O_{R1}		0.155	or1, timerRep	(Ackers et al., 1982)
		Rep	O_{R2}	no Rep at O_{R1}	3.99	or2,timerRepLow	(Ackers et al., 1982)
		Rep	O_{R2}	Rep at O_{R1} (cooperative binding)	0.155	or2,timerRepHigh	(Shea and Ackers, 1985)
(3.1.1)	k_{off}	Rep	O_{R3}		20.22	or3,timerRep	(Koblan and Ackers, 1992)
		Cro	O_{R1}		2.45	or1,timerCro	(Shea and Ackers, 1985)
		Cro	O_{R2}		2.45	or2,timerCro	(Shea and Ackers, 1985)
		Cro	O_{R3}		0.29	or3,timerCro	(Shea and Ackers, 1985)
(3.1.2)	k_{on}	RNAP	P_{RM}	O_{R3} vacant	0.098	rnap,bind	(Berg et al., 1981; Wagner, 2000)
	k_{off}	RNAP	P_{RM}		0.788	prm,unbind	(Li et al., 1997)
(3.1.3)	k_{init}	RNAP	P_{RM}	Rep at O_{R2} (positive control)	0.086	prm,highTimer	(Li et al., 1997)
		RNAP	P_{RM}	no Rep at O_{R2}	0.005	prm,lowTimer	(Li et al., 1997)
(3.1.4)	k_{on}	RNAP	P_R	O_{R1} and O_{R2} vacant	0.098	rnap,bind	(Berg et al., 1981; Wagner, 2000)
	k_{off}	RNAP	P_R		0.155	pr,unbind	(Hawley et al., 1985)
(3.1.5)	k_{init}	RNAP	P_R		0.05	pr,initiate	(Hawley et al., 1985)

Table 3.2: Parameterizing the λ switch

The *dissociation* constant k_{off} specifies the speed of the de-complexation. It measures the proportion of complexes that is resolved per second. As we will see, for the case of Rep binding to $\mathrm{O_{R1}}$ we can assume it to be $k_{off} = \frac{0.155}{\mathrm{sec}}$.

However, it is less obvious to infer such dissociation rates the literature. What is determined experimentally for such reactions is mostly *Gibbs free energy* ΔG – a notion from thermodynamics. The value of ΔG quantifies the effort necessary for decomplexation. In the concrete example of $\mathrm{O_{R1}}$, Ackers et al. (1982) provide $\Delta G = -12.5\frac{kcal}{\mathrm{mol}}$. This energy is negative, reflecting that binding requires an effort by the environment, while unbinding happens voluntarily. Non cooperative binding of *Rep* at the weaker binding site $\mathrm{O_{R2}}$ yields a value of $\Delta G = -10.5\frac{\mathrm{kcal}}{\mathrm{mol}}$, for $\mathrm{O_{R3}}$ we obtain $\Delta G = -9.5\frac{\mathrm{kcal}}{\mathrm{mol}}$. Note that a smaller value indicates *stronger* binding, and that a difference of $1kcal$ ensues a tenfold difference in binding strength.

Gibbs free energy correlates with the equilibrium constant K_{eq} of the binding reaction, which expresses the quantities of unbound pairs Rep and $\mathrm{O_{R1}}$ compared to complexes $Rep \cdot \mathrm{O_{R1}}$ in equilibrium. The relationship is expressed through the equation:

$$K_{eq} = \exp(\frac{-\Delta G}{R \cdot T}) \tag{3.2.3}$$

where $R = 1.9872\frac{cal}{\mathrm{mol\ Kelvin}}$ is the universal gas constant and $T = 310.15$ Kelvin is the absolute temperature at which the experiments were performed (it corresponds to 37 Celsius).

The equilibrium constant K_{eq} represents the ratio of association and dissociation rate constants as shows the following kinetic equation:

$$K_{eq} = \frac{k_{on}}{k_{off}}\ \mathrm{mol} \tag{3.2.4}$$

The experimental data on Gibbs energy together with equations (3.2.2), (3.2.3), and (3.2.4) are sufficient to compute the dissociation rate k_{off} by straightforward arithmetics[2].

The rate constants k_{on} and k_{off} we have met so far are *macroscopic* – as in chemical kinetics. They do not depend on the actual numbers of

a protein first diffuses three-dimensionally through the cytoplasm, hits the DNA and subsequently slides along the DNA, rapidly scanning it for its specific site. A model explaining this has been proposed by Slutsky and Mirny (2004).

[2]The following set of equations determines the values of all rates for the example:

$$\begin{array}{rcl@{\qquad}rcl}
\Delta G & = & -12.5 \cdot 10^3 \mathrm{cal/mol} & K_{eq} & = & \exp^{-\Delta G/(R\ T)} \\
R & = & 1.9872\ \mathrm{cal/(mol\ Kelvin)} & k_{on} & = & 10^8/(\mathrm{mol}\ sec) \\
T & = & 310.15\ \mathrm{Kelvin} & k_{off} & = & k_{on}/K_{eq}\ \mathrm{mol}
\end{array}$$

	ΔG	k_{off}	binding strength
O_{R1}	-12.5	0.155	strongest
O_{R2} (coop)	-12.5	0.155	
O_{R2} (isolated)	-10.5	3.99	
O_{R3}	-9.5	20.22	weakest

Figure 3.5: Parameters for λ repressor binding to the three operator sites

molecules, but on concentrations. Gillespie's algorithm, however, and thus the stochastic π-calculus use *mesoscopic* rate constants as their stochastic rates. These refer to actual numbers of molecules and are determined from their macroscopic counterparts as follows:

$$k_{on}^{meso} = \frac{k_a}{A\ V}, \qquad k_{off}^{meso} = k_{off},$$

where $A = 6.023 \cdot 10^{23}$ is Avogadro's number – i.e. number of molecules per mole – and $V = 1.7 \cdot 10^{-15} l$ is the E. coli cell volume. We need to divide by $A \cdot V$ for reactions involving two reactants, such as binding; for reactions that transform a single reactant as unbinding, the macroscopic and mesoscopic rates coincide. Note that we assume the cell volume to be constant while ignoring cell growth. Evaluating our equation yields the following final rates for the considered example reaction between O_{R1} and Rep:

$$k_{on}^{meso} = 0.098/\sec \qquad k_{off}^{meso} = 0.155/\sec$$

We can now quantify the effects of *cooperative binding* between repressors at O_{R1} and O_{R2}. Cooperativity adds a favorable term of $-2\frac{\text{kcal}}{\text{mol}}$ to the Gibbs binding energy of Rep at O_{R2} (Shea and Ackers, 1985) [3]. Due to the exponential relation between free energies and equilibrium constants this massively strengthens the binding: the mesoscopic dissociation rate k_{off} for O_{R2} decreases from 3.99 to 0.155, the same value as for O_{R1}. Table 3.5 summarizes.

Finally, we need rates for transcription initiation, in which we follow the scheme of McClure (1985): After binding, the complex of RNAP and promoter P undergoes an irreversible transition from a *closed* state into an *open* one, in which the two strands of DNA have locally been separated. Then transcription proceeds.

$$\text{RNAP} + P \rightleftharpoons^{K_{eq}} (\text{RNAP} \cdot P)_{\text{closed}} \rightarrow_{k_{init}} (\text{RNAP} \cdot P)_{\text{open}} \tag{3.2.5}$$

[3] Cooperativity also has a helping effect to binding at O_{R1}, however we chose to neglect this in our model as the effect at O_{R2} predominates.

The k_{init} rates for the promoter $\mathrm{P_R}$ and $\mathrm{P_{RM}}$ can be found in Hawley et al. (1985) and Li et al. (1997). *Positive control* of RNAP by repressor binding at $\mathrm{O_{R2}}$ increases the k_{init} rate of $\mathrm{P_{RM}}$ roughly tenfold. Note that the dissociation rate of RNAP binding at $\mathrm{P_{RM}}$ is not affected, which distinguishes this mechanism from cooperative binding of regulatory proteins.

Throughout this paper we assume a constant RNAP concentration of $c = 30 \cdot 10^{-9}$ mol according to Shea and Ackers (1985). This corresponds to a population of circa 30 RNAP molecules via the simple calculation $\#RNAP = c \cdot V \cdot A = 30.7$, with A and V as above.

Finally, we assume the rate at which repressor monomers associate to dimers to be $0.025\,\mathrm{sec}^{-1}(nM)^{-1}$ (mesoscopic: 0.048) while setting the dissociation rate to $0.5/\,\mathrm{sec}$ following Bundschuh et al. (2003).

3.3 Related models

The λ switch might be the case of gene expression that is most frequently approached by modeling and simulation studies. The central questions addressed at a systems level are:

- How is the initial decision between lytic and lysogenic pathways determined, after infection?
- How is the lysogenic state maintained?

We review hallmark studies in modeling and simulation of bacteriophage λ, in chronological order. Note that the questions addressed by these models are distinct from ours, as well as the modeling frameworks applied. Also, covering a maximal number of existing models remains beyond our intention.

The first quantitative model of λ switch regulation through protein binding, to the best of our knowledge, is that of Ackers, Johnson, and Shea (1982). They started from experimental data of Johnson, Meyer, and Ptashne (1979) reporting the concentration of repressor required for half-saturation of operator sites, and converted these into a set of thermodynamic free energies. Ackers et al. derived the probability to find repressor proteins bound to the operator sites $\mathrm{O_{R1}}$, $\mathrm{O_{R2}}$ and $\mathrm{O_{R3}}$, distinguishing eight possible binding configurations. This allows to calculate the degrees of promoter repression, depending on repressor levels. Importantly, the model also allows to predict promoter repression under the assumption that repressors at $\mathrm{O_{R1}}$ and $\mathrm{O_{R2}}$ do *not* bind cooperatively. This data supported the view that would have been difficult to confirm experimentally, that cooperativity improves $\mathrm{P_R}$ repression in lysogeny.

Shea and Ackers (1985) extend this model, to include binding of Rnap to the promoters as well as Cro's to the operators. The number of possible configurations increases from 8 to 40 by the additional information. Importantly, in contrast to the previous, this model version is dynamic. It was used to predict the maintenance of lysogeny, and induction of the lytic pathway.

The model of McAdams and Shapiro (1995) was the first to follows λ's dynamics from the moment the virus infects the host cell, through the decision between the lytic and lysogenic pathway. It does so in a *qualitative* manner, and concludes that detailed simulation of genetic mechanisms are necessary to model regulatory circuits quantitatively.

Arkin, Ross, and McAdams (1998) deal with the initial decision between the lysogenic and lytic pathways bacteriophage lambda in *quantitative* terms. This model integrates a promoter control model following Shea and Ackers (1985) with a stochastic account of transcript elongation and mRna processing.

The next two studies deal with the maintenance of the lysogenic state. Aurell, Brown, Johanson, and Sneppen (2002) mathematically deal with the remarkable stability at which it is inherited from one bacterial genetics to the next. Their investigation includes experimental data of Little et al. (1999), that report robustness of the switch with respect to modifications of protein binding strengths to the operators O_{R1}, O_{R2}, O_{R3}. Aurell et al. conclude that the knowledge available at the time of their work does not yet allow to account for the experimentally observed stability, and thus appears incomplete. Aurell and Sneppen (2002) develop a generic framework for the investigation of the stability of epigenetic states (i.e. passing on lysogeny from one generation to the next), and apply it to λ.

Other models centering on *induction* were proposed by Chung and Stephanopoulos (1996), Tian and Burrage (2004), and Zhu, Yin, Hood, and Ao (2004), while Thomas et al. (1976) and Thieffry and Thomas (1995) dealt with the initial lysis-lysogeny decision in logical formalisms.

Discussion of parameterization in Arkin et al. (1998)

It is worthwhile mentioning that while the work of Arkin et al. was an important source of inspiration for our own model of transcription and translation, in what regards the parameterization of the λ model our choices widely differ from theirs. Regarding *dimerization*, we follow Bundschuh et al. (2003), who corrects the rates of Arkin et al. by a factor nano, i.e. nine orders of magnitude. Without this, it would not have been possible to simulate the dynamics of dimer formation and breakage. Regarding *transcription control*, our parameter set developed in Section 3.2 is novel[4]; the level of resolution of Arkin et al. disregards those aspects that are of importance to us. Regarding *protein decay*, they assume repressor half-lives of 40 min. However to the best of our knowledge, the λ repressor is remarkably stable in the absence of the signal for induction. We follow Parsell et al. (1990) who report half-lives in time scales of 10 hours or more, i.e. repressor degradation is not relevant to our model since we consider shorter time scales[5]. Last it was a choice of this thesis *not* cover the quantitative control of *translation* for the λ proteins. However, we report that to the best of our knowledge, we identified important *flaws* in the parameter set of Arkin et al.. Their work assumes an average crop of 10 proteins for all transcripts, including those of the cI gene transcribed from P_{RM}. This clearly contradicts to the findings of Balakin et al. (1992) that this transcript belongs to a minority with unusual translation initiation properties, the so called *leaderless* mRna. P_{RM} initiated cI transcripts lack a ribosomal binding sites, translation initiates directly on the start codon. This results in poor efficiency, yielding in average 1 protein per cI transcript. Ribosomal assembly on such transcripts follows a peculiar pathway, see Moll et al. (2002), Moll et al. (2004). It was previously not reported for bacterial mRna– however exhibits parallels to translation initiation in archeal and eukaryotic cells. We deem these choices in parameterization to have important impact on the simulation results of Arkin et al., notably on the prediction of variability in protein levels.

[4]An similarly derived by Saiz and Vilar (2006).

[5]Degradation would importantly matter if considering induction of the lytic pathway.

Part II

Modeling Language and Concepts

CHAPTER 4

A Stochastic Pi Calculus for Concurrent Objects

The π-*calculus* by Milner, Parrow, and Walker (1992) is an expressive formal language for describing systems of concurrent actors and their interactions. It supports a minimalistic set of operators, describing concurrent systems on a high level of abstraction, while ignoring specific aspects of applications. The synchronous π-calculus is more expressive than the asynchronous π-calculus, in that it can express centralized control (Palamidessi, 2003). Distributed programming languages are the prototypical application of the asynchronous π-calculus, since no centralized control is available there.

The *stochastic* π-calculus of Priami (1995) adds a temporal perspective to the synchronous π-calculus. The speed of reaction x is specified by a stochastic parameter, that is assigned to channel x. Such parameters define exponential distributions of waiting times, associated with reactions on the channel. A one-step reduction based semantics[1] is given in Priami et al. (2001). This semantics can be implemented by the simulation algorithm of Gillespie (1976).

The two implementations available so far, SPiM (Phillips and Cardelli, 2004) and BioSpi (Priami et al., 2001), were already applied in a number of simulation cases studies in systems biology beside those presented in this thesis (Lecca et al., 2004; Blossey et al., 2006; Kwiatkowska et al., 2006). From the modeling perspective in systems biology, the minimality of the π-calculus is sometimes unfortunate. Objects help specifying the interfaces of

[1]The alternative SOS style semantics was used in the original π-calculus (Milner et al., 1992) and in the original stochastic π-calculus (Priami, 1995).

concurrent actors, so that they can be refined by inheritance. This permits to add new functionality to concurrent actors, while remaining consistent with their previous interface.

The lack of object-oriented features in the asynchronous π-calculus was perceived early from the programming language perspective. Vasconcelos and Tokoro (1993) proposed a suitable extension of the asynchronous π-calculus to remedy the situation, that lead to the development of the TyCO programming language (Paulino et al., 2003). The main idea is to extend the asynchronous π-calculus by input patterns with function names, in order to define object-oriented programming abstractions.

In this chapter, we extend the stochastic π-calculus by input patterns with function names. In doing so, we are driven by the motivation to define notions of concurrent objects, that we introduce later in Chapter 5. These notions will be more general than those in TyCO, and are based on synchronous communication. This increase expressiveness is required for our modeling studies. Beside discovering the usefulness of input pattern for the synchronous π-calculus, the main contribution of this chapter is a *stochastic semantics* for the π-calculus with input patterns. It assigns stochastic rates to pairs of channel and function names. The challenge is to define the stochastic semantics such that it specifies a continuous time Markov chain for every process. Gillespie's algorithm allows to execute this Markov chain, i.e. to compute simulation traces.

Our semantics copes with *instantaneous* reactions. Again, the problem is to appropriately define the Markov chain of a process, under the assumption that there are no immediate loops. Previous work on the stochastic π-calculus does either not define Markov chains explicitly (Priami et al., 2001; Phillips and Cardelli, 2004) so that correctness propositions can only be stated partially, or differently (Priami, 1995) which is less relevant to applications in systems biology. Experiments with the existing implementations – SPiM– confirm a correct treatments nevertheless.

Finally, we show how to *encode* the stochastic π-calculus with input patterns back into the original stochastic π-calculus without. We prove that our translation is *correct*, in that it preserves Markov chains. Hence, existing simulation machines can be used to execute models formulated in our π-calculus with input patterns.

Outline. We present our π-calculus with input patterns (Section 4.1), recall continuous time Markov chains and relate them to chemical reaction (Section 4.2). The stochastic semantics of our π-calculus extension is presented in Section 4.3. We then show how to encode input patterns (Section 4.4). Finally we discuss yet another extension, one by higher-order definitions, that allows to express classes (Section 4.5). Chapter 5 separately

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

Table 4.1: Syntax of Core SpiCO

discusses how to define different notions of concurrent objects.

4.1 The language

The core of SpiCO (Core SpiCO) consists in a novel stochastic π-calculus with *input patterns*, a linguistic feature introduced by Vasconcelos and Tokoro for typed concurrent objects in the asynchronous π-calculus (TyCO) (Paulino et al., 2003; Vasconcelos and Tokoro, 1993). Input patterns are motivated by pattern matching in functional programming languages of the ML family. In TyCO, they are closely tied to communication: objects only receive tuples if they provide a matching input pattern.

Core SpiCO's vocabulary consists in an infinite set of *channel names* $\mathscr{N} = \{x, y, z, \ldots\}$, a set of *process names* A, and a set of *function names* $f \in \mathscr{F}$. Process and function names have fixed arities. We write A/n or f/n for a symbol of arity $n \geq 0$. In order to account for *stochastic rates*, the vocabulary comprises functions $\rho : \mathscr{F} \rightarrow]0, \infty]$ to define *stochastic rates* for every channel. If some function ρ is assigned to x then $\rho(f)$ is the rate of the pair (x, f).

Table 4.1 defines the syntax of Core SpiCO. We write $\tilde{x}$ for finite, possibly empty sequences of channels $x_1, \ldots, x_n$ where $n \geq 0$. When using tuples $f(\tilde{x})$ or terms $A(\tilde{x})$ the number of arguments (the length of $\tilde{x}$) is assumed equal to the respective arity of f or A. Process expressions are ranged over by P. The only atomic expression (not decomposable into others) is the guarded choice of length $n = 0$ that we write as $\mathbf{0}$. Expressions $P_1|P_2$ denote the parallel composition of processes P_1 and P_2. A term **new** $x{:}\rho.\ P$ introduces a new channel x scoping 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

$$\begin{array}{rcl}
fv(x?f(\tilde{y}).P) & = & \{x\} \cup (fv(P) - \{\tilde{y}\}) \\
fv(P_1 \mid P_2) & = & fv(P_1) \cup fv(P_2) \\
fv(x!f(\tilde{y}).P) & = & \{x\} \cup fv(P) \cup \{\tilde{y}\} \\
fv(\mathbf{new}\ x{:}\rho.\ P) & = & fv(P) - \{x\} \\
fv(C_1 + \ldots + C_n) & = & fv(C_1) \cup \ldots \cup fv(C_n)
\end{array}$$

Table 4.2: Free variables

$f \in \mathscr{F}$. An expression $A(\tilde{x})$ applies the definition of a parametric process A with actual parameters $\tilde{x}$.

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

A definition of a parametric process has the form $A(\tilde{x}) \triangleq P$ where A is a process name with $\tilde{x}$ as formal parameters - that is, a sequence of bound channels. For modeling convenience, we permit free channel names in P besides the parameters in $\tilde{x}$. The set of *free channel names* for processes P and guarded processes C are denoted by $fv(P)$ and $fv(C)$ respectively. There are three scope baring constructs: new binder **new** $x{:}\rho.\ P$, input patterns $\_?f(\tilde{x}).P$, and definitions $A(\tilde{x}) \triangleq P$. Formally, these sets are defined by induction on the structure of such expressions in Table 4.2.

We define an (non-stochastic) operational semantics for the π-calculus in terms of a binary relation between expressions, the so called (one step) reduction. We will later refine it to a ternary relation adding stochastic labels. The reduction relation is closed under the usual structural congruence between expressions.

The *structural congruence* is the smallest relation induced by the axioms in Table 4.3. 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 sums, and scope extrusion. We also assume congruence of α-convertible processes, i.e. that can be obtained from another by renaming bound names, without capturing free names. See for instance (Honda, 1992) for a formal definition of α-conversion.

Table 4.4 defines the *reduction relation*. The first axiom tells how to interpret choices; it comprises channel communication and pattern matching.

$$\begin{array}{rcl}
(P_1|P_2)|P_3 & \equiv & P_1|(P_2|P_3) \\
P_1|P_2 & \equiv & P_2|P_1 \\
\ldots + C_1 + C_2 + \ldots & \equiv & \ldots + C_2 + C_1 + \ldots \\
P|\mathbf{0} & \equiv & P \\
P_1 & \equiv & P_2 \quad \text{if } P_1 \equiv_\alpha P_2 \\
\mathbf{new}\ x_1{:}\rho_1.\ \mathbf{new}\ x_2{:}\rho_2.\ P & \equiv & \mathbf{new}\ x_2{:}\rho_2.\ \mathbf{new}\ x_1{:}\rho_1.\ P \quad \text{if } x_1 \neq x_2 \\
\mathbf{new}\ x{:}\rho.\ (P_1|P_2) & \equiv & P_1|\ \mathbf{new}\ x{:}\rho.\ P_2 \quad \text{if } x \notin fv(P_1)
\end{array}$$

Table 4.3: Axioms of the structural congruence

Communication, choice, and pattern matching:

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

Application of definitions:

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

Context and congruence closure:

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

Table 4.4: Reduction relation for a finite set of definitions Δ

It applies to two complementary matching alternatives in parallel choices, an output alternative $x!f(\tilde{y}).P_1$ willing to send a term $f(\tilde{y})$ and an input pattern $x?f(\tilde{z}).P_2$ on the same channel x; this pattern matches in that it is built using the same function symbol f. Reduction cancels all other alternatives, substitutes the pattern's variables $\tilde{z}$ by the received channels $\tilde{y}$ in the continuation P_2 of the input, and reduces the result in parallel with the continuation of the output P_1.

Only matching tuples can be received over a channel. Other sending attempts suspend until a suitable input pattern becomes available. This fact proves extremely useful for concurrent modeling. Upon reception, tuples are immediately decomposed, in contrast to the π-calculus with data terms (Baldamus et al., 2005).

The application axiom unfolds one of the definitions of the parametric processes in a given set Δ. An application $A(\tilde{y})$ reduces in one step to definition P in which the formal parameters $\tilde{y}$ were replaced by the actual parameters $\tilde{x}$. Parametric definitions may be recursive, e.g. A may occur in P. Reduction can be applied in arbitrary contexts, however not under choices or in definitions.

Syntactic sugar for polyadic input and output. The syntax of the biochemical stochastic π-calculus is the same as ours except for function names, and our more flexible assignment of stochastic rates. We can express polyadic input and output by using dummy function names $\textsc{unit}_i$ for all arities $i \geq 0$ in following shortcuts for all sequences $\tilde{y}$ of channel names of length i:

$$x?(\tilde{y}).P \;=_{\text{def}}\; x?(\textsc{unit}_i(\tilde{y})).P \qquad \text{and} \qquad x!(\tilde{y}).P \;=_{\text{def}}\; x!(\textsc{unit}_i(\tilde{y})).P$$

Discussion on pattern matching. Input patterns as presented here are motivated by pattern matching in functional programming languages of the ML family. Pattern matching is the standard operation used there to decompose data structures containing value tuples.

In contrast, the input patterns presented here are closely tied to communication. A process can only receive tuples for which it provides a matching input pattern. Otherwise it refuses the communication offer, until it a change of state provides the required pattern.

Vasconcelos and Tokoro (1993) proposed input patterns for the asynchronous π-calculus [2]. They restrict sums to input patterns on the same channel $x?f_1(\tilde{y}_1) + \ldots + x?f_n(\tilde{y}_n)$. Outputs $x!f(\tilde{y}).\mathbf{0}$ are asynchronous there; they have no continuation and cannot appear in proper sums. In our modeling work we will need mixed inputs and outputs, both over possibly different channels, all with their continuations.

Compared to the π-calculus with data terms of Baldamus, Parrow, and Victor (2005), the π-calculus presented here cannot pass arbitrary nested tuples around. One can only pass flat tuples $f(\tilde{y})$, and needs an f headed pattern in order to receive them.

Example. Semaphores control the access to shared resources in concurrent systems (Dijkstra, 1971). They are widespread in programming languages, operating systems, or distributed databases. A semaphore restricts the access to a resource to single user at a time. We consider simple semaphores with two states - free or bound. When free, they can become bound, and when bound they can become free. Importantly, a bound semaphore may not be bound a second time. Any binding attempt on a bound semaphore is suspended until the semaphore becomes free.

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

[2]Note that mutually recursive definitions were absent in the original proposal of Vasconcelos and Tokoro (1993) but have been added on the way to the distributed programming language TyCO (Paulino et al., 2003).

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 bound semaphore s unfolds its definition. This creates an input offer on s, able to receive a free message. Other messages cannot be received on s in this state, namely no bind requests. Hence, the site s cannot get bound a second time. Only after the free message was received, s is able to accept the next bind request, while becoming bound again.

4.2 Markov chains for chemical reactions

The stochastic semantics of our π-calculus is guided by the analogy to continuous time Markov chains (CTMCs) for chemical reactions.

We first recall CTMCs with countably infinite state spaces. We assume a countable set S called the *state space*. A *continuous time stochastic process* with states $q \in S$ is a family $\{X_t \mid t \in \mathbb{R}^+\}$ of random variables with values in S. These define probabilities $Pr(X_t \in S')$ for all subsets $S' \subseteq S$, i.e. the probability that the process is in some state of S' at time t.
A *continuous time Markov chain (CTMC)* is a continuous time stochastic process (CTSP), with memoryless sojourn times for all states. More formally, a CTMC over S is a CTSP $\{X_t \mid t \in \mathbb{R}^+\}$ with states in S, that satisfies the Markov property, i.e. for all $q_0, \ldots, q_{n+1} \in S$ and all time points $0 \leq t_0 < \ldots < t_{n+1}$:

$$Pr(X_{t_{n+1}} = q_{n+1} \mid X_{t_n} = q_n, \ldots, X_{t_0} = q_0) = Pr(X_{t_{n+1}} = q_{n+1} \mid X_{t_n} = q_n)$$

The probabilistic behavior of a CTMC is determined by the distribution of its initial states (at time 0) and its *transition rates*. The transition rate r from state q to state q' is a value that "scales how the (one step) transition probability between q and q' increases with time" (Hermanns, 2002). We write $q \xrightarrow{r} q'$ in this case. For simplicity, we consider CTMCs with a single initial state. These can be identified with a Markovian transition system $(S, (\xrightarrow{r})_{r \in R^+}, q_0)$ where $q_0 \in S$ is the initial state and $\xrightarrow{r} \subseteq S \times S$ are transition relations for all $r \in \mathbb{R}^+$, such that for all $q, q' \in S$ there exists at most one $r \in R^+$ satisfying $q \xrightarrow{r} q'$.

The stochastic time evolution of a CTMC can be computed by Gillespie's *first reaction method* (1976) (Gillespie, 1976) if each state permits only a finite number of transitions, as we assume in the sequel. At time 0 the process starts in state q_0. Suppose that the process has moved to state q at time point t and let $q\{\xrightarrow{r_i} q_i\}_i$ be all (finitely many) transitions starting in q. Draw delays $t_i > 0$ for all i from an exponential distribution with rate r_i. Draw with equal probability some j, with minimal t_j. Move to state q_j at time point $t + t_j$.

Gillespie's direct method equivalently determines the stochastic behavior of a CTMC (Gillespie, 1976). In state q at time t it first computes the delay until the next transition (called *sojourn time*), by drawing a number from the exponential distribution with rate $\downarrow s =_{\text{def}} \sum_{q \xrightarrow{r_i} q_i} r_i$. Second, the state q_j to go to is drawn with probability $Pr(q \rightarrow q_j) =_{\text{def}} r_j / \sum_{q \xrightarrow{r'} q'} r'$ if $q \xrightarrow{r_j} q_j$ and 0 otherwise.

4.2.1 Chemical reactions

We next illustrate CTMCs for systems of chemical reaction rules. We start from a set of chemical species X, Y, Z and a set of chemical reaction rules of the following form, where $r \in \mathbb{R}^+$, reserving the symbol $+$ for choice:

$$X \mid Y \xrightarrow{r} Z_1 \mid \ldots \mid Z_k$$

Chemical solutions P are multisets of species, where each occurrence in the multiset represents a molecule of the species. Chemical rules as above apply as follows to a chemical solution P. Each pair of molecules of species X and Y can interact at rate r, yielding one molecule of each of the species $Z_1, \ldots, Z_k$. The solution obtained is $P - \{\!|\ X, Y\ |\!\} \cup \{\!|\ Z_1, \ldots, Z_k\ |\!\}$. According to the *Chemical Law of Mass Action*, the speed of a chemical reaction in a solution is proportional to the number of possible interactions of its reactants in the solution. It is distributed exponentially, and defines a CTMC with chemical solutions as states and the following transitions:

$$P \xrightarrow{n \cdot r} \begin{cases} P - \{\!|\ X, Y\ |\!\} \\ \cup \{\!|\ Z_1, \ldots, Z_k\ |\!\} \end{cases} \quad \text{where} \quad n = \begin{cases} \sharp(X \in P) \times \sharp(Y \in P) & \text{if } X \neq Y \\ \binom{\sharp(X \in P)}{2} & \text{else} \end{cases}$$

The expression $\binom{m}{2} = \frac{1}{2}\, m\, (m-1)$ counts the number of two-element subsets in sets of cardinality m.

4.3 Stochastic semantics of core SpiCO

We define the stochastic semantics of Core SpiCO by associating a π-calculus process with a CTMC. The states of this Markov chain are the (countably infinite) set of congruence classes of π-calculus processes with respect to structural congruence. This differs from (Priami, 1995) where two congruent processes are associated with two different states. Since congruent processes are behaviorally equivalent we believe that their associated stochastic states should not be distinguished neither. Moreover, in (Priami, 1995), the author proposes a *labeled semantics* where labels are so-called *proof terms*, i.e. (possibly long) strings used to localize interacting sub-terms. Those labels are necessary to properly calculate interaction rates. We instead propose a *reduction semantics*, a style for defining semantics known to be more intuitive and elegant. Still, we temporarily use labels but in a much simpler form: a label is an integer or a tuple of four integers. Finally, and contrary to (Priami, 1995), our semantics takes into account immediate transitions of which we emphasized the importance in the biological example in section 6.3. Such transitions require specific consideration: we show how they can be removed in order to obtain an equivalent Markovian transition system. The theorem 1 states the correctness of this transformation.

4.3.1 Transition relations

We first consider the *fragment* of the π-calculus without proper summation, parametric processes, infinite rates, and **new**-binders. The remaining processes are parallel compositions $C_1 \mid \dots \mid C_n$. The structural congruence turns them into multisets of guarded processes, i.e. into chemical solutions whose species are guarded processes.

Suppose we know the rate functions $\varrho(x)$ for all channels x. The π-calculus with input patterns then defines the following chemical reaction rule:

$$x?f(\tilde{z}).Q_1 \mid x!f(\tilde{y}).Q_2 \xrightarrow{\varrho(x)(f)} Q_1[\tilde{z} \mapsto \tilde{y}] \mid Q_2$$

This defines a CTMC. For example, assume n molecules of a first species $x!f().P_1$ and m of another different one $x!f().P_2$, which all want to react with a single molecule of a third kind $x?f().P$. The Markovian transitions are:

$$\prod_{i=1}^{n} x!f().P_1 \mid \prod_{i=1}^{m} x!f().P_2 \mid x?f().P \left\{ \begin{array}{ll} \xrightarrow{n\times\varrho(x)(f)} & \prod_{i=1}^{n-1} x!f().P_1 \mid \prod_{i=1}^{m} x!f().P_2 \mid P \\ \xrightarrow{m\times\varrho(x)(f)} & \prod_{i=1}^{n} x!f().P_1 \mid \prod_{i=1}^{m-1} x!f().P_2 \mid P \end{array} \right.$$

Labeled reduction steps

$$(\text{COM})\quad \frac{C_{i_1}^{j_1} = x?f(\tilde z).\mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ Q_1 \qquad C_{i_2}^{j_2} = x!f(\tilde y).\mathbf{new}\ \widetilde{x_2{:}\rho_2}.\ Q_2}{\Pi_{i=1}^{n} \sum_{j=1}^{m_i} C_i^j \xrightarrow[i_1,j_1,i_2,j_2]{\varrho(x)(f)} \left\{ \begin{array}{l} \mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ \mathbf{new}\ \widetilde{x_2{:}\rho_2}. \\ (Q_1[\tilde z \mapsto \tilde y] \mid Q_2 \mid \Pi_{i=1,i\neq i_1,i_2}^{n} \sum_{j=1}^{m_i} C_i^j) \end{array}\right.}$$

where Q_1, Q_2 have no top-level **new**-binders and
$1 \leq i_1 \neq i_2 \leq n,\ 1 \leq j_1 \leq m_{i_1},\ 1 \leq j_2 \leq m_{i_2}$

$$(\text{APP})\quad \frac{P_{i_1} = A(\tilde y) \qquad A(\tilde x) \triangleq \mathbf{new}\ \widetilde{z{:}\rho}.\ Q \text{ in } \Delta}{\Pi_{i=1}^{n} P_i \xrightarrow[i_1]{\infty} \mathbf{new}\ \widetilde{z{:}\rho}.\ (Q[\tilde x \mapsto \tilde y] \mid \Pi_{i=1,i\neq i_1}^{n} P_i)}$$

where Q has no top-level **new**-binders and $1 \leq i_1 \leq n$

$$(\text{NEW})\quad \frac{P \xrightarrow[w]{s} Q \qquad \varrho(x) = \rho}{\mathbf{new}\ x{:}\rho.\ P \xrightarrow[w]{s} \mathbf{new}\ x{:}\rho.\ Q} \quad \text{where } s \in \mathbb{R}^+ \cup \{\infty\},\ w \in \mathbb{N} \cup \mathbb{N}^4$$

Time consuming transitions $(r, r' \in \mathbb{R}^+,\ w \in \mathbb{N}^4)$

$$(\text{SUM})\quad \frac{P \equiv P' \qquad r = \sum_{P' \xrightarrow[w]{r'} Q' \equiv Q} r' \neq 0 \qquad \neg\exists R \exists w' \in \mathbb{N} \cup \mathbb{N}^4.\ P' \xrightarrow[w']{\infty} R}{P \xrightarrow{r} Q}$$

Immediate transitions

$$(\text{COUNT})\quad \frac{P \equiv P' \qquad \begin{array}{l} n = \sharp\{w \in \mathbb{N} \cup \mathbb{N}^4 \mid P' \xrightarrow[w]{\infty} Q' \equiv Q\} \neq 0 \\ m = \sharp\{w \in \mathbb{N} \cup \mathbb{N}^4 \mid P' \xrightarrow[w]{\infty} Q''\} \end{array}}{P \xrightarrow{\infty(n/m)} Q}$$

Table 4.5: Timed transitions of Core SpiCO with respect to a set Δ of definitions in prenex normal form, and a global assignment ϱ of channels to rate functions.

We first discuss time consuming transitions $P \xrightarrow{r} P'$ where $r \in \mathbb{R}^+$. These capture everything, except parametric process unfolding and invocation of functions of rate ∞.

We first define labeled reduction steps $P \xrightarrow[w]{s} Q$ where P and Q are in prenex normal form, that is a parallel composition of sums where restrictions have been pushed ahead and in which bound variables are renamed apart. The rate function $\varrho(x)$ is then read off from the quantifier prefix in rule

(NEW).
Definition. P is in prenex normal form (pnf for short) iff $P = \mathbf{new}\ \widetilde{x{:}\rho}.\ (P_1 \mid \ldots \mid P_m)$ where each P_i either is an application $A(\tilde{y})$, or a sum $C_1 + \ldots + C_n$ where each C_j is in pnf, or a guarded process $x?f(\tilde{y}).Q$ or $x!f(\tilde{y}).Q$ where Q is in pnf. Moreover, a definition $A(\tilde{y}) \triangleq P$ is in pnf iff P is in pnf.

What remains from pnfs after removing top-level **new**-binders are multisets of sums and applications. All applications must have been reduced before time consuming transitions can apply, so we have a multiset of sums. Each sum is like a molecule, except that each of its choices offers its own interactions.

In $x?f().\mathbf{0}+x?f().\mathbf{0} \mid x!f().\mathbf{0}$ there are two possible interactions with rate $r = \varrho(x)(f)$ leading to the same state. We can think of $x?f().\mathbf{0} + x?f().\mathbf{0}$ as a protein with two identical domains, complementary to one domain of some other protein represented by $x!f().\mathbf{0}$. The overall rate of the interaction thus doubles:

$$\begin{array}{rl} & \boxed{\mathrm{x?f()}}.\mathbf{0} + x?f().\mathbf{0} \mid \boxed{\mathrm{x!f()}}.\mathbf{0} \xrightarrow[1.1.2.1]{r} \mathbf{0} \\ \text{and} & x?f().\mathbf{0} + \boxed{\mathrm{x?f()}}.\mathbf{0} \mid \boxed{\mathrm{x!f()}}.\mathbf{0} \xrightarrow[1.2.2.1]{r} \mathbf{0} \\ \text{sums up to} & x?f().\mathbf{0} + x?f().\mathbf{0} \mid x!f().\mathbf{0} \xrightarrow{2r} \mathbf{0} \end{array}$$

Rule (COM) defines labeled reductions $P \xrightarrow[i_1,j_1,i_2,j_2]{r} Q$ that distinguish communication actions with identical reactants and results, while using different occurrences of choice alternatives in sums. Those occurrences are identified by *labels* in $\mathbb{N}^4$ that specify the numbers of the reacting sums (i_1, i_2) and the reacting choices (j_1, j_2). Rule (SUM) defines transitions $P \xrightarrow{r} Q$ by summing up all rates of all different interactions leading from P to Q. These reduction rules are defined with care, so that corresponding interactions in structurally congruent processes are not counted twice.

We next turn to *immediate transitions* $P \xrightarrow{\infty(p)} Q$, where $p \in [0, 1]$ is a probability. Rule (SUM) ensures that time consuming transitions apply only after all immediate have been reduced. In this case, all calls $A(\tilde{y})$ on top level must have been reduced before. Note that this order is important for a proper count of the possible interactions. Indeed, if an application hides an interaction on some pattern, the application unfolding changes the rate of the action involving this pattern. Immediate transitions can be licensed by communication (COM), or by applications of parametric process definitions (APP). Their labels are in $\mathbb{N} \cup \mathbb{N}^4$. Note that the labeled reduction is independent of the choice of the pnf.

Lemma 1 *Let P and P' be in prenex normal form and $s \in \mathbb{R}^+ \cup \{\infty\}$. If $P \equiv P'$ and $P \xrightarrow[w]{s} Q$, then there exist $Q' \equiv Q$ and w' such that $P' \xrightarrow[w']{s} Q'$.*

Proof. Straightforward.

We merge labeled immediate transitions with rule (COUNT). Although being immediate we want to associate probabilities, which characterize the number of immediate interactions leading to a common state with respect to the total number of enabled immediate interactions. For instance, let $\varrho(x)(f) = \infty$, in $x?f().P + x?f().P \mid x?f().Q \mid x!f().\mathbf{0}$, for some $P \not\equiv Q$, the associated probabilities reflect that 2 out of 3 interactions lead to P, and 1 out of 3 to Q:

$$x?f().P + x?f().P \mid x?f().Q \mid x!f().\mathbf{0} \xrightarrow{\infty(2/3)} P$$
$$x?f().P + x?f().P \mid x?f().Q \mid x!f().\mathbf{0} \xrightarrow{\infty(1/3)} Q$$

Lemma 2 *The following properties hold:*

1. $\equiv \circ \xrightarrow{r} \circ \equiv \subseteq \xrightarrow{r}$,
2. $\equiv \circ \xrightarrow{\infty(p)} \circ \equiv \subseteq \xrightarrow{\infty(p)}$,
3. *if* $P \xrightarrow{r} Q$ *and* $P \xrightarrow{r'} Q$ *then* $r = r'$,
4. *if* $P \xrightarrow{\infty(p_1)} Q$ *and* $P \xrightarrow{\infty(p_2)} Q$ *then* $p_1 = p_2$, *and*
5. *for all* P: $\sum_{P \xrightarrow{\infty(p)} Q} p = 1$.

Proof.

1. Let $P \equiv \circ \xrightarrow{r} \circ \equiv Q$, then there are P' and Q' such that $P \equiv P'$, $P' \xrightarrow{r} Q'$ and $Q' \equiv Q$. Since $P' \xrightarrow{r} Q'$ is necessarily inferred by (SUM), there is $P'' \equiv P'$ (1) such that $r = \sum_{P'' \xrightarrow[w]{r'} Q'' \equiv Q'} r' \neq 0$ (2), and $\neg\exists R \exists w' \in \mathbb{N} \cup \mathbb{N}^4 . P'' \xrightarrow[w']{\infty} R$ (3). From $P' \equiv P$ and (1), we have $P \equiv P''$ (1'), and by $Q' \equiv Q$ and (2) we have $r = \sum_{P'' \xrightarrow[w]{r'} Q'' \equiv Q} r' \neq 0$ (2'). Then, by (1'), (2'), (3) and by (SUM), we conclude $P \xrightarrow{r} Q$.
2. Similar to the previous point, by rule (COUNT) and Lemma 1.

3. Suppose that $P \xrightarrow{r} Q$ and $P \xrightarrow{r'} Q$, then by rule (SUM), there are P_1 and P_2 such that $P_1 \equiv P \equiv P_2$ and $r = \sum_{P_1 \xrightarrow[w_1]{r_1} Q_1 \equiv Q} r_1$ and $r' = \sum_{P_2 \xrightarrow[w_2]{r_2} Q_2 \equiv Q} r_2$. However, by Lemma 1, there is $Q_1 \equiv Q$ such that $P_1 \xrightarrow[w_1]{r_1} Q_1$ iff there is $Q_2 \equiv Q$ such that $P_2 \xrightarrow[w_2]{r_2} Q_2$ and $r_1 = r_2$. Therefore, $r = r'$.

4. Similar to the previous point, using Lemma 1.

5. Assume, without loss of generality, that P is in prenex normal form. Let $\{Q_1, \ldots, Q_k\}$ be the set (up-to $\equiv$) of all the possible immediate and distinct derivatives of P, that is satisfying $\forall i \in \{1, \ldots, k\}\ P \xrightarrow{\infty(p_i)} Q_i$ and $i \neq j \Rightarrow Q_i \not\equiv Q_j$. Let $X = \{w \in \mathbb{N} \cup \mathbb{N}^4 \mid P \xrightarrow[w]{\infty} Q''\}$ and $Y_i = \{w \in \mathbb{N} \cup \mathbb{N}^4 \mid P \xrightarrow[w]{\infty} Q' \equiv Q_i\}$. It is clear that, $i \neq j \Rightarrow Y_i \cap Y_j = \emptyset$ (since $Q_i \not\equiv Q_j$) and $\bigcup_{1 \leq i \leq k} Y_i = X$. Therefore $\sum_{1 \leq i \leq k} \sharp Y_i = \sharp X$, and, since $p_i = \sharp Y_i / \sharp X$, $\sum_{1 \leq i \leq k} p_i = 1$. $\square$

4.3.2 CTMCs with immediate reactions

In the presence of immediate transitions, the reduction relation $\xrightarrow{r}$ does not define a Markovian transition system (in which all rates are finite). To capture the stochastic dynamics of processes, we instead define the *sojourn time parameters* (i.e. the parameter of an exponentially distributed probability which determine the sojourn time in a given state) and the *probabilities of state changes* for all P, Q as follows[3]:

$$\begin{aligned}
\downarrow P &= \begin{cases} \infty & \text{if } P \xrightarrow{\infty(p)} Q, \\ \sum_{P \xrightarrow{r} Q} r & \text{otherwise.} \end{cases} \\
Pr(P \to Q) &= \begin{cases} r / \sum_{P \xrightarrow{r'} Q'} r' & \text{if } P \xrightarrow{r} Q \\ p & \text{if } P \xrightarrow{\infty(p)} Q \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

We are now giving an interpretation of the reduction semantics with immediate transitions in terms of CTMCs for processes that can not exhibit infinite sequences of immediate transitions. The Markovian transition system deriving statements $P \overset{r}{\Longrightarrow} Q$ is defined in Table 4.6. The idea is quite similar

[3]we assume if X is exponentially distributed with parameter ∞ then $Pr(X = 0) = 1$.

$$(\textsc{elim}_1)\quad \frac{P \xrightarrow[w]{\infty} Q \qquad n = \sharp\{w' \in \mathbb{N} \cup \mathbb{N}^4 \mid P \xrightarrow[w']{\infty} Q'\}}{P \xrightarrow[w]{\infty(1/n)} Q} \quad w \in \mathbb{N} \cup \mathbb{N}^4$$

$$(\textsc{elim}_2)\quad \frac{P \xrightarrow[w]{r} Q \qquad Q \xrightarrow[w_1]{\infty(p_1)} \dots \xrightarrow[w_n]{\infty(p_n)} Q_n \not\xrightarrow{\infty}}{P \overset{rp_1\dots p_n}{\underset{ww_1\dots w_n}{\Longrightarrow}} Q_n} \quad r \in \mathbb{R}^+$$

$$(\textsc{elim}^{\text{sum}})\quad \frac{P \equiv P' \qquad r = \sum_{P' \overset{r'}{\underset{w_1\dots w_n}{\Longrightarrow}} Q' \equiv Q} r'}{P \overset{r}{\Longrightarrow} Q}$$

Table 4.6: Elimination of immediate transitions and merging timed transitions

to that of (Bernardo et al., 1994): the transitions are obtained by integrating immediate transitions into time consuming transitions. An example for this transformation is as follows:

$$P \begin{cases} \xrightarrow{r_1} & Q_1 \not\xrightarrow{\infty} \\ \xrightarrow{r_2} & Q_2 \begin{cases} \xrightarrow{\infty(p)} & Q_{21} \not\xrightarrow{\infty} \\ \xrightarrow{\infty(1-p)} & Q_{22} \not\xrightarrow{\infty} \end{cases} \end{cases} \quad \text{becomes} \quad P \begin{cases} \overset{r_1}{\Longrightarrow} & Q_1 \\ \overset{r_2 p}{\Longrightarrow} & Q_{21} \\ \overset{r_2(1-p)}{\Longrightarrow} & Q_{22} \end{cases}$$

In general, a sequence of reductions $P \xrightarrow{r} P_1 \xrightarrow{\infty(p_1)} \dots P_n \xrightarrow{\infty(p_n)} Q \not\xrightarrow{\infty}$ reduces to $P \overset{rp_1\dots p_n}{\Longrightarrow} Q$. However, we must beware of merging initially distinct states. Indeed, in the previous example, if $Q_{22} \equiv Q_1$ then the CTMC should have transitions $P \overset{r_1+r_2(1-p)}{\Longrightarrow} Q_1$ and $P \overset{r_2 p}{\Longrightarrow} Q_{21}$. In order to infer these transitions correctly, the elimination procedure defines labeled transitions $\overset{r}{\underset{w}{\Longrightarrow}}$ with *labels* $w \in (\mathbb{N} \cup \mathbb{N}^4)^\star$ representing *paths* in the labeled derivation trees of $\xrightarrow{r}$.

Lemma 3 *Let P and P' be in prenex normal form. If $P \equiv P'$ and $P \overset{r}{\underset{w}{\Longrightarrow}} Q$ (resp. $P \overset{\infty(p)}{\underset{w}{\Longrightarrow}} Q$), then there exists $Q' \equiv Q$ and w' such that $P' \overset{r}{\underset{w'}{\Longrightarrow}} Q'$ (resp. $P' \overset{\infty(p)}{\underset{w'}{\Longrightarrow}} Q'$).*

Proof.

1. Suppose that $P \overset{r}{\underset{w}{\Longrightarrow}} Q$, by rule $(\textsc{elim}_2)$, we have $P \xrightarrow[w_0]{r_0} Q_0$ and $Q_0 \xrightarrow[w_1]{\infty(p_1)} \cdots \xrightarrow[w_n]{\infty(p_n)} Q_n = Q \not\xrightarrow{\infty}$ where $r = r_0 p_1 \dots p_n$ and $w =$

$w_0 w_1 \dots w_n$. By Lemma 1, there exists $Q'_0 \equiv Q_0$ such that $P' \xrightarrow[w'_0]{r_0} Q'_0$. Q'_0 is necessarily in prenex normal form. Thus by Lemma 1, there are $Q'_1 \equiv Q_1, \dots Q'_n = Q' \equiv Q$ all in prenex normal form such that $Q'_0 \xrightarrow[w'_1]{\infty(p_1)} \cdots \xrightarrow[w'_n]{\infty(p_n)} Q'_n \overset{\infty}{\not\rightarrow}$. Therefore, by rule (ELIM$_2$), $P' \overset{r}{\underset{w'}{\Longrightarrow}} Q' \equiv Q$ where $w' = w'_0 w'_1 \dots w'_n$.

2. Suppose that $P \xrightarrow[w]{\infty(p)} Q$, by rule (ELIM$_1$), we have $P \xrightarrow[w]{\infty} Q$ and $p = 1/\sharp\{w_0 \in \mathbb{N} \cup \mathbb{N}^4 \mid P \xrightarrow[w_0]{\infty} R\}$. By Lemma 1, there exist w' and $Q' \equiv Q$ such that $P' \xrightarrow[w']{\infty} Q'$ and $\sharp\{w_0 \in \mathbb{N} \cup \mathbb{N}^4 \mid P \xrightarrow[w_0]{\infty} R\} = \sharp\{w'_0 \in \mathbb{N} \cup \mathbb{N}^4 \mid P' \xrightarrow[w'_0]{\infty} R'\}$. Therefore, by (ELIM$_1$), we have $P' \xrightarrow[w']{\infty(p)} Q'$. □

Lemma 4 $\equiv \circ \overset{r}{\Longrightarrow} \circ \equiv \subseteq \overset{r}{\Longrightarrow}$ *and if* $P \overset{r}{\Longrightarrow} Q$ *and* $P \overset{r'}{\Longrightarrow} Q$ *then* $r = r'$.

Proof. Similar to the proof of Lemma 2 using Lemma 3.

For any P such that $P \overset{\infty}{\not\rightarrow}$, $(\mathscr{P}_{/\equiv}, (\overset{r}{\Longrightarrow})_{r \in \mathbb{R}^+}, P_{/\equiv})$ is a Markovian transition system[4] with sojourn time parameters and transition probabilities:

$$\Downarrow P = \sum_{P \overset{r}{\Longrightarrow} Q} r \qquad \text{and} \qquad Pr(P \Rightarrow Q) = \begin{cases} r / \sum_{P \overset{r'}{\Longrightarrow} Q'} r' & \text{if } P \overset{r}{\Longrightarrow} Q \\ 0 & \text{otherwise} \end{cases}$$

In order to show that this defines a Markovian model for the reduction semantics with immediate transitions, we show that their dynamics coincide, that is: the sojourn time parameters and the transition probabilities with respect to $\xrightarrow{r}$ are identical to those of $\overset{r}{\Longrightarrow}$. However, transition probabilities can be compared only for processes performing timed transitions. We thus define a suitable transition probability $Pr(P \twoheadrightarrow Q)$ for $P \overset{\infty}{\not\rightarrow}$ and $Q \overset{\infty}{\not\rightarrow}$, that is the probability to reach Q from P by a sequence of transitions made of one timed transition and possibly several intermediate immediate transitions. Formally, $Pr(P \twoheadrightarrow Q)$ is the sum of the probabilities of all such sequences:

$$Pr(P \twoheadrightarrow Q) = \sum_{P \xrightarrow{r} Q_1 \xrightarrow{\infty(p_1)} \dots Q_n \xrightarrow{\infty(p_n)} Q \overset{\infty}{\not\rightarrow}} \left(Pr(P \rightarrow Q_1) \times \prod_{i=1}^{n} p_i \right)$$

[4]For $P \xrightarrow{\infty}$ it suffices to start with a process $Q = x!f().\mathbf{0} \mid x?f().P$ such that $\varrho(x)(f) = 1$ in order to obtain a set of initial processes together with an initial probability distribution of those processes rather than a single initial process.

Theorem 1 *If $P \stackrel{\infty}{\not\to}$ and if no infinite sequence of immediate transitions is reachable from P, then*

- *(Timed correctness)* $\downarrow P \; = \Downarrow P$*,*
- *(Probabilistic correctness)* $Pr(P \twoheadrightarrow Q) = Pr(P \Rightarrow Q)$*.*

Proof of Theorem 1

- *Timed correctness.* First, note that for any P we have $\sum_{P \xrightarrow[w]{\infty(p)} Q} p = 1$. In order to show that $\downarrow P \; = \Downarrow P$, it is sufficient, by rules (SUM) and (ELIM$^{\text{sum}}$) to show that

$$\sum_{P \xrightarrow[w]{r} Q} r \quad = \quad \sum_{P \xRightarrow[w]{r} Q} r \tag{4.3.1}$$

For any $n \geq 1$, we define the relation $P(\xrightarrow[w]{\infty(p)})^n Q$ meaning the longest derivation of immediate actions leading from P to Q and which length is smaller or equal to n. More formally, $P(\xrightarrow[w]{\infty(p)})^n Q$ iff

  - either if there exists a (unique) sequence $P \xrightarrow[w_1]{\infty(p_1)} Q_1 \ldots \xrightarrow[w_n]{\infty(p_n)} Q_n = Q$ and such that $p = p_i \times \ldots \times p_n$ and $w = w_1 \ldots w_n$, or
  - or if $n > 1$ and $P(\xrightarrow[w]{\infty(p)})^{(n-1)} Q$.

Given P, let us define n_P as the length of the longest derivation of immediate actions leading from P to some process Q. By the hypothesis of none infinite sequences of immediate actions, such an integer always exists and if $P(\xrightarrow[w]{\infty(p)})^{n_P} Q$ then $Q \stackrel{\infty}{\not\to}$. It is clear that $P \xRightarrow[w]{r} Q$ iff either $P \xrightarrow[w]{r} Q \stackrel{\infty}{\not\to}$ or, $P \xrightarrow[w_0]{r'} R(\xrightarrow[w']{\infty(p)})^{n_R} Q$ for some R and where $r = pr'$ and $w = w_0 w'$. Moreover, one can easily show (by induction on n) that for any P and n, $\sum_{P(\xrightarrow[w]{\infty(p)})^n Q} p = 1$. Therefore, given P, we have

$$\begin{aligned}
\sum_{P \overset{r}{\underset{w}{\Longrightarrow}} Q} r &= \sum_{P \xrightarrow[w_0]{r'} R} \Big(r \times \big(\sum_{R(\xrightarrow[w']{\infty(p)})^n R Q} p\big)\Big) \\
&= \sum_{P \xrightarrow[w_0]{r'} R} \big(r \times 1\big) \\
&= \sum_{P \xrightarrow[w]{r} Q} r
\end{aligned}$$

which proves (4.3.1).

- *Probabilistic correctness.* By timed correctness, we have $\sum_{P \overset{r'}{\Longrightarrow} Q} r' = \sum_{P \xrightarrow{r'} Q} r'$ (1). Moreover, one can easily show that $P \xrightarrow{\infty(p)} Q$, iff $\sum_{P \xrightarrow[w]{\infty(p')} R \equiv Q} p' = p$ (2). Given P, Q and length derivation n, we have:

$$\begin{aligned}
&Pr(P \xrightarrow{r} Q_1 \xrightarrow{\infty(p_1)} \dots Q_n \xrightarrow{\infty(p_n)} Q_{n+1} \equiv Q \not\xrightarrow{\infty}) \\
&\quad = \big(\textstyle\sum_{P \xrightarrow{r'} Q'} r'\big)^{-1}\big(r \times \prod_{1\le i\le n} p_i\big) \\
&\quad = \big(\textstyle\sum_{P \overset{r'}{\Longrightarrow} Q'} r'\big)^{-1}\big(r \times \prod_{1\le i\le n} p_i\big) && \text{by (1)} \\
&\quad = \big(\textstyle\sum_{P \overset{r'}{\Longrightarrow} Q'} r'\big)^{-1}\Big(\big(\textstyle\sum_{P \xrightarrow[w]{r'} Q' \equiv Q_1} r'\big) \times \prod_{1\le i\le n}\big(\textstyle\sum_{Q_i \xrightarrow[w]{\infty(p'_i)} Q' \equiv Q_{i+1}} p'_i\big)\Big) && \text{by (2)} \\
&\quad = \big(\textstyle\sum_{P \overset{r'}{\Longrightarrow} Q'} r'\big)^{-1}\big(\textstyle\sum_{P \xrightarrow[w_0]{r_0} Q'_1 \xrightarrow[w_1]{\infty(p'_1)} \dots \xrightarrow[w_n]{\infty(p'_n)} Q'_{n+1} \equiv Q} (r_0 \times \prod_{1\le i\le n} p'_i)\big)
\end{aligned}$$

Therefore, summing over all length derivations n, we conclude that $Pr(P \twoheadrightarrow Q) = Pr(P \Rightarrow Q)$. $\square$

4.4 Encoding input patterns

We now encode SpiCO back into the stochastic π-calculus. The latter can be identified as the special case with a unique function name per arity (we assume arities bounded by some max): $\mathscr{F}' = \{\textsc{unit}_i \mid 0 \le i \le max\}$. In what follows, we write $\textsc{unit}$ instead of $\textsc{unit}_i$.

$$
\begin{array}{l}
[\![\mathbf{new}\ x{:}\rho.\ P]\!] \ =_{\text{def}}\ \mathbf{new}\ x_{f_1}{:}\rho(f_1).\ \cdots\ .\mathbf{new}\ x_{f_n}{:}\rho(f_n).\ [\![P]\!] \\
[\![P_1 \mid P_2]\!] \ =_{\text{def}}\ [\![P_1]\!] \mid [\![P_2]\!] \\
[\![A(\tilde{y})]\!] \ =_{\text{def}}\ A(\tilde{y}_{\mathscr{F}}) \\
[\![C_1 + \cdots + C_n]\!] \ =_{\text{def}}\ [\![C_1]\!] + \cdots + [\![C_n]\!] \\
[\![A(\tilde{x}) \triangleq P]\!] \ =_{\text{def}}\ A(\tilde{x}_{\mathscr{F}}) \triangleq [\![P]\!] \\
[\![x?f(\tilde{y}).P]\!] \ =_{\text{def}}\ x_f?(\tilde{y}_{\mathscr{F}}).[\![P]\!] \\
[\![x!f(\tilde{y}).P]\!] \ =_{\text{def}}\ x_f!(\tilde{y}_{\mathscr{F}}).[\![P]\!]
\end{array}
$$

Table 4.7: Encoding of input patterns

4.4.1 Translation

We assume a total ordering $<$ on a finite set of function names $\mathscr{F}$. This means that $\mathscr{F}$ has a unique representation $\mathscr{F} = \{f_1, \dots, f_n\}$ with $f_1 < \dots < f_n$. Our encoding uses channel names from the set $\mathscr{N} \times \mathscr{F}$. We denote elements (x, f) of this set by x_f. For each channel x we define a sequence of n channels $x_{\mathscr{F}}$ as follows: $x_{\mathscr{F}} =_{\text{def}} x_{f_1}, \dots, x_{f_n}$. Channels in the target language are associated a rate (that may be infinite) by means of the encoding of ϱ defined as $[\![\varrho]\!](x_f) = \varrho(x)(f)$. We write $\tilde{x}, \tilde{y}$ for the concatenation of two sequences $\tilde{x}$ and $\tilde{y}$. If $\tilde{x} = x_1, \dots, x_n$ then we let $\tilde{x}_{\mathscr{F}} =_{\text{def}} x_{1\mathscr{F}}, \dots, x_{n\mathscr{F}}$. The encoding is given in Table 4.7.

Lemma 5 *For all processes P, P' with functions in $\mathscr{F}$ and variable sequences $\tilde{y}, \tilde{z}$ of the same length:*

1. *$\tilde{y}$ is free for $\tilde{z}$ in P if and only if $\tilde{y}_{\mathscr{F}}$ is free for $\tilde{z}_{\mathscr{F}}$ in $[\![P]\!]$.*
2. *$[\![P[\tilde{y} \mapsto \tilde{z}]]\!] = [\![P]\!]\ [\tilde{y}_{\mathscr{F}} \mapsto \tilde{z}_{\mathscr{F}}]$*
3. *If $[\![P]\!] \equiv Q$ then there exists $P' \in [\![Q]\!]^{-1}$ such that $P \equiv P'$.*
4. *$P \equiv P'$ if and only if $[\![P]\!] \equiv [\![P']\!]$.*
5. *P is in prenex normal form iff $[\![P]\!]$ is.*

Proof.

1. By induction on the definition of freeness conditions.
2. By induction on the structure of P. Note that the lemma may fail for processes P with function symbols outside $\mathscr{F}$. As a counter example let $\mathscr{F} = \emptyset$ and consider $x?f().\mathbf{0}[x \mapsto y]$.
3. By induction on the structure of P.

4. By induction on derivations of $P \equiv P'$ resp. $[\![P]\!] \equiv [\![P']\!]$.
5. By induction on the structure of P.

The following theorem states the correctness of our encoding. It allows us to run simulations of models expressed in SpiCO, via an implementation of the original stochastic π-calculus, as implemented in the SPiM system (Phillips and Cardelli, 2004).

Theorem 2 *The encoding defines a stochastic bisimulation: for all processes P, Q and finite sets of definitions Δ, and all rates $s \in \mathbb{R}^+ \cup \{\infty(p) \mid p \in]0,1]\}$ it holds that $P \xrightarrow{s} Q$ relative to Δ if and only if $[\![P]\!] \xrightarrow{s} [\![Q]\!]$ relative to $[\![\Delta]\!]$.*

The statement $P \xrightarrow{s} Q$ relative to Δ means that there exists some function $\varrho : \mathscr{N} \rightarrow \mathscr{F} \rightarrow (\mathbb{R}^+ \cup \{\infty\})$ such that $P \xrightarrow{s} Q$ relative to Δ and ϱ. The values $\varrho(x)$ will be the rate ρ assigned to x in the declaration **new** $x{:}\rho$. It holds for all ρ and x that $\varrho(x) = \rho$ iff $[\![\varrho]\!](x_f) = \rho(f)$ for all $f \in \mathscr{F}$.

The statement $[\![P]\!] \xrightarrow{s} [\![Q]\!]$ relative to $[\![\Delta]\!]$ means that there exists some function $\varrho' : \{x_f \mid f \in \mathscr{F}, x \in \mathscr{N}\} \rightarrow (\mathbb{R}^+ \cup \{\infty\})$ such that $[\![P]\!] \xrightarrow{s} [\![Q]\!]$ relative to $[\![\Delta]\!]$ and ϱ'. The situation differs in that there exists only a single function UNIT for all arities. We are a little sloppy in identifying a constant function with its constant value, i.e. $\varrho'(x_f) = \varrho'(x_f)(\text{UNIT})$.

4.4.2 Correctness proof

We prove a slightly stronger proposition than Theorem 2. We define a translation of functions $\varrho : \mathscr{N} \rightarrow \mathscr{F} \rightarrow (\mathbb{R}^+ \cup \{\infty\})$ to functions $[\![\varrho]\!] : \{x_f \mid f \in \mathscr{F}, x \in \mathscr{N}\}$ such that for all $x \in \mathscr{N}$ and $f \in \mathscr{F}$:

$$[\![\varrho]\!](x_f) =_{\text{def}} \varrho(x)(f)$$

The translation is onto, i.e for all $\varrho' : \{x_f \mid f \in \mathscr{F}, x \in \mathscr{N}\} \rightarrow (\mathbb{R}^+ \cup \{\infty\})$ there exists some $\varrho : \mathscr{N} \rightarrow \mathscr{F} \rightarrow (\mathbb{R}^+ \cup \{\infty\})$ such that $\varrho' = [\![\varrho]\!]$. Hence, the theorem follows from the following proposition:

Proposition 1 *$P \xrightarrow{s} Q$ with respect to Δ and ϱ iff $[\![P]\!] \xrightarrow{s} [\![Q]\!]$ with respect to $[\![\Delta]\!]$ and $[\![\varrho]\!]$.*

We need some auxiliary lemmas, before we can show the proposition for infinite rates in Lemma 9 and for finite rates in Lemma 10. All these Lemmas hold for all processes P, Q, Q' and definitions Δ with functions in $\mathscr{F}$, rates

$s \in \mathbb{R}^+ \cup \{\infty\}$ or $r \in \mathbb{R}^+$, functions ϱ, labels $w \in \mathbb{N} \cup \mathbb{N}^4$, and probabilities $p \in]0,1]$. For convenience, we define

$$\begin{array}{l} [\![\mathbf{new}\ y_1{:}\rho_1.\ \cdots\ .\ \mathbf{new}\ y_n{:}\rho_n]\!] =_{\text{def}} [\![\mathbf{new}\ y_1{:}\rho_1]\!].\ \cdots\ .\ [\![\mathbf{new}\ y_n{:}\rho_n]\!] \\ [\![\mathbf{new}\ y{:}\rho]\!] =_{\text{def}} \mathbf{new}\ y_{f_1}{:}\rho(f_1).\ \cdots\ .\ \mathbf{new}\ y_{f_n}{:}\rho(f_n) \end{array}$$

Lemma 6 *If $P \xrightarrow[w]{s} Q$ with respect to Δ and ϱ, then $[\![P]\!] \xrightarrow[w]{s} [\![Q]\!]$ with respect to $[\![\Delta]\!]$ and $[\![\varrho]\!]$.*

Proof. By rule induction. We need to consider the three rules (NEW), (APP), and (COM) by which to infer $P \xrightarrow[w]{s} Q$.

- Rule (NEW). Suppose that $\mathbf{new}\ x{:}\rho.\ P \xrightarrow[w]{s} \mathbf{new}\ x{:}\rho.\ Q$ is inferred as follows:

$$\frac{P \xrightarrow[w]{s} Q \qquad \varrho(x) = \rho}{\mathbf{new}\ x{:}\rho.\ P \xrightarrow[w]{s} \mathbf{new}\ x{:}\rho.\ Q}$$

The induction hypothesis yields that $[\![P]\!] \xrightarrow[w]{s} [\![Q]\!]$. As argued above, $\varrho(x) = \rho$ is equivalent to that $[\![\varrho]\!](x_f) = \rho(f)$ for all $f \in \mathscr{F}$. We can thus infer $[\![\mathbf{new}\ x{:}\rho.\ P]\!] \xrightarrow[w]{r} [\![\mathbf{new}\ x{:}\rho.\ Q]\!]$ by applying the (NEW) rule in an iterative manner:

$$\frac{[\![P]\!] \xrightarrow[w]{s} [\![Q]\!] \qquad \varrho(x_{f_1}) = [\![\rho]\!](f_1) \qquad \ldots \qquad [\![\varrho]\!](x_{f_n}) = \rho(f_n)}{\begin{array}{l}\mathbf{new}\ x_{f_1}{:}\rho(f_1).\ \cdots\ .\ \mathbf{new}\ x_{f_n}{:}\rho(f_n).\ [\![P]\!] \xrightarrow[w]{s} \\ \mathbf{new}\ x_{f_1}{:}\rho(f_1).\ \cdots\ .\ \mathbf{new}\ x_{f_n}{:}\rho(f_n).\ [\![Q]\!]\end{array}}$$

- Rule (APP). In this case, the judgment has been derived as follows:

$$\frac{P_{i_1} = A(\tilde{y}) \qquad A(\tilde{x}) \triangleq \mathbf{new}\ \widetilde{z{:}\rho}.\ Q \text{ in } \Delta}{\Pi_{i=1}^n P_i \xrightarrow[i_1]{\infty} \mathbf{new}\ \widetilde{z{:}\rho}.\ (Q[\tilde{x} \mapsto \tilde{y}] \mid \Pi_{i=1,i\neq i_1}^n P_i)}$$

By translation the following rule instance applies too:

$$\frac{[\![P_{i_1}]\!] = A(\tilde{y}_{\mathscr{F}}) \qquad A(\tilde{x}_{\mathscr{F}}) \triangleq [\![\mathbf{new}\ \widetilde{z{:}\rho}.\ Q]\!] \text{ in } [\![\Delta]\!]}{\Pi_{i=1}^n [\![P_i]\!] \xrightarrow[i_1]{\infty} [\![\mathbf{new}\ \widetilde{z{:}\rho}]\!]\ ([\![Q]\!][\tilde{x}_{\mathscr{F}} \mapsto \tilde{y}_{\mathscr{F}}] \mid \Pi_{i=1,i\neq i_1}^n [\![P_i]\!])}$$

By Lemma 5, this is $[\![\Pi_{i=1}^n P_i]\!] \xrightarrow[i_1]{\infty} [\![\mathbf{new}\ \widetilde{z{:}\rho}.\ (Q[\tilde{x} \mapsto \tilde{y}] \mid \Pi_{i=1,i\neq i_1}^n P_i)]\!]$.

- Rule (COM). The judgment has thus been inferred by an application of the communication rule:

$$\frac{C^{j_1}_{i_1} = x?f(\tilde{z}).\mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ Q_1 \qquad C^{j_2}_{i_2} = x!f(\tilde{y}).\mathbf{new}\ \widetilde{x_2{:}\rho_2}.\ Q_2}{\Pi^n_{i=1} \sum^{m_i}_{j=1} C^j_i \xrightarrow[i_1,j_1,i_2,j_2]{\varrho(x)(f)} \begin{cases} \mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ \mathbf{new}\ \widetilde{x_2{:}\rho_2}. \\ (Q_1[\tilde{z} \mapsto \tilde{y}] \mid Q_2 \mid \Pi^n_{i=1, i\neq i_1,i_2} \sum^{m_i}_{j=1} C^j_i) \end{cases}}$$

  We can then apply the communication rule as follows too:

$$\frac{[\![C^{j_1}_{i_1}]\!] = x_f?(\tilde{z}_{\mathscr{F}}).[\![\mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ Q_1]\!] \qquad [\![C^{j_2}_{i_2}]\!] = x_f!(\tilde{y}_{\mathscr{F}}).[\![\mathbf{new}\ \widetilde{x_2{:}\rho_2}.\ Q_2]\!]}{\Pi^n_{i=1} \sum^{m_i}_{j=1} [\![C^j_i]\!] \xrightarrow[i_1,j_1,i_2,j_2]{\varrho(x_f)} \begin{cases} [\![\mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ \mathbf{new}\ \widetilde{x_2{:}\rho_2}. \\ (Q_1[\tilde{z} \mapsto \tilde{y}] \mid Q_2 \mid \Pi^n_{i=1, i\neq i_1,i_2} \sum^{m_i}_{j=1} C^j_i)]\!] \end{cases}}$$

Lemma 7 *If* $[\![P]\!] \xrightarrow[w]{s} Q'$ *with respect to* $[\![\Delta]\!]$ *and* $[\![\varrho]\!]$ *then there exists* $Q \in [\![Q']\!]^{-1}$ *such that* $P \xrightarrow[w]{s} Q$ *with respect to* Δ *and* ϱ.

Proof. By rule induction. We need to consider the three rules (NEW), (APP), and (COM) by which to infer $[\![P]\!] \xrightarrow[w]{s} Q'$ with respect to $[\![\Delta]\!]$ and $[\![\varrho]\!]$.

- Rule (NEW). In this case, the above judgment was inferred as follows:

$$\frac{[\![P_1]\!] \xrightarrow[w]{s} Q'_1 \qquad [\![\varrho]\!](x_{f_1}) = \rho(f_1) \qquad \dots \qquad [\![\varrho]\!](x_{f_n}) = \rho(f_n)}{\begin{array}{l}[\![P]\!] = \mathbf{new}\ x_{f_1}{:}\rho(f_1).\ \cdots .\ \mathbf{new}\ x_{f_n}{:}\rho(f_n).\ [\![P_1]\!] \xrightarrow[w]{s} \\ Q' = \mathbf{new}\ x_{f_1}{:}\rho(f_1).\ \cdots .\ \mathbf{new}\ x_{f_n}{:}\rho(f_n).\ Q'_1\end{array}}$$

  By induction hypothesis, there exists $Q_1 \in [\![Q'_1]\!]^{-1}$ such that $P_1 \xrightarrow[w]{s} Q_1$ with respect to Δ and ϱ. The hypotheses of the rule yield $\varrho(x) = \rho$. We can thus apply the (NEW) rule as follows:

$$\frac{P_1 \xrightarrow[w]{s} Q_1 \qquad \varrho(x) = \rho}{\mathbf{new}\ x{:}\rho.\ P_1 \xrightarrow[w]{s} \mathbf{new}\ x{:}\rho.\ Q_1}$$

  Lemma 6 shows that $[\![\mathbf{new}\ x{:}\rho.\ P_1]\!] \xrightarrow[w]{s} [\![\mathbf{new}\ x{:}\rho.\ Q_1]\!]$. This is equivalent to:

$$[\![P]\!] \xrightarrow[w]{s} \mathbf{new}\ x_{f_1}{:}\rho(f_1).\ \cdots .\ \mathbf{new}\ x_{f_n}{:}\rho(f_n).\ [\![Q_1]\!] = Q'$$

- Rule (APP). The judgment has in this case been inferred as follows:

$$\frac{[\![P_{i_1}]\!] = A(\tilde{y}_{\mathscr{F}}) \qquad A(\tilde{x}_{\mathscr{F}}) \triangleq [\![\mathbf{new}\ \widetilde{z{:}\rho}.\ Q_1]\!] \text{ in } [\![\Delta]\!]}{[\![P]\!] = \Pi_{i=1}^{n}[\![P_i]\!] \xrightarrow[i_1]{\infty} Q' = [\![\mathbf{new}\ \widetilde{z{:}\rho}]\!]\ ([\![Q_1]\!][\tilde{x}_{\mathscr{F}} \mapsto \tilde{y}_{\mathscr{F}}] \mid \Pi_{i=1,i\neq i_1}^{n}[\![P_i]\!])}$$

By Lemma 5, we have $[\![Q_1]\!][\tilde{x}_{\mathscr{F}} \mapsto \tilde{y}_{\mathscr{F}}] = [\![Q_1[\tilde{x} \mapsto \tilde{y}]]\!]$. Hence, $Q' = [\![Q]\!]$ where $Q = \mathbf{new}\ \widetilde{y{:}\rho}.\ (Q_1[\tilde{x} \mapsto \tilde{y}] \mid \Pi_{i=1,i\neq i_1}^{n} P_i)$. Furthermore, we can infer $P \xrightarrow[i_1]{\infty} Q$ as follows:

$$\frac{P_{i_1} = A(\tilde{y}) \qquad A(\tilde{x}) \triangleq \mathbf{new}\ \widetilde{y{:}\rho}.\ Q_1 \text{ in } \Delta}{P = \Pi_{i=1}^{n} P_i \xrightarrow[i_1]{\infty} Q = \mathbf{new}\ \widetilde{y{:}\rho}.\ (Q_1[\tilde{x} \mapsto \tilde{y}] \mid \Pi_{i=1,i\neq i_1}^{n} P_i)}$$

- Rule (COM). The judgment is now inferred as follows:

$$\frac{[\![C_{i_1}^{j_1}]\!] = x_f?(\tilde{z}_{\mathscr{F}}).[\![\mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ Q_1]\!] \qquad [\![C_{i_2}^{j_2}]\!] = x_f!(\tilde{y}_{\mathscr{F}}).[\![\mathbf{new}\ \widetilde{x_2{:}\rho_2}.\ Q_2]\!]}{[\![P]\!] = \Pi_{i=1}^{n} \sum_{j=1}^{m_i} [\![C_i^j]\!] \xrightarrow[i_1,j_1,i_2,j_2]{\varrho(x_f)} Q'}$$

where $Q' = [\![\mathbf{new}\ \widetilde{x_1{:}\rho_1}]\!]\ [\![\mathbf{new}\ \widetilde{x_2{:}\rho_2}]\!]\ ([\![Q_1]\!][\tilde{z}_{\mathscr{F}} \mapsto \tilde{y}_{\mathscr{F}}] \mid [\![Q_2]\!] \mid \Pi_{i=1,i\neq i_1,i_2}^{n} \sum_{j=1}^{m_i} [\![C_i^j]\!])$. The substitution Lemma 5 yields equality between $[\![Q_1]\!][\tilde{z}_{\mathscr{F}} \mapsto \tilde{y}_{\mathscr{F}}]$ and $[\![Q_1[\tilde{z} \mapsto \tilde{y}]]\!]$. Thus, $Q' = [\![Q]\!]$ where $Q = \mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ \mathbf{new}\ \widetilde{x_2{:}\rho_2}.\ (Q_1[\tilde{z} \mapsto \tilde{y}] \mid Q_2 \mid \Pi_{i=1,i\neq i_1,i_2}^{n} \sum_{j=1}^{m_i} C_i^j)$. Hence, we can infer $P \xrightarrow[i_1,j_1,i_2,j_2]{\varrho(x)(f)} Q$ as follows:

$$\frac{C_{i_1}^{j_1} = x?f(\tilde{z}).\mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ Q_1 \qquad C_{i_2}^{j_2} = x!f(\tilde{y}).\mathbf{new}\ \widetilde{x_2{:}\rho_2}.\ Q_2}{\Pi_{i=1}^{n} \sum_{j=1}^{m_i} C_i^j \xrightarrow[i_1,j_1,i_2,j_2]{\varrho(x)(f)} \left\{ \begin{array}{l} \mathbf{new}\ \widetilde{x_1{:}\rho_1}.\ \mathbf{new}\ \widetilde{x_2{:}\rho_2}. \\ (Q_1[\tilde{z} \mapsto \tilde{y}] \mid Q_2 \mid \Pi_{i=1,i\neq i_1,i_2}^{n} \sum_{j=1}^{m_i} C_i^j) \end{array} \right.}$$

Lemma 8 *$P \xrightarrow[w]{s} Q$ with respect to Δ and ϱ iff $[\![P]\!] \xrightarrow[w]{s} [\![Q]\!]$ with respect to $[\![\Delta]\!]$ and $[\![\varrho]\!]$.*

Proof. The implication from the left to the right is shown by Lemma 6. For the converse assume $[\![P]\!] \xrightarrow[w]{s} [\![Q]\!]$ with respect to $[\![\Delta]\!]$ and $[\![\varrho]\!]$. By Lemma 7 there exists $R \in [\![[\![Q]\!]]\!]^{-1}$ such that $P \xrightarrow[w]{s} R$. Since $[\![\cdot]\!]$ is injective, it follows that $R = Q$ so that $P \xrightarrow[w]{s} Q$.

Lemma 9 *$P \xrightarrow{\infty(p)} Q$ with respect to Δ and ϱ iff $[\![P]\!] \xrightarrow{\infty(p)} [\![Q]\!]$ with respect to $[\![\Delta]\!]$ and $[\![\varrho]\!]$.*

Proof.

'$\Rightarrow$' Judgments $P \xrightarrow{\infty(p)} Q$ with respect to Δ and ϱ are derived by rule (COUNT):

$$\frac{P \equiv P' \qquad \begin{array}{l} n = \sharp\{w \in \mathbb{N} \cup \mathbb{N}^4 \mid P' \xrightarrow[w]{\infty} R' \equiv Q\} \neq 0 \\ m = \sharp\{w \in \mathbb{N} \cup \mathbb{N}^4 \mid P' \xrightarrow[w]{\infty} R'\} \end{array}}{P \xrightarrow{\infty(n/m)} Q} \tag{4.4.1}$$

The corresponding judgment can be derived as follows by rule (COUNT):

$$\frac{[\![P]\!] \equiv [\![P']\!] \qquad \begin{array}{l} n = \sharp\{w \in \mathbb{N} \cup \mathbb{N}^4 \mid [\![P']\!] \xrightarrow[w]{\infty} R \equiv [\![Q]\!]\} \neq 0 \\ m = \sharp\{w \in \mathbb{N} \cup \mathbb{N}^4 \mid [\![P']\!] \xrightarrow[w]{\infty} R\} \end{array}}{[\![P]\!] \xrightarrow{\infty(n/m)} [\![Q]\!]} \tag{4.4.2}$$

To see this, we must show that $\exists R'.P' \xrightarrow[w]{\infty} R' \equiv Q$ is equivalent to $\exists R.[\![P']\!] \xrightarrow[w]{\infty} R \equiv [\![Q]\!]$. This follows from Lemmas 5, 6, and 7.

'$\Leftarrow$' The judgment $[\![P]\!] \xrightarrow{\infty(n/m)} [\![Q]\!]$ must be inferred as follows:

$$\frac{[\![P]\!] \equiv P_1 \qquad \begin{array}{l} n = \sharp\{w \in \mathbb{N} \cup \mathbb{N}^4 \mid P_1 \xrightarrow[w]{\infty} R \equiv [\![Q]\!]\} \neq 0 \\ m = \sharp\{w \in \mathbb{N} \cup \mathbb{N}^4 \mid P_1 \xrightarrow[w]{\infty} R\} \end{array}}{[\![P]\!] \xrightarrow{\infty(n/m)} [\![Q]\!]}$$

Since $[\![P]\!] \equiv P_1$ we can apply Lemma 5 which yields the existence of $P' \in [\![P_1]\!]^{-1}$ such that $P \equiv P'$. With this P' the inference step (4.4.2) becomes valid, so that we can infer $P \xrightarrow{\infty(n/m)} Q$ with respect to Δ and ϱ as in (4.4.1).

Lemma 10 *$P \xrightarrow{r} Q$ with respect to Δ and ϱ iff $[\![P]\!] \xrightarrow{r} [\![Q]\!]$ with respect to $[\![\Delta]\!]$ and $[\![\varrho]\!]$.*

Proof.

'⇒' The judgment $P \xrightarrow{r} Q$ with respect to Δ and ϱ is inferred by rule (SUM).

$$\frac{P \equiv P' \qquad r = \sum_{P' \xrightarrow[w]{r'} R' \equiv Q} r' \neq 0 \qquad \neg \exists R'' \exists w'.\ P' \xrightarrow[w']{\infty} R''}{P \xrightarrow{r} Q} \tag{4.4.3}$$

The corresponding judgment can be inferred as follows:

$$\frac{[\![P]\!] \equiv [\![P']\!] \qquad r = \sum_{[\![P']\!] \xrightarrow[w]{r'} R \equiv [\![Q]\!]} r' \neq 0 \qquad \neg \exists R'' \exists w'.\ [\![P']\!] \xrightarrow[w']{\infty} R''}{[\![P]\!] \xrightarrow{r} [\![Q]\!]} \tag{4.4.4}$$

To see this, it is sufficient to show two equivalences, both following from Lemmas 5, 6, and 7:

1. $\exists R'.P' \xrightarrow[w]{r'} R' \equiv Q$ iff $\exists R.[\![P']\!] \xrightarrow[w]{r'} R \equiv [\![Q]\!]$.
2. $\exists R'.P' \xrightarrow[w]{\infty} R'$ iff $\exists R.[\![P']\!] \xrightarrow[w]{\infty} R$.

'⇐' The judgment $[\![P]\!] \xrightarrow{r} [\![Q]\!]$ must be inferred as follows:

$$\frac{[\![P]\!] \equiv P_1 \qquad r = \sum_{P_1 \xrightarrow[w]{r'} R \equiv [\![Q]\!]} r' \neq 0 \qquad \neg \exists R'' \exists w'.\ P_1 \xrightarrow[w']{\infty} R''}{[\![P]\!] \xrightarrow{r} [\![Q]\!]}$$

Since $[\![P]\!] \equiv P_1$, Lemma 5 yields the existence of $P' \in [\![P_1]\!]^{-1}$ such that $P \equiv P'$. With this P' the inference step (4.4.4) becomes valid, so that we can infer $P \xrightarrow{r} Q$ with respect to Δ and ϱ as in (4.4.3).

4.5 Higher-order definitions

Our process definitions up to now are first-order, in that they need to be provided statically at compile time. We show how to render definitions higher-order, so that they may also be determined dynamically at run time. The definitions we obtain hereby are reminiscent of procedures in Mozart-Oz[5] of Smolka (1995).

[5] `http://www.mozart-oz.org`

Processes	P	$::=$	$P_1 \mid P_2$	parallel composition
		$\mid$	$\mathbf{new}\ x{:}\rho.\ P$	channel creation
		$\mid$	$C_1 + \ldots + C_n$	sum $(n \geq 0)$
		$\mid$	$x(\tilde{y})$	application
		$\mid$	$x(\tilde{y})\ \triangleq\ P$	definition
Guarded processes	C	$::=$	$x?f(\tilde{y}).P$	pattern input
		$\mid$	$x!f(\tilde{y}).P$	tuple output

Table 4.8: Syntax of the stochastic π-calculus with higher-order definitions and input patterns

In a first-order setting, it is impossible to define classes of objects as functions which create objects. Rather, all objects must be defined statically. Dynamic values are passed as parameters during object invocation. The drawback of this programming style is that argument lists quickly reach a considerable length.

We obtain a simpler syntax by incorporating higher-order definitions into the π-calculus. The syntactic categories of names of channels x and parametric processes A are unified, same for the two syntactic categories of definitions D and processes P. Parametric definitions become replicated inputs and belong to the category of processes; definitions can be composed by parallel composition in order to build sets thereof.

The syntax of the extended π-calculus is listed in Table 4.8. It is built from an infinite set of channel names $\mathscr{N} = \{x, y, z, \ldots\}$, and a set of *function names* $f \in \mathscr{F}$. Note that definitions $x(\tilde{y})\ \triangleq\ P$ are drawn from the same alphabet as channels. In the literature, they are often written as replicated inputs $!x?(\tilde{y}).P$. The fragment with higher-order definitions, yet no sums, was studied by Niehren (2000). It can express both the eager and the lazy λ-calculus.

The reduction semantics needs to be adapted in the application rule for parametric processes, where the definitions become internal.

$$z(\tilde{x}) \mid z(\tilde{y}) \triangleq P \quad \rightarrow \quad P[\tilde{y} \mapsto \tilde{x}] \mid z(\tilde{y}) \triangleq P$$

if $\tilde{y}$ free is for $\tilde{x}$ in P. The stochastic semantics is not affected either, except for the application rule:

$$\frac{P_{i_1} = z(\tilde{y}) \qquad P_{i_2} = z(\tilde{x}) \triangleq \mathbf{new}\ \widetilde{y{:}\rho}.\ Q}{|_{i=1}^{n} P_i \ \xrightarrow[i_1]{\infty}\ \mathbf{new}\ \widetilde{y{:}\rho}.\ (Q[\tilde{x} \mapsto \tilde{y}] \mid |_{i=1, i \neq i_1}^{n} P_i)}$$

where $1 \leq i_1, i_2 \leq n$ and Q contain no top-level **new**-binders. We illustrate the usefulness of higher-order definitions at the example of class definitions. Consider promoter objects with two profiles:

```
PR_free(me,op)   ≜ rnap?bind().PR_bound(me,op) + ...
PR_bound(me,op)  ≜ me?free().PR_free(me,op)
                 + op!block().PR_bound + ...
```

Instead of threading identities me, one might want them to become global names, that are passed at object creation time. Promoter creation can then be defined as follows, assuming that all promoters start in free:

```
make_pr(me,op) ≜ new pr_free. new pr_bound.
   pr_free()  ≜ rnap?bind().pr_bound() + ... |
   pr_bound() ≜ me?free().pr_free()
              + op!block().pr_bound + ... |
   pr_free()
```

With this we can create distinct promoters dynamically, by applying the definition of make_pr. We need to pass as arguments the actual name $\mathsf{me_i}$ of the object, and the name of its operator site $\mathsf{op_i}$. The channel rnap can remain global.

```
make_pr(me1,op1) | make_pr(me2,op1)
```

We consider processes as erroneous if they use the same name for naming different definitions, i.e. if some x occurs in subexpressions of the following form: $x(\tilde{y}) \triangleq P \mid x(\tilde{y}') \triangleq P'$. Such errors can be excluded statically by syntax restrictions as in TyCO, or raise errors dynamically as in Mozart-Oz. The static way would be to introduce all definitions by recursive let-operators. This is generally sufficient; see for the above example:

```
make_pr(me,op) ≜
     letrec
        pr_free()  ≜ rnap?bind().pr_bound() + ... |
        pr_bound() ≜ me?free().pr_free()
                   + op!block().pr_bound() + ... |
     in
        pr_free()
     end
```

Finally, let us note a second kind of type errors which may arise now, since we no longer fix the arity of defined names (in contrast to names of parametric processes before). These errors have the form $x(\tilde{z}) \mid x(\tilde{y}) \triangleq P$ where the lengths of $\tilde{z}$ and $\tilde{y}$ differ.

4.6 Summary

In this chapter, we have presented the stochastic π-calculus with input patterns. We have defined the stochastic semantics of processes in this calculus in terms of continuous time Markov chains. Our definition of these CTMCs is new, even for the case without input patterns, but it induces existing simulation machines. We have shown how to encode input patterns, so that their stochastic semantics in terms of CTMCs is preserved.

As we will see in the next chapter, input pattern permit to define object-oriented abstractions. These can greatly facilitate π-calculus based modeling and simulation for systems biology, as illustrated by the modeling studies in the remaining chapters.

CHAPTER 5

Concurrent Objects and Modules

This chapter first introduces notions of concurrent *objects* in the stochastic π-calculus with input patterns. We then show how to define object *extension* on meta level, making use of suitable notions of *inheritance*. Objects with inheritance are among the central reasons for the extensibility of the model of transcription and translation presented in Chapter 7. Note that previous π-calculus based approaches to biomolecular modeling do not use object-orientation.

We consider different object notions with increasing expressiveness. Objects may be passive, active, or mixed, depending on whether they may send or receive messages, or both. They may use particular channels for different kinds of interactions, or may restrict themselves to a single channel, the object's identity. Multi-profile objects offer different interfaces in different states, while single-profile objects stick to a permanent interface.

The objects of the TyCO language are passive, interact on a single channel, and may have multiple profiles. In this context multi-profile objects are termed non-uniform, see Ravara and Vasconcelos (2000). The objects presented in our work are more general, in that they may comprise both active and passive aspects, and permit interactions on multiple channels.

The use of multi-profile concurrent objects for modeling in systems biology was originally proposed by Duchier and Kuttler (2006). At that time however nor formal language existed to express such objects. The addition of input patterns to the synchronous π-calculus solves this problem at the symbolic level, the definition of the stochastic semantics of the previous chapter solves the numeric question.

The purpose of inheritance is to extend objects with new functions. We define inheritance for multi-profile objects on meta-level, rather than expressing it within the π-calculus. We keep the notion of inheritance static, so that all extension operations can be compiled away; the result is an expression of the stochastic π-calculus. Richer dynamic notions of inheritance are left to future research.

We present a *module* system for defining multi-profile objects in the stochastic π-calculus, while using inheritance. Modules are collections of definitions; we use them to specify multi-profile objects or collections thereof. The module system is inspired by that of Mozart-Oz, except for the aspect of inheritance. This latter is in turn inspired by that of SML, except for typing aspects.

5.1 Multi-profile objects

Multi-profile objects are objects with multiple *states*, each of which may offer a distinct *interface*. An interface is the set of function names for which the object provides definitions.

In the stochastic π-calculus, we express classes Obj of multi-profile objects by a set of definitions, one per state. For all states p of the object, we assume a parametric process name Obj_p in order to name this profile of the object Obj. The class of a multi-profile object Obj with profiles p_1, ..., p_n is defined by a collection of mutually recursive definitions of the following form:

$$\begin{aligned}
\mathsf{Obj\_p_1}(\tilde{z_1}) &\triangleq \mathsf{C^1_1} + \ldots + \mathsf{C^1_{k_1}} \\
&\ldots \\
\mathsf{Obj\_p_n}(\tilde{z_n}) &\triangleq \mathsf{C^n_1} + \ldots + \mathsf{C^n_{k_n}}
\end{aligned}$$

An object of this class in some profile $\mathsf{p_i}$ is created by the application $\mathsf{Obj\_p_i}(\tilde{y})$. Single profile objects are a special case of multi-profile objects, where the number of profiles n equals 1.

Let us consider Semaphore objects as a first example. These are like rooms or sites, that can be occupied at most by a single visitor. We represent semaphores as objects with two profiles free and bound. Each Semaphore is given an identity by some channel name me. The state transitions of Semaphores are depicted in Figure 5.1. In state free, a Semaphore can only accept a bind message and become bound. In state bound a Semaphore can only receive a free message and become Semaphore_free. It crucially matters that Semaphore may not accept bind messages, while in the bound profile. We obtain this behavior with the definitions in Figure 5.2.

We call a multi-profile object *passive* if all guarded processes C^j_i in its

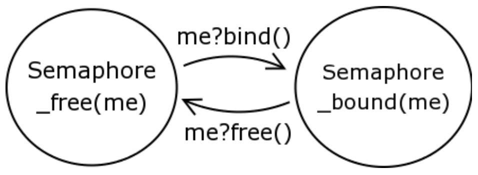


Figure 5.1: State transitions of a Semaphore

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

Figure 5.2: Defining semaphores in the π-calculus as multi-profile objects

definitions are inputs, active if all of them are outputs, and mixed otherwise. In TyCO, all objects are passive and furthermore restricted to input on a same dedicated channel me, that identifies the object. In this case, all definitions have the following form:

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

The symbols $\mathsf{f_i}$ name the functions offered by the object in that profile, and determine its *interface* or *type*.

5.2 Inheritance

In our modeling studies, we will frequently extend object classes by new functions. Our class specifications by inheritance can be compiled into ordinary π-calculus definitions.

Let us begin with single profile objects, considering the class Obj with the following definition:

$$\mathsf{Obj}(\tilde{z}) \triangleq \mathsf{C_1} + \ldots + \mathsf{C_k}$$

We extend the class Obj to Obj2 by adding new alternative behaviors:

```
Obj2 extends Obj
Obj2(z̃) extended by C_{k+1} + ...+ C_l
```

This definition by inheritance can be resolved into a regular π-calculus definition as follows:

$$\mathsf{Obj2}(\tilde{z}) \triangleq \mathsf{C_1} + \ldots + \mathsf{C_l}\ [\mathsf{Obj} \mapsto \mathsf{Obj2}]$$

The substitution [Obj $\mapsto$ Obj2] denotes the renaming of all recursive calls to some Obj into recursive calls to Obj2. For multi-profile objects, the situation is somewhat more tedious. We consider some class Obj with n profiles:

$$\begin{array}{l} \mathsf{Obj\_p_1}(\tilde{z_1}) \triangleq \mathsf{C}^1_1 + \ldots + \mathsf{C}^1_{k_1} \\ \quad\ldots \\ \mathsf{Obj\_p_n}(\tilde{z_n}) \triangleq \mathsf{C}^n_1 + \ldots + \mathsf{C}^n_{k_n} \end{array}$$

We extend class Obj to Obj2 in the following manner:

$$\begin{array}{l} \mathsf{Obj2}\ \textbf{extends}\ \mathsf{Obj} \\ \mathsf{Obj2\_p_1}(\tilde{z_1})\ \textbf{extended by}\ \mathsf{C}^1_{k_1+1} + \ldots + \mathsf{C}^1_{l_1} \\ \quad\ldots \\ \mathsf{Obj2\_p_n}(\tilde{z_n})\ \textbf{extended by}\ \mathsf{C}^n_{k_n+1} + \ldots + \mathsf{C}^n_{l_n} \end{array}$$

This specification with inheritance is to be resolved into the following π-calculus definitions:

$$\begin{array}{l} \mathsf{Obj2\_p_1}(\tilde{z_1}) \triangleq \mathsf{C}^1_1 + \ldots + \mathsf{C}^1_{l_1}\ [\mathsf{Obj} \mapsto \mathsf{Obj2}] \\ \quad\ldots \\ \mathsf{Obj2\_p_n}(\tilde{z_n}) \triangleq \mathsf{C}^n_1 + \ldots + \mathsf{C}^n_{l_n}\ [\mathsf{Obj} \mapsto \mathsf{Obj2}] \end{array}$$

The substitution renames all recursive calls to profiles $\mathsf{Obj\_p_i}$ into recursive calls to $\mathsf{Obj2\_p_i}$ for $1 \leq i \leq n$.

5.3 Module system

We present a module notation for the definition of multi-profile objects in the stochastic π-calculus, taking advantage of inheritance. This module system is inspired by that of Mozart-Oz (Roy, 2005), except for the aspect of inheritance. This latter is in turn inspired by the module system of SML except for typing aspects.

Basically, a module is a name for a *collection of definitions*. Such collections can be defined by importing definitions from other modules, or by inheritance from existing definitions. Modules export only distinguished definitions, while keeping others internal. Modules are self contained, in that all names used in a module must be declared either in the module's header, or within the module itself. We provide some type annotations in module headers, aiming to improve legibility and facilitate the application of the contributed definitions.

As a first example, we turn the Semaphore definitions from Figure 5.2 into the module 'semaphore' in Figure 5.3. This module exports a unique process named Semaphore, with 0-ary function symbols bind and free. The definitions of Semaphore_free and Semaphore_bound remain internal to the module.

```
module 'semaphore'
export
   Semaphore with bind/0,free/0
define
   Semaphore(me) ≜ Semaphore_free(me)
   Semaphore_free(me) ≜ me?bind().Semaphore_bound(me)
   Semaphore_bound(me) ≜ me?free().Semaphore_free(me)
```

Figure 5.3: Semaphore module

Preamble. Modules start with a preamble, containing declarations and other statements, followed by definitions. The first line of the preamble defines the module's name.

```
module <module_name>
```

Module names are character strings. We discuss further preamble statements in the order they typically occur.

Public channels. Some objects communicate via public channels, which are neither part of their parameter list, nor locally created. We provide the following statements to declare such channels:

```
public <channel_name>
public <channel_name> with <type>
public <channel_name> as <channel_name>
public <channel_name> from  <Table>
```

The first statement authorizes output on a public[1] channel, the second specifies the patterns locally defined objects may receive over some public channel. The third statement <public_channel> as < local_channel> is useful to address a public channel by another name, local to the module. The last declares channels introduced and assigned a stochastic parameterization in some table.

Import statements. Each module delivers material that can be re-used by other modules. We reserve the following statements for imports:

```
import <module_name>
import <definition_name> from <module_name>
import <module_name>(<definition_name >,...,
    definition_name) from <module_name>
```

[1]We use the alternative wordings *global* channel and *public*.

The first makes all exported definitions of the imported module available, while the second selects a few. The third allows to tag the component Object imported from module Othermodule, that we can address by the name Othermodule.Object. This is useful when importing definitions using the same name in different modules, or when extending on a definition from another module.

Rate function statements. We have not yet considered the parameterization of our objects, i.e. the rates determining the actual temporal behavior of interaction in the π-calculus. As introduced in Chapter 4 channels are associated with rate functions ρ upon creation (new x:ρ). These fix stochastic rates ρ(f) for all pairs (x,f) where f are the patterns exchanged over channel x. We conveniently import such rate definitions into a module, defined in an external table, using the following statement:

```
rate <rho> from <Table>
```

Export statements. We specify the *interface* of defined objects by export statements.

```
export <definition_name> with <type>
export <definition_name>
```

In <type>, we list all input patterns an object may receive over its identifier channel me. Note that this does not reveal any internal detail of the object. For instance, it does not tell *when* input patterns may by received, which may depend on the current profile of an object. Note also that the export statement only fixes the arity of functions, yet not their stochastic parameterization.

We document inheritance relations at the level of export statements.

```
<definition_name> extends <Othermodule.Object> by <type>
<definition_name> extends <Othermodule.Object>
```

The first statement is used to signalize the addition of new pattern input over an object's identifier channel me, the second for new interaction potential using another channel.

Definitions. Finally, the keyword

```
define
```

delimits the end of the preamble. The module's definitions follow. Note that in the module's core part, we use

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

Figure 5.4: Persistent list module

```
<definition_name_profile> extended by C_{k+1} + ...+ C_l
```

to detail on an extension, that in the preamble was announced as

```
<definition_name> extends <module_name>.<
    definition_name>
```

Comments. Modules may comprise comment lines starting in the delimiter //, which must be repeated if the comment exceeds one line. In this document, for the sake of legibility we color comments in grey.

```
Nothing ≜ 0 // this process has vanished!
```

5.4 Examples

We first present persistent lists as an example for single-profile objects, which we turn into degradable list by inheritance. As an example of multi-profile objects we then discuss persistent queuing lists, and render them degradable by inheritance.

5.4.1 Persistent lists

A list consists of a sequence of nodes, each with a successor next and a value val. We design the class Node with three parameters: me, next, and val. The successor of a list's last Node is represented by a dedicated Nil object. Interfaces of Node objects specified in Figure 5.4 export three unary functions: getNext, getValue, and isNil. The Nil object provides only the unary function isNil.

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

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

We fix the stochastic behavior of persistent Node and Nil objects as follows:

$$\begin{array}{ll} \rho_{\mathsf{plist}}(\mathsf{getNext})=30 & \rho_{\mathsf{plist}}(\mathsf{getValue})=\infty \\ \rho_{\mathsf{plist}}(\mathsf{isNil})=\infty & \rho_{\mathsf{pnil}}(\mathsf{isNil})=\infty \end{array}$$

The rate function ρ_{plist} fixes the temporal behavior of the second node of the list, located at n_2, ρ_{pnil} that of the Nil object. We illustrate list processing with a Walker. It traverses the list by querying each node for it successor via calling the function 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)
```

Two notes about the Walker are in order.

First, it does not strictly adhere to our definition of object, since it uses a **new**-binder in front of a sum. More importantly, it is unclear how to define a useful notion of inheritance for Walker like processes, so that they they do different things while walking over a list. The same problem applies to all devices processing representatives of macromolecules (i.e., data structures) in Chapters 7 and 8: the abstractions of RNAP and ribosome proceed by calling functions that are offered by DNA and mRNA representatives.

The second point worthwhile stating regards the *creation of new channels* in this example. When *emulating a function call* in the π-calculus, we pass a fresh private channel on which the result comes back[2], in this case the Walker's c_1 and c_2. Such channels are of infinite rate. Note that for channels not assigned any ρ upon creation, all interactions are instantaneous (see page 58).

After importing module 'persistent list [a,b]', we let the Walker run over an example list instantiated at channel n_1:

[2]See (Milner, 2004).

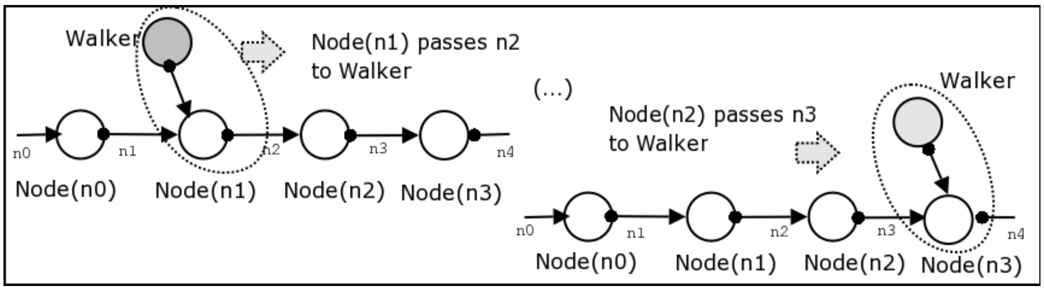


Figure 5.5: Walker stepping through a list; $\mathsf{Node}(n_i,n_{i+1},v_i)$ is here abbreviated as $\mathsf{Node}(n_i)$.

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

Suppose the first node's identifier n_1 was introduced with the same rates ρ_{plist}. All calls to getNext functions are then associated with a stochastic rate of 30. This is the single parameter determining an exponential distribution of waiting times, i.e. the inverse of its mean. Our Walker hence 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. This could be done by a Copier inspired by the Walker such that:

```
List(n1) | Copier(n1,n2) →* List(n1) | List(n2)
```

While we leave the Copier's definition to the reader, we will later present our abstraction of Rnap for the related task of transcription from Dna into mRna.

5.4.2 Degradable lists

We now illustrate object extension by deriving a degradable list variant from the previous, persistent one. The distinguishing feature is that degradable may be destroyed while processing.

The import statement in line 2 of their defining module (Figure 5.6) 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

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

Figure 5.6: Degradable list module

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 'degradable 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 of Priami et al. (2001). We will discuss the reasons on page 94.

5.4.3 Queuing 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 how-

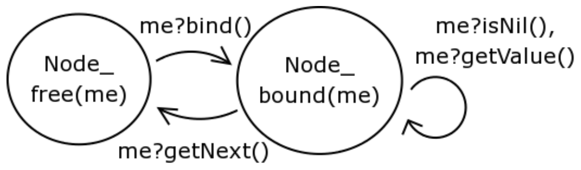


Figure 5.7: Transition diagram: persistent queuing list Node

```
 1 module 'persistent queuing 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)
```

Figure 5.8: Persistent queuing list module

ever not possible in transcription of DNA, nor in translation of mRNA or its degradation.

We impose *queuing* on visitors of a *persistent* list by incorporating a semaphore style behavior, distinguishing free and bound. Figure 5.7 illustrates the transitions between these profiles of a Node, defined by the module 'persistent queuing list' in Figure 5.8. The functions exported on the interface can only be used when the Node is in profile bound. Note that the function bind is not exported outside the module, but can only be called on a node by its predecessor. This happens during the call of the getNext function, which is re-defined 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 queuing list, we need to bind its first node:

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

Figure 5.9: Degradable queuing list module

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

A *degradable* queuing list can easily be obtained by inheritance, see Figure 5.9. 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.

5.5 Discussion: input patterns or protocols?

Let us discuss an alternative encoding of persistent nodes in the π-calculus. Instead of defining object interfaces by input patterns, we use smart handshake protocols. This works fine, but beyond poor legibility, it has the major inconvenience that inheritance gets out of reach.

Each function of Node_bound is now represented by a fresh name, which is created by the Node_free before profile switching. These names are passed from the nodes to the visitor following a handshake protocol illustrated in Figure 5.10. In a first step, the Node receives input some channel c over its me channel, knowing this is a binding request and that in step 2, it is supposed to return the channels associated with its interactions in the bound profile over channel c. The receiver in turn has precise knowledge on how to make subsequent use of these channels.

Let us now consider the *difference* between persistent and degradable node variants from a Visitor's perspective. Node extrudes one fresh private channels per possible interaction in step 2 of the protocol: getValue and getNext are extruded in either case, while only the degradable node passes kill. Visitors designed to interact with persistent Nodes can not be used on

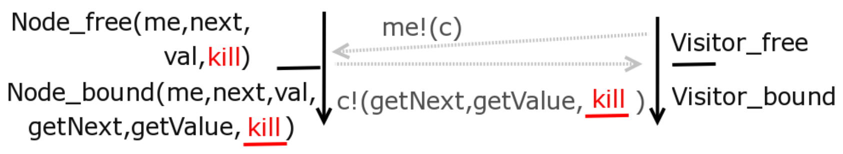


Figure 5.10: Handshake protocol for Node binding in π-calculus without pattern input. Degradable and persistent nodes differ in their protocols since only the former creates and extrudes a kill channel in step 2.

degradable ones, in contrast to our previous Walker encoded in our object π-calculus on page 90.

The reason is the modified protocol. Hence if we wanted to design both Walkers and Killers for such lists in the π-calculus without pattern input, we would have to pass on the full channel set to both Walkers and Killers - even if the former never made use of kill.

The code spells out as follows for the *persistent* versions:

```
Node_free(me, next , val) ≜
   me?(c).// handshake step 1
      new getNext.new getValue.
      c!(getNext,getValue). // handshake step 2
      Node_bound(me, next , val , getNext , getValue)

Node_bound(me, next , val , getNext , getValue) ≜
   getValue?(c).c!(val).
      Node_bound(me, next , val , getNext , getValue)
 + getNext?(c).c!(next).Node_free(me, next , val)
```

While for the *degradable* case, it is:

```
Node_free(me, next , val ,kill) ≜
   me?(c). // handshake step 1
      new getNext.new getValue.new kill_n.
      c!(getNext,getValue,kill_n). // handshake step 2
      Node_bound(me, next , val , getNext , getValue ,kill_n)
 + kill?().0

Node_bound(me, next , val , getNext , getValue ,kill) ≜
   getValue?(c).c!(val).
      Node_bound(me, next , val , getNext , getValue ,kill)
 + getNext?(c).c!(next).Node_free(me, next , val ,kill)
```

CHAPTER 6

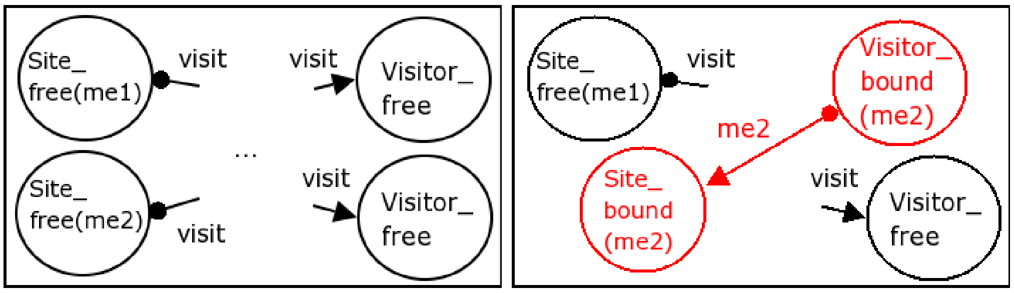
Modeling Techniques

This chapter presents recurrent modeling techniques emanating from the modeling studies of Chapters 7 and 8. We illustrate these by artificial examples, focusing on elements of concurrent control that are indeed encountered in biological cases, but may also occur at other places.

These techniques address phenomena of many-to-many communication (Section 6.1), applied for multiple entry points to queuing lists (Section 6.2), overlapping sites with mutually exclusive binding (Section 6.3), where we develop a mechanism that is useful to other cases of mutual dependencies beyond pairs of actors. We introduce timers (Section 6.4), and show the application of these to the modulation of stochastic rates of object functions (Section 6.5). Our approach to these phenomena in novel to modeling in systems biology, since it heavily relies on multi-profile objects (beyond those of TyCO). Thus, our technical solutions could not easily be expressed in previous π-calculus dialects.

Several among these phenomena regard the stochastic behavior. Indeed, the stochastic semantics is essential to the correct treatment overlapping sites, and other examples. We must prevent events at one site, that would disregard the peer site's state. In our models, inconsistent states can indeed be reached, yet our stochastic semantics ensures that they may not be abused of. Inconsistent states are left by immediate transitions, which are executed prior to any time-consuming transitions that might abuse of it.

The avoidance of inconsistent intermediate states is a well-known practical issue, e.g. in distributed databases (Vossen and Weikum, 2001). The solution there is to introduce transactions, and by them to render sequences of



(a) ***between visits*** (b) ***during a visit***

Figure 6.1: Many-to-many communication: at most one Visitor per Site.

actions atomic. Transactions for the π-calculus were proposed by Ciocchetta and Priami (2006). Fortunately, such heavier operations are not needed for the purpose of this thesis, thanks to our stochastic semantics with immediate transitions.

6.1 Many to many communication

Many-to-many communication over the same channel is inherently difficult, it requires not to mix up the messages of different partners.

Handshake protocols are the common technique to model many-to-many communication over the same channel in the π-calculus. As two actors establish a contact over this channel, they exchange a private channel, on which they subsequently continue their interaction. This private data exchange prohibits other concurrent actors from interfering.

We suggest a convenient alternative to handshaking for multi-profile objects. It applies to scenarios with many sites and visitors, where each site accepts at most one visitor at any time. Figure 6.1(a) illustrates the scenario without ongoing visits. We model sites and visitors as objects, assuming that Sites have unique identities $\mathsf{me_i}$. During the initial interaction, a Site passes its identity to the Visitor, by communication over the public channel visit. Visitors can then invocate functions on the Site. This constellation during interaction is depicted in Figure 6.1(b).

The semaphore-type requirement that Sites accept maximally one Visitor at a time can be fulfilled as previously with semaphores, i.e. by state changes of multi-profile objects. A Site grants access to its identity only while in free state. As this occurs, it changes into a bound state. Once the Visitor leaves the Site, the latter becomes free again. Before leaving, the Visitor must discard any further access capabilities on the Site's identifier.

```
public visit
Site_free(me)   ≜  visit!bind(me).Site_bound(me)
Site_bound(me)  ≜  me?unbind().Site_free(me)
Visitor_free()  ≜  visit?bind(site).Visitor_bound(site)
Visitor_bound(site) ≜  site!unbind().Visitor_free()

new me₁. new me₂.
Site_free(me₁) | Site_free(me₂) |
Visitor_free() | Visitor_free()
```

Figure 6.2: Implementing scenario with 2 visitors and 2 sites

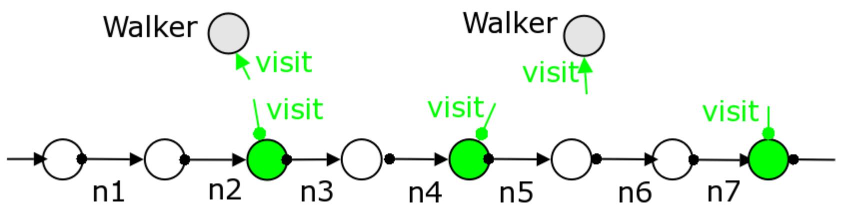


Figure 6.3: A list offering multiple entry points (green) to Walkers

The above scenario with each two Visitors and Sites is implemented in Figure 6.2. Note that Sites and Visitors may be enriched with further functions by inheritance.

In biological examples, many-to-many communication over global channels occurs between abstractions of RNAP and promoters in Section 7.1, in protein binding to operators modeled in Section 8.2, as well as in mRNA processing from Section 7.2.

6.2 Queuing lists with multiple entry points

Our next matter of interest is a list offering multiple points of entry, as illustrated in Figure 6.3. Each list Node can be visited by at most one Walker at the same time. Walkers can either come from the previous Node, or access the Node directly by using the public channel visit.

This is indeed a special case of the many-to-many scenario from Section 6.1. The Nodes in this list correspond to the previous Sites, the Walkers to Visitors. In contrast the previous setting, the sites are no longer independent, but part of a queuing list. This is because we want to prohibit a Walker from overtaking others, as appears when modeling RNA polymerases that

```
import
 QList(Node, Nil) from 'persistent queuing list'//Fig.5.8
public
   visit
export
   Node extends QList.Node // sends bind over visit
   Walker
define
   Node_free(me, next, val) extended by
      visit!bind(me).Node_bound(me, next, val)
   Walker() ≜ Walker_free()
   Walker_free() ≜ visit?bind(n).Walker_bound(n)
   Walker_bound(n) ≜ new c.
      me!getNext(c).c?(next).Walker_bound(next)
```

Figure 6.4: Implementing queuing lists with multiple entry points

transcribe through DNA, after starting at promoters.

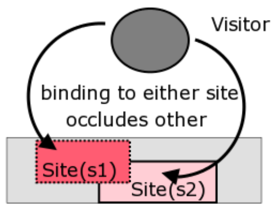
The implementation is listed in Figure 6.4. Nodes extend those of persistent queuing lists. The profile Node_free is extended by the capability to receive a visitor, as Site_free in Figure 6.2. Walker_free binds to a site in the same way as Visitor_free did before. But now, instead of being able to unbind, the walker needs to continue traversing the remaining list, similarly to the walker on page 90. It should be noted that multi-profile objects permit both mixed input and output, while using more than one interaction channel.

In our biological modeling, multiple entry points to lists are useful for several cases. One are tandem promoters, dealt with in Section 7.4.2. The abstraction of the mRNA transcribed there applies a solution developed for internal ribosomal binding sites, transcribed within polycistronic mRNA from operons (Figure 7.11).

6.3 Overlapping sites

We next model overlapping sites at s_1 and s_2 which may not receive visitors simultaneously, as illustrated in Figure 6.5(a). Whenever either of them is bound, the other becomes blocked. We must carefully prohibit inconsistent configurations in which the Site at s_1 is bound while that at s_2 is free, or vice versa.

In our models, inconsistent states can indeed be reached, yet may not be abused. We always provide immediate transitions leaving inconsistent con-



(a) ***unbound state***

Site(s_1,s_2)	Site(s_2,s_1)	interaction	delay
free	free	s_1.bind()	timed
bound	free	s_2.block()	immediate
bound	blocked	s_1.unbind()	timed
free	blocked	s_2.unblock()	immediate
free	free		

(b) ***Sample trajectory***

Figure 6.5: Overlapping sites at s_1 and s_2

figurations. Their execution precedes any time-consuming transitions that might abuse it. The higher priority of immediate transitions is enforced by rule (SUM) of the stochastic semantics of our π-calculus in Table 4.5.

One possible state trajectory is given in Figure 6.5(b). Initially, we assume a parallel composition of each two Site_free and Visitor_free. The first parameter of a Site refers to its identity, and the second to its peer's:

```
Site_free(s₁,s₂) | Site_free(s₂,s₁) | Visitor_free |
  Visitor_free
```

The first reduction is an application s_1.bind() which we assume time consuming. With some finite rate r we enter an inconsistent configuration:

$$\xrightarrow{r} \texttt{Site\_bound}(s_1,s_2) \mid \texttt{Site\_free}(s_2,s_1) \mid \texttt{Visitor\_free} \mid \texttt{Visitor\_bound}(s_1)$$

Now comes the trick. Site_free(s_2,s_1) has a choice between a time consuming transition through function s_2.bind() (with the Visitor_free), and an immediate one by function s_2.block() for which Site_bound(s_1,s_2) acts as counterpart. The priority given to immediate transitions in our semantics ensures that only the latter function is applied:

$$\xrightarrow{\infty} \texttt{Site\_bound}(s_1,s_2) \mid \texttt{Site\_blocked}(s_2,s_1) \mid \texttt{Visitor\_bound}(s_1) \mid \texttt{Visitor\_free}$$

Thus, it is impossible to enter an erroneous configuration in which both Sites are bound at the same time:

$$\overset{r}{\not\rightarrow} \texttt{Site\_bound}(s_1,s_2) \mid \texttt{Site\_bound}(s_2,s_1) \mid \texttt{Visitor\_bound}(s_1) \mid \texttt{Visitor\_bound}(s_2)$$

An implementation of Site objects with these three profiles is listed in Figure 6.6. The first two profiles free and bound are analogous to the previous Semaphore's. The third profile blocked accounts for mutual exclusion of site inhibition. To complete the example, we provide an implementation of visitors in Figure 6.7.

```
module 'overlapping sites'
export
   Site with bind/0,unbind/0,block/0,unblock/0
define
   Site(me,other) ≜ Site_free(me,other)

   Site_free(me,other) ≜
      me?bind().Site_bound(me,other) // timed
    + me?block().Site_blocked(me,other) // immediate
    + other!unblock().Site_free(me,other) // immediate

   Site_bound(me,other) ≜
      me?unbind().Site_free(me,other) // timed
    + other!block().Site_bound(me,other) // immediate

   Site_blocked(me,other) ≜
      me?unblock().Site_free(me,other) // immediate
```

Figure 6.6: Module 'overlapping sites'

```
module 'visitors of overlapping sites'
public s1 s2
export
   Visitor
define
   Visitor() ≜ Visitor_free()

   Visitor_free()≜
      s1!bind().Visitor_bound(s1)
    + s2!bind().Visitor_bound(s2)

   Visitor_bound(site) ≜ site!unbind().Visitor_free()
```

Figure 6.7: Visitor module

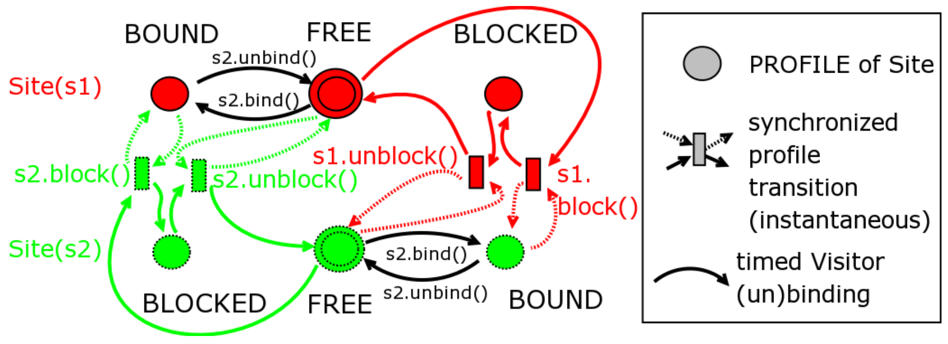


Figure 6.8: Transitions of overlapping sites at s_1 and s_2 in Petri net style

```
module 'timer(f₁,...,fₙ)'
export
    Timer
define
    Timer(c) ≜
        c!f₁().Timer(c)
      + ...
      + c!fₙ().Timer(c)
```

Figure 6.9: Module 'timer(f_1,...,f_n)'

Figure 6.8 illustrates the synchronized state transitions of the pair of objects Site(s_1,s_2) and Site(s_2,s_1). Colored transitions are due to calls of immediate functions block and unblock, and pass through the boxed synchronization points. Black edges denote timed interaction with the Visitor, not included in the illustration for the sake of clarity.

6.4 Timers

Timers proposed by Regev (2002) catalyze activities involving a single entity. When such are modeled in the π-calculus, due to its binary interaction paradigm these reactions nevertheless require some partner. An elementary timer on channel c sends the void message to c, whenever it is requested. We will use a somewhat more powerful Timers for multi-profile objects, instantiated at channel c. Such timers defined in Figure 6.9 provide co-actions f_i requested by alternative profiles. A first significant use case for these Timer processes, that can trigger actions with different rates depending on the function name, is shown in the following. The construction will also

```
module 'elementary timer'
export Timer
define
    Timer(c) ≜ c!().Timer(c)
```

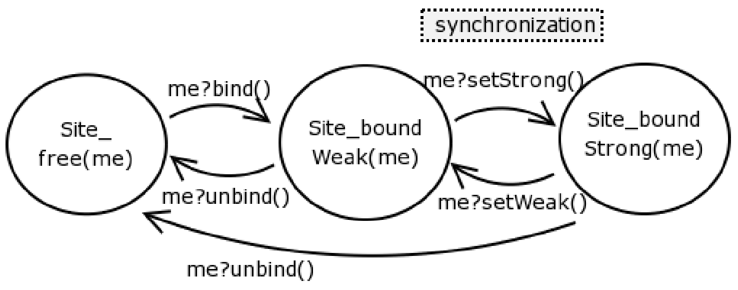


Figure 6.10: Transitions of Site with unbind function of mutable rate

demonstrate its versatility when modeling transcription regulation at the λ switch in Chapter 8.

6.5 Mutable rates

It is sometimes wishful to modify the stochastic rate of an object's function f. Achieving this is however not evident, since the function rates are determined upon creation of the object's identifier channel, by means of the associated ρ.

The alternative to rate modification would be to introduce a new function f', performing the same activity as the original f yet with a different rate. This would come at the expense of affecting the object's interface, require any interaction partners to know when to invocate which among f and f'. We favor as solution preserving the object's interface.

We illustrate our technique at the example of a Site, where as usual visitors may come and go. While binding itself occurs at a fixed rate, the stability of the bound state is mutable, i.e the response rate to calls to function unbind depends on some external condition. Figure 6.10 illustrates. Both Site_boundWeak and Site_boundStrong offer the unbind. Site is synchronized with an external partner, that via calls of its functions setStrong and setWeak may enforce profile adjustments. The trick is now simple: while unbind itself is immediate, its externally experienced rate is determined by timed internal

```
module 'Site with mutable unbinding rate'
import
    Timer from 'timer(weakTimer,strongTimer)//Fig.6.9
export
    Site  with bind/1,unbind/0,setWeak/0,setStrong/0
define
    Site(me) ≜ Site_free(me) | Timer(me)

    Site_free(me) ≜ me?bind(c).c!(me).Site_boundWeak(me)

    Site_boundWeak(me) ≜
      me?weakTimer().me?unbind().Site_free(me)
    + me?setStrong().Site_boundStrong(me)

    Site_boundStrong(me) ≜
      me?strongTimer().me?unbind().Site_free(me)
    + me?setWeak().Site_boundWeak(me)
```

Figure 6.11: Site with unbind function of mutable rate

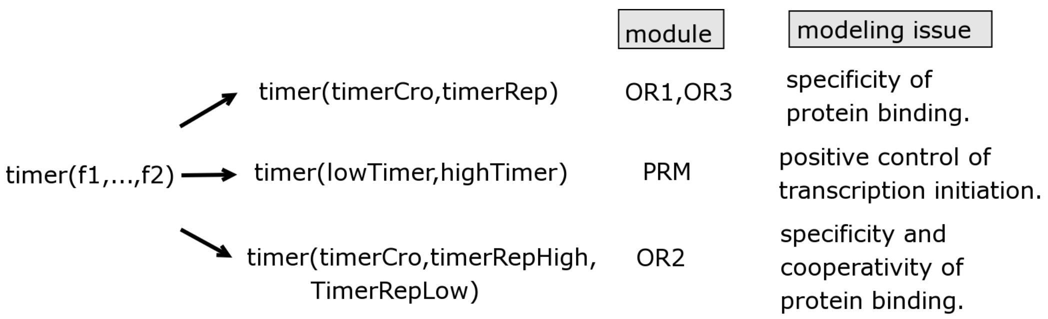

module
modeling issue
timer(f1,...,f2)
timer(timerCro,timerRep)
OR1,OR3
specificity of protein binding.
timer(lowTimer,highTimer)
PRM
positive control of transcription initiation.
timer(timerCro,timerRepHigh, TimerRepLow)
OR2
specificity and cooperativity of protein binding.

Figure 6.12: Module dependencies when using 'timer(f1,...,fn)'

communication with a tailored Timer.

The complete definition is given in Figure 6.11. We obtain the desired temporal behavior with a suitable choice of timed versus immediate function rates, such as:

$$\text{bind} \mapsto 1,\ \text{strongTimer} \mapsto 1,\ \text{weakTimer} \mapsto 10,$$
$$\text{setStrong} \mapsto \infty,\ \text{setWeak} \mapsto \infty\ \text{unbind} \mapsto \infty.$$

Notice that the mechanism for switching between Site_boundWeak and Site_boundStrong depends on immediate functions, and is reminiscent of that for overlapping sites from Section 6.3.

In our biological modeling studies, the combination of Timers with immediate profile synchronizations from Section 6.3 will prove to deal with: *specificity* between proteins and DNA operator sites, *cooperative* binding of proteins on distinct DNA operator sites, and *positive control* of transcription initiation. Figure 6.12 illustrates the module dependencies, when dealing with specific biological modeling issues.

Part III

Modeling and Simulation Studies

CHAPTER 7

Transcription and Translation

This chapter is dedicated to our models of bacterial transcription and translation in the stochastic π-calculus with pattern input, and stochastic simulation based on these. Our design of biological model components benefits from the toolset introduced in the previous chapters. Objects with multiple profiles are essential to our modeling. The re-use of functionalities from existing components, i.e. object extension, greatly facilitates the development of our biological abstractions.

Abstracting macromolecules. In Section 7.1 we develop abstractions of Rnap and Dna. To deal with Dna, we introduce three different multi-profile objects sketched in Figure 7.1: Promoter, Nucleotide, and Terminator. Two among these objects provide functionalities ascribed sequences as a whole – promoters and terminators. In contrast, we represent individual coding nucleotides, again as objects. These abstraction levels are distinguished by

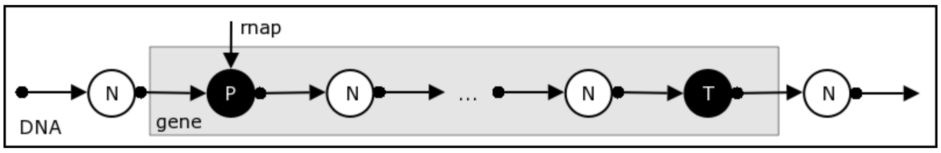


Figure 7.1: The DNA Module defines the objects Promoter, Nucleotide, Terminator, here abbreviated as P, N, and T respectively

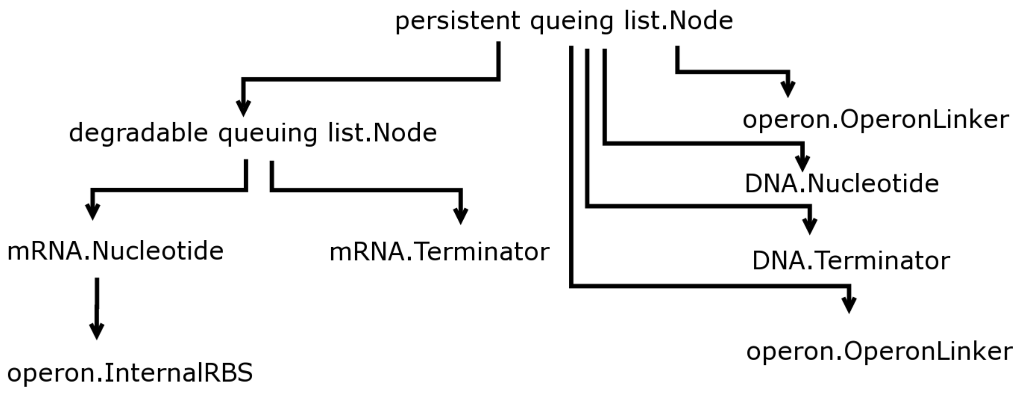


Figure 7.2: Module dependencies and inheritance

coloring in Figure 7.1. Our switching between abstraction levels follows the reaction-based representation in Table 2.1, which as previously discussed (see Section 2.5) constitutes the core of related stochastic models of gene expression. Our abstraction of mRna in Section 7.2 follows similar rough lines.

Extending on lists. Figure 7.2 overviews how in this chaper, we extend the available list variants: we base models of Dna on persistent queing lists, and those of mRna on their degradable extension. For promoters, we initially present a stand-alone abstraction. It is later refined to comprise the functionalities of a Dna nucleotide representative. As such it may smoothly combine with a predecessor Nucleotide, as illustrated in Figure 7.1. To certain biological cases, this aspect matters, while for the simulation of others it may be disregarded.

Chapter outline. After introducing our generic models of Dna and mRna, we provide suitable ρ definitions for the stochastic *parameterization* of our objects in Section 7.3. To illustrate how our models can accommodate further biological information, in Section 7.4 we *refine* our genetic library to special cases of transcriptional organization in bacteria, as introduced in Section 2.3. To keep the balance, in Section 7.5 we discuss *open challenges* to the coverage of further regulatory mechanisms within our framework. Last but not least, we move on to the *simulation* of the combined dynamics of bacterial transcription and translation in Section 7.6. We namely investigate the effect of *translational bursting*.

reaction in Tab. 2.1	**Rnap** in profile	line in Fig 7.4	sends	*to* **DNA** representative	line in Fig.7.6
				Promoter	
(2.2.1)	_free	7	bind	_free	15
(2.2.2)	_closed	9	unbind	_closed	17
(2.2.3)	_closed	10	initiate	_closed	18
(2.2.4)	_open	12	startTranscript	_open	21
(2.2.5)	_elongating	17	elongate	**Nucleotide** _bound	27
(2.2.6)	_elongating	16	terminate	**Terminator** _bound	32

Figure 7.3: Relating the reaction-based view of transcription to interactions between π-calculus objects

7.1 DNA and transcription

Figure 7.3 summarizes our mapping of the reaction-based view from Table 2.1 on communication between concurrent π-calculus objects: interactions between Rnap and promoter sequences on Dna during initiation as sketched in reactions (2.2.1) to (2.2.4), transcript *elongation* in steps of one Dna nucleotide from reaction (2.2.5), and *termination* on terminator Dna sequences, i.e. reaction (2.2.6).

Let us precede the discussion of the components with a comment on a central *design choice*. In our model, we want to explicit the growth of the transcript, that is not represented in the reaction-based view. Recall that Rna is assembled *on* Dna *by* Rnap. In the π-calculus model, the transcript representative must be spawned by *one* among these. We attribute the task to the Dna representative, which by its sequence determines the information content of the transcript. Due to this choice, our representative of Rnap is simpler to explain than those of Dna.

Abstraction of RNAP. Rnap has four profiles introduced in Figure 7.4: free, closed, open, transcribing. The first 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, i.e. binding of Rnap to some promoter as summarized by reaction (2.2.1): Rnap_free invocates bind over the global

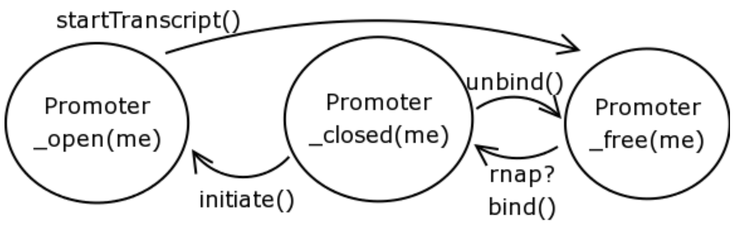
```
 1  module 'RNAP'
 2  public rnap
 3  export Rnap
 4  define
 5   Rnap ≜ Rnap_free
 6   Rnap_free ≜
 7      new c.rnap!bind(c).c?(prom).Rnap_closed(prom)
 8   Rnap_closed(prom) ≜
 9      prom!unbind().Rnap_free
10    + prom!initiate().Rnap_open(prom)
11   Rnap_open(prom)≜ new c1.new c2.
12      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

Figure 7.4: Rnap module

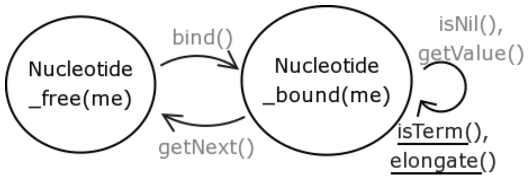
channel rnap (line 7).[1] It waits for satisfaction by a Promoter, that is nondeterministically selected among several available in profile free, and extrudes its me channel. As the bind interaction succeeds, Rnap and Promoter switch to their closed profiles. They now jointly represent the closed promoter 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_{off} in reaction (2.2.2) and k_{init} in reaction (2.2.3), which enter the model over the ρ function that quantifies the Promoter's channel me. Unbinding without transcription initiation is straightforward. It causes transitions from closed to free (line 9). If conversely initiate succeeds, both switch to their open profile (line 10). Transcription subsequently launches upon a startTranscript call, reflecting reaction (2.2.4). 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 (line 13).

Rnap_elongating traverses the Dna representative, calling elongate on each

[1]See Section 6.1 for establishment of a private connection over a global channel in a many-to-many setting.



(a) ***Transition diagram:*** **Promoter**



(b) ***Transition diagram:*** **Nucleotide*****. The functions with shaded labels are inherited from a persistent queuing list*** **Node*****, the underlined are new.***

Figure 7.5: DNA abstraction: profile transitions

Nucleotide over its extruded me channelo, see line 17. This accounts for reaction (2.2.5). Recall that we left it to the abstraction of DNA to actually create that of mRNA, upon these calls of function elongate. After reaching the Terminator, Rnap returns to free (line 16), while the Terminator actually completes and releases the transcript. This behavior corresponds to reaction (2.2.6). Notice that the recycling of Rnap objects differs from the behaviour of the previous Walker, that terminates as 0 reaching the end of a list.

Abstraction of DNA. Our module 'DNA' in Figure 7.6 provides the complementary actions to Rnap. Promoter implements the access control to genes, with three profiles analogous to those of Rnap: Promoter_free is dual to Rnap_free, and so forth. Transcription ensues once both Rnap and Promoter are open. As a result of a startTranscript call (line 12), Promoter_open spawns a RBS abstracting the first chunk of the transcript, and returns the channel rna pointing to its growing end to Rnap_open. The transition to Promoter_free is caused by Rnap_open's following call to getNext.

Both Nucleotide and Terminator extend on the persistent queuing Node. Rnap distinguishes them via isTerm queries, determining whether elongation should cease: Nucleotide returns false, whereas Terminator returns true. When calling elongate, Rnap sends a reference rna to the transcript's growing end.

```
module 'DNA('your favorite protein')'
public
   rnap with bind/1
import
 List(Node,Nil) from 'persistent queuing list'// Fig 5.8
 Mrna(Nucleotide,Terminator,RBS) from 'mRNA('your
     favorite protein')'// Fig 7.8
rate $\rho_{rna}$, $\rho_{rbs}$ from 'Table 7.1'
export
  Promoter with  unbind/0,initiate/0,startTranscript/1
  Nucleotide extends List.Node by isTerm/1,elongate/2
  Terminator extends List.Node by isTerm/1,terminate/1
define
  Promoter(me,next) ≜ Promoter_free(me,next)
  Promoter_free(me,next)  ≜
     rnap?bind(c).c!(me).Promoter_closed(me,next)
  Promoter_closed(me,next) ≜
     me?unbind().Promoter_free(me,next)
   + me?initiate().Promoter_open(me,next)
  Promoter_open(me,next)  ≜
     new me$_2$:$\rho_{rbs}$.new rna:$\rho_{rna}$.
     me?startTranscript(c$_1$).c$_1$!(rna).
     me?getNext(c$_2$).next!bind().c$_2$!(next).
        Promoter_free(me,next)| Mrna.RBS(me$_2$,rna)

  Nucleotide_bound(me,next,v)  extended by
     me?isTerm(c).c!false().Nucleotide_bound(me,next,v)
   + me?elongate(rna,c).new rna_nxt:$\rho_{rna}$. c!(rna_nxt).
     Nucleotide_bound(me,next,v)
     | Mrna.Nucleotide(rna,rna_nxt,v')
  Terminator_bound(me,next,v)  extended by
     me?isTerm(c).c!true().Terminator_bound(me,next,v)
   + me?terminate(rna).new last:$\rho_{rna}$.
     Terminator_free(me,next,v)
     | Mrna.Terminator(rna,last,v') | List.Nil(last)
```

Figure 7.6: DNA module

The DNA Nucleotide appends a Nucleotide of complementary content (for simplification indicated by v') imported from the 'mRNA' module. It returns the new growing extremity rna_nxt to Rnap. The hybrid Terminator completes the nascent transcript with an mRNA.Terminator, followed by a Nil.

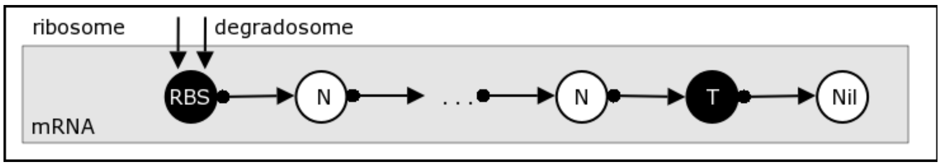


Figure 7.7: mRNA abstraction comprising processes RBS, Nucleotide, and Terminator. The latter two (here N and T) are based on the module 'degradable queuing list'. Note that the mRNA representative actually ends in a Nil object.

7.2 mRNA, translation and degradation

We base our model of mRNA on the module 'degradable queuing list' (Figure 5.9 on page 94). This accounts for two biologically relevant properties via object inheritance: the molecule's *unstable* character, and the *queuing* of 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 Figure 7.7. Different than in DNA processing, we do not explicity track the growth of the nascent transript while the abstraction of the translating ribosome proceeds along mRNA.

RBS. The two-profile RBS defined in Figure 7.8 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 successor (that it sends to profile bound) it becomes the inert process **0** in line 15. If alternatively a Ribosome binds over the global channel ribosome, it causes a switch from RBS_free to RBS_bound (line 16). Similarly to the unstable intermediate on promoters, two interactions unbind (line 19) and initiate (line 18) become possible, reflecting reactions (2.2.8) and (2.2.9) in Table 2.2. Their quantitative behavior is fixed as usual by the ρ function associated with the RBS's me channel. We will give sample values in the following section.

Ribosome. In discussing translation elongation, we first consider the ribosome representative specified in Figure 7.9. Similarly as Rnap, it advances over individual Nucleotides of mRNA, while calling their elongate function in line 16. This accounts for reaction (2.2.10) in Table 2.2. Notice that elongate on mRNA takes no parameter, in contrast to that of DNA propagating the

```
 1 module 'mRNA('your favorite protein')'
 2 public
 3    ribosome with bind/1
 4    degradosome with bind/1
 5 import
 6   List(Node,Nil) from 'degradable queuing list'//Fig5.9
 7   Protein from 'your favorite protein'
 8 export
 9    RBS with init/1,unbind/0
10    Nucleotide extends List.Node by isTerm/1,elongate/0
11    Terminator extends List.Node by isTerm/1,terminate/0
12 define
13    RBS(me,next) ≜ RBS_free(me,next)
14    RBS_free(me,next) ≜
15       degradosome?bind(c).next!bind().c!(next).0
16     + ribosome?bind(c).c!(me).RBS_bound(me,next)
17    RBS_bound(me,next) ≜
18       me?init(c).next!bind().c!(next).RBS_free(me,next)

19     + me?unbind().RBS_free(me,next)
20
21    Nucleotide_bound(me,next,val) extended by
22       me?isTerm(c).c!false().Nucleotide_bound(me,next,v)
23     + me?elongate().Nucleotide_bound(me,next,v)
24
25    Terminator_bound(me,next,val) extended by
26       me?isTerm(c).c!true().Terminator_bound(me)
27     + me?terminate().
28         Terminator_free(me,next,val) | Protein
```

Figure 7.8: mRNA module

reference to the transcript's growing end. That is because our abstraction of translation disregards the actual assembly of a sequence of amino acids (which however may be covered by further refinement).

Reaching the Terminator the full protein representative is spawn via function terminate called by Ribsome_elongating in line 14. This corresponds to reaction (2.2.11).

Nucleotide and Terminator. Nucleotide objects propagate both translation and degradation, where the latter occurs via the function kill inherited

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

Figure 7.9: Ribosome module

from a degradable queuing nodes. The Ribosome uses the inherited function getNext to step through the mRNA representative in translation; it stops on the Terminator due to its positive response to isTerm. The function elongate (line 27) that Ribosome calls remains without effect in this basic Nucleotide model.

Degradosome. The degradosome specification in Figure 7.10 is straightforward: after gaining access to the RBS, it stepwise destructs the whole mRNA, calling getNext (line 12) and kill (line 13) on each of its Nucleotides. Degradation stops at the end of the transcript (line 11) - note this occurs on Nil, *not* on the Terminator as does translation, by using functions isNil rather than isTerm. This distinction will prove useful when dealing with polycistronic mRNA. Another point worthwhile stating is that the propagation of decay is not covered by the reaction based view in Table 2.2.

Proteins. Proteins have many functions in the cell, that are not covered in this chapter. 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:

```
module 'degradosome'
public degradosome
export Degradosome
define
 Degradosome() ≜ Degradosome_free()
 Degradosome_free() ≜ new c.
     degradosome!bind(c).c?(rna).Degradosome_working(rna)

 Degradosome_working(rna) ≜
     new b.rna!isNil(b).
        b?true()  .rna!kill().Degradosome_free
     + b?false().new c.rna!getNext(c).c?(next).
           rna!kill().Degradosome_working(next)
```

Figure 7.10: Degradosome module

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

Table 7.1: ρ definitions for transcription, translation and mRNA decay

```
Protein ≜ prot?kill().0
```

```
module 'operon'
public ribosome with bind/1
rate ρrna from 'Table 7.1'
import
 List(Node) from 'persistent queuing list' // Fig.5.8
 Mrna(Nucleotide,Terminator,RBS) from 'mRNA' // Fig.7.8
export
 OperonLinker extends List.Node  by isTerm/1,elongate/2
 InternalRBS  extends Mrna.Nucleotide
define
 OperonLinker_free(me,next,v) ≜ List.Node(me,next,v)
 OperonLinker_bound(me,next,v)  extended by
    me?isTerm(c).c!false().OperonLinker(me,next,v)
  + me?elongate(rna,c).
        new rna_nxt:ρrna. new_rna_nxt2:ρrna.c!(rna_nxt2).
        OperonLinker_bound(me,next,v)
        | Mrna.Terminator(rna,rna_nxt)
        | InternalRBS(rna_nxt,rna_nxt2)
 InternalRBS(me,next,v) ≜ InternalRBS_free(me,next,v)
 InternalRBS_free(me,next,v)  extended by
  + ribosome?bind(c).c!(me).InternalRBS_bound(me,next,v)

 InternalRBS_bound(me,next,v)≜Mrna.RBS_bound(me,next,v)
```

Figure 7.11: Operon module

7.3 Parameterization

Table 7.1 gives sample values of the ρ function attributing rates to functions for transcription and translation. Functions not associated with a rate are instantaneous. Recall that each object is identified by its me channel, over which its function are invocated. This allows to associate method calls on different objects of the same class with distinct rates, as notably useful for initiation rates on promoters or ribosomal binding sites.

7.4 Particular promoter arrangements

We extend our components to cover particular promoter arrangements introduced in Section 2.3 on page 34.

7.4.1 Operons and polycistronic mRna

We devote a module to operons, and the polycistronic mRNA transcribed from them (Figure 7.11). It defines the processes OperonLinker and InternalRBS, that can be used in abstactions of DNA and mRNA respectively. OperonLinker is used to connect two genes that are co-transcribed within an operon. The interface it offers to Rnap_elongating is that of a regular Nucleotide on DNA, namely functions isTerm in line 13, and elongate in line 14. This second is of particular interest: it spawns a transcript that comprises a mRNA Terminator, on which translation of the first protein stops (line 17), combined with an InternalRBS (spawn in line 18). On this latter, translation of the second protein may initiate – yet not degradation. Initial binding of the Ribosome occurs as on a regular RBS, via the global channel ribosome (line 21) while decay propagation is inherited from a regular mRNA Nucleotide. Ribosome unbinding and translation initiation are inherited from a regular RBS object in profile bound.

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

```
Operon ≜
    Dna.Promoter
    $\Pi_{i=1}^{\text{length(gene 1)}}$ Dna.Nucleotide | OperonLinker |
    $\Pi_{i=1}^{\text{length(gene 2)}}$ Dna.Nucleotide | Dna.Terminator
```

The transcription of our operon yields polycistronic mRNA coding for two different proteins[2], which are translated with distinct efficiencies (depending on the ρ function of RBS's me). Notice that translation of the first cistron ceases on the first terminator, while propagation continues over it.

```
PolycistronicMrna ≜
  Mrna.RBS | $\Pi_{i=1}^{\text{length(gene 1)}}$ Mrna.Nucleotide | Mrna.
      Terminator
  | InternalRBS | $\Pi_{i=1}^{\text{length(gene 2)}}$ | Mrna.Nucleotide
  | Mrna.Terminator | Node.Nil
```

When composing the operon in parallel with the molecular machines of transcription, translation, and mRNA decay, we obtain the following reduction sequence:

```
Operon | Rnap | Ribosome | Degradosome
$\rightarrow^*$ Operon | Rnap | Ribosome | Degradosome |
   | PolycistronicMrna
$\rightarrow^*$ Operon | Rnap | Ribosome | Degradosome
```

[2] Notice the convenient parametricity of our module 'mRNA' in proteins.

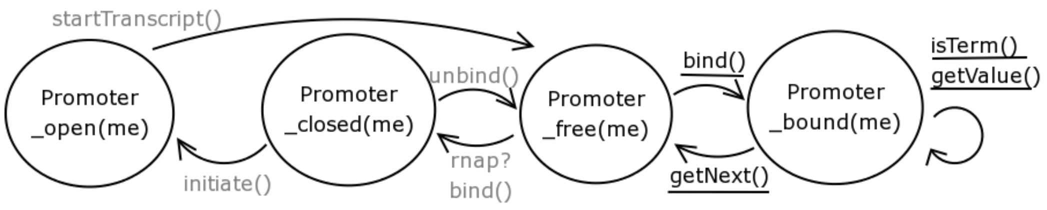


Figure 7.12: Transitions of tandem promoter. Underlined functions extend on those of the corresponding profiles of a DNA Promoter.

$$\Pi_{i=1}^{n}\ \mathsf{ProteinA}\ \mid\ \Pi_{i=1}^{m}\ \mathsf{ProteinB}$$

Eventually, the polycistronic mRNA has been transcribed, has yielded distinct numbers n and m of A and B proteins, and been degraded.

7.4.2 Tandem promoter

Let us now consider promoters arranged in a tandem, illustrated in Figure 2.6(c) on page 35. The interesting question lies in the representation of the *internal* promoter, over which transcription proceeds after it has initiated at the outer. Figure 7.13 presents the profile transitions of the tandem promoter as a stochastic π-calculus object, specified in Figure 7.13.

Promoter_free allows for binding of RNAP, and transcription initiation, as inherited from a regular promoter. The profile bound, that is not present for the previous promoter abstraction, offers RNAP_elongating the same interface as a regular Nucleotide from the 'DNA' module[3]. It is entered as RNAP_elongationg is about to leave the preceding Nucleotide, relying on the usual binding mechanism for queuing nodes: a call of function bind in line 10, triggered by RNAP's call of getNext on the predecessor nucleotide. In Promoter_bound the interesting point is the function elongate in line 15. Upon its call the transcript is appended a new element, offering translation initiation, but only propagation of decay. We previously designed an element of this functionality for transcripts resulting from operons: InternalRBS.

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 2.4(a). The model can be extended with an additional profile to

[3]It would fully do so by offering an additional function isNil, present in a persistent queing list node.

```
module 'tandem promoter'
import
   P(Promoter) from 'DNA'   // see Figure 7.6
   InternalRBS from 'operon'   // see Figure 7.11
rate ρrna from 'Table 7.1'
export
   Promoter extends P.Promoter by bind/0,isTerm/1,
      getNext/1,elongate/2
define
 Promoter_free(me,next) extended by
    + me?bind().Promoter_bound(me,next)

  Promoter_bound(me,next) ≜
     me?isTerm(c).c!false().Promoter_bound(me,next)
   + me?getNext(c).next!bind().c!next.Promoter_free(me,
      next)
   + me?elongate(rna,c).new rna_nxt:ρrna.c!(rna_nxt).
      Promoter_bound(me,next) | InternalRBS(rna,rna_nxt)
```

Figure 7.13: Tandem promoter module

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 7.6.

7.5 Discussion: challenges in modeling

The integration of more biological detail can become increasingly challenging, namely when it comes to cover aspects related to secondary structures of mRNA (intrinsic termination of transcription, transcriptional attenuation), and two-way traffic on DNA and mRNA.

Attenuation of transcription. Consider the regulatory mechanism of *transcriptional attenuation*. In this case, transcription termination is determined by the efficiency at which translation proceeds over the nascent transcript. The crucial detail is that intrinsic termination depends on more than mere recognition of a terminator sequence. Terminator sequences actually comprise two sub-sequences 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 require 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. Regarding certain cases 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 queuing 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.

7.6 Simulation of translational bursting

In this section we present simulations obtained from our model components with the BioSpi tool (Priami et al., 2001), after encoding input patterns within the stochastic π-calculus. Our focus lies on the effect of *translational bursting* introduced in page 38. 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. (2002), 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 *average* expression levels, nor can they be reproduced by one step models of gene expression as that of Blossey, Cardelli, and Phillips (2006). 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. They are taken from Ozbudak et al. (2002) as the above parameters, and the number of Rnap, ribosomes and degradosomes. We executed both combinations for 7 hours of simulated time.

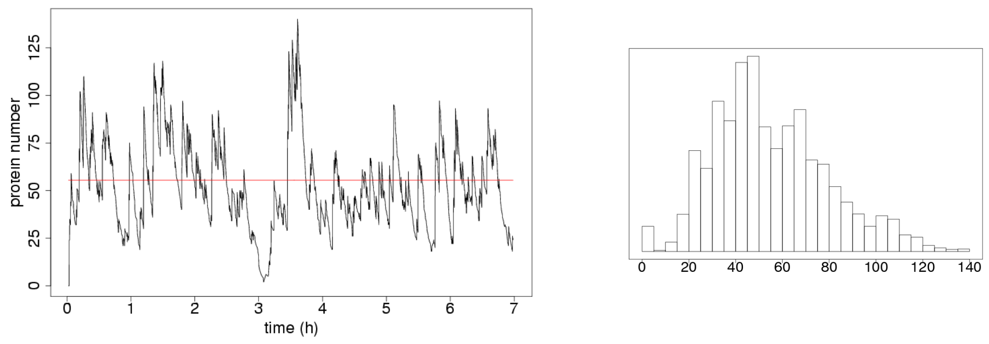
Figure 7.14 reports a representative run for each of the two settings. The left curve in Figure 7.14(a) displays the evolution of the protein level over time for an execution of the *bursty* combination. The protein level fluctuates strongly around an average of 55, marked by a horizontal line. The fourth 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 Figure 7.14(a). Here the simulated time is divided into equally long intervals. The bars indicate by their height how often a given number of proteins (labeled on the horizontal axis) is observed.

Simulations based on the *smooth* setting have a clearly distinct behavior, as shows Figure 7.14(b) where the protein level only weakly fluctuates around an average of 46[4]. The histogram confirms that the distribution is pronouncedly narrowed compared with the previous setting.

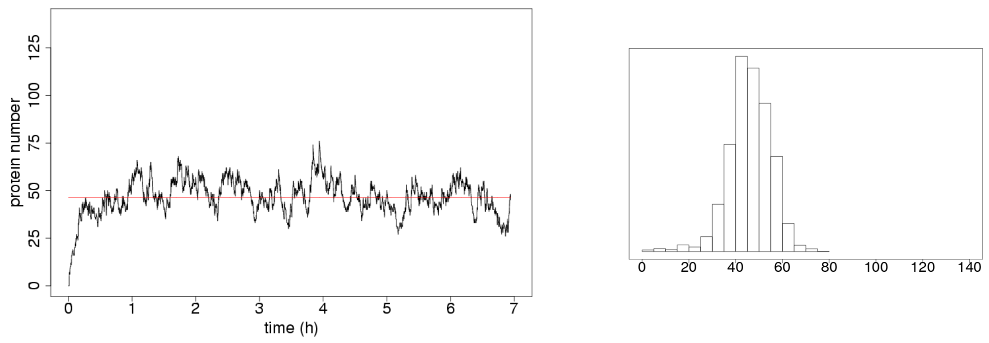
An alternative interpretation of the histograms is as *population snapshots*, with respect to the expression level of a given protein. In this view the height of the bars indicate the fraction of the population with a certain protein level. This shows how for the setting *bursty*, the variability propagates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

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

[4]When averaging over many simulation runs, both settings yield the same average protein level.



(a) ***bursty gene expression***



(b) ***smooth gene expression***

Figure 7.14: Basal expression of a single gene under parameter variation, yielding average protein crops per mRNA: (a) 10, (b) 1. Left: time course of protein numbers, right: histogram of protein number over the simulated period.

seconds (3rd column). This explains the drops observed in Figure 7.14(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):

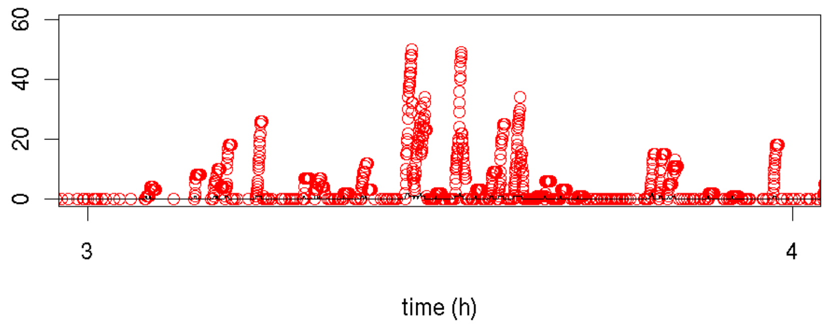


Figure 7.15: Concurrent translation of mRNA by multiple ribosomes during simulation from Figure 7.14(a).

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

Origin of translational bursting. We could not yet observe the origin of the strong bursts in protein numbers in Figure 7.14(a). This motivates further inspection of setting *bursty*'s simulation data. Figure 7.15 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 a never times exceeds two (data not shown), these are simultaneously processed by up to 50 ribosomes. As discussed page 38 the strongest bursts in protein levels occur when for a mRNA, the number of translations by far exceeds the average. The column-reminiscent peaks mark transcripts yielding exceptionally high protein crops; this is in agreement with a geometric distribution.

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 (which 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.

CHAPTER 8

Transcription Regulation at the Lambda Switch

The discussion in the previous chapter considered only *basal* gene expression. The expression of any actual gene is however subject to *regulation.* This chapter is dedicated to modeling transcription initiation control at the λ switch in our stochastic π-calculus with concurrent objects, and validation of this model through simulation with the BioSpi engine. We will cover the details of regulatory interactions described in Chapter 3, involving the two promoters P_{RM} and P_R, the three operator regions on DNA controlling them, and the regulatory proteins Cro and λ repressor. In modeling transcriptional regulation at the λ switch, we make large use of the modeling techniques from Chapter 6.

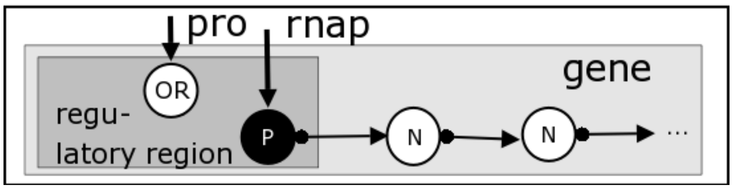


Figure 8.1: Sketch of regulatory region

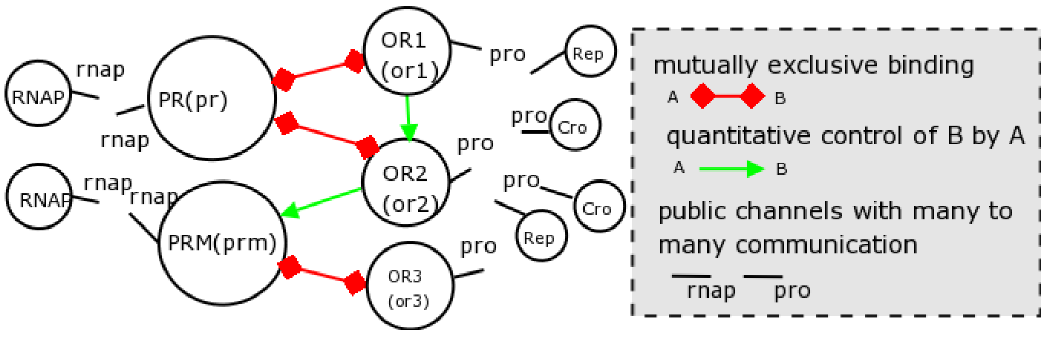

RNAP
rnap
PR(pr)
rnap
RNAP
rnap
rnap
PRM(prm)
OR1
(or1)
OR2
(or2)
OR3
(or3)
pro
pro
pro
pro
pro
Rep
Cro
Cro
Rep
mutually exclusive binding
A
B
quantitative control of B by A
A
B
public channels with many to many communication
rnap
pro

Figure 8.2: Molecular population and regulatory interactions

8.1 Overview

Before presenting the comprehensive model of the λ switch, we discuss modeling techniques for isolated aspects. Let us begin with a promoter, that is subject to control by one DNA operator region to which proteins bind. The outline of its π-calculus model in Figure 8.1, does not yet specify the actual mechanism regulating transcription initiation. *Negative control* would mean mutually exclusive access relation between the promoter and operator region. This can be covered by the mutual exclusion mechanism introduced in Section 6.3. Alternatively the gene may be *positively controlled*: RNAP's transcription initiation frequency could increase through the presence of a regulatory protein at the operator (not overlapping the promoter, but located in its vicinity). This would in a stochastic π-calculus model be reflected by adjustment of the promoter's initiate rate, based on the mechanism from Section 6.5

Population members and interaction topology. Let us now consider more detail to incorporate into our model. Figure 8.2 summarizes the λ switches' population: the operators OR1, OR2, and OR3 can be bound by the regulatory proteins Cro and Rep– in our model this occurs over the global channel pro. The promoters PRM and PR can be bound by RNAP via channel rnap. These two bindings occur in a many-to-many fashion using the mechanism from Section 6.1. Our model introduces the following *profiles*. Operator representatives can be free, rep, cro, or blocked. Promoter PR complements the profiles introduced in Section 7.1, where we did not yet consider repression, by a blocked profile. PRM additionally replaces the RNAP-bound profile closed by two others, in oder to distinguish the efficiency

	O_{R1}	O_{R2}	O_{R3}	P_R	P_{RM}
protein specific unbinding rate	✓	✓	✓		
mutual exclusion	✓	✓	✓	✓	✓
quantitative control of other	✓	✓			
quantitatively controlled by other		✓			✓

Figure 8.3: Summary: interactions and dependencies at the λ switch

of transcription initiation at high or low level, depending on OR2's binding state.

Concurrent control. We distinguishing two types of *edges* that indicate relations between operators and promoters.[1] Red edges represent *mutual exclusion* relations between population members. Each such edge yields four functions, for blocking and unblocking of the respective molecular actors. We introduce mnemonic names. For instance, the red edge between PRM and OR3 will be realized by OR3's function blockedByPrm, PRM's blockedByOr3, and two functions for the reverse actions of unblocking. All these basically rely on the mutual exclusion mechanism from Section 6.3.

Green edges indicate *quantitative control* of reactions, i.e. cooperative binding of the λ repressor protein to O_{R2} and O_{R1}, and positive control of transcription initiation frequency at P_{RM}. We here instantiate the mechanism for rate adjustment from Section 6.5, i.e. for up- and downregulating OR2's unbind rate and that of PRM's initiate. Figure 8.3 summarizes the mechanisms observed for the various members of the population. Each entry is realized as an object function in our model.

Promoters. Summarizing, the abstractions of the λ switch promoters P_R and P_{RM} extend on the generic Promoter from the module 'DNA' (see Figure 7.6) as follows. They are *repressible* by the operators overlapping them. Modeling this relies on the mutual exclusion mechanism from Section 6.3. In addition, the efficiency of transcription initiation at PRM increases as Repressor is bound to OR2. For this, we instantiate the technique introduced in Section 6.5. Notice that we could, in principle, obtain the model of PR by extension from the generic one, since all necessary operations represent additional input patterns in the respective profiles.[2] Regarding PRM, we would need to rewrite some more, in order to distinguish the transcription

[1]Each of these edges replaces an *update channel* in the topology of our earlier stochastic π-calculus model (Kuttler and Niehren, 2006).

[2]The only real obstacle would be to separately fix the ρ of the RBS representative.

```
1  Protein_free ≜
2     new c.pro!bind(c).c?(or).Protein_bound(or)
3  Protein_bound(or) ≜ or!unbind().Protein_free
4
5  OR_free(me) ≜ pro?bind(c).c!(me).OR_bound(me)
6  OR_bound(me) ≜ me?unbind().OR_free(me)
```

Figure 8.4: Many-to-many docking of proteins to DNA operator sites.

initiation levels. For the sake of presentation, in this document we however spell out the complete 'PRM' and 'PR' modules.

Operator regions. Throughout this chapter we will incrementally refine models of *operator regions* on DNA to include all required aspects. Let us start with the minimal version, that is analogous to the previous Site from Figure 6.2 (page 98). The OR defined in Figure 8.4 offers many-to-many binding to proteins over a global channel pro while in profile free, and subsequent communication over its private channel while bound.

Let us relate this π-calculus model to the chemical reaction in Table 3.1, page 47. The communication on channel pro corresponds to the forward direction in reaction (3.1.1). The offer pro?bind made by the process OR_free (line 5) accounts for the participation of molecule OR, that of process Protein_free in line 2 to that of the transcription factor TF.

Upon binding, Protein obtains the OR's identifier channel, over which it later invocates function unbind (lines 3 and 6)

Specificity of operator protein pairs. Proteins specifically attach to DNA, i.e. with a stability reflecting their match with the operator region's sequence: the same protein can bind more or less stably to different regions; while at the same site, the binding of different proteins can vary in strength. We introduced a technique to modify a function's apparent rate in Section 6.5. Here we use the same strategy to build an abstraction of an operator region offering a uniform unbind function to any protein attached, while adjusting its *rate* to the actual protein. The operator's profile name explicitly reflects this information.

Figure 8.5 lists this module 'OR with specificity'. In order to select the appropriate unbinding rate, the site identifies the protein upon binding, see lines 10–11. We therefore introduce an explicit profile OR_switching, exposing two functions specCro and specRep. These are requested by the proteins Cro

```
module 'OR with specificity'
public pro with bind/1
export
  OR with unbind/0,specCro/0,specRep/0
  Timer
define
    OR(me) ≜ OR_free(me) | Timer(me)
    OR_free(me) ≜ pro?bind(c).c!(me).OR_switching(me)
    OR_switching(me) ≜
        me?specCro().OR_cro(me)
     + me?specRep().OR_rep(me)
    OR_cro(me) ≜ me?timerCro().me?unbind().OR_free(me)
    OR_rep(me) ≜ me?timerRep().me?unbind().OR_free(me)
    Timer(c)≜
        c!timerCro().Timer(c)
     + c!timerRep().Timer(c)
```

Figure 8.5: Operator adjusting its unbinding rate to proteins Cro and Rep

and Rep binding the operator, thus identifying themselves and causing the operator's continuation in the appropriate bound profile.[3]

Repressor protein. Figure 8.6 specifies the abstraction of the λ repressor protein, that as a *dimer* regulates the λ switch through binding to the sites O_{R1}, O_{R2}, and O_{R3}. Rep_dimer binds DNA through many-to-many communication over pro, hereby identifying itself to operators by calling the function specRep. For completeness the module also includes dimerization of the monomeric form. We omit a statement in the module that would be used to create and export a population Rep_monomers of a given size, together with one timer providing co-actions on lam_undimerize.

Our following discussion of the λ switch constituents proceeds from the simpler O_{R3} (with only a mutual exclusion edge in the above population diagram), to the other operator regions and promoters with refined control

[3]A note on the possibility of *unspecific* protein binding. Our Operator_switching could additionally offer a function unspecific, and proteins could request this upon binding, as an alternative to their respective specific functions. The problem is that such a solution does not yet combine with our semantics, in which alternative (instantaneous) choices are resolved non-deterministically. It would however work if choices between instantaneous alternatives were worked through sequentially, until the first match is found. An alternative would consist in an explicit conditional statement with *otherwise*, as offered by the BioSpi tool.

```
module 'lambda repressor'
public pro,lam_undimerize,lam_dimerize
export Rep_monomer
define
    Rep_monomer ≜
        lam_dimerize!().Rep_dimer
     + lam_dimerize?().0
    Rep_dimer ≜ new c.
        pro!bind(c).c?(or).or!specRep().Rep_bound(or)
     + lam_undimerize?().Rep_monomer | Rep_monomer
    Rep_bound(or) ≜ or!unbind().Rep_dimer
```

Figure 8.6: Module 'lambda repressor'

mechanism (see their increasing connective).

Parameterization. Figure 8.7 gives an overview of the functions offered by our objects (identified by the channels of corresponding names), together with their parameterization which is based on Table 3.2 on page 48. We would like to emphasize the crucial importance of the functions with infinite stochastic rates, which are executed without advance of the simulation clock.

8.2 Model components

We now specify the abstractions of our three operator sites and two promoters at the λ switch.

8.2.1 Abstraction of O_{R3}

We consider the abstraction of the site O_{R3}, that overlaps with the promoter P_{RM} and thus determines repression of the cI gene. It is the least complicated component of the λ switch, as indicated by the smaller number of edges interconnecting it, in the population overview of Figure 8.2. It nonetheless integrates the mechanism of specific binding for the proteins Rep and Cro.

Protein binding via the global channel pro occurs in the usual many-to-many manner, introduced in Section 6.1. Identification of the protein type occurs on the fly in lines 10-11, as introduced by the generic operator in Figure 8.5. Profile OR_free offers two other interactions: Line 12 accepts pattern input matching blockedByPrm, this corresponds to *become occluded* and is performed by PRM residing in some bound profile. As a consequence, PR

channel	ρ_{channel}(function)	
rnap	$\rho_{\text{rnap}}(\text{bind})=0.098$	
prm	$\rho_{\text{prm}}(\text{initiate})=\infty$	$\rho_{\text{prm}}(\text{unbind})=0.788$
	$\rho_{\text{prm}}(\text{highTimer})=0.086$	$\rho_{\text{prm}}(\text{lowTimer})=0.005$
	$\rho_{\text{prm}}(\text{getNext})=30$	$\rho_{\text{prm}}(\text{startTranscript})=30$
	$\rho_{\text{prm}}(\text{freedByOr3})=\infty$	$\rho_{\text{prm}}(\text{blockedByOr3})=\infty$
	$\rho_{\text{prm}}(\text{raisedByOr2})=\infty$	$\rho_{\text{prm}}(\text{resetByOr2})=\infty$
pr	$\rho_{\text{pr}}(\text{initiate})=0.05$	$\rho_{\text{pr}}(\text{unbind})=0.155$
	$\rho_{\text{pr}}(\text{freedByOr2})=\infty$	$\rho_{\text{pr}}(\text{blockedByOr2})=\infty$
	$\rho_{\text{pr}}(\text{freedByOr1})=\infty$	$\rho_{\text{pr}}(\text{blockedByOr1})=\infty$
	$\rho_{\text{pr}}(\text{getNext})=30$	$\rho_{\text{pr}}(\text{startTranscript})=30$
pro	$\rho_{\text{pro}}(\text{bind})=0.098$	
or1	$\rho_{\text{or1}}(\text{unbind})=\infty$	
	$\rho_{\text{or1}}(\text{timerRep})=0.155$	$\rho_{\text{or1}}(\text{timerCro})=2.45$
	$\rho_{\text{or1}}(\text{specRep})=\infty$	$\rho_{\text{or1}}(\text{specCro})=\infty$
	$\rho_{\text{or1}}(\text{freedByPr})=\infty$	$\rho_{\text{or1}}(\text{blockedByPr})=\infty$
or2	$\rho_{\text{or2}}(\text{unbind})=\infty$	$\rho_{\text{or2}}(\text{timerRepHigh})=0.155$
	$\rho_{\text{or2}}(\text{timerRepLow})=3.99$	$\rho_{\text{or2}}(\text{timerCro})=2.45$
	$\rho_{\text{or2}}(\text{specRep})=\infty$	$\rho_{\text{or2}}(\text{specCro})=\infty$
	$\rho_{\text{or2}}(\text{freedByPr})=\infty$	$\rho_{\text{or2}}(\text{blockedByPr})=\infty$
	$\rho_{\text{or2}}(\text{raisedByOr1})=\infty$	$\rho_{\text{or2}}(\text{resetByOr1})=\infty$
or3	$\rho_{\text{or3}}(\text{unbind})=\infty$	
	$\rho_{\text{or3}}(\text{timerRep})=20.22$	$\rho_{\text{or3}}(\text{timerCro})=0.29$
	$\rho_{\text{or3}}(\text{specRep})=\infty$	$\rho_{\text{or3}}(\text{specCro})=\infty$
	$\rho_{\text{or3}}(\text{freedByPrm})=\infty$	$\rho_{\text{or3}}(\text{blockedByPrm})=\infty$
lam_dimerize	$\rho_{\text{lam\_dimerize}}=0.048$	
lam_undimerize	$\rho_{\text{lam\_undimerize}}=0.5$	

Figure 8.7: Public channels of the λ switch model and their ρ definitions

switches to its blocked profile. In line 13, OR3_free releases a block on PRM by calling its freeByOr3. This is performed instantaneously upon *becoming* free. This mutual exclusion with P_{RM} relies on the technique introduced in the module 'Overlapping Space' from Section 6.3. Note that loops of free/block calls without advance of the simulation are excluded, under the assumption of an appropriate stochastic parameterization. In such, all communications are instantaneous, except for calls of bind over the global channel pro, and unbind over the operator's identifier.

```
module 'OR3'
public pro with bind/1, or3 as me, prm
import
   Timer from 'timer(timerCro,timerRep)' // see Fig.6.9
export OR3 with unbind/0,specCro/0,specRep/0,
   blockedbyPrm/0,freedByPrm/0
define
   OR3 ≜ OR3_free | Timer(me)
   OR3_free ≜
      pro ?bind(c).c!(me).
         me?specCro().OR3_cro
       + me?specRep().OR3_rep
    + me?blockedByPrm().OR3_blocked
    + prm!freedByOr3().OR3_free
   OR3_rep ≜
      me?timerRep().me?unbind().OR3_free
    + prm!blockedByOr3().OR3_rep
   OR3_cro ≜
      me?timerCro().me?unbind().OR3_free
    + prm!blockedByOr3().OR3_cro
   OR3_blocked ≜
      me?freedByPrm().OR3_free
```

Figure 8.8: O_{R3} module

8.2.2 Abstraction of O_{R1}

O_{R1} exhibits only a single feature not yet present at O_{R3}, beyond mutual exclusion with P_R and specific protein binding. It quantitatively controls the binding rate of the λ repressor protein at O_{R2}, as symbolized by a green edge in Figure 8.2. In order to reflect this, our model OR1_rep calls the function raisedByOr1 on OR2, see line 20 in Figure 8.9. The reverse reaction occurs once the repressor protein has left, see OR1_free's call of resetByOr1 on OR2 in line 13. We discuss OR2's responses to these function calls in the following section.

8.2.3 Abstraction of O_{R2}

O_{R2} is the most demanding λ switch component from the modeling point of view, since it combines all control aspects as reported in Figure 8.3. We summarize the state transition of OR2 in Figure 8.10 and list its specification in Figure 8.11.

```
 1  module 'OR1'
 2  public pro with bind/1, pr, or2, or1 as me
 3  import
 4     Timer from 'timer(timerCro,timerRep)' //Fig.6.9
 5  export OR1 with unbind/0,specCro/0,specRep/0,
 6     freedByPr/0, blockedByPr/0
 7  define
 8     OR1 ≜ OR1_free | Timer(me)
 9     OR1_free ≜
10        pro?bind(c).c!(me).
11            me?specCro().OR1_cro
12          + me?specRep().OR1_rep
13      + or2!resetByOr1().OR1_free
14      + me?blockedByPr().OR1_blocked
15     OR1_cro ≜
16        me?timerCro().me?unbind().pr!freedByOr1().OR1_free
17      + pr!blockedByOr1().OR1_cro
18     OR1_rep ≜
19        me?timerRep().me?unbind(). pr!freedByOr1().
             OR1_free
20      + or2!raisedByOr1().OR1_rep
21      + pr!blockedByOr1().OR1_rep
22     OR1_blocked ≜
23        me?freedByPr().OR1_free
```

Figure 8.9: $\mathrm{O_{R1}}$ module

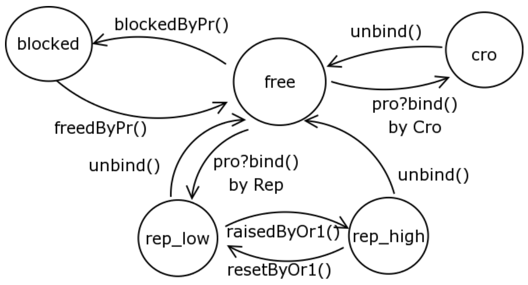


Figure 8.10: Transitions of OR2

Specific protein binding. Docking of regulatory proteins occurs in the usual fashion in profile free, line 12. Note that all three protein-bound profiles offer the unbind function. The timing of this latter relies on the complementary action of a tailored Timer, created from module 'timer(f1,...,fn)'.

Cooperative repressor binding with OR1. Note that as a repressor protein binds, OR2 initially switches to rep_low, regardless of the possible presence of another repressor at the O_{R1} representative. Figure 8.10 summarizes what happens next. A transition to OR2_rep_high ensues instantaneously by OR1_rep, calling raisedByOr1 in its line 29. O_{R2}'s profile for cooperative binding rep_high is left in line 29, because OR1 becomes free, or in line 27 upon local unbinding of repressor.

Positive control of initiation at PRM. OR2 exerts positive control of transcription initiation by calling raisedByOr2 over PRM's identifier channel prm. It does so in profile rep_low (line 24), as well as rep_high (lines 30).

Mutual exclusion of binding with PR. In all three protein bound profiles, OR2 inhibits PR by calling blockeByOr2 (lines 19, 23, and 31).

8.2.4 Abstraction of P_R

The abstractions of two λ switch promoters P_R and P_{RM} resemble that of the generic model in Figure 7.6, while adding their distinct features. The specialty of PR is to depend on a more intricate mutual exclusion mechanism than PRM. We need to cope with mutual exclusion with *both* operator regions O_{R1} and O_{R2}. Each of these is covered by our usual mechanism from Figure 8.13 on page 139, i.e. by dedicated functions for blocking and reversing the block.

It is worthwhile pointing out that both representatives OR1 and OR2 send PR_blocked to PR_free as themselves become free upon protein unbinding. In this case however, the respectively other operator forces PR back to blocked instantaneously if itself is still bound by some protein: OR1 ensures this in lines 17 and 21, see Figure 8.9. For OR2 listed in Figure 8.11 see lines 19 (in profile cro), in profiles rep_low and rep_high lines 23 and 31. These multiple dependencies are not evident from PR's isolated transition diagram in Figure 8.12.

8.2.5 Abstraction of P_{RM}

Last but not least, we move to the abstraction of P_{RM}. There is one significant difference to consider, compared to P_R: *positive control* of transcription

```
module 'OR2'
public pro with bind/1, or2 as me, prm, pr
import
 Timer from 'timer(timerRepHigh,timerRepLow,timerCro)'
    // see Fig. 6.9
export
  OR2 with    unbind/0,specCro/0,specRep/0,
  blockedByPr/0,freedByPr/0,raisedByOr1/0,resetByOr1/0
define
    OR2 ≜ OR2_free | Timer(me)
    OR2_free ≜
    +  pro ?bind(c).c!(me).
            me?specCro().OR2_cro
          + me?specRep().OR2_rep_low
    + prm!resetByOr2().OR2_free
    + me?blockedByPr().OR2_blocked
    OR2_cro ≜
        me?timerCro().me?unbind().pr!freedByOr2().OR2_free
     + pr!blockedByOr2().OR2_cro
    OR2_rep_low ≜
        me?timerRepLow().me?unbind().pr!freedByOr2().
            OR2_free
     + me?raisedByOr1().OR2_rep_high
     + pr!blockedByOr2().OR2_rep_low
     + prm!raisedByOr2().OR2_rep_low
    OR2_rep_high ≜
        me?timerRepHigh().
            me?unbind().
            pr!freedByOr2().OR2_free
     + me?resetByOr1().OR2_rep_low
     + prm!raisedByOr2().OR2_rep_high
     + pr!blockedByOr2().OR2_rep_high
    OR2_blocked ≜ me?freedByPr().OR2_free
```

Figure 8.11: O$_{\mathrm{R2}}$ module

initiation. Rnap switches between two alternative rates for its function initiate, reflecting the presence of a repressor protein at O$_{\mathrm{R2}}$.

Our model in Figure 8.14 deals with this quantitative control in the previously introduced manner. It maintains the interface, i.e. the functions offered to the Rnap representative while varying the initiation rate. As previously the technical solution consists in introducing alternative profiles for

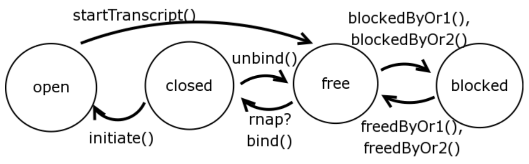


Figure 8.12: Transitions of PR

the Rnap bound state, PRM_low and PRM_high. In both the externally apparent rate of function initiate is internally determined by a Timer. Switching between the two profiles is subject to concurrent control by OR2, via invocations of the instantaneous functions raisedByOr2 (line 20) and resetByOr2 (line 25).

8.3 Simulation and discussion

We next validate our π-calculus model of the dynamics at the λ switch by exhaustive stochastic simulation. These are performed by execution with the BioSpi system (Priami et al., 2001).

Given its complexity, it does not make sense to directly simulationg the complete system. We use a sequence of models of distinguished subsystems in order to evaluate the different components independently. From the software engineering perspective, this allows to to successively validate the components. During our own work, the bottom-up approach to simulation was central to the establishment of a faithful set of stochastic parameters. From the biological standpoint, it is a current practice to isolate subsystems in order to observe their aspects as independently as possibly. The π-calculus programming approach is advantageous in that perspective, in that it allows to freely design and compose subsystems of interest.

It is a particular concern of ours to perform simulations of subsystems that can be clearly related to existing knowledge, either experimental or from established other simulation studies. Our strategy is incremental and bottom up. First we present simulations of repressor *dimerization*. We continue with *binding of the protein* Rep to Dna, and the impact of dimerization on binding patterns and cooperative interaction between Rep *on* Dna. Finally we investigate interactions between Rnap, the *promoter* $\mathrm{P_{RM}}$, and Rep's positive control thereof. The *control of transcription* initiation from $\mathrm{P_{RM}}$ is highly relevant. It has been subject to a number of theoretical and experimental studies as those of Baek et al. (2003), Bakk (2005) Li et al. (1997), Shea and Ackers (1985). The key to *repression* of this promoter has recently

```
module 'PR'
public rnap with bind/1, pr as me,
   cro_gene as next, or1, or2
rate  ρrbs_pr, ρrna from 'Table 7.1'
import RBS from module 'mRNA' //Fig7.8
export PR with blockedByOr1/0,freedByOr1/0,
   blockedByOr2/0,freedByOr2/0,unbind/0,initiate/1,
   free/1,startTranscript/1
define
   PR ≜ PR_free
   PR_free ≜
     rnap?bind(c).c!(me).PR_closed
   + me?blockedByOr2().PR_blocked
   + me?blockedByOr1().PR_blocked
   + or1!freedByPr().PR_free
   + or2!freedByPr().PR_free

   PR_blocked ≜
     me?freedByOr1().PR_free
   + me?freedByOr2().PR_free

   PR_closed ≜
      me?unbind().PR_free
    + me?initiate().PR_open
    + or1!blockedByPr().PR_closed
    + or2!blockedByPr().PR_closed

   PR_open ≜
     new rbs_pr:ρrbs_pr.new rna:ρrna.
     me?startTranscript(c).c!(rna).
     me?getNext(c2).c2!(next).
        PR_free | RBS(rbs_pr,rna)
```

Figure 8.13: P_R module

been revealed (Dodd et al., 2001). The model presented in this thesis may be extended to cover it in the future.

8.3.1 Dynamics of dimer formation and breakage

The essential point about repressor dimerization is the concentration dependent equilibrium (Ptashne, 2004). Figure 8.15 visualizes the dynamics of

```
module 'PRM'
public rnap with bind/1, prm as me,
   cI_gene as next, or2, or3
rate ρrbs_prm, ρrna from 'Table 7.1'
import
 RBS from module 'mRNA' //Fig7.8
 Timer from 'timer(lowTimer,highTimer)' //Fig.6.9
export PRM with getNext/0,unbind/0,initiate/0,
   startTranscript/1,freedByOr3/0,blockedByOr3/0,
   raisedByOr2/0,resetByOr2/0
define
   PRM ≜ PRM_vacant | Timer(me)
   PRM_free ≜
      me?blockedByOr3().PRM_blocked
   + or3!freedByPrm().PRM_free
   + rnap?bind(c).c!(me).PRM_low
   PRM_low ≜
     me?lowTimer().me?initiate().PRM_open
   + me?unbind().PRM_free
   + me?raisedByOr2().PRM_high
   + or3!blockedByPrm().PRM_low
   PRM_high ≜
     me?highTimer().me?initiate().PRM_open
   + me?unbind().PRM_free
   + me?resetByOr2().PRM_low
   + or3!blockedByPrm().PRM_high
   PRM_open ≜
      new rbs_prm:ρrbs_prm. new rna:ρrna.
      me?startTranscript(c).c!(rna).
      me?getNext(c2).c2!(next).
         PRM_free | RBS(rbs_prm,rna)
   PRM_blocked ≜
       me?freedByOr3().PRM_free
```

Figure 8.14: P_{RM} module

dimerization, starting with different numbers of monomers, as declared in Figure 8.17.

When launched with 20 monomers, such a system tends towards a mean setting in which around around half the total repressors can be found as monomers, while the others are present as dimers. Note that in this case, one observes strong fluctuations (see Figure 8.15 left). Only a rough quarter

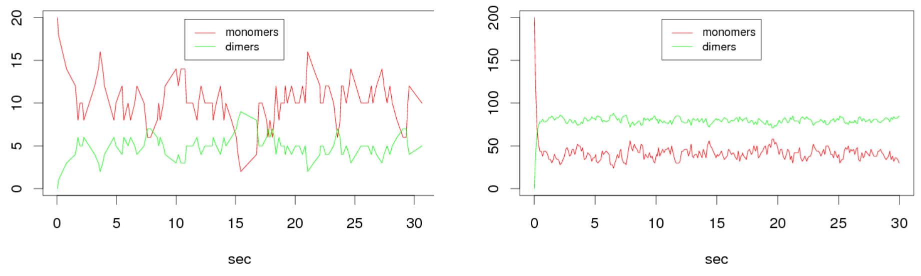


Figure 8.15: Dynamics of formation and breakage of λ repressor dimers over 30 simulated seconds, starting from 20 monomers (left) or 200 (right)

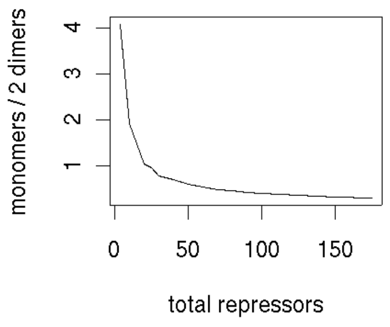


Figure 8.16: Shift of concentration dependent equilibrium between monomers and dimers

```
public lam_undimerize, lam_dimerize  from 'Table 8.7'
import
   'lambda repressor' // see Figure 8.6
    Timer from 'elementary timer'  // see page 103
define
   Π_{i=1}^{n}  Rep_monomer | Timer(lam_undimerize)
```

Figure 8.17: Setup: simulating dimerization starting from n repressor monomers

Figure 8.18: Binding to isolated operator sites, assuming 100 repressor monomers.

	ΔG	**mean sojourn**	**bound**
O_{R1}	-12.5	6.4	96 %
O_{R2}	-10.5	0.25	46 %
O_{R3}	-9.5	0.05	15 %

of initially 200 monomers are present as such in average, while around 75% are dimer-bound - with less important fluctuations. The *shift* of the equilibrium towards dimers becomes more obvious as we plot the average ratio of repressors present as monomers to that of dimer-bound ones over a long time range for various total repressor amounts, see Figure 8.16.

8.3.2 Repressor interactions with and on DNA

We now consider the binding of repressor dimers to operator sites on DNA. The set up in Figure 8.20 permits to simulate site O_{R1} and n repressor dimers that can reversibly attach to it, or dissociate back to monomers: By adjustment of ρ_{ori}(unbind) we can simulate binding to the *isolated* sites O_{R1}, O_{R2} and O_{R3}. Figure 8.18 summarizes the corresponding simulations, emphasizing the impact of different binding site site affinities. Recall from Section 3.2 that a smaller value of the Gibbs free energy ΔG indicates a stronger binding, hence a higher saturation of the site.

We make two corresponding observations. The complex of repressor and operator site is most stable at O_{R1}, where we observe an average sojourn time of Rep of 6.4 seconds (this value is the mean of an exponential distribution not shown here). This is consistent with reports of Ptashne (2004) that binding of repressor to O_{R1} persists in the order of up to 10 seconds. For O_{R2} and O_{R3} less favorable ΔGs lead to drastic drops of complex stability.

The effect is also visible when considering not individual binding events, but average behavior. For a given concentration and a long time scale, O_{R1} is better saturated with repressor than any of the other sites. The last column

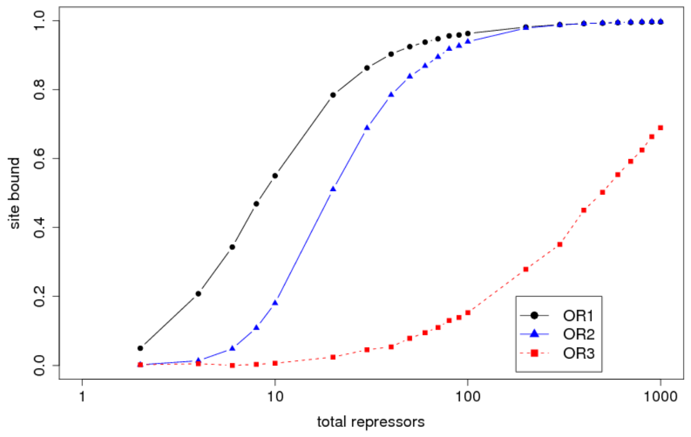


Figure 8.19: Occupancy of sites O_{R1}, O_{R2} and O_{R3} in the presence of dimerization and cooperative binding. Each point represents the relative occupancy of a site over 5000 simulated seconds, for some total repressor number. In a lysogen, one can expect 100-200 total repressors.

```
public pro, ori, lam_dimerize, lam_undimerize //i ∈ {1,2,3}'
import
    'lambda repressor'  // see Figure 8.6
    OR from 'OR with specificity'  // see Figure 8.5
    Timer from 'elementary timer'  // see page 103
rate ρori from 'Table 8.7'
define
    OR(ori) | Π_{k=1}^{n} Rep_dimer | Timer(lam_undimerize)
```

Figure 8.20: Setup: simulating repressor binding to isolated operator O_{Ri}

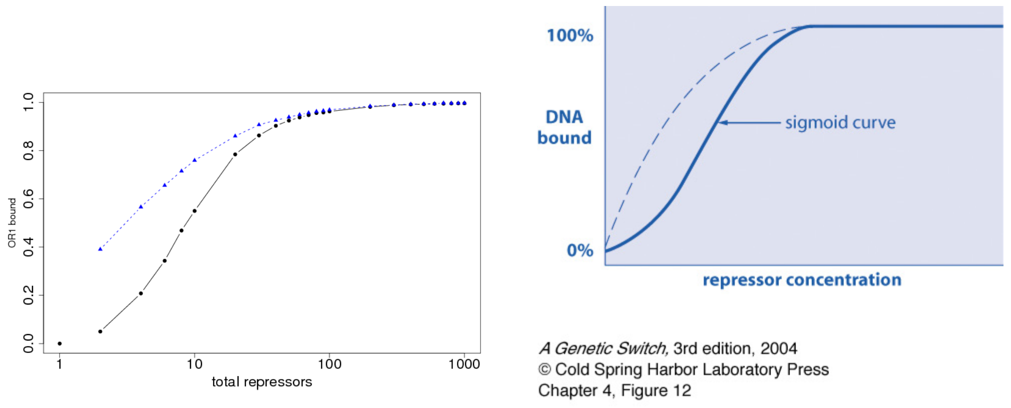


Figure 8.21: Occupancy of isolated operator region O_{R1} as a function of repressor concentration and dimerization. Our results (left), benchmark from Ptashne (2004) (right). Dashed lines: all repressors are found as dimers regardless of concentration. Solid lines: dimerization of repressors is included, hence the concentration-dependent equilibrium affects the binding curve.

in Table 8.18 reports the fraction of time the respective sites are bound when 100 Rep are included and dimerization activated. With respect to promoter control, these results show that P_{RM} is not yet efficiently repressed (via O_{R3} binding), based on the knowledge that our model integrates, neither at lysogenic repressor concentrations, nor when exceeding this level tenfold [4].

Synopsis: repressor binding to the three operator sites

Figure 8.19 shows the saturation of sites O_{R1}, O_{R2} and O_{R3} as they arise in our simulations of the λ switch when both repressor dimerization and cooperative repressor binding between O_{R1} and O_{R2} are enabled. Each of the curves summarizes a series of experiments for varying Rep levels. Before discussing the full system, we investigate the underlying components and mechanisms one by one.

Dimerization sharpens response at O_{R1}

Figure 8.21 (left) illustrates the saturation of O_{R1} as a function of repressor level. Each of the two curves summarizes a series of experiments. For

[4]In this light, the relevance of the quantitative predictions of repression of the cI gene, of Blossey, Cardelli, and Phillips (2006), encompassed by a representation of the sytem of Guet et al. (2002) appear questionable.

the solid line dimerization is enabled, i.e. only part of the total repressors are present as dimers and thus able to bind the operator. The dashed line assumes that dimers are stable at all concentrations, meaning that 100% of total repressors are found as dimers regardless of the concentration[5]. The horizontal axis indicates the number of total repressors on a logarithmic scale, while the vertical axis gives the relative occupancy of O_{R1}. Each data point represents the relative occupancy of O_{R1} for an experiment simulating the full dynamics of docking to DNA with or without dimerization over 5000 seconds.

Over this time scale, we can relate our results based on a stochastic discrete event approach to other's from deterministic continuous models, which compute only averages: one sees both qualitative and quantitative agreement with results[6] reported in Ptashne (2004), reproduced in Figure 8.21 (right). Dimerization has the effect to change the shape of the binding curve, namely to give a sharper response in terms of site occupancy as the amount of repressor increases.

Superimposing dimerization and cooperative binding at O_{R2}

Figure 8.22 summarizes how the second operator site fills with Rep for three scenarios. The dashed curve illustrates binding to O_{R2} in presence of O_{R1}, cooperative binding[7] and dimerization. We contrast this with binding to the isolated O_{R2} with and without dimerization. Note that the effect of *dimerization* is far less pronounced at an isolated O_{R2} than it was O_{R1}, where dimerization lead to a sharp increase of sensitivity in the lower concentration range. This can be explained because the isolated weaker O_{R2} only fills notably at higher Rep concentrations, when the equilibrium is heavily biased toward dimers. However, now the *combined effect of cooperativity and dimerization* becomes prominent. Recall that binding at O_{R2} is cooperatively strengthened as O_{R1} is placed next to it. As can be seen from the dashed curve in Figure 8.22 the predominant cause of O_{R2}'s saturation at lysogenic repressor concentration levels is cooperative binding with O_{R1}. This cooperativity propagates O_{R1}'s stronger sensitivity to O_{R2}.

[5]We eliminate communication over the undimerization channel in the module 'lambda repressor', by commenting out the corresponding line in Figure 8.20.

[6]Simulations by Keith Shearwin.

[7]Technically, we eliminate cooperative binding by commenting out the corresponding lines in the definition of either module 'OR2', or 'OR1'.

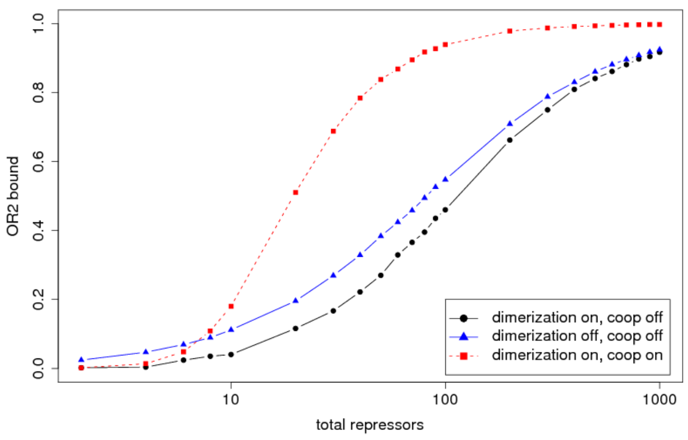


Figure 8.22: Occupancy of site O_{R2}, as a function of repressor level, dimerization and cooperativity. We consider *isolated* O_{R2} for the curves 'with'/'without' dimerization, and a system comprising both O_{R1} and O_{R2} for the curve 'O_{R2}, coop and dimerization'. Because at very low concentrations, binding occurs mainly at O_{R1}, the cooperative advantage only becomes visible at a certain level.

Negative auto-regulation of Rep at O_{R3}

Our results on Rep binding to the third operator O_{R3} are included in Figure 8.19. The binding curve is again based on the observation of the isolated operator, under variation of repressor level while disregarding interfering traffic between Rnap and P_{RM}. The most striking effect when comparing the binding curve with the other operators' is that the site fills to a significantly lower degree. It reaches around 30% when 200 repressors are included in the simulation. Even with an amount of 1000 repressors the isolated O_{R3} remains unsaturated. As we will report later the saturation further decreases to 4% as Rnap docking to P_{RM} interferes.

These results agree with recent experimental findings. Dodd et al. (2001) have demonstrated that an *additional layer of cooperativity* is needed for effective repression of P_{RM} at lysogenic repressor concentrations. Révet et al. (1999) first observed a long-range DNA loop between the right operator and another distal region in λ's genome. As was subsequently understood, this loop is stabilized by an assembly of eight repressor proteins, in which the two repressor dimers cooperatively bound to O_{R1} and O_{R2} participate. They cooperatively interact with another repressor tetramer bound to λ's *left operator* region O_L - while looping the DNA between the two regions. The large assembly further stabilizes all participants. More importantly, it juxtaposes O_{R3} with a third site at the left operator, O_{L3}. This allows for cooperative binding of repressor at O_{R3} and O_{L3}.

This additional level of cooperativity is a recent finding and not, as yet, fully characterized. However its importance is clearly described by Vilar and Saiz (2005). It allows to repress P_{RM} and maintain a low level of Rep, that is not yet ensured by binding to O_{R3} alone. Thus in a bacterium hosting phage λ, the lysogenic repressor concentration never surpasses a range allowing to return to the level in which O_{R1} and O_{R2} can be vacated. This is the key to induction back from lysogeny to the state of lytic growth (Dodd et al., 2004).

8.3.3 Monitoring the activity of P_{RM}

We further widen the scope of the model, by adding P_{RM} and Rnap to the three operator sites and repressors simulated so far (see Figure 8.23). We study this system over 20 minutes simulated time, comparable with the life span of an individual bacterium. This allows to observe the occupancy patterns of both P_{RM} and the operators, as well as the initiation of transcription for the cI gene. We first consider the impact of varying repressor levels in a model including the essential cooperative features - dimerization, cooperative repressor binding and positive control of transcription initiation. Next

```
module 'PRM activity with n repressors'
public rnap,pro,or1,or2,or3,pr,prm,cI_gene,
    lam_dimerize,lam_undimerize  from 'Table 8.7'
import 'lambda repressor','PRM',
    'OR1','OR2','OR3','RNAP','Timer'
define
    OR1 | OR2 | OR3 | PRM | Timer(lam_undimerize) |
      Π^n_{i=1} | Rep_monomer | Π^30_{i=1} RNAP
```

Figure 8.23: Setup: simulating the activity of P_{RM} in the presence of n repressor monomers

we perturb our model and study the consequences.

Table 8.1 summarizes a series of simulations. For each set-up we indicate the number of repressors included, and whether cooperative binding and positive control are enabled. Each line of the table summarizes one simulation run, for which we report the following quantities:

- absolute number of transcription initiations from P_{RM} observed. This should be related to the theoretical upper bound of 103, estimated form P_{RM}'s maximal rate of 0.086 initiations per second and the simulated time of 1200 seconds.
- absolute time that RNAP is bound to P_{RM} (in sec),
- absolute time P_{RM} is repressed as a consequence of Rep binding to O_{R3} (in sec),
- vacancy of P_{RM} (in sec),
- occupancy of O_{R2} by repressor (in sec).

To begin with, we mimic the system's behavior under *lysogenic repressor concentrations* with 100 repressors. The first block in Table 8.1 summarizes five executions of our model. For all runs P_{RM} is bound by RNAP in approximately 70 % of the time, P_{RM} is inhibited via competing Rep binding to O_{R3} for 4 % of the time, and otherwise vacant. The second operator site O_{R2} is bound by Rep around 92 % of the time for all five runs. The numbers of transcription initiations range between 58 and 76. Note that the variability of this figure is significantly higher than that of any other considered.

The actual number of transcript initiations from P_{RM} in a single lysogenic bacterium is difficult to determine experimentally. For long a precise

coop	pos ctrl	Rep	transcription initiations from P_{RM}	RNAP at P_{RM} (sec)	P_{RM} repressed (sec)	P_{RM} vacant (sec)	Rep at O_{R2} (sec)
on	on	100	58	873	46	280	1111
			76	870	46	279	1120
			71	873	46	279	1120
			58	889	44	266	1110
			74	903	43	253	1111
		50	67	901	19	281	990
			69	921	14	265	985
			70	899	18	283	972
		25	46	917	5	278	728
			52	929	5	266	730
			51	922	5	273	727
		10	19	927	< 1	272	305
			26	932	< 1	267	323
			28	922	< 1	277	300
off	on	100	38	886	41	273	570
			35	867	45	285	587
			31	890	43	267	575
on	off	100	5	899	41	260	1108
			6	915	39	246	1118
			4	898	44	258	1101

Table 8.1: Simulation runs over 1200 seconds: P_{RM} activity (absolute number of transcription initiations). Values are in units of simulated seconds for the following columns: P_{RM} repression, P_{RM} vacancy, and occupancy of O_{R2} by Rep. First block: results for varied repressor levels when both cooperative repressor binding and positive control are enabled. Second and third block: simulation results under elimination of either of the two mechanisms, for a level of 100 repressors.

level was deemed necessary for the maintenance of lysogeny (Johnson et al., 1981). This was rectified by recent experiments of Baek, Svenningsen, Eisen, Sneppen, and Brown (2003) showing that repressor levels varies widely from cell to cell. This phenomenon is known as *transcriptional noise*. Our result are in agreement with an estimated average number of transcripts per cell cycle of 70.

Reactions to partial and near-total Rep depletion:

Our next step is to preserve the system's essential characteristics - dimerization, positive control and cooperative binding - but to thin out the repressor pool. In the remainder of Table 8.1's first block we report the outcomes of each three simulations with 50, 25 and 10 total repressors.

The primary effect is that saturation of O_{R2} drops nonlinearly. This has important secondary effects on transcription initiation from P_{RM}. A first reduction to 50 repressors de-represses P_{RM} and seems to favor initiation, at least in these runs. As a reaction to further depletion transcription visibly reduces, while the actual P_{RM} saturation by Rnap increases: in the presence of 10 repressors, initiations drop to around a third of those seen with 50 repressors. This should be related to the increasing vacancy of O_{R2}. It gives a first impression of how the system of positive auto-regulation breaks down.

Examining this question in detail seems promising for two reasons. First, the λ switch is known to be extremely robust. It needs to cope with transient fluctuations of Rep level. Nevertheless, *induction* relies on the system's ability to escape from the lysogenic state when repressor falls below a critical threshold. Recall that as both O_{R1} and O_{R2} are vacated, P_{RM}'s antagonist P_R becomes likely to take over. A detailed investigation remains beyond the scope of this thesis.

The impact of cooperative repressor binding

on P_{RM} activity is another point of interest. After we have observed its immediate impact at O_{R2}, we move on to a larger perspective. We perturb our π-calculus model by lowering the cooperative dissociation rate of Rep at O_{R2} to the basal one. This lowers O_{R2}'s saturation to half the previous amount, see Table 8.1 (second block, last column). And it has consequences for Rnap at P_{RM}. The binding itself is not lowered - our simulations even indicate a slightly higher promoter saturation. Nevertheless, the number of transcription initiations drops to half that of the wild type.

Impact of positive control

We last eliminated the positive control of transcription initiation in our model. This is reached by setting the parameter for promoted transcription the basal one, i.e. $\rho_{\mathsf{prm}}(\mathsf{highTimer}) = \rho_{\mathsf{prm}}(\mathsf{lowTimer}) = 0.005$. Our resulting in silico experiments are summarized in the last block in Table 8.1. The number of initiations in presence of 100 total repressors dramatically decreases from an average of 67 in the initial system to an average of 5. All the while, immediate RNAP bindings as well as all other features are not affected, when compared to the original setting. This underlines the importance of positive control.

This simulation scenario was motivated by wet lab experiments with modified λ repressors. These mutants bind cooperatively but fail to stimulate transcription. As Hawley and McClure (1983) reported the λ switch was no longer functional. Our simulation outcomes seem in rough agreement with this, even though we can not directly compare the results. Michalowski and Little (2005) suggested that positive auto-regulation may be a dispensable feature altogether. They experimentally observed that the λ switch remains functional if positive control is eliminated, but at the same time P_{RM}'s intrinsic initiation rate k_f increased.

CHAPTER 9

Conclusion

We followed the approach of Regev (2002) in using the stochastic π-calculus as a modeling language yielding simulation in systems biology. Our driving question was how well the stochastic π-calculus lends itself to modeling gene expression and regulation. It motivated us to investigate bacterial gene expressions through two larger case studies.

Our experience soon showed that the pure stochastic π-calculus lacks many useful programming language concepts. As a bottleneck we identified difficulties to create object-oriented abstractions. This motivated us to augment the stochastic π-calculus with input patterns, that render appropriate object-oriented abstractions available. The main difficulty was to define the stochastic semantics. We proposed a module system atop the stochastic π-calculus with input patterns, that provides a syntax for specifying objects in the π-calculus while using inheritance.

Based on these contributions, we performed two modeling case studies: transcription initiation at the λ switch, and bacterial transcription and translation. Both confirmed the appropriateness and conciseness of our new modeling language. We proved that this language can be compiled into the original stochastic π-calculus. This in turn enabled us to implement and test our models within the BioSpi system. Based on this we have conducted exhausting simulations with excellent results.

In future work, it remains to design a new simulation system directly implementing the new modeling language. Open questions are on remain on how to integrate higher-order definitions and dynamic notions of inheritance.

On the modeling side, the largest remaining challenge is to tackle simulation questions of interest for experimentalists, beyond bacteriology. Phenomena as DNA looping at the λ switch are of particular interest, due to their fundamental relevance to eukaryotic gene regulation.

Bibliography

Abbondanzieri, E. A., W. J. Greenleaf, J. W. Shaevitz, R. Landick, and S. M. Block (2005). Direct observation of base-pair stepping by RNA polymerase. *Nature 438*, 460–465.

Ackers, G. K., A. D. Johnson, and M. A. Shea (1982, February). Quantitative model for gene regulation by λ phage repressor. *Proceedings of the National Academy of Sciences USA 79*(4), 1129–1133.

Adalsteinsson, D., D. McMillen, and T. Elston (2004). Biochemical Network Stochastic Simulator (BioNetS): software for stochastic modeling of biochemical networks. *BMC Bioinformatics 5*(1), 24.

Alberts, B., D. Bray, and J. Lewis (2002). *Molecular Biology of the Cell.* Garland Science.

Alon, U. (2006). *An Introduction to Systems Biology.* Mathematical and Computational Biology Series. Chapman & Hall/Crc.

Arkin, A., J. Ross, and H. H. McAdams (1998). Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected Escherichia coli cells. *Genetics 149*, 1633–1648.

Aurell, E., S. Brown, J. Johanson, and K. Sneppen (2002). Stability puzzles in phage λ. *Physical Review E 65*, 051914.

Aurell, E. and K. Sneppen (2002, January). Epigenetics as a first exit problem. *Physical Review Letters 88*(4), 048101.1–048101.4.

Baek, K., S. Svenningsen, H. Eisen, K. Sneppen, and S. Brown (2003). Single-cell analysis of λ immunity regulation. *Journal of Molecular Biology 334*(3), 363–372.

Bakk, A. (2005). Transcriptional activation mechanisms of the P_{RM} promoter of λ phage. *Biophysical Chemistry 114*(2–3), 229–234.

Balakin, A., E. Skripkin, I. Shatsky, and A. Bogdanov (1992). Unusual ribosome binding properties of mRNA encoding bacteriophage lambda repressor. *Nucleic Acids Research 20*(3), 563–571.

Baldamus, M., J. Parrow, and B. Victor (2005). A fully abstract encoding of the π-calculus with data terms. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung (Eds.), *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, Volume 3580 of *Lecture Notes in Computer Science*, pp. 1202–1213. Springer.

Banerjee, S., J. Chalissery, I. Bandey, and R. Sen (2006). Rho-dependent transcription termination: More questions than answers. *Journal of Microbiology 44*(1), 11–22.

Banâtre, J.-P., P. Fradet, and D. L. Métayer (2001). Gamma and the chemical reaction model: fifteen years after. In *Multiset Programming*, Volume 2235 of *Lecture Notes in Computer Science*. Springer Verlag.

Banâtre, J.-P., P. Fradet, and Y. Radenac (2005, June). Principles of chemical programming. In S. Abdennadher and C. Ringeissen (Eds.), *5th International Workshop on Rule-Based Programming*, Volume 124 of *Electronical notes in theoretical computer science*, pp. 133–147. Elsevier.

Banâtre, J.-P. and D. L. Metayer (1986, September). A new computational model and its discipline of programming. Technical Report RR0566, INRIA.

Banâtre, J.-P. and D. L. Métayer (1993). Programming by multiset transformation. *Communications of the ACM 36*(1), 98–111.

Barnard, A., A. Wolfe, and S. Busby (2004). Regulation at complex bacterial promoters: how bacteria use different promoter organizations to produce different regulatory outcomes. *Current Opinion in Microbiology 7*, 102–108.

Berg, O. G., R. B. Winter, and P. H. von Hippel (1981). Diffusion-driven mechanisms of protein translocation on nucleic acids: 1 - models and theory. *Biochemistry 20*, 6929–6948.

Bernardo, M., L. Donatiello, and R. Gorrieri (1994). MPA: A stochastic process algebra. Technical Report UBLCS-94-10, University of Bologna, Computer Science Laboratory.

Berry, G. and G. Boudol (1990). The chemical abstract machine. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pp. 81–94. The ACM Press.

Blossey, R., L. Cardelli, and A. Phillips (2006). A compositional approach to the stochastic dynamics of gene networks. *Transactions on Computational Systems Biology IV*, 99–122. LNBI vol 3939.

Bower, J. M. and H. Bolouri (Eds.) (2001). *Computational Modeling of Genetic and Biochemical Networks*. MIT Press.

Brenner, S., F. Jacob, and M. Meselson (1961). An unstable intermediate carrying information from genes to ribosomes for protein synthesis. *Nature 190*, 576–581.

Browning, D. F. and S. J. Busby (2004). The regulation of bacterial transcription initiation. *Nat Rev Microbiol 2*, 57–65.

Bryant, G. O. and M. Ptashne (2003). Independent recruitment in vivo by Gal4 of two complexes required for transcription. *Molecular Cell 11*, 1301–1309.

Bundschuh, R., F. Hayot, and C. Jayaprakash (2003). The role of dimerization in noise reduction of simple genetic networks. *Journal of Theoretical Biology 220*, 261–269.

Calder, M., S. Gilmore, and J. Hillston (2006). Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *Transactions on Computational Systems Biology 4230*(VII), 1–23.

Cardelli, L. (2005). Brane calculi: interactions of biological membranes. In *Proceedings of CMSB 2004*, Volume 3082 of *Lecture Notes in Bioinformatics*, pp. 257–278.

Cardelli, L. and A. Gordon (2000). Mobile ambients. *Theoretical Computer Science 240*(1), 177–213.

Carpousis, A. J. (2002). The Escherichia coli RNAdegradosome: structure, function and relationship to other ribonucleolytic multienzyme complexes. *Biochemical Society Transactions 30*(2), 150–154.

Carrier, T. A. and J. D. Keasling (1997). Mechanistic modeling of mRNA decay. *Journal of Theoretical Biology 189*, 195–209.

Chabrier-Rivier, N., M. Chiaverini, V. Danos, F. Fages, , and V. Schächter (2004). Modeling and querying biomolecular interaction networks. *Theoretical Computer Science 325*(1), 24–44.

Chabrier-Rivier, N., F. Fages, and S. Soliman (2005). The biochemical abstract machine BioCham. In *Proceedings of CMSB 2004*, Volume 3082 of *Lecture Notes in Bioinformatics*, pp. 172–191.

Chang, B.-Y. E. (2007). PML: Toward a high-level formal language for biological systems. In *1st International Workshop on Concurrent Models in Molecular Biology (Bioconcur 2003)*, Volume 180 of *ENTCS*, pp. 15–30. Elsevier.

Chiarugi, D., M. Chinellato, P. Degano, G. L. Brutto, and R. Marangoni (2006). Feedbacks and oscillations in the virtual cell VICE. In *Computational Methods in Systems Biology (CMSB)*, Volume 4210 of *Lecture Notes in Bioinformatics*, pp. 93–107.

Chung, J. D. and G. Stephanopoulos (1996). On physiological multiplicity and population heterogeneity of biological systems. *Chem Eng Sci 51*, 1509–1521.

Ciocchetta, F. and C. Priami (2006). Biological transactions for quantitative models. In *Proceedings of MeCBIC 2006*, Electronic Notes in Theoretical Computer Science. Elsevier. to appear.

Curti, M., D. Chiarugi, P. Degano, and R. Marangoni (2005). VICE: A VIrtual CEll. In *Proceedings of CMSB 2004*, Volume 3082 of *Lecture Notes in Bioinformatics*.

Danos, V. and J. Krivine (2004). Reversible communicating systems. In *CONCUR 2004*, Volume 3170 of *LNCS*, pp. 292–307.

Danos, V. and J. Krivine (2005). Transactions in RCCS. In *CONCUR 2005*, Volume 3753, pp. 398–412.

Danos, V. and J. Krivine (2007). Formal molecular biology done in CCS-R. In *1st International Workshop on Concurrent Models in Molecular Biology (Bioconcur 2003)*, Volume 180 of *ENTCS*, pp. 31–49. Elsevier.

Danos, V. and C. Laneve (2004). Formal molecular biology. *Theoretical Computer Science 325*(1), 69–110.

Davidson, E. H. (2006). *The Regulatory Genome: Gene Regulatory Networks In Development And Evolution*. Academic Press.

Dennis, P. P., M. Ehrenberg, and H. Bremer (2004). Control of rRNA synthesis in Escherichia coli: a systems biology approach. *Microbiology and Molecular Biology Reviews 68*(4), 639–668.

Dijkstra, E. W. (1971). Hierarchical ordering of sequential processes. *Acta Inf. 1*, 115–138.

Dodd, I. B., A. Perkins, D. Tsemitsidis, and J. Egan (2001). Octamerization of CI repressor is needed for effective repression of P_{RM} and efficient switching from lysogeny. *Genes & Development 15*, 3013–3022.

Dodd, I. B., K. E. Shearwin, and J. B. Egan (2005). Revisited gene regulation in bacteriophage λ. *Current Opinion in Genetics & Development 15*(2), 145–152.

Dodd, I. B., K. E. Shearwin, A. J. Perkins, T. Burr, A. Hochschild, and J. B. Egan (2004). Cooperativity in long-range gene regulation by the lambda CI repressor. *Genes Dev. 18*(3), 344–354.

Duchier, D. and C. Kuttler (2006). 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*, pp. 31–49.

Emerson, E. (1990). *Handbook of Theoretical Computer Science*, Volume B, Chapter Temporal and Modal Logic, pp. 995–1072. Cambridge, Massachusetts: The MIT Press.

Fedoroff, N. V. and W. Fontana (2002). Small numbers of big molecules. *Science 297*, 1129–1131.

Fisher, J., N. Piterman, J. Hubbard, M. Stern, and D. Harel (2005). Computational insights into C. elegans vulval development. *Proceedings of the National Academy of Sciences USA 102*(5), 1951–1956.

Fontana, W. and L. W. Buss (1996). *Boundaries and Barriers*, Chapter The barrier of objects: From dynamical systems to bounded organizations, pp. 56–116. Addison-Wesley.

Frank, J. and R. K. Agrawal (2000). A ratchet-like inter-subunit reorganization of the ribosome during translocation. *Nature 406*, 318–322.

Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics 22*, 403–434.

Golding, I., J. Paulsson, S. Zawilski, and E. Cox (2005). Real-time kinetics of gene activity in individual bacteria. *Cell 123*(6), 1025–1036.

Goss, P. J. and J. Peccoud (1998). Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. *Proceedings of the National Academy of Sciences USA 95*(12), 6750–6755.

Gottesman, M. (1999). Bacteriophage λ: The untold story. *Journal of Molecular Biology 293*, 177–180.

Gottesman, M. E. and R. A. Weisberg (2004). Little Lambda, Who Made Thee? *Microbiol. Mol. Biol. Rev. 68*(4), 796–813.

Gowrishankar, J. and R. Harinarayanan (2004). Why is transcription coupled to translation in bacteria? *Molecular Microbiology 54*(3), 598–603.

Greive, S. J. and P. H. von Hippel (2005). Thinking quantitatively about transcriptional regulation. *Nature Reviews Molecular Cell Biology 6*, 221.

Gros, F., H. Hiatt, W. Gilbert, C. Kurland, R. Risebrough, and J. Watson (1961). Unstable ribonucleic acid revealed by pulse labeling of Escherichia coli. *Nature 190*, 581–585.

Grunberg-Manago, M. (1999). Messenger RNA stability and its role in control of gene expression in bacteria and phages. *Annual Reviews Genetics 33*, 193–227.

Guet, C. C., M. B. Elowitz, W. Hsing, and S. Leibler (2002). Combinatorial synthesis of genetic networks. *Science 296*, 1466–1470.

Guido, N. J., X. Wang, D. Adalsteinsson, D. McGillen, J. Hasty, C. R. Cantor, T. C. Elston, and J. Collins (2006). A bottom-up approach to gene regulation. *Nature 439*(16), 856–860.

Halford, S. E., A. J. Welsh, and M. D. Szczelkun (2004). Enzyme-mediated DNA looping. *Annual Review of Biophysics and Biomolecular Structure 33*(1), 1–24.

Harel, D. (1987, June). Statecharts: A visual formalism for Complex Systems. *Science of Computer Programming 8*(3), 231–274.

Hasty, J., D. McMillen, F. Isaacs, and J. J. Collins (2001). Computational studies of gene regulatory networks: In numero molecular biology. *Nature Reviews Genetics 2*, 268–279.

Hawley, D., A. Johnson, and W. McClure (1985). Functional and physical characterization of transcription initiation complexes in the bacteriophage lambda O_R region. *J. Biol. Chem. 260*(14), 8618–8626.

Hawley, D. and W. McClure (1983). The effect of a lambda repressor mutation on the activation of transcription initiation from the lambda P_{RM} promoter. *Cell 32*, 327–333.

Heath, J., M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn (2006). Probabilistic model checking of complex biological pathways. In *Computational Methods in Systems Biology (CMSB)*, Volume 4210 of *Lecture Notes in Bioinformatics*, pp. 32–47.

Henkin, T. M. (2000). Transcription termination control in bacteria. *Current Opinion in Microbiology 3*(2), 149–153.

Hermanns, H. (2002). *Interactive Markov Chains: The Quest for Quantified Quality*, Volume 2428 of *Lecture Notes in Computer Science.* Springer.

Hillston, J. (1995). *A Compositional Approach to Performance Modelling.* Ph. D. thesis, University of Edinburgh. Distinguished Dissertations Series. Cambridge University Press, 1996.

Honda, K. (1992). Two bisimilarities in **new**-calculus. Technical Report 92-002, Keio University, Department of Computer Science.

Hsu, L. M. (2002). Promoter clearance and escape in prokaryotes. *Biochimica et Biophysica Acta 1577*, 191–207.

Hucka, M., A. Finney, H. Sauro, H. Bolouri, J. Doyle, H. Kitano, A. Arkin, B. Bornstein, D. Bray, A. Cornish-Bowden, A. Cuellar, S. Dronov, E. Gilles, M. Ginkel, V. Gor, I. Goryanin, W. Hedley, T. Hodgman,

J. Hofmeyr, P. Hunter, N. Juty, J. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. Loew, D. Lucio, P. Mendes, E. Minch, E. Mjolsness, Y. Nakayama, M. Nelson, P. Nielsen, T. Sakurada, J. Schaff, B. Shapiro, T. Shimizu, H. S. andJ Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics 19*, 524–531.

Ideker, T., T. Galitski, and L. Hood (2001). A new approach to decoding life: systems biology. *Annual Review of Genomics and Human Genetics 2*, 343–72.

Jacob, F. and J. Monod (1961). Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology 3*, 318–356.

Johnson, A., B. Meyer, and M. Ptashne (1979). Ineractions between DNA-bound repressors govern regulation by the phage λ repressor. *Proceedings of the National Academy of Sciences USA 76*, 5061–5065.

Johnson, A. D., A. R. Poteete, G. Lauer, R. T. Sauer, G. K. Ackers, and M. Ptashne (1981). λ repressor and Cro – components of an efficient molecular switch. *Nature 294*(5838), 217–223.

Kaern, M., T. Elston, W. Blake, and J. Collins (2005). Stochasticity in gene expression: from theories to phenotypes. *Nature Reviews Genetics 6*(6), 451–467.

Kampen, N. V. (2001). *Stochastic Processes in Physics and Chemistry.* North Holland.

Kierzek, A. M. (2002). STOCKS: STOChastic Kinetic Simulations of biochemical systems with Gillespie algorithm. *Bioinformatics 18*(3), 470–481.

Kierzek, A. M., J. Zaim, and P. Zielenkiewicz (2001). The effect of transcription and translation initiation frequencies on the stochastic fluctuations in prokaryotic gene expression. *Journal of Biological Chemistry 276*, 8165–8172.

Kitano, H. (Ed.) (2001). *Foundations of Systems Biology.* MIT Press.

Kobiler, O., A. Rokney, N. Friedman, D. L. Court, J. Stavans, and A. B. Oppenheim (2005). Quantitative kinetic analysis of the bacteriophage λ genetic network. *Proceedings of the National Academy of Sciences USA 102*(12), 4470–4475.

Koblan, K. S. and G. K. Ackers (1992). Site-specific enthalpic regulation of DNA transcription at bacteriophage λ O_R. *Biochemistry 31*, 57–65.

Kou, S. C., B. J. Cherayil, W. Min, B. P. English, and X. S. Xie (2005). Single-molecule michaelis-menten equations. *J. Phys. Chem. B 109*(41), 19068 – 19081.

Kuttler, C. (2006). Simulating bacterial transcription and translation in a stochastic pi-calculus. In G. Plotkin and C. Priami (Eds.), *Transactions on Computational Systems Biology VI*, Volume 4220 of *Lecture Notes in Bioinformatics*, pp. 113–149. Springer. A preliminary version was presented at 3rd International Workshop on Computational Methods in Systems Biology, Edinburgh, April 2005.

Kuttler, C., C. Lhoussaine, and J. Niehren (2006). A stochastic pi calculus for concurrent objects. In *1st International Workshop on Probabilistic Automata and Logics*.

Kuttler, C. and J. Niehren (2006). Gene regulation in the pi-calculus: Simulating cooperativity at the lambda switch. In C. Priami, A. Ingólfsdóttir, H. R. Nielson, and B. Mishra (Eds.), *Transactions on Computational Systems Biology VII*, Volume 4230 of *Lecture Notes in Bioinformatics*, pp. 24–55. Springer. Preliminary versions presented at the 2nd International Workshop on Concurrent Models in Molecular Biology (Bioconcur 2004), and the Dagstuhl seminar 04281 (July 2004).

Kwiatkowska, M., G. Norman, and D. Parker (2004). Probabilistic symbolic model checking with prism: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT) 6*(2), 128–142.

Kwiatkowska, M., G. Norman, D. Parker, O. Tymchyshyn, J. Heath, and E. Gaffney (2006). Simulation and verification for computational modelling of signalling pathways. In L. Perrone, F. Wieland, J. Liu, B. Lawson, D. Nicol, and R. Fujimoto (Eds.), *Proc. 2006 Winter Simulation Conference*. To appear.

Lecca, P., C. Priami, P. Quaglia, B. Rossi, C. Laudanna, and G. Constantin (2004, June). A stochastic process algebra approach to simulation of autoreactive lymphocyte recruitment. *SCS Simulation 80*(6), 273–288.

Leloup, J.-C. and A. Goldbeter (1999). Chaos and biorhythmicity in a model for circadian oscillations of the PER and TIM proteins in *Drosophila*. *Journal of Theoretical Biology 198*(3), 445–459.

Lewin, B. (2003). *Genes VIII.* Prentice Hall.

Li, M., W. McClure, and M. M. Susskind (1997). Changing the mechanism of transcriptional activation by phage λ repressor. *Proceedings of the National Academy of Sciences USA 94*, 3691–3696.

Little, J. W., D. P. Shepley, and D. W. Wert (1999). Robustness of a gene regulatory circuit. *European Molecular Biology Organization (EMBO) 18*(15), 42999–4307.

Lodish, H. (2003). *Molecular Cell Biology.* Freeman.

Maciag, K., S. J. Altschuler, M. D. Slack, N. J. Krogan, A. Emili, J. F. Greenblatt, T. Maniatis, and L. F. Wu (2006). Systems-level analyses identify extensive coupling among gene expression machines. *Molecular Systems Biology 2.*

Manabe, T. (1981). Theory of regulation by the attenuation mechanism: stochastic model for the attenuation of the Escherichia coli tryptophan operon. *Journal of Theoretical Biology 91*, 527–544.

Maniatis, T. and R. Need (2002). An extensive network of coupling among gene expression machines. *Nature 416*, 499–506.

McAdams, H. H. and A. Arkin (1997). Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences USA 94*, 814–819.

McAdams, H. H. and L. Shapiro (1995). Circuit simulation of genetic networks. *Science 269*, 650–656.

McClure, W. R. (1985). Mechanism and control of transcription initiation in prokaryotes. *Annual Review Biochemistry 54*, 171–204.

McKean, E. (Ed.) (2005). *The New Oxford American Dictionar* (Second Edition ed.). Oxford University Press.

Michalowski, C. B. and J. W. Little (2005). Postitive autoregulation of cI is a dispensable feature of the phage λ gene regulatory circuitry. *Journal of Bacteriology 187*(18), 6430–6442.

Milner, R. (1980). *A calculus of communicating systems.* Lecture Notes in Computer Science. Springer.

Milner, R. (2004). *Computer Systems: Theory, Technology, and Applications. A tribute to Roger Needham*, Chapter What's in a name?, pp. 205–211. Monographs in Computer Science. Springer.

Milner, R., J. Parrow, and D. Walker (1992). A calculus of mobile processes (I and II). *Information and Computation 100*, 1–77.

Moll, I., S. Grill, C. O. Gualerzi, and U. Bläsi (2002). Leaderless mRNA in bacteria: surprises in ribosomal recruitment and translational control. *Molecular Microbiology 43*(1), 239–246.

Moll, I., G. Hirokawa, M. C. Kiel, A. Kaji, and U. Bläsi (2004). Translation initiation with 70S ribosomes: an alternative pathway for leaderless mRNAs. *Nucleic Acids Research 32*(11), 3354–3363.

Morton-Firth, C. and D. Bray (1998). Predicting temporal fluctuations in an intracellular signalling pathway. *Journal of Theoretical Biology 192*(1), 117–128.

Morton-Firth, C. J. (1998). *Stochastic simulation of cell signalling pathways.* Ph. D. thesis, University of Cambridge.

Müller-Hill, B. (1996). *The Lac operon: a short history of a genetic paradigm.* Walter de Gruyter.

Müller-Hill, B. (2006). What is life? The paradigm of DNA and protein cooperation at high local concentrations. *Molecular Microbiology 60*(2), 253–255.

Niehren, J. (2000, September). Uniform confluence in concurrent computation. *Journal of Functional Programming 10*(5), 453–499.

Oppenheim, A. B., O. Kobiler, J. Stavans, D. L. Court, and S. Adhya (2005). Switches in bacteriophage lambda development. *Annual Reviews Genetics 39*, 409–429.

Orphanides, G. and D. Reinberg (2002). A unified theory of gene expression. *Cell 108*, 439–451.

Orrell, D., S. Ramsey, P. de Atauri, and H. Bolouri (2005). A method for estimating stochastic noise in large genetic regulatory networks. *Bioinformatics 21*(2), 208–217.

Ozbudak, E. M., M. Thattai, I. Kurtser, A. Grossman, and A. van Oudenaarden (2002). Regulation of noise in the expression of a single gene. *Nature Genetics 31*, 69–73.

Palamidessi, C. (2003). Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science 13*(5), 685–719.

Palsson, B. O. (2006). *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press.

Parsell, D., K. R. Silber, and R. Sauer (1990). Carboxy-terminal determinants of intracellular protein degradation. *Genes Dev. 4*(2), 277–286.

Paul, B. J., W. Ross, T. Gaal, and R. L. Gourse (2004). rRNA transcription in E. Coli. *Annual Review Genetics 38*, 749–770.

Paulino, H., P. Marques, L. Lopes, V. T. Vasconcelos, and F. Silva (2003). A multi-threaded asynchronous language. In *7th International Conference on Parallel Computing Technologies*, Volume 2763 of *Lecture Notes in Computer Science*, pp. 316–323. Springer.

Paulsson, J. (2005). Models of stochastic gene expression. *Physics of Life Reviews 2*(2), 157–175.

Petri, C. A. (1962). *Kommunikatin mit Automaten*. Ph. D. thesis, Bonn University.

Phillips, A. and L. Cardelli (2004). A correct abstract machine for the stochastic pi-calculus. In *2nd International Workshop on Concurrent Models in Molecular Biology (Bioconcur 2004)*.

Priami, C. (1995). Stochastic π-calculus. *Computer Journal 6*, 578–589.

Priami, C. and P. Quaglia (2005). Beta binders for biological interactions. In *Proceedings of CMSB 2004*, Volume 3082 of *Lecture Notes in Bioinformatics*, pp. 20–33.

Priami, C., A. Regev, E. Shapiro, and W. Silverman (2001). Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters 80*, 25–31.

Ptashne, M. (2004). *A Genetic Switch: Phage Lambda Revisited* (3rd ed.). Cold Spring Harbor Laboratory Press.

Ptashne, M. (2005). Regulation of transcription: from lambda to eukaryotes. *Trends in Biochemical Sciences 30*(6), 275–279.

Ptashne, M. and A. Gann (2002). *Genes and Signals*. Cold Spring Harbor Laboratory Press.

Păun, G. (2000). Computing with membranes. *Journal of Computer and System Science 61*(1), 108–143.

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0.

Ramsey, S., D. Orrell, and H. Bolouri (2005). Dizzy: stochastic simulation of large-scale genetic regulatory networks. *Journal of Bioinformatics and Computational Biology 3*(2), 415–436.

Raser, J. M. and E. K. O'Shea (2005). Noise in Gene Expression: Origins, Consequences, and Control. *Science 309*(5743), 2010–2013.

Ravara, A. and V. T. Vasconcelos (2000). Typing non-uniform concurrent objects. In *CONCUR'00*, Volume 1877 of *Lecture Notes in Computer Science*, pp. 474–488. Springer.

Ray, P. and M. Pearson (1974). Evidence for post-transcriptional control of the morphogenetic genes of bacteriophage lambda. *Journal Molecular Biology 85*(1), 163–175.

Regev, A. (2002). *Computational Systems Biology: A Calculus for Biomolecular Knowledge*. Tel Aviv University. PhD thesis.

Regev, A., E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro (2004). BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science 325*(1), 141–167.

Regev, A. and E. Shapiro (2002). Cells as computation. *Nature 419*, 343.

Regonesi, M. E., M. D. Favero, F. Basilico, F. Briani, L. Benazzi, P. Tortora, P. Mauri, and G. Deho (2006). Analysis of the Escherichia coli RNA degradosome composition by a proteomic approach. *Biochimie 88*(2), 151–161.

Roy, P. V. (Ed.) (2005). *Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004, Charleroi, Belgium, October 7-8, 2004, Revised Selected and Invited Papers*, Volume 3389 of *Lecture Notes in Computer Science*. Springer-Verlag.

Révet, B., B. von Wilcken-Bergmann, H. Bessert, A. Barker, and B. Müller-Hill (1999). Four dimers of lambda repressor bound to two suitably spaced pairs of lambda operators form octamers and DNA loops over large distances. *Current Biology 9*(3), 151–154.

Saiz, L. and J. M. G. Vilar (2006). Stochastic dynamics of macromolecular-assembly networks. *Molecular Systems Biology*. `doi:10.1038/msb4100061`.

Schmeissner, U., D. Court, H. Shimatake, and M. Rosenberg (1980). Promoter for the establishment of repressor synthesis in bacteriophage lambda. *Proceedings of the National Academy of Sciences USA 77*(6), 3191–3195.

Shea, M. and G. K. Ackers (1985). The O_R control system of bacteriophage lambda: A physical-chemical model for gene regulation. *Molecular Biology 181*, 211–230.

Shearwin, K., B. Callen, and J. Egan (2005). Transcriptional interference - a crash course. *Trends in Genetics 21*, 339–345.

Slutsky, M. and L. A. Mirny (2004). Kinetics of protein-DNA interaction: Facilitated target location in sequence-dependent potential. *Biophys. J. 87*(6), 4021–4035.

Smolka, G. (1995). The Oz programming model. In J. van Leeuwen (Ed.), *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pp. 324–343. Berlin: Springer-Verlag.

Sneppen, K. and G. Zocchi (2005). *Physics in Molecular Biology*. Cambridge University Press.

Steege, D. A. (2000). Emerging features of mRNA decay in bacteria. *RNA 6*(8), 1079–1090.

Svenningsen, S. L., N. Costantino, D. L. Court, and S. Adhya (2005). On the role of Cro in λ prophage induction. *Proceedings of the National Academy of Sciences USA 102*(12), 4465–4469.

Szallasi, Z., J. Stelling, and V. Periwal (Eds.) (2006). *System modeling in cellular biology*. MIT press.

Thieffry, D. and R. Thomas (1995). Dynamical behaviour of biological networks: II. immunity control in bacteriophage lambda. *Bulletin Mathematical Biology 57*(2), 277–297.

Thomas, R., A.-M. Gathoye, and L. Lambert (1976). A complex control circuit: Regulation of immunity in temperate bacteriophages. *Eur. J. Biochem 71*, 211–227.

Thompson, M. P. and R. Kurzrock (2004). Epstein-Barr Virus and Cancer. *Clin Cancer Res 10*(3), 803–821.

Tian, T. and K. Burrage (2004). Bistability and switching in the lysis/lysogeny genetic regulatory network of bacteriophage λ. *Journal of Theoretical Biology 227*(2), 229–237.

Trun, N. and J. Trempy (2003). *Fundamental bacterial genetics.* Blackwell.

Tymchyshyn, O., G. Norman, J. K. Heath, and M. Z. Kwiatkowska (2006). Computer assisted biological reasoning: the simulation and analysis of fgf singalling pathway dynamics. In *NCRI Cancer Conference, Birmingham, UK.*

Uhrmacher, A. M. and C. Kuttler (2006). Multi-level modeling in systems biology by discrete event approaches. *it – Information Technology 48*(3), 148–153. Special issue on Systems Biology.

Vasconcelos, V. T. and M. Tokoro (1993). 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*, pp. 460–474. Springer.

Vilar, J. M. and L. Saiz (2005). DNA looping in gene regulation: from the assembly of macromolecular complexes to the control of transcriptional noise. *Current Opinion in Genetics & Development 15*, 1–9.

Voit, E. (2000). *Computational Analysis of Biochemical Systems: A Practical Guide for Biochemists and Molecular Biologists.* Cambridge University Press.

von Dassow, G., E. Meir, E. M. Munro, and G. M. Odell (2000). The segment polarity network is a robust developmental module. *Nature 406*, 188–192.

von Heijne, G., L. Nilsson, and C. Blomberg (1977). Translation and messenger RNA secondary structure. *Journal of Theoretical Biology 68*, 321–329.

Vossen, G. and G. Weikum (2001). *Fundamentals of Transactional Information Systems.* Morgan Kaufmann.

Wagner, R. (2000). *Transcription Regulation in Prokaryotes.* Oxford University Press.

Yanofsky, C. (1981). Attenuation in the control of expression of bacterial operons. *Nature 289*, 751–758.

Young, L. S. and A. B. Rickinson (2004). Epstein-Barr virus: 40 years on. *Nature Reviews Cancer 4*, 757–768.

Zeigler, B. (1984). *Multifacetted Modelling and Discrete Event Simulation.* London: Academic Press.

Zhu, X.-M., L. Yin, L. Hood, and P. Ao (2004). Calculating biological behaviors of epigenetic states in the phage lambda life cycle. *Funct Integr Genomics 4*, 188–195.