Journal Subline

LNBI 4220

Transactions on Computational Systems Biology VI

Corrado Priami
Editor-in-Chief

Springer

Corrado Priami   Gordon Plotkin (Eds.)

Transactions on Computational Systems Biology VI

Springer

Series Editors

Sorin Istrail, Brown University, Providence, RI, USA
Pavel Pevzner, University of California, San Diego, CA, USA
Michael Waterman, University of Southern California, Los Angeles, CA, USA

Editor-in-Chief

Corrado Priami
The Microsoft Research - University of Trento
Centre for Computational and Systems Biology
Piazza Manci, 17, 38050 Povo (TN), Italy
E-mail: priami@dit.unitn.it

Volume Editor

Gordon Plotkin
The University of Edinburgh
Division of Informatics
Mayfield Road, Edinburgh EH9 3JZ, UK
E-mail: gdp@inf.ed.ac.uk

Library of Congress Control Number: 2006935913

CR Subject Classification (1998): J.3, F.1, F.4, I.6

LNCS Sublibrary: SL 8 – Bioinformatics

ISSN	1861-2075
ISBN-10	3-540-45779-8 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-45779-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11880646 06/3142 5 4 3 2 1 0

Preface

This issue of Transactions on Computational Systems Biology contains a fully-refereed selection of papers from the Fourth International Conference on Computational Methods in Systems Biology, held in Edinburgh, Scotland, April 3–5, 2005. I would like to thank both the referees for all their hard work and also the CMSB 2005 programme committee for their help in choosing which papers to invite for submission.

June 2006

Gordon Plotkin
Program Chair
CMSB 2005

LNCS Transactions on Computational Systems Biology – Editorial Board

Table of Contents

Property-Driven Statistics of Biological Networks

Pierre-Yves Bourguignon[1], Vincent Danos[2], François Képes[3], Serge Smidtas[1], and Vincent Schächter[1]

[1] Genoscope
[2] CNRS & Université Paris VII
[3] CNRS

Abstract. An analysis of heterogeneous biological networks based on randomizations that preserve the structure of component subgraphs is introduced and applied to the yeast protein-protein interaction and transcriptional regulation network. Shuffling this network, under the constraint that the transcriptional and protein-protein interaction subnetworks are preserved reveals statistically significant properties with potential biological relevance. Within the population of networks which embed the same two original component networks, the real one exhibits simultaneously higher bi-connectivity (the number of pairs of nodes which are connected using both subnetworks), and higher distances. Moreover, using restricted forms of shuffling that preserve the interface between component networks, we show that these two properties are independent: restricted shuffles tend to be more compact, yet do not lose any bi-connectivity.

Finally, we propose an interpretation of the above properties in terms of the signalling capabilities of the underlying network.

1 Introduction

The availability of genome-scale metabolic, protein-protein interaction and regulatory networks [25,7,3,5,21] —following closely the availability of large graphs derived from the Internet hardware and software network structure, from social or collaborative relationships— has spurred considerable interest in the empirical study of the statistical properties of these 'real-world' networks. As part of a wider effort to reverse-engineer biological networks, recent studies have focused on identifying [illegible] graph properties that can be interpreted as 'traces' of underlying biological mechanisms, shedding light either on their dynamics [23,11,6,28] ([illegible], how the connectivity structure of the biological process reflects its dynamics), on their evolution [10,30,27] ([illegible], likely scenarios for the evolution of a network exhibiting the observed property or properties), or both [9,14,15]. The statistical graph properties that have been studied in this context include the distribution of vertex degrees [10,9], the distribution of the clustering coefficient and other notions of density [17,18,19,22,4], the distribution of vertex-vertex distances [22], and more recently the distribution of network motifs occurrences [15].

Identification of a salient property in an empirical graph —for example the fact that the graph exhibits a unexpectedly skewed vertex degree distribution— requires a prior notion of the distribution of that property in a class of graphs relatively to which saliency is determined. The approach chosen by most authors so far has been to use a [illegible], typically given by a probabilistic graph generation algorithm that constructs graphs by local addition of vertices and edges [20,1,24]. For the simplest random graph models, such as the classical Erdös-Rényi model (where each pair of vertices is connected with constant probability p, [2]), analytical derivations of the simplest of the above graph properties are known [20,1].

In the general case, however, analytical derivation is beyond the reach of current mathematical knowledge and one has to retort to numerical simulation. The random graph model is used to generate a sample of the corresponding class of graphs and the distribution of the graph property of interest is evaluated on that sample, providing a standard against which the bias of the studied graph can be measured [23,14,29]. Perhaps because of the local nature of the random graph generation process, it is mostly simple [illegible] network properties that have been successfully reproduced in that fashion. Another, somewhat more empirical, category of approaches reverses the process: variants are generated from the network of interest using a random rewiring procedure. The procedure selects and moves edges randomly, preserving the global number of edges, and optionally their type, as well as local properties such as the degree of each vertex. Rewirings are thus heuristic procedures which perform a sequence of local modifications on the structure of the network.

The specific focus of the present paper is on measuring the degree of cooperation between the two subgraphs of the yeast graph of interactions induced by the natural partition of edges as corresponding either to transcriptional interaction (directed) or to protein protein interaction (undirected). To evaluate a potential deviation with respect to such a measure, one needs as a first ingredient a suitable notion of random variation of the original graph. The goal is here, as in many other cases, to contrast values of a given observable on the real graph, against the distribution of those same values in the population of variants. We define [illegible] of the original graph as those graphs that are composed exactly of the original two subgraphs of interest, the variable part being the way these are 'glued' together.

From the probabilistic point of view, this notion of randomisation coincides with a traditional Erdös-Renyi statistics, except that it is conditioned by the preservation of the original subgraphs. Designing a generative random graph model that would only yield networks preserving this very precise property seems to be a hard endeavor ; it is not as easy as in the unconditional Erdös-Renyi model to draw edges step by step yet ensure that component subgraphs will be obtained in the end. Shuffling might also be seen as rewiring, except the invariant is large-scale and extremely precise: it is not edges that are moved around but entire subgraphs. Moving edges independently would break the structure of the subnetworks, and designing a sequential rewiring procedure that eventually recovers that structure is not an obvious task. Moreover, it would be in general difficult to ensure the uniformity of the sample ; see [16] for a thorough analysis of rewiring procedures. This choice of an invariant seems rather natural in that one is interested in qualifying the interplay between the original subgraphs in the original graph. Now, it is not enough to have a sensible notion of randomisation, it is also crucial to have a computational handle on it. Indeed, whatever the observable one wants to use to mark cooperation is, there is little hope of obtaining an analytic expression for its distribution, hence one needs sampling. Fortunately, it turns out it is easy to generate shuffles uniformly, since these can be described by pairs of permutations over nodes, so that one can always sample this distribution for want of an exact expression. As explained below in more details, the analysis will use two different notions of subgraph-preserving sampling: [illegible] shuffles, and [illegible] ones that also preserve the interface between our two subgraphs. Equatorial shuffles are feasible as well, and in both cases the algorithms for sampling and evaluating our measures turn out to be fast enough so that one can sweep over a not so small subset of the total population of samples.

Regarding the second necessary ingredient, namely which observable to use to measure in a meaningful way the otherwise quite vague notion of cooperation, there are again various possibilities. We use two such observables in the present study: the [illegible], defined as the percentage of disconnected pairs of nodes, and a refined quantitative version of connectivity, namely the full distance distribution between pairs of nodes. The latter is costlier, requiring about three hours of computation for each sample on a standard personal computer.

Once we have both our notion of randomisations and our observables in place, together with a feasible way of sampling the distribution of the latter, we can start. Specifically we run four experiments, using general or equatorial shuffling, and crude or refined connectivity measures. The sampling process allows us to compare the values of these measures for the original graph with the mean value for the sample, and, based on the assumption that those values follow a normal distribution over the sample, one can also provide a *p*-value that gives a rough estimate of the statistical deviation of the observable in the given graph.

The general shuffle based experiments show with significant statistical confidence that shuffling reduces connectivity (1), and at the same time contracts distances (2). More precisely, both bi-connectivity (the amount of pairs of nodes which are connected using both subgraphs) and distances are higher than average in the real network. A first interpretation might be that the real graph is trading off compactness for better bi-connectivity. In order to obtain a clearer picture and test this interpretation, we perform two other experiments using equatorial shuffles. Surprisingly, under equatorial shuffles connectivity hardly changes, while the global shift to shorter distances is still manifest. It seems therefore there is actually no trade-off, and both properties (1) and (2) have to be thought of as being independently captured by the real graph. With appropriate caution, we may try to provide a biological interpretation of this phenomenon. Since all notions of connectivity and distances are understood as directed, we propose to relate this to signalling, and interpret bi-connectivity as a measure of the capability to convey a signal between subgraphs. With this interpretation, the above properties may be read as:(1) signal flows better than average and (2) signal is more specific than average. The second point requires explanation. At constant bi-connectivity, longer average distances imply that upon receipt of a signal, the receiver has a better chance of guessing the emitter. In other words, contraction of distances (which can be easily achieved by using hubs) will anonymise signals, clearly not a desirable feature in a regulatory network. Of course this is only part of the story, since some hubs will also have an active role in signal integration and decision making. The latter is probably an incentive for compactness. If our reading of the results is on track, we then may think of the above experiments as showing that the tropism to compactness due to the need for signal integration, is weaker than the one needed for signal specificity.

Beyond the particular example we chose to develop here because of the wealth of knowledge available on the yeast regulatory and protein interaction networks, one can think of many other applications of the shuffling methodology for heterogeneous networks. The analyses performed here rely on edges corresponding to different types of experimental measurements, but edges could also represent different types of predicted functional links. Indeed, there are many situations where a biological network of interactions can be naturally seen as heterogeneous. Besides, the notions of shuffle we propose can also accomodate the case where one would use a partition of nodes, perhaps given by clustering, or localisation, or indeed any relevant biological information, and they may therefore prove useful in other scenarios.

The paper is organised as follows: first, we set up the definitions of edge-based general and equatorial shuffles based, and also consider briefly node-based shuffles though these are not used in the sequel; then we describe the interaction network of interest and the way it was obtained; finally we define our observables and experiments, and interpret them. In the conclusion, we discuss generalization and potential applications of the method. The paper ends with an appendix on the algorithmical aspects of the experiments, and a brief recall of the elementary notions of statistics we use to assert their significance.

2 Shuffles

Let $G = (V, E)$ be a directed graph, where V is a finite set of nodes, and E is a finite set of directed edges over V. We write M for the incidence matrix associated to G. Since G is directed, M may not be symmetric. In the absence of parallel edges M has coefficients in $\{0, 1\}$, where parallel edges are allowed.

Given such a matrix M and a permutation σ over V, one writes $M\sigma$ for the matrix defined as for all u, v in V:

$$M\sigma(u, v) := M(\sigma^{-1}u, \sigma^{-1}v)$$

Note that $M\sigma$ defines the same abstract graph as M does, since all σ does is changing the nodes names.

2.1 Shuffles Induced by Properties on Edges

We consider first shuffles induced by properties on edges. Suppose given a partition of $E = \sum E_i$; this is equivalent to giving a map $\kappa : E \to \{1, \dots, p\}$ which one can think of as colouring edges.

Define M_i as the incidence matrix over V containing the edges in E_i (of colour i).

Define also V_I, where $I \subseteq \{1, \dots, p\}$, as the subset of nodes v having for each $i \in I$ at least one edge incident to v with colour i, and no incident edge coloured j for $j \notin I$. We abuse notation and still write $\kappa(u) = I$ when $u \in V_I$. This represents the set of colours seen by the nodes.

Clearly $V = \sum V_I$, $V_\varnothing$ is the set of isolated nodes of G, and the set of nodes of G_i is the union of the graphs generated by V_I, for $i \in I$.

Given σ_1, …, σ_p permutations over V, define the [illegible] of M as:

$$M(\sigma_1, \dots, \sigma_p) := \textstyle\sum_i M_i \sigma_i$$

The preceding definition of $M\sigma$ is the particular case where $p = 1$ (one has only one colour common to all edges). Each G_i (the abstract graph associated to M_i) is preserved up to isomorphism under this transformation. However the way the G_is are glued together is not, since one uses a different local shuffle on each.

For moral comfort, we can check that any means of glueing together the G_is is obtainable using a general shuffle in the following sense: given G' and $\sum q_i : \sum G_i \to G'$ where the disjoint sum $\sum_i q_i$ is an isomorphism on edges, one has that G' is a general shuffle of G. To see this, define $\sigma_i(u) := q_i p_i^{-1}(u)$ if $u \in \kappa^{-1}(i)$, $\sigma_i(u) = u$ else (we have written p_i for the inclusion of G_i in G), one then has $G' = \sum G_i \sigma_i = G(\sigma_1, \dots, \sigma_p)$.

Note also that $(M(\sigma_1, \dots, \sigma_p))\tau := \sum_i M_i(\tau\sigma_i)$, and so in particular, without loss of generality one can take any the σ_i's to be the identity (just take $\tau = \sigma_i^{-1}$). This is useful when doing actual computations, and avoids some redundancy in generating samples.

An additional definition will help us refine the typology of shuffles. One says a shuffle $M\sigma$ is [illegible] if in addition for all I, and all i, V_I is closed under σ_i. Equivalently, one can ask that $\kappa \circ \sigma_i = \kappa$. An [illegible] preserves the set of colours associated with each node and in particular preserves for a given pair of nodes (u, v) the fact that (u, v) is heterochromatic, [illegible], $\kappa(u) \cap \kappa(v) = \varnothing$. This in turn implies that the distance between u and v must be realised by a path which uses edges of different colours. In the application such paths are mixing different types of interaction, and are therefore of particular interest; without preserving this attribute, an observable based on path with different colours would not make sense. In the particular case of two colours, nodes at the 'equator', having both colours, will be globally preserved, hence the name.

2.2 Shuffles Induced by Properties on Vertices

One can also consider briefly shuffles induced by properties on nodes. Suppose then given a partition of nodes $V = \sum_i V_i$, again that can be thought of as a colouring of nodes $\kappa : V \to \{1, \ldots, p\}$, and extended naturally to the assignment of one or two colours to each edge.

A node shuffle is defined as a shuffle associated to σ which can be decomposed as $\sum_i \sigma_i$, σ_i being a permutation over each cluster V_i. Clearly each graph G_i generated by V_i is invariant under the transformation: only the inter-cluster connectivity is modified.

The equivalent of the equatorial constraint would be to require in addition $\sigma(u) \in \partial V_i$ if $u \in \partial V_i$, where ∂V_i is defined as those nodes of V_i with an edge to some V_j, $i \neq j$. Other variants are possible and the choice of the specific variant will likely depend on the particular case study. We now turn to the description of the network the shuffle experiments will be applied to.

3 A Combined Network of Regulatory and Protein-Protein Interactions in Yeast

With our definitions in place, we can now illustrate the approach on a heterogeneous network obtained by glueing two component networks.

It is known that regulatory influences, including those inferred from expression data analysis or genetic experiments, are implemented by the cell through a combination of direct regulatory interactions and protein-protein interactions, which propagate signals and modulate the activity level of transcription factors. The detailed principles underlying that implementation are not well understood, but one guiding property is the fact that protein interaction and transcriptional regulation events take place in the regulatory network at different time-scales.

In order to clarify the interplay between these two types of interactions, we have combined protein-protein (PPI) and protein-DNA (TRI, for 'transcriptional regulation interaction') interaction data coming from various sources into a heterogeneous network by glueing together these two networks on the underlying set of yeast proteins.

The data from which the composite network was built includes: 1440 protein complexes identified from the literature, through HMS-PCI or TAP [3,5], 8531 physical interactions generated using high-throughput Y2H assays [26], and 7455 direct regulatory interactions compiled from literature and from ChIP-Chip experiments [4,12], connecting a total of 6541 yeast proteins. A subnetwork of high-reliability interactions was selected, using a threshold on the confidence levels associated to each inferred interaction. For the ChIP-Chip data produced by Lee et al. [12], interactions with a p-value inferior to 3.10^{-2} were conserved ; for the Y2H data produced by Ito et al. [26], a threshold of 4.5 on the Interaction Sequence Tag was used (see [8]). The PPI network was built by connecting two proteins, in both directions, whenever there was a protein-protein or a complex interaction between the two corresponding proteins. In the case of the TRI network, an edge connects a regulator protein with its regulatee. To simplify the discussion, we will refer in the rest of the paper to the TRI graph as TRI, and to the PPI graph as PPI. With some more precision, define G as the real graph,

TRI as the subgraph induced by the set of TRI nodes, [illegible] nodes such that $TRI \in \kappa(u)$, and PPI as the subgraph induced by the set of PPI nodes.

Their respective sizes are:

$$TRI = 3387,\ PPI = 2517,\ TRI \cup PPI = 4489,\ TRI \cap PPI = 1415$$

The set of nodes $TRI \cap PPI$ of both colours is also referred to in the sequel as the [illegible] or the [illegible]. Since the object of the following is to discuss the interplay between the TRI and PPI subgraphs, the interface naturally plays an important role. A qualitative measure of the connectivity between TRI and PPI which will be useful later in the discussion, is the number of bi-connected pairs in G (these are the pairs which are connected in G, but not connected in either TRI or PPI), which is roughly $p_{bi} = 23\%$. To complete this statistical portrait of the data, we provide in figure 1 the histograms of degree distributions in the PPI and TRI networks, with in and out degrees pictured separately for the latter. Figure 1 also shows the hub size distribution for the TRI network (the PPI network has no non-trivial hubs). Note that hubs are defined as sets of nodes connected to a single node. The TRI network (here considered as unoriented) has 124 such hubs ; the histogram of the distribution of their sizes is given in figure 1.

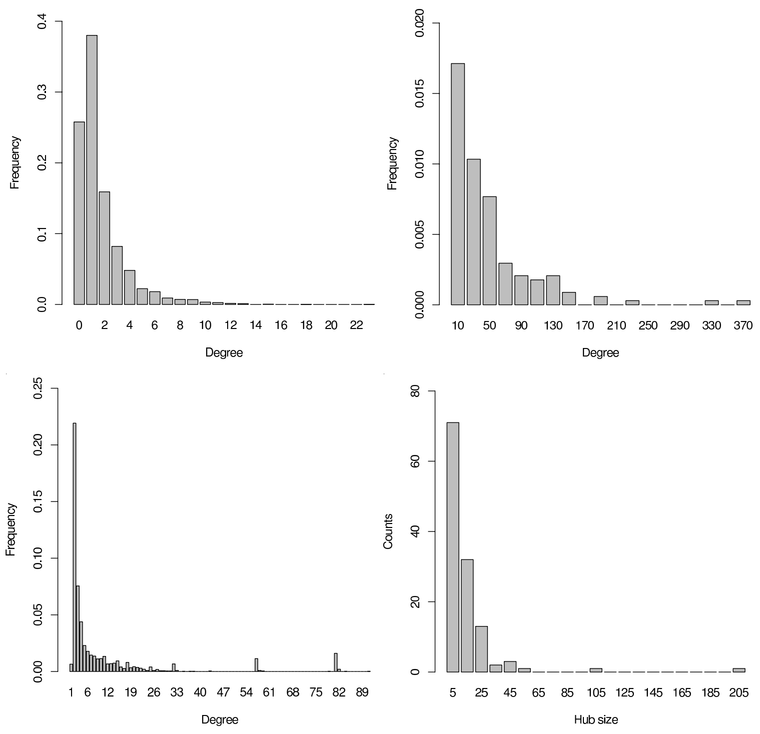


Fig. 1. First row: Histograms of the in and out degree distributions of the TRI network. Second row: Histogram of the degree distribution of the PPI network and of the distribution of the hub size in the TRI network.

4 Results and Interpretations

Hereafter, notions of connectivity, distance, etc. should be understood as [illegible] unless explicitly stated otherwise. We now turn to the various shuffle experiments and consecutive observations.

4.1 General Shuffle vs Connectivity

We take here as a rough measure of the connectivity of a graph the percentage of unconnected pairs. Comparing first the real graph with the randomised versions under the general shuffle, one finds that in the average 4% of the population pairs are disconnected under shuffle. So general shuffle disconnects, or in other words G maximises bi-connectivity.

Clearly mono-connected pairs (pairs connected in either PPI or TRI) cannot be disconnected under general shuffle; a pair is 'breakable' only if bi-connected in G; therefore a more accurate measure of the connectivity loss under general shuffle is that about 17.5% of the breakable pairs are actually broken (this obtained by dividing by p_{bi}), a rather strong deviation with a p-value below 10^{-11}.

Inasmuch as a directed path can be thought of as a signal-carrying pathway, one can interpret the above as saying that the real graph connects PPI and TRI so as to maximise the bandwidth between the subgraphs.

4.2 Equatorial Shuffle vs Connectivity

Keeping with the same observable, we now restrict to equatorial shuffles. One sees in this case that no disconnection happens, and actually about 1% more pairs are connected [illegible] shuffling. The default of connected pairs of the real graph has a far less significant p-value of 3%. However the point is that equatorial shuffles leaves bi-connectivity rather the same.

This complements the first observation and essentially says that the connectivity maximisation seen above is a property of the set of equatorial nodes ({TRI,PPI} nodes) itself, and not of the precise way TRI and PPI edges meet at the equator.

Both observations can be understood as saying that the restriction of G to the equator is a much denser subgraph than its complement (as evidenced by the connectivity loss under general shuffle), and dense enough so that equatorial shuffling does not impact connectivity.

Note that so far the observable is somewhat qualitative, being only about whether a pair is connected or not. Using a refined and quantitative version of connectivity, namely the distribution of distances (meaning for each n the proportion of pairs at distance n), will reveal more.

4.3 Impact of Equatorial Shuffles on Distance Distribution

Using this refined observable, one sees that the whole histogram shifts to the left, so equatorial shuffle contracts the graph (Fig. 2). This is confirmed by the equality between the number of lost pairs at distance 7 to 9 and the number of

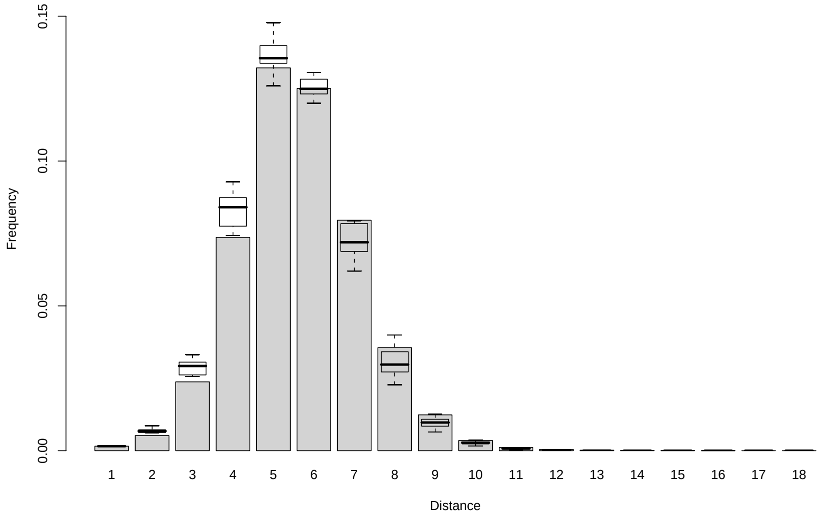


Fig. 2. Equatorial shuffle distance histogram: grey boxes stand for the real graph; one sees that shuffles have more pairs at shorter distance, and consequently (because the number of connected pairs is about the same) less such at higher distances

new ones at distance 3 to 5. In accordance with the preceding experiment, one also does not see any disconnection under equatorial shuffle.

This is to be compared with the general shuffle version (Fig. 3) where both effects are mixed, and the cumulated excess of short pairs does not account for the loss of long pairs (indeed we know 4% are broken, , disappear at infinity and are not shown on the histogram).

To summarize the distance distribution results in a single number, one can compute the deviation of the real graph mean distance under both shuffles. As expected the mean distance is higher in the real graph with respective p-values of 0.2% and 2% in the general and equatorial shuffles (see Appendix for details). We conclude that while the real graph does maximise bi-connectivity, it does not try to minimise the associated distances.

To provide an intuition on the potential interpretation of the above result, let us again consider paths as rough approximations of signalling pathways. Now compare a completely linear chain-shaped graph and star-shaped one, with the same number of nodes and edges. In the star case, any two nodes are close, at constant distance 2, while in the chain distances are longer. As said, compactness comes with a price, namely that in a star graph all signals go through the hub and are anonymised, , there may be a signal, but there is no information whatsoever in the signal about where the signal originated from. Quite the opposite happens in a linear graph. Of course this is an idealized version of the real situation; nevertheless it is tempting to interpret this last observation as an indication that the real graph is trading off fast connectivity against specificity of signals. The heterogeneous network is likely to result from a trade-off between causality and signal integration.

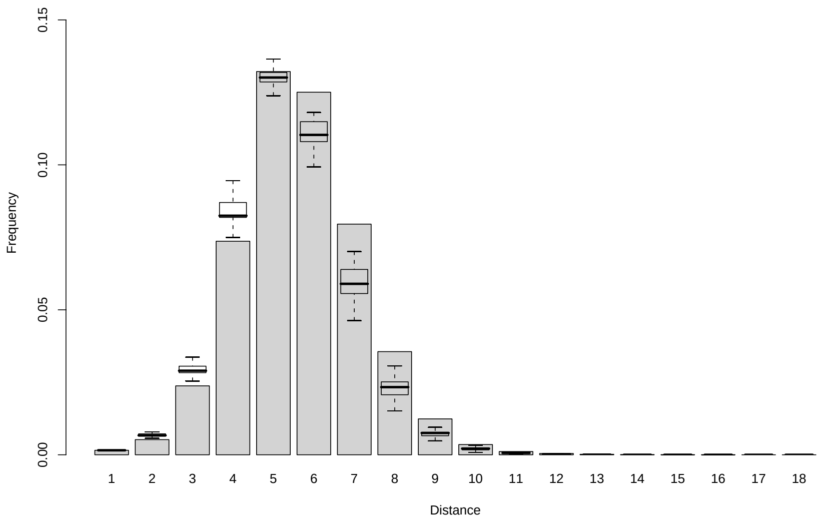


Fig. 3. Global shuffle distance histogram

As suggested in the introduction, finer observables would have to be developed to further refine this interpretation. Furthermore, there are intrinsic limits on the nature of properties that can be identified using pure topology; deeper, reliable insights about signal transmission in the joint network will ultimately require a dynamical view of signaling with corresponding experimental data.

5 Conclusion

In order to assess the cooperation between the network of protein-protein interactions and the regulatory network in yeast, we have defined two notions of shuffle, tractable randomisations of the original network that preserve global invariants. While general shuffles preserve the entire structure of the component subnetworks, equatorial shuffles also preserve the interface between the networks. We assessed cooperation between the subnetworks using two observables: the percentage of connected node pairs, and the distribution of distances between nodes. For each shuffle-observable pair, the observable in the real network was assessed against the distribution of observables in the set of network variants generated by the respective shuffle.

To summarise the results of this case study, we can say that the statistical analysis of G shuffles under the constraint of preserving its component subnetworks suggests the existence of two properties of G regarding the cooperation of its components:

- bi-connectivity, i.e. the proportion of node pairs connected only by paths using both types of edges, is higher in the actual network than in the shuffles;
- distances between pairs of nodes are higher in the actual network;

The first property can be given an interpretation in terms of bandwidth: signals flow better between the two networks than would be expected if they were connected randomly. The second property can be interpreted as favoring signal specificity: for cellular interaction networks (in contrast with telecommunication networks, for instance, where each packet carries significant intrinsic information) the information borne by a signal is very much related to the path it has followed. Longer paths thus provide more opportunity for specific signals. Note that the fact that we worked with directed notions (and not with undirected ones as we did in a first version of this paper) makes the interpretation of paths as potential signaling pathways somewhat more convincing.

We have been careful in the discussion of the results of our statistical experiments in terms of signalling capacities, and this needs to be thrashed out in subsequent work. To do so one would first need refined and yet feasible observables pertaining to the dynamics of the network of interest. A recent paper equips the subgraph induced by the major molecular players in the budding yeast cell cycle (cyclins, their inhibitors, and major complexes) with a discrete Boolean dynamics [13], and obtains a dynamics with a stable state corresponding to the G_1 phase, which is attracting a significantly higher number of states than a random graph (with the same number of nodes and edges) would. It seems therefore possible to explicitly construct signal-related observables. However there are several problems: first, this analysis relies on sorting positive and negative regulation edges, and that is an information which one doesn't have for the full graph; second it also critically relies on the rather small size of the subgraph; finally the model only handles a limited number of signals (corresponding to the various cell cycle phases). Nevertheless, a comparable study, using shuffles as a means of randomising, and confined to a well-chosen subgraph could help in qualifying our speculative interpretation of the contraction phenomenon we have observed.

On the methodological front, both the general notion of shuffle and the restricted notion of equatorial shuffle proved useful: they reveal different properties and complement one another. The same holds for the pair of observables: both the qualitative connectivity observable and its refined distance-based version are useful, and yield different and complementary insights on the cooperation between the two component networks.

We believe that the shuffling methodology developed for this case study has general applicability to the study of heterogeneous biological networks, i.e. networks that can be seen as the "glueing" of two or more component networks. Shuffles preserve global invariant properties (the structure of component networks), and define rigorously and unambiguously the class of networks which obey these properties. They are also easily computable and can be generated uniformly, by drawing from a set of acceptable permutations. Note that the latter property is in contrast with randomizations based on sequential rewiring strategies, where each rewiring step perturbs the structure while preserving one or more local invariants. While these approaches may prove to be asymptotically equivalent in some cases, they typically do not provide a direct definition nor the means to uniformly sample the set of randomizations which preserve the invariant, since the order of the rewiring steps matters.

Given an interaction network between biochemical species, any biological property on edges (type of interaction, degree of confidence, localization of interaction...) or on nodes (type of entity, functional annotation, inclusion within clusters generated using a given data type and methodology, etc...) with discrete values can be used to define a heterogeneous version of that network. Then, either the type of edge shuffles used above, or shuffles preserving other categories of top-down invariants, such as the projection of a network onto a given network of abstract clusters, could be explored. Likewise, a variety of observable properties may be used to investigate cooperation between component subnetworks. Perhaps the foremost promise of the shuffling approach resides in the interplay between different shuffle-observable pairs, which allows an exploratory assessment of cooperation adapted to the heterogeneous network at hand.

References

1. William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *32nd Annual ACM Symposium on Theory of Computing*, pages 171–180, 2000.
2. P. Erdös and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:1761, 1960.
3. AC Gavin, M Bosche, R Krause, P Grandi, M Marzioch, A Bauer, J Schultz, JM Rick, AM Michon, CM Cruciat, M Remor, C Hofert, M Schelder, M Brajenovic, H Ruffner, A Merino, K Klein, M Hudak, D Dickson, T Rudi, V Gnau, A Bauch, S Bastuck, Huhse, C Leutwein, MA Heurtier, RR Copley, A Edelmann, E Querfurth, V Rybin, G Drewes, M Raida, T Bouwmeester, P Bork, B Seraphin, B Kuster, G Neubauer, and G. Superti-Furga. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868)(Jan 10):141–7., 2002.
4. N. Guelzim, S. Bottani, P. Bourgine, and F. Kepes. Topological and causal structure of the yeast transcriptional regulatory network. *Nat Genet*, 31(1):60–3, 2002.
5. Y Ho, A Gruhler, A Heilbut, GD Bader, L Moore, SL Adams, A Millar, P Taylor, K Bennett, K Boutilier, L Yang, C Wolting, I Donaldson, S Schandorff, J Shewnarane, M Vo, J Taggart, M Goudreault, B Muskat, C Alfarano, D Dewar, Z Lin, K Michalickova, AR Willems, H Sassi, PA Nielsen, KJ Rasmussen, JR Andersen, LE Johansen, LH Hansen, H Jespersen, A Podtelejnikov, E Nielsen, J Crawford, V Poulsen, BD Sorensen, J Matthiesen, RC Hendrickson, F Gleeson, T Pawson, MF Moran, D Durocher, M Mann, CW Hogue, D Figeys, and M Tyers. Systematic identification of protein complexes in saccharomyces cerevisiae by mass spectrometry. *Nature*, 415(6868)(Jan 10):180–3, 2002.
6. J. Ihmels, G. Friedlander, S. Bergmann, O. Sarig, Y. Ziv, and N. Barkai. Revealing modular organization in the yeast transcriptional network. *Nat Genet*, 31(4):370–7, 2002.
7. T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, and Y. Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proc Natl Acad Sci U S A*, 98(8):4569–74, 2001.

8. Takashi Ito, Tomoko Chiba, Ritsuko Ozawa, Masahira Yoshida, Mikio a nd Hattori, and Yoshiyuki Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein inter actome. *PNAS*, 98(8):4569–4574, 2001.
9. H. Jeong, S. P. Mason, A. L. Barabasi, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–2, 2001.
10. H. Jeong, B. Tombor, R. Albert, Z.N. Oltvai, and A.-L. Barabasi. The large-scale organization of metabolic networks. *Nature*, 407:651–654, 2000.
11. N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 2004.
12. T. I. Lee, N. J. Rinaldi, F. Robert, D. T. Odom, Z. Bar-Joseph, G. K. Gerber, N. M. Hannett, C. T. Harbison, C. M. Thompson, I. Simon, J. Zeitlinger, E. G. Jennings, H. L. Murray, D. B. Gordon, B. Ren, J. J. Wyrick, J. B. Tagne, T. L. Volkert, E. Fraenkel, D. K. Gifford, and R. A. Young. Transcriptional regulatory networks in saccharomyces cerevisiae. *Science*, 298(5594):799–804, 2002.
13. Fangting Li, Tao Long, Ying Lu, Qi Ouyang, and Chao Tang. The yeast cell-cycle network is robustly designed. *PNAS*, 101(14):11250–11255, April 2004.
14. R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–42, 2004.
15. R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–7, 2002.
16. R. Milo, S. S. Shen-Orr, S. Itzkowitz, N. Kashtan, D. Chklovskii, and U. Alon. On the uniform generation of random graphs with prescribed degree sequence. *ArXiv*, 2003.
17. M. E. Newman. Assortative mixing in networks. *Phys Rev Lett*, 89(20):208701, 2002.
18. M. E. Newman. Properties of highly clustered networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 68(2 Pt 2):026121, 2003.
19. M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 69(2 Pt 2):026113, 2004.
20. M. E. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys Rev E Stat Nonlin Soft Matter Phys*, 64(2 Pt 2):026118, 2001.
21. N. D. Price, J. A. Papin, C. H. Schilling, and B. O. Palsson. Genome-scale microbial in silico models: the constraints-based approach. *Trends Biotechnol*, 21(4):162–9, 2003.
22. E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A. L. Barabasi. Hierarchical organization of modularity in metabolic networks. *Science*, 297(5586):1551–5, 2002.
23. S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nat Genet*, 31(1):64–8, 2002.
24. S. H. Strogatz. Exploring complex networks. *Nature*, 410(6825):268–76, 2001.
25. P. Uetz, L. Giot, G. Cagney, T. A. Mansfield, R. S. Judson, J. R. Knight, D. Lockshon, V. Narayan, M. Srinivasan, P. Pochart, A. Qureshi-Emili, Y. Li, B. Godwin, D. Conover, T. Kalbfleisch, G. Vijayadamodar, M. Yang, M. Johnston, S. Fields, and J. M. Rothberg. A comprehensive analysis of protein-protein interactions in saccharomyces cerevisiae. *Nature*, 403(6770):623–7, 2000.
26. C. von Mering, R. Krause, B. Snel, M. Cornell, S. G. Oliver, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417(6887):399–403, 2002.

27. A. Wagner. The yeast protein interaction network evolves rapidly and contains few redundant duplicate genes. *Mol Biol Evol*, 18(7):1283–92, 2001.
28. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–2, 1998.
29. E. Yeger-Lotem, S. Sattath, N. Kashtan, S. Itzkovitz, R. Milo, R. Y. Pinter, U. Alon, and H. Margalit. Network motifs in integrated cellular networks of transcription-regulation and protein-protein interaction. *Proc Natl Acad Sci U S A*, 101(16):5934–9, 2004.
30. S. H. Yook, H. Jeong, A. L. Barabasi, and Y. Tu. Weighted evolving networks. *Phys Rev Lett*, 86(25):5835–8, 2001.

A Computation of the Shortest Paths Length Distribution

This section is devoted to a brief description of the algorithms and methods used to derive the various statistics used in the study of the yeast regulation and protein interaction networks.

Clearly the (i, j) coefficient of M^n is the number of oriented paths of length n connecting i to j in the graph underlying M. Since we are only interested in knowing whether two nodes are connected by an oriented path of a given length we may use a simplified matrix product defined as:

$$M^n(i,j) = \begin{cases} 1 & \text{if } \exists k \in V : M^{n-1}(i,k) = M(k,j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

which is just forgetting the numbers of connecting paths, only to remember whether there is at least one.

Furthermore, the addition of the identity matrix I to the adjacency matrix before the computation of the products gives an immediate access to the value of the cumulative distribution function of the oriented, shortest path length distances in the network. Indeed, writing $\widehat{M} = M + I$:

$$\widehat{M}^n(i,j) = \begin{cases} 1 & \text{if } \exists k \in V, M^{n-1}(i,k) = M(k,j) = 1 \\ & \text{or } \widehat{M}^{n-1}(i,j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Thus the number of 1s in $\widehat{M}(G)^n$ is the number of ordered pairs connected by at least one path of length $\leq n$, and the entire distribution is obtained when the computation reaches a fixpoint. Computing the distribution on the real PPI-TRI graph takes about 180' on a recent PC ; the distribution for the 100 shuffles were computed down in less than 10 hours on a cluster of 41 computers hosted by Genoscope.

B Statistics

This section details the definition and computation of p-values shown in the statistical results, concerning both the amount of connected pairs and the average distance.

In order to compute p-values for the deviation of the observable on the real graph from its disribution over the set of shuffled ones, we need to approximate this distribution by a Gaussian one, with mean and standard deviation fixed to the empirical values computed on the sample. This is necessary, since the rather low amount of shuffled networks (100) prevents a direct estimation of the p-value as the proportion of shuffled networks with a larger observable.

Concerning the amount of disconnected pairs, which is the first observable considered in the results, the empirical mean over the set of general shuffles is $m_g = 0.574$, and the standard deviation $s_g = 0.005$. In the case of the equatorial shuffle, the mean falls to $m_e = 0.534$, with a standard deviation of 0.002.

Assuming this average proportion is a Gaussian random variable A with those parameters, the p-value of the deviation of the average proportion of disconnected pairs in the real network from its distribution over the sample of general shuffled networks is defined as:

$$p_g = \mathbb{P}(A < m_G), \quad \textit{with } A \sim \mathcal{N}(m_g, s_g)$$

where $m_G = 0.538$ is the observed proportion of disconnected pairs in the real network. In this case, this yields $p_g = 9 \times 10^{-12}$.

Since the proportion of disconnected pairs in the real graph is higher than the average amount of disconnected pairs in the equatorially shuffled ones, one computes the p-value p_e using the upper tail of the distribution instead of the lower one:

$$p_e = \mathbb{P}(A > m_G), \quad \textit{with } A \sim \mathcal{N}(m_e, s_e)$$

so that $p_e = 0.03$.

The computation of the p-value for the deviation of the mean distance from its value on shuffled networks follows the same scheme. The mean distance in the real graph is $m_G^d = 5.66$, while its average over the set of shuffled graphs is $m_g^d = 5.38$ for the general shuffle, and $m_e^d = 5.5$ for the equatorial one. Standard deviation is $s_g^d = 0.09$ with the general shuffle, and $s_e^d = 0.08$ with the equatorial one. The p-values for these deviations are $p_g^d = 0.002$ and $p_e^d = 0.02$, respectively.

On the Computational Power of Brane Calculi[⋆]

Nadia Busi and Roberto Gorrieri

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A. Zamboni 7, I-40127 Bologna, Italy
`{busi, gorrieri}@cs.unibo.it`

Abstract. Brane calculi are a family of biologically inspired process calculi proposed in [3] for modeling the interactions of dynamically nested membranes.

In [3] a basic calculus for membranes interactions – called Phago/Exo/Pino – is proposed, whose primitives are inspired by endocytosis and exocytosis. An alternative basic calculus – called Mate/Bud/Drip and inspired by membrane fusion and fission – is also outlined and shown to be encodable in Phago/Exo/Pino in [3].

In this paper we investigate and compare the expressiveness of such two calculi w.r.t. their ability to act as computational devices.

We show that (a fragment of) the Phago/Exo/Pino calculus is Turing powerful, by providing an encoding of Random Access Machines.

On the other hand, we show the impossibility to define a "faithful" encoding of Random Access Machines in the Mate/Bud/Drip calculus, by providing a proof of the decidability of the existence of a divergent computation in Mate/Bud/Drip.

1 Introduction

Brane calculi [3] are a family of process calculi proposed for modeling the behavior of biological membranes. The formal investigation of biological membranes has been initiated by G. Păun [13], in the field of automata and formal language theory, with the definition of P systems. In a process algebraic setting, the notions of membranes and compartments are explicitly represented in BioAmbients [16], a variant of Mobile Ambients [5] based on a set of biologically inspired primitives of interaction. Brane calculi represent an evolution of BioAmbients: the main difference w.r.t. previous approaches consists in the fact that the active entities reside on membranes, and not inside membranes. In this paper we are interested in two basic instances of brane calculi proposed in [3]: the Phago/Exo/Pino (PEP) and the Mate/Bud/Drip (MBD) calculi.

The interaction primitives of PEP are inspired by *endocytosis* (the process of incorporating external material into a cell by engulfing it with the cell membrane) and *exocytosis* (the reverse process). A relevant feature of such primitives is *bitonality*, a property ensuring that there will never be a mixing of what is inside a

[⋆] Revised and full version of the extended abstract in Proc. Workshop on Computational Methods in Systems Biology, Edinburgh, April 2005.

membrane with what is outside, although external entities can be brought inside if safely wrapped by another membrane. As endocytosis can engulf an arbitrary number of membranes, it turns out to be a rather uncontrollable process. Hence, it is replaced by two simpler operations: *phagocytosis*, that is engulfing of just one external membrane, and *pinocytosis*, that is engulfing zero external membranes.

The primitives of MBD are inspired by membrane fusion (mate) and fission (mito). Because membrane fission is an uncontrollable process that can split a membrane at an arbitrary place, it is replaced by two simpler operations: *budding*, that is splitting off one internal membrane, and *dripping*, that consists in splitting off zero internal membranes. An encoding of the MBD primitives in PEP is provided in [3]. Cardelli also observed that the reverse encoding does not exist, if the encoding must preserve the nesting structure of membranes. The reason is that in MBD the maximum nesting level of membranes cannot grow during the computation, while this property does not hold for PEP.

The aim of this work is to investigate the expressiveness of PEP and MBD as computational devices. We show that a fragment of PEP, namely, the calculus comprising only the phago and exo primitives, is Turing powerful. The proof is carried out by showing how to encode a Random Access Machine [17], a well known Turing powerful formalism. As a consequence, universal termination turns out to be undecidable on PEP. As far as MBD is concerned, we show that universal termination is a decidable property. The proof of the decidability of universal termination is based on the theory of well-structured transition systems [8]. The decidability of universal termination for MBD provides an expressiveness gap between MBD and PEP, as Random Access Machines can be encoded in the second calculus, but not in the first calculus. As a corollary, we get the impossibility to provide an encoding of PEP in MBD that preserves universal termination of systems.

The paper is organized as follows: in Section 2 we present the syntax and the semantics of the two calculi; Section 3 contains the encoding of Random Access Machines in PEP, while the decidability of universal termination for MBD is presented in Section 4. Section 5 reports some conclusive remarks.

2 Brane Calculi: Syntax and Semantics

In this section we recall the syntax and the semantics of Brane Calculi [3]. A system consists of nested membranes, and a process is associated to each membrane.

Definition 1. *The set of systems is defined by the following grammar:*

$$P, Q \quad ::= \quad \diamond \mid P \circ Q \mid\ !P \mid \sigma(\!| P |\!)$$

The set of brane processes is defined by the following grammar:

$$\sigma, \tau \quad ::= \quad 0 \mid \sigma|\tau \mid\ !\sigma \mid a.\sigma$$

Variables a, b range over actions, that will be detailed later.

The term $\diamond$ represents the empty system; the parallel composition operator on systems is $\circ$. The replication operator ! denotes the parallel composition of an unbounded number of instances of a system. The term $\sigma(\!| P |\!)$ denotes the brane that performs process σ and contains system P.

The term 0 denotes the empty process, whereas $|$ is the parallel composition of processes; with $!\sigma$ we denote the parallel composition of an unbounded number of instances of process σ. Term $a.\sigma$ is a guarded process: after performing the action a, the process behaves as σ.

We adopt the following abbreviations: with a we denote $a.0$, with $(\!| P |\!)$ we denote $0(\!| P |\!)$, and with $\sigma(\!|\ |\!)$ we denote $\sigma(\!| \diamond |\!)$.

The structural congruence relation on systems and processes is defined as follows:[1]

Definition 2. *The structural congruence $\equiv$ is the least congruence relation satisfying the following axioms:*

$$\begin{array}{ll}
P \circ Q \equiv Q \circ P & \sigma \mid \tau \equiv \tau \mid \sigma \\
P \circ (Q \circ R) \equiv (P \circ Q) \circ R & \sigma \mid (\tau \mid \rho) \equiv (\sigma \mid \tau) \mid \rho \\
P \circ \diamond \equiv P & \sigma \mid 0 \equiv \sigma \\
& \\
!\diamond \equiv \diamond & !0 \equiv 0 \\
!(P \circ Q) \equiv !P \circ !Q & !(\sigma \mid \tau) \equiv !\sigma \mid !\tau \\
!!P \equiv !P & !!\sigma \equiv !\sigma \\
P \circ !P \equiv !P & \sigma \mid !\sigma \equiv !\sigma \\
& \\
0(\!| \diamond |\!) \equiv \diamond &
\end{array}$$

Definition 3. *The basic reaction rules are the following:*

$$(\texttt{par})\ \frac{P \ \rightarrow\ Q}{P \circ R \ \rightarrow\ Q \circ R} \qquad\qquad (\texttt{brane})\ \frac{P \ \rightarrow\ Q}{\sigma(\!| P |\!) \ \rightarrow\ \sigma(\!| Q |\!)}$$

$$(\texttt{strucong})\ \frac{P' \equiv P \qquad P \ \rightarrow\ Q \qquad Q \equiv Q'}{P' \ \rightarrow\ Q'}$$

Rules (`par`) and (`brane`) are the contextual rules that respectively permit to a system to execute also if it is in parallel with another process or if it is inside a membrane, respectively. Rule (`strucong`) ensures that two structurally congruent systems have the same reactions.

With $\rightarrow^*$ we denote the reflexive and transitive closure of a relation $\rightarrow$. Given a reduction relation $\rightarrow$, we say that a system P has a *divergent computation* (or infinite computation) if there exists an infinite sequence of systems $P_0, P_1, \ldots, P_i, \ldots$ such that $P = P_0$ and $\forall i \geq 0 : P_i \rightarrow P_{i+1}$. We say that a system P *universally terminates* if it has no divergent computations. We say that

[1] With abuse of notation we use $\equiv$ to denote both structural congruence on systems and structural congruence on processes.

P is *deterministic* iff for all P', P'': if $P \to P'$ and $P \to P''$ then $P' \equiv P''$. We say that P has a *terminating computation* (or a deadlock) if there exists Q such that $P \to^* Q$ and $Q \not\to$. A system P satisfies the universal termination property if P has no divergent computations. A system P satisfies the existential termination property if P has a deadlock. Note that the existential termination and the universal termination properties are equivalent on deterministic systems.

The system P' is a *derivative* of the system P if $P \to^* P'$; the set of *derivatives* of a system P is denoted by $Deriv(P)$.

We use $\prod$ (resp. $\bigcirc$) to denote the parallel composition of a set of processes (resp. systems), i.e., $\prod_{i\in\{1,\dots,n\}} \sigma_i = \sigma_1 \mid \dots \mid \sigma_n$ and $\bigcirc_{i\in\{1,\dots,n\}} P_i = P_1 \circ \dots \circ P_n$. Moreover, $\prod_{i\in\emptyset} \sigma_i = 0$ and $\bigcirc_{i\in\emptyset} P_i = \diamond$.

2.1 The Phago/Exo/Pino Calculus (PEP)

The first calculus we investigate is proposed in [3], and it is inspired by endocytosis/exocytosis. Endocytosis is the process of incorporating external material into a cell by "engulfing" it with the cell membrane, while exocytosis is the reverse process. As endocytosis can engulf an arbitrary amount of material, giving rise to an uncontrollable process, in [3] two more basic operations are used: *phagocytosis*, engulfing just one external membrane, and *pinocytosis*, engulfing zero external membranes.

Definition 4. *Let Name be a denumerable set of ambient names, ranged over by $n, m, \dots$. The set of actions of PEP is defined by the following grammar:*

$$a \ ::= \ \circlearrowleft_n \mid \circlearrowleft_n^{\perp}(\sigma) \mid \circlearrowright_n \mid \circlearrowright_n^{\perp} \mid \circledcirc(\sigma)$$

Action $\circlearrowleft_n$ denotes phagocytosis; the co-action $\circlearrowleft_n^{\perp}$ is meant to synchronize with $\circlearrowleft_n$; names n are used to pair-up related actions and co-actions. The co-phago action is equipped with a process σ, this process will be associated to the new membrane that engulfs the external membrane. Action $\circlearrowright_n$ denotes exocytosis, and synchronizes with the co-action $\circlearrowright_n^{\perp}$. Exocytosis causes an irreversible mixing of membranes. Action $\circledcirc$ denotes pinocytosis. The pino action is equipped with a process σ: this process will be associated to the new membrane, that is created inside the brane performing the pino action.

Definition 5. *The reaction relation for PEP is the least relation containing the following axioms, and satisfying the rules in Definition 3:*

$$(\texttt{phago}) \qquad \circlearrowleft_n.\sigma|\sigma_0 \llparenthesis P \rrparenthesis \ \circ \ \circlearrowleft_n^{\perp}(\rho).\tau|\tau_0 \llparenthesis Q \rrparenthesis \to \tau|\tau_0 \llparenthesis \rho \llparenthesis \sigma|\sigma_0 \llparenthesis P \rrparenthesis \rrparenthesis \circ Q \rrparenthesis$$

$$(\texttt{exo}) \qquad \circlearrowright_n^{\perp}.\tau|\tau_0 \llparenthesis \circlearrowright_n.\sigma|\sigma_0 \llparenthesis P \rrparenthesis \circ Q \rrparenthesis \to P \ \circ \ \sigma|\sigma_0|\tau|\tau_0 \llparenthesis Q \rrparenthesis$$

$$(\texttt{pino}) \qquad \circledcirc(\rho).\sigma|\sigma_0 \llparenthesis P \rrparenthesis \to \sigma|\sigma_0 \llparenthesis \rho \llparenthesis\ \rrparenthesis \circ P \rrparenthesis$$

2.2 The Mate/Bud/Drip Calculus (MBD)

The second calculus, also proposed in [3], is inspired by membrane fusion and splitting. To make membrane splitting more controllable, in [3] two more basic operations are used: *budding*, consisting in splitting off one internal membrane, and *dripping*, consisting in splitting off zero internal membranes. Membrane fusion, or merging, is called *mating*.

Definition 6. *The set of actions of MBD is defined by the following grammar:*

$$a \;\; ::= \;\; mate_n \mid mate_n^{\perp} \mid bud_n \mid bud_n^{\perp}(\sigma) \mid drip(\sigma)$$

Actions $mate_n$ and $mate_n^{\perp}$ will synchronize to obtain membrane fusion. Action bud_n permits to split one internal membrane, and synchronizes with the co-action $bud_n^{\perp}$. Action *drip* permits to split off zero internal membranes. Actions $bud^{\perp}$ and *drip* are equipped with a process σ, that will be associated to the new membrane created by the brane performing the action.

Definition 7. *The reaction relation for MBD is the least relation containing the following axioms, and satisfying the rules in Definition 3:*

$$(\texttt{mate}) \qquad mate_n.\sigma|\sigma_0(\!|\, P \,|\!) \;\circ\; mate_n^{\perp}.\tau|\tau_0(\!|\, Q \,|\!) \to \sigma|\sigma_0|\tau|\tau_0(\!|\, P \circ Q \,|\!)$$

$$(\texttt{bud}) \qquad bud_n^{\perp}(\rho).\tau|\tau_0(\!|\, bud_n.\sigma|\sigma_0(\!|\, P \,|\!) \circ Q \,|\!) \to \rho(\!|\, \sigma|\sigma_0(\!|\, P \,|\!) \,|\!) \;\circ\; \tau|\tau_0(\!|\, Q \,|\!)$$

$$(\texttt{drip}) \qquad drip(\rho).\sigma|\sigma_0(\!|\, P \,|\!) \to \rho(\!|\;|\!) \;\circ\; \sigma|\sigma_0(\!|\, P \,|\!)$$

In [3] it is shown that the operations of mating, budding and dripping can be encoded in PEP.

3 PEP Is Turing Powerful

In this section we show that a fragment of PEP, namely, the calculus without the pino action, is Turing powerful. The result is proved by showing how to model Random Access Machines (RAMs) [17], a well known Turing powerful formalism. A direct consequence of this result is the undecidability of universal termination for PEP. We start by recalling what RAMs are.

3.1 Random Access Machines

RAMs are a computational model based on finite programs acting on a finite set of registers. More precisely, a RAM R is composed of the registers $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a sequence of indexed instructions $(1 : I_1), \ldots, (m : I_m)$. In [12] it is shown that the following two instructions are sufficient to model every recursive function:

– $(i : Succ(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;

– $(i : \mathit{DecJump}(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the first instruction and it continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached.

A state of a RAM is modelled by $(i, c_1, \ldots, c_n)$, where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current contents of the registers $r_1, \ldots, r_n$, respectively. We use the notation $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ to denote that the state of the RAM R changes from $(i, c_1, \ldots, c_n)$ to $(i', c'_1, \ldots, c'_n)$, as a consequence of the execution of the i-th instruction.

A state $(i, c_1, \ldots, c_n)$ is *terminated* if the program counter i is strictly greater than the number of instructions m. We say that a RAM R *terminates* if its computation reaches a terminated state.

3.2 Modelling RAMs in PEP

In this section we show how to model RAMs in PEP. The modelling of RAMs is based on an encoding function, which transforms instructions and registers independently.

The basic idea for modelling the natural numbers contained in the registers is the following: the natural number n is represented by the nesting of $2n + 1$ branes. The increment is performed by producing a new membrane that performs a phago on the representation of n, while a decrement is performed by executing an exo of the membrane representing $n - 1$, that lies inside the membrane representing n.

Consider a RAM R with instructions $(1 : I_1)$, …, $(m : I_m)$ and registers r_1, …, r_n; the encoding of an initial state $(1, c_1, \ldots, c_n)$ is defined as follows:

$$\begin{aligned}\llbracket(1, c_1, \ldots, c_n)\rrbracket = \circlearrowleft_{p_1}(\!|\ |\!) \circ ! \ \llbracket(1 : I_1)\rrbracket \ \circ \ \ldots \ \circ \ ! \ \llbracket(m : I_m)\rrbracket \ \circ \\ \llbracket r_1 = c_1\rrbracket \ \circ \ \ldots \ \circ \ \llbracket r_n = c_n\rrbracket \ \circ \ \mathit{GARB}(\!|\ |\!)\end{aligned}$$

where

$$\mathit{GARB} = ! \circlearrowleft^{\perp}_{exitpc} \mid ! \circlearrowleft^{\perp}_{garbage}(\circlearrowleft^{\perp}_{exitpc})$$

is the process on the garbage collector membrane.

The encoding of an initial state of the RAM is composed by the following parts: $\circlearrowleft_{p_1}(\!|\ |\!)$, representing the program counter, (an unbounded number of occurrences of) the encodings of each instruction, the encodings of the initial contents of registers, and a garbage membrane used to collect no longer used membranes and to inhibit their actions.

The presence of a "program counter" system $\circlearrowleft_{p_i}(\!|\ |\!)$ denotes the fact that the next instruction to be executed is I_i.

The encoding of register r_j with content zero is

$$\llbracket r_j = 0 \rrbracket = Z_j \llparenthesis\ \rrparenthesis$$

where

$$Z_j = ! \circlearrowright_{inc_j} \mid ! \circlearrowright^{\perp}_{decreq_j}(\circlearrowleft^{\perp}_{z}).\circlearrowleft^{\perp}_{exitpc}$$

If an increment operation on r_j is executed, then – by action $\circlearrowright_{inc_j}$ – the system $\llbracket r_j = 0 \rrbracket$ is engulfed in a new membrane, thus obtaining a representation of $\llbracket r_j = 1 \rrbracket$.

If the membrane of system $\llbracket r_j = 0 \rrbracket$ is entered by a request for decrement or jump – action $\circlearrowright^{\perp}_{decreq_j}$ – the choice corresponding to the zero case is selected, and the program counter corresponding to the jump is expelled by $\circlearrowleft^{\perp}_{exitpc}$.

The encoding of register r_j with content $n+1$ is

$$\llbracket r_j = n+1 \rrbracket = S_j \llparenthesis \circlearrowleft_{exitreg} \llparenthesis\ \llbracket r_j = n \rrbracket\ \rrparenthesis \rrparenthesis$$

where

$$S_j = ! \circlearrowright_{inc_j} \mid \circlearrowright^{\perp}_{decreq_j}(\circlearrowleft^{\perp}_{nz}).\circlearrowleft^{\perp}_{exitreg}.\circlearrowright_{garbage}.\circlearrowleft^{\perp}_{exitpc}$$

The case of an increment operation is treated in the same way as in the encoding of $r_j = 0$.

If the membrane of system $r_j = n+1$ is entered by a request for decrement or jump, then the choice corresponding to the non-zero case is selected by $\circlearrowleft^{\perp}_{nz}$, and the membrane representing $r_j = n$ is expelled by $\circlearrowleft^{\perp}_{exitreg}$. At this point, the no longer used external membrane of the system $\llbracket r_j = n+1 \rrbracket$ is engulfed by the garbage collector membrane, by $\circlearrowright_{garbage}$. Finally, the program counter corresponding to the next instruction is expelled by $\circlearrowleft^{\perp}_{exitpc}$.

Note that it is necessary to engulf the external membrane of the system $\llbracket r_j = n+1 \rrbracket$ in the garbage collector, to inhibit the possibility for the process $! \circlearrowright_{inc_j}$ to capture a subsequent request for increment of register r_j.

The encoding for the instruction $(i : I_i)$ is as follows:

$$\begin{aligned}
\llbracket (i : Succ(r_j)) \rrbracket &= \circlearrowright^{\perp}_{p_i}(0).\circlearrowright^{\perp}_{inc_j}(\circlearrowleft_{exitreg}).\circlearrowleft^{\perp}_{exitpc}.S_j \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowright_{p_{i+1}} \llparenthesis\ \rrparenthesis \rrparenthesis \rrparenthesis \\
\llbracket (i : DecJump(r_j, s)) \rrbracket &= \circlearrowright^{\perp}_{p_i}(0).\circlearrowright_{decreq_j}.J_{i+1,s}
\end{aligned}$$

where

$$\begin{aligned}
J_{h,k} = \circlearrowleft^{\perp}_{znz} \llparenthesis \circlearrowleft_{znz} \llparenthesis\ & \circlearrowleft_{nz}\ \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowright_{p_h} \llparenthesis\ \rrparenthesis \rrparenthesis \rrparenthesis \rrparenthesis \rrparenthesis \rrparenthesis \quad \circ \\
& \circlearrowleft_{z}\ \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowright_{p_k} \llparenthesis\ \rrparenthesis \rrparenthesis \rrparenthesis\ \rrparenthesis\ \rrparenthesis
\end{aligned}$$

The encoding of each instruction consists in a membrane, and the encoding of a RAM contains an unbounded number of copies of the encoding of each instruction.

When a program counter system $\circlearrowright_{p_i} \llparenthesis\ \rrparenthesis$ appears at top-level, an (occurrence of) instruction $(i : I_i)$ is activated by engulfing the program counter.

If the i-th instruction is an increment of register r_j, then the external membrane of the encoding of such an instruction will become the new external membrane of the encoding of the contents of register r_j. To this aim, the external membrane of $[\![(i : Succ(r_j))]\!]$ engulfs the system $[\![r_j = k]\!]$ – where k is the current contents of r_j – by $\circlearrowleft^{\perp}_{inc_j}$; $[\![r_j = k]\!]$ is wrapped with a membrane decorated with process $\circlearrowleft_{exitreg}$, that will permit to $[\![r_j = k]\!]$ to be expelled when a decrement operation will be performed on system $[\![r_j = k + 1]\!]$. At this point, the internal program counter $\circlearrowleft_{p_{i+1}}(\!|\ |\!)$ is expelled to the top-level, by action $\circlearrowleft^{\perp}_{exitpc}$, and the process associated to the external brane will behave as S_j, i.e., the process on the external brane of $[\![r_j = k + 1]\!]$.

Suppose that the i-th instruction is a decrement of register r_j, or jump to instruction s if the contents of r_j is zero.

If the contents of r_j is not zero, e.g., $r_j = k + 1$, then it is necessary to remove the two outer membranes of $[\![r_j = k + 1]\!]$, or, in other words, to permit to $[\![r_j = k]\!]$ to be expelled from the system $[\![r_j = k + 1]\!]$. Moreover, the program counter $\circlearrowleft_{p_{i+1}}(\!|\ |\!)$ should be produced at top-level.

On the other hand, if the contents of r_j is zero, the program counter $\circlearrowleft_{p_s}(\!|\ |\!)$ should be produced.

In both cases, the membrane representing the i-th instruction is engulfed by the encoding of register r_j, by performing action $\circlearrowleft_{decreq_j}$. The system engulfed by the encoding of r_j is $J_{i+1,s}$, and essentially permits to perform a choice between the two program counters $\circlearrowleft_{p_{i+1}}(\!|\ |\!)$ and $\circlearrowleft_{p_s}(\!|\ |\!)$. After entering the encoding of r_j, $J_{i+1,s}$ evolves to $\circlearrowleft_{nz}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{p_{i+1}}(\!|\ |\!)|\!)|\!)|\!)|\!) \circ \circlearrowleft_{z}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{p_s}(\!|\ |\!)|\!)|\!)$. If $r_j = 0$, and hence the encoding of the i-th instruction has been engulfed by $[\![r_j = 0]\!]$, then $J_{i+1,s}$ has been engulfed by an internal membrane with associated process $\circlearrowleft^{\perp}_{z}$, and only the program counter $\circlearrowleft_{p_s}(\!|\ |\!)$ will be expelled. (The other program counter will remain forever inside the system $[\![r_j = 0]\!]$, surrounded by a membrane with an associated empty process.)

On the other hand, if $r_j > 0$, then $J_{i+1,s}$ has been engulfed by an internal membrane with associated process $\circlearrowleft^{\perp}_{nz}$, and only the the program counter $\circlearrowleft_{p_{i+1}}(\!|\ |\!)$ will be expelled.

3.3 Correctness of the Encoding

In this section we show that our encoding of RAMs preserves universal termination. In the previous section, we noted that during the computation some innocuous garbage is created inside the garbage collector membrane and inside the membrane of the system $[\![r_j = 0]\!]$.

To show the correctness of the encoding we need to keep this additional garbage systems into account; hence, we define the encoding $[\![\![\ ]\!]\!]$, mapping a state of the RAM on a set of processes.

The set $[\![\![r_j = n]\!]\!]$ contains all the processes that are equal to the encoding of registers, but for the presence of some innocuous garbage inside the inner membrane.

Definition 8. *Let R be a system.*

We say that $R \in [\![\![r_j = 0]\!]\!]$ iff there exists I such that $R \equiv Z_j(\!|GZ_I|\!)$ where $GZ_I = \bigcirc_{i\in I} 0(\!|\circlearrowleft_{nz}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{p_{h_i}}(\!|\ |\!)|\!)|\!)|\!)|\!)|\!)$ and $Z_j = !\circlearrowleft_{inc_j} \mid !\circlearrowleft^{\perp}_{decreq_j}(\circlearrowleft^{\perp}_{z}).\circlearrowleft^{\perp}_{exitpc}$.

We say that $R \in \llbracket r_j = n+1 \rrbracket$ *iff there exists* $R' \in \llbracket r_j = n \rrbracket$ *such that*
$R \;\equiv\; S_j (\!| {\nwarrow}\!\!\circlearrowleft_{exitreg} (\!|\, R' |\!)\, |\!)$
and $S_j = !\,{\searrow}\!\!\circlearrowleft_{inc_j} \mid {\searrow}\!\!\circlearrowleft^{\perp}_{decreq_j}({\nwarrow}\!\!\circlearrowleft^{\perp}_{nz}).{\nwarrow}\!\!\circlearrowleft^{\perp}_{exitreg}.{\searrow}\!\!\circlearrowleft_{garbage}.{\nwarrow}\!\!\circlearrowleft^{\perp}_{exitpc}.$

Given a state $(i, c_1, \ldots, c_n)$, with $[\![\![(i, c_1, \ldots, c_n)]\!]\!]$ we denote the set of processes that are equal to process $\llbracket (i, c_1, \ldots, c_n) \rrbracket$ but for the presence of some innocuous garbage.

Definition 9. *Let* P *be a system.*
We say that $P \in [\![\![(i, c_1, \ldots, c_n)]\!]\!]$ *iff there exist* $J, R_1, \ldots, R_n$ *such that*
$R_i \in \llbracket r_i = c_i \rrbracket$ *for* $i = 1, \ldots, n$ *and*
$P \equiv {\searrow}\!\!\circlearrowleft_{p_i}(\!|\;|\!) \circ !\ \llbracket (1 : I_1) \rrbracket \ \circ\ \ldots\ \circ\ !\ \llbracket (m : I_m) \rrbracket \ \circ$
$\quad R_1 \circ\ \ldots\ \circ\ R_n\ \circ\ GARB(\!|\ GG_J\ |\!)$
where $GG_J = \bigcirc_{j \in J} 0 (\!| !\,{\searrow}\!\!\circlearrowleft_{inc_j} (\!| {\nwarrow}\!\!\circlearrowleft_z (\!| {\nwarrow}\!\!\circlearrowleft_{exitpc} (\!| {\searrow}\!\!\circlearrowleft_{p_{k_j}} (\!|\;|\!) |\!) |\!) |\!) |\!).$

As the encoding of the initial state of a RAM defined in the previous section does not contain any garbage, it is easy to see that it belongs to the set $[\![\![\;]\!]\!]$ of encodings corresponding to the initial state:

Proposition 1. *Let* R *be a RAM with program* $(1 : I_1), \ldots, (m : I_m)$ *and initial state* $(1, c_1, \ldots, c_n)$*. Then* $\llbracket (1, c_1, \ldots, c_n) \rrbracket \in [\![\![(1, c_1, \ldots, c_n)]\!]\!]$.

We show that a system belonging to the set of encodings of a RAM state is able to mimic a computational step of the RAM by a (nonempty) sequence of steps.

Lemma 1. *Let* R *be a RAM with program* $(1 : I_1), \ldots, (m : I_m)$ *and initial state* $(1, c_1, \ldots, c_n)$*. Let* $(i, c_1, \ldots, c_n)$ *be a state of* R*. If* $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ *then for all systems* $P \in [\![\![(i, c_1, \ldots, c_n)]\!]\!]$ *there exists* $Q \in [\![\![(i', c'_1, \ldots, c'_n)]\!]\!]$ *such that* $P \rightarrow^{+} Q$.

Proof. The proof is by case analysis. Four cases can happen:

1. the ith instruction is an increment on register r_j and $c_j = 0$;
2. the ith instruction is an increment on register r_j and $c_j > 0$;
3. the ith instruction is a decrement on register r_j and $c_j = 0$;
4. the ith instruction is a decrement on register r_j and $c_j > 0$.

We report only the first case.
Suppose that the ith instruction is an increment on r_j and $c_j = 0$.
Hence, $i' = i + 1$, $c'_j = 1$ and $c'_i = c_i$ for $i = 1, \ldots, n$ and $i \neq j$.
As $P \in [\![\![(i, c_1, \ldots, c_n)]\!]\!]$, we have that exist $J, R_1, \ldots, R_n$ such that $R_i \in \llbracket r_i = c_i \rrbracket$ for $i = 1, \ldots, n$ and
$P \;\equiv\; {\searrow}\!\!\circlearrowleft_{p_i}(\!|\;|\!) \circ !\ \llbracket (1 : I_1) \rrbracket \ \circ\ \ldots\ \circ\ !\ \llbracket (m : I_m) \rrbracket \ \circ$
$\quad R_1 \circ\ \ldots\ \circ\ R_n\ \circ\ GARB(\!|\ GG_J\ |\!)$
with $GG_J = \bigcirc_{j \in J} 0 (\!| !\,{\searrow}\!\!\circlearrowleft_{inc_j} (\!| {\nwarrow}\!\!\circlearrowleft_z (\!| {\nwarrow}\!\!\circlearrowleft_{exitpc} (\!| {\searrow}\!\!\circlearrowleft_{p_{k_j}} (\!|\;|\!) |\!) |\!) |\!) |\!).$
As the ith instruction is an increment on register r_j, we have that $\llbracket (i : I_i) \rrbracket = {\searrow}\!\!\circlearrowleft^{\perp}_{p_i}(0).{\searrow}\!\!\circlearrowleft^{\perp}_{inc_j}({\nwarrow}\!\!\circlearrowleft_{exitreg}).{\nwarrow}\!\!\circlearrowleft^{\perp}_{exitpc}.S_j (\!| {\nwarrow}\!\!\circlearrowleft_{exitpc} (\!| {\searrow}\!\!\circlearrowleft_{p_{i+1}} |\!) |\!).$
Hence the program counter system ${\searrow}\!\!\circlearrowleft_{p_i}(\!|\;|\!)$ is engulfed by $\llbracket (i : I_i) \rrbracket$. Formally, $P \rightarrow P_1$ where

$$\begin{aligned}P_1 = {} & \circlearrowright^{\perp}_{inc_j}(\circlearrowleft_{exitreg}).\circlearrowright^{\perp}_{exitpc}.S_j(\!|\circlearrowleft_{exitpc}(\!|\circlearrowright_{p_{i+1}}(\!|\ |\!)|\!)|\!)\circ \\ & !\ [\![(1:I_1)]\!]\ \circ\ \ldots\ \circ\ !\ [\![(m:I_m)]\!]\ \circ \\ & R_1\circ\ \ldots\ \circ\ R_n\ \circ\ GARB(\!|\ GG_J\ |\!)\end{aligned}.$$

As $R_j\in[\![r_j=c_j]\!]$, we have that there exists I such that $R_j\ \equiv\ Z_j(\!|GZ_I|\!)$, with $GZ_I=\bigcirc_{i\in I}0(\!|\circlearrowleft_{nz}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowleft_{exitpc}(\!|\circlearrowright_{p_{h_i}}(\!|\ |\!)|\!)|\!)|\!)|\!)|\!)$ and $Z_j=!\circlearrowright_{inc_j}\ |\ !\circlearrowright^{\perp}_{decreq_j}(\circlearrowright^{\perp}_{z}).\circlearrowright^{\perp}_{exitpc}$.

By performing action $\circlearrowright^{\perp}_{inc_j}(\circlearrowleft_{exitreg})$, the external membrane of the encoding of the ith instruction engulfs system R_j. Formally, $P_1\rightarrow P_2$ with

$$\begin{aligned}P_2 = {} & \circlearrowright^{\perp}_{exitpc}.S_j(\!|\circlearrowleft_{exitpc}(\!|\circlearrowright_{p_{i+1}}(\!|\ |\!)|\!)\circ\circlearrowleft_{exitreg}(\!|R_j|\!)|\!)\circ \\ & !\ [\![(1:I_1)]\!]\ \circ\ \ldots\ \circ\ !\ [\![(m:I_m)]\!]\ \circ \\ & R_1\circ\ \ldots R_{j-1}\ \circ\ R_{j+1}\ldots\ \circ\ R_n\ \circ\ GARB(\!|\ GG_J\ |\!)\end{aligned}$$

By performing $\circlearrowright^{\perp}_{exitpc}$, the new program counter $\circlearrowright_{p_{i+1}}(\!|\ |\!)$ is expelled; hence $P_2\rightarrow P_3$ with

$$\begin{aligned}P_3 = {} & \circlearrowright_{p_{i+1}}(\!|\ |\!)\circ S_j(\!|\circlearrowleft_{exitreg}(\!|R_j|\!)\ |\!)\circ \\ & !\ [\![(1:I_1)]\!]\ \circ\ \ldots\ \circ\ !\ [\![(m:I_m)]\!]\ \circ \\ & R_1\circ\ \ldots R_{j-1}\ \circ\ R_{j+1}\ldots\ \circ\ R_n\ \circ\ GARB(\!|\ GG_J\ |\!)\end{aligned}$$

As $R_j\in[\![r_j=0]\!]$, we have that $S_j(\!|\circlearrowleft_{exitreg}(\!|R_j|\!)\ |\!)\in[\![r_j=1]\!]$; hence, $P_3\in[\![(i+1,c_1,\ldots,c_{j-1},1,c_{j+1},\ldots,c_n)]\!]$.

Summing up, we have that $(i,c_1,\ldots,c_{j-1},0,c_{j+1},\ldots,c_n)\rightarrow_R(i+1,c_1,\ldots,c_{j-1},1,c_{j+1},\ldots,c_n)$ and $P\rightarrow^+P_3$ with $P_3\in[\![(i+1,c_1,\ldots,c_{j-1},1,c_{j+1},\ldots,c_n)]\!]$.

We show that a sufficiently long sequence of computational steps of a weak encoding of a RAM state corresponds to one (or more) steps of the RAM.

Lemma 2. *Let R be a RAM with program $(1:I_1),\ldots,(m:I_m)$ and initial state $(1,c_1,\ldots,c_n)$. Let $(i,c_1,\ldots,c_n)$ be a state of R and $P\in[\![(i,c_1,\ldots,c_n)]\!]$. If $P\rightarrow P_1\rightarrow\ldots\rightarrow P_j\rightarrow\ldots$ then one of the following holds:*

- *the ith instruction is $(i:Succ(r_j))$ and $P_3\in[\![(i+1,c_1,\ldots,c_j+1,\ldots,c_n)]\!]$;*
- *the ith instruction is $(i:DecJump(r_j,s))$, $c_j=0$ and $P_5\in[\![(s,c_1,\ldots,c_n)]\!]$;*
- *the ith instruction is $(i:DecJump(r_j,s))$, $c_j>0$ and $P_9\in[\![(i+1,c_1,\ldots,c_j-1,\ldots,c_n)]\!]$.*

Proof. (Sketch) The proof is by case analysis.

It is easy to see that the systems contained in the garbage collector membrane, as well as the systems in the membrane of the encoding of registers with content zero, are innocuous and can perform no move.

Concerning the first two cases, the first three (resp. five) steps of computation are univocally determined and lead to a system belonging to the set of encodings of the next state of the RAM.

In the last case, some nondeterminism is introduced by the interleaving of actions $\circlearrowright^{\perp}_{znz}$ and $\circlearrowright^{\perp}_{nz}$ – performed by the system encoding the DecJump instruction – with actions $\circlearrowleft^{\perp}_{exitreg}$ and $\circlearrowright_{garbage}$ – performed by the encoding of register r_j. Whatever execution order is chosen, after the execution of the four actions mentioned above the reached system is the same.

We can now conclude with the Theorem which states that our modelling of RAMs preserves universal termination.

Theorem 1. *Let R be a RAM with program $(1 : I_1), \ldots, (m : I_m)$ and initial state $(1, c_1, \ldots, c_n)$. Then we have that the RAM R terminates if and only if all computations of the system $\llbracket(i, c_1, \ldots, c_n)\rrbracket$ terminate.*

An immediate consequence is the undecidability of universal termination:

Corollary 1. *The universal termination property is undecidable for PEP systems.*

The above theorem provides no information on the decidability of existential termination: if the RAM does not terminate, we only deduce that there exists at least one divergent computation starting from the encoding of the RAM.

A first possibility to prove the undecidability of existential termination consists in showing that the encoding presented in this section is *uniform w.r.t. termination.* A process P is uniform w.r.t. termination if the following property holds: if P has a terminating computation, then all computations starting from P terminate. An encoding satisfying this uniformity property provides a faithful modeling of the behaviour of the RAM: if the RAM terminates (resp. diverges) then all the computations of the encoding terminate (resp. diverge).

An alternative possibility consists in providing a deterministic encoding of RAMs; as for deterministic systems existential and universal termination are equivalent properties, the undecidability of existential termination directly follows from Corollary 1.

3.4 A Deterministic Encoding of RAMs

In this section we show how to obtain a deterministic encoding of RAMs by a slight modification of the encoding presented in section 3.2. In the previous encoding some nondeterminism is present in the execution of a decrement operation. After the membrane of the DecJump instruction entered the encoding of the register to decrement, the action $\circlearrowright^{\perp}_{znz}$ (performed by the encoding of the instruction) can be executed in parallel with actions $\circlearrowright^{\perp}_{exitreg}$ and $\circlearrowright_{garb}$ (performed by the encoding of register). Here we introduce a synchronization membrane that forces a sequential execution of the two sequences of actions mentioned above.

Consider a RAM R with instructions $(1 : I_1)$, …, $(m : I_m)$ and registers r_1, …, r_n; the encoding of an initial state $(1, c_1, \ldots, c_n)$ is defined as follows:

$$\begin{aligned}\langle\!\langle(1, c_1, \ldots, c_n)\rangle\!\rangle = \circlearrowright_{p_1}(\!|\;|\!) \circ ! \; \langle\!\langle(1 : I_1)\rangle\!\rangle \;\circ\; \ldots \;\circ\; ! \; \langle\!\langle(m : I_m)\rangle\!\rangle \;\circ \\ \langle\!\langle r_1 = c_1\rangle\!\rangle \;\circ\; \ldots \;\circ\; \langle\!\langle r_n = c_n\rangle\!\rangle \;\circ\; GARB(\!|\;|\!)\end{aligned}$$

where $GARB = ! \circlearrowright^{\perp}_{exitpc} \mid ! \circlearrowright^{\perp}_{garbage}(\circlearrowright^{\perp}_{exitpc})$ is the process on the garbage collector membrane.

The encoding of register r_j with content zero is $\langle\!\langle r_j = 0\rangle\!\rangle = Z_j(\!|\;|\!)$, where $Z_j = ! \circlearrowright_{inc_j} \mid ! \circlearrowright^{\perp}_{decreq_j}(\circlearrowright^{\perp}_{z}).\circlearrowright^{\perp}_{exitpc}$.

The encoding of register r_j with content $n+1$ is

$$\langle\!\langle r_j = n+1 \rangle\!\rangle = S_j \llparenthesis \circlearrowleft_{exitreg} \llparenthesis \; \langle\!\langle r_j = n \rangle\!\rangle \; \rrparenthesis \, \rrparenthesis$$

where

$$S_j = ! \circlearrowright_{inc_j} \mid \circlearrowright^{\perp}_{decreq_j}(\circlearrowleft^{\perp}_{nz}).\circlearrowleft^{\perp}_{sync}.\circlearrowleft^{\perp}_{exitreg}.\circlearrowright_{garbage}.\circlearrowleft^{\perp}_{exitpc}$$

The encoding for the instruction $(i : I_i)$ is as follows:

$$\begin{aligned}
\langle\!\langle (i : Succ(r_j)) \rangle\!\rangle &= \circlearrowright^{\perp}_{p_i}(0).\circlearrowright^{\perp}_{inc_j}(\circlearrowleft_{exitreg}).\circlearrowleft^{\perp}_{exitpc}.S_j \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowright_{p_{i+1}} \llparenthesis \; \rrparenthesis \rrparenthesis \rrparenthesis \\
\langle\!\langle (i : DecJump(r_j, s)) \rangle\!\rangle &= \circlearrowright^{\perp}_{p_i}(0).\circlearrowright_{decreq_j}.J_{i+1,s}
\end{aligned}$$

where

$$\begin{aligned}
J_{h,k} = \circlearrowleft^{\perp}_{znz} \llparenthesis \circlearrowleft_{znz} \llparenthesis \circlearrowright_{nz} \llparenthesis \circlearrowleft_{sync} \llparenthesis \; \rrparenthesis \circ \circlearrowright_{exitpc} \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowright_{p_h} \llparenthesis \; \rrparenthesis \rrparenthesis \rrparenthesis \rrparenthesis \rrparenthesis \quad \circ \\
\circlearrowright_{z} \llparenthesis \circlearrowleft_{exitpc} \llparenthesis \circlearrowright_{p_k} \llparenthesis \; \rrparenthesis \rrparenthesis \rrparenthesis \; \rrparenthesis \; \rrparenthesis
\end{aligned}$$

It is easy to show that this encoding is deterministic, and that (a slight variation of) the results presented in the previous section hold also for this deterministic encoding.

An immediate consequence of Corollary 1 is the undecidability of existential termination:

Corollary 2. *The existential termination property is undecidable for PEP systems.*

4 Decidability of Termination for MBD

In this section we show that the existence of a divergent computation is decidable for MBD.

The decidability proof exploits the techniques similar to the ones developed in [2] for (fragments of) Mobile Ambients [5], and is based on the theory of well-structured transition systems [8]: the existence of an infinite computation starting from a given state is decidable for finitely branching transition systems, provided that the set of states can be equipped with a well-quasi-ordering, i.e., a quasi-ordering relation which is compatible with the transition relation and such that each infinite sequence of states admits an increasing subsequence.

We start by providing some basic definitions and results on well-structured transition systems and on well-quasi-ordering on sequences of elements belonging to a well-quasi-ordered set, that will be used in the following parts of this Section.

Then, we prove the decidability of termination for MBD; to this aim, we first provide an alternative semantics that is equivalent w.r.t. termination to the one presented in Section 2, but which is based on a finitely branching transition system and permits to define a well-quasi-ordering on the derivatives of a given system (i.e., the set of systems reachable from a given initial system). Then, by exploiting the theory developed in [8], we show that termination is decidable for MBD systems.

4.1 Well-Structured Transition System

We start by recalling some basic definitions and results from [8], concerning well-structured transition systems, that will be used in the following.

A *quasi-ordering* (qo) is a reflexive and transitive relation.

Definition 10. *A* well-quasi-ordering *(wqo) is a quasi-ordering $\leq$ over a set X such that, for any infinite sequence $x_0, x_1, x_2, \ldots$ in X, there exist indexes $i < j$ such that $x_i \leq x_j$.*

Note that, if $\leq$ is a wqo, then any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \ldots$ (with $i_0 < i_1 < i_2 < \ldots$).

Transition systems can be formally defined as follows.

Definition 11. *A* transition system *is a structure $TS = (S, \rightarrow)$, where S is a set of* states *and $\rightarrow \subseteq S \times S$ is a set of* transitions*.*
We write $Succ(s)$ to denote the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of $s \in S$.

TS is finitely branching *if $\forall s \in S : Succ(s)$ is finite. We restrict to finitely branching transition systems.*

Well-structured transition systems, defined as follows, provide the key tool to decide properties of computations.

Definition 12. *A* well-structured transition system (with strong compatibility) *is a transition system $TS = (S, \rightarrow)$, equipped with a quasi-ordering $\leq$ on S, also written $TS = (S, \rightarrow, \leq)$, such that the two following conditions hold:*

1. **well-quasi-ordering***: $\leq$ is a well-quasi-ordering, and*
2. **strong compatibility***: $\leq$ is (upward) compatible with $\rightarrow$, i.e., for all $s_1 \leq t_1$ and all transitions $s_1 \rightarrow s_2$, there exists a state t_2 such that $t_1 \rightarrow t_2$ and $s_2 \leq t_2$.*

The following theorem (a special case of a result in [8]) will be used to obtain our decidability result.

Theorem 2. *Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with decidable $\leq$ and computable $Succ$. The existence of an infinite computation starting from a state $s \in S$ is decidable.*

4.2 Higman's Lemma

To show that the quasi-ordering relation we will define on MBD systems is a well-quasi-ordering we need the following result, due to Higman [10] and stating that the set of the finite sequences over a set equipped with a wqo is well-quasi-ordered.

Given a set S, with S^* we denote the set of finite sequences of elements in S.

Definition 13. *Let S be a set and $\leq$ a wqo over S. The relation $\leq_*$ over S^* is defined as follows. Let $t, u \in S^*$, with $t = t_1 t_2 \ldots t_m$ and $u = u_1 u_2 \ldots u_n$. We have that $t \leq_* u$ iff there exists an injection f from $\{1, 2, \ldots, m\}$ to $\{1, 2, \ldots, n\}$ such that $t_i \leq u_{f(i)}$ and $i \leq f(i)$ for $i = 1, \ldots, m$.*

Note that relation $\leq_*$ is a quasi-ordering over S^*.

Lemma 3. *[Higman] Let S be a set and $\leq$ a wqo over S. Then, the relation $\leq_*$ is a wqo over S^*.*

Also the following propositions will be used to prove that the relation on systems is a well-quasi-ordering:

Proposition 2. *Let S be a finite set. Then the equality is a wqo over S.*

Proposition 3. *Let S, T be sets and $\leq_S$, $\leq_T$ be wqo over S and T, respectively. The relation $\leq$ over $S \times T$ is defined as follows: $(s_1, t_1) \leq (s_2, t_2)$ iff ($s_1 \leq_S s_2$ and $t_1 \leq_T t_2$). The relation $\leq$ is a wqo over $S \times T$.*

4.3 A Finitely Branching Semantics for MBD Systems

Because of the structural congruence rules, the reaction transition system for MBD is not finitely branching. To obtain a finitely branching transition system (with the same behavior w.r.t. termination), we take the transition system whose states are the equivalence classes of structural congruence.

Technically, it is possible to define a normal form for systems, up to the commutative and associative laws for the $\circ$ and $|$ operators.

In a system in normal form, the presence of a replicated version of a sequential process $!a.\sigma$ (resp. system $!(\sigma(\!|\, P \,|\!)))$ forbids the presence of any occurrence of the nonreplicated version of the same process (resp. system), as well as of other occurrences of the replicated version of the process (resp. system). Moreover, replication is distributed over the components of parallel composition operators, and redundant replications and empty systems and terms are removed.

Definition 14. *Let $\overset{ca}{=}$ be the least congruence on systems satisfying the commutative and associative rules for $\circ$ and $|$.*

A brane process σ is in normal form if $\sigma \overset{ca}{=} \prod_{i \in I} a_i.\sigma_i \mid \prod_{j \in J} !a'_j.\sigma'_j$, where

- *σ_i and σ'_j are in normal form for $i \in I$ and $j \in J$;*
- *if $a_i = bud_n^\perp(\rho)$ or $a_i = drip(\rho)$ then ρ is in normal form;*
  if $a'_j = bud_n^\perp(\rho)$ or $a'_j = drip(\rho)$ then ρ is in normal form;
- *if $\sigma_i \overset{ca}{=} \sigma'_j$ then $a_i \overset{ca}{\neq} a'_j$;*
- *if $\sigma'_i \overset{ca}{=} \sigma'_j$ and $a'_i \overset{ca}{=} a'_j$ then $i = j$.*

A system P is in normal form if $P \overset{ca}{=} \bigcirc_{i \in I} \sigma_i(\!|\, P_i \,|\!) \;\circ\; \bigcirc_{j \in J} !(\sigma'_j(\!|\, P'_j \,|\!))$, where

- *σ_i, P_i, σ'_j and P'_j are in normal form for $i \in I$ and $j \in J$;*
- *if $P_i \overset{ca}{=} P'_j$ then $\sigma_i \overset{ca}{\neq} \sigma'_j$;*
- *if $P'_i \overset{ca}{=} P'_j$ and $\sigma'_i \overset{ca}{=} \sigma'_j$ then $i = j$.*

The function nf produces the normal form of a process or a system:

Definition 15. *The normal form of a process is defined inductively as follows:*

$$\begin{array}{ll} nf(0) & = 0 \\ nf(mate_n.\sigma) & = mate_n.nf(\sigma) \\ nf(mate_n^{\perp},\sigma) & = mate_n^{\perp}.nf(\sigma) \\ nf(bud_n.\sigma) & = bud_n.nf(\sigma) \\ nf(bud_n^{\perp}(\rho).\sigma) & = bud_n^{\perp}(nf(\rho)).nf(\sigma) \\ nf(drip(\rho).\sigma) & = drip(nf(\rho)).nf(\sigma) \end{array}$$

Let $nf(\sigma) = \prod_{i\in I} a_i.\sigma_i \mid \prod_{j\in J} !a'_j.\sigma'_j$ *and* $nf(\tau) = \prod_{h\in H} b_h.\tau_h \mid \prod_{k\in K} !b'_k.\tau'_k$. *Then*

$$\begin{array}{rl} nf(\sigma \mid \tau) = & \prod\{a_i.\sigma_i \mid i \in I \wedge \forall k \in K : a_i.\sigma_i \overset{ca}{\not\equiv} b'_k.\tau'_k\} \mid \\ & \prod\{b_h.\tau_h \mid h \in H \wedge \forall j \in J : b_h.\tau_h \overset{ca}{\not\equiv} a'_j.\sigma'_j\} \mid \\ & \prod\{!a'_j.\sigma'_j \mid j \in J\} \mid \\ & \prod\{!b'_k.\tau'_k \mid k \in K \wedge \forall j \in J : b'_k.\tau'_k \overset{ca}{\not\equiv} a'_j.\sigma'_j\} \end{array}$$

and

$$nf(!\sigma) = \prod\{!a_i.\sigma_i \mid i \in I\} \mid \prod\{!a'_j.\sigma'_j \mid j \in J\}$$

Definition 16. *The normal form of a system is defined inductively as follows:*

$$\begin{array}{rl} nf(\diamond) = & \diamond \\ nf(\sigma.P) = & nf(\sigma).nf(P) \end{array}$$

Let $nf(P) = \bigcirc_{i\in I}\sigma_i(\!| P_i |\!) \circ \bigcirc_{j\in J}!(\sigma'_j(\!| P'_j |\!))$ *and* $nf(Q) = \bigcirc_{h\in H}\tau_h(\!| Q_h, |\!) \circ \bigcirc_{k\in K}!(\tau'_k(\!| Q'_k |\!))$. *Then*

$$\begin{array}{rl} nf(P \circ Q) = & \bigcirc\{\sigma_i(\!| P_i |\!) \mid i \in I \wedge \forall k \in K : \sigma_i(\!| P_i |\!) \overset{ca}{\not\equiv} \tau'_k(\!| Q'_k |\!)\} \circ \\ & \bigcirc\{\tau_h(\!| Q_h |\!) \mid h \in H \wedge \forall j \in J : \tau_h(\!| Q_h |\!) \overset{ca}{\not\equiv} \sigma'_j(\!| P'_j |\!)\} \circ \\ & \bigcirc\{!\sigma'_j(\!| P'_j |\!) \mid j \in J\} \circ \\ & \bigcirc\{!\tau'_k(\!| Q'_k |\!) \mid k \in K \wedge \forall j \in J : \tau'_k(\!| Q'_k |\!) \overset{ca}{\not\equiv} \sigma'_j(\!| P'_j |\!)\} \end{array}$$

$$nf(!P) = \bigcirc\{!\sigma_i(\!| P_i |\!) \mid i \in I\} \circ \bigcirc\{!\sigma'_j(\!| P'_j |\!) \mid j \in J\}$$

It is easy to see that function nf produces processes and systems in normal form.

Proposition 4. *Let σ be a process. Then $nf(\sigma)$ is a process in normal form. Let P be a system. Then $nf(P)$ is a system in normal form.*

Proof. By induction of the structure of σ (resp. P).

The normal form of two structurally congruent processes (resp. systems) is the same, up to associativity and commutativity of $\mid$ (resp. $\circ$) operator.

Table 1. The axioms for the reduction relation $\mapsto$ (mating)

`(mate1)` $mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis \circ mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis \circ R \mapsto$
$nf(\sigma|\sigma_0|\tau|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ R)$

`(mate2)` $(!mate_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \circ mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis \circ R \mapsto$
$nf(\sigma|(!mate_n.\sigma)|\sigma_0|\tau|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ R)$

`(mate3)` $!(mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis \circ R \mapsto$
$nf(\sigma|\sigma_0|\tau|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ !(mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ R)$

`(mate4)` $mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis \circ (!mate_n^\perp.\tau)|\tau_0\llparenthesis Q \rrparenthesis \circ R \mapsto$
$nf(\sigma|\sigma_0|\tau|(!mate_n^\perp.\tau)|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ R)$

`(mate5)` $(!mate_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \circ (!mate_n^\perp.\tau)|\tau_0\llparenthesis Q \rrparenthesis \circ R \mapsto$
$nf(\sigma|(!mate_n.\sigma)|\sigma_0|\tau|(!mate_n^\perp.\tau)|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ R)$

`(mate6)` $!(mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ !(mate_n^\perp.\tau)|\tau_0\llparenthesis Q \rrparenthesis \circ R \mapsto$
$nf(\sigma|\sigma_0|\tau|!(mate_n^\perp.\tau)|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ !(mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ R)$

`(mate7)` $mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis \circ !(mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis) \circ R \mapsto$
$nf(\sigma|\sigma_0|\tau|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ !(mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis) \circ R)$

`(mate8)` $(!mate_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \circ !(mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis) \circ R \mapsto$
$nf(\sigma|(!mate_n.\sigma)|\sigma_0|\tau|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ !(mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis) \circ R)$

`(mate9)` $!(mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ !(mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis) \circ R \mapsto$
$nf(\sigma|\sigma_0|\tau|\tau_0\llparenthesis P \circ Q \rrparenthesis \circ !(mate_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ !(mate_n^\perp.\tau|\tau_0\llparenthesis Q \rrparenthesis) \circ R)$

`(mate10)` $!(mate_n.\sigma|mate_n^\perp.\sigma'|\sigma_0\llparenthesis P \rrparenthesis) \circ R \mapsto$
$nf(\sigma|\sigma'|\sigma_0|\sigma_0\llparenthesis P \circ P \rrparenthesis \circ !(mate_n.\sigma|mate_n^\perp.\sigma'|\sigma_0\llparenthesis P \rrparenthesis) \circ R)$

`(mate11)` $!((!mate_n.\sigma)|mate_n^\perp.\sigma'|\sigma_0\llparenthesis P \rrparenthesis) \circ R \mapsto$
$nf(\sigma|(!mate_n.\sigma)|\sigma'|\sigma_0|\sigma_0\llparenthesis P \circ P \rrparenthesis \circ$
$!((!mate_n.\sigma)|mate_n^\perp.\sigma'|\sigma_0\llparenthesis P \rrparenthesis) \circ R)$

`(mate12)` $!(mate_n.\sigma|(!mate_n^\perp.\sigma')|\sigma_0\llparenthesis P \rrparenthesis) \circ R \mapsto$
$nf(\sigma|\sigma'|(!mate_n^\perp.\sigma')|\sigma_0|\sigma_0\llparenthesis P \circ P \rrparenthesis \circ$
$!(mate_n.\sigma|(!mate_n^\perp.\sigma')|\sigma_0\llparenthesis P \rrparenthesis) \circ R)$

`(mate13)` $!((!mate_n.\sigma)|(!mate_n^\perp.\sigma')|\sigma_0\llparenthesis P \rrparenthesis) \circ R \mapsto$
$nf(\sigma|(!mate_n.\sigma)|\sigma'|(!mate_n^\perp.\sigma')|\sigma_0|\sigma_0\llparenthesis P \circ P \rrparenthesis \circ$
$!((!mate_n.\sigma)|(!mate_n^\perp.\sigma')|\sigma_0\llparenthesis P \rrparenthesis) \circ R)$

Proposition 5. *Let σ, τ be two processes. If $\sigma \equiv \tau$ then $nf(\sigma) \overset{ca}{=} nf(\tau)$. Let P, Q be two systems. If $P \equiv Q$ then $nf(P) \overset{ca}{=} nf(Q)$.*

Proof. By induction on the structure of the proof of $\sigma \equiv \tau$ (resp. $P \equiv Q$).

We define an alternative semantics for systems in normal form. This semantics turns out to be finitely branching, and equivalent to the reduction semantics of section 2.

Definition 17. *The reaction relation $\mapsto$ for systems in normal form is the least relation satisfying the axioms and rules in Tables 1, 2 and 3.*

The reduction relation $\mapsto$ is finitely branching over the set of systems in normal form:

Proposition 6. *Let P be a system in normal form. The set of immediate successors $Succ(P) = \{P' \mid P \mapsto P'\}$ is finite.*

Table 2. The axioms for the reduction relation $\mapsto$ (budding)

(bud1) $bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis \circ Q \rrparenthesis \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|\tau_0\llparenthesis Q \rrparenthesis \circ R)$

(bud2) $(!bud_n^{\perp}(\rho).\tau)|\tau_0\llparenthesis\, bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis \circ Q \rrparenthesis \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|(!bud_n^{\perp}(\rho).\tau)|\tau_0\llparenthesis Q \rrparenthesis \circ R)$

(bud3) $!(bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis \circ Q \rrparenthesis) \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|\tau_0\llparenthesis Q \rrparenthesis \circ$
$!(bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis \circ Q \rrparenthesis) \circ R)$

(bud4) $bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, !(bud_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \circ Q \rrparenthesis \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|!(bud_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|\tau_0\llparenthesis Q \rrparenthesis \circ R)$

(bud5) $(!bud_n^{\perp}(\rho).\tau)|\tau_0\llparenthesis\, !(bud_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \circ Q \rrparenthesis \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|!(bud_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|(!bud_n^{\perp}(\rho).\tau)|\tau_0\llparenthesis Q \rrparenthesis \circ R)$

(bud6) $!(bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, !(bud_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis) \circ Q \rrparenthesis) \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|!(bud_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|\tau_0\llparenthesis Q \rrparenthesis \circ$
$!(bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, !(bud_n.\sigma)|\sigma_0\llparenthesis P \rrparenthesis \circ Q \rrparenthesis) \circ R)$

(bud7) $bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, !(bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ Q \rrparenthesis \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|\tau_0\llparenthesis\, !(bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ Q \rrparenthesis \circ R)$

(bud8) $(!bud_n^{\perp}(\rho).\tau)|\tau_0\llparenthesis\, !(bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ Q \rrparenthesis \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|(!bud_n^{\perp}(\rho).\tau)|\tau_0\llparenthesis\, !(bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ Q \rrparenthesis \circ R)$

(bud9) $!(bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, !(bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ Q \rrparenthesis) \circ R \mapsto$
$nf(\rho\llparenthesis \sigma|\sigma_0\llparenthesis P \rrparenthesis \rrparenthesis \circ \tau|\tau_0\llparenthesis\, !(bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ Q \rrparenthesis \circ$
$!(bud_n^{\perp}(\rho).\tau|\tau_0\llparenthesis\, !(bud_n.\sigma|\sigma_0\llparenthesis P \rrparenthesis) \circ Q \rrparenthesis) \circ R)$

Table 3. The axioms and rules for the reduction relation $\mapsto$

$(\texttt{drip1})$	$drip(\rho).\sigma\vert\sigma_0(\!\vert P \vert\!) \circ R \mapsto nf(\rho(\!\vert\,\vert\!) \circ \sigma\vert\sigma_0(\!\vert P \vert\!) \circ R)$
$(\texttt{drip2})$	$(!drip(\rho).\sigma)\vert\sigma_0(\!\vert P \vert\!) \circ R \mapsto nf(\rho(\!\vert\,\vert\!) \circ \sigma\vert(!drip(\rho).\sigma)\vert\sigma_0(\!\vert P \vert\!) \circ R)$
$(\texttt{drip3})$	$!(drip(\rho).\sigma\vert\sigma_0(\!\vert P \vert\!)) \circ R \mapsto nf(\rho(\!\vert\,\vert\!) \circ \sigma\vert\sigma_0(\!\vert P \vert\!) \circ !(drip(\rho).\sigma\vert\sigma_0(\!\vert P \vert\!)) \circ R)$
$(\texttt{brane1})$	$\dfrac{P \rightarrow Q}{\sigma(\!\vert P \vert\!) \circ R \mapsto \sigma(\!\vert Q \vert\!) \circ R}$
$(\texttt{brane2})$	$\dfrac{P \rightarrow Q}{!(\sigma(\!\vert P \vert\!)) \circ R \mapsto \sigma(\!\vert Q \vert\!) \circ !(\sigma(\!\vert P \vert\!)) \circ R}$

Proof. By induction on the structure of P. Let $P{=}\bigcirc_{i\in I}\sigma_i(\!| P_i |\!) \circ \bigcirc_{j\in J}!(\sigma'_j(\!| P'_j |\!))$. By inductive hypothesis, the sets $Succ(P_i)$ and $Succ(P'_j)$ are finite for all $i \in I$ and $j \in J$.

Let $\#act(P)$ be the number of occurrences of actions in system P.

Consider a reduction $P \mapsto P'$. The last rule applied in the proof of this reduction can either be one of the axioms of Tables 1, 2 and 3 or one of the rules in Table 3.

Consider axiom (`mate1`): the number of reductions obtained by application of this axiom is bounded by the product of the number of actions *mate* occurring in P with the number of actions $mate^{\perp}$ occurring in P, which is smaller than $(\#act(P))^2$.

The same reasoning can be applied to each axiom, hence the number of immediate successors of P obtained by application of an axiom is bounded by $(10 + 9 + 3) \times (\#act(P))^2$.

Consider now the set of immediate successors obtained by application of rule (`brane1`). This set is bounded by $|I| \times max\{|Succ(P_i)| \mid i \in I\}$. By inductive hypothesis the sets of immediate successors of systems P_i, for $i \in I$, are finite. Hence also the set of immediate successors of P obtained by application of rule (`brane1`) is finite.

The same reasoning applies to rule (`brane2`), hence the set of immediate successors of P is finite.

We now show that a system P terminates (according to the reaction rule $\rightarrow$) iff $nf(P)$ terminates (according to the reaction rule $\mapsto$). To this aim, we need the following auxiliary results.

Lemma 4. *Let P be a system. If $P \rightarrow P'$ then $nf(P) \mapsto nf(P')$.*

Proof. By induction on the proof of $P \rightarrow P'$ (the difficult case is when the last rule in the proof tree is (`par`)).

Lemma 5. *Let P be a system. If $nf(P) \mapsto Q$ then $nf(P) \rightarrow Q$.*

Proof. By induction on the proof of $nf(P) \mapsto Q$.

Corollary 3. *Let P be a system. The system P terminates iff $nf(P)$ terminates.*

4.4 Decidability of Termination for MBD Systems

Let us consider a system P in normal form. In this section we provide a quasi-order on the derivatives of P (and a quasi-order on brane processes) that turns out to be a wqo compatible with $\mapsto$. Hence, exploiting the results in section 4.2, we obtain decidability of termination.

We note that each system (resp. process) in normal form is essentially a finite sequence of objects of kind $\sigma(\!| Q |\!)$ or $!(\sigma(\!| Q |\!))$ (resp. of objects of kind $a.\sigma$ or $!a.\sigma$). If we consider the nesting level of membranes, we note that each subsystem Q contained in a subterm $\sigma(\!| Q |\!)$ or $!(\sigma(\!| Q |\!))$ of a system R is simpler than R. More precisely, the maximum nesting level of membranes in Q is strictly smaller than the maximum nesting level of membranes in R. As already observed in [4], the reactions in MBD preserve the nesting level of membranes; hence, the nesting level of membranes in a system P provides an upper bound to the nesting level of membranes in the set of the (normal forms of the) derivatives of P.

Definition 18. *The nesting level of a system is defined inductively as follows:*

$$\begin{array}{ll} nl(\diamond) = & 0 \\ nl(\sigma(\!|P|\!)) = & nl(P) + 1 \\ nl(P \circ Q) = & max\{nl(P), nl(Q)\} \\ nl(!P) = & nl(P) \end{array}$$

Proposition 7. *Let P be a system in normal form. If $P \mapsto P'$ then $nl(P') \leq nl(P)$.*

Proof. By induction of the proof of $P \mapsto P'$.

Thanks to normal forms, we have that the set of processes of kind $a.\sigma$ or $!a.\sigma$ that occur as subterms in the derivatives (w.r.t. $\mapsto$) of a process in normal form is finite. This fact will be used to show that the quasi-orders on processes and on systems are wqo.

Definition 19. *Let P be a system in normal form. The set of derivatives of P w.r.t. $\mapsto$ is defined as follows: $nfDeriv(P) = \{P' \mid P \mapsto^* P'\}$.*

Definition 20. *The set of sequential and replicated sequential subprocesses of a process is defined inductively as follows:*

$$\begin{array}{ll}
Subp(0) = & \emptyset \\
Subp(mate_n.\sigma) = & \{mate_n.\sigma\} \cup Subp(\sigma) \\
Subp(mate_n^{\perp}.\sigma) = & \{mate_n^{\perp}.\sigma\} \cup Subp(\sigma) \\
Subp(bud_n.\sigma) = & \{bud_n.\sigma\} \cup Subp(\sigma) \\
Subp(bud_n^{\perp}(\rho).\sigma) = & \{bud_n^{\perp}(\rho).\sigma\} \cup Subp(\rho) \cup Subp(\sigma) \\
Subp(drip(\rho).\sigma) = & \{drip(\rho).\sigma\} \cup Subp(\rho) \cup Subp(\sigma) \\
Subp(\sigma \mid \tau) = & Subp(\sigma) \cup Subp(\tau) \\
Subp(!\sigma) = & \{!\sigma' \mid \sigma' \in Subp(\sigma)\} \cup Subp(\sigma)
\end{array}$$

The set of sequential and replicated sequential subprocesses of a system is defined inductively as follows:

$$\begin{array}{ll}
Subp(\diamond) = & \emptyset \\
Subp(\sigma(\!| P |\!)) = & Subp(\sigma) \cup Subp(P) \\
Subp(P \circ Q) = & Subp(P) \cup Subp(Q) \\
Subp(!P) = & Subp(P)
\end{array}$$

It is easy to see that the set of sequential and replicated sequential subprocesses of a process (resp. system) is finite:

Proposition 8. *Let σ be a process. The set $Subp(\sigma)$ is finite.*
Let P be a system. The set $Subp(P)$ is finite.

Proof. By induction on the structure of σ (resp. P).

The following proposition – stating that the transformation of a process (resp. system) in normal form does not increase its set of sequential and replicated sequential subprocesses – will be used to show that the set of sequential and replicated sequential subprocesses does not increase after execution of a reduction step.

Proposition 9. *Let σ be a process. Then $Subp(nf(\sigma)) \subseteq Subp(\sigma)$.*
Let P be a system. Then $Subp(nf(P)) \subseteq Subp(P)$.

Proof. By induction on the structure of σ (resp. P).

Proposition 10. *Let P be a process in normal form. If $P \mapsto P'$ then $Subp(P') \subseteq Subp(P)$.*

Proof. By induction on the proof of $P \mapsto P'$.

Definition 21. *Let P be a process in normal form. The set of subprocesses of the derivatives of P is defined as $SubDeriv(P) = \bigcup_{P' \in nfDeriv(P)} Subp(P')$.*

Proposition 11. *Let P be a process in normal form. Then the set $SubDeriv(P)$ is finite.*

Proof. Let $P' \in nfDeriv(P)$. Then $P \mapsto^* P'$. By induction on the length of the derivation $P \mapsto^* P'$ and by Proposition 10 we obtain $Subp(P') \subseteq Subp(P)$. By Proposition 8 we obtain that $SubDeriv(P)$ is finite.

We introduce a quasi-order $\preceq_{proc}$ on processes in normal form such that $\sigma \preceq_{proc} \tau$ if

– for each occurrence of a replicated guarded process at top-level in σ there is a corresponding occurrence of the same process at top-level in τ;
– for each occurrence of a guarded process at top-level in σ there is either a corresponding occurrence of the same process or an occurrence of the replicated version of the process at top-level in τ.

Definition 22. *Let σ and τ be two processes in normal form. Let $\sigma = \prod_{i \in I} a_i.\sigma_i \mid \prod_{j \in J} !a'_j.\sigma'_j$ and $\tau = \prod_{h \in H} b_h.\tau_h \mid \prod_{k \in K} !b'_k.\tau'_k$, and $H \cap K = \emptyset$. We say that $\sigma \preceq_{proc} \tau$ if there exists a pair of functions (f, g) such that:*

– *$f : I \to H \cup K$ and $g : J \to K$*
– *$\forall i, i' \in I$: if $f(i) = f(i')$ and $f(i) \in H$ then $i = i'$*
– *$\forall i \in I$: if $f(i) \in H$ then $b_{f(i)}.\tau_{f(i)} \overset{ca}{=} a_i.\sigma_i$*
– *$\forall i \in I$: if $f(i) \in K$ then $b'_{f(i)}.\tau'_{f(i)} \overset{ca}{=} a_i.\sigma_i$*
– *$\forall j \in J$: $b'_{g(j)}.\tau'_{g(j)} \overset{ca}{=} a'_j.\sigma'_j$*

We define a quasi-order on systems such that $R \preceq_{sys} S$ if

– for each replicated membrane $!(\rho(\!| R_1 |\!))$ at top-level in R there is a corresponding replicated membrane $!(\sigma(\!| S_1 |\!))$ at top-level in S such that ρ is smaller than σ and R_1 is smaller than S_1;
– for each occurrence of a membrane $\rho(\!| R_1 |\!)$ at top-level in R there is
  - either a corresponding occurrence of a membrane $\sigma(\!| S_1 |\!)$ at top-level in S such that ρ is smaller than σ and R_1 is smaller than S_1
  - or an occurrence of a replicated membrane $!(\rho(\!| R_1 |\!))$ at top-level in S.

Definition 23. *Let P, Q be systems. Let $P = \bigcirc_{i \in I} \sigma_i(\!| P_i |\!) \circ \bigcirc_{j \in J} !(\sigma'_j(\!| P'_j |\!))$ and $Q = \bigcirc_{h \in H} \tau_h(\!| Q_h, |\!) \circ \bigcirc_{k \in K} !(\tau'_k(\!| Q'_k |\!))$ and $H \cap K = \emptyset$. We say that $P \preceq_{sys} Q$ if there exists a pair of functions (f, g) such that:*

– *$f : I \to H \cup K$ and $g : J \to K$*
– *$\forall i, i' \in I$: if $f(i) = f(i')$ and $f(i) \in H$ then $i = i'$*
– *$\forall i \in I$: if $f(i) \in H$ then $\sigma_i \preceq_{proc} \tau_{f(i)}$ and $P_i \preceq_{sys} Q_{f(i)}$*
– *$\forall i \in I$: if $f(i) \in K$ then $\sigma_i \preceq_{proc} \tau'_{f(i)}$ and $P_i \preceq_{sys} Q'_{f(i)}$*
– *$\forall j \in J$: $\sigma'_j \preceq_{proc} \tau'_{g(j)}$ and $P'_j \preceq_{sys} Q'_{g(j)}$*

It is easy to see that $\preceq_{proc}$ and $\preceq_{sys}$ are quasi-orderings.

We start showing that the relation $\preceq_{sys}$ is strongly compatible with $\mapsto$. To this aim, we need the following auxiliary propositions:

Proposition 12. *Let $\sigma_1, \sigma_2, \tau_1, \tau_2$ be processes in normal form. If $\sigma_i \preceq_{proc} \tau_i$ for $i = 1, 2$ then $nf(\sigma_1 \mid \sigma_2) \preceq_{proc} nf(\tau_1 \mid \tau_2)$.*

Proposition 13. *Let P_1, P_2, Q_1, Q_2 be systems in normal form. If $P_i \preceq_{sys} Q_i$ for $i = 1, 2$ then $P_1 \circ P_2 \preceq_{sys} Q_1 \circ Q_2$.*

Theorem 3. *Let P, P', Q be systems in normal form. If $P \mapsto P'$ and $P \preceq_{sys} Q$ then there exists Q' in normal form such that $Q \mapsto Q'$ and $Q \preceq_{sys} Q'$.*

Proof. The proof is by induction on the structure of P, then by case analysis on the last axiom or rule applied to obtain the reduction $P \mapsto P'$.

The following proposition will be used to show that $\preceq_{proc}$ is a wqo over the set of subprocesses of the derivatives of a system P.

Proposition 14. *Let σ and τ be two processes in normal form.*
Let $\sigma = \prod_{i=1}^{n_1} a_i.\sigma_i \mid \prod_{j=n_1+1}^{n_1+n_2} !a'_j.\sigma'_j$ and $\tau = \prod_{h=1}^{m_1} b_h.\tau_h \mid \prod_{k=m_1+1}^{m_1+m_2} !b'_k.\tau'_k$.
If $a_1.\sigma_1 \dots a_{n_1}.\sigma_{n_1} !a'_{n_1+1}.\sigma'_{n_1+1} \dots !a'_{n_1+n_2}.\sigma'_{n_1+n_2} =_*$
$b_1.\tau_1 \dots b_{m_1}.\tau_{m_1} !b'_{m_1+1}.\tau'_{m_1+1} \dots !b'_{m1+m_2}.\tau'_{m1+m_2}$ then $\sigma \preceq_{proc} \tau$.

Proof. If $a_1.\sigma_1 \dots a_{n_1}.\sigma_{n_1} !a'_{n_1+1}.\sigma'_{n_1+1} \dots !a'_{n_1+n_2}.\sigma'_{n_1+n_2} =_*$
$b_1.\tau_1 \dots b_{m_1}.\tau_{m_1} !b'_{m_1+1}.\tau'_{m_1+1} \dots !b'_{m1+m_2}.\tau'_{m1+m_2}$ then there exists an injection $f : \{1, \dots, n_1 + n_2\} \to \{1, \dots, m_1 + m_2\}$ mapping the each element of the first sequence into a corresponding, equal element in the second sequence. The first n_1 (resp m_1) elements of the first (resp. second) sequence are sequential processes, whereas the last n_2 (resp. m_2) elements of the first (resp. second) sequence are replicated sequential processes. Hence the first n_1 (resp. last n_2) elements of the first sequence will be mapped on the first m_1 (resp. last m_2) elements of the second sequence.

Consider the pair of functions (g_1, g_2) such that

- $g_1(i) = f(i)$ for $i = 1, \dots, n_1$
- $g_2(i) = f(i)$ for $i = n_1 + 1, \dots, n_1 + n_2$

The pair of functions (g_1, g_2) satisfies the conditions of Definition 22, hence $\sigma \preceq_{proc} \tau$.

By Higman lemma and Proposition 2 it easy to prove that

Lemma 6. *Let P be a system in normal form. The relation $\preceq_{proc}$ is a wqo over the set of processes in $SubDeriv(P)$.*

Proof. Take an infinite sequence $\sigma_1, \sigma_2, \dots, \sigma_h, \dots$ of normal form processes in $SubDeriv(P)$. Let $\sigma_h {=} \prod_{i=1}^{n_h} a_{i,h}.\sigma_{i,h} \mid \prod_{j=1}^{m_h} !a'_{j,h}.\sigma'_{j,h}$. By definition of $SubDeriv$ and $Subp$, we have that $\forall h > 0 : \forall 1 \le i \le n_h : a_{i,h}.\sigma_{i,h} \in SubDeriv(P)$ and $\forall h > 0 : \forall 1 \le i \le m_h : !a'_{j,h}.\sigma_{j,h} \in SubDeriv(P)$. Hence, we have an infinite sequence of elements of $SubDeriv(P)^*$; as $SubDeriv(P)$ is finite (by Proposition 11), by Proposition 2 and Higman Lemma (Lemma 3) we have that $=_*$ is a wqo over $SubDeriv(P)$. Thus there exist i, j such that
$a_{1,i}.\sigma_{1,i} \dots a_{n_i,i}.\sigma_{n_i,i} !a'_{1,i}.\sigma'_{1,i} \dots !a'_{m_i,i}.\sigma'_{m_i,i} =_*$
$a_{1,j}.\sigma_{1,j} \dots a_{n_j,j}.\sigma_{n_j,j} !a'_{1,j}.\sigma'_{1,j} \dots !a'_{m_j,j}.\sigma'_{m_j,j}$.

By Proposition 14 we have $\sigma_i \preceq_{proc} \sigma_j$.

Now it is possible to prove that $\preceq_{sys}$ is a wqo.

The set of derivatives of a system P w.r.t. $\mapsto$ with nesting level not greater than n is defined as

Definition 24. *Let P be a system in normal form and $n \geq 0$. We define $nfDeriv_n(P) = \{Q \mid Q \in nfDeriv(P) \wedge nl(Q) \leq n\}$.*

Proposition 15. *Let P be a system in normal form. Then $nfDeriv(P) = nfDeriv_{nl(P)}(P)$.*

Proof. A consequence of Proposition 7.

Now we show that $\preceq_{sys}$ is a wqo over a subset of derivatives whose elements have a nesting level smaller than a given natural number. The proof proceeds by induction on the nesting level of membranes, and makes use of Higman's Lemma, of Lemma 6 and of Proposition 3.

Theorem 4. *Let P be a system in normal form and $n \geq 0$. The relation $\preceq_{sys}$ is a wqo over the set $nfDeriv_n(P)$.*

Proof. The proof is by induction on n.

The case $n = 0$ is trivial, as $nfDeriv_0(P) \subseteq \{\diamond\}$.

For the inductive step, take $n > 0$ and an infinite sequence $P_1, P_2, \ldots, P_h, \ldots$ with $P_h \in nfDeriv_n(P)$.

Let $P_h = \bigcirc_{i=1}^{n_h} \sigma_{i,h}(\!| P_{i,h} |\!) \circ \bigcirc_{j=n_h+1}^{m_h} !(\sigma'_{j,h}(\!| P'_{j,h} |\!))$.

As $P_h \in nfDeriv_n(P)$, we have that $\sigma_{i,h} \in Subp(P)$ for $i = 1, \ldots, n_h$ and $\sigma'_{j,h} \in Subp(P)$ for $j = n_h + 1, \ldots, m_h$.

Moreover, $nl(P_h) \leq n$; by definition of nesting level, we obtain $nl(P_{i,h}) \leq n-1$ for $i = 1, \ldots, n_h$ and $nl(P'_{j,h}) \leq n - 1$ for $j = n_h + 1, \ldots, m_h$.

Hence, $P_{i,h} \in nfDeriv_{n-1}(P)$ for $i = 1, \ldots, n_h$ and $P'_{j,h} \in nfDeriv_{n-1}(P)$ for $j = n_h + 1, \ldots, m_h$.

By Lemma 6 we know that $\preceq_{proc}$ is a wqo over $Subp(P)$. By inductive hypothesis, we have that $\preceq_{sys}$ is a wqo over $nfDeriv_{n-1}(P)$.

By Proposition 3 we obtain that $\preceq = (\preceq_{proc}, \preceq_{sys})$ is a wqo over $Subp(P) \times nfDeriv_{n-1}(P)$.

By Higman Lemma we obtain that $\preceq_*$ is a wqo over
$(Subp(P) \times nfDeriv_{n-1}(P))^*$.

By Proposition 3 we obtain that $\preceq' = (\preceq_*, \preceq_*)$ is a wqo over
$(Subp(P) \times nfDeriv_{n-1}(P))^* \ \times \ (Subp(P) \times nfDeriv_{n-1}(P))^*$.

Consider the infinite sequence $s_1, \ldots, s_h, \ldots$ with

$$\begin{aligned} s_h = (((&\sigma_{1,h}, P_{1,h}), \ldots, (\sigma_{n_h,h}, P_{n_h,h})),\\ ((&\sigma'_{n_h+1,h}, P'_{n_h+1,h}), \ldots, (\sigma'_{n_h+m_h,h}, P'_{n_h+m_h,h}))) \end{aligned}$$

We have that $s_h \in (Subp(P) \times nfDeriv_{n-1}(P))^* \times (Subp(P) \times nfDeriv_{n-1}(P))^*$ for $h > 0$. As $\preceq'$ is a wqo over such a set, there exist k, q such that $s_k \preceq' s_q$.

This means that

$$(\sigma_{1,k}, P_{1,k}), \ldots, (\sigma_{n_k,k}, P_{n_k,k}) \preceq_* (\sigma_{1,q}, P_{1,q}), \ldots, (\sigma_{n_q,q}, P_{n_q,q})$$

Thus there exists an injection $f : \{1, \ldots, n_k\} \to \{1, \ldots, n_q\}$ such that $\sigma_{i,k} \preceq_{proc} \sigma_{i,f(k)}$ and $P_{i,k} \preceq_{sys} P_{i,f(k)}$ for $i = 1, \ldots n_k$.

We also have that

$$\begin{array}{l}(\sigma'_{n_k+1,k}, P'_{n_k+1,k}), \ldots, (\sigma'_{n_k+m_k,k}, P'_{n_k+m_k,k}) \preceq_* \\ (\sigma'_{1,q}, P'_{1,q}), \ldots, (\sigma'_{n_q+m_q,q}, P'_{n_q+m_q,q})\end{array}$$

Thus there exists an injection $g : \{n_h + 1, \ldots, n_h + m_k\} \to \{n_q + 1, \ldots, n_q + m_q\}$ such that $\sigma'_{i,k} \preceq_{proc} \sigma'_{i,g(k)}$ and $P'_{i,k} \preceq_{sys} P'_{i,g(k)}$ for $i = n_k + 1, \ldots n_k + m_k$.

Consider the systems P_k and P_q and the pair of functions (f, g). We have that $P_k = \bigcirc_{i=1}^{n_k} \sigma_{i,k} (\!| P_{i,k} |\!) \circ \bigcirc_{j=1}^{m_k} !(\sigma'_{j,k} (\!| P'_{j,k} |\!))$ and $P_q = \bigcirc_{i=1}^{n_q} \sigma_{i,q} (\!| P_{i,q} |\!) \circ \bigcirc_{j=1}^{m_q} !(\sigma'_{j,q} (\!| P'_{j,q} |\!))$. Moreover, the pair of functions (f, g) satisfies the requirements of Definition 23. Hence, $P_k \preceq_{sys} P_q$.

Summing up, we have showed that $\preceq_{sys}$ is a wqo over $nfDeriv_n(P)$.

Theorem 5. *Let P be a system in normal form. The relation $\preceq_{sys}$ is a wqo over the set $nfDeriv(P)$.*

Proof. An easy consequence of Theorem 4 and of Proposition 15.

The following theorem ensures that the hypothesis of Theorem 2 are satisfied.

Theorem 6. *Let P be a system in normal form. Then the transition system $(nfDeriv(P), \mapsto, \preceq_{sys})$ is a well-structured transition system with decidable $\preceq_{sys}$ and computable Succ.*

Proof. Strong compatibility of $\preceq_{sys}$ with the transition relation $\mapsto$ has been proved in Theorem 3. By Theorem 5 we have that $\preceq_{sys}$ is a wqo over $nfDeriv(P)$. From Definition 23 it is possible to deduce an effective procedure to check $\preceq_{sys}$. From Definitions 16 and 17 it is possible to deduce an effective procedure to compute $Succ$.

By the above theorem and Theorem 2 we get the following

Corollary 4. *Let P be a MBD system. The termination of P is decidable.*

5 Conclusion

The aim of this paper is to investigate and compare the expressiveness of the two basic brane calculi PEP and MBD w.r.t. their ability to encode computable functions.

Regarding the PEP calculus we provided the following results:

(1) an encoding of RAMs that preserves the universal termination property, i.e., the existence of a divergent computation;

(2) an improved, deterministic encoding of RAMs that faithfully models the RAM behavior, in the following sense:
  - If the RAM terminates then the (unique) computation of the RAM encoding terminates;

  – if the RAM does not terminate then the (unique) computation of the RAM encoding diverges.

The impact of the above results on the decidability of termination properties in PEP is the following:

– as a consequence of the existence of the encoding (1), we obtain the undecidability of universal termination for PEP systems. Note that the existence of the encoding (1) does not imply the undecidability of existential termination on PEP systems, because PEP systems are in general nondeterministic, and may have both a terminating and a divergent computation.
– as a consequence of the existence of the deterministic encoding (2), and of the equivalence of existential and universal termination properties for deterministic systems, we also obtain the undecidability of existential termination for PEP systems.

Regarding the MBD calculus, we provided the following result:

(3) Universal termination is decidable on MBD systems.

As a consequence of the decidability of universal termination, we obtain the impossibility to provide an encoding of RAMs in MBD that preserves the universal termination property, i.e., an encoding that has a divergent computation if and only if the RAM diverges. Hence, it is also impossible to define a deterministic encoding of RAMs in MBD.

From the above results we deduce the following expressiveness gap between PEP and MBD, w.r.t. their ability to reproduce the behavior of a RAM: there exists a deterministic (hence, both universal and existential termination preserving) encoding of RAMs in PEP, but there exist neither a deterministic nor a universal termination preserving encoding of RAMs in MBD. Regarding the decidability of properties, we have that universal termination is decidable in MBD, whereas it turns out to be undecidable in PEP. Hence, there exist no encoding of PEP in MBD that preserves the existence of a divergent computation.

Regarding the MBD calculus, the decidability of universal termination (3) does not tell anything about the decidability of existential termination, and about the possibility to define a weaker, nondeterministic encoding of RAMs in MBD, that preserves existential termination (i.e., the encoding has a terminating computation if and only if the RAM terminates). This topic has been investigated in [1], where a nondeterministic encoding of RAMs in MBD, which preserves existential termination, is provided. As a consequence of this result, we obtain the undecidability of the existential termination property for MBD.

The comparison of a (nondeterministic) model with its deterministic fragment is an interesting topic in automata theory, that has recently attracted the interest of the research community working on membrane computing (see, e.g., [11] and the references therein). From the results presented in this paper and in [1], we deduce the following:

– The deterministic fragment of PEP is as powerful as the full (nondeterministic) PEP calculus w.r.t. the ability to encode RAMs;

– There exists a gap between the deterministic fragment and the full MBD calculus, as there exists a weak, existential termination preserving encoding of RAMs in MBD [1], but there exist no such encoding in the deterministic fragment of MBD (this is a consequence of the decidability of universal termination on MBD, and of the equivalence of universal and existential termination on deterministic systems).

The results presented in this paper hold for the interleaving semantics, according to which a single interaction is executed at each computational step. While interleaving semantics is the classical semantics for process calculi, the usual semantics in membrane computing is based on maximal parallelism: at each computational step, a maximal set of independent interactions must be executed simultaneously. As for many variants of P systems the computational power decreases when moving from the maximal parallelism to the interleaving semantics (see [9]), it seems worthwhile to investigate if the decidability of universal termination for MBD with the interleaving semantics also holds when moving to the maximal parallelism semantics. Concerning MBD with the maximal parallelism semantics, in [1] we provided a deterministic encoding of RAMs. Hence, both the universal and existential termination properties turn out to be undecidable in MBD with the maximal parallelism semantics. From the decidability of universal termination in MBD with interleaving semantics (3), we obtain an expressiveness gap between MBD with the interleaving and with the maximal parallelism semantics, thus confirming the intuition emerging from [9] for P systems: in most cases the computational power increases when moving from interleaving to maximal parallelism. Note that the undecidability of universal termination for MBD with maximal parallelism semantics does not contradict the decidability of universal termination for MBD with interleaving semantics provided in the present paper, as the well-quasi-ordering on systems $\preceq_{sys}$ defined in Section 4 is not compatible with the maximal parallelism reduction semantics. Consider, e.g., the systems $P_1 = mate_n(\!|\ |\!) \circ mate_n^{\perp}(\!|\ |\!) \circ mate_m(\!|\ |\!)$ and $P_2 = P_1 \circ mate_m^{\perp}(\!|\ |\!)$. We have that $P_1 \preceq_{sys} P_2$; the only reduction step that P_1 can perform is the fusion of the two membranes $mate_n(\!|\ |\!)$ and $mate_n^{\perp}(\!|\ |\!)$; thus, according to the maximal parallelism semantics, $P_1 \Rightarrow mate_m(\!|\ |\!)$. On the other hand, two independent fusions can be performed by P_2; thus, according to the maximal parallelism semantics, both fusions must be performed, and the only computational step for P_2 is $P_2 \Rightarrow \diamond$. Thus, $P_1 \preceq_{sys} P_2$ and $P_1 \Rightarrow mate_m(\!|\ |\!)$, but there exist no system P_2' such that $P_2 \Rightarrow P_2'$ and $mate_m(\!|\ |\!) \preceq_{sys} P_2'$, thus preventing the compatibility of $\preceq_{sys}$ with $\Rightarrow$ to hold.

In [6] a variant of P systems, inspired by the interaction primitives of the Brane Calculi, has been proposed and investigated. Quite surprisingly, it is shown that the class of P systems containing only the *mate* and *drip* operations is Turing powerful. A deep comparison of Brane Calculi and the class of P systems proposed in [6] deserves further investigation. At a first sight, it seems that the interaction primitives used in P systems are more powerful than the MBD primitives. Some other, evident differences are the following: Brane Calculi are equipped with an interleaving semantics, whereas P systems have a maximal parallelism semantics; in [6] only a finite number of membranes is sufficient to achieve Turing completeness, whereas in the present paper (and in [1]) an unbounded number of membranes is required.

In the present paper we showed that universal termination is a decidable property for MBD. The technique employed to prove the decidability of universal termination is based on the theory of well-structured transition systems: besides universal termination, such a theory permits to analyse other interesting properties, such as, e.g., coverability, boundedness, and eventuality properties [8]. We plan to investigate the possibility to use this theory for the analysis of biologically relevant properties.

We also plan to extend our investigation to recent refinements of Brane Calculi, such as, e.g., the Projective Brane Calculus [7], as well as to quantitative variants of Brane Calculi, along the lines of, e.g., [15].

Acknowledgement. We thank the anonymous referees for their comments. We are indebted to Fabien Tarissan for his careful comments and precious suggestions.

References

1. N. Busi. On the computational power of the Mate/Bud/Drip Brane Calculus: interleaving vs. maximal parallelism. Proc 6th International Workshop on Membrane Computing (WMC6), LNCS 3850, Springer, 2006.
2. N. Busi and G. Zavattaro. On the expressive power of movement and restriction in pure mobile ambients. *Theoretical Computer Science*, 322:477–515, 2004.
3. L. Cardelli. Brane Calculi - Interactions of biological membranes. Proc. Computational Methods in System Biology 2004 (CMSB 2004), LNCS 3082, Springer, 2005.
4. L. Cardelli. Abstract Machines for System Biology. Draft, 2005.
5. L. Cardelli and A.D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
6. L. Cardelli and G. Păun. An universality result for a (Mem)Brane Calculus based on mate/drip operations. International Journal of Foundations of Computer Science, to appear.
7. V. Danos and S. Pradalier. Projective Brane Calculus. Proc. Computational Methods in System Biology 2004 (CMSB 2004), LNCS 3082, Springer, 2005.
8. A. Finkel and Ph. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256:63–92, Elsevier, 2001.
9. R. Freund. Asynchronous P Systems and P Systems Working in the Sequential Mode Proc. 5th International Workshop on Membrane Computing (WMC5), LNCS 3365, Springer, 2005.
10. G. Higman. Ordering by divisibility in abstract algebras. In *Proc. London Math. Soc.*, vol. 2, pages 236–366, 1952.
11. O. H. Ibarra. Some Recent Results Concerning Deterministic P Systems. Proc 6th International Workshop on Membrane Computing (WMC6), LNCS 3850, Springer, 2006.
12. M.L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, 1967.

13. G. Păun. *Membrane Computing. An Introduction.* Springer, 2002.
14. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
15. C. Priami, A. Regev, W. Silverman, and E. Shapiro. Application of a stochastic passing-name calculus to representation and simulation of molecular processes. Information Processing Letter, 80:25-31, 2001.
16. A. Regev, E. M. Panina, W. Silverman, L. Cardelli, E. Shapiro. BioAmbients: An Abstraction for Biological Compartments. *Theoretical Computer Science*, 325(1):141–167, Elsevier, 2004.
17. J.C. Shepherdson and J.E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.

Analysis of Signalling Pathways Using Continuous Time Markov Chains

Muffy Calder[1], Vladislav Vyshemirsky[2], David Gilbert[2], and Richard Orton[2]

[1] Department of Computing Science, University of Glasgow
[2] Bioinformatics Research Centre, University of Glasgow

Abstract. We describe a quantitative modelling and analysis approach for signal transduction networks.

We illustrate the approach with an example, the RKIP inhibited ERK pathway [CSK+03]. Our models are high level descriptions of continuous time Markov chains: proteins are modelled by synchronous processes and reactions by transitions. Concentrations are modelled by discrete, abstract quantities. The main advantage of our approach is that using a (continuous time) stochastic logic and the PRISM model checker, we can perform quantitative analysis such as *what is the probability that if a concentration reaches a certain level, it will remain at that level thereafter?* or *how does varying a given reaction rate affect that probability?* We also perform standard simulations and compare our results with a traditional ordinary differential equation model. An interesting result is that for the example pathway, only a small number of discrete data values is required to render the simulations practically indistinguishable.

Keywords: signalling pathways; stochastic processes; continuous time Markov chains; model checking; continuous stochastic logic.

1 Introduction

Signal transduction pathways allow cells to sense an environment and make suitable responses. External signals detected by cell membrane receptors activate a sequence of reactions, allowing the cell to recognise the signal and pass it into the nucleus. The cellular response is then activated inside the nucleus. This signalling mechanism is involved in a number of important processes, such as proliferation, cell growth, movement, apoptosis, and cell communication. The pathways include feedback and may be embedded in more complex networks, some form of automated analysis required.

Our aim is to develop quantitative techniques for signal transduction pathway[1] modelling and analysis, based on continuous time and semiquantitative data. Our models are distinctive in two ways. First, we model populations (rather than individuals), that is we model molar concentrations (rather than molecules). Second, we model semiquantitative data. Contemporary methods for biochemical experiments do not, in general, permit the measurement of absolute or continuous

[1] In this paper we use the terms pathway and network synonymously.

values of concentrations. Consequently, some quantitative models are over constrained. We avoid this by considering discrete, abstract concentrations. Thus our treatment of time is continuous, but concentrations are discrete. Our analysis is (model) checking for quantitative, temporal, biological queries.

Our modelling approach is motivated by the observation that signalling pathways have *stochastic*, *computational* and *concurrent* behaviour. Our models are continuous time Markov chains (CTMCs). They are defined, in a natural way, by high level descriptions of concurrent processes: the processes correspond to proteins (the reactants in the pathway) and the transitions to reactions. Concentrations are modelled by discrete, abstract quantities. We use a continuous stochastic logic and the probabilistic symbolic model checker PRISM [KNP02] to express and check a variety of temporal queries for both transient behaviours and steady state behaviours. We can also perform standard simulations and so we compare our results with a traditional ordinary differential equation model. Throughout, we illustrate our approach with an example pathway: the RKIP inhibited ERK pathway [CSK$^+$03].

The paper is organised as follows. In section 2 we present the example pathway. The CTMC model is developed in section 3. In the following section, we discuss analysis by model checking, we present four types of probabilistic, temporal query and give instances for the example pathway. We express the queries in the continuous stochastic logic CSL [BHHK00, ASSB00], and check their validity. In section 5 we discuss how simulations in the stochastic setting compare with simulations in the deterministic setting: in a MATLAB implementation of ordinary differential equations. In section 6 we discuss our results and we review related work in section 7. We conclude in section 8.

2 RKIP and the ERK Pathway

The example system we consider is the RKIP inhibited ERK pathway. We give only a brief overview, further details are presented in [CSK$^+$03, CGH04].

The ERK pathway (also called Ras/Raf, or Raf-1/MEK/ERK pathway) is a ubiquitous pathway that conveys mitogenic and differentiation signals from the cell membrane to the nucleus. The kinase inhibitor protein RKIP inhibits activation of Raf and thus can "dampen" down the ERK pathway.

We consider the pathway as given in the graphical representation of Figure 1. This figure is taken from [CSK$^+$03], where a number of nonlinear ordinary differential equations (ODEs) representing the kinetics are given. We take Figure 1 as our starting point, and explain informally, its meaning. Each node is labelled by a protein (or species). For example, Raf-1*, RKIP and Raf-1*/RKIP are proteins, the last being a complex built up from the first two. A suffix -P or -PP denotes a (single or double, resp.) phosphorylated protein, for example MEK-PP and ERK-PP. Each protein has an associated concentration, given by $m1$, $m2$ etc. *Reactions* define how proteins are built up and broken down. In Figure 1, bi-directional arrows correspond to both forward and backward reactions; uni-directional arrows to forward reactions. Each reaction has a rate

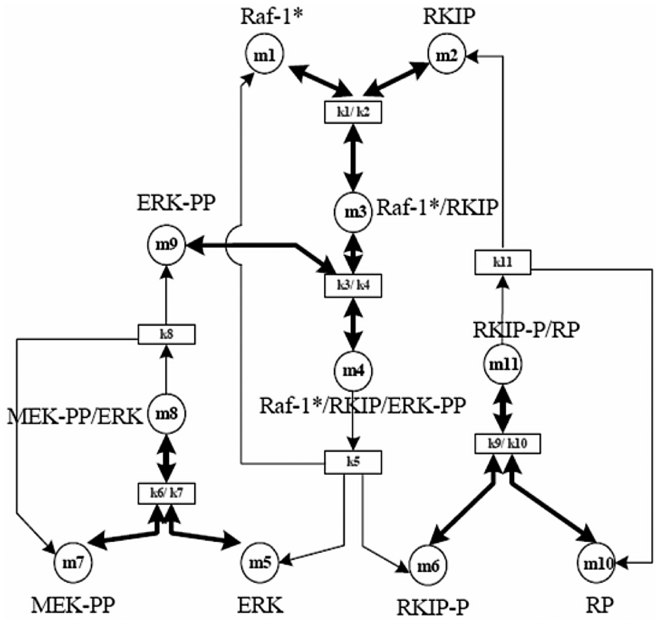


Fig. 1. RKIP inhibited ERK pathway

given by the rate constants $k1$, $k2$, etc. These are given in the rectangles, with $kn/kn+1$ denoting that kn is the forward rate and $kn+1$ the backward rate. Initially, all concentrations are unobservable, except for m_1, m_2, m_7, m_9, and m_{10} [CSK+03].

The dynamic behaviour of the pathway is quite complex, because proteins are involved in more than one reaction and there are several feedbacks. In the next section we develop a model which captures that dynamic behaviour. We note that the example system is part of a larger pathway which can be found elsewhere [KDMH99, SEJGM02].

3 Modelling Signalling Networks by CTMCs

In this section we describe how we model concentrations of proteins by discrete variables, and the dynamic behaviour of proteins by computational processes.

3.1 Discrete Concentrations

Each protein defined in a network has a molar concentration which changes with time, i.e. $m = f(t)$, where m is a concentration of the protein and t is time. As we have indicated earlier, there is a difficulty in obtaining precise and/or continuous concentration values using the methods of contemporary biochemistry. We therefore make discrete abstractions as follows. When the maximum molar concentration is M, then for a given N, the abstract values $0 \ldots N$ represent the concentration intervals $[0, 1 * M/N), [1 * M/N, 2 * M/N), \ldots [N - 1 * M/N, N * M/N]$. We refer to $0 \ldots N$ as *levels* of concentration.

We note that we could define a different N for each protein (depending on experimental accuracy for that species) but in this paper, without loss of generality, we assume the same N, for all proteins.

3.2 Proteins as Processes

We associate a concurrent, computational process with each of the proteins in the network and define these processes using the PRISM modelling language. This language allows the definition of systems of concurrent processes which when synchronised, denote continuous time Markov chains (CTMCs). In sections 3.3 and 3.4 we discuss in detail how CTMCs provide a natural semantics for signalling networks; in this section we focus on the way in which the proteins are represented by PRISM processes (modules) and reactions are represented by transitions.

Below, we give a brief overview of the language, illustrating each concept with a simple example; the reader is directed to [KNP02] for further details of PRISM.

Transitions are labelled with performance rates and (optional) names. For each transition, the performance rate is defined as the parameter λ of an exponential distribution of the transition duration. A key feature is synchronisation: concurrent processes are synchronised on transitions with common names (i.e. the transitions occur simultaneously). Transitions with distinct names are not synchronised. The performance rate for the synchronised transition is the *product* of the performance rates of the synchronising transitions. For example, if process A performs α with rate λ_1, and process B performs α with rate λ_2, then the performance rate of α when A is synchronised with B is $\lambda_1 \cdot \lambda_2$.

As an example, consider the simple single reaction of Figure 2 which describes the binding of (active) Raf-1*, called RAF1 henceforth, and RKIP. Call this reaction $r1$.

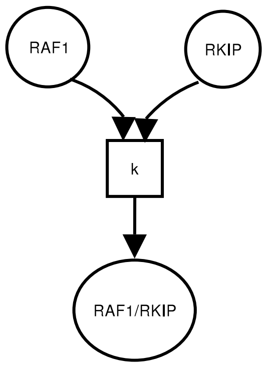


Fig. 2. Simple biochemical reaction $r1$

The PRISM model for this system is listed as Model 1. The model begins with the keyword *stochastic* and consists of some preliminary constants (N and R), four modules: $RAF1$, $RKIP$, $RAF1/RKIP$, and $Constants$, and a system description which states that the four modules should be run concurrently. The constant N defines the concentration levels, as discussed in section 3.1; R is simply an abbreviation for N^{-1}. Consider the first three modules which represent the proteins $RAF1$, $RKIP$ etc. Each module has the form: a state variable which denotes the protein concentration (we use the same name for process and variable, the type can be deduced from context) followed by a single transition named $r1$. The transition has the form *precondition* $\rightarrow$ *rate: assignment*, meaning when the precondition is true, then perform the assignment at the given rate. The rate for transitions of the first two modules is protein concentration multiplied by R, the rate for the third is 1. The assignments in the first two modules decrease the protein level by 1; the level is increased by 1 in the third module. These correspond to the fact that the rate of the reaction is determined by the concentrations of the reactants, and the reactants are *consumed* in the reaction to *produce* $RAF1/RKIP$. But, we must not forget that there is a fourth module, $Constants$; this simply defines the constants for reaction kinetics. In this case the module contains a "dummy" state variable called x, and one (always) enabled transition named $r1$ which defines the rate (i.e. $0.8/R$) for the transition $r1$. For readabilty, our variable and process names include the character '/', strictly speaking, this is not allowed in PRISM.

Since all four transitions have the same name, they will all *synchronise*, and when they do, the resulting transition has rate $\frac{N \cdot R \cdot N \cdot R \cdot 0.8}{R} = 2.4$ ($RAF1$ and $RKIP$ are initialised to N, $RAF1/RKIP$ is initialised to 0, $N = 3$ and $R = 1/3$).

In this simple PRISM model, all the proteins are involved in only one reaction. The reaction can occur three times, until all the $RAF1$ and $RKIP$ has been consumed. Thus in the underlying CTMC, there are 3 transitions (plus a loop) over 4 states, as indicated in Figure 3. The states are labelled by tuples representing the (molar) concentrations of $RAF1$, $RKIP$ and $RAF1/RKIP$, respectively. The edges are labelled with the transition rates.

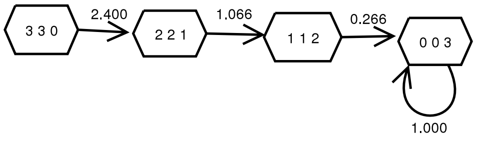


Fig. 3. CTMC for Model 1

In the RKIP inhibited ERK pathway, each protein is involved in several reactions. We model this quite easily by introducing different names ($r1, r2, \ldots$) for each reaction (and the corresponding transitions). We use the convention that reaction rx has rate parameter kx.

Notice that we can describe all the transitions of the processes independently of the number of concentration levels: we simply make the appropriate comparison

Model 1. RAF1 binding to RKIP

```
stochastic

const int N = 3;
const double R = 1/N;

module RAF1
    RAF1: [0..N] init N;
    [r1] (RAF1>0) -> RAF1*R:
           (RAF1' = RAF1 - 1);
endmodule

module RKIP
    RKIP: [0..N] init N;
    [r1] (RKIP>0) -> RKIP*R:
           (RKIP' = RKIP - 1);
endmodule

module RAF1/RKIP
    RAF1/RKIP: [0..N] init 0;
    [r1] (RAF1/RKIP < N) -> 1:
           (RAF1/RKIP' = RAF1/RKIP + 1);
endmodule

module Constants
    x: bool init true;
    [r1] (x=true) -> 0.8/R:
           (x'=true);
endmodule

system
    RAF1 || RKIP || RAF1/RKIP || Constants
endsystem
```

(in the precondition). The size of the complete underlying CTMC depends on N, some examples are:

- when $N = 3$ there are 273 states and $1,316$ transitions;
- when $N = 5$ there are $1,974$ states and $12,236$ transitions;
- when $N = 9$ there are $28,171$ states and $216,282$ transitions.

The full PRISM model for the RKIP inhibited pathway is given in Appendix A.1. In the full model, R is calibrated by the initial concentration(s), i.e. $R = 2.5/N$.

3.3 Continuous Time Markov Chains as Models

The PRISM description allows us to focus on the overall structure of the stochastic system, whilst saving us from the detail of defining the large and complex underlying CTMC.

In this section, we give more detail of the underlying CTMCs and why they provide good, sound models for signalling networks. We assume some familiarity with Markov chains, for completeness we give the following definition of a CTMC.

Definition. Given a finite set of atomic propositions AP, a continuous time Markov chain (CTMC) is a triple (S,R,L) where

- S is a finite set of states
- L: $S \to 2^{AP}$ is labelling of states
- R: $S \times S \to \Re_{\geq 0}$ is a rate matrix.

For a given state s, when $R(s, s') > 0$ for more than one s', then there is a *race* between the outgoing transitions from s. The rates determine *probabilities* according to the "memoryless" negative exponential: when $R(s, s') = \lambda$, then the probability that transition from s to s' completes within time t is $1 - e^{-\lambda t}$.

A path through a CTMC is an alternating sequence $\sigma = s_0 t_0 s_1 t_1 \ldots$ such that $(s_i, s_{i+1}) \in R$ and each time stamp t_i denotes the time spent in state s_i, $\forall i$.

In the case of PRISM system descriptions, the atomic propositions refer to PRISM variables, for example atomic propositions include $RAF1 = 0$, $RKIP = 1$, etc. CTMCs are often presented using a graphical notation, as in Figure 3.

3.4 Soundness of Stochastic Model

In this section we explain why the underlying CTMCs are sound models for signalling networks; in particular, we show how the rates associated with transitions relate to mass action kinetics.

By way of illustration, consider the simple reaction in Figure 2. Recall that for each reaction, a protein is either a *producer* or a *consumer*; thus in the PRISM representation, producers have their concentrations decreased, and consumers have their concentrations increased. Now consider the equations for standard reaction (mass action) kinetics, given by the following:

$$\begin{cases} \dfrac{dm_1}{dt} = -k \cdot m_1 \cdot m_2 \\ \dfrac{dm_2}{dt} = -k \cdot m_1 \cdot m_2 \\ \dfrac{dm_3}{dt} = k \cdot m_1 \cdot m_2 \end{cases}$$

where m_1,m_2, and m_3 are the concentrations of RAF1, RKIP, and RAF1/RKIP respectively.

In the CTMC denoted by the PRISM model (Model 1), from the initial state, the first transition is the synchronisation of four processes ($RAF1$, $RKIP$, $RAF/RKIP$ and $Constants$). Recall the rates for synchronising actions in PRISM are *multiplied*, so the first transition has rate

$$(RAF1 \cdot R) \cdot (RKIP \cdot R) \cdot (k \cdot N) \tag{1}$$

where $k = 0.8$. The crucial question is is *how does this rate compare with, or relate to, the standard mass action semantics?*

First, consider how the concentration variables relate to each other: the ODEs above refer to continuous concentrations, whereas our model has *discrete* natural number levels. Let m be a continuous variable and let m^d be the corresponding PRISM variable (e.g. $RAF1$, $RAF/RKIP$). Then

$$m = m^d \cdot R = m^d \cdot \frac{1}{N} \tag{2}$$

Second, derive a rate expressed in terms of the PRISM variables. From the continuous rate:

$$\frac{dm_3}{dt} = k \cdot m_1 \cdot m_2 \tag{3}$$

the simplest way to derive a new concentration m_3' from m_3 is by Euler's method thus:

$$m_3' = m_3 + (k \cdot m_1 \cdot m_2 \cdot \Delta t) \tag{4}$$

But the abstract (PRISM) concentrations can only increase in units of 1 level of molar concentration, or $\frac{1}{N}$ molars, so

$$\Delta t = \frac{1}{k \cdot m_1 \cdot m_2 \cdot N} \tag{5}$$

PRISM implements rates as the "memoryless" negative exponential, that is for a given rate λ, $P(t) = 1 - e^{-\lambda t}$ is the probability that the action will be completed before time t. Taking λ as $\frac{1}{\Delta t}$, in this example we have

$$\lambda = k \cdot m_1 \cdot m_2 \cdot N \tag{6}$$

Replacing the continuous variables by their abstract forms, we have

$$\lambda = k \cdot (m_1^d \cdot R) \cdot (m_2^d \cdot R) \cdot N \tag{7}$$

or

$$\lambda = k \cdot (RAF1 \cdot R) \cdot (RKIP \cdot R) \cdot N \tag{8}$$

which is exactly the rate given in (1) above.

We can generalise this to arbitrary reactions as follows. In the PRISM description, for a given reaction r, let C^d and P^d be the set of variables representing consumers and producers, respectively. There is a r-named transition defined for each member of C^d and P^d; together they synchronise as a single r-named transition in the underlying CTMC. Let the transition representing that reaction be described by $\lambda : \bigwedge P$, where P is the set of the atomic propositions holding in the target state. Assume λ is defined as described above. The underlying CTMC is given by:

$$\forall c_l \in C_l.\ \forall p_l \in P_l.\ \bigwedge(c_l > 0) \Rightarrow \lambda : \bigwedge(c_l' = c_l - 1) \bigwedge(p_l' = p_l + 1) \tag{9}$$

Note the use of $\Rightarrow$ (logical implication), as opposed to $\rightarrow$ in the PRISM description of a transition. Note also that in the PRISM model, we include the rate factor $(\cdot N)$ in the *Constants* module, and multiply the concentrations by R in the protein processes.

In the next section we give a more detailed example.

3.5 A More Complex Example

Consider the network given in Figure 4; this is more complex than the single reaction given in Figure 2, in particular, some of the arrows are bidirectional.

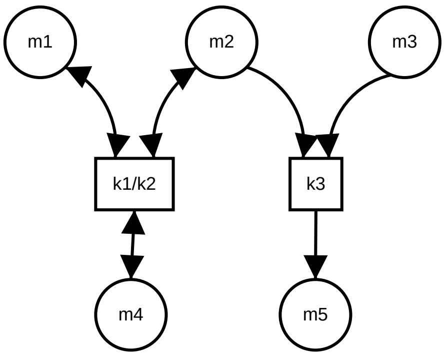


Fig. 4. Network with three reactions, 5 proteins

Assume $N = 2$. The CTMC model for this network is given in Figure 5, tuples of concentration $\langle m_1\ m_2\ m_3\ m_4\ m_5\rangle$ label the states. Since one reaction is reversible, there are simple loops in this topology. (More complicated pathway topologies can produce more nontrivial loops.)

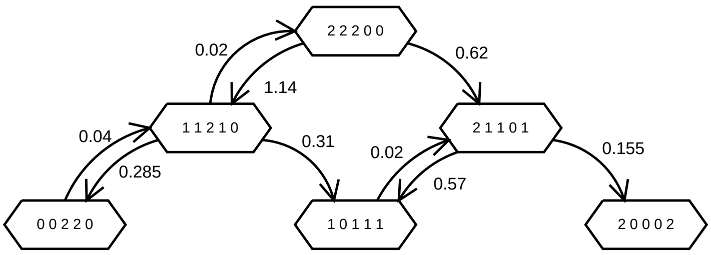


Fig. 5. CTMC for network in Figure 4

Assume rate constants $k1 = 0.57$, $k2 = 0.02$, and $k3 = 0.31$ and initial concentrations $m_1(0) = m_2(0) = m_3(0) = 2$ and $m_4(0) = m_5(0) = 0$. The performance rates for the transitions are calculated as follows:

- $\langle 2\ 2\ 2\ 0\ 0\rangle \longrightarrow \langle 1\ 1\ 2\ 1\ 0\rangle$: $\lambda = \frac{k_1 \cdot \frac{m1}{N} \cdot \frac{m2}{N}}{\frac{1}{N}} = \frac{k_1 \cdot \frac{2}{2} \cdot \frac{2}{2}}{\frac{1}{2}} = 2 \cdot k_1 = 1.14$
- $\langle 1\ 1\ 2\ 1\ 0\rangle \longrightarrow \langle 2\ 2\ 2\ 0\ 0\rangle$: $\lambda = \frac{k_2 \cdot \frac{m4}{N}}{\frac{1}{N}} = \frac{k_2 \cdot \frac{1}{2}}{\frac{1}{2}} = k_2 = 0.02$
- $\langle 2\ 2\ 2\ 0\ 0\rangle \longrightarrow \langle 2\ 1\ 1\ 0\ 1\rangle$: $\lambda = \frac{k_3 \cdot \frac{m2}{N} \cdot \frac{m3}{N}}{\frac{1}{N}} = \frac{k_3 \cdot \frac{2}{2} \cdot \frac{2}{2}}{\frac{1}{2}} = 2 \cdot k_3 = 0.62$

Other performance rates can be calculated exactly in the same way, the results are given in Figure 5.

In conclusion, we propose that CTMCs are good models for networks because they allow us to model performance and nondeterminism explicitly; however, it would be unrealistic to construct a CTMC model manually. The high level modelling abstractions of PRISM allow us to separate system structure from performance and to generate the appropriate underlying rates automatically. Moreover, we can *model check* stochastic properties of CTMCs, a more powerful reasoning mechanism than (stochastic) simulation. For example, consider evaluation of the probability that the state $\langle 2\ 0\ 0\ 0\ 2\rangle$ is reachable. We cannot determine this probability by simulation, since there are an infinite number of paths leading to this state. However, the PRISM model checking algorithms can compute the probabilities (e.g. in the steady state) *automatically*, as we will consider in the next section.

4 Analysis

Temporal logics are powerful tools for expressing temporal queries which may be generic (e.g. state reachability, deadlock) or application specific (e.g. referring to variables representing application characteristics). For example, we can express queries such as *what is the probability that a protein concentration reaches a certain level, and then remains at that level thereafter?*, or *if we vary the rate of a particular reaction, how does this impact that probability?* Whereas simulation is the exploration of a *single* behaviour over a given time interval, model checking allows us to investigate the truth (or otherwise) of temporal queries over (possibly infinite) sets of behaviours over (possibly) unbounded time intervals.

Since we have a stochastic model, we employ the logic CSL (Continuous Stochastic Logic) [BHHK00, ASSB00], and the symbolic probabilistic model checker PRISM [PNK04] to compute validity. We can not only check validity of logical properties, but using PRISM we can analyse open formulae, i.e. we can perform *experiments* as we vary instances of variables in a formula expressing a property. Typically, we will vary reaction rates or concentration levels.

CSL is a continuous time logic that allows one to express a probability measure that a temporal property is satisfied, in either transient behaviours or in steady state behaviours. We assume a basic familiarity with the logic, it is based upon the computational tree logic CTL. The operators of CSL are given in Table 1, for more details see [PNK04]. The $P_{\bowtie p}[\phi]$ properties are *transient*, that is, they depend on time; $S_{\bowtie p}[\phi]$ properties are *steady state*, that is they hold in the long run. To check the latter properties, we use a linear algebra package in PRISM to generate the steady state solution. Note that in this context steady state solutions are not (generally) single states, rather a network of states (with cycles) which define the probability distributions in the long run.

$\bowtie p$ specifies a bound, for example $P_{\bowtie p}[\phi]$ is true in a state s if the probability that ϕ is satisfied by the paths from state s meets the bound $\bowtie p$. Examples of bounds are > 0.99 and < 0.01. A special case of $\bowtie p$ is no bound, in which case we calculate a probability. We write $P_{=?}[\psi]$ which returns the probability, from the initial state, of ψ. If we don't want to start at the initial state, we can apply a filter thus: $P_{=?}[\psi\{\phi\}]$, which returns the probability, from the state satisfying ϕ, of ψ. (Note: if more than one state satisfies ϕ then the first one in a lexicographic ordering is chosen.)

Table 1. Continuous Stochastic Logic operators

Operator	**CSL Syntax**
True	$true$
False	$false$
Conjunction	$\phi \wedge \phi$
Disjunction	$\phi \vee \phi$
Negation	$\neg\phi$
Implication	$\phi \Rightarrow \phi$
Next	$P_{\bowtie p}[\mathbf{X}\phi]$
Unbounded Until	$P_{\bowtie p}[\phi\mathbf{U}\phi]$
Bounded Until	$P_{\bowtie p}[\phi\mathbf{U}^{\leq t}\phi]$
Bounded Until	$P_{\bowtie p}[\phi\mathbf{U}^{\geq t}\phi]$
Bounded Until	$P_{\bowtie p}[\phi\mathbf{U}^{[t_1,t_2]}\phi]$
Steady-State	$S_{\bowtie p}[\phi]$

In the next section we use CSL and PRISM to formulate and check a number of biological queries against our model for the RKIP inhibited ERK pathway. We consider four different kinds of temporal property:

1. steady state analysis of stability of a protein, i.e. a protein reaches a level and then remains there, within certain bounds,
2. transient analysis of monotonic decrease of a protein, i.e. the levels of a protein only decrease,
3. steady state analysis of protein stability when varying reaction rates, i.e. a protein is more likely to be stable for certain reaction rates,
4. transient analysis of protein activation sequence, i.e. concentration peak ordering.

4.1 Stability of Protein in Steady State

This type of property is particularly applicable to the analysis of networks where transient and sustained signal responses can produce markedly different cellular outcomes. For example, a transient signal could lead to cell proliferation, whereas a sustained signal would result in differentiation.

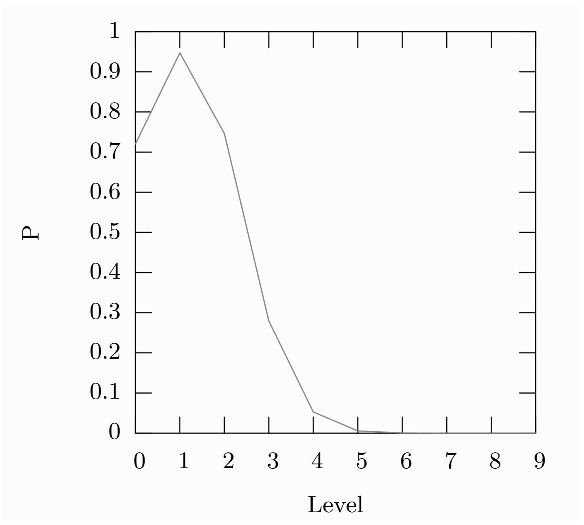


Fig. 6. Stability of $RAF1$ wrt C in steady state

Consider the concentration of $RAF1$. Stability for this protein (within bounds $C-1, C+1$) is expressed by the CSL formula:

$$S_{=?}[(RAF1 \geq C-1) \wedge (RAF1 \leq C+1)] \tag{10}$$

In other words, the level of $RAF1$ is at most 1 increment/decrement away from C. The results are given Figure 6, with C ranging over $0 \ldots 9$ ($N = 9$). They illustrate that $RAF1$ is most likely to be stable at level 1, with a relatively high probability of stability at levels 0 and 2. It is unlikely to be stable at levels 3 or more.

4.2 Monotonic Decrease of Protein

This type of property expresses the notion that the system does not allow an accumulation of a protein. To decide it, consider two properties. The first property is:

$$P_{\geq 1}[(true)\mathbf{U}((Protein = C) \wedge (P_{\geq 0.95}[\mathbf{X}(Protein = C-1)]))] \tag{11}$$

This property expresses "Is it possible to reach a state in which $Protein$ concentration is at level C and after the next step this concentration is $C-1$ with the probability $\geq 95\%$?". Figure 7 illustrates evaluating this property for $RAF1$, with $N = 9$ and C ranging over $1 \ldots 9$: the result is *false* for levels 1 to 4 and *true* for levels 5 to 9.

The second property evaluates the probability of accumulating a protein (assume $RAF1$) within 120 seconds, but only once $RAF1$ has reached a given level of (lower) concentration. The property is defined by :

$$P_{=?}[(true)\mathbf{U}^{\leq 120}(RAF1 > C)\{(RAF1 = C)\}] \tag{12}$$

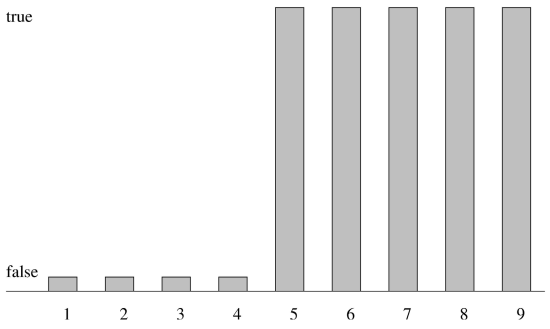


Fig. 7. *RAF*1 decreases with probability 95%

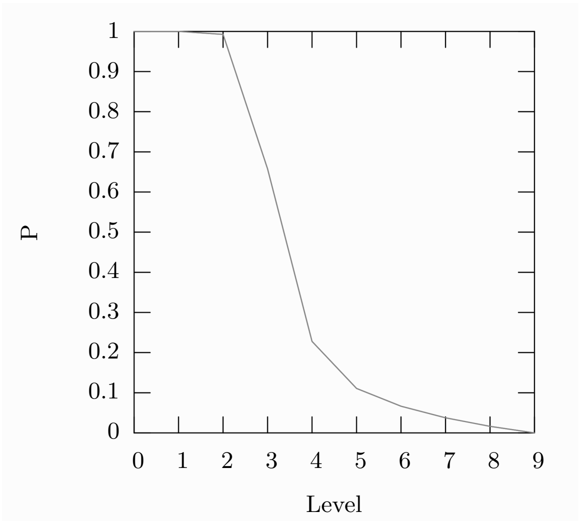


Fig. 8. *RAF*1 increases from a given concentration level

This property expresses "What is the probability of reaching a state with a higher level of *RAF*1 from the state where the concentration level is *C*?" The result is given in Figure 8, with *C* ranging over $0 \ldots 9$.

Note that the first property concerns the probability of protein decrease, from a given level; the second property concerns the probability of exceeding a given level, within a given time. From the combined results of the two experiments, we conclude there is a low probability of accumulating *RAF*1, when the concentration is between levels 5 and 9.

4.3 Protein Stability in Steady State While Varying Rates

This type of property is particularly useful during model fitting, i.e. fitting the model to experimental data. As an example, consider evaluating the probability that *RAF*1 is stable at level 2 or level 3 (in the steady state), whilst varying the performance of the reaction $r1$ (the reaction which binds $RAF1$ and RKIP). We vary the parameter k_1 (which determines the rate of $r1$) over the interval $[0 \dots 1]$. The stability property is expressed by:

$$S_{=?}[(RAF1 \geq 2) \wedge (RAF1 \leq 3)] \tag{13}$$

Consider also the probability that $RAF1$ is stable at levels 0 and 1; the formula for this is:

$$S_{=?}[(RAF1 \geq 0) \wedge (RAF1 \leq 1)] \tag{14}$$

Figure 9 gives results for both these properties, when $N = 5$. The likelihood of property (13) (solid line) peaks at $k1 = 0.03$ and then decreases; the likelihood of property (14) (dashed line) increases dramatically, becoming very likely when $k1 > 0.4$.

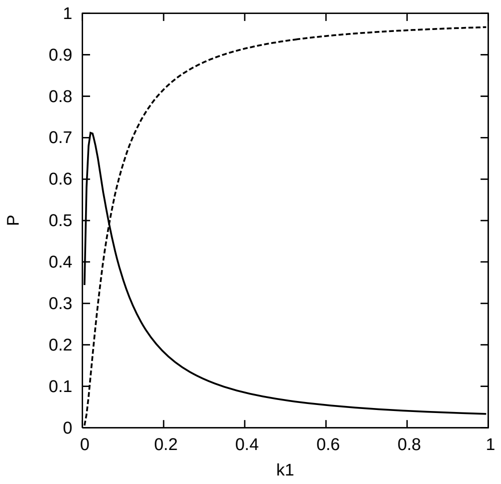


Fig. 9. Stability of $RAF1$ at levels {2,3} and {0,1}

4.4 Activation Sequence Analysis

The last example illustrates queries over several proteins: sequences of protein activations. Consider two proteins: $RAF1/RKIP$ and $RAF1/RKIP/ERK - PP$. Is it possible that the (concentration of the) former "peaks" before the latter? Let M be the peak level.

The formula for this property is:

$$P_{=?}[(RAF1/RKIP/ERK - PP < M)\mathbf{U}(RAF1/RKIP = C)] \tag{15}$$

This property expresses "What is the probability that the concentration of $RAF1/RKIP/ERK - PP$ is less than level M, until $RAF1/RKIP$ reaches concentration level C?" The results of this query, for C ranging over $0, 1, 2$ and M ranging over $1 \dots 5$ are given in Figure 10: the line with steepest slope represents $M = 1$, the line which is nearly horizontal is $M = 5$. For example, the probability

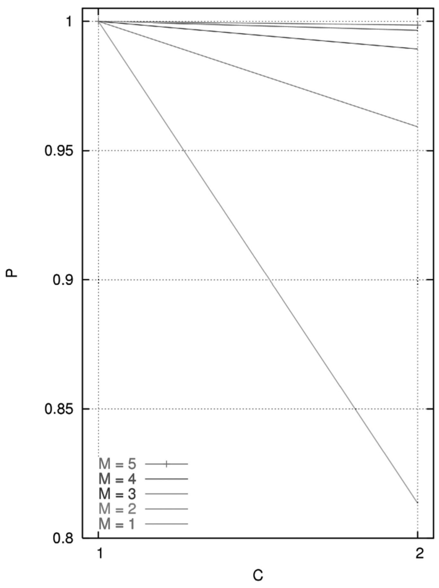


Fig. 10. Activation sequence

$RAF1/RKIP$ reaches concentration level 2 before $RAF1/RKIP/ERK - PP$ reaches concentration level 5 is more than 99%, the probability $RAF1/RKIP$ reaches concentration level 2 before
$RAF1/RKIP/ERK - PP$ reaches concentration level 2 is almost 96%.

To confirm these results, we conducted the inverse experiment – check if it is possible for $RAF1/RKIP/ERK - PP$ to reach concentration level 5 *before* RAF1/RKIP reaches concentration level 2. The property is:

$$P_{=?}[(RAF1/RKIP < C)\mathbf{U}(RAF1/RKIP/ERK - PP = M)] \qquad (16)$$

This property expresses "What is the probability that the concentration of $RAF1/RKIP$ is less than level C until $RAF1/RKIP/ERK - PP$ reaches concentration level M?" The results are given in Figure 11 which is symmetric to Figure 10: for example, the probability $RAF1/RKIP/ERK - PP$ reaches concentration level 5 before $RAF1/RKIP$ reaches concentration level 2 is less than 0.14%.

This concludes our analysis of temporal queries, we now consider using our stochastic model for simulations, and relating those simulations to (deterministic) ODE simulations.

5 Comparison with ODE Simulations

While our primary motivation is analysis with respect to temporal logic properties, it is interesting to consider simulation as well. Our stochastic models permit simulation, in PRISM, using the concept of state rewards [Pri]. For comparison,

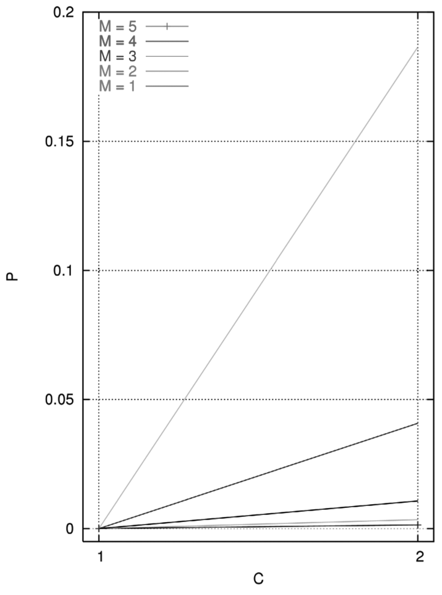


Fig. 11. Inverse activation sequence

we also implemented a standard deterministic model, given by a set of ODEs, in the MATLAB toolset. The ODEs are given in Appendix A.2.

To compare simulation results between the two types of model (i.e. stochastic and deterministic), consider the concentration of phosphorylated MEK, $MEK - PP$, over the time interval $[0 \dots 100]$. Concentration is the vertical axis. Figure 12 plots the results, using the ODE model and two instances of our

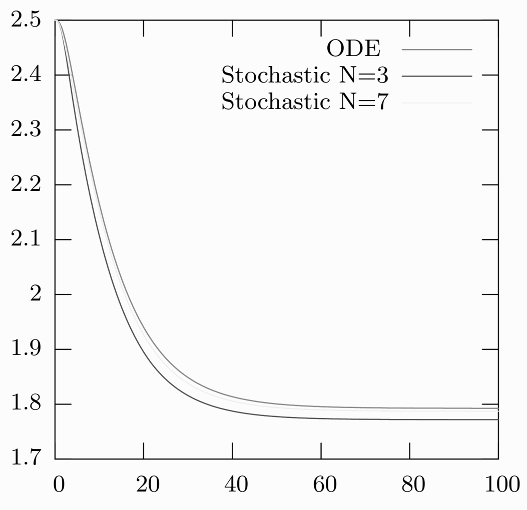


Fig. 12. Comparison ODE and Stochastic models: MEK-PP simulation

stochastic model, with $N = 3$ and $N = 7$. The "upper" curve is the ODE simulation, the "lower" curve is the stochastic simulation, when $N = 3$; the curve in between the two is the stochastic behaviour when $N = 7$. As N increases, the closer the plots; with $N = 7$ the difference is barely discernable.

We make the comparison more precise by defining the distance between the stochastic and deterministic models:

$$\Delta = \sum_{i=1}^{x} \int_0^y (m_i(t) - \tilde{m}_{i,N}(t))^2 \cdot dt$$

where x is the number of proteins, $0 \ldots N$ are concentrations levels in the stochastic model, m_i is the concentration of i^{th} protein in the deterministic model, and $\tilde{m}_{i,N}$ is the concentration of the i^{th} protein in the stochastic model. $[0 \ldots y]$ is the time interval for the comparison. As N increases, the stochastic and deterministic models converge, namely

$$\lim_{N \to \infty} \Delta = 0.$$

Convergence is surpisingly quick. For example Table 2 gives the number of cumulative error metrics over 200 data points, in the time interval [0..100], of the protein $RAF1/RKIP$ (which exhibits the maximum error in this pathway).

Table 2. Error measurements

N	ϵ_a	ϵ_r	$C\epsilon_a$	$C\epsilon_a^2$
3	0.126 mM	0.28	21.557 mM	2.58
4	0.103 mM	0.217	17.569 mM	1.727
5	0.086 mM	0.176	14.582 mM	1.191
7	0.061 mM	0.122	10.402 mM	0.605
11	0.036 mM	0.071	6.042 mM	0.204

The metrics are:

- Maximal absolute error of simulation ϵ_a,
- Maximal relative error of simulation ϵ_r,
- Cumulative absolute error of simulation $C\epsilon_a$,
- Cumulative square error of simulation $C\epsilon_a^2$.

We conclude that in this network, $N = 7$ is quite sufficient to make the two models indistinguishable, for all practical purposes. This was a surprising and very useful result, since computation with small N is tractable on a single processor. This means that for the example network, our stochastic approach offers a new, practical analysis and simulation technique.

6 Discussion

A number of interesting (generic) temporal biological properties were proposed in [CF03], but we have not repeated that analysis here. Rather, we have concentrated on further properties which are specific to signalling networks and population based models. Mainly, we have found steady-state analysis most useful, but we have also illustrated the use of transient properties.

PRISM has been a useful tool for model checking, experimentation, and even simulation. All computations have been tractable on a single standard processor (the times are trivial and have been omitted).

We have assumed that the duration of a reaction is exponentially distributed. We choose the negative exponential because this is the only "memoryless" distribution. Our underlying assumption is that reactions are independent of history, that is they depend only on the current concentration of each reagent. Mass action kinetics are based on a similar assumption. It would be interesting to consider whether other distributions have a physical interpretation, and if so, to investigate how they relate to experimental and statistical results.

In Section 3.4 we showed that our PRISM model relates to mass action kinetics. While simulation is not the primary goal of our approach, in Section 5 we demonstrated that with small N, our model provides (more than) sufficient simulation accuracy, for the example system. This is because the example pathway has reactions which are all on a similar scale. If we were to apply our approach to a pathway where the changes of concentrations are on different scales, i.e. the corresponding ODE model is a set of stiff equations, then we could still reason about the stochastic model using temporal logic queries. However, simulations would not be as accurate, for small N. If more accuracy of simulation was required, then we would have to either increase N, or encode a more sophisticated solver within the PRISM representation (at the expense of transparency).

7 Related Work

The standard models of functional dynamics are ordinary differential equations (ODEs) for population dynamics [Voi00, CSK$^+$03], or stochastic simulations for individual dynamics [Gil77].

The recent work of Regev et al on π-calculus models [RSS01, PRSS01] has been deeply influential. In this work, a correspondence is made between molecules and processes. Here we have proposed a more abstract correspondence between species (i.e. concentrations) and processes. Whereas the emphasis of Regev et al is on simulation, we have concentrated on temporal properties expressed in CSL. More closely related work is presented in [CGH04] where the stochastic process algebra PEPA is used to model the same example pathway. The main advantage is that using the algebra, different formulations of the model can be compared (by bisimulation). One formulation relates clearly to the approach here (proteins as processes) whereas another permits abstraction over sub-pathways. Throughput analysis is main form of qualitative reasoning, though it is possible to "translate" the algebraic models into PRISM and then model check. The algebraic models cannot be used directly for simulation.

Petri nets provide an alternative to Markov chains [PWM03, KH04], with time, hybrid and stochastic extensions [PZHK04, MDNM00, GP98]. However, there are no appropriate model checkers for quantitative analysis (e.g. for stochastic Petri nets), or there are difficulties encoding our nonlinear dynamics (e.g. in time Petri nets), thus we cannot directly compare approaches.

The BIOCHAM workbench [CF03, CRCD+04] provides an interface to the symbolic model checker NuSMV; the interface is based on a simple language for representing biochemical networks. The workbench provides mechanisms to reason about reachability, existence of partially described stable states, and some types of temporal behaviour. However, quantitative model checking is not supported, only qualitative queries can be verified.

8 Conclusions

We have described a new quantitative modelling and analysis approach for signal transduction networks. We model the dynamics of networks by continuous time Markov chains, making discrete approximations to protein molar concentrations. We describe the models in a high level language, using the PRISM modelling language: proteins are synchronous processes and concentrations are discrete, abstract quantities. Throughout, we have illustrated our approach with an example, the RKIP inhibited ERK pathway [CSK+03].

The main advantage of our approach is that using a (continuous time) stochastic logic and the PRISM model checker, we can perform quantitative analysis such as *what is the probability that a protein concentration reaches a certain level and remains at that level thereafter?* and *how does varying a reaction rate affect that probability?* The approach offers considerably more expressive power than simulation or qualitative analysis. We can also perform standard simulations and we have compared our results with traditional ordinary differential equation-based (simulation) methods, as implemented in MATLAB. An interesting and useful result is that in the example pathway, only a small number of discrete data values is required to render the simulations practically indistinguishable. Future work will include the addition of spatial dimensions (e.g. scaffolds) to our models.

Acknowledgements

This research is supported by the project *A Software Tool for Simulation and Analysis of Biochemical Networks*, funded by the DTI Beacon Bioscience Programme.

References

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous time markov chains. *ACM Transactions on Computational Logic*, 1:162–170, 2000.

[BHHK00] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. *In CAV 2000*, 2000.

[CF03] Nathalie Chabrier and Fran+ois Fages. Symbolic model checking of biochemical networks. *Lecture Notes in Computer Science*, 2602:149–162, 2003.

[CGH04] M. Calder, S. Gilmore, and J. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *In Proceedings of Bio-Concur 2004*, 2004.

[CRCD$^+$04] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, Fran+ois Fages, and Vincent Sch+chter. Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 325(1):25–44, 2004.

[CSK$^+$03] K.-H. Cho, S.-Y. Shin, H.-W. Kim, O. Wolkenhauer, B. McFerran, and W. Kolch. Mathematical modeling of the influence of RKIP on the ERK signaling pathway. *Lecture Notes in Computer Science*, 2602:127–141, 2003.

[Gil77] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340 –2361, 1977.

[GP98] Peter J.E. Goss and Jean Peccoud. Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. *Proc. Natl. Acad. Sci. USA (Biochemistry)*, 95:6750 – 6755, 1998.

[KDMH99] B. N. Kholodenko, O. V. Demin, G. Moehren, and J. B. Hoek. Quantification of short term signaling by the epidermal growth factor receptor. *The Journal of Biological Chemistry*, 274(42):30169–30181, October 1999.

[KH04] I. Koch and M. Heiner. Qualitative modelling and analysis of biochemical pathways with petri nets. *Tutorial Notes, 5th Int. Conference on Systems Biology - ICSB 2004*, Heidelberg/Germany, October 2004.

[KNP02] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. *Lecture Notes in Computer Science*, 2324:200–204, 2002.

[MDNM00] H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid petri net representation of gene regulatory network. *Pacific Symposium on Biocomputing*, 5:341–352, 2000.

[PNK04] D. Parker, G. Norman, and M. Kwiatkowska. *PRISM 2.1 Users' Guide.* The University of Birmingham, September 2004.

[Pri] PRISM Web page. http://www.cs.bham.ac.uk/~dxp/prism/.

[PRSS01] C. Priami, A. Regev, W. Silverman, and E. Shapiro. Application of a stochastic name passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80:25–31, 2001.

[PWM03] J.W. Pinney, D.R. Westhead, and G.A. McConkey. Petri Net representations in systems biology. *Biochem. Soc. Trans.*, 31:1513 – 1515, 2003.

[PZHK04] L. Popova-Zeugmann, M. Heiner, and I. Koch. Modelling and analysis of biochemical networks with time petri nets. *Informatik-Berichte der HUB Nr. 170*, 1(170):136 – 143, 2004.

[RSS01] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using π-calculus process algebra. *Pacific Symposium on Biocomputing 2001 (PSB 2001)*, pages 459–470, 2001.

[SEJGM02] B. Schoeberl, C. Eichler-Jonsson, E. D. Gilles, and G. Muller. Computational modelling of the dynamics of the map kinase cascade activated by surface and internalised egf receptors. *Nature Biotechnology*, 20:370–375, April 2002.

[Voi00] E. O. Voit. *Computational Analysis of Biochemical Systems.* Cambridge University Press, 2000.

A Models

A.1 PRISM

The PRISM model is defined by the following; the system description is omitted - it simply runs all modules concurrently.

```
stochastic

const int N = 7;
const double R = 2.5/N;

module RAF1
RAF1: [0..N] init N;

[r1] (RAF1 > 0) -> RAF1*R: (RAF1' = RAF1 - 1);
[r2] (RAF1 < N) -> 1: (RAF1' = RAF1 + 1);
[r5] (RAF1 < N) -> 1: (RAF1' = RAF1 + 1);
endmodule

module RKIP
RKIP: [0..N] init N;

[r1] (RKIP > 0) -> RKIP*R: (RKIP' = RKIP - 1);
[r2] (RKIP < N) -> 1: (RKIP' = RKIP + 1);
[r11] (RKIP < N) -> 1: (RKIP' = RKIP + 1);
endmodule

module RAF1/RKIP
RAF1/RKIP: [0..N] init 0;

[r1] (RAF1/RKIP < N) -> 1: (RAF1/RKIP' = RAF1/RKIP + 1);
[r2] (RAF1/RKIP > 0) -> RAF1/RKIP*R:
              (RAF1/RKIP' = RAF1/RKIP - 1);
[r3] (RAF1/RKIP > 0) -> RAF1/RKIP*R:
              (RAF1/RKIP' = RAF1/RKIP - 1);
[r4] (RAF1/RKIP < N) -> 1: (RAF1/RKIP' = RAF1/RKIP + 1);
endmodule

module ERK-PP
ERK-PP: [0..N] init N;

[r3] (ERK-PP > 0) -> ERK-PP*R: (ERK-PP' = ERK-PP - 1);
[r4] (ERK-PP < N) -> 1: (ERK-PP' = ERK-PP + 1);
[r8] (ERK-PP < N) -> 1: (ERK-PP' = ERK-PP + 1);
endmodule
```

```
module RAF1/RKIP/ERK-PP
RAF1/RKIP/ERK-PP: [0..N] init 0;

[r3] (RAF1/RKIP/ERK-PP < N) -> 1:
(RAF1/RKIP/ERK-PP' = RAF1/RKIP/ERK-PP + 1);
[r4] (RAF1/RKIP/ERK-PP > 0) ->
RAF1/RKIP/ERK-PP*R:
(RAF1/RKIP/ERK-PP' = RAF1/RKIP/ERK-PP - 1);
[r5] (RAF1/RKIP/ERK-PP > 0) ->
RAF1/RKIP/ERK-PP*R:
(RAF1/RKIP/ERK-PP' = RAF1/RKIP/ERK-PP - 1);
endmodule

module ERK
ERK: [0..N] init 0;

[r5] (ERK < N) -> 1: (ERK' = ERK + 1);
[r6] (ERK > 0) -> ERK*R: (ERK' = ERK - 1);
[r7] (ERK < N) -> 1: (ERK' = ERK + 1);
endmodule

module RKIP-P
RKIP-P: [0..N] init 0;

[r5] (RKIP-P < N) -> 1: (RKIP-P' =RKIP-P + 1);
[r9] (RKIP-P > 0) -> RKIP-P*R: (RKIP-P' =RKIP-P - 1);
[r10] (RKIP-P < N) -> 1: (RKIP-P' =RKIP-P + 1);
endmodule

module RP
RP: [0..N] init N;

[r9] (RP > 0) -> RP*R: (RP' = RP - 1);
[r10] (RP < N) -> 1: (RP' = RP + 1);
[r11] (RP < N) -> 1: (RP' = RP + 1);
endmodule

module MEK-PP
MEK-PP: [0..N] init N;

[r6] (MEK-PP > 0) ->  MEK-PP*R: (MEK-PP' = MEK-PP - 1);
[r7] (MEK-PP < N) -> 1: (MEK-PP' = MEK-PP + 1);
[r8] (MEK-PP < N) -> 1: (MEK-PP' = MEK-PP + 1);
endmodule
```

```
module  MEK-PP/ERK
MEKPPERK: [0..N] init 0;

[r6] (MEK-PP/ERK < N) -> 1: (MEK-PP/ERK' =  MEK-PP/ERK + 1);
[r7] (MEK-PP/ERK > 0) ->  MEK-PP/ERK*R:
           (MEK-PP/ERK' =  MEK-PP/ERK - 1);
[r8] (MEK-PP/ERK > 0) ->  MEK-PP/ERK*R:
           (MEK-PP/ERK' =  MEK-PP/ERK - 1);
endmodule

module RKIP-P/RP
  RKIP-P/RP: [0..N] init 0;

[r9] (RKIP-P/RP < N) -> 1: (RKIP-P/RP' = RKIP-P/RP + 1);
[r10] (RKIP-P/RP > 0) -> RKIP-P/RP*R:
           (RKIP-P/RP' = RKIP-P/RP - 1);
[r11] (RKIP-P/RP > 0) -> RKIP-P/RP*R:
           (RKIP-P/RP' = RKIP-P/RP - 1);
endmodule

module Constants
x: bool init true;

[r1] (x) -> 0.53/R: (x' = true);
[r2] (x) -> 0.0072/R: (x' = true);
[r3] (x) -> 0.625/R: (x' = true);
[r4] (x) -> 0.00245/R: (x' = true);
[r5] (x) -> 0.0315/R: (x' = true);
[r6] (x) -> 0.8/R: (x' = true);
[r7] (x) -> 0.0075/R: (x' = true);
[r8] (x) -> 0.071/R: (x' = true);
[r9] (x) -> 0.92/R: (x' = true);
[r10] (x) -> 0.00122/R: (x' = true);
[r11] (x) -> 0.87/R: (x' = true);

endmodule
```

A.2 ODE Model

The ODE based model, given by the following reactions, is implemented in MATLAB toolset. The kinetics are taken from [CSK+03].

1. RAF1 + RKIP → RAF1/RKIP
   $\frac{dv}{dt} = 0.53 \cdot RAF1 \cdot RKIP$
2. RAF1/RKIP → RAF1 + RKIP
   $\frac{dv}{dt} = 0.0072 \cdot RAF1/RKIP$

3. RAF1/RKIP + ERK-PP → RAF1/RKIP/ERK-PP
   $\frac{dv}{dt} = 0.625 \cdot RAF1/RKIP \cdot ERK-PP$
4. RAF1/RKIP/ERK-PP → RAF1/RKIP + ERK-PP
   $\frac{dv}{dt} = 0.00245 \cdot RAF1/RKIP/ERK-PP$
5. RAF1/RKIP/ERK-PP → ERK +RKIP-P + RAF1
   $\frac{dv}{dt} = 0.0315 \cdot RAF1/RKIP/ERK-PP$
6. MEK-PP + ERK → MEK-PP/ERK
   $\frac{dv}{dt} = 0.8 \cdot MEK-PP \cdot ERK$
7. MEK-PP/ERK → MEK-PP + ERK
   $\frac{dv}{dt} = 0.0075 \cdot MEK-PP/ERK$
8. MEK-PP/ERK → MEK-PP + ERK-PP
   $\frac{dv}{dt} = 0.071 \cdot MEK-PP/ERK$
9. RKIP-P + RP →RKIP-P/RP
   $\frac{dv}{dt} = 0.92 \cdot RKIP-P \cdot RP$
10. RKIP-P/RP →RKIP-P + RP
    $\frac{dv}{dt} = 0.00122 \cdot RKIP-P/RP$
11. RKIP-P/RP → RKIP + RP
    $\frac{dv}{dt} = 0.87 \cdot RKIP-P/RP$

The initial concentrations are: $RAF1 = RKIP = ERK-PP = MEK-PP = RP = 2.5$ Molar; all other proteins are 0.

Machine Learning Biochemical Networks from Temporal Logic Properties

Laurence Calzone, Nathalie Chabrier-Rivier, François Fages, and Sylvain Soliman

Projet Contraintes, INRIA Rocquencourt,
BP105, 78153 Le Chesnay Cedex, France
Firstname.Lastname@inria.fr
http://contraintes.inria.fr/

Abstract. One central issue in systems biology is the definition of formal languages for describing complex biochemical systems and their behavior at different levels. The biochemical abstract machine BIOCHAM is based on two formal languages, one rule-based language used for modeling biochemical networks, at three abstraction levels corresponding to three semantics: boolean, concentration and population; and one temporal logic language used for formalizing the biological properties of the system. In this paper, we show how the temporal logic language can be turned into a specification language. We describe two algorithms for inferring reaction rules and kinetic parameter values from a temporal specification formalizing the biological data. Then, with an example of the cell cycle control, we illustrate how these machine learning techniques may be useful to the modeler.

1 Introduction

One promise of systems biology is to model biochemical processes at a sufficiently large scale so that the behavior of a complex system can be predicted under various conditions. The language approach to systems biology aims at designing formal languages for describing biochemical mechanisms, processes and systems at different levels of abstraction. The pioneering use in [1] of the π-calculus process algebra for modeling cell signalling pathways, has been the source of inspiration of numerous works in the line of process calculi [2,3,4] and their stochastic extensions [5].

Recently, the question of formalizing the biological properties of the system has also been raised, and formal languages have been proposed for this task, most notably using temporal logics in either boolean [6,7], discrete [8,9,10] or continuous models [11,12].

The biochemical abstract machine BIOCHAM [13,14] has been designed as a simplification of the process calculi approach using a language of reaction rules that is both more natural to the biologists and well suited to apply model-checking techniques [15]. The rule-based language is used for modeling biochemical networks at three abstraction levels corresponding to three formal semantics:

boolean, concentration and population. In the boolean case, where one reasons only about the presence or absence of molecules, the reaction rules are highly non-deterministic rewriting rules. This setting is similar to Pathway Logic [6] and Petri nets. In the concentration (resp. population) semantics, the rules are equipped with kinetic expressions which provide a continuous dynamics with Ordinary Differential Equations (ODEs) (resp. continuous time Markov chains), somewhat similarly to the bio-calculus [16,17]. One striking feature of this multi-level approach is that in the three cases, temporal logic can be used to formalize the biological properties of the system, and verify them by model-checking techniques.

Turning the temporal logic language into a specification language for expressing the observed behavior of the system opens the way to the use of machine learning techniques for completing or correcting such formal models semi-automatically. There has been work on the use of machine learning techniques, such as inductive logic programming [18], to infer gene functions [19], metabolic pathway descriptions [20,21] or gene interactions [8]. However learning biochemical reactions from temporal properties is quite new, both from the machine learning perspective and from the systems biology perspective.

In this paper, we first describe the basic BIOCHAM language of biochemical processes with its three semantical levels, and the temporal logic language used for formalizing the biological properties of the system. We then present a structural learning algorithm for inferring reaction rules, or more generally model revisions, from a temporal specification of the system at the boolean abstraction level. Next, we detail a similar algorithm for finding kinetic parameter values from a temporal specification at the concentration abstraction level. These algorithms and their implementation in BIOCHAM[1] are illustrated in Sect. 6 on a model of the cell cycle control after Qu et al. [22], with examples of model revision, refinement and parameter search. Finally, we conclude on the merits of this approach, its limits and on some perspectives for future work.

2 BIOCHAM's Description Language of Biochemical Processes

2.1 Syntax

BIOCHAM rules represent biomolecular reactions between chemical or biochemical compounds, ranging from small molecules to proteins and genes. The syntax of the formal objects involved, and their reactions, is given by the following (simplified) grammar:

[1] BIOCHAM is a free software implemented in Prolog and distributed under the GPL license. It is downloadable on the web at `http://contraintes.inria.fr/BIOCHAM`. The data used in this paper are available at `http://contraintes.inria.fr/BiochamLearning/TCSB`

object   = molecule | molecule :: location
molecule = name | molecule-molecule |molecule$\sim${name,. . . ,name}
reaction = solution => solution | kinetics *for* solution => solution
solution = _ | object | number*object | solution + solution

The basic object is a molecular compound. Thanks to the "::" operator, it can be given a precise location, which is a simple name representing a (fixed) compartment, such as the nucleus, the cytoplasm, the membrane, etc. The binding operator "–" is used to represent complexations and other forms of intermolecular bindings. The alteration operator "$\sim$" makes it possible to attach to a compound a set of modifications, such as the set of phosphorylated sites of a protein. For instance, `A~{p}` denotes a phosphorylated form of the compound `A`, and `A~{p}-B` denotes its complexation with `B`.

Reaction rules transform one formal solution into another one. The following abbreviations are used: `A =[C]=> B` for the catalyzed reaction `A+C => C+B`, and `A <=> B` for the reversible reaction equivalent to the two symmetrical reactions `A => B` and `B => A`. The constant "_" represents the empty solution. It is used for instance in protein *degradation rules*, like `A => _`, and in *synthesis rules*, like `_ =[G]=> A` for the synthesis of `A` by the (activated gene) catalyst `G`. The other main rule schemas are *(de)complexation rules*, like `A + B => A-B` for the complexation of `A` and `B`, *(de)phosphorylation rules*, like `A =[B]=> A~{p}` for the phosphorylation of `A` catalyzed by the kinase `B`, and *transport rules*, like `A::nucleus => A::cytoplasm` for the transport of `A` from the nucleus to the cytoplasm.

Reactions can be given kinetic expressions. For instance,
`k*[A]*[B] for A=[B]=>A~{p}` specifies a mass action law kinetics with parameter k for the reaction. These expressions can be written explicitly, allowing any kinetics, or using shortcuts like `MA(k)` for a Mass Action law with parameter k, or `MM(Vm,Km)` for a Michaelian kinetics.

For an example of a complete model, we refer to the Appendix A, which contains a transcription of the model of the cell cycle control of Qu et al. [22], explained in Sect. 6.

BIOCHAM also offers a rich language of patterns and constraints [23] used to denote sets of objects or reaction rules, in a concise manner. A rule pattern basically replaces, in a rule expression, some objects by variables, written `$A, $B`, and adds constraints on the values these variables can take. For instance, the constraint `$A diff $B` imposes that the two molecules are different, or `$A more_phos_than $B` imposes that `$A` is more phosphorylated than `$B`. The precise description of patterns is however beyond the scope of this article, and we refer to [14] for details. Patterns are used to write large models in a concise manner, and to specify the kinds of rules to learn as explained in Sect. 4.

2.2 Boolean Semantics

The most abstract semantics of BIOCHAM rules is the boolean one. In that semantics, one associates to each BIOCHAM object a boolean variable representing its presence or absence in the system. Reaction rules are then interpreted as an *asynchronous transition system* over states defined by the vector of boolean variables. A rule such as `A+B=>C+D` defines four possible transitions corresponding to the complete or incomplete consumption of the reactants `A` and `B`. Such a rule can only be applied when both `A` and `B` are present in the current state. In the next state, `C` and `D` are then present, while `A` and `B` can either be present (partial consumption) or absent (complete consumption). This choice is made in a non-deterministic fashion, in order to *over-approximate* all the possible behaviors of the system. The transition system is asynchronous, only one rule is applied at a given time, in order to represent the basic biological phenomena such as competitive inhibition, where a reaction "hides" another one. On the other hand, the hypothesis of synchrony (i.e. all rules that can be fired in a given state are fired simultaneously) would not faithfully represent the competition between reactions.

The boolean semantics is thus highly non-deterministic. The temporal evolution of the system is modeled by the succession of states in a path, and the different possible behaviors by the non-deterministic choice of a transition at each step.

The Appendix B shows a standard graphical representation of the model in Appendix A where the degree of non-determinism can be roughly estimated visually by the number of edges outgoing from the circular nodes representing the molecules.

Formally, the boolean semantics of BIOCHAM rules is defined via a *Kripke structure* $K = (S, R)$ where S is the set of states defined by the vector of boolean variables, and $R \subseteq S \times S$ is the transition relation between states, supposed to be total (i.e. $\forall s \in S, \exists s' \in S$ s.t. $(s, s') \in R$). When the transition relation defined by the reaction rules of a BIOCHAM model is not total, it is automatically completed by loops on states having no successor. A path in K, starting from state s_0 is an infinite sequence of states $\pi = s_0, s_1, \cdots$ such that $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. In the following, we will denote by π^k the path $s_k, s_{k+1}, \cdots$.

2.3 Concentration Semantics

The concentration semantics is a more concrete semantics, where one associates to each BIOCHAM object a real number representing its concentration. Reaction rules are interpreted with their kinetic expressions by a set of nonlinear ordinary differential equations (ODE)[2]. Formally, to a set of BIOCHAM reaction rules $E = \{e_i \texttt{ for } S_i \texttt{ => } S_i'\}_{i=1,\ldots,n}$ with variables $\{x_1, \ldots, x_m\}$, one associates the system of ODEs :

$$dx_k/dt = \sum_{i=1}^{n} r_i(x_k) * e_i - \sum_{j=1}^{n} l_j(x_k) * e_j$$

[2] The kinetic expressions in BIOCHAM can actually contain conditional expressions, in which case the reaction rules are interpreted by a deterministic hybrid automaton. For the sake of simplicity, we shall not detail this more general setting here.

where $r_i(x_k)$ (resp. l_i) is the stoichiometric coefficient of x_k in the right (resp. left) member of rule i.

Given an initial state, i.e. initial concentrations for each of the objects, the evolution of the system is deterministic, and numerical integration algorithms compute a time series describing the temporal evolution of the system variables. The integration methods actually implemented in BIOCHAM are the adaptive step-size Runge-Kutta method and the Rosenbrock implicit method for stiff systems, which both produce simulation traces with variable time steps.

Figures 1, 2 and 3 in Sect. 6 show graphical views of the result of such computations in the model of Qu et al. with different parameter values.

2.4 Population Semantics

The population semantics is the most realistic semantics but also the most difficult to compute. This semantics associates to each BIOCHAM object an integer representing the number of molecules in the system. Rules are interpreted as a continuous time Markov chain where transition probabilities are defined by the kinetic expressions of BIOCHAM reaction rules.

Stochastic simulation techniques [24] compute realizations of the process. The results are generally noisy versions of those obtained with the concentration semantics. However, in models with, for instance, very few molecules of some kind, qualitatively different behaviors may appear in the stochastic simulation, and thus justify the recourse to that semantics in such cases. A classical example is the model of the lambda phage virus [25] in which a small number of molecules, promotion factors of two genes, can generate an explosive multiplication (lysis) after a more or less long period of passive wait (lysogeny).

In the population semantics, for a given volume V_i of the location where the reaction occurs, a concentration C is translated into a number of molecules $N = C \times V_i \times K$, where K is Avogadro's number. The kinetic expression e_i for the reaction i is converted into a transition rate τ_i (giving a transition probability after normalization) as follows [25]:

$$\tau_i = e_i \times (V_i \times K)^{(1-\sum_{k=1}^{m} l_i(x_k))} \times \prod_{k=1}^{m} (!l_i(x_k))$$

where l_i is the stoichiometric coefficient of the reactant x_k in the reaction rule i. In particular we have:

- $\tau_i = e_i$ for reactions of the form `A =>...`,
- $\tau_i = \frac{e_i}{V_i \times K}$ for reactions of the form `A+B=>...`,
- $\tau_i = 2 \times \frac{e_i}{V_i \times K}$ for reactions of the form `A+A=>...`,
- etc.

3 BIOCHAM's Description Language of Biological Properties

A second language based on temporal logic is used in BIOCHAM to formalize the biological properties of a system. The temporal logics CTL (*Computation Tree*

Logic), LTL (*Linear Time Logic*) and PLTL (*Probabilistic LTL*) with numerical constraints are used in the three semantics respectively. They are used to express in a formal manner the biological properties of the system that the model is supposed to capture. These properties correspond to the biological experiments (observations and measures done in wild-life or mutated organisms, etc.) that are used to build and validate the model.

Their formalization makes it possible to consider these properties as a *specification* to be preserved through operations of model correction, refinement, simplification or composition. The possibility to verify automatically such a specification with model-checking techniques opens the way to the curation, comparison and re-use of different models in a much more systematic fashion than what the current state-of-the-art permits.

3.1 CTL for the Boolean Semantics

The *Computation Tree Logic* CTL* [15] is an extension of classical logic that allows reasoning about an infinite tree of state transitions. It uses operators about branches (non-deterministic choices) and time (state transitions). Two path quantifiers A and E are thus introduced to handle non-determinism: $A\phi$ meaning that ϕ is true on all branches, and $E\phi$ that it is true on at least one branch. The time operators are F, G, X, U and W; $X\phi$ meaning ϕ is true at the next transition, $G\phi$ that ϕ is always true, $F\phi$ that ϕ is eventually true, $\phi\ U\ \psi$ meaning ϕ is always true until ψ becomes true, and $\phi\ W\ \psi$ meaning ϕ is always true until ψ might become true. In this logic, $F\phi$ is equivalent to $true\ U\ \phi$, $\phi\ W\ \psi$ to $(\phi\ U\ \psi)|G\phi$, and the following duality properties hold: $!(EF(\phi)) = AG(!\phi)$, $!(E\ \phi\ U\ \psi) = A(!\psi\ W\ !\phi)$ and $!(E\phi\ W\ \psi) = A(!\psi\ U\ !\phi)$, where ! denotes negation.

In the CTL fragment of CTL* used in BIOCHAM for implementation reasons, each temporal operator must be preceded by a path operator, and each path operator has to be immediately followed by a temporal operator. Table 1 defines the truth value of a formula in a Kripke structure where states are defined by boolean variables.

A BIOCHAM model $\mathcal{M}$ is defined by a set of rules and a set of possible initial states. The rules of $\mathcal{M}$ define the Kripke structure K associated to the model. As the initial state may not be completely defined, we distinguish between the truth of a formula ϕ in all initial states of $\mathcal{M}$, noted $M \models Ai\ \phi$ (or simply $M \models \phi$), and the truth of ϕ in some initial state of $\mathcal{M}$, noted $M \models Ei\ \phi$.

We refer to [7,26,13,23], Sect. 6.3 and to Appendix A for examples of biological properties of a system expressed by CTL formulae. Some of the most used CTL formulae are abbreviated in BIOCHAM as follows:

– `reachable(P)` stands for $EF(P)$;
– `steady(P)` stands for $EG(P)$;
– `stable(P)` stands for $AG(P)$;
– `checkpoint(Q,P)` stands for $!E(!Q\ U\ P)$;
– `oscil(P)` stands for $AG((P \Rightarrow EF\ !P) \wedge (!P \Rightarrow EF\ P))$.
– `loop(P,Q)` stands for $AG((P \Rightarrow EF\ Q) \wedge (Q \Rightarrow EF\ P))$.

Table 1. Inductive definition of the truth value of a CTL* formula in a given state s or path π, for a Kripke structure K

$s \models \alpha$	iff α is a propositional formula true in the state s,
$s \models E\psi$	iff there exists a path π starting from s s.t. $\pi \models \psi$,
$s \models A\psi$	iff for all paths π starting from s, $\pi \models \psi$,
$s \models !\psi$	iff $s \not\models \psi$,
$s \models \psi \ \& \ \psi'$	iff $s \models \psi$ and $s \models \psi'$,
$s \models \psi \mid \psi'$	iff $s \models \psi$ or $s \models \psi'$,
$s \models \psi \Rightarrow \psi'$	iff $s \models \psi'$ or $s \not\models \psi$,
$\pi \models \phi$	iff $s \models \phi$ where s is the first state of π,
$\pi \models X\psi$	iff $\pi^1 \models \psi$,
$\pi \models F\psi$	iff there exists $k \geq 0$ s.t. $\pi^k \models \psi$,
$\pi \models G\psi$	iff for all $k \geq 0$, $\pi^k \models \psi$,
$\pi \models \psi \ U \ \psi'$	iff there exists $k \geq 0$ s.t. $\pi^k \models \psi'$ and $\pi^j \models \psi$ for all $0 \leq j < k$.
$\pi \models \psi \ W \ \psi'$	iff either there exists $k \geq 0$ s.t. $\pi^k \models \psi'$ and $\pi^j \models \psi$ for all $0 \leq j < k$, or for all $k \geq 0$, $\pi^k \models \psi$.
$\pi \models !\psi$	iff $\pi \not\models \psi$,
$\pi \models \psi \ \& \ \psi'$	iff $\pi \models \psi$ and $\pi \models \psi'$,
$\pi \models \psi \mid \psi'$	iff $\pi \models \psi$ or $\pi \models \psi'$,
$\pi \models \psi \Rightarrow \psi'$	iff $\pi \models \psi'$ or $\pi \not\models \psi$,

Without strong fairness assumption, it is worth noting that the last two abbreviations are actually necessary but not sufficient conditions for oscillations. The correct formula for oscillations is indeed a CTL* formula that cannot be expressed in CTL: $EG((P \Rightarrow F\ !P) \wedge (!P \Rightarrow F\ P))$ [27].

The abbreviations can be used inside CTL formulae. For instance, the formula `reachable(steady(P))` expresses that the steady state denoted by formula P is reachable, or the formula `AG(!P -> checkpoint(Q,P))` expresses that Q is a checkpoint for P not only in the initial state but in all reachable states.

3.2 LTL with Numerical Constraints for the Concentration Semantics

The *Linear Time Logic*, LTL is the fragment of CTL* that uses only temporal operators. A first-order version of LTL is used to express temporal properties about the molecular concentrations in the simulation trace. A similar approach is used in the DARPA BioSpice project [11]. The choice of LTL is motivated by the fact that the concentration semantics given by ODEs is deterministic, and there is thus no point in considering path quantifiers.

The version of LTL with arithmetic constraints we use, considers first-order atomic formulae with equality, inequality and arithmetic operators ranging over real values of concentrations and of their derivatives. For instance `F([A]>10)` expresses that the concentration of `A` eventually gets above the threshold value 10. `G([A]+[B]<[C])` expresses that the concentration of C is always greater than the sum of the concentrations of A and B. Oscillation properties, abbreviated

as `oscil(M,K)`, are defined as a change of sign of the derivative of M at least K times:
`F((d[M]/dt > 0) & F((d[M]/dt < 0) & F((d[M]/dt > 0)...)))`. The abbreviated formula `oscil(M,K,V)` adds the constraint that the maximum concentration of M must be above the threshold V in at least K oscillations.

For practical purposes, some limited forms of quantified first-order LTL formulae are also allowed. As an example of this, constraints on the periods of oscillations can be expressed with a formula such as `period(A,75)`, defined as $\exists t\, \exists v\, F(Time = t \;\&\; [A] = v \;\&\; d([A])/dt > 0 \;\&\; X(d([A])/dt < 0) \;\&\; F(Time = t + 75 \;\&\; [A] = v \;\&\; d([A])/dt > 0 \;\&\; X(d([A])/dt < 0)))$ where $Time$ is the time variable.

Note that the notion of *next state* (operator X) refers to the state of the following time point computed by the (variable step-size) simulation, and thus does not necessarily imply real-time neighborhood. Nevertheless, for computing for instance local maxima as in the formula above, the numerical integration methods do compute the relevant time points with a good accuracy.

3.3 PLTL with Integer Constraints for the Population Semantics

For the stochastic semantics, it is natural to consider the PCTL logic [28] which basically replaces the path operators of CTL, E and A, by the operator $P_{\bowtie p}$, which represents a constraint $\bowtie_p$ on the probability that the formula under $P_{\bowtie p}$ is true. For instance, $A(\psi\ U\ \psi')$ becomes $P_{\geq 1}(\psi\ U\ \psi')$, i.e. the probability that $\psi\ U\ \psi'$ is realized is 1. The atomic formulae considered here are first-order formulae with arithmetic constraints, ranging on integers representing numbers of molecules.

However, for efficiency reasons explained in Sect. 5.3, a fragment of PCTL formulae, called PLTL, in which the $P_{\bowtie p}$ operator can only appear once as head of the formula, is actually considered in BIOCHAM.

4 Learning Reaction Rules from CTL Properties

The description language of biological properties based on temporal logic can be used to *specify* the expected behavior of a system, and to let a machine learning algorithm automatically search for possible model revisions. In BIOCHAM, the structural learning of reaction rules relies on the boolean semantics. A CTL specification is thus used to formalize the expected temporal properties of the model, in the process of learning rules.

4.1 Symbolic Model-Checking Algorithm

The learning algorithm necessitates to check the truth of CTL formulae, and makes use of counter-examples to direct the search of corrections to the model. In BIOCHAM, the CTL formulae are evaluated through an interface to the symbolic model-checker NuSMV [29]. NuSMV also computes counter-examples in the form of pathways, which is used, for instance, in the search process to revise the model.

The performances obtained on a large model of the cell cycle control after Kohn's map [30], involving 800 rules and 500 variables, have been shown to be of the order of a few tenths of seconds to compile the model, and check simple CTL formulae [26]. These performances are nevertheless far below those obtained classically with NuSMV on much more strongly structured models of circuits or programs. One characteristic of the boolean models of BIOCHAM is their very high degree of non-determinism. This may lead to performance problems on small size models having a high number of parallel pathways, which is the case for instance in the MAPK model of [31]. These difficulties are recognized in the symbolic model-checking community [32] and biochemical networks provide interesting examples of problems with a high circuit width [33]. On the other hand, the most efficient model-checking tools based on a representation with Petri nets assume a 1-boundedness condition stating that at most one token can be present in a place [34], which is generally not satisfied in biochemical networks.

4.2 Learning One Rule

The boolean semantics of BIOCHAM can be used straightforwardly to search for one rule to add to, or suppress from, a model in order to satisfy a CTL specification. The search can be restricted by providing rule patterns with constraints.

The command `learn_one_addition(reaction_pattern,spec_CTL)` enumerates each instance of the rule pattern provided as first argument, that, when added to the model, is sufficient to satisfy the CTL specification given as second argument (or given implicitly by the command `add_spec` if there is no second argument). The time complexity grows linearly in the number of instances of the rule pattern, which is thus the main complexity factor added to the checking of the CTL specification.

The command `learn_one_deletion(reaction_pattern,spec_CTL)` enumerates each instance of the rule pattern that is sufficient to delete from the model to satisfy the CTL specification. Here, the time complexity added to the checking of the CTL specification is linear in the number of rules in the model.

It is worth noting that, by iterating the search of rules that can be deleted in a model while preserving its specification, one can easily implement a command `reduce_model(spec_CTL)` to compute a *minimal* subset of rules satisfying a given specification.

4.3 Learning Several Rules

In order to guide the automatic search for the addition and deletion of several rules, the CTL formulae can be divided into three classes. The ECTL class regroups CTL formulae that do not contain the A operator (i.e. no positive occurrence of A or negative occurrence of E). The ACTL class regroups CTL formulae that do not contain the E operator. The other formulae are regrouped in the UCTL class. The reason for this classification is that, in order to satisfy a false ACTL formula it is necessary to *delete* rules from the model, whereas to satisfy a false ECTL formula, it is necessary to *add* rules to the model:

Proposition 1. *Let $K = (S, R)$ and $K' = (S, R')$ be two Kripke structures such that $R \subseteq R'$. For any ECTL formula ϕ, if $s \not\models_{K'} \phi$ then $s \not\models_K \phi$. For any ACTL formula ϕ, if $s \not\models_K \phi$ then $s \not\models_{K'} \phi$.*

Proof. Let us prove the proposition for ECTL formulae by contrapositive, i.e. if $s \models_K \phi$ then $s \models_{K'} \phi$. The proof is done by induction on the size of the proof of $s \models_K \phi$, with ϕ supposed to be in negation-free form, except inside propositional formulae, and using only the temporal operators X, G and U, thanks to the equivalences and duality properties given in section 3.1.

If $\phi = \alpha$ is propositional, we have α is true in the state s, and since s is also a state of K' we get $s \models_{K'} \phi$.

If $\phi = \psi_1 \ \& \ \psi_2$ (resp. $\psi_1 \ | \ \psi_2$) then we have by definition $s \models_K \psi_1$ and (resp. or) $s \models_K \psi_2$, by induction hypothesis we get $s \models_{K'} \psi_1$ and (resp. or) $s \models_{K'} \psi_2$ and we can conclude that $s \models_{K'} \phi$.

The only remaining case is when $\phi = E\psi$ with ψ starting with a temporal operator. We know that there exists a path π of K such that $\pi \models_K \psi$. Note that since $R \subseteq R'$, π is also a path in the structure K'. Now,

– if $\psi = X\psi'$, we have $\pi^1 \models_K \psi'$, by induction hypothesis, $\pi^1 \models_{K'} \psi'$ and since π is a path of K' we get $\pi \models_{K'} X\psi'$, q.e.d.
– if $\psi = G\psi'$, we have $\pi^n \models_K \psi'$ for all $n \geq 0$, by applying the induction hypothesis to all states π^n, we get $\pi^n \models_{K'} \psi'$ for all $n \geq 0$, and since π is a path of K' we get $\pi \models_{K'} G\psi'$, q.e.d.
– if $\psi = \psi' \ U \ \psi''$, we have $\pi^n \models_K \psi''$ for some $n \geq 0$ and $\pi^m \models_K \psi'$ for all $0 \leq m < n$, by applying the induction hypothesis to all π^m for $0 \leq m \leq n$ we get, $\pi^n \models_{K'} \psi''$, and $\pi^m \models_{K'} \psi'$ for all $0 \leq m < n$, thus $\pi \models_{K'} \psi' \ U \ \psi''$, q.e.d.

For an ACTL formula ϕ, we once again prove the proposition by contrapositive and induction on the proof of $s \models_{K'} \phi$. The only difference with the above proof is when we get to the case $\phi = A\psi$, we have to prove that $\pi \models_K \psi$ for all paths π starting from s in K but since they are all paths of K' we can apply the induction hypothesis and the same reasoning as above applies. We get $s \models_K \phi$, q.e.d. □

Following this proposition, model revision algorithms allowing to add and delete several rules in a model, can be defined by fixing strategies for choosing the next CTL formula to satisfy, and for searching the rules to add to, or delete from the the model, according to the class of the formula.

In BIOCHAM, the command

```
revise_model(reaction_pattern, spec_CTL)
```

implements such a model revision algorithm. It enumerates sets of rule additions and deletions that are sufficient to satisfy the specification. The CTL specification, as well as a reaction pattern for the rules allowed to be added or deleted, can be provided as arguments in the command.

To satisfy an unsatisfied ECTL formula, the algorithm tries to add one rule among the instances of the pattern, and checks it by model-checking. To satisfy an unsatisfied ACTL formula, the algorithm searches for rule deletions only, among the pattern instances that are in the set of the rules provided by the model-checker as a counter-example. To satisfy an unsatisfied UCTL formula, the algorithm tries both adding and removing rules, as described formally below. More than one set of additions/deletions being possible at each step, the algorithm implements a backtracking procedure to explore all choices.

ECTL formulae are treated first, UCTL second and then ACTL. However, the deletions necessary to satisfy an ACTL formula are allowed to dissatisfy some ECTL or UCTL formulae, in which case they are treated again with the constraint not to dissatisfy the ACTL formulae already satisfied.

Formally, we present the *model revision algorithm* by a non-deterministic transition system over configurations of the form $[Q_t, Q, R]$ where:

- R is the current set of reaction rules.
- $Q_t = (E_t, U_t, A_t)$ is the set of *treated* CTL formulae that are true in R. These formulae are sorted in three groups: the set E_t of ECTL formulae, the set U_t of unclassified (UCTL) formulae, and the set A_t of ACTL formulae.
- $Q = (E, U, A)$ is the set of *untreated* CTL formulae, sorted similarly in three groups of ECTL, UCTL and ACTL formulae.

By a slight abuse of notation, we write $R \models \phi$ if the rule set R satisfies the CTL formula ϕ. The rules considered for addition (noted r in the transitions E' and U' below) are instances of the rule pattern. The rule sets considered for deletion (noted Re in transitions U" and A' below) are chosen among the rules computed by the model-checker as a counter-example to the formula.

The initial configuration is the configuration $[(\emptyset, \emptyset, \emptyset),\ (E, U, A),\ R]$, that is, we start with a model R and an untreated CTL specification. The model revision algorithm performs the following transitions:

E: $[(E_t, U_t, A_t),\ (E \cup \{e\}, U, A),\ R] \rightarrow [(E_t \cup \{e\}, U_t, A_t),\ (E, U, A),\ R]$
if $R \models e$

E': $[(E_t, U_t, A_t),\ (E \cup \{e\}, U, A),\ R] \rightarrow [(E_t \cup \{e\}, U_t, A_t),\ (E, U, A),\ R \cup \{r\}]$
if $R \not\models e$ and $\forall f \in \{e\} \cup E_t \cup U_t \cup A_t \quad R \cup \{r\} \models f$

U: $[(E_t, U_t, A_t),\ (\emptyset, U \cup \{u\}, A),\ R] \rightarrow [(E_t, U_t \cup \{u\}, A_t),\ (\emptyset, U, A),\ R]$
if $R \models u$

U': $[(E_t, U_t, A_t),\ (\emptyset, U \cup \{u\}, A),\ R] \rightarrow [(E_t, U_t \cup \{u\}, A_t),\ (\emptyset, U, A),\ R \cup \{r\}]$
if $R \not\models u$ and $\forall f \in \{u\} \cup E_t \cup U_t \cup A_t \quad R \cup \{r\} \models f$

U": $[(E_t, U_t, A_t),\ (\emptyset, U \cup \{u\}, A),\ R \cup Re] \rightarrow [(E_t, U_t \cup \{u\}, A_t),\ (\emptyset, U, A),\ R]$
if $R \cup R_e \not\models u$ and $\forall f \in \{u\} \cup E_t \cup U_t \cup A_t \quad R \models f$

A: $[(E_t, U_t, A_t),\ (\emptyset, \emptyset, A \cup \{a\}),\ R] \rightarrow [(E_t, U_t, A_t \cup \{a\}),\ (E_p, U_p, A),\ R]$
if $R \models a$

A': $[(E_t \cup E', U_t \cup U', A_t),\ (\emptyset, \emptyset, A \cup \{a\}),\ R \cup Re] \rightarrow [(E_t, U_t, A_t \cup \{a\}),\ (E', U', A),\ R]$
if $R \cup Re \not\models a$, $\forall f \in \{a\} \cup E_t \cup U_t \cup A_t \quad R \models f$ and $\forall g \in E' \cup U' \ R \not\models g$.

Proposition 2 (termination and correctness). *Given a CTL specification (E, U, A), the model revision algorithm terminates in at most $|A| * |E \cup U|$ transitions. Moreover, if the terminal configuration is of the form $[(E_t, U_t, A_t), (\emptyset, \emptyset, \emptyset),$*

R] then the revised model R satisfies the specification (E_t, U_t, A_t) which is equivalent to the initial specification (E, U, A).

Proof. The correctness of the algorithm comes from the fact that each transition maintains only true formulae in the satisfied set, which can be trivially checked for each transition. Moreover, the rules merely exchange formulae between the untreated and satisfied parts. The complete CTL specification is thus preserved in the union of the satisfied and untreated sets. Therefore if the algorithm terminates in a configuration containing no untreated formulas, $[(E_t, U_t, A_t), (\emptyset, \emptyset, \emptyset), R]$, its rule set R satisfies the specification (E_t, U_t, A_t) that is equivalent to the original specification.

The termination of the algorithm is proved by considering the lexicographic ordering over the couple $< a, n >$ where a is the number of untreated ACTL formulae, and n is the number of untreated ECTL and UCTL formulae. The transitions **E**, **E'**, **U**, **U'**, and **U"** do not change a but strictly decrease n. The transition **A** strictly decreases a (and does not change n) while the transition **A'** strictly decreases a. Hence, the complexity measure strictly decreases at each transition, and, as n is bounded by the initial number of ECTL and UCTL formulae, there is at most $a * n$ transitions. □

This algorithm thus finitely enumerates model revisions that satisfy a given specification. However, this algorithm is incomplete for two reasons. First, the satisfaction of an ECTL or an UCTL formula is searched by adding only one rule to the model (transition E' and U'), whereas several rules might be needed. If this is the case for all ECTL and UCTL formulae, the algorithm terminates in a failed configuration containing unsatisfied formulae. Second, in the Kripke structure associated to the BIOCHAM model, the accessibility relation is completed into a total relation (which is necessary for the definition of Kripke structures). When a rule is deleted, another rule (loop) may thus be automatically added, thereby invalidating the assumptions of proposition 1.

This algorithm is therefore a heuristic algorithm which may fail in cases where the CTL specification is satisfiable. It is nevertheless usable in practice as illustrated in Sect. 6, and has been actually designed from our practical experience.

The search for model revisions is obviously a very computation intensive process that can be controlled here with the patterns given for the rules allowed to be added or deleted. These patterns limit the branching factor of the search tree explored by the algorithm, while the depth is bounded by the product of the number of ACTL formulae and the number of other formulae in the specification.

5 Learning Parameter Values from Constraint LTL Properties

In the same manner as for CTL, one can consider the LTL formulae in the concentration semantics as a specification of the expected behavior of a numerical model. Using constraint model-checking techniques, this specification can be checked for correctness. When it is not satisfied, a search algorithm can be used to revise the parameter values.

5.1 Constraint LTL Model-Checking Algorithm

Under the hypothesis that the initial state is completely defined, numerical integration methods (such as Runge-Kutta or Rosenbrock methods) provide a discrete simulation trace. This trace constitutes a linear Kripke structure in which LTL formulae can be interpreted. Since constraints refer not only to concentrations, but also to their derivatives, we consider traces of the form

$$(< t_0, x_0, dx_0/dt >, < t_1, x_1, dx_1/dt >, ...)$$

At each time point, t_i, the trace associates the concentration values of the x_i's and the values of their derivatives dx_i/dt. It is worth noting that in variable step size integration methods, the step size $t_{i+1} - t_i$ is not constant and is computed following an estimation of the error made by the discretization.

To verify a temporal property ϕ within a finite time horizon, one can thus use the following trace-based constraint model-checking algorithm:

1. compute a finite simulation trace;
2. label each trace point by the atomic sub-formulae of ϕ that are true at this point;
3. add sub-formulae of the form $F\phi$ (resp. $X\phi$) to the predecessors (resp. immediate predecessor) of a point labeled with ϕ;
4. add sub-formulae of the form $\phi_1\ U\ \phi_2$ to the points preceding a point labeled with ϕ_2 as long as ϕ_1 holds;
5. add sub-formulae of the form $G\phi$ to the last state if it is labeled by ϕ, and to the predecessors of the points labeled by $G\phi$ as long as ϕ holds.

Being limited to finite simulation traces, and since $G\phi = !F(!\phi)$, we choose to label by $G\phi$ the last state if it satisfies ϕ.

The rationale of this algorithm is that the numerical trace contains enough relevant points, and in particular those where the derivative changes abruptly, to correctly evaluate temporal logic formulae. This has been very well verified in practice with various examples of published mathematical models.

5.2 Parameter Search

One can use constraint LTL model-checking to design a *generate and test* algorithm for finding parameter values such that a given LTL specification is satisfied.

A set of parameters, together with intervals of possible values and a precision parameter, are input to an enumeration algorithm. All value combinations are then scanned with a step size corresponding to the given precision, until the specification is satisfied. The syntax of the command is:

```
learn_parameter(parameters, ranges, precision, spec_LTL, time)
```

where the last argument is the time horizon of the simulation.

This search procedure actually replicates and automates part of what the modeler currently does by hand: trying different parameter values, between bounds that are thought reasonable, or computed by other methods such as bifurcation diagrams, in order to obtain behaviors in accordance with the experimental knowledge. BIOCHAM provides a way to explore much faster this parameter space, once the effort for formalizing the expected behavior in LTL is done.

The main novel feature of this method is its capability to express and combine in LTL both qualitative and quantitative constraints on the expected behavior of the model. Section 6.5 shows an example where this algorithm already provides interesting solutions for a two-parameter search.

The computational complexity grows linearly in the number of combinations of parameter values to try, that is in $O(d^n)$ where n is the number of parameter values to find and d the number of values to try for each parameter. The difficulty to use better search algorithms than generate-and-test (such as local search or simulated annealing for instance) comes from the criterion of satisfaction of LTL formulae which is naturally boolean and for which a multi-valued measure of satisfaction can hardly be defined.

5.3 Constraint PLTL Model-Checking Algorithm

The existing probabilistic model-checking tools, like that of PRISM [35], do not handle well highly non-deterministic examples, nor those where variables have a large domain as it is the case in BIOCHAM's population semantics. This led us to actually consider the PLTL fragment of PCTL formulae in which the $P_{\bowtie_p}$ operator can only appear once as head of the formula, and to use a Monte-Carlo method as done in the APMC system [36].

To evaluate the probability of realization of the underlying LTL formula, BIOCHAM samples a certain number of stochastic simulations using standard algorithms like that of Gillespie [24] or of Gibson [37]. The outer probability is then estimated by counting. It is worth noting that this method provides a real estimate of realization of the LTL formula, whereas PCTL expresses the boolean satisfaction of a probability constraint ($\bowtie_p$) over the formula.

In principle, the Monte-Carlo algorithm can thus be used for model-checking and learning along the same lines as in the concentration semantics. However, both the stochastic simulation process and the model-checking process are computationally more expensive than in the concentration semantics by several orders of magnitude. For this reason, our machine learning approach is currently not practical in the stochastic population semantics.

6 Example of the Cell Cycle Control

In this section, we illustrate the use of our machine learning techniques based on temporal logic specifications, through an example of the cell cycle control developed by Qu et al. ([22]).

6.1 The Model of Qu et al. Transcribed in BIOCHAM

The model of Qu et al. describes the transitions between the different phases of the cell cycle. The cell cycle of somatic cells is usually composed of two alternate phases: the DNA synthesis (S phase) and the chromosome segregation and mitosis (M phase). These phases are separated by gap phases (G1 and G2) that ensure that one phase is finished before the other one starts.

The model of Qu presents the dynamics of one transition, either G1/S or G2/M, depending on the type of cyclin used in the model, i.e. `CycE` or `CycB` respectively. Here we choose to deal with the G2/M transition and thus with the B-type cyclin.

In this model, the cyclin is continuously synthesized and degraded, according to the following rules:

```
k1                          for _=>CycB.
k2*[CycB]                   for CycB=>_.
k2u*[APC]*[CycB]            for CycB=[APC]=>_.
```

The cyclin associates with a cyclin-dependent kinase called `CDK` to form a complex which plays a major role in the control of the transition.

```
(k3*[CDK]*[CycB],k4*[CDK-CycB~{p1,p2}]) for CDK+CycB<=>CDK-CycB~{p1,p2}.
```

Qu's model details the network of proteins that regulates the activity of the complex. This activity depends on the phosphorylation state of the complex, the presence of an inhibitor and the availability of the cyclin component. For that reason, Qu considered three positive (1-3) and one negative (4) feedback loops :

1. the active form `CycB-CDK~{p1}` is inactivated by the `Wee1` kinase, itself inactivated by `CycB-CDK~{p1}`;

```
[Wee1]*[CycB-CDK~{p1}]     for CycB-CDK~{p1}=[Wee1]=>CycB-CDK~{p1,p2}.
cw*[CycB-CDK~{p1}]*[Wee1]  for Wee1=[CycB-CDK~{p1}]=>Wee1~{p1}.
```

2. similarly, the Cdc25 phosphatase (`C25~{p1,p2}`) activates `CycB-CDK~{p1}` which in turn activates Cdc25;

```
k5u*[C25~{p1,p2}]*[CycB-CDK~{p1,p2}]
       for CycB-CDK~{p1,p2}=[C25~{p1,p2}]=>CycB-CDK~{p1}.
cz*[CycB-CDK~{p1}]*[C25]
       for C25=[CycB-CDK~{p1}]=>C25~{p1}.
cz*[CycB-CDK~{p1}]*[C25~{p1}]
       for C25~{p1}=[CycB-CDK~{p1}]=>C25~{p1,p2}.
```

3. when present, an inhibitor, `CKI`, binds to the active form of the complex to form an inactive trimer (`CKI-CycB-CDK~{p1}`) that is recognized for degradation when phosphorylated by `CycB-CDK~{p1}` itself;

```
(k14*[CKI]*[CycB-CDK~{p1}],k15*[CKI-CycB-CDK~{p1}])
      for CKI+CycB-CDK~{p1}<=>CKI-CycB-CDK~{p1}.
ci*[CycB-CDK~{p1}]*[CKI-CycB-CDK~{p1}]
      for CKI-CycB-CDK~{p1}=[CycB-CDK~{p1}]=>(CKI-CycB-CDK~{p1})~{p2}.
k16*[(CKI-CycB-CDK~{p1})~{p2}]
      for (CKI-CycB-CDK~{p1})~{p2}=>CDK.
```

4. `CycB-CDK~{p1}` activates `APC` which is responsible for its degradation.

```
  (([CycB-CDK~{p1}]^2)/(a^2+([CycB-CDK~{p1}]^2)))/tho
        for _=[CycB-CDK~{p1}]=>APC.
  k7u*[APC]*[CycB-CDK~{p1}] for CycB-CDK~{p1}=[APC]=>CDK.
```

The Appendix A contains the complete transcription of Qu's model, with the list of kinetic parameter values, and with the definition of an initial state. In the initial state, the cyclin-dependent kinase `CDK`, the `Wee1` kinase and the cyclin inhibitor `CKI` are present with respective concentrations 200, 1 and 1, all the other molecules being absent (concentrations set to 0).

In addition, a BIOCHAM model can contain a formal specification in temporal logic, accounting for the relevant biological properties captured by the model. For instance, the activation of `CycB-CDK~{p1}`, the oscillatory behavior of the active form of the complex, or the compulsory role of `CycB-CDK~{p1}` in `APC` activation are formalized as follows:

```
reachable(CycB-CDK~{p1}).
oscil(CycB-CDK~{p1}).
checkpoint(CycB-CDK~{p1},APC).
```

The command `genCTL(CTLpattern)` can also be used to automatically generate a specification from the reaction rules. This specification contains, for each compound, all the `reachable`, `oscil`, and `checkpoint` properties (or any other properties specified by a pattern) that are true in the model. Appendix C shows the specification generated from the rules in this example.

6.2 Boolean Abstraction

Qu's model was conceived to specifically study numerical aspects of the cell cycle. Reasoning about such models at a boolean level may lead to some problems. For instance, a fast and a slow reaction are treated equally in the boolean semantics whereas their kinetics defines which one is preponderant. A property revealing the difference between these two reactions would then be invalid in the boolean semantics, which can be checked by BIOCHAM. In that case, the model needs to be adapted.

For example, in most organisms, Cdc25 (more precisely here `C25~{p1,p2}`) is necessary for the activation of the complex `CycB-CDK~{p1}`. This property can be checked in the numerical model by blocking `C25~{p1,p2}` (setting to 0 the kinetic parameters of the rules producing this compound). In the numerical model, the activation of `CycB-CDK~{p1}` is done by a (fast) reaction involving the phosphatase `C25~{p1,p2}` and by a background reaction (slower by one order of magnitude). In the boolean semantics however, the property `checkpoint(C25~{p1,p2},CycB-CDK~{p1})` is false since the second pathway does not use `C25~{p1,p2}`.

That property can be enforced by adding it as a specification and by revising the model. The command `revise_model` produces the following output[3]:

[3] The CPU times indicated in this paper have been obtained on a Pentium IV, 1,7GHz, 1GB RAM, under Linux.

```
biocham:add_spec(Ai(AG(!(CycB-CDK~{p1})
                          ->checkpoint(C25~{p1,p2},CycB-CDK~{p1})))).
biocham: revise_model.
Success
Time: 441.00 s
40 properties treated
Modifications found:
  Deletion(s):
k5*[CycB-CDK~{p1,p2}] for CycB-CDK~{p1,p2}=>CycB-CDK~{p1}.
k15*[CKI-CycB-CDK~{p1}] for CKI-CycB-CDK~{p1}=>CKI+CycB-CDK~{p1}.
  Addition(s):
```

The solution found consists in deleting two rules: the first one corresponding to the background activation of the complex `CycB-CDK~{p1}` and the second one corresponding to another pathway using the dissociation of the inactive trimer. With this modification, the revised model satisfies the complete CTL specification.

6.3 Rule Inference

The automatic search for rules can also be used to complete an erroneous model. To illustrate this, let us delete the rule of `CDK-CycB~{p1}` activation by `Cdc25`: `CDK-CycB~{p1,p2}=[C25~{p1,p2}]=>CDK-CycB~{p1}.` and let the system recover the rule from the specification. After deletion, the model does not verify the CTL specification anymore. When asking to revise the model, the system recovers the deleted rule:

```
biocham: revise_model.
Success
Time: 76.00 s
40 properties treated
Modifications found:
  Deletion(s):
  Addition(s):
CycB-CDK~{p1,p2}=[C25~{p1,p2}]=>CycB-CDK~{p1}.
```

When asking more precisely to learn one rule (as described in Sect. 4.2), the system tests 464 rules, in 25 seconds, and returns only one possible answer in this example. To reduce the time of search (and the number of outputs if several are found), one can also make the rule pattern more precise, like `learn_one_addition(dephosphorylation)`, where the dephosphorylation pattern is `$A~? =[?]=> $A`. In that case, fewer rules (80) are tested, and the missing rule is found in less than 5 seconds.

6.4 Model Refinement

The automated method described for learning one rule and revising models can also be used in an interactive fashion to refine a model, for instance by adding a molecule to the model.

Let us include the protein CycE in Qu's generic cell cycle in order to reproduce the G1/S transition. New properties are formulated and added to the specification. As it is the case for CycB, we assume that CycE is only active in a complex with another cyclin-dependent kinase, called for our purpose CDKp, and the complex can be inactivated by an inhibitor CKI:

```
Ai(reachable(CycE-CDKp)),          Ai(oscil(CycE-CDKp)),
Ai(reachable(CKI-CycE-CDKp)),      Ai(oscil((CycE-CDKp)~{p1})),
Ai(reachable((CycE-CDKp)~{p1})), Ai(loop(CycE-CDKp,(CycE-CDKp)~{p1})),
```

Rules for synthesis and degradation of CycE and CDKp are added. The specification is no longer verified and with revise_model, BIOCHAM is asked to propose some rules to correct the model.

```
biocham: revise_model.
Success
Time: 158.00 s
6 properties treated
Modifications found:
  Deletion(s):
  Addition(s):
CDKp+CycE=>CDKp-CycE.
CKI+CDKp-CycE=>CDKp-CKI-CycE.
CDKp-CycE=>(CDKp-CycE)~{p1}.
(CDKp-CycE)~{p1}=>CDKp-CycE.
```

The rules found are quite simple and constitute a first step for refining the model. This first solution found is followed by roughly 30^4 other solutions corresponding to 30 different ways to achieve the same results for each rule. For instance the inhibition of (CDKp-CycE)~{p1} performed by dephosphorylation in the last rule, can be performed as well by other complexation, or degradation rules. One way to restrict the combinatorics of these revisions is thus to refine the rule patterns given for the search. Another way is to augment the specification if more knowledge about the proteins and their activity is available.

6.5 Parameter Search

Now to illustrate the parameter search method, let us consider the estimation of the two parameters $k1$ and $k5u$. These parameters, studied in lengthy details in Qu's article, correspond respectively to the parameter for CycB synthesis, and to the rate of the CycB-CDK~{p1} activation by C25~{p1,p2}. When set to 0, the numerical simulation exhibits a stable steady state, as shown in Fig. 1.

In order to find satisfactory values, the desired oscillation property may be expressed as follows: the concentration of the complex CycB-CDK~{p1} oscillates at least twice in the time horizon of 300 time units, with peaks above the threshold value 10. Based on its LTL formalization, a search is done for the two parameters $k5u$ and $k1$ on a domain of (0,10) and (0,500) respectively, among 20 different possibilities for each parameter:

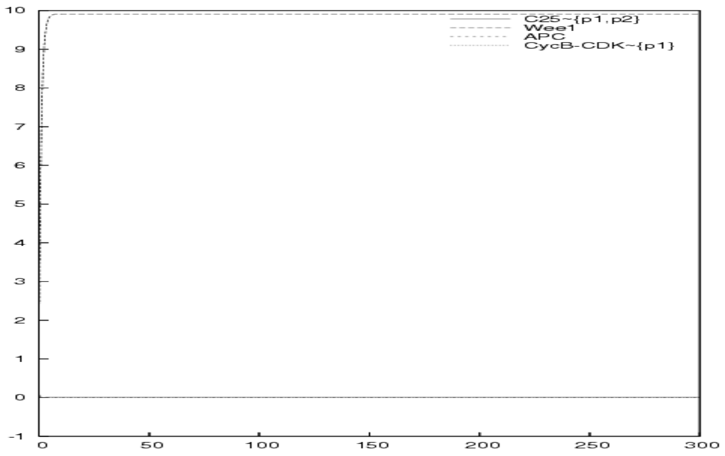


Fig. 1. Behavior obtained with parameter values: $k5u$=0 and $k1$=0

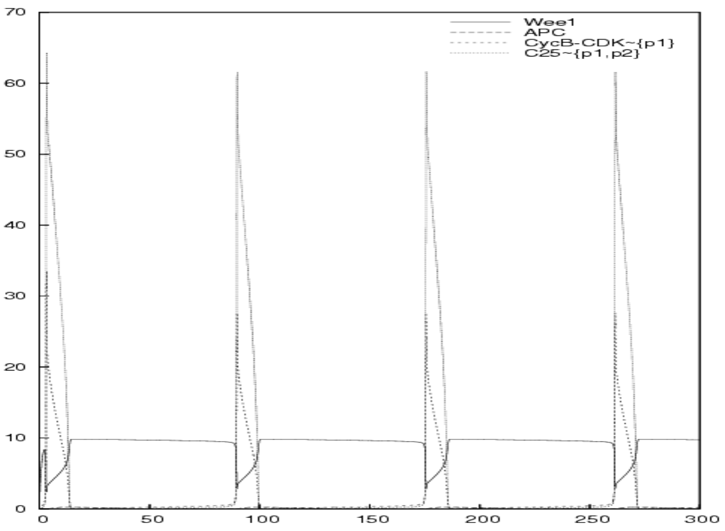


Fig. 2. Behavior obtained with parameter values $k5u$=0.5, $k1$=350 found in response to the query `oscil(CycB-CDK~{p1},2,10.0)` over a time horizon of 300 t.u.

```
biocham: learn_parameters([k5u,k1],[(0,10),(0,500)], 20,
                          oscil(CycB-CDK~{p1},2,10.0),300).
First values found that make oscil(CycB-CDK~{p1},2,10.0) true:
parameter(k5u,0.5).
parameter(k1,350).
Search time: 21.54 s
```

BIOCHAM proposes the first solution that satisfies the specification, here $k5u$=0.5 and $k1$=350. The simulation is depicted in Fig. 2.

The specification can be further refined to enforce a period of approximately 65 time units:

```
biocham: learn_parameters([k5u,k1],[(0,10),(0,500)], 20,
                          period(CycB-CDK~{p1},65), 300).
First values found that make period(CycB-CDK~{p1},65) true:
parameter(k5u,2).
parameter(k1,150).
Search time: 236.37 s
```

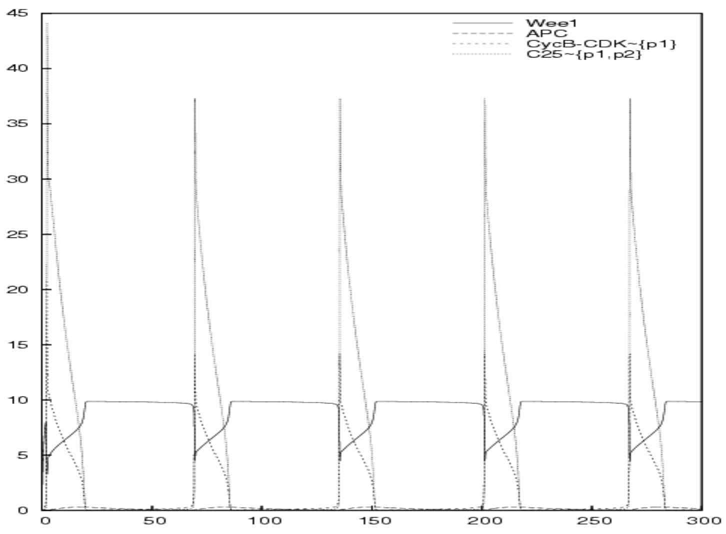


Fig. 3. Behavior obtained with parameter values $k5u$=2, $k1$=150 found in response to the query `period(CycB-CDK~{p1},65)` over a time horizon of 300 t.u.

These values produce Fig. 3, and are actually close to the values published by Qu et al ($k5u$=1 and $k1$=300 with an exact period of 67.65 time units). It is worth noting that they have been produced in response to a request containing both qualitative (oscillatory behavior) and quantitative (period and threshold values) aspects, without overspecifying the expected curve of concentration values.

7 Conclusion

Temporal logic is a powerful formalism for expressing the biological properties of a living system, such as state reachability, checkpoints, stability, oscillations, etc. This can be done both qualitatively and quantitatively, by considering first-order temporal logic formulae with numerical constraints.

In this paper we have shown how temporal logic can be turned into a specification language of the behavior of a biochemical system, and how model-checking and machine learning techniques can be jointly used to infer reaction rules, or more generally model revisions, in order to satisfy a temporal logic specification. More specifically we have described a model revision algorithm which, given a boolean model and a CTL specification, enumerates a set of solutions in the form of sets of rules to add to, or delete from, the model in order to satisfy the specification. The depth of the search tree explored by this algorithm is bounded by $a * n$ where a is the number of ACTL formulae and n is the number of other CTL formulae in the specification. The source of incompleteness of this algorithm has been analyzed, leaving place for further improvements.

Similarly, we have presented an algorithm which, given a numerical model, a value range for some parameters and an LTL specification with numerical constraints over concentrations, searches for parameter values satisfying the specification.

These algorithms have been ilslustrated with different scenarios of specification, rule inference, model refinement and parameter learning in an example of the cell cycle control after Qu et al. [22].

These first results implemented in BIOCHAM are quite encouraging and motivate us to go further in the direction of the formal specification of biological systems and in the improvement of the search algorithms. They also show some limitations concerning the boolean abstraction level used to reason about biochemical systems. There are many ways to define a boolean model from a numerical model. The boolean abstraction currently used in BIOCHAM is a strong abstraction as it simply forgets the kinetic expressions, and considers all possible transitions in an equal setting. Less crude abstractions are however possible and would be technically more effective. We are currently investigating these abstractions, as well as their formal relationship, in the framework of abstract interpretation.

Acknowledgments

This work has been partly supported by the European STREP project APRIL II[4]. We are especially grateful to Stephen Muggleton, Luc de Raedt, David Gilbert, Vlad Vyshemirsky and Monika Heiner for fruitful discussions.

References

1. Regev, A., Silverman, W., Shapiro, E.Y.: Representation and simulation of biochemical processes using the pi-calculus process algebra. In: Proceedings of the sixth Pacific Symposium of Biocomputing. (2001) 459–470
2. Cardelli, L.: Brane calculi - interactions of biological membranes. In Danos, V., Schächter, V., eds.: CMSB'04: Proceedings of the second Workshop on Computational Methods in Systems Biology. Volume 3082 of Lecture Notes in BioInformatics., Springer-Verlag (2004) 257–280
3. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.: Bioambients: An abstraction for biological compartments. Theoretical Computer Science **325** (2004) 141–167

[4] `http://www.aprill.org`

4. Danos, V., Laneve, C.: Formal molecular biology. Theoretical Computer Science **325** (2004) 69–110
5. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. Transactions on Computational Systems Biology (to appear) Special issue of Bio-Concur 2004.
6. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., Sönmez, M.K.: Pathway logic: Symbolic analysis of biological signaling. In: Proceedings of the seventh Pacific Symposium on Biocomputing. (2002) 400–412
7. Chabrier, N., Fages, F.: Symbolic model cheking of biochemical networks. In Priami, C., ed.: CMSB'03: Proceedings of the first Workshop on Computational Methods in Systems Biology. Volume 2602 of Lecture Notes in Computer Science., Rovereto, Italy, Springer-Verlag (2003) 149–162
8. Bernot, G., Comet, J.P., Richard, A., Guespin, J.: A fruitful application of formal methods to biological regulatory networks: Extending thomas' asynchronous logical approach with temporal logic. Journal of Theoretical Biology **229** (2004) 339–347
9. Batt, G., Bergamini, D., de Jong, H., Garavel, H., Mateescu, R.: Model checking genetic regulatory networks using gna and cadp. In: Proceedings of the 11th International SPIN Workshop on Model Checking of Software SPIN'2004, Barcelona, Spain (2004)
10. Calder, M., Vyshemirsky, V., Gilbert, D., Orton, R.: Analysis of signalling pathways using the prism model checker. In Plotkin, G., ed.: CMSB'05: Proceedings of the third Workshop on Computational Methods in Systems Biology. (2005)
11. Antoniotti, M., Policriti, A., Ugel, N., Mishra, B.: Model building and model checking for biochemical processes. Cell Biochemistry and Biophysics **38** (2003) 271–286
12. Calzone, L., Chabrier-Rivier, N., Fages, F., Soliman, S.: A machine learning approach to biochemical reaction rules discovery. In III, F.J.D., ed.: Proceedings of Foundations of Systems Biology and Engineering FOSBE'05, Santa Barbara (2005) 375–379
13. Fages, F., Soliman, S., Chabrier-Rivier, N.: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. Journal of Biological Physics and Chemistry **4** (2004) 64–73
14. Chabrier, N., Fages, F., Soliman, S.: BIOCHAM's user manual. INRIA. (2003–2006)
15. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
16. Nagasaki, M., Onami, S., Miyano, S., Kitano, H.: Bio-calculus: Its concept and molecular interaction. In: Proceedings of the Workshop on Genome Informatics. Volume 10. (1999) 133–143
17. Nagasaki, M., Onami, S., Miyano, S., Kitano, H.: Bio-calculus: Its concept, and an application for molecular interaction. In: Currents in Computational Molecular Biology. Volume 30 of Frontiers Science Series. Universal Academy Press, Inc. (2000) This book is a collection of poster papers presented at the RECOMB 2000 Poster Session.
18. Muggleton, S.H.: Inverse entailment and progol. New Generation Computing **13** (1995) 245–286
19. Bryant, C.H., Muggleton, S.H., Oliver, S.G., Kell, D.B., Reiser, P.G.K., King, R.D.: Combining inductive logic programming, active learning and robotics to discover the function of genes. Electronic Transactions in Artificial Intelligence **6** (2001)
20. Angelopoulos, N., Muggleton, S.H.: Machine learning metabolic pathway descriptions using a probabilistic relational representation. Electronic Transactions in Artificial Intelligence **7** (2002) also in Proceedings of Machine Intelligence 19.

21. Angelopoulos, N., Muggleton, S.H.: Slps for probabilistic pathways: Modeling and parameter estimation. Technical Report TR 2002/12, Department of Computing, Imperial College, London, UK (2002)
22. Qu, Z., MacLellan, W.R., Weiss, J.N.: Dynamics of the cell cycle: checkpoints, sizers, and timers. Biophysics Journal **85** (2003) 3600–3611
23. Chabrier-Rivier, N., Fages, F., Soliman, S.: The biochemical abstract machine BIOCHAM. In Danos, V., Schächter, V., eds.: CMSB'04: Proceedings of the second Workshop on Computational Methods in Systems Biology. Volume 3082 of Lecture Notes in BioInformatics., Springer-Verlag (2004) 172–191
24. Gillespie, D.T.: General method for numerically simulating stochastic time evolution of coupled chemical-reactions. Journal of Computational Physics **22** (1976) 403–434
25. Gibson, M.A., Bruck, J.: A probabilistic model of a prokaryotic gene and its regulation. In Bolouri, H., Bower, J., eds.: Computational Methods in Molecular Biology: From Genotype to Phenotype. MIT press (2000)
26. Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., Schächter, V.: Modeling and querying biochemical interaction networks. Theoretical Computer Science **325** (2004) 25–44
27. Batt, G.: Validation de modèles qualitatifs de réseaux de régulation génique : une méthode basée sur des techniques de vérication formelle. PhD thesis, Université Joseph Fourier - Grenoble I (2006)
28. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6** (1994) 512–535
29. Cimatti, A., Clarke, E., Enrico Giunchiglia, F.G., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: Proceedings of the International Conference on Computer-Aided Verification, CAV'02, Copenhagen, Danmark (2002)
30. Kohn, K.W.: Molecular interaction map of the mammalian cell cycle control and DNA repair systems. Molecular Biology of the Cell **10** (1999) 2703–2734
31. Schoeberl, B., Eichler-Jonsson, C., Gilles, E., Muller, G.: Computational modeling of the dynamics of the map kinase cascade activated by surface and internalized egf receptors. Nature Biotechnology **20** (2002) 370–375
32. Wang, D., Clarke, E.M., Zhu, Y., Kukula, J.: Using cutwidth to improve symbolic simulation and boolean satisfiability. In: IEEE International High Level Design Validation and Test Workshop 2001 (HLDVT 2001). (2001) 6
33. Berman, C.L.: Circuit width, register allocation, and reduced function graphs. Research Report RC 14127, IBM (1988)
34. Murata, T.: Petri nets: properties, analysis and applications. Proceedings of the IEEE **77** (1989) 541–579
35. Kwiatkowska, M.Z., Norman, G., Parker, D.: Prism 2.0: A tool for probabilistic model checking. In: st International Conference on Quantitative Evaluation of Systems (QEST 2004), IEEE Computer Society (2004) 322–323
36. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Proceedings of the 5th Verification, Model Checking and Abstract Interpretation (VMCAI 2004). Volume 2937 of Lecture Notes in Computer Science., Springer-Verlag (2004) 73–84
37. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. Journal of Physical Chemistry **104** (2000) 1876–1889

A BIOCHAM Model of the Cell Cycle Control After [22]

```
% Cell cycle Qu et al. 2003 Biophysical journal

% Cyclin
k1                             for _=>CycB.
k2*[CycB]                      for CycB=>_.
k2u*[APC]*[CycB]               for CycB=[APC]=>_.
(k3*[CDK]*[CycB],k4*[CycB-CDK~{p1,p2}])
                               for CDK+CycB<=>CycB-CDK~{p1,p2}.
k5*[CycB-CDK~{p1,p2}]          for CycB-CDK~{p1,p2}=>CycB-CDK~{p1}.
k5u*[C25~{p1,p2}]*[CycB-CDK~{p1,p2}]
                        for CycB-CDK~{p1,p2}=[C25~{p1,p2}]=>CycB-CDK~{p1}.
k6*[CycB-CDK~{p1}]             for CycB-CDK~{p1}=>CycB-CDK~{p1,p2}.
[Wee1]*[CycB-CDK~{p1}]         for CycB-CDK~{p1}=[Wee1]=>CycB-CDK~{p1,p2}.
k7*[CycB-CDK~{p1}]             for CycB-CDK~{p1}=>CDK.
k7u*[APC]*[CycB-CDK~{p1}]      for CycB-CDK~{p1}=[APC]=>CDK.

% Cdc25
k8                             for _=>C25.
k9*[C25]                       for C25=>_.
k9*[C25~{p1}]                  for C25~{p1}=>_.
k9*[C25~{p1,p2}]               for C25~{p1,p2}=>_.
bz*[C25]                       for C25=>C25~{p1}.
cz*[CycB-CDK~{p1}]*[C25]       for C25=[CycB-CDK~{p1}]=>C25~{p1}.
az*[C25~{p1}]                  for C25~{p1}=>C25.
bz*[C25~{p1}]                  for C25~{p1}=>C25~{p1,p2}.
cz*[CycB-CDK~{p1}]*[C25~{p1}]  for C25~{p1}=[CycB-CDK~{p1}]=>C25~{p1,p2}.
az*[C25~{p1,p2}]               for C25~{p1,p2}=>C25~{p1}.

% Wee1
k10                            for _=>Wee1.
k11*[Wee1]                     for Wee1=>_.
k11*[Wee1~{p1}]                for Wee1~{p1}=>_.
bw*[Wee1]                      for Wee1=>Wee1~{p1}.
cw*[CycB-CDK~{p1}]*[Wee1]      for Wee1=[CycB-CDK~{p1}]=>Wee1~{p1}.
aw*[Wee1~{p1}]                 for Wee1~{p1}=>Wee1.

% APC
(([CycB-CDK~{p1}]^2)/(a^2+([CycB-CDK~{p1}]^2)))/tho
                               for _=[CycB-CDK~{p1}]=>APC.
[APC]/tho                      for APC=>_.

% CKI
k12                            for _=>CKI.
k13*[CKI]                      for CKI=>_.
(k14*[CKI]*[CycB-CDK~{p1}],k15*[CKI-CycB-CDK~{p1}])
                               for CKI+CycB-CDK~{p1}<=>CKI-CycB-CDK~{p1}.
bi*[CKI-CycB-CDK~{p1}]    for CKI-CycB-CDK~{p1}=>(CKI-CycB-CDK~{p1})~{p2}.
ci*[CycB-CDK~{p1}]*[CKI-CycB-CDK~{p1}]
```

```
         for CKI-CycB-CDK~{p1}=[CycB-CDK~{p1}]=>(CKI-CycB-CDK~{p1})~{p2}.
ai*[(CKI-CycB-CDK~{p1})~{p2}]
                          for (CKI-CycB-CDK~{p1}){p2}=>CKI-CycB-CDK~{p1}.
k16*[(CKI-CycB-CDK~{p1})~{p2}]
                              for (CKI-CycB-CDK~{p1})~{p2}=>CDK.
k16u*[APC]*[(CKI-CycB-CDK~{p1})~{p2}]
                              for (CKI-CycB-CDK~{p1})~{p2}=[APC]=>CDK.

parameter(k1,300).  parameter(k5u,1).
parameter(k2,5).    parameter(k3,0.15). parameter(k4,30).
parameter(k5,0.1).  parameter(k6,1).    parameter(k7,10).
parameter(k8,100).  parameter(k9,1).    parameter(k10,10).
parameter(k11,1).   parameter(k12,10).  parameter(k13,1).
parameter(k14,1).   parameter(k15,1).   parameter(k16,2).
parameter(k2u,50).  parameter(k7u,0).   parameter(k16u,25).
parameter(a,4).     parameter(tho,25).  parameter(az,10).
parameter(aw,10).   parameter(ai,10).   parameter(bz,0.1).
parameter(bw,0.1).  parameter(bi,0.1).  parameter(cz,1).
parameter(cw,1).    parameter(ci,1).

% Initial state
present(CDK,200).   present(Wee1,1).   present(CKI,1).
make_absent_not_present.
```

B Graphical View of the Model

The Fig. 4 shows a bipartite graph representation of Qu's model, where the reaction rules are in rectangular boxes, the molecules are in circular boxes, the hard lines materialize the reactants and products of the reactions, and the dashed lines materialize the catalysts.

C CTL Specification

```
% Simple specification of Qu's model generated by genCTL
add_specs({
Ei(reachable(CycB)),
Ei(reachable(!(CycB))),
Ai(oscil(CycB)),

Ei(reachable(APC)),
Ei(reachable(!(APC))),
Ai(oscil(APC)),
Ai(AG(!(APC)->checkpoint(CycB-CDK~{p1},APC))),

Ei(reachable(CDK)),
Ei(reachable(!(CDK))),
Ai(oscil(CDK)),
```

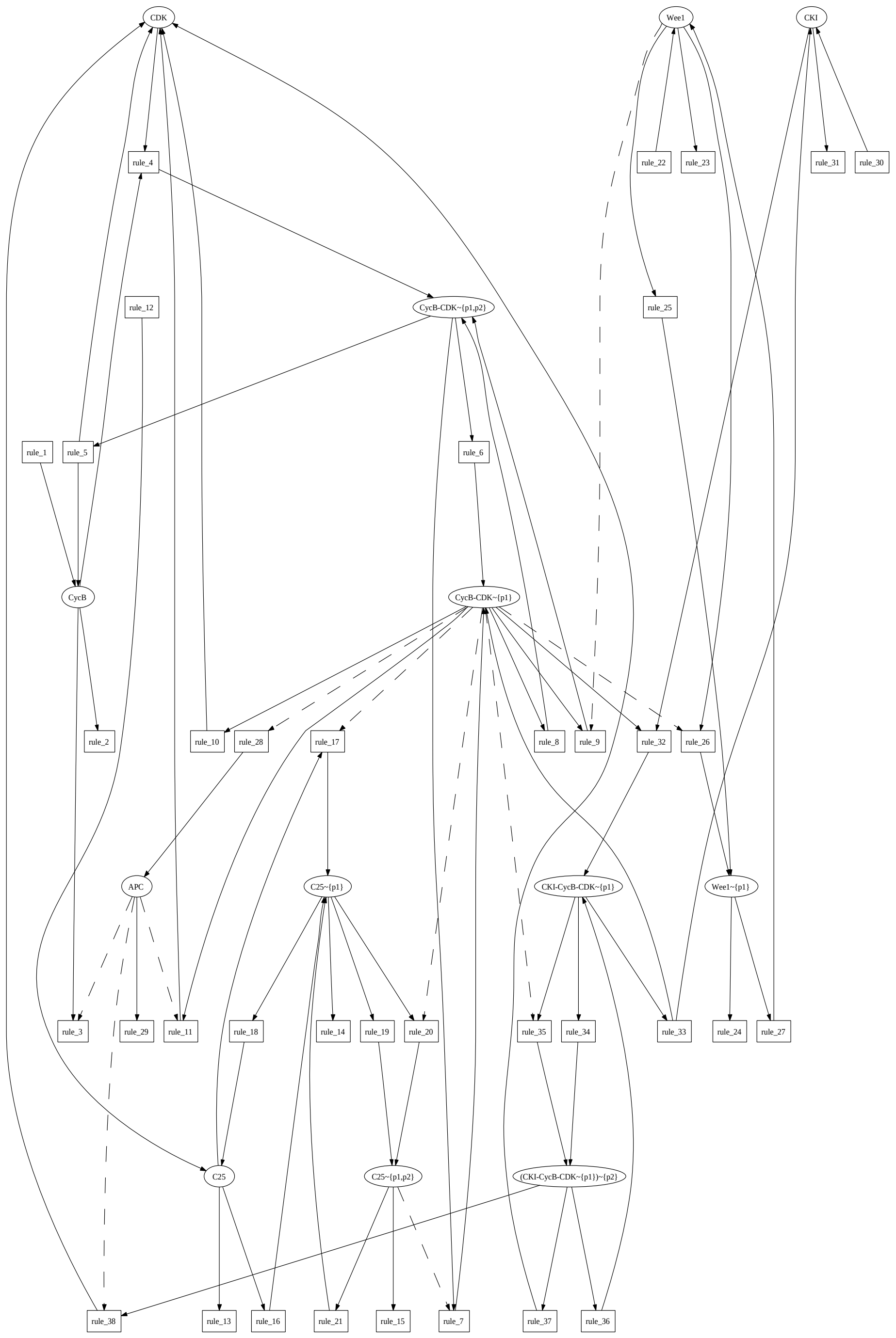


Fig. 4. Graphical view of Qu's model

```
Ei(reachable(CycB-CDK~{p1,p2})),
Ei(reachable(!(CycB-CDK~{p1,p2}))),
Ai(oscil(CycB-CDK~{p1,p2})),
Ei(reachable(CycB-CDK~{p1})),
Ei(reachable(!(CycB-CDK~{p1}))),
Ai(oscil(CycB-CDK~{p1})),

Ei(reachable(C25~{p1,p2})),
Ei(reachable(!(C25~{p1,p2}))),
Ai(oscil(C25~{p1,p2})),
Ai(AG(!(C25~{p1,p2})
    ->checkpoint(C25~{p1},C25~{p1,p2}))),
Ei(reachable(C25)),
Ei(reachable(!(C25))),
Ai(oscil(C25)),
Ei(reachable(C25~{p1})),
Ei(reachable(!(C25~{p1}))),
Ai(oscil(C25~{p1})),

Ei(reachable(Wee1)),
Ei(reachable(!(Wee1))),
Ai(oscil(Wee1)),
Ei(reachable(Wee1~{p1})),
Ei(reachable(!(Wee1~{p1}))),
Ai(oscil(Wee1~{p1})),
Ai(AG(!(Wee1~{p1})->checkpoint(Wee1,Wee1~{p1}))),

Ei(reachable(CKI)),
Ei(reachable(!(CKI))),
Ai(oscil(CKI)),
Ei(reachable(CKI-CycB-CDK~{p1})),
Ei(reachable(!(CKI-CycB-CDK~{p1}))),
Ai(oscil(CKI-CycB-CDK~{p1})),
Ei(reachable((CKI-CycB-CDK~{p1})~{p2})),
Ei(reachable(!((CKI-CycB-CDK~{p1})~{p2}))),
Ai(oscil((CKI-CycB-CDK~{p1})~{p2})),
Ai(AG(!((CKI-CycB-CDK~{p1})~{p2})
    ->checkpoint(CKI-CycB-CDK~{p1},CKI-CycB-CDK~{p1})~{p2}))}).
```

Qualitative Petri Net Modelling of Genetic Networks

Claudine Chaouiya[1], Elisabeth Remy[2], and Denis Thieffry[1]

[1] Institut de biologie du Développement de Marseille Luminy
UMR 6216, Case 907 - Luminy, 13288 Marseille Cedex 9 - France
{chaouiya, thieffry}@ibdm.univ-mrs.fr
[2] Institut de Mathématiques de Luminy,
UMR 6206, case 907 - Luminy, 13288 Marseille Cedex 9 - France
remy@iml.univ-mrs.fr

Abstract. The complexity of biological regulatory networks calls for the development of proper mathematical methods to model their structures and to give insight in their dynamical behaviours. One qualitative approach consists in modelling regulatory networks in terms of logical equations (using either Boolean or multi-level discretisation). Petri Nets (PNs) offer a complementary framework to analyse large systems.

In this paper, we propose to articulate the logical approach with PNs. We first revisit the definition of a rigourous and systematic mapping of multi-level logical regulatory models into specific standard PNs, called *Multi-level Regulatory Petri Nets* (MRPNs). In particular, we consider the case of multiple arcs representing different regulatory effects from the same source. We further propose a mapping of multi-level logical regulatory models into Coloured PNs, called *Coloured Regulatory Petri Nets* (CRPNs). These CRPNs provide an intuitive graphical representation of regulatory networks, relatively easy to grasp.

Finally, we present the PN translation and the analysis of a multi-level logical model of the core regulatory network controlling the differentiation of T-helper lymphocytes into Th1 and Th2 types.

1 Introduction

Most biological processes are spatially and temporally regulated by networks of interactions between regulatory products and genes. To cope with the complexity and characterise the dynamical properties of these *genetic (regulatory) networks*, various formal approaches have been proposed (for a review, see [6]). The lack of precise, quantitative information about the shape of regulatory functions or about the values of involved parameters pleads for the development of qualitative approaches. One qualitative approach consists in modelling regulatory networks in terms of logical equations (using a Boolean or multi-level discretisation, [8,21]). The development of logical models for various biological networks has already induced interesting relationships between topological network features and specific dynamical properties (*e.g.* the crucial roles of regulatory feedback circuits [22]).

The Petri net (PN) formalism offers a complementary framework to analyse the dynamical properties of concurrent systems, either from a qualitative or a quantitative point of view (see [17] for an extensive introduction to PNs). Petri nets have been successfully applied to the modelling and the analysis of metabolic and signal transduction networks (*e.g.* [18,13,12]). As emphasised in [23], one can draw extensive relationships between the traditional biochemical modelling and the PN theory. In particular, the stoichiometry matrix of a metabolic network corresponds to the PN incidence matrix. In this context, PN invariants are associated to conservation relations and flux modes (for a review of PN modelling of biochemical networks, see [11]).

Published PN models of genetic networks are mostly quantitative models written for particular systems, detailing the mechanisms related to transcription and translation (as, for example in [15], where the authors employ hybrid PNs). Logical regulatory networks consider a higher level of abstraction where the semantics associated with the interactions between components varies, and regulatory products are not consumed during the regulatory processes. We have previously introduced systematic procedures to obtain standard PN models from logical regulatory graphs (see [3] for the Boolean case, and [4] for the multi-level case with no multiple arcs). In [5], Comet *et al.* also proposed a rewriting of logical regulatory graphs into Coloured PNs comprising just one place and one transition. Their objective was to provide a mean to automatically generate correct sets of logical parameters, given the topology of a regulatory graph and temporal logic formulae expressing observed behaviours of the biological system.

Our proposal of a PN framework (CRPNs and MRPNs) for the modelling of regulatory networks opens the way for the use of standard PN tools, including model checking techniques, to identify interesting dynamical properties, or to confront models with available dynamical data (*e.g.* temporal gene expression profiles).

In this paper, after recalling the logical formalism basis, we reassess the mapping of multi-level logical regulatory models into standard PNs called Multi-level Regulatory Petri Nets (MRPNs). Here, the main novelty consists in taking into account non-monotonous regulatory effects through multiple arcs. Indeed, this situation can arise in regulatory networks as demonstrated in Section 4. Next, we briefly define a Coloured PN representation of logical regulatory graphs. The resulting Coloured Regulatory Petri Nets (CRPNs) constitute an intuitive graphical representation of regulatory networks, relatively easy to grasp by biologists. In Section 4, we illustrate our approach with the PN representation and the analysis of a multi-level logical model of the core regulatory network controlling the differentiation of T-helper lymphocytes into Th1 and Th2 types.

2 Logical Regulatory Graphs

In this section, we revisit the definition of logical regulatory graphs, introducing some additional notations which will be useful for the PN rewriting (see [21,22,2] for further detail on the logical formalism).

A *regulatory graph* is a labelled directed graph where nodes represent genes (or, more generally, regulatory components) and arcs (directed edges) represent interactions between genes. Let $\mathcal{G} = \{g_1, \ldots, g_n\}$ be the set of genes (or nodes of the regulatory graph). For each $g_i \in \mathcal{G}$, we define its *maximum expression level* Max_i ($\mathsf{Max}_i \in \mathbb{N}^*$), and x_i denotes its current expression level ($x_i \in \{0, 1, \ldots, \mathsf{Max}_i\}$).

For each gene g_i, $Reg(i)$ denotes the set of its regulators, *i.e.*: $g_j \in Reg(i)$ iff there exists an interaction from g_j to g_i in the regulatory graph. Note that a regulatory graph may contain self-regulated genes.

Depending on its levels of expression, a gene $g_j \in Reg(i)$ may have distinct regulatory effects on gene g_i. This situation is represented by multiple arcs joining g_j to g_i. Therefore, the specification of an interaction comprises (in addition to its source and target) an interval of integers included in $[1, \mathsf{Max}_j]$ defining the range of the levels of the source for which the interaction is *operating*. Consequently, a couple of integers $(S_i(j), m_i(j))$ is associated to each $g_j \in Reg(i)$, where:

- $S_i(j)$ is the lowest expression level for which g_j has a regulatory effect upon g_i. It verifies: $0 < S_i(j) \leqslant \mathsf{Max}_j$;
- $m_i(j)$ is the multiplicity of the arc from g_j to g_i, *i.e.* the number of interactions from g_j towards g_i. It verifies: $0 < m_i(j) \leqslant \mathsf{Max}_j - S_i(j) + 1$.

Summing-up, if g_j is a regulator of g_i, either it regulates g_i through a unique interaction ($m_i(j) = 1$), or through several interactions ($m_i(j) > 1$). In this case, a multiple arc connects g_j to g_i, composed by $m_i(j)$ simple arcs labelled by integer intervals $[s_k, s'_k]$, $k = 1, \ldots, m_i(j)$, with,

- $s_1 = S_i(j)$, $s'_{m_i(j)} = \mathsf{Max}_j$ and $\cup_{k=1}^{m_i(j)} [s_k, s'_k] = [S_i(j), \mathsf{Max}_j]$,
- for any $k \neq l$, $1 \leqslant k$, $l \leqslant m_i(j)$, $[s_k, s'_k] \cap [s_l, s'_l] = \emptyset$.

For each gene g_i, we define the set $\mathcal{I}(i)$, called *input of* g_i, which contains all the interactions towards g_i in the regulatory graph and their corresponding intervals: $\mathcal{I}(i) = \{(g_j, [s_k, s'_k]),\ g_j \in Reg(i),\ k = 1, \ldots, m_i(j)\}$.

Remark 1. For all $g_i \in \mathcal{G}$ we have:

- all its levels of expression are pertinent: $\min_{g_k \in \mathcal{G}\,/ g_i \in Reg(k)} S_k(i) = 1$,
- the indegree of the node g_i is given by: $\#\mathcal{I}(i) = \sum_{g_j \in Reg(i)} m_i(j)$,
- $Reg(i) = \mathcal{I}(i)$ if and only if all regulators of g_i are the sources of simple arcs towards g_i: $Reg(i) = \mathcal{I}(i) \Leftrightarrow m_i(j) = 1,\ \forall g_j \in Reg(i)$.

For sake of conciseness, considering a set of interactions $X \subseteq \mathcal{I}(i)$, we write "$g_j \in X$" instead of "$\exists\ [s_k, s'_k] \subseteq [1, \mathsf{Max}_j]$ such that $(g_j, [s_k, s'_k]) \in \mathcal{I}(i)$".

Admissible sets $X \subseteq \mathcal{I}(i)$ as defined below correspond to possible combination of interactions operating together upon gene g_i.

Definition 1. *A subset X of $\mathcal{I}(i)$ is said to be* admissible *if it does not contain several interactions from the same regulator: $g_j \in X$ and $g_k \in X \implies g_j \neq g_k$.*

Now, given X an admissible subset of $\mathcal{I}(i)$, it defines a partition of $Reg(i)$:

$$Reg(i) = Reg(i)^X \cup \overline{Reg(i)}^X ,$$

$Reg(i)^X$ being the set of regulators of g_i which are source of one interaction in X, and $\overline{Reg(i)}^X$ the set of the other regulators: $\overline{Reg(i)}^X = \{g_j \in Reg(i),\ g_j \notin X\}$.

When the expression levels of all the genes are given, one can determine the operating interactions of the network and, for each gene, its relevant admissible set of operating interactions. Then, the effects of these sets of interactions are represented by *logical parameters* defined as follows.

Definition 2. *For $g_i \in \mathcal{G}$, the application K_i associates a value $K_i(X)$ in $[0, \mathsf{Max}_i]$ to each admissible subset X of $\mathcal{I}(i)$. The value $K_i(X)$, called logical parameter, defines the level towards which g_i tends when X is the set of operating incoming interactions. We denote $\mathcal{K}$ the set of all the applications K_i.*

Thus, for each gene g_i, the $K_i(X)$'s constitute parameters of the model as they define the qualitative effects of all possible combinations of incoming interactions. They are said *logical* because $X \subseteq \mathcal{I}(i)$ can be described by a conjunction of conditions on the levels of expression of the regulators of g_i.

Summarising, a logical regulatory graph is defined by three components:

- a set of nodes $\mathcal{G} = \{g_1, \ldots, g_n\}$ with the maximum level $\mathsf{Max}_i \in \mathbb{N}$ of each g_i;
- a set of labelled arcs $\mathcal{I}$ defined by the union of the sets of interactions targeting the genes g_i of $\mathcal{G}$: $\bigcup_{i=1,\ldots,n} \mathcal{I}(i)$;
- a set of parameters $\mathcal{K} = \{K_i(X), i = 1, \ldots, n, X \subseteq \mathcal{I}(i), X \text{ admissible } \}$.

Note that the biologists commonly consider two types of interactions: *activation* (respectively *repression*, or inhibition) has a positive (resp. negative) effect on the targeted gene, *i.e.* induces an increase (resp. a decrease) of its level of expression. However, an effective activatory or inhibitory regulation generally depends on the level of cofactors. Indeed, as one gene may be regulated by several genes, the regulatory effect of one of them may depend on the state of the others. Therefore, in the sequel, we will not explicitly consider the *sign* of an interaction which could be derived from the values of the logical parameters.

A state $\mathbf{x}$ of a regulatory graph $(\mathcal{G}, \mathcal{I}, \mathcal{K})$ is a n-tuple $(x_1, \ldots, x_n)$ of the expression levels of the n genes of $\mathcal{G}$: $\mathbf{x} \in \prod_{i=1}^{n} [0,\ \mathsf{Max}_i]$.

The (discrete) dynamics of the system can be represented by *state transition graphs*, where nodes represent *states*, and arcs represent *transitions* between states. In the sequel, we consider only asynchronous elementary transitions (for each transition, only one variable value is changed by an unitary increase or decrease). Thus, a state differs from its predecessor by exactly one component.

For each gene $g_i \in \mathcal{G}$, the application $\mathcal{T}_i$ associates to each state $\mathbf{x}$ of the system the logical parameter $K_i(X)$ to be considered in state $\mathbf{x}$. The set X is determined by the set of interactions which are operating upon g_i in state $\mathbf{x}$:

$$\mathcal{T}_i(\mathbf{x}) = \sum_{X \subset \mathcal{I}(i)} K_i(X) \prod_{(g_j, [s_k, s'_k]) \in X} \mathbf{1}_{[s_k, s'_k]}(x_j) \prod_{g_j \in \overline{Reg(i)}^X} \left(1 - \mathbf{1}_{[S_i(j), \mathsf{Max}_j]}(x_j)\right), \tag{1}$$

where $\mathbf{1}_A$ is the indicator function of A: $\mathbf{1}_A(x) = 1$ if $x \in A$, 0 otherwise.

Now, the applications $\mathcal{T}_i$ $(i = 1, \ldots, n)$ allow us to formally define the dynamics of the system. For any state $\mathbf{x} = (x_1, \ldots, x_n)$, if $x_i \neq \mathcal{T}_i(\mathbf{x})$ $(i = 1, \ldots, n)$, there is one transition (arc) in the state transition graph from state $\mathbf{x}$ to state $\mathbf{y}$ defined by:

$$y_j = x_j \text{ for all } j \neq i, \qquad y_i = x_i + sign(\mathcal{T}_i(\mathbf{x}) - x_i)\,,$$

with, $sign(a) = +1$ if $a > 0$ and $sign(a) = -1$ if $a < 0$, for all $a \in \mathbb{Z}^*$.

For a given initial state, the corresponding state transition graph defines all the possible trajectories of the system from the selected initial conditions. In this graph, terminal strongly connected components correspond to *attractors* of the system, *i.e.* sets of states in which the system dynamics is trapped (*e.g.* cyclic behaviour or stable states). Therefore, it is interesting to determine such structures, as well as, for each attractor, its basin of attraction (*i.e.* the maximal set of states S such that all paths containing a state in S reach the attractor).

3 Multi-level Regulatory Petri Nets (MRPNs)

In this section, we define a systematic rewriting of logical regulatory networks into PNs called Multi-level Regulatory Petri Nets (MRPNs). The previously defined Boolean Regulatory Petri Nets (cf. [3]) are a special case of MRPNs. In [4], we have introduced MRPNs and their application to the genetic switch controlling the lysis-lysogeny decision in the bacteriophage lambda. But this definition of MRPNs did not take into account the possible presence of multiple arcs in the regulatory graph. Here, we reconsider the definition of MRPNs, to handle multiple regulatory effects of one gene upon another.

Consider a regulatory graph where each gene g_i has $\mathsf{Max}_i + 1$ significant levels of expression and a current level $x_i \in \{0, \ldots \mathsf{Max}_i\}$. Recall that $\mathcal{I}(i)$ is the set of all possible incoming interactions towards g_i.

- For all gene $g_i \in \mathcal{G}$, two (complementary) places are defined, denoted g_i and $\widetilde{g_i}$. The sum of their marking must always equal Max_i. More precisely, the marking of place g_i represents the current expression level of the corresponding gene, and then g_i has x_i tokens, while $\widetilde{g_i}$ has $\mathsf{Max}_i - x_i$ tokens.
- For all parameter $K_i(X)$ and all admissible set $X \subseteq \mathcal{I}(i)$ $(i = 1, \ldots, n)$, two transitions are defined, denoted $t^+_{i,X}$ and $t^-_{i,X}$, corresponding to an increase or a decrease of the expression level of g_i. Indeed, only three situations are possible. When the current expression level is greater (*resp.* smaller) than $K_i(X)$, we allow an increase (*resp.* a decrease) of the gene level by one unit at a time. The case where the current level equals $K_i(X)$ is omitted in the PN representation as it implies no change in the gene expression. Transitions $t^+_{i,X}$ and $t^-_{i,X}$ are enabled when the interactions in X are operating and those in $\mathcal{I}(i) \setminus X$ are not.
  Therefore, $\#Reg(i)$ places are connected to $t^+_{i,X}$ and $t^-_{i,X}$ by test arcs (double arcs which test the presence of a given number of tokens):

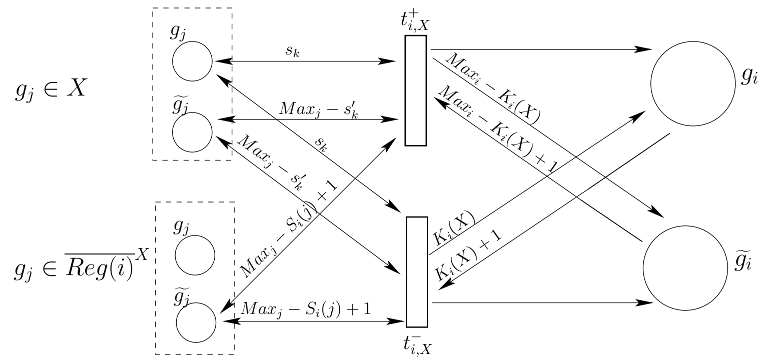


Fig. 1. Weighted arcs and places connected to transitions $t^+_{i,X}$ and $t^-_{i,X}$

- for all $(g_j, [s_k, s'_k]) \in X$, the condition $s_k \leqslant x_j \leqslant s'_k$ must hold (to ensure that the interaction is operating): places g_j and $\widetilde{g_j}$ are connected to both transitions with test arcs weighted s_k and $\mathsf{Max}_j - s'_k$, respectively;
- for all $g_j \in \overline{Reg(i)}^X$, the condition $x_j \leqslant S_i(j) - 1$ must hold (recall that $S_i(j)$ denotes the lowest level for which g_j has an effect upon g_i): place $\widetilde{g_j}$ is connected to both transitions with a test arc weighted $\mathsf{Max}_j - S_i(j) + 1$.

Transitions $t^+_{i,X}$ and $t^-_{i,X}$ are connected to g_i and $\widetilde{g_i}$ in the following way,

- transition $t^-_{i,X}$ is enabled if $x_i \geqslant K_i(X) + 1$ and its firing decreases the level of g_i. Therefore place g_i is connected to $t^-_{i,X}$ by an output arc weighted $K_i(X) + 1$ and an input arc weighted $K_i(X)$ (when enabled, $t^-_{i,X}$ removes one token from g_i). Moreover, $\widetilde{g_i}$ is connected to $t^-_{i,X}$ by an input arc (the firing of $t^-_{i,X}$ adds the token removed from g_i to $\widetilde{g_i}$);
- transition $t^+_{i,X}$ is enabled if $x_i \leqslant K_i(X) - 1$ and its firing increases the level of gene g_i. Therefore $\widetilde{g_i}$ is connected to $t^+_{i,X}$ by an output arc weighted $\mathsf{Max}_i - K_i(X) + 1$, and an input arc weighted $\mathsf{Max}_i - K_i(X)$ (when enabled, $t^+_{i,X}$ removes one token from $\widetilde{g_i}$). Moreover, g_i is connected to $t^+_{i,X}$ by an input arc (the firing of $t^+_{i,X}$ adds one token in g_i).

The Figure 1 illustrates the connections between transitions $t^+_{i,X}$ and $t^-_{i,X}$ and places g_i, $\widetilde{g_i}$, g_j and $\widetilde{g_j}$ (for $g_j \in Reg(i)$).

Remark 2. In the case of a self-regulator, $(g_i, [s_k, s'_k]) \in \mathcal{I}(i)$, we have also: $s_k \leqslant x_i \leqslant s'_k$ if $(g_i, [s_k, s'_k]) \in X$, or $x_i \leqslant S_i(i) - 1$ if $g_i \in \overline{Reg(i)}^X$.

3.1 MRPNs, Definition and Properties

The following definition provides the re-writing rules which define the MRPN corresponding to a regulatory graph.

Definition 3. *Given a multi-level logical regulatory graph, $\mathcal{R} = (\mathcal{G}, \mathcal{I}, \mathcal{K})$ and an initial state $\mathbf{x^0} = (x^0_1, \ldots x^0_n)$, the associated* ***Multi-level Regulatory Petri Net*** *$\mathbf{N}(\mathcal{R}) = (P, T, Pre, Post, \mathcal{M}_0)$ is defined as follows:*

- *$P = \mathcal{G} \cup \widetilde{\mathcal{G}} = \{g_1, \widetilde{g_1}, \ldots, g_n, \widetilde{g_n}\}$ is the set of places,*
- *$T_{\mathcal{K}} = \{t^+_{i,X}, t^-_{i,X},\ i = 1, \ldots n,\ X \subseteq \mathcal{I}(i)$ admissible$\}$ is the set of transitions,*
- *$Pre : P \times T \to \{0, \ldots max\}$ (with $max = max\{\mathsf{Max}_i, i = 1, \ldots, n\}$) is the mapping defining weighted arcs from places to transitions,*
- *$Post : T \times P \to \{0, \ldots max\}$ is the mapping defining weighted arcs from transitions to places,*
- *$\mathcal{M}_0$ is the initial marking defined by: $\mathcal{M}_0(g_i) = x_i^0$ and $\mathcal{M}_0(\widetilde{g_i}) = \mathsf{Max}_i - x_i^0$.*

For all $g_i \in \mathcal{G}$, Pre and Post are defined as follows:
1- *Case $g_i \notin \mathcal{I}(i)$ (g_i is not a self-regulator). For all admissible $X \subseteq \mathcal{I}(i)$, consider the transitions $t^+_{i,X}$ and $t^-_{i,X}$; only the following terms have to be defined (all the other terms being equal to zero):*

$$\left.\begin{array}{lll}\forall (g_j, [s_k, s'_k]) \in X\,, & Pre(g_j, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, g_j) = s_k \\ & Pre(\widetilde{g_j}, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, \widetilde{g_j}) = \mathsf{Max}_j - s'_k \\ \forall g_j \in \overline{Reg(i)}^X & Pre(\widetilde{g_j}, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, \widetilde{g_j}) = \mathsf{Max}_j - S_i(j) + 1\end{array}\right\}_{\alpha \in \{+,-\}} \tag{2}$$

$$\begin{array}{ll} Pre(g_i, t^-_{i,X}) = K_i(X) + 1 & Pre(\widetilde{g_i}, t^+_{i,X}) = \mathsf{Max}_i - K_i(X) + 1 \\ Post(t^-_{i,X}, g_i) = K_i(X) & Post(t^+_{i,X}, \widetilde{g_i}) = \mathsf{Max}_i - K_i(X) \\ Post(t^-_{i,X}, \widetilde{g_i}) = 1 & Post(t^+_{i,X}, g_i) = 1\,. \end{array} \tag{3}$$

2- *Case $g_i \in \mathcal{I}(i)$ (g_i is a self-regulator). For all admissible $X \subseteq \mathcal{I}(i)$, consider the transitions $t^+_{i,X}$ and $t^-_{i,X}$; only the following terms have to be defined (all the other terms being equal to zero):*

- *if $(g_i, [s_k, s'_k]) \in X$, let define $\mu_i = max\{s_k, K_i(X) + 1\}$, $\lambda_i = min\{s'_k, K_i(X) - 1\}$,*

$$\left.\begin{array}{lll}\forall (g_j, [s_k, s'_k]) \in X\,, j \neq i, & Pre(g_j, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, g_j) = s_k \\ & Pre(\widetilde{g_j}, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, \widetilde{g_j}) = \mathsf{Max}_j - s'_k \\ \forall g_j \in \overline{Reg(i)}^X\,, & Pre(\widetilde{g_j}, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, \widetilde{g_j}) = \mathsf{Max}_j - S_i(j) + 1\end{array}\right\}_{\alpha \in \{+,-\}} \tag{4}$$

$$\begin{array}{ll} Pre(g_i, t^-_{i,X}) = \mu_i & Pre(g_i, t^+_{i,X}) = s_k \\ Post(t^-_{i,X}, g_i) = \mu_i - 1 & Pre(\widetilde{g_i}, t^+_{i,X}) = \mathsf{Max}_i - \lambda_i \\ Pre(\widetilde{g_i}, t^-_{i,X}) = \mathsf{Max}_i - s'_k & Post(t^+_{i,X}, \widetilde{g_i}) = \mathsf{Max}_i - \lambda_i - 1 \\ Post(t^-_{i,X}, \widetilde{g_i}) = \mathsf{Max}_i - s'_k + 1 & Post(t^+_{i,X}, g_i) = s_k + 1\,. \end{array} \tag{5}$$

- *if $g_i \in \overline{Reg(i)}^X$, let define $\nu_i = min\{S_i(i), K_i(X)\}$,*

$$\left.\begin{array}{lll}\forall (g_j, [s_k, s'_k]) \in X\,, & Pre(g_j, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, g_j) = s_k \\ & Pre(\widetilde{g_j}, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, \widetilde{g_j}) = \mathsf{Max}_j - s'_k \\ \forall g_j \in \overline{Reg(i)}^X\,, j \neq i, & Pre(\widetilde{g_j}, t^\alpha_{i,X}) = Post(t^\alpha_{i,X}, \widetilde{g_j}) = \mathsf{Max}_j - S_i(j) + 1\end{array}\right\}_{\alpha \in \{+,-\}} \tag{6}$$

$$\begin{array}{ll} Pre(g_i, t^-_{i,X}) = K_i(X) + 1 & Pre(\widetilde{g_i}, t^+_{i,X}) = \mathsf{Max}_i - \nu_i + 1 \\ Pre(\widetilde{g_i}, t^-_{i,X}) = \mathsf{Max}_i - S_i(i) + 1 & Post(t^+_{i,X}, \widetilde{g_i}) = \mathsf{Max}_i - \nu_i \\ Post(t^-_{i,X}, \widetilde{g_i}) = \mathsf{Max}_i - S_i(i) + 2 & Post(t^+_{i,X}, g_i) = 1 \\ Post(t^-_{i,X}, g_i) = K_i(X). & \end{array} \tag{7}$$

In the absence of self-regulation, equations (2) define the test arcs: for all $(g_j, [s_k, s'_k]) \in X$, the number of tokens in place g_j must lay between s_k and s'_k; and for all $g_j \in \overline{Reg(i)}^X$ the number of tokens in g_j must be less than $S_i(j) - 1$ (or the number of tokens in $\widetilde{g_j}$ must be as least $\mathsf{Max}_j - S_j(i) + 1$).

Equations (3, left), state that if g_i contains at least $K_i(X)+1$ tokens, then $t^-_{i,X}$ is enabled and one token is removed from g_i to be added to $\widetilde{g_i}$. Symmetrically, equations (3, right) state that if the number of tokens in g_i is less than $K_i(X)-1$, then $t^+_{i,X}$ is enabled (its firing removes one token from g_i and adds one to $\widetilde{g_i}$).

In the case of a self-regulator, if $(g_i, [s_k, s'_k]) \in X$ equations (5, left) state that if the marking of g_i belongs to $[s_k, s'_k]$ (the self-regulation is operating), and is also greater than $K_i(X) + 1$, then $t^-_{i,X}$ is enabled. Whereas if the marking of g_i belongs to $[s_k, s'_k]$ and is smaller than $K_i(X) - 1$, then $t^+_{i,X}$ is enabled (equations (5, right)). The case where $g_i \in \overline{Reg(i)}^X$ is symmetrical (cf. equations 7).

Recall that markings of complementary places together represent levels of expression. They thus satisfy the condition introduced in the following definition.

Definition 4. *Given a MRPN* $\mathbf{N}(\mathcal{R})$*, a* ***valid marking*** $\mathcal{M}$ *corresponds to a state of the multi-level regulatory graph* $\mathcal{R} = (\mathcal{G}, \mathcal{I}, \mathcal{K})$ *and verifies:*

$$\forall g_i \in \mathcal{G},\ \mathcal{M}(g_i) = \mathsf{Max}_i - \mathcal{M}(\widetilde{g_i})\,. \tag{8}$$

Property 1. Given a MRPN $\mathbf{N}(\mathcal{G}, \mathcal{I}, \mathcal{K})$ and a valid initial marking, any reachable marking is still valid. Therefore, the MRPN is bounded: places g_i and $\widetilde{g_i}$ are Max_i-bounded (for all g_i in $\mathcal{G}$).

The proof is straightforward. Moreover, the marking graph of the MRPN is isomorphic to the state transition graph of the corresponding regulatory graph. This property has been formally stated for the Boolean case in [3].

The MRPN associated to a logical regulatory graph $\mathcal{R} = (\mathcal{G}, \mathcal{I}, \mathcal{K})$ has $2\#\mathcal{G}$ places and $2\sum_{g_i \in \mathcal{G}} 2^{\#\mathcal{I}(i)}$ transitions. In most of the cases, this last number can be significantly reduced applying the rules discussed in the following remarks.

Remark 3. The first reduction consists in avoiding all transitions which are never enabled by construction (their enabling markings are not valid). As illustrated in the Figure 2, for a gene g_i and an admissible set X of incoming interactions, transitions $t^+_{i,X}$ and $t^-_{i,X}$ are enabled for exclusive ranges of g_i values ($[inf_i, sup_i]$ denotes the interval of possible values of the expression levels of g_i). These ranges can reduce to the empty set in some cases. This is easily seen in the absence of self-loops, when parameter values are extremal (*i.e.* 0 or Max_i). Indeed in this case $[inf_i, sup_i] = [0, \mathsf{Max}_i]$ and,

- if $K_i(X) = 0$, we can omit transition $t^+_{i,X}$ as equations (3,right) state that $t^+_{i,X}$ is enabled if $\widetilde{g_i}$ contains $\mathsf{Max}_i - K_i(X) + 1 = \mathsf{Max}_i + 1$ tokens, and this marking is not valid;
- if $K_i(X) = \mathsf{Max}_i$, we can omit transition $t^-_{i,X}$ as equations (3,left) state that $t^-_{i,X}$ is enabled if g_i contains $K_i(X) + 1 = \mathsf{Max}_i + 1$ tokens, and this marking is not valid.

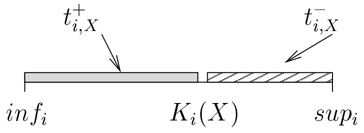


Fig. 2. Transitions $t^+_{i,X}$ and $t^-_{i,X}$ defined for each parameter $K_i(X)$ are exclusive: if $x_i \leqslant K_i(X) - 1$, then $t^+_{i,X}$ is enabled, whereas if $x_i \geqslant K_i(X) + 1$, then $t^-_{i,X}$ is enabled

The case where g_i is a self-regulator is somewhat fastidious to describe, but it follows the same principles. Let consider gene g_i self-regulated. Then,

- if $(g_i, [s_k, s'_k]) \in X$, then $[inf_i, sup_i] = [s_k, s'_k]$ (with $\mu_i = max\{s_k, K_i(X) + 1\}$, $\lambda_i = min\{s'_k, K_i(X) - 1\}$ as in Definition 3), then,
  - if $K_i(X) = 0$ then $\lambda_i = -1$ and we can omit transition $t^+_{i,X}$ as equations (5,right) state that $t^+_{i,X}$ is enabled if $\widetilde{g_i}$ contains $\mathsf{Max}_i - \lambda_i = \mathsf{Max}_i + 1$ tokens, and this marking is not valid;
  - if $K_i(X) = \mathsf{Max}_i$ then $\mu_i = \mathsf{Max}_i + 1$ and we can omit transition $t^-_{i,X}$ as equations (5,left) state that $t^-_{i,X}$ is enabled if g_i contains $\mu_i = \mathsf{Max}_i + 1$ tokens, and this marking is not valid;
- if $g_i \in \overline{Reg(i)}^X$, then the set of possible values of g_i is $[0, \mathsf{Max}_i] \setminus [s_k, s'_k]$, so that the interval in Figure 2 should be divided in two parts. Considering $\nu_i = min\{S_i(i), K_i(X)\}$ as in Definition 3,
  - if $K_i(X) = 0$ then $\nu_i = 0$ and we can omit transition $t^+_{i,X}$ as equations (7,right) state that $t^+_{i,X}$ is enabled if $\widetilde{g_i}$ contains $\mathsf{Max}_i - \nu_i + 1 = \mathsf{Max}_i + 1$ tokens, and this marking is not valid;
  - if $K_i(X) = \mathsf{Max}_i$, we can omit transition $t^-_{i,X}$ as equations (7,left) state that $t^-_{i,X}$ is enabled if g_i contains $K_i(X) + 1 = \mathsf{Max}_i + 1$ tokens, and this marking is not valid.

Moreover, if g_i is a self-regulator, both $t^+_{i,X}$ and $t^-_{i,X}$ can be omitted in the two following cases:

- if $(g_i, [s_k, s'_k]) \in X$ and $K_i(X) = s_k = s'_k = \mathsf{Max}_i$ (in this case $inf_i = \mathsf{Max}_i$, the range of values enabling both transitions is empty),
- if $g_i \in \overline{Reg(i)}^X$ and $K_i(X) = S_i(i) - 1 = 0$.

The second kind of reduction which can be performed is presented next.

Remark 4. All admissible set $X \subseteq \mathcal{I}(g_i)$ defines a logical formula which is a conjunction of literals of the form $[x_j \geqslant s_k]$ and $not[x_j \geqslant s'_k]$ for all $(g_j, [s_k, s'_k]) \in X$ and $not[x_j \geqslant S_j(i)]$ for all $g_j \in \overline{Reg(i)}^X$ $(i = 1, \ldots, n)$. In general two transitions are associated to such formula (excepting the reductions derived from the previous remark).

Now, consider all the logical parameters having the same value $x \in [0, \mathsf{Max}_i]$. They define a disjunction of conditions (the corresponding admissible sets $X \subseteq \mathcal{I}(g_i)$ such that $K_i(X) = x$) under which g_i should tend to level x. This disjunctive formula can often be simplified. In a work in progress, we consider extensions of Binary Decision Diagrams (introduced in [1]) to represent, for a given gene and a given value, the set of logical parameters taking this value. This representation can lead to a simplified disjunctive formula expressing the condition under which g_i should tend to x. This reduction implies a lower number of transitions in the MRPN. Indeed, it is easy to verify that the number of transitions is at most twice the number of terms in the reduced disjunctive formula.

Remarks 3 and 4 can lead to a significant reduction of the MRPN corresponding to a logical regulatory graph. An illustration is provided in the Figure 6 in Section 4. Observe that, in any case, the number of transitions is related to the (reasonably low) indegrees of the genes in the logical regulatory graph.

3.2 Coloured Regulatory Petri Nets (CRPNs)

One drawback of the MRPNs is that they are not easily readable. This point has motivated the use of Coloured PNs for the modelling of logical regulatory networks (for an introduction to Coloured PNs, see [14]). In addition to readability, Coloured PNs are well suited for model checking techniques.

In the following definition, we specify the rewriting of a logical regulatory graph $\mathcal{R}$ into a *Coloured Regulatory Petri Net* (CRPN).

Definition 5. *Given a regulatory graph $\mathcal{R} = (\mathcal{G}, \mathcal{I}, \mathcal{K})$ and an initial state $\mathbf{x}^0$, we define the Coloured Regulatory Petri Net $\mathbf{C}(\mathcal{R}) = (\Sigma, P, T, A, \mathcal{C}, G, E, \mathbf{x}^0)$ as follows:*

- *Σ the finite set of colour sets: $\Sigma = \{[0, \mathsf{Max}_i], i = 1, \ldots, n\}$.*
- *$P = \{g_1, \ldots, g_n\}$ the set of places.*
- *$\mathcal{C}$ the color function associates to each place its expression domain (or colour set): $\mathcal{C} : P \to \Sigma$, $\mathcal{C}(g_i) = [0, \mathsf{Max}_i]$.*
- *$T = \{T_1, \ldots, T_n\}$ is the set of transitions.*
- *$A \subseteq (P \times T \cup T \times P)$ is the set of arcs linking places and transitions; $\forall T_i \in T$,*

$$\forall g_j \in Reg(i), g_j \neq g_i, \ (g_j, T_i) \in A \text{ and } (T_i, g_j) \in A, \tag{9}$$

$$(g_i, T_i) \in A, (T_i, g_i) \in A. \tag{10}$$

Let denote ${}^\bullet T_i = Reg(i) \cup \{g_i\}$ the set of input places of T_i.

- *E the arc expression function defined as follows: $\forall T_i \in T$,*

$$\forall g_j \in {}^{\bullet}T_i \setminus \{g_i\}, \quad E(g_j, T_i) = E(T_i, g_j) = x_j, \quad x_j \in \mathcal{C}(g_j), \tag{11}$$

$$E(g_i, T_i) = x_i, \qquad x_i \in \mathcal{C}(g_i), \tag{12}$$

$$E(T_i, g_i) = x_i + sign(\mathcal{T}_i(\mathbf{x}) - x_i)\,, \qquad \mathbf{x} \in \prod_{g_k \in P} \mathcal{C}(g_k)\,. \tag{13}$$

- *$G = \{G_1, \dots, G_n\}$ is the set of guards; to each transition T_i is associated a guard G_i, a Boolean function defined as follows:*

$$\forall \mathbf{x} \in \prod_{g_j \in P} \mathcal{C}(g_j), \qquad G_i(\mathbf{x}) = [\ \mathcal{T}_i(\mathbf{x}) \neq x_i\ ]\,. \tag{14}$$

- *The initial marking $\mathbf{x}^0 = (x_1^0, \dots, x_n^0)$ assigns to each place g_i one token $x_i^0 \in \mathcal{C}(g_i)$ with x_i^0 being the value of gene g_i in the initial state $\mathbf{x^0}$.*

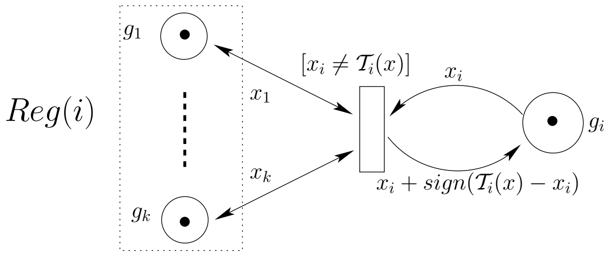

$Reg(i)$
g_1
g_k
x_1
x_k
$[x_i \neq \mathcal{T}_i(x)]$
x_i
g_i
$x_i + sign(\mathcal{T}_i(x) - x_i)$

Fig. 3. The CRPN structure representing the regulation of gene g_i by its regulators

The arcs defined in (9) are test arcs, labelled by the current marking of the places in $Reg(i) \setminus \{g_i\}$ (Equation 11); the input arc (g_i, T_i) defined in (10) has also x_i as a label (Equation 12). The enabling of the transition T_i depends on the current state of its input places ${}^{\bullet}T_i$ and on the Boolean value of its guard G_i. When enabled, the firing of T_i modifies the marking of place g_i according to Equation (1) through the arc (T_i, g_i), g_i expression takes the value $x_i + 1$ or $x_i - 1$, depending on whether $\mathcal{T}_i(\mathbf{x})$ is greater or smaller than x_i (Equation 13).

In (13) and (14), we could consider a projection of the state vector $\mathbf{x}$ on the subset of places $Reg(i) \cup \{g_i\}$, as it is only necessary to consider the markings of places in ${}^{\bullet}T_i$.

Note that when $\mathcal{T}_i(\mathbf{x}) = x_i$, the guard of T_i is false. Hence, the stable states of the regulatory network correspond to dead markings in the CRPN (in contrast with [5] where a stable state is a marking which is its unique successor).

While Coloured PNs are more compact than the corresponding standard PNs, these are amenable to more powerful analyses. Consequently, depending on the questions to be addressed, one may preferably represent logical regulatory graphs using one or the other formalism.

A procedure to recover a MRPN from a CRPN can be easily defined. It first consists in applying the usual method to unfold Coloured PNs. The resulting ordinary net can then be reduced in order to obtain a MRPN complying the Definition 3.

4 Qualitative Dynamical Modelling of Th-lymphocyte Differentiation

4.1 Introducing Th-cell Differentiation

When human are challenged by a microbial infection, various cell types are mobilized to protect vital functions and eliminate the pathogen. Among these different cell populations, the Th lymphocytes play a crucial role in the regulation of the immune response through the integration and excretion of specific molecular signals (lymphokines) (the letters Th refer to the thymus, the organ where Th cells maturate, and to the *helping* function of this cell population). Depending on the challenge and on the activity of other cell lines, the virgin T lymphocytes may differentiate into different subtypes (Th1 and Th2) characterised by specific gene expression and lymphokine excretion patterns. On the basis of an extensive analysis of the literature, L. Mendoza has recently proposed a logical model encompassing the most crucial regulatory components and the cross-interactions involved in these differentiative decisions [16].

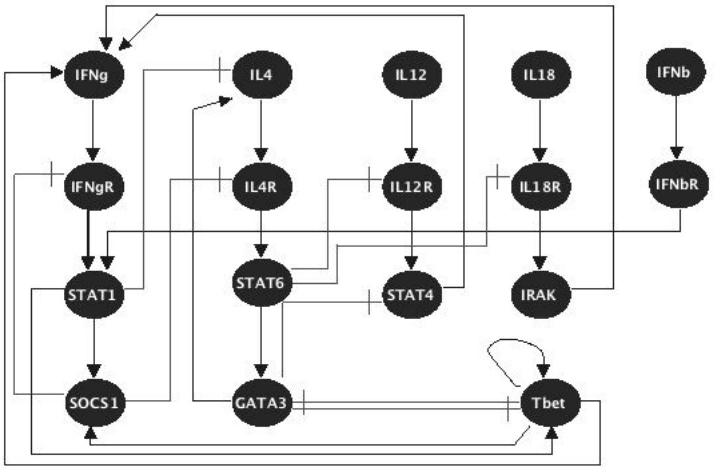


Fig. 4. Logical regulatory graph for the network controlling Th differentiation described in [16]. The nodes represent transcription regulatory factors (T-bet, GATA-3), signaling transduction factors (STAT1, STAT4, STAT6, SOCS1, IRAK), lymphokines (IFN-β, IFN-γ, IL-4, IL-12, IL-18), and lymphokine receptors (IFN-βR, IFN-γR, IL-4R, IL-12R, IL-18R). Normal arrows represent activations, blunt arrows inhibitions.

The model of Mendoza is briefly described in the supplementary material provided on the GINsim web site [25], including an XML (GINML) file containing the definition of the logical regulatory graph, interaction intervals and parameter values. This logical model can be analysed using the software suite GINsim, which can be downloaded from the same url. GINsim provides an interface to define logical regulatory graphs and to construct state transition graphs [9]. In

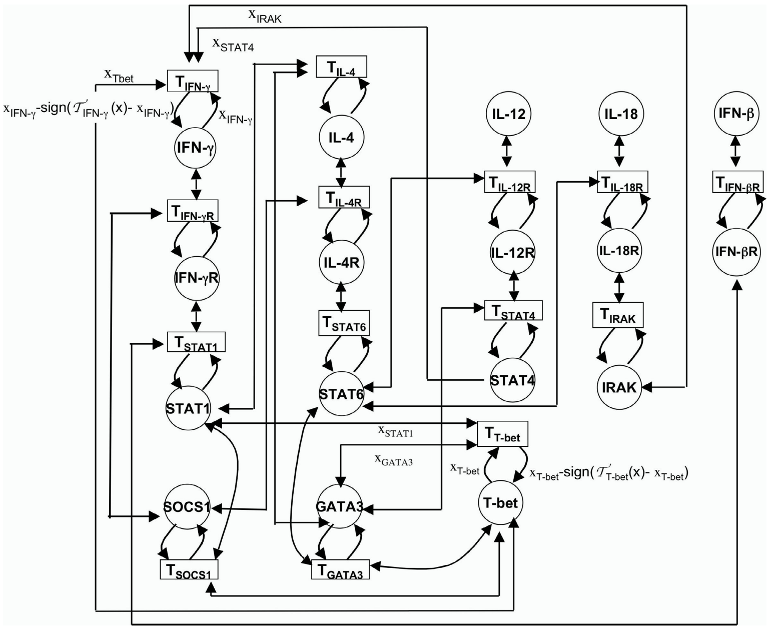
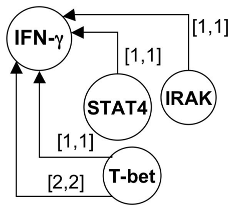


Fig. 5. CRPN translation of the logical model of the network controlling Th differentiation. Note that the topology of the regulatory graph given in Figure 4 can be easily recovered from the CRPN.

particular, GINsim can be used to identify all the stable states of a system (considering all possible initial conditions) and to check whether these states can be reached from specific initial conditions.

Using the Definition 5, we first construct the CRPN associated with the logical model defined by Mendoza (including parameter values and considering an initial state corresponding to virgin Th cells). The resulting CRPN is shown in the Figure 5. Each place of the CRPN of the Figure 5 corresponds to one element of the original logical regulatory graph. The marking of a place represents the level of expression (or protein activity) of the corresponding regulatory element. Each place is fed by one transition, which encodes the logical function of the corresponding regulatory element. A transition is further linked by test arcs to each regulator of the corresponding component.

Alternatively, using the Definition 3, the same logical regulatory graph can be rewritten as a MRPN, where each regulatory node is represented by two complementary places. The logical level of a gene is then represented by the marking of the reference place, while the number of tokens is constant for each pair of complementary places. The transitions then correspond to relevant logical parameters. As mentioned in the previous section, this MRPN can also be obtained by a proper deployment of the CRPN just described. The Figure 6 shows the subnet of the MRPN corresponding to the regulation of IFN-γ and illustrates the application of the reduction rules of Remarks 3 and 4.



0	1	2
$\mathbf{K}_{\text{IFN-}\gamma}(\emptyset)$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{IRAK},1)\}$	$\mathbf{K}_{\text{IFN-}\gamma}\{(\text{STAT-4},1)\}$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{STAT-4},1),(\text{T-bet},1))$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{IRAK},1),(\text{T-bet},1)\}$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{T-bet},1)\}$	$\mathbf{K}_{\text{IFN-}\gamma}\{(\text{STAT-4},1),(\text{IRAK},1)\}$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{STAT-4},1),(\text{IRAK},1),(\text{T-bet},1)\}$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{STAT-4},1),(\text{IRAK},1),(\text{T-bet},2)\}$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{STAT-4},1),(\text{T-bet},2)\}$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{IRAK},1),(\text{T-bet},2)\}$ $\mathbf{K}_{\text{IFN-}\gamma}\{(\text{T-bet},2)\}$
$(\text{STAT-4}=0)\wedge(\text{T-bet}=0)$	$(\text{STAT-4}=1)\wedge(\text{T-bet}<2)\wedge(\text{IRAK}=0)$ $\vee$ $(\text{STAT-4}=0)\wedge(\text{T-bet}=1)$	$(\text{T-bet}=2)$ $\vee$ $(\text{STAT-4}=1)\wedge(\text{IRAK}=1)$
$t_{\text{IFN-}\gamma}^{1-}$	$t_{\text{IFN-}\gamma}^{2+}$, $t_{\text{IFN-}\gamma}^{2-}$, $t_{\text{IFN-}\gamma}^{3+}$, $t_{\text{IFN-}\gamma}^{3-}$	$t_{\text{IFN-}\gamma}^{4+}$, $t_{\text{IFN-}\gamma}^{5+}$

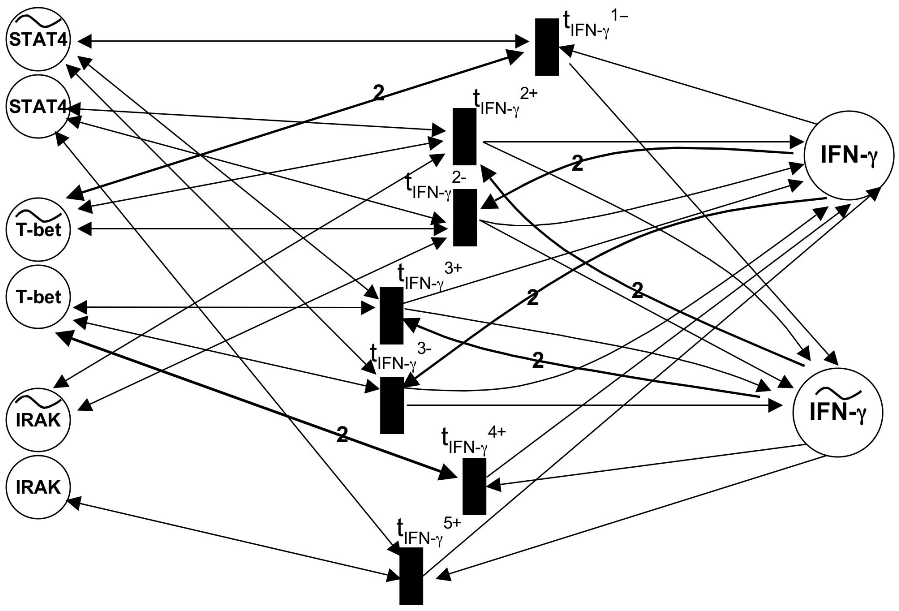

STAT4
STAT4
T-bet
T-bet
IRAK
IRAK
IFN-γ
IFN-γ
$t_{\text{IFN-}\gamma}^{1-}$
$t_{\text{IFN-}\gamma}^{2+}$
$t_{\text{IFN-}\gamma}^{2-}$
$t_{\text{IFN-}\gamma}^{3+}$
$t_{\text{IFN-}\gamma}^{3-}$
$t_{\text{IFN-}\gamma}^{4+}$
$t_{\text{IFN-}\gamma}^{5+}$
2
2
2
2
2
2

Fig. 6. Top: graph of the Figure 4 restricted to the INF-γ regulators. Middle: each column corresponds to a specific parameter value given in the first row $(0, 1, 2)$; row 2, logical parameters for all admissible sets of interactions; row 3, simplified conditions under which IFN-γ tends to the given values (cf. remark 4); row 4, transitions representing these conditions. The reduction rules of remarks 3 and 4 result in lowering the number of transitions from 24 to 7. Bottom: the MRPN representing the regulations of IFN-γ.

At this point, we thus have two formally equivalent PN representations of a single parametrised logical regulatory graph. The CRPN representation has the advantage of graphical simplicity. Furthermore, standard CPN tools can be used to perform simulations or even develop model checking approaches. In contrast, although more difficult to grasp visually, the MRPN representation is amenable to more extensive algebraic analyses. In the sequel, we briefly discuss the analysis of the MRPN of the regulatory network controlling the Th differentiation. This analysis has been performed using INA [24] (the INA and PNML files can be downloaded from [25]).

4.2 Stable States and Their Biological Interpretation

Using a logical simulation tool (e.g. GINsim [9]) or constraint programming [7], one can identify all existing stable states of a logical model (considering all possible initial conditions). In the case of Mendoza's model, four different logical stable states have been readily identified, each corresponding to a specific cellular differentiation state:

– the first stable state has all nodes at the level zero and corresponds to the *naive* (or *virgin*) Th lymphocytes;
– the second stable state encompasses four nodes at level one: the interferon-γ (IFN-γ) and its receptor, the signal transduction factors STAT1 and SOCS1, and the transcription factor T-bet (all the other nodes are at level zero);
– the third stable state is identical to the second one, excepting that IFN-γ and T-bet are at their highest levels (two);
– finally, the fourth stable state encompasses four nodes at level one: the interleukin-4 and its receptor, as well as the signal transduction factor STAT6 and the transcription factor GATA3 (all the other nodes are at level zero).

The last stable state clearly corresponds to the Th2 differentiation state, whereas the second and third stable states correspond to Th1 variants. The co-existence of these two Th1 states accounts for different lymphokine dose effects (IFN-γ) and synergic effects of IL12 and IL18, which favor Th1 polarisation.

This multistability property can be related to the presence of specific positive regulatory circuits (each involving an even number of negative interactions) found in the original regulatory graph. As shown in [16], four circuits are playing a crucial role in this process:

– the INF-γ pathway, including its receptor, STAT1, and T-bet, which in turn regulates INF-γ expression;
– the IL-4 pathway, including its receptor, STA6, and GATA3, which in turn regulates IL-4 expression;
– the positive circuit made of the cross-regulation between T-bet and GATA3;
– and finally the self-regulation of T-bet.

The two first positive circuits ensure a cohesive expression of all the regulatory elements characteristic of the Th1 or Th2 cell populations, respectively. The third circuit ensures the mutual exclusion between these expression patterns. Finally, the last circuit enables the differentiation of Th1 subtypes, characterised by different qualitative levels of T-bet and IFN-γ.

Using INA, it is easy to check the stability of the corresponding markings in our MRPN translation of Mendoza's model. Furthermore, one can check their reachability from specific initial conditions, e.g. for proper combinations of lymphokines (initially and transiently). To illustrate this point, let us consider three situations reported in experimental articles (cf. citations in [16]):

– starting from an initial condition with all nodes at zero but in the presence of a medium level of INF-γ, the system can reach the virgin state (early extinction of IFN-γ signal) or the Th1 state characterized by medium level of IFN-γ and T-bet;
– starting from the same initial condition but in the presence of a high level of INF-γ, the system can reach the virgin state or two Th1 states, depending on the duration of the IFN-γ signal;
– starting from the virgin state plus a combination of IL-12 and IL-18, the system can reach the same three stable states as in the preceding situation;
– finally, starting from the virgin state in the presence of IL-4, the system can reach the virgin state (early extinction of IL-4 signal) or the Th2 state.

5 Conclusions, Discussion and Prospects

We have presented a combined modelling approach encompassing two main steps. First, the model specification is done in terms of a generic regulatory graph, followed by its parameterisation, taking advantage of the flexibility of the definition of the logical parameters. Next, the resulting parameterised regulatory graph is translated into the Petri net formalism. In this respect, we have proposed two formally equivalent rewritings, the first based on standard Petri nets (MRPNs), the second based on Coloured Petri nets (CRPNs). Our approach has been illustrated through the PN translation of a logical model of the core regulatory network controlling the differentiation of T lymphocytes into Th1 and Th2 subtypes. We have shown that the derived CRPN and MRPN models allow to fully recover the salient dynamical properties delineated in the original logical model analysis, in particular the stable states and their reachability from given initial conditions. Note that the current version of GINsim supports the definition of logical models and the construction of the state transition graphs [25]. A forthcoming version of the software, currently in development, will provide a PN export functionality based on the rules defined in this paper.

The combination of the logical approach with the PN framework offers promising prospects for the modelling and analysis of complex regulatory systems. Consequently, logical models become amenable to the numerous tools developed by the PN community, including model checking techniques. MRPNs are somewhat difficult to grasp but readily appropriate for algebraic analyses. In particular, it is possible to identify dead markings (enabling no transition, they correspond to stable states of regulatory networks). The identification of livelocks (*i.e.* cycles of transition firings in which the dynamics is trapped) is also relevant. In marking graphs, livelocks correspond to terminal strongly connected components of more than one node. They represent cyclical attractors of the biological system and can denote periodic behaviours (e.g. the cell cycle) or homeostasis (e.g. the control of temperature in the cell). Attractors and their basins of attraction can be identified applying classical graph theory algorithms on the marking graphs. But such methods may be intractable when facing complex systems. Moreover, it can be useful to derive general properties (independently of initial conditions). Thus, one challenge consists in delineating structural properties of MRPNs (and CRPNs), in order to derive specific theorems on induced dynamical features. Taking inspiration from the logical approach (cf. [22]), we presently focus on the characterization of the dynamical roles of regulatory circuits (cf. [19] for some preliminary results in the Boolean case). This should ease the analysis of large and complex regulatory systems, which remain difficult to explore through systematic simulations. In addition, CRPNs offer an intuitive graphical representation, and can still be readily used to perform simulations or to perform model checking.

At this stage, the resulting marking graphs cover various (and often incompatible) temporal behaviours. In principle, the distinction between alternative pathways can be forced through assumptions on transition delays or on priorities. In this context, Stochastic PNs enable the representation of such assumptions taking into account experimental noise.

In the future, this combined approach will be challenged through its application to more complex regulatory networks, eventually combining genetic and metabolic interactions (cf. [20] for a first step in this direction).

Acknowledgements. We thank A. Naldi for his active participation in the software developments, as well as L. Mendoza and P. Ruet for insightful discussions. We acknowledge financial support from the French Research Ministry through the *ACI IMPbio LumImDynNet project*, the *ANR project* JC05-53969, and the European Commission (contract LSHG-CT-2004-512143).

References

1. Bryant, R.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, **C-35** (1986): 677–691.
2. Chaouiya, C., Remy, E., Mossé, B., Thieffry, D.: Qualitative analysis of regulatory graphs: a computational tool based on a discrete formal framework. LNCIS, **294** (2003): 119–126.
3. Chaouiya, C., Remy, E., Ruet, P., Thieffry, D.: Qualitative Modelling of Genetic Networks: From Logical Regulatory Graphs to Standard Petri Nets. LNCS, **3099** (2004): 137–156.
4. Chaouiya, C., Remy, E., Thieffry, D.: Petri Net Modelling of Biological Regulatory Networks. Proc. of CompBioNets'04, KCL publications, London, 2004.

5. Comet, J.-P., Klaudel, H., Liauzu, S.: Modeling Multi-valued Genetic Regulatory Networks Using High-Level Petri Nets. LNCS **3536** (2005): 208-227.
6. de Jong, H.: Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. J. Comput. Biol. **9** (2002): 67–103.
7. Devloo, V., Hansen, P., Labbe M.: Identification of all steady states in large networks by logical analysis. Bull. Math. Biol. **65** (2003): 1025–1051.
8. Glass, L., Kauffman, S.A.: The logical analysis of continuous, non-linear biochemical control networks. J. theor. Biol. **39** (1973): 103–129.
9. Gonzalez, A.G., Naldi A., Sánchez, L., Thieffry, D., Chaouiya, C.: GINsim: a software suite for the qualitative modelling, simulation and analysis of regulatory networks. Biosystems **84-2** (2006): 91-100.
10. Goss, P.J.E., Peccoud, J.: Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. Proc. Nat. Acad. Sci. USA. **95** (1998): 6750–6755.
11. Hardy, S., Robillard, P. N.: Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches. J. Bioinform. Comput. Biol. **2** (2004): 595–613.
12. Heiner, M., Koch, I.: Petri Net Based Model Validation in Systems Biology. LNCS, **3099** (2004): 216–237.
13. Hofestädt, R., Thelen, S.: Quantitative Modeling of Biochemical Networks. In Silico Biol. **1** (1998): 39–53.
14. Jensen, K.: An introduction to the theoretical aspects of coloured Petri nets. LNCS **803** (1993): 230-272.
15. Matsuno, H., Doi, A., Nagasaki, M., Miyano, S.: Hybrid Petri net representation of gene regulatory networks. Proc. Pac. Symp. Biocomput. (2000): 341–352.
16. Mendoza, L.: A network model for the control of the differentiation process in Th cells. Biosystems **84-2** (2006): 101-14.
17. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proc. IEEE **77** (1989): 541–580.
18. Reddy, V.N., Liebman, M.N., Mavrovouniotis, M.L.: Qualitative analysis of biochemical reaction systems. Comput. Biol. Med. **26** (1996): 9–24.
19. Remy, E., Ruet, P., Mendoza, L., Thieffry, D. and Chaouiya C.: From Logical Regulatory Graphs to Standard Petri Nets: Dynamical Roles and Functionality of Feedback Circuits. Proceedings of BioConcur 2004, London, England, 2004.
20. Simao, E., Remy, E., Thieffry, D. and Chaouiya C.: Qualitative Modelling of Regulated Metabolic Pathways: Application to the Tryptophan Biosynthesis in *E. Coli.* Bioinformatics **21** (2005): ii190-196.
21. Thomas, R.: Regulatory networks seen as asynchronous automata: a logical description, J. theor. Biol. **153** (1991): 1-23.
22. Thomas, R., Thieffry, D., Kaufman, M.: Dynamical behaviour of biological regulatory networks–I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state. Bull. Math. Biol. **57** (1995): 247–276.
23. Zevedei-Oancea, I., Schuster, S.: Topological analysis of metabolic networks based on Petri net theory. In Silico Biol. **3** (2003): 0029.
24. INA: Integrated Net Analyzer URL: `http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina/`.
25. GINsim (Gene Interaction Network simulation) URL: `http://gin.univ-mrs.fr/GINsim`.

Simulating Bacterial Transcription and Translation in a Stochastic π Calculus

Céline Kuttler

Interdisciplinary Research Institute* and LIFL, Lille, France

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

1 Introduction

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

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

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

* FRE 2963 of CNRS.

waiting times between events are drawn can lead to significant variability between different executions of a model.

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

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

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

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

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

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

Degradation of mRna competing with translation.

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

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

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

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

Related work. This paper builds on previous work on stochastic simulation of bacterial gene expression. Heijne and co-authors suggested the the first stochastic treatment of ribosome movement during translation almost 30 years ago [70]. Carrier and Keasling elucidated the relation between molecular actors in translation and mRNA decay in 1997 [11]. In the same year McAdams and Arkin attracted wide attention with a scheme for stochastic simulation of gene expression based on Gillespie's algorithm [38]. It combines a continuous representation of transcription initiation in the tradition of [63] with a stochastic account of transcript elongation and subsequent processing of mRNA. This schema was successfully applied to bacteriophage lambda [3]. However, this latter work neglected important differences in stochastic fluctuations between genes due to distinct translational efficiencies. This aspect was systematically investigated by Kierzek, Zaim, and Zielenkiewitz [32]. They simulated bacterial gene expression while systematically varying translation and transcription initiation frequencies. Their predictions were confirmed experimentally by Ozbudak and co-authors [45]. The coupling between transcription and translation in the control of tryptophan expression [72] was recently investigated by Elf and Ehrenberg [19].

$$\begin{tikzcd}\text{DNA} \arrow[r, "\text{transcription}"'] & m\text{RNA} \arrow[r, "\text{translation}"'] & \text{proteins}\end{tikzcd}$$

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

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

Several dedicated formal languages for biological modeling have been suggested in recent years, each with different objectives [9,12,13,14,51]. They are reviewed in [8,49].

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

2 Bacterial Transcription and Translation

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

2.1 Overview of Genetic Actors and Gene Expression

Each cell of an organism contains its complete genetic information, which is passed on from one generation to the next. It is encoded in a linear, double-stranded DNA macromolecule that winds up to a helix. Each DNA strand contains a sequence over the alphabet of nucleotides $\{A, C, G, T\}$. A *gene* is a segment of one strand of DNA, with explicit begin and end delimiters. Its information content can be transcribed into a single-stranded RNA molecule. DNA also comprises regions that are never transcribed, but contain regulatory information. Figure 1 summarizes the two phase of gene expression.

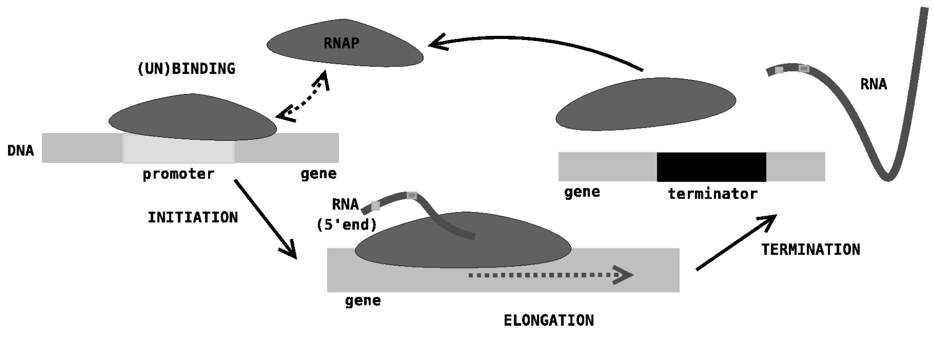


Fig. 2. DNA processing by RNA polymerase (RNAP): promoter binding and initiation, transcript elongation, termination with release of RNA

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

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

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

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

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

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

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

Proteins are the most prominent active constituents of a cell. In brief, proteins carry out instructions that are hard-wired in DNA. They can be enzymes that

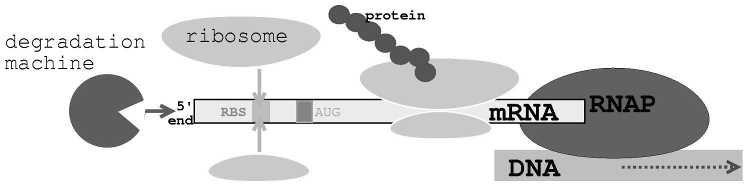


Fig. 3. mRNA is subject to competing translation and transcription. Degradation is initiated at the 5′ end, while the ribosome assembles on the nearby ribosomal binding site RBS. The actual translation initiates at the start codon, here 'AUG'.

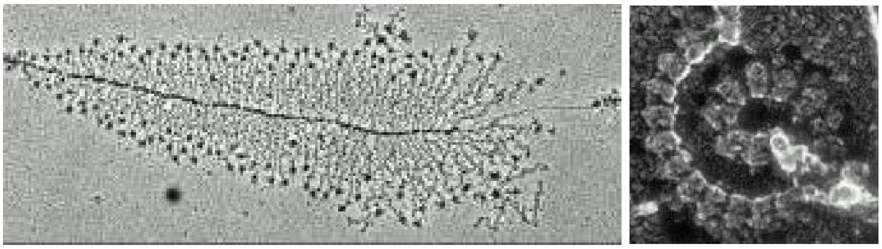


Fig. 4. Simultaneous processing of DNA and mRNA [20]

catalyze reactions, receptors sitting on the cell's outer membrane and conferring information about the environment to the inside, signaling molecules that carry on information within the cell, transcription factors that control gene expression through binding to DNA, or others. All proteins are subject to degradation, their half-lives usually exceed those of mRNA.

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

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

Table 1. Equations for transcription of DNA

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

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

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

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

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

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

Figure 4(b) shows the analogous phenomenon in translation. While the mRNA itself can not be seen, the visible blobs are ribosomes.

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

2.2 DNA and Transcription

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

Equations (2.1) to (2.3) represent the three essential steps of transcription initiation [39], omitting intermediary ones that are not fully characterized: initial reversible binding of RNAP to promoter DNA (P), followed by transition to the open complex. The parameter k_{on} in equation (2.1) indicates the speed at which RNAP scans the DNA, recognizes and binds to arbitrary promoters. The *closed* complex formed hereby is reversible, its stability is reflected by the promoter specific k_{off} in (2.2). In successful initiation RNAP unwinds the duplex DNA locally, and reaches an *open* complex (2.3). This requires a promoter specific effort reflected by the parameter k_{init}. We will later report parameter ranges.

Regulation of initiation.[1] Bacteria apply various strategies to use their genetic material with great effectiveness, in the correct amount and at the appropriate time [5]. Transcription initiation is controlled by DNA binding proteins. Repressors exclude RNAP from promoters by stable binding to overlapping sequences. Activators conversely attach in the vicinity of the promoter, and favor initiation

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

Table 2. Equations for translation and and degradation of mRNA

$m\text{RNA}_{RBS} + \mathit{Ribosome}$	$\rightarrow_{k_{on}}$	$m\text{RNA}_{RBS} \cdot \mathit{Ribosome}$	(2.7)
$m\text{RNA}_{RBS} \cdot \mathit{Ribosome}$	$\rightarrow_{k_{off}}$	$m\text{RNA}_{RBS} + \mathit{Ribosome}$	(2.8)
$m\text{RNA}_{RBS} \cdot \mathit{Ribosome}$	$\rightarrow_{k_{init}}$	$m\text{RNA}_1 \cdot \mathit{Ribosome}$	(2.9)
$m\text{RNA}_n \cdot \mathit{Ribosome}$	$\rightarrow_{k_{elong}}$	$m\text{RNA}_{n+1} \cdot \mathit{Ribosome}$	(2.10)
$m\text{RNA}_{\text{terminator}} \cdot \mathit{Ribosome}$	$\rightarrow_{k_{elong}}$	$m\text{RNA}_{\text{terminator}} + \mathit{Ribosome} + \mathit{Protein}$	(2.11)
$m\text{RNA}_{RBS} + \mathit{Degradosome}$	$\rightarrow_{k_d}$	$m\text{RNA}_{RBS} \cdot \mathit{Degradosome}$	(2.12)

by increasing the transition rate to the open complex k_{init}, or stabilize RNAP by lowering k_{off}.

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

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

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

2.3 mRNA, Translation and Degradation

The flow of mRNA encoded information into proteins is again organized in three phases, and summarized by Table 2. Equation (2.7) represents the initial step of ribosome binding and assembly on a dedicated mRNA sequence, the *r*ibosomal *b*inding *s*ite. As depicted in Fig. 3, the RBS is located close to the mRNA's 5′ end. The ribosome may dissociate readily (2.8). Its stability k_{off} depends on the agreement with an ideal sequence.

Table 3. Quantitative control of gene expression

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

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

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

2.4 Quantitative Control of Gene Expression

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

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

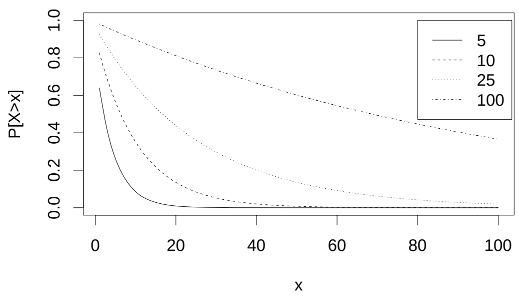


Fig. 5. A *geometric distribution* characterizes the fluctuations around the mean crop of proteins per mRNA. $P[X > x]$ for different mean values.

30 nucleotides per second. This combines to an average transcript elongation delay of roughly 30 seconds, with an average gene length of 1000 nucleotides.

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

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

2.5 Translational Bursting

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

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

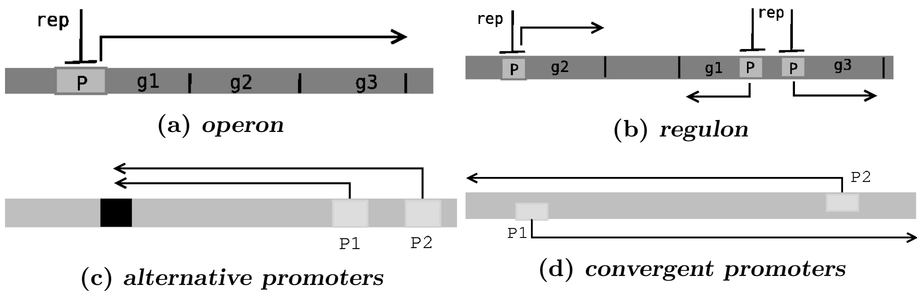


Fig. 6. Particular cases of promoter arrangements

transcript, $P[X > x]$. For example, if the mean crop per transcript is 10, 9% of the transcripts yield each over 25 proteins.

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

2.6 Transcriptional Organization in Bacteria

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

Operons: In bacteria, sequences of several genes are typically co-transcribed in one go from a common promoter. Figure 6(a) presents such an *operon*; operons yield *polycistronic* mRna molecules in which each *cistron* codes for a different protein, bears its own ribosomal binding site and translation start signal. The translational efficiency can vary up to a factor of 1000 across cistrons on the same mRna [58]. Operons eliminate the need for multiple promoters subject to the same regulatory signal, called *regulon* and illustrated in Fig. 6(b). The proteins encoded by the operon are made available at the same time, even if in different quantities.

Alternative promoters for one gene sketched in Fig. 6(c) offer two interesting regulatory strategies in bacteria. Alternative promoter can be activated independently, depending on different environmental signals. Second, alternative

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

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

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

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

3 Stochastic Pi Calculus with Pattern Guarded Inputs

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

3.1 Process Expressions and Reduction Semantics

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

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

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

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

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

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

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

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

Table 6 defines the *reduction relation*. The first axiom tells how to interpret choices; it comprises channel communication and pattern matching. It applies to two complementary matching alternatives in parallel choices, an output alterna-

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

Table 5. Axioms of the structural congruence

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

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

Communication, choice, pattern matching:

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

Unfolding of parametric processes:

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

Closure rules:

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

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

The unfolding axiom applies one of the definitions of the parametric processes in a given set Δ. An application $A(\overline{y})$ reduces in one step to definition P, in which the formal parameters $\overline{y}$ have been replaced by the actual parameters $\overline{x}$. Note that parametric definitions can be recursive, and that another call of A may be contained in P. The usual closure rules state that reduction can be applied in arbitrary contexts, but not below choices or in definitions.

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

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

3.2 Example: Semaphore

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

$$\begin{aligned}\mathsf{Semaphore\_free(me)} &\triangleq \mathsf{me?bind().Semaphore\_bound(me)}\\ \mathsf{Semaphore\_bound(me)} &\triangleq \mathsf{me?free().Semaphore\_free(me)}\end{aligned}$$

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.

$$\begin{aligned}&\mathsf{Semaphore\_bound(s)} \mid \mathsf{s!bind().}\mathbf{0} \mid \mathsf{s!free().}\mathbf{0}\\ \to\ &\mathsf{s?free().Semaphore\_free(s)} \mid \mathsf{s!bind().}\mathbf{0} \mid\\ &\mathsf{s!free().}\mathbf{0} \to \mathsf{Semaphore\_free(s)} \mid \mathsf{s!bind().}\mathbf{0} \to\\ &\mathsf{s?bind().Semaphore\_bound(s)} \mid \mathsf{s!bind().}\mathbf{0} \to\\ &\mathsf{Semaphore\_bound(s)}\end{aligned}$$

In a first step, the Semaphore_bound at s unfolds its definition. This creates an input offer on s to receive a free message. Other messages cannot be received over s in this state, in particular no bind requests. Hence, the site s cannot get bound a second time. Only once the free message got received, s was able to accept the next bind request, while becoming bound again.

3.3 Stochastic Scheduling

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

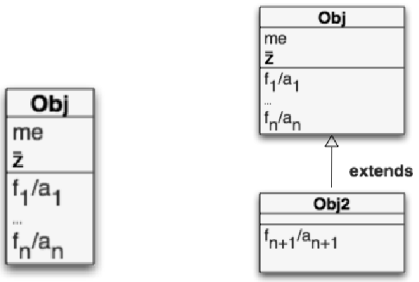
An abstract chemical expression is a multiset of molecules. In the set of the π-calculus, a molecule will be either a choice $C_1 + \ldots + C_n$ or an application $A(\overline{y})$, or more precisely, a congruence class of such an expression with respect to the structural congruence. A parallel composition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

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

$$\begin{aligned}&x!f(\overline{y}).P_1 + \ldots \mid x?f(\overline{z}).P_2 + \ldots \quad \to^{\rho_x(f)} \quad P_1 \mid P_2[\overline{z} \mapsto \overline{y}]\\ &A(\overline{x}) \quad \to^{\infty} \quad P[\overline{y} \mapsto \overline{x}] \qquad \text{if } A(\overline{y}) \triangleq P \text{ in } \Delta\end{aligned}$$

```
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)
```

Fig. 7. Semaphore



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

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

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

3.4 Modules

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

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

4 Concurrent Objects

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

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

4.1 Single-Profile Objects

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

In the π-calculus with pattern guarded choices, we represent objects as follows. The names of object functions correspond to the function names of the π-calculus; the class of the object is a parametric process name, say Obj. Every object of this class is represented by an application Obj(me,$\overline{z}$) of the class to a channel me identifying the object and a sequence of parameters $\overline{z}$. Its interface is defined by a choice of input offers on channel me, each guarded by one of the functions of the object:

```
Obj(me,$\overline{z}$) ≜
     me?f$_1$($\overline{x_1}$).P$_1$
  +  ...
  +  me?f$_n$($\overline{x_n}$).P$_n$
```

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

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

```
Obj(me,$\overline{z}$) ≜ C$_1$ + ... + C$_n$
```

Inheritance refines this class to Obj2 by adding further choices:

```
Obj2 **extends** Obj

Obj2(me,$\overline{z}$) **extended by** C$_{n+1}$ + ...+ C$_m$
```

This definition by inheritance can be resolved as follows:

```
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,Cal)
   Nil(me) ≜ me?isNil(c).c!true().Nil(me)
```

Fig. 9. Persistent list

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

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

4.2 Examples: Persistent and Degradable Lists

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

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

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

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

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

The attentive reader may have noticed a detail of our Walker process. It is actively sending requests to objects, which keep *waiting* for input over their respective me channels. The same holds for all devices processing representatives

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

Fig. 10. Degradable list

of macromolecules (i.e., data structures) in the remainder of this paper: the RNAP and ribosome abstractions proceed by *calling* functions[3] that are *offered* by DNA and mRNA representatives.

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

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

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

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

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

A non-persistent list [a,b] can now be built by the same definition as a persistent list [a,b]. The only difference is that we have to import module 'non-persistent list' instead of 'persistent list'. Destructing a non-persistent list is easy. A Killer

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

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

proceeds like a Walker, except that it kills a Node before continuing with the next:

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

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

4.3 Multi-profile Objects

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

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

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

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

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

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

The latter substitution renames all recursive calls to some profile Obj_$\mathsf{p_i}$ into recursive calls to Obj2_$\mathsf{p_i}$ for $1 \leq i \leq n$.

4.4 Examples: Persistent and Degradable Queueing Lists

The persistent and degradable lists presented so far can be traversed by several visitors at the same time, e.g. by a Walker and by a Reader. Each visitor proceeds

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

Fig. 11. Persistent queueing list

over the nodes, and hereby draws waiting times independently of the others. This permits overtaking of one visitor by another, which is however not possible in transcription of DNA, nor in translation of mRNA or its degradation.

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

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

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

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

5 Modeling Transcription and Translation

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

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

Fig. 12. Degradable queueing list

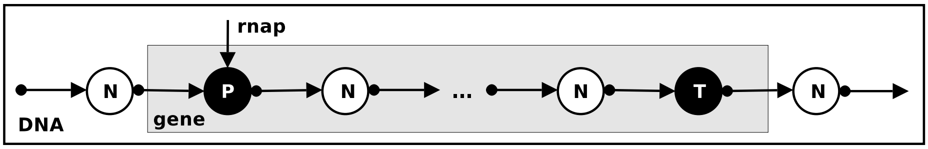


Fig. 13. We abstract DNA as list, extending on the module 'persistent queueing list'. Each list member has the private knowledge of its successor. **P**romoters control the access of RNAP to coding DNA, they can be contacted over the globally visible channel rnap. They abstract the behavior of the promoter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

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

5.1 DNA and Transcription

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

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

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

Fig. 14. Rnap module

Rnap has four profiles, listed in Fig. 14. Three deal with transcription initiation – they have corresponding Promoter profiles. The fourth covers elongation, and roughly resembles our previous Walker. We first consider formation of the closed promoter complex, summarized by equation (2.1): Rnap_free invocates bind over the global channel rnap (line 6). It waits for satisfaction by a Promoter, that is nondeterministically selected among several available in profile free, and extrudes its me channel. As the bind interaction succeeds, Rnap and Promoter switch to their closed profiles. They now jointly represent the closed promoter complex. As such they can interact over Promoter's shared me channel, by the competing functions unbind and initiate. The race between these is controlled by the rates k_{init} (2.2) and k_{off} (2.3), which enter the model over the ρ function that quantifies the Promoter's channel me. Unbinding without transcription initiation is straightforward. It causes transitions from bound to free (line 8). If conversely initiate succeeds, both switch to their open profile (line 9). Transcription subsequently launches upon a startTranscript call, reflecting (2.4). Promoter_open creates the first transcript segment, and returns a reference to its growing end rna. Rnap_open continues as elongating, using rna and the second parameter dna, that it obtains by a getNext call.

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

Dna. Our module 'DNA' includes the complementary specification of Promoter, Nucleotide, and Terminator (Fig. 15). Promoter implements the explicit

```
module 'DNA'
channel
   rnap with bind/1
import
   List(Node, Nil) from 'persistent queueing list '
   Mrna(Node, Terminator ,RBS) from 'mRNA'
export
  Promoter extends List.Node by unbind/0,
           initiate/0,startTranscript/1
  Nucleotide extends List.Node by isTerm/1,elongate/2
  Terminator extends List.Node by isTerm/1,elongate/1
define
  Promoter(me, next) ≜ Promoter_free(me, next)
  Promoter_free(me, next)  extended by
     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)  extended List.Node by
     me?startTranscript(c).
        new him(ρ_rbs).new rna(ρ_rna).c!rna.
        Promoter_open(me, next)| Mrna.RBS(him , 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(ρ_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?elongate(rna).new last(ρ_rna).
     Terminator_bound(me, next , v)
     | Mrna.Terminator(rna , last , v')  |  List.Nil(last)
```

Fig. 15. Dna module

access control for genes. It has three profiles analogous to those of Rnap, with which it synchronizes transitions: Promoter_free is dual to Rnap_free.[5]

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

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

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

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

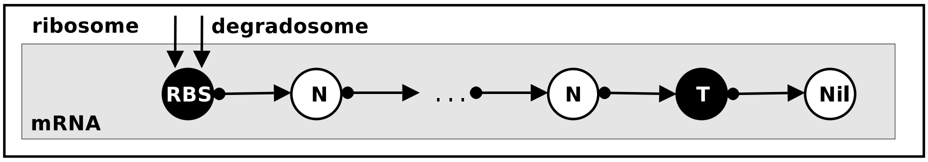


Fig. 16. Rna is based on the module 'degradable queueing list'

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

Parameterization. Table 7 gives sample values of the ρ function that attributes rates to on object's functions. Functions not associated with a rate are instantaneous. Recall that each object is identified by its me channel, over which its function are invocated. This allows to associate method calls on different objects of the same class with distinct rates, as useful for initiation rates of promoters or ribosomal binding sites.

5.2 mRNA, Translation and Degradation

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

The two-profile RBS represents the $5'$ end of mRna, including the ribosomal binding site and the translation start signal. It implements the co-transcriptional

```
 1  module 'mRNA'
 2  channel
 3     ribosome with bind/1
 4     degradosome with bind/1
 5  import
 6     List(Node, Nil) from 'degradable queueing list'
 7     Protein         from 'favorite protein'
 8  export
 9     RBS extends List.Node by init/1, unbind/0
10     Nucleotide extends List.Node by isTerm/1, elongate/0
11     Terminator extends List.Node by isTerm/1, elongate/0
12  define
13     RBS(me, next) ≜ RBS_free(me, next)
14     RBS_free(me, next) ≜ extended by
15        degradosome?bind(c).next!bind().c!next.0
16      + ribosome?bind(c).c!me.RBS_bound(me, next)
17     RBS_bound(me, next) ≜ extended by
18        me?init(c).next!bind().c!next.RBS_free(me, next)
19      + me?unbind().RBS_free(me, next)
20     Nucleotide_bound(me, next, val) extended by
21        me?isTerm(c).c!false().Nucleotide_bound(me, next, v)
22      + me?elongate().Nucleotide_bound(me, next, v)
23     Terminator_bound(me, next, val) extended by
24        me?isTerm(c).c!true().Terminator(me)
25      + me?elongate?().
26          Terminator_bound(me, next, val) | Protein
```

Fig. 17. mRNA module

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

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

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

```
module 'ribosome'
channel ribosome
export Ribosome
define
   Ribosome() ≜ Ribosome_free()
   Ribosome_free() ≜ ribosome!bind(c).c?rna.
       Ribosome_bound(rna)
   Ribosome_bound(rna) ≜ new c.
      rna!init(c).c?next.Ribosome_elongating(next)
    + rna!unbind().Ribosome_free
   Ribosome_elongating(rna) ≜ new c1.rna!isTerm(c1)
       c1?true().rna!elongate().Ribosome_free()
    +  c1?false().rna!elongate().
          new  c2.rna!getNext(c2).  c2?next.
           Ribosome_elongating(next)
```

Fig. 18. Ribosome module

```
module 'degradosome'
channel 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() .node!kill().Degradosome_free
    + b?false().new c.node!getNext(c).c?next.rna!kill().
          Degradosome_working(next)
```

Fig. 19. Degradosome module

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

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

```
Protein ≜ prot?kill().0
```

5.3 Extensions

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

Operons and polycistronic mRNA. We devote a module to operons, and the polycistronic mRNA transcribed from them (Fig. 20). It defines the processes

```
module 'operon'
channel ribosome
import
   List(Node) from 'persistent queueing list'
   Mrna(Nucleotide ,Terminator ,RBS) from 'mRNA'
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(ρ_rna). new_rna_nxt2(ρ_rna).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)
```

Fig. 20. Operon module

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

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

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

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

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

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

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

Fig. 21. Tandem promoter module

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

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

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

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

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

5.4 Challenging Cases

The integration of more biological detail can become increasingly challenging. This becomes clearer on closer inspection of the biological matter, as points related to secondary structures of mRNA (intrinsic termination of transcription, transcriptional attenuation), and two-way traffic on DNA and mRNA.

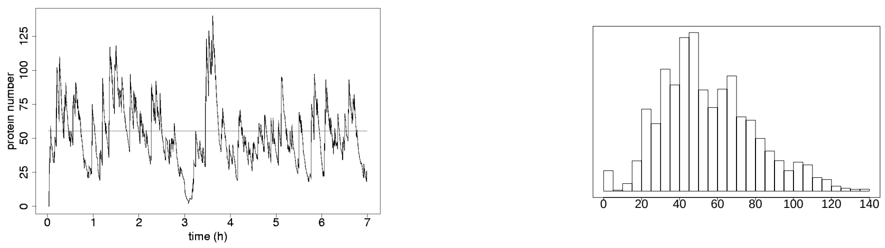
An example is *transcriptional attenuation*, in which transcription termination is determined by the efficiency of translation of the nascent transcript. The crucial detail is that intrinsic termination depends on more than mere recognition of a terminator sequence which actually comprises two portions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

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

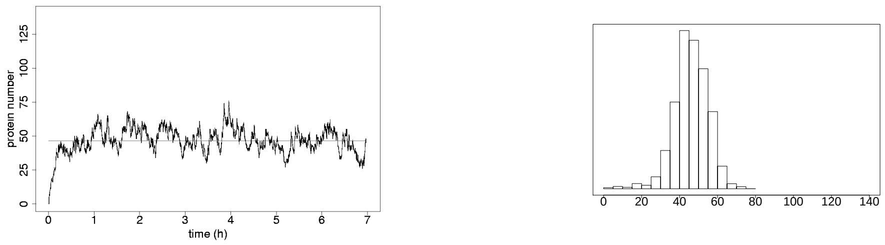
Two-way traffic occurs both on DNA and mRNA. The concurrent control supported so far can not yet account for traffic problems on double-stranded DNA (one of two RNAP falls off after a head-on collision), nor for details of mRNA decay. So far we only realized queueing control on single stranded macromolecules,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

6 Simulation

In this section we present simulations obtained from our model components with the BioSPI tool [52], after encoding pattern guarded inputs within the stochastic π-calculus [35]. Our focus lies on the effect of *translational bursting* introduced in Sec. 2.5. We hence compare the dynamics of unregulated expression of a single gene under variation of two crucial parameters. The underlying model is that of a single gene comprising 1000 nucleotides, its transcription into mRNA, and subsequent translation and degradation. As experimentally confirmed by Ozbudak et. al [45], the variation of transcription of translation initiation parameters leads to significant differences in expression patterns. These would not appear in continuous deterministic simulation, which nevertheless yield comparable *average* expression levels, nor can they be reproduced by one step models of gene expression as [6]. Our parameter combinations are the following:



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



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

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

(a) In the setting we refer to as *bursty*, the promoter yields rare transcription initiations (k_{init} of 0.01). This is combined with efficient translation (k_{init} of 1).

(b) The *smooth* setting inverts the parameters: transcription initiates more frequently (k_{init} of 0.1), while translation initiation is rarer (k_{init} of 0.1).

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

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

Simulations based on the *smooth* setting have a clearly distinct behavior, as shows Fig. 22(b) where the protein level only weakly fluctuates around an averate

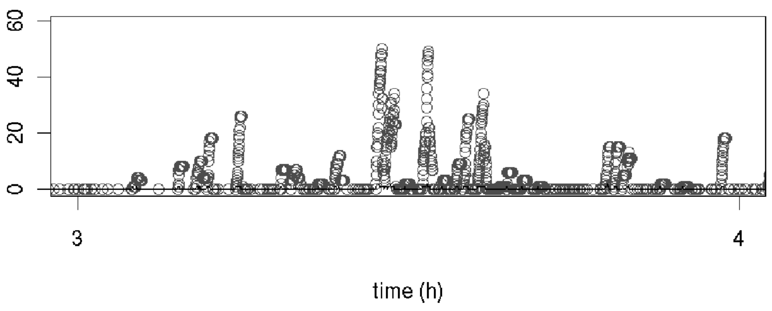


Fig. 23. Concurrent translation of mRNA by multiple ribosomes

of 46. The histogram confirms that the distribution is pronouncedly narrowed compared with the previous setting.

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

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

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

Translational bursting. We could not yet observe the origin of the strong bursts in protein numbers in Fig. 22(a). This motivates further inspection of setting *bursty*'s simulation. Figure 23 displays the numbers of translating ribosomes (circles) within the fourth hour of the simulation period, in which the protein pool empties, and then rapidly replenishes to the maximum. While the number of full mRNA molecules never exceeds two (data not shown), these are simultaneously processed by up to 50 ribosomes. As discussed in Sec. 2.5 the strongest bursts in protein levels occur when for a mRNA, the number of translations by far exceeds the average. The circles forming the bottom line may first appear peculiar; they can be explained as follows. For setting *bursty* new transcriptions are *completed* every 100 seconds. However, recall that nascent transcripts are translated co-transcriptionally. This means that whenever some RNAP is producing a transcript (wich takes around 100 seconds), one or more ribosomes closely follow it on the nascent mRNA. Hence the bottom line reflects that there is virtually always *some* coupled transcription and concomitant translation going on. The column-reminiscent peaks mark transcripts yielding exceptionally high protein crops; this is in agreement with a geometric distribution.

7 Conclusion

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

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

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

References

1. Elio A. Abbondanzieri, William J. Greenleaf, Joshua W. Shaevitz, Robert Landick, and Steven M. Block. Direct observation of base-pair stepping by RNA polymerase. *Nature*, 438:460–465, 2005.
2. David Adalsteinsson, David McMillen, and Timothy Elston. Biochemical Network Stochastic Simulator (BioNetS): software for stochastic modeling of biochemical networks. *BMC Bioinformatics*, 5(1):24, 2004.

3. Adam Arkin, John Ross, and Harley H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected Escherichia coli cells. *Genetics*, 149:1633–1648, 1998.
4. S Banerjee, J Chalissery, I Bandey, and RJ Sen. Rho-dependent transcription termination: More questions than answers. *Journal of Microbiology*, 44(1):11–22, 2006.
5. Anne Barnard, Alan Wolfe, and Stephen Busby. Regulation at complex bacterial promoters: how bacteria use different promoter organizations to produce different regulatory outcomes. *Current Opinion in Microbiology*, 7:102–108, 2004.
6. Ralf Blossey, Luca Cardelli, and Andrew Phillips. A compositional approach to the stochastic dynamics of gene networks. *Transactions on Computational Systems Biology*, IV:99–122, 2006. LNCS vol 3939.
7. S Brenner, F Jacob, and M Meselson. An unstable intermediate carrying information from genes to ribosomes for protein synthesis. *Nature*, 190:576–581, 1961.
8. Luca Cardelli. Abstract machines of systems biology. *Transactions on Computational Systems Biology*, III:145–168, 2005. LNCS vol 3737.
9. Luca Cardelli. Brane calculi: interactions of biological membranes. In *Proceedings of CMSB 2004*, volume 3082 of *Lecture Notes in Bioinformatics*, pages 257–278, 2005.
10. A. J. Carpousis. The Escherichia coli RNAdegradosome: structure, function and relationship to other ribonucleolytic multienzyme complexes. *Biochemical Society Transactions*, 30(2):150–154, 2002.
11. Trent A. Carrier and J. D. Keasling. Mechanistic modeling of mRNA decay. *Journal of Theoretical Biology*, 189:195–209, 1997.
12. Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, François Fages, , and Vincent Schächter. Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 325(1):24–44, 2004.
13. Nathalie Chabrier-Rivier, Francois Fages, and Sylvain Soliman. The biochemical abstract machine BioCham. In *Proceedings of CMSB 2004*, volume 3082 of *Lecture Notes in Bioinformatics*, pages 172–191, 2005.
14. Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.
15. Madhukar S. Dasika, Anshuman Gupta, and Costas D. Maranas. DEMSIM: a discrete event based mechanistic simulation platform for gene expression and regulation dynamics. *Journal of Theoretical Biology*, 232(1):55–69, 2005.
16. Patrick P. Dennis, Mans Ehrenberg, and Hans Bremer. Control of rRNA synthesis in Escherichia coli: a systems biology approach. *Microbiology and Molecular Biology Reviews*, 68(4):639–668, 2004.
17. Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138, 1971.
18. Denys Duchier and Céline Kuttler. Biomolecular agents as multi-behavioural concurrent objects. In *Proceedings of the First International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord 2005)*, volume 150 of *Electronic Notes in Theoretical Computer Science*, pages 31–49, 2006.
19. Johan Elf and Mans Ehrenberg. What makes ribosome-mediated trascriptional attenuation sensitive to amino acid limitation? *PLoS Computational Biology*, 1(1):14–23, 2005.
20. B. Alberts et al. *Molecular Biology of the Cell*. Garland Science, 2002.
21. Joachim Frank and Rajendra Kumar Agrawal. A ratchet-like inter-subunit reorganization of the ribosome during translocation. *Nature*, 406:318–322, 2000.

22. Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976.
23. I Golding, J Paulsson, SM Zawilski, and EC Cox. Real-time kinetics of gene activity in individual bacteria. *Cell*, 123(6):1025–1036, 2005.
24. J. Gowrishankar and R. Harinarayanan. Why is transcription coupled to translation in bacteria? *Molecular Microbiology*, 54(3):598–603, 2004.
25. F Gros, H Hiatt, W Gilbert, CG Kurland, RW Risebrough, and JD Watson. Unstable ribonucleic acid revealed by pulse labeling of Escherichia coli. *Nature*, 190:581–585, 1961.
26. Marianne Grunberg-Manago. Messenger RNA stability and its role in control of gene expression in bacteria and phages. *Annual Reviews Genetics*, pages 193–227, 1999.
27. Tina M Henkin. Transcription termination control in bacteria. *Current Opinion in Microbiology*, 3(2):149–153, 2000.
28. Lilian M. Hsu. Promoter clearance and escape in prokaryotes. *Biochimica et Biophysica Acta*, 1577:191–207, 2002.
29. Mads Kaern, William J. Blake, and J. J. Collins. The engineering of gene regulatory networks. *Annual Review Biomedical Engineering*, 5:179–206, 2003.
30. Mads Kaern, Timothy Elston, William Blake, and James Collins. Stochasticity in gene expression: from theories to phenotypes. *Nature Reviews Genetics*, 6(6):451–467, 2005.
31. Andrzej M. Kierzek. STOCKS: STOChastic Kinetic Simulations of biochemical systems with Gillespie algorithm. *Bioinformatics*, 18(3):470–481, 2002.
32. Andrzej M. Kierzek, Jolanta Zaim, and P. Zielenkiewicz. The effect of transcription and translation initiation frequencies on the stochastic fluctuations in prokaryotic gene expression. *Journal of Biological Chemistry*, 276:8165–8172, 2001.
33. Oren Kobiler, Assaf Rokney, Nir Friedman, Donald L. Court, Joel Stavans, and Amos B. Oppenheim. Quantitative kinetic analysis of the bacteriophage λ genetic network. *Proceedings of the National Academy of Sciences USA*, 102(12):4470–4475, 2005.
34. Céline Kuttler and Joachim Niehren. Gene regulation in the pi calculus: Simulating cooperativity at the lambda switch. *Transactions on Computational Systems Biology VII*, 2006. volume 4230 of *Lecture Notes in Computer Science*. Springer, 2006.
35. Céline Kuttler, Cédric Lhoussaine, and Joachim Niehren. A stochastic pi calculus for concurrent objects. Technical report, INRIA, 2006.
36. Karolina Maciag, Steven J Altschuler, Michael D Slack, Nevan J Krogan, Andrew Emili, Jack F Greenblatt, Tom Maniatis, and Lani F Wu. Systems-level analyses identify extensive coupling among gene expression machines. *Molecular Systems Biology*, 2006.
37. Tom Maniatis and Robin Need. An extensive network of coupling among gene expression machines. *Nature*, 416:499–506, 2002.
38. Harley H. McAdams and Adam Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences USA*, 94:814–819, 1997.
39. William R. McClure. Mechanism and control of transcription initiation in prokaryotes. *Annual Review Biochemistry*, 54:171–204, 1985.
40. Robin Milner. *Communicating and Mobile Systems: the π-calculus.* Cambridge University Press, 1999.

41. Robin Milner. *Computer Systems: Theory, Technology, and Applications. A tribute to Roger Needham*, chapter What's in a name?, pages 205–211. Monographs in Computer Science. Springer, 2004.
42. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100:1–77, 1992.
43. Petra J. Neufing, Keith E. Shearwin, and J. Barry Egan. Establishing lysogenic transcription in the temperate coliphage 186. *Journal of Bacteriology*, 183(7):2376–2379, 2001.
44. George Orphanides and Danny Reinberg. A unified theory of gene expression. *Cell*, 108:439–451, 2002.
45. Ertugrul M. Ozbudak, M Thattai, I Kurtser, A Grossman, and A.D. van Oudenaarden. Regulation of noise in the expression of a single gene. *Nature Genetics*, 31:69–73, 2002.
46. Brian J. Paul, Wilma Ross, Tamas Gaal, and Richard L. Gourse. rRNA transcription in E. Coli. *Annual Review Genetics*, 38:749–770, 2004.
47. Hervé Paulino, Pedro Marques, Luis Lopes, Vasco T. Vasconcelos, and Fernando Silva. A multi-threaded asynchronous language. In *7th International Conference on Parallel Computing Technologies*, volume 2763 of *Lecture Notes in Computer Science*, pages 316–323. Springer, 2003.
48. Andrew Phillips and Luca Cardelli. A correct abstract machine for the stochastic pi-calculus. *Transactions on Computational Systems Biology*, 2006. Special issue of BioConcur 2004. In the press.
49. D. Prandi, C. Priami, and P. Quaglia. Process calculi in a biological context. *Bulletin of the EATCS*, 85:53–69, 2005.
50. Corrado Priami. Stochastic π-calculus. *Computer Journal*, 6:578–589, 1995.
51. Corrado Priami and Paola Quaglia. Beta binders for biological interactions. In *Proceedings of CMSB 2004*, volume 3082 of *Lecture Notes in Bioinformatics*, pages 20–33, 2005.
52. Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80:25–31, 2001.
53. Mark Ptashne. *A Genetic Switch: Phage Lambda Revisited.* Cold Spring Harbor Laboratory Press, 3rd edition, 2004.
54. Mark Ptashne and Alexander Gann. *Genes and Signals.* Cold Spring Harbor Laboratory Press, 2002.
55. Stephen Ramsey, David Orrell, and Hamid Bolouri. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *Journal of Bioinformatics and Computational Biology*, 3(2):415–436, 2005.
56. Jonathan M. Raser and Erin K. O'Shea. Noise in Gene Expression: Origins, Consequences, and Control. *Science*, 309(5743):2010–2013, 2005.
57. António Ravara and Vasco T. Vasconcelos. Typing non-uniform concurrent objects. In *CONCUR'00*, volume 1877 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2000.
58. PN Ray and ML Pearson. Evidence for post-transcriptional control of the morphogenetic genes of bacteriophage lambda. *Journal Molecular Biology*, 85(1):163–175, 1974.
59. Aviv Regev and Ehud Shapiro. Cells as computation. *Nature*, 419:343, 2002.
60. Maria Elena Regonesi, Marta Del Favero, Fabrizio Basilico, Federica Briani, Louise Benazzi, Paolo Tortora, Pierluigi Mauri, and Gianni Deho. Analysis of the Escherichia coli RNA degradosome composition by a proteomic approach. *Biochimie*, 88(2):151–161, 2006.

61. Hessam S. Sarjoughian and Francois E. Cellier, editors. *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-based Theories and Methodologies.* Springer Verlag New York Inc., 2001.
62. Ursula Schmeissner, Donald Court, Hiroyuki Shimatake, and Martin Rosenberg. Promoter for the establishment of repressor synthesis in bacteriophage lambda. *Proceedings of the National Academy of Sciences USA*, 77(6):3191–3195, 1980.
63. Madeline Shea and Gary K. Ackers. The O_R control system of bacteriophage lambda: A physical-chemical model for gene regulation. *Molecular Biology*, 181:211–230, 1985.
64. KW Shearwin, BP Callen, and JB Egan. Transcriptional interference - a crash course. *Trends in Genetics*, 21:339–345, 2005.
65. Kim Sneppen and Giovanni Zocchi. *Physics in Molecular Biology.* Cambridge University Press, 2005.
66. D. A. Steege. Emerging features of mRNA decay in bacteria. *RNA*, 6(8):1079–1090, 2000.
67. Peter S. Swain. Efficient attenuation of stochasticity in gene expression through post-transcriptional control. *Journal of Molecular Biology*, 344:965–976, 2004.
68. Vasco T. Vasconcelos and Mario Tokoro. A typing system for a calculus of objects. In *1st International Symposium on Object Technologies for Advanced Software*, volume 472 of *Lecture Notes in Computer Science*, pages 460–474. Springer, 1993.
69. E.O. Voit. *Computational Analysis of Biochemical Systems: A Practical Guide for Biochemists and Molecular Biologists.* Cambridge University Press, 2000.
70. G von Heijne, L Nilsson, and C Blomberg. Translation and messenger RNA secondary structure. *Journal of Theoretical Biology*, 68:321–329, 1977.
71. Rolf Wagner. *Transcription Regulation in Prokaryotes.* Oxford University Press, 2000.
72. Charles Yanofsky. Trancription attenuation. *Journal Biological Chemistry*, pages 609–612, 1988.

Automated Abstraction Methodology for Genetic Regulatory Networks*

Hiroyuki Kuwahara[1], Chris J. Myers[1],
Michael S. Samoilov[2], Nathan A. Barker[1], and Adam P. Arkin[2]

[1] University of Utah, Salt Lake City, UT 84112, USA
{kuwahara, myers, barkern}@vlsigroup.ece.utah.edu
[2] Howard Hughes Medical Institute, University of California, Berkeley
Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
{mssamoilov, aparkin}@lbl.gov

Abstract. In order to efficiently analyze the complicated regulatory systems often encountered in biological settings, abstraction is essential. This paper presents an automated abstraction methodology that systematically reduces the small-scale complexity found in genetic regulatory network models, while broadly preserving the large-scale system behavior. Our method first reduces the number of reactions by using rapid equilibrium and quasi-steady-state approximations as well as a number of other stoichiometry-simplifying techniques, which together result in substantially shortened simulation time. To further reduce analysis time, our method can represent the molecular state of the system by a set of scaled Boolean (or n-ary) discrete levels. This results in a chemical master equation that is approximated by a Markov chain with a much smaller state space providing significant analysis time acceleration and computability gains. The genetic regulatory network for the phage λ lysis/lysogeny decision switch is used as an example throughout the paper to help illustrate the practical applications of our methodology.

1 Introduction

Numerous methods have been proposed for modeling genetic regulatory networks [1,2]. While many traditional approaches have relied on a differential equation representation as inferred from a set of underlying biochemical reactions, there has been a growing appreciation of their limitations [3,4,5,6]. In particular, differential equation analysis of genetic networks generally assumes that the number of molecules in a cell is high and their concentrations can be viewed as continuous quantities, while the underlying reactions are presumed to occur deterministically. However, in genetic networks these assumptions frequently do not hold. For example, DNA molecules are typically present in single digit quantities while some promoters can lead to substantial fluctuations in transcription/translation rates and essentially non-deterministic expression characteristics [7,8].

* This material is based upon work supported by the National Science Foundation under Grant No. 0331270.

In these situations, accurate genetic regulatory network modeling requires the use of a discrete and stochastic process description such as the *master equation formalism* [9]. This approach describes well-stirred (bio)chemical systems at the individual reaction level by exactly tracking the quantities of each molecular species and treating each reaction as a separate random event. It further allows the exact discrete simulation of system behavior via Gillespie's *Stochastic Simulation Algorithm* (SSA) [10]. Unfortunately, the computational requirements for such simulations are generally practical only for relatively small systems with no major reaction time-scale separations. Even then, the computational demands can be very high. For example, Arkin et al.'s numerical analysis of the phage λ decision network using a hybrid stochastic kinetic and statistical-thermodynamic model required a 200-node supercomputer [3].

Several other techniques for reducing the computational costs of stochastic simulations have been proposed [11]. For instance, Gibson and Bruck [12] developed a method to improve Gillespie's first reaction method [10] by reducing the random numbers generated allowing them to simulate the λ network on 10 desktop workstations [13]. While this method does improve efficiency significantly for systems with many species and reaction channels, every reaction event must still be simulated one at a time. Ultimately, given the substantial computational requirements of stochastic simulations and comparative complexities of *in situ* genetic regulatory networks, abstraction is absolutely essential for efficient analysis of any realistic system. This abstraction can be achieved either during the simulation or the model generation stage. An example of abstraction during simulation is Gillespie's explicit τ-leaping method [14]. This method approximates the number of firings of each reaction in a pre-defined interval rather than executing each reaction individually. While this and similar methods [15,16,17] are very promising, they may not perform well for systems with rapidly changing reaction rates, such as those driven by small molecular counts as frequently encountered in gene-regulatory systems. Rao and Arkin performed model abstraction by using (bio)chemical insight in combination with the *quasi-steady-state assumption* to remove fast reactions (thus reducing the problem dimensionality), and then applied a modified version of SSA to the simplified model [18]. Since this method is not automated, it is difficult to apply to reactions within complex biological networks. Furthermore, the quasi-steady-state approximation only represents one of many approximations that can be used to estimate the dynamics of a biological reaction network.

This paper presents a generalized and automated model abstraction methodology that systematically reduces the small scale complexity found in *REaction-Based* (REB) models of genetic regulatory networks (i.e., models composed of a set of chemical reactions), while broadly preserving the large scale system behavior. Notably, though many model reductions of (bio)chemical networks have traditionally been performed manually (i.e., "by hand") this practice is time-consuming and is susceptible to errors in large model transformations. For example, whereas the quasi-steady-state approximation has long been carried out to reduce the complexity of biochemical networks, this practice can result in inaccuracies when the underlying hypothesis of the approximation is ignored and the application of the quasi-steady-state approximation is inappropriate [19]. Thus, by automating and systematizing this process, our method is able to significantly ameliorate the problem of computability by lowering the model translation error rate and improving simulation times by reducing system complexity.

Our methodology—outlined in Figure 1—begins with a REB model, which could be simulated using SSA or one of its variants though at a substantial computational cost. To reduce the cost of simulation, this paper describes an automatic method that we have implemented for simplifying the original REB model by applying several abstraction methods that leverage the *rapid equilibrium* and quasi-steady-state assumptions. The result is a new abstracted model with less reactions and species which substantially lowers the cost of stochastic simulation. To further reduce the complexity of the system as well as analysis time, this simplified REB model can be automatically translated into a *Stochastic Asynchronous Circuit* (SAC) model by encoding chemical species molecule counts into Boolean (or n-ary) levels. This model can then be efficiently analyzed using the Markov chain analysis method within the asynchronous circuit tool `ATACS` [20].

This paper further exemplifies the applicability of our model abstraction methodology to genetic regulatory networks by applying it to the phage λ developmental decision circuit [3]. Hence, various abstractions in this work target the specific features found in genetic regulatory networks to reduce the analysis time. For example, one abstraction method is used to reduce the low-level description of the interactions of transcription factors and *cis*-regulatory elements which, as noted earlier, are elements that are typically present in small quantity and therefore susceptible to the discrete-stochastic effect. This abstraction substantially accelerates the computation that otherwise would require an expensive non-deterministic treatment. Moreover, the ON-OFF switching behavior often found in genetic regulatory networks is suitable for our abstraction methodology, especially for a SAC model generation. Thus, whereas we believe that our model abstraction methodology can in theory be applied to any (bio)chemical networks, this paper concentrates on genetic regulatory networks as a proof of concept to evaluate our methodology which includes abstraction methods tailored for such networks.

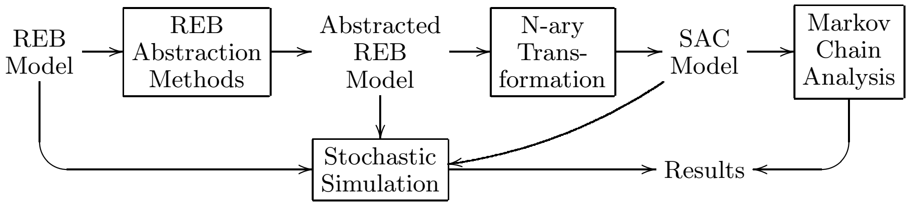


Fig. 1. Automated model abstraction tool flow

2 Reaction-Based Abstraction Methods

In chemical and biological molecular systems, including genetic regulatory networks, reaction-based representations typically provide the most detailed level of specification for the underlying system structure and dynamics [21]. Reaction-based abstraction methods are used to reduce a REB model's size by merging reactions, removing irrelevant reactions, etc. We have implemented several such techniques, each traversing the graph structure of the REB model and applying transformations to it when the respective conditions are satisfied. The result is a new REB model with fewer reactions and/or species. This section first describes the REB model formally, and then presents several abstraction methods as well as discusses how they are applied to our λ model. Finally, this section presents simulation results for the λ model before and after these abstractions.

2.1 Reaction-Based Model

The REB model is a bipartite weighted directed graph that connects species based on the interactions that they can have via a set of reaction channels.[1]

Definition 1. *A REB model is specified with a 5-tuple* $\langle \mathbf{S}, \mathbf{R}, \mathbf{R_{rev}}, \mathbf{E}, \mathbf{K} \rangle$ *where* $\mathbf{S}$ *is the set of species nodes,* $\mathbf{R}$ *is the set of reaction nodes,* $\mathbf{R_{rev}} \subseteq \mathbf{R}$ *is the set of reversible reactions,* $\mathbf{E} : ((\mathbf{S} \times \mathbf{R}) \cup (\mathbf{R} \times \mathbf{S})) \to \mathbb{N}$ *is a function that returns the stoichiometry of the species with respect to a reaction, and* $\mathbf{K} : \mathbf{R} \to (\mathbb{R}^{|\mathbf{S}|} \to \mathbb{R})$ *are the kinetic rate laws for the reactions.*

For example, a REB model with a single reversible reaction r_1 of the form:

$$2s_1 + s_2 \underset{k_{-1}}{\overset{k_1}{\rightleftarrows}} s_1 + s_2 + s_3$$

contains the following sets:

$$\begin{aligned}
\mathbf{S} &= \{s_1, s_2, s_3\}, \\
\mathbf{R} &= \{r_1\}, \\
\mathbf{R_{rev}} &= \{r_1\}, \\
\mathbf{E} &= \{((s_1, r_1), 2), ((s_2, r_1), 1), ((s_3, r_1), 0), \\
&\quad ((r_1, s_1), 1), ((r_1, s_2), 1), ((r_1, s_3), 1)\}, \\
\mathbf{K} &= \{(r_1 \to (([s_1], [s_2], [s_3]) \to (k_1[s_1]^2[s_2] - k_{-1}[s_1][s_2][s_3])))\}.
\end{aligned}$$

In the kinetic rate law, $[s]$ is a variable that represents the state of species s. Note that $[s]_0$ is used to denote the initial number of molecules of species s. Also note that the user can specify a set of *interesting species*, (i.e., $\mathbf{S_i} \subseteq \mathbf{S}$), which should never be abstracted, so that they can be analyzed.

If a reaction consumes a species, then that species is a *reactant* for that reaction. If a reaction produces a species, then that species is a *product* for that reaction. If a species is neither produced nor consumed by a reaction, then it is a *modifier* to that reaction. $\mathbf{R_s^r}$, $\mathbf{R_s^p}$, and $\mathbf{R_s^m}$ are the sets of reactions in which

[1] A REB model can be described via an emerging standard, the *Systems Biology Markup Language* (SBML) [22]. Graphical user interface tools such as BioSPICE's PathwayBuilder [23] allow researchers a convenient mechanism to create such models.

species s appears as a reactant, product, and modifier, respectively. Similarly, $\mathbf{S_r^r}$, $\mathbf{S_r^p}$, and $\mathbf{S_r^m}$, are the sets of species that appear in reaction r as a reactant, product, and modifier, respectively. These sets are defined formally below:

$$\begin{aligned}
\mathbf{R_s^r} &= \{r \in \mathbf{R} \mid \mathbf{E}(s,r) > \mathbf{E}(r,s)\},\\
\mathbf{R_s^p} &= \{r \in \mathbf{R} \mid \mathbf{E}(r,s) > \mathbf{E}(s,r)\},\\
\mathbf{R_s^m} &= \{r \in \mathbf{R} \mid \mathbf{E}(s,r) > 0 \wedge \mathbf{E}(s,r) = \mathbf{E}(r,s)\},\\
\mathbf{S_r^r} &= \{s \in \mathbf{S} \mid \mathbf{E}(s,r) > \mathbf{E}(r,s)\},\\
\mathbf{S_r^p} &= \{s \in \mathbf{S} \mid \mathbf{E}(r,s) > \mathbf{E}(s,r)\},\\
\mathbf{S_r^m} &= \{s \in \mathbf{S} \mid \mathbf{E}(s,r) > 0 \wedge \mathbf{E}(s,r) = \mathbf{E}(r,s)\}.
\end{aligned}$$

Note that in these definitions a species is considered a reactant of a reaction only if the net change of the state of that species by a reaction is negative. Similarly, it is a product only if the net change of the state of that species by a reaction is positive. Finally, it is considered a modifier if it is used in a reaction but the state of that species is not changed by that reaction. Our example includes the following nonempty sets:

$$\begin{aligned}
\mathbf{R_{s_1}^r} &= \{r_1\},\\
\mathbf{R_{s_3}^p} &= \{r_1\},\\
\mathbf{R_{s_2}^m} &= \{r_1\},\\
\mathbf{S_{r_1}^r} &= \{s_1\},\\
\mathbf{S_{r_1}^p} &= \{s_3\},\\
\mathbf{S_{r_1}^m} &= \{s_2\}.
\end{aligned}$$

Sections 2.2 to 2.5 describe general abstraction methods that substantially reduce the model complexity. Section 2.6 describes how these abstraction methods can be combined together and applied to reduce the complexity of REB models. Finally, Section 2.7 shows the improvements gained by our approach.

2.2 Michaelis-Menten Approximation

Consider the following elementary enzymatic reactions where E is an enzyme, S is a substrate, C is a complex form of E and S, and P is a product.

$$\mathrm{E} + \mathrm{S} \underset{k_{-1}}{\overset{k_1}{\rightleftarrows}} \mathrm{C} \xrightarrow{k_2} \mathrm{E} + \mathrm{P} \tag{1}$$

If the complex C dissociates into E and S much faster than it is converted into the product P (i.e., $k_{-1} >> k_2$), then the substrate can be assumed to be in rapid equilibrium with the complex. In this case, these reactions can be transformed to the following *Michaelis-Menten* (MM) form:

$$\mathrm{S} \xrightarrow{\frac{k_2 E_{tot} K_1 [S]}{1+K_1[S]}} \mathrm{P}\ , \tag{2}$$

where E_{tot} is the total concentration of E and C, and K_1.

Using this approximation, a REB model can be reduced by searching for patterns matching the reaction given by Expression 1 above with rate laws that satisfy the conditions. When such a pattern is found and the species matched to C only appears in the reactions in this pattern, all the reactions in the pattern are removed from the model along with the complex C. Finally, a new reaction is introduced with the form given in Expression 2 above.

The rapid equilibrium approximation has two advantages: (1) the state space of the process is reduced since intermediate species are eliminated, (2) simulation time is reduced by removing fast reactions. Figure 2 shows a graphical representation of a more complex competitive enzymatic reaction from Arkin et al.'s phage λ model [3]. In Figure 2(a), proteins *CII* and *CIII* compete for binding to protease *P1*—producing complexes *P1·CII* and *P1·CIII*, respectively. (Note that each reaction node that is connected to a species with a double arrow is a shorthand way of showing a reversible reaction.) In this complex form, this protease acts to degrade *CII* and *CIII*. An extended form of the rapid equilibrium approximation can be applied to this network to remove this protease, its complex forms, and the reactions that form these complexes. Importantly, this also clarifies the essential biological meaning of the process by removing intermediate steps, which may otherwise obscure the functional logic of the mechanism. The resulting abstracted reaction model is shown in Figure 2(b).

The algorithms shown in Figure 3 implement the rapid equilibrium approximation for multiple alternative substrate systems which is a generalization of the complete characterization of enzyme-substrate and enzyme-substrate-competitor reactions [24]. First, Algorithm 1 considers each species, s, as a potential enzyme. Each species is checked using Algorithm 2. If s is an interesting species or does not occur as a reactant in any reaction, then s is not considered further (line 2). Otherwise, each reaction, r_1, in which s is a reactant is considered in turn. If r_1 is not reversible, does not have two reactants, or does not have a rate law of the right form (i.e., $k_f[s][s_1] - k_r[s_c]$), then again s is not considered further (line 4). Reaction r_1 combines s and s_1 into a complex, s_c. If the initial molecule count of this complex is not 0, s_c is an interesting species, s_c does not occur as a reactant or product in exactly one reaction, or occurs as a modifier in any, then again this approximation is terminated for s (lines 5 and 6). The reaction r_2 converts s_c into a product and releases the enzyme s. If this reaction is reversible, does not have exactly one reactant and no modifiers, does not have s as a product, has more than two products, or does not have a rate law of the form, $k_2[s_c]$, then s is not considered further (lines 7-9). Finally, it checks the validity of the rapid equilibrium assumption by comparing the ratio of the product dissociation rate constant and the substrate dissociation rate constant to the predefined threshold constant T_1 (line 10). For each reaction, a configuration is formed that includes the substrate s_1, complex s_c, equilibrium constant k_f/k_r, production rate k_2, complex forming reaction r_1, and product forming reaction r_2 (line 11). If Algorithm 2 terminates successfully (i.e., returns a nonempty set of configurations, $\mathbf{C}$), then Algorithm 3 is called to apply the transformation to the REB model. First,

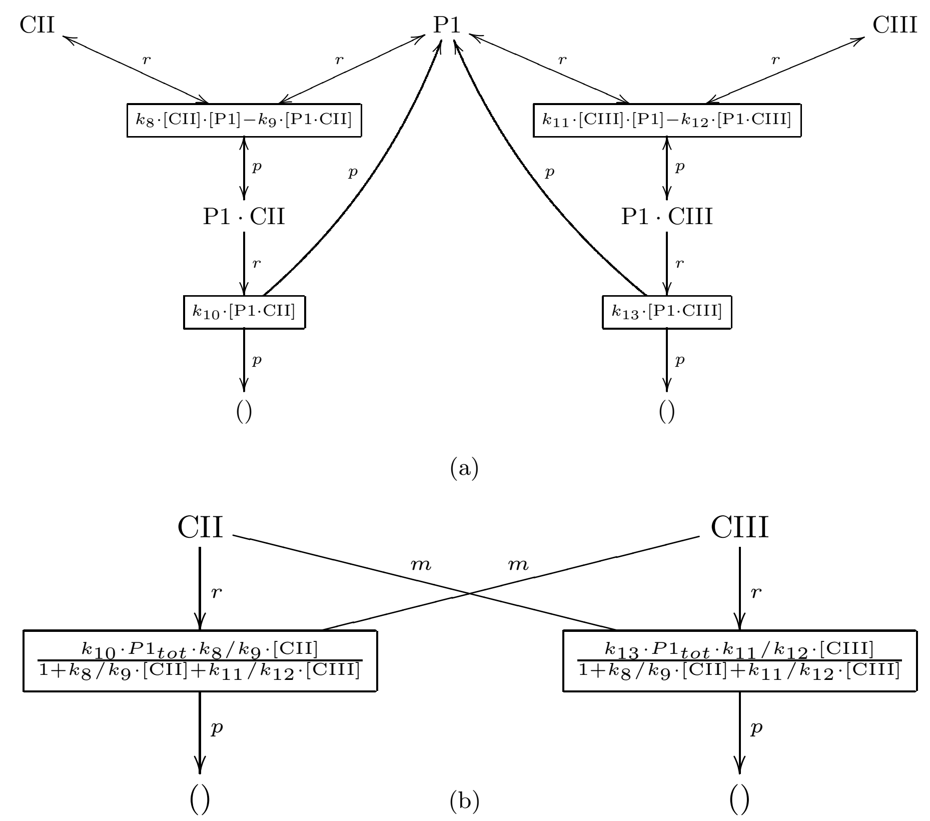

CII
P1
CIII
r
p
m
$k_8 \cdot [\mathrm{CII}] \cdot [\mathrm{P1}] - k_9 \cdot [\mathrm{P1 \cdot CII}]$
$k_{11} \cdot [\mathrm{CIII}] \cdot [\mathrm{P1}] - k_{12} \cdot [\mathrm{P1 \cdot CIII}]$
P1 · CII
P1 · CIII
$k_{10} \cdot [\mathrm{P1 \cdot CII}]$
$k_{13} \cdot [\mathrm{P1 \cdot CIII}]$
()
(a)
$\frac{k_{10} \cdot P1_{tot} \cdot k_8 / k_9 \cdot [\mathrm{CII}]}{1 + k_8/k_9 \cdot [\mathrm{CII}] + k_{11}/k_{12} \cdot [\mathrm{CIII}]}$
$\frac{k_{13} \cdot P1_{tot} \cdot k_{11} / k_{12} \cdot [\mathrm{CIII}]}{1 + k_8/k_9 \cdot [\mathrm{CII}] + k_{11}/k_{12} \cdot [\mathrm{CIII}]}$
(b)

Fig. 2. Rapid equilibrium approximation: (a) original model, and (b) abstracted model

it loops through the set of configurations to form an expression that is used in the denominator in each new rate law as well as forming a list of all the substrates that bind to the enzyme s (lines 1-6). Next, for each configuration $(s_1, s_c, K_1, k_2, r_1, r_2)$, it makes the substrate s_1 a reactant for r_2, makes all other substrates modifiers for r_2, creates a new rate law for r_2, and removes species s_c and reaction r_1 (lines 7-13). Finally, this algorithm removes the enzyme, s (line 14).

When k_{-1} is not much greater than k_2, the rapid equilibrium approximation cannot be applied. In such cases, however, if the total concentration of the enzyme is much less than the sum of the initial concentration of the substrate and the MM constant (i.e., $[E]_0 << [S]_0 + K_M$ where $K_M = (k_{-1} + k_2)/k_1$), then the *standard quasi-steady-state approximation* can be used instead to transform an enzymatic one-substrate reaction to the MM form with a somewhat more complex kinetic rate law (i.e., K_1 in Expression 2 is replaced with ${K_M}^{-1}$).

Algorithm 1. *Rapid equilibrium approximation*
Model RapidEqApprox(Model M)

```
 1: for all $s \in \mathbf{S}$ do
 2:     $\mathbf{C} \leftarrow RapidEqConditionSatisfied(M, s)$
 3:     if $\mathbf{C} \neq \emptyset$ then $M \leftarrow RapidEqTransform(M, s, \mathbf{C})$
 4: end for
 5: return $M$
```

Algorithm 2. *Check the conditions for rapid equilibrium approximation*
Configs RapidEqConditionSatisfied(Model M, Species s)

```
 1: $\mathbf{C} \leftarrow \emptyset$
 2: if $(s \in \mathbf{S_i}) \vee (|\mathbf{R^r_s}| = 0)$ then return $\emptyset$
 3: for all $r_1 \in \mathbf{R^r_s}$ do
 4:     if $(r_1 \notin \mathbf{R_{rev}}) \vee (|\mathbf{S^r_{r_1}}| \neq 2) \vee (\mathbf{K}(r_1) \neq$ “$k_f[s][s_1] - k_r[s_c]$”$)$ then return $\emptyset$
 5:     if $([s_c]_0 \neq 0) \vee (s_c \in \mathbf{S_i})$ then return $\emptyset$
 6:     if $(|\mathbf{R^r_{s_c}}| \neq 1) \vee (|\mathbf{R^m_{s_c}}| \neq 0) \vee (\mathbf{R^p_{s_c}}| \neq 1)$ then return $\emptyset$
 7:     $\{r_2\} \leftarrow \mathbf{R^r_{s_c}}$
 8:     if $(r_2 \in \mathbf{R_{rev}}) \vee (|\mathbf{S^r_{r_2}}| \neq 1) \vee (|\mathbf{S^m_{r_2}}| \neq 0)$ then return $\emptyset$
 9:     if $(s \notin \mathbf{S^p_{r_2}}) \vee (|\mathbf{S^p_{r_2}}| \notin \{1, 2\}) \vee (\mathbf{K}(r_2) \neq k_2[s_c])$ then return $\emptyset$
10:     if $k_2/k_r > T_1$ then return $\emptyset$
11:     $\mathbf{C} \leftarrow \mathbf{C} \cup \{(s_1, s_c, k_f/k_r, k_2, r_1, r_2)\}$
12: end for
13: return $\mathbf{C}$
```

Algorithm 3. *Perform the rapid equilibrium approximation*
Model RapidEqTransform(Model M, Species s, Configs **C***)*

```
 1: kinetic law expression $Z \leftarrow 1$
 2: $\mathbf{L} \leftarrow \emptyset$
 3: for all $(s_1, s_c, K_1, k_2, r_1, r_2) \in \mathbf{C}$ do
 4:     $Z \leftarrow Z + (K_1 * [s_1])$
 5:     $\mathbf{L} \leftarrow \mathbf{L} \cup \{s_1\}$
 6: end for
 7: for all $(s_1, s_c, K_1, k_2, r_1, r_2) \in \mathbf{C}$ do
 8:     $M \leftarrow addReactant(M, s_1, r_2, \mathbf{E}(s_1, r_1))$
 9:     $\forall m \in \mathbf{L} \setminus \{s_1\}.\ M \leftarrow addModifier(M, m, r_2)$
10:     $\mathbf{K}(r_2) \leftarrow (k_2 * [s]_0 * k_e * [s_1])/Z$
11:     $M \leftarrow removeSpecies(M, s_c)$
12:     $M \leftarrow removeReaction(M, r_1)$
13: end for
14: $M \leftarrow removeSpecies(M, s)$
15: return $M$
```

Fig. 3. Algorithms to perform rapid equilibrium approximation

2.3 Operator Site Reduction

REB models of genetic networks generally include multiple operator sites which transcription factors may occupy. It is often the case that the rates at which transcription factors bind and unbind to these operator sites are rapid with respect to the rate of *open complex formation* (i.e., initiation of transcription). It is also typically the case that the number of operator sites is much smaller than the number of *RNA polymerase* (RNAP) and transcription factor molecules. Therefore, a method similar to rapid equilibrium approximation called *operator site reduction* can be used to systematically merge reactions and remove operator sites and their complexes from REB models. Note that this method may also be applicable to other molecular scaffolding systems such as those found in signal transduction networks.

The first step in this transformation is to identify operators within the REB model. This is done by assuming that an operator is a species small in number that is neither produced nor degraded. Suppose our algorithm has identified an operator O, and there are $N + 1$ configurations in which transcription factors and RNAP can bind to it. Let O_i, K_i, and X_i with $i \in [1, N]$, be the i-th bound complex of the operator O, the equilibrium constant for forming this configuration, and the product of the concentrations of the substrates for each component of the complex in this configuration, respectively. Let O_0 be the operator in free form (i.e., not bound to anything). Let C_i with $i \in [0, N]$ be each of the operator configurations. Then, assuming rapid equilibrium, the probability of this operator being in each configuration is:

$$Pr(C_i) = \begin{cases} \frac{1}{Z} & \text{if } i = 0 \\ \frac{K_i \cdot X_i}{Z} & \text{if } 1 \leq i \leq N \end{cases}$$

where $Z = 1 + \sum_{j=1}^{N} K_j \cdot X_j$. This probability is the same as the equilibrium statistical thermodynamic model when $K_i = exp(\Delta G_i/RT)$ where ΔG_i is the relative free energies for the i-th configuration, R is the gas constant, and T is the absolute temperature [25]. Assuming that $O_{tot} = [O_0]_0$, then $[O_i] = Pr(C_i)O_{tot}$ is the fraction of operators in the i-th configuration.

Figure 4(a) shows the graphical representation of a portion of a REB model for the P_{RE} promoter from the phage λ decision network [3]. The top three reactions involve the binding of *RNAP* and *CII* to P_{RE} and the bottom two reactions result in the production of 10 molecules of the protein *CI*. In this example, there are 4 configurations of the operator, namely, *PRE*, *PRE·RNAP*, *PRE·CII·RNAP*, and *PRE·CII*. Assuming that the operator-binding and unbinding rates are much faster than those of open complex formation, our method can apply operator site reduction. Figure 4(b) is the result of applying this abstraction method to Figure 4(a). The result has only three species and two reactions. The transformed model represents the probability of P_{RE} being in a configuration that results in production of *CI* instead of modeling every binding and unbinding of transcription factors and *RNAP* to the promoter precisely. After performing operator site reduction on all of the operators, the resulting two reactions are found to be structurally similar and can be further combined into a single reaction by applying our *similar reaction combination* method. Also, after performing operator site reduction on all of the operators, *RNAP* only appears as a modifier in every reaction. In this case, another abstraction method—known as *modifier constant propagation*—can be applied. Namely, in each kinetic law

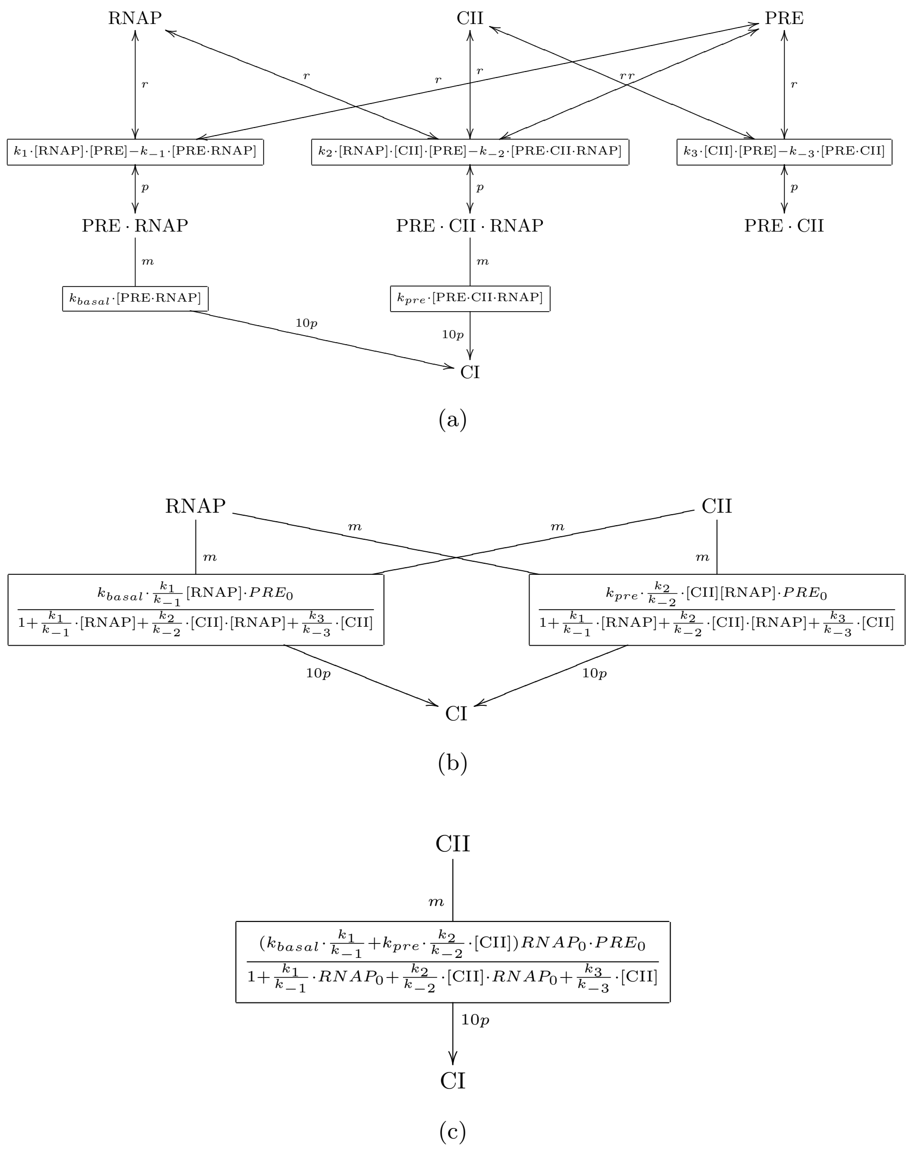

RNAP
CII
PRE
r
$k_1 \cdot [\mathrm{RNAP}] \cdot [\mathrm{PRE}] - k_{-1} \cdot [\mathrm{PRE \cdot RNAP}]$
$k_2 \cdot [\mathrm{RNAP}] \cdot [\mathrm{CII}] \cdot [\mathrm{PRE}] - k_{-2} \cdot [\mathrm{PRE \cdot CII \cdot RNAP}]$
$k_3 \cdot [\mathrm{CII}] \cdot [\mathrm{PRE}] - k_{-3} \cdot [\mathrm{PRE \cdot CII}]$
p
PRE · RNAP
PRE · CII · RNAP
PRE · CII
m
$k_{basal} \cdot [\mathrm{PRE \cdot RNAP}]$
$k_{pre} \cdot [\mathrm{PRE \cdot CII \cdot RNAP}]$
10p
CI
(a)
$\frac{k_{basal} \cdot \frac{k_1}{k_{-1}} [\mathrm{RNAP}] \cdot PRE_0}{1 + \frac{k_1}{k_{-1}} \cdot [\mathrm{RNAP}] + \frac{k_2}{k_{-2}} \cdot [\mathrm{CII}] \cdot [\mathrm{RNAP}] + \frac{k_3}{k_{-3}} \cdot [\mathrm{CII}]}$
$\frac{k_{pre} \cdot \frac{k_2}{k_{-2}} \cdot [\mathrm{CII}][\mathrm{RNAP}] \cdot PRE_0}{1 + \frac{k_1}{k_{-1}} \cdot [\mathrm{RNAP}] + \frac{k_2}{k_{-2}} \cdot [\mathrm{CII}] \cdot [\mathrm{RNAP}] + \frac{k_3}{k_{-3}} \cdot [\mathrm{CII}]}$
(b)
$\frac{(k_{basal} \cdot \frac{k_1}{k_{-1}} + k_{pre} \cdot \frac{k_2}{k_{-2}} \cdot [\mathrm{CII}]) RNAP_0 \cdot PRE_0}{1 + \frac{k_1}{k_{-1}} \cdot RNAP_0 + \frac{k_2}{k_{-2}} \cdot [\mathrm{CII}] \cdot RNAP_0 + \frac{k_3}{k_{-3}} \cdot [\mathrm{CII}]}$
(c)

Fig. 4. Operator site reduction: (a) original model, (b) abstracted model, and (c) after similar reaction combination and modifier constant propagation

in which $[RNAP]$ appears, it can be replaced with the constant, $RNAP_0$, which is the initial molecule count of *RNAP*. The result after both of these steps is shown in Figure 4(c).

The algorithms shown in Figure 5 implement operator site reduction. First, Algorithm 4 considers each species, s, as a potential operator site. Each species

Algorithm 4. *Operator site reduction*
Model OpSiteReduction(Model M)

```
 1: for all $s \in \mathbf{S}$ do
 2:    $\mathbf{C} \leftarrow OpSiteConditionSatisfied(M, s)$
 3:    if $\mathbf{C} \neq \emptyset$ then $M \leftarrow OpSiteTransform(M, s, \mathbf{C})$
 4: end for
 5: return $M$
```

Algorithm 5. *Check the conditions for operator site reduction*
Configs OpSiteConditionSatisfied(Model M, Species s)

```
 1: $\mathbf{C} \leftarrow \emptyset$
 2: if $[s]_0 > maxOperatorThreshold$ then return $\emptyset$
 3: if $(s \in \mathbf{S_i}) \vee (|\mathbf{R^p_s}| \neq 0)$ then return $\emptyset$
 4: for all $r_1 \in \mathbf{R^r_s}$ do
 5:    if $(r_1 \notin \mathbf{R_{rev}}) \vee (|\mathbf{S^r_{r_1}}| < 2) \vee (|\mathbf{S^p_{r_1}}| \neq 1)$ then return $\emptyset$
 6:    if $(\mathbf{K}(r_1) \neq$ “$k_f \prod_{s' \in \mathbf{S^r_{r_1}}} [s']^{\mathbf{E}(s', r_1)} - k_r[s_c]$”$)$ then return $\emptyset$
 7:    if $(s_c \in \mathbf{S_i}) \vee (|\mathbf{R^p_{s_c}}| \neq 1) \vee (|\mathbf{R^r_{s_c}}| \neq 0)$ then return $\emptyset$
 8:    for all $r_2 \in \mathbf{R^m_{s_c}}$ do
 9:       if $(|\mathbf{S^r_{r_2}}| \neq 0) \vee (|\mathbf{S^m_{r_2}}| \neq 1) \vee (|\mathbf{S^p_{r_2}}| \neq 1) \vee (\mathbf{K}(r_2) \neq k_2[s_c])$ then return $\emptyset$
10:    end for
11:    $e \leftarrow (k_f/k_r) \prod_{s' \in (\mathbf{S^r_{r_1}} \setminus \{s\})} [s']^{\mathbf{E}(s', r_1)}$
12:    $\mathbf{C} \leftarrow \mathbf{C} \cup \{(s_c, e, r_1)\}$
13: end for
14: return $\mathbf{C}$
```

Algorithm 6. *Perform transformation for operator site reduction*
Model OpSiteTransform(Model M, Species s, Configs **C***)*

```
 1: kinetic law expression $Z \leftarrow 1$
 2: $\mathbf{L} \leftarrow \emptyset$
 3: for all $(s_c, e, r_1) \in \mathbf{C}$ do
 4:    $Z \leftarrow Z + e$
 5:    $\mathbf{L} \leftarrow \mathbf{L} \cup \mathbf{S^r_{r_1}}$
 6: end for
 7: for all $(s_c, e, r_1) \in \mathbf{C}$ do
 8:    for all $r_2 \in \mathbf{R^m_{s_c}}$ with $\mathbf{K}(r_2) = k_2[s_c]$ do
 9:       $\forall m \in \mathbf{L}.\ M \leftarrow addModifier(M, m, r_2)$
10:       $\mathbf{K}(r_2) \leftarrow (k_2 * [s]_0 * e)/Z$
11:    end for
12:    $M \leftarrow removeSpecies(M, s_c)$
13:    $M \leftarrow removeReaction(M, r_1)$
14: end for
15: $M \leftarrow removeSpecies(M, s)$
16: return $M$
```

Fig. 5. Algorithms for operator site reduction

is checked using Algorithm 5. First, it is assumed that the molecule count of operator sites is small, so if s has an initial molecule count greater than a given threshold, then it is assumed not to be an operator site (line 2). Next, if s is an interesting species or occurs as a product in any reaction, then s is not considered further (line 3). Otherwise, each reaction, r_1, in which s is a reactant is considered in turn. If r_1 is not reversible, has less than two reactants, does not have exactly one product, or does not have a rate law of the right form, then again s is not considered further (lines 5 and 6). Each reaction, r_1, combines the potential operator site, s, with RNAP and/or transcription factors forming a complex, s_c. If s_c is an interesting species, s_c does not occur as a product in exactly one reaction, or occurs as a reactant in any, then again this approximation is terminated for s (line 7). The species s_c may appear as a modifier in any number of reactions that result in the transcription and translation of proteins. Each of these reactions, r_2, is checked that it has no reactants, only one modifier, only one product, and a rate law of the right form (lines 8-10). For each complex, s_c, a configuration is formed that includes the complex s_c, an equilibrium expression for this configuration, and the complex forming reaction r_1 (lines 11 and 12). If Algorithm 5 terminates successfully, then Algorithm 6 is called to apply the transformation to the REB model. This algorithm is very similar to the one for rapid equilibrium. Again, it loops through the set of configurations to form an expression that is used in the denominator in each new rate law as well as forming a list of all the transcription factors that bind to the operator site s (lines 1-6). Next, it considers each configuration, (s_c, e, r_1). For each reaction r_2 in which s_c appears as a modifier, it adds all the transcription factors as modifiers and creates a new rate law for r_2 (lines 8-11). It then removes species s_c and reaction r_1 from the model (lines 12-13). Finally, at the end, this algorithm removes the operator site, s (line 15).

2.4 Dimerization Reduction

Dimerization is another type of reaction, which often involves only regulatory molecules and could thus frequently proceed very rapidly compared to the rate of transcription initiation. Therefore, it might also be useful to abstract away these reactions whenever possible by using a version of rapid-equilibrium constraints. The dimerization reduction method is used to express dimer and monomer forms of species in terms of their total concentration as follows [26]. Let us consider a species s_m that forms a dimer s_d. If $[s_t]$ is the total number of the monomer molecules, then it is defined as follows:

$$[s_t] = [s_m] + 2[s_d]. \tag{3}$$

Let K_e be the equilibrium constant for dimerization (i.e., $K_e = k_+/k_-$), then, by assuming s_m and s_d are in rapid equilibrium, we have

$$[s_d] = K_e[s_m]^2. \tag{4}$$

Using Equations 3 and 4, we can derive the following equation:

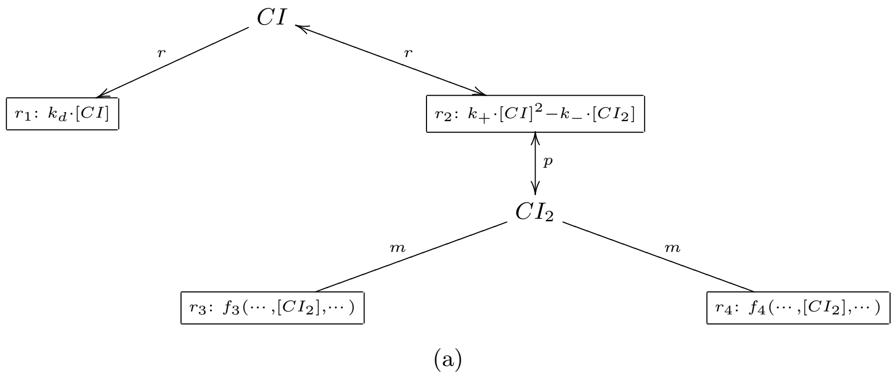
$$K_e[s_t]^2 - (4K_e[s_t] + 1)[s_d] + 4K_e[s_d]^2 = 0. \tag{5}$$

Solving Equation 5, we can express $[s_m]$ and $[s_d]$ in terms of $[s_t]$ as follows:

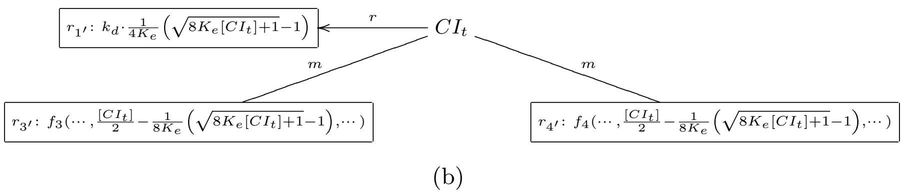
$$[s_m] = \frac{1}{4K_e}\left(\sqrt{8K_e[s_t]+1}-1\right), \tag{6}$$

$$[s_d] = \frac{[s_t]}{2} - \frac{1}{8K_e}\left(\sqrt{8K_e[s_t]+1}-1\right). \tag{7}$$

As an example, consider the reactions for the species CI from the phage λ decision network shown in Figure 6(a). This species can only effectively degrade in the monomer form (reaction r_1), but it is transcriptionally active (reactions r_3 and r_4) only as a dimer (reaction r_2). Using Equations 6 and 7, the reactions r_1, r_3, and r_4 can be transformed to $r_{1'}$, $r_{3'}$, and $r_{4'}$, respectively, with kinetic laws that are now all expressed in terms of total amount of CI as shown in Figure 6(b). Note that the dimerization reaction r_2 is eliminated completely.



CI
r
r
$r_1\colon k_d\cdot[CI]$
$r_2\colon k_+\cdot[CI]^2-k_-\cdot[CI_2]$
p
CI_2
m
m
$r_3\colon f_3(\cdots,[CI_2],\cdots)$
$r_4\colon f_4(\cdots,[CI_2],\cdots)$
(a)



$r_{1'}\colon k_d\cdot\frac{1}{4K_e}\left(\sqrt{8K_e[CI_t]+1}-1\right)$
r
CI_t
m
m
$r_{3'}\colon f_3(\cdots,\frac{[CI_t]}{2}-\frac{1}{8K_e}\left(\sqrt{8K_e[CI_t]+1}-1\right),\cdots)$
$r_{4'}\colon f_4(\cdots,\frac{[CI_t]}{2}-\frac{1}{8K_e}\left(\sqrt{8K_e[CI_t]+1}-1\right),\cdots)$
(b)

Fig. 6. Dimerization reduction: (a) original model, and (b) abstracted model

The algorithms to perform dimerization reduction are shown in Figure 7. First, Algorithm 7 is used to identify a dimerization reaction. It checks each reaction, r, using Algorithm 8. A dimerization reaction must include exactly one reactant, one product, and no modifiers (line 1). It must also have a rate law of the right form (line 2). The dimerization reduction also requires that the monomer is never used as a modifier, and that there is only one reaction (this one) which produces the dimer (lines 3-4). If these conditions are met, a record is made of the monomer species s_m, dimer species s_d, and equilibrium constant

k_+/k_- (line 5). The transformation is performed by Algorithm 9. First, a new species s_t is introduced into the model with an initial concentration $[s_m]_0+2[s_d]_0$ (lines 1-2). Next, s_m is replaced by s_t in each reaction in which s_m is a reactant, and the rate law is updated as described in Equation 6 (lines 3-6). The dimer s_d is also replaced with s_t in the reactions that it appears as a reactant or modifier, and the rate law is updated using Equation 7 (lines 7-11). Finally, the species s_m, the species s_d, and reaction r are all removed from the model (lines 12-14).

Algorithm 7. *Dimerization reduction*
Model DimerReduction(Model M)

```
 1: for all $r \in \mathbf{R_{rev}}$ do
 2:    $C \leftarrow DimerConditionSatisfied(M, r)$
 3:    if $C \neq$ nil then $M \leftarrow DimerTransform(M, r, C)$
 4: end for
 5: return $M$
```

Algorithm 8. *Check the conditions for the dimerization reduction*
Record DimerConditionSatisfied(Model M, Reaction r)

```
 1: if $(|\mathbf{S_r^r}| \neq 1) \vee (|\mathbf{S_r^p}| \neq 1) \vee (|\mathbf{S_r^m}| \neq 0)$ then return nil
 2: if $(\mathbf{K}(r) \neq$ "$k_+[s_m]^2 - k_-[s_d]$") then return nil
 3: $\{s_m\} \leftarrow \mathbf{S_r^r}$ and $\{s_d\} \leftarrow \mathbf{S_r^p}$
 4: if $(|\mathbf{R_{s_m}^m}| \neq 0) \vee (|\mathbf{R_{s_d}^p}| \neq 1)$ then return nil
 5: return $\langle s_m, s_d, k_+/k_- \rangle$
```

Algorithm 9. *Perform the dimerization reduction transformation*
Model DimerTransform(Model M, Reaction r, Record $\langle s_m, s_d, K_e \rangle$)

```
 1: $M \leftarrow addSpecies(M, s_t)$
 2: $[s_t]_0 \leftarrow [s_m]_0 + 2[s_d]_0$
 3: for all $r' \in \mathbf{R_{s_m}^r}$ do
 4:    $M \leftarrow addReactant(M, s_t, r', E(s_m, r'))$
 5:    replace $[s_m]$ with $\frac{1}{4K_e}\left(\sqrt{8K_e[s_t]+1} - 1\right)$ in $\mathbf{K}(r')$
 6: end for
 7: for all $\forall r' \in (\mathbf{R_{s_d}^r} \cup \mathbf{R_{s_d}^m})$ do
 8:    if $r' \in \mathbf{R_{s_d}^r}$ then $M \leftarrow addReactant(M, s_t, r', E(s_d, r'))$
 9:    if $r' \in \mathbf{R_{s_d}^m}$ then $M \leftarrow addModifier(M, s_t, r')$
10:    replace $[s_d]$ with $\frac{[s_t]}{2} - \frac{1}{8K_e}\left(\sqrt{8K_e[s_t]+1} - 1\right)$ in $\mathbf{K}(r')$
11: end for
12: $M \leftarrow removeSpecies(M, s_m)$
13: $M \leftarrow removeSpecies(M, s_d)$
14: $M \leftarrow removeReaction(M, r)$
15: return $M$
```

Fig. 7. Algorithms to perform dimerization reduction

2.5 Irrelevant Node Elimination

In a large system, there may be species that do not have significant influence on the species of interest, $\mathbf{S_i}$. Even when all the species in the original model are coupled, after applying abstractions, a species may no longer influence the species of interest. In such cases, computational performance can be gained by removing such irrelevant species and reactions. *Irrelevant node elimination* performs a reachability analysis on the REB model and detects nodes that are not used to influence the species in $\mathbf{S_i}$. For example, in Figure 8(a), s_6 is the only species in $\mathbf{S_i}$. Therefore, the production and degradation reactions of s_6, r_3 and r_2, must be relevant. The reaction r_3 uses s_3 as a reactant and s_2 as a modifier, so these species are relevant too. Since s_2 is relevant, the degradation reaction of s_2, r_1, is also relevant. This reaction uses s_1 as a modifier, so s_1 is relevant. Using these deductions, *irrelevant nodes elimination* results in the reduced model shown in Figure 8(b).

Whereas the irrelevant node elimination guarantees that all the removed nodes are irrelevant to the species in $\mathbf{S_i}$ by statically analyzing the structure of the model, there may still be nodes in the transformed model that can be safely removed without any significant effect on the model. In such cases, a more extensive and expensive dynamic analysis such as sensitivity analysis [27,28] can be applied to further reduce the model complexity.

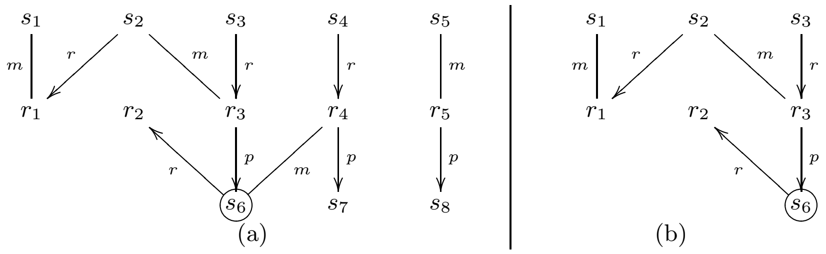


Fig. 8. Irrelevant node elimination: (a) original model and (b) after reduction

2.6 Top Level Abstraction Algorithm

The top level algorithm that combines all the abstraction methods described above is shown in Figure 9, and it is implemented within the tool `REB2SAC` [29]. The seven abstraction methods, irrelevant node elimination (line 3), modifier constant propagation (line 4), rapid equilibrium approximation (line 5), standard quasi-steady-state approximation (line 6), operator site reduction (line 7), similar reaction combination (line 8), and dimerization reduction (line 9), are applied iteratively until there is no change in the model. The irrelevant node elimination and the modifier constant propagation are applied first to reduce the complexity of the model without compromising accuracy. The rapid equilibrium approximation is applied before the standard quasi-steady-state approximation so that, whenever the model contains patterns that match the conditions for both methods, the former has precedence in order to reduce the complexity of the reaction rate laws. The similar reaction combination is applied right after the operator site reduction to immediately combine the structurally similar reactions that are often generated by operator site reduction. The dimerization reduction is placed after operator site reduction since an operator site with a dimer molecule as a transcription factor cannot be reduced otherwise.

Algorithm 10. *Top level abstraction algorithm*
Model AbstractionEngine(Model M)

```
 1: repeat
 2:     $M' \leftarrow M$
 3:     $M \leftarrow IrrelevantNodeElim(M)$
 4:     $M \leftarrow ModifierConstantProp(M)$
 5:     $M \leftarrow RapidEqApprox(M)$
 6:     $M \leftarrow StandardQSSA(M)$
 7:     $M \leftarrow OpSiteReduction(M)$
 8:     $M \leftarrow SimilarReactionComb(M)$
 9:     $M \leftarrow DimerReduction(M)$
10: until $M' = M$
11: return $M$
```

Fig. 9. Top level abstraction algorithm

2.7 Abstraction Results

As a case study, we built a REB model of the phage λ decision circuit based on the one described in [3]. Phage λ is a virus that infects *E. coli* cells. This virus replicates either by making new copies of itself within the *E. coli* and lysing the cell (i.e., *lysis*) or embedding its DNA into the cell's DNA and replicating through cell division (i.e., *lysogeny*). The goal of the analysis is to determine the probability that lysogeny is chosen under various conditions. For example, it has been shown experimentally that the probability of lysogeny increases as the *multiplicity of infection* (MOI)—the number of phages simultaneously infecting the same cell—increases [30]. The initial REB model includes 55 species and 69 reactions, and the set of interesting species, $\mathbf{S_i}$, includes *CI* and *Cro*. This model is available with the `REB2SAC` tool [29].

After applying the reaction-based abstraction methods as specified in Section 2.6, the REB model is reduced to only 5 species and 11 reactions as shown graphically in Figure 10. This figure shows the biological gene-regulatory network of the phage λ lysis/lysogeny decision circuit, and it is quite similar to the high-level hand-generated diagram in [3]. The structure of this graph, however, is automatically generated using abstractions from the low level model. This highlights the additional benefit of abstraction in facilitating a higher level view of the network being analyzed, since it removes the low level details such as intermediate species and reactions which involve them. This makes it easier to visualize crucial interactions including identification of the key species which

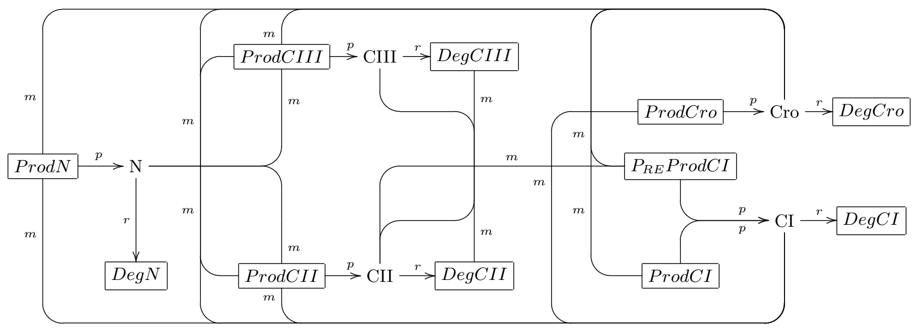


Fig. 10. Structure of the abstracted model of the phage λ developmental decision gene-regulatory pathway

ultimately inhibit and/or activate transcription. The `REB2SAC` tool can also output the abstracted model as SBML to allow it to be visualized or further analyzed using any SBML compliant tool. Finally, `REB2SAC` can output the model in presentation MathML to visualize complex rate laws using an XML/HTML browser.

Both the original model and the abstracted one are simulated for 10,000 runs using the same simulator, an optimized implementation of SSA within `REB2SAC`, on a 3GHz Pentium4 with 1GB of memory to estimate the probability of lysogeny with a reasonable statistical confidence as well as to measure the speed-up gained via abstractions. Each simulation is run for up to one cell cycle while tracking the number of molecules of *CI* and *Cro*. If the number of *CI* molecules exceeds 328 (i.e., 145 *CI* dimers) before the number of *Cro* molecules exceeds 133 (i.e., 55 *Cro* dimers), then the simulation run is said to result in lysogeny [3]. The simulations are run for MOIs ranging from 1 to 50. While the simulation of the original REB model takes 56.5 hours, the abstracted model only takes 9.8 hours, which is a speed-up of more than 5.7 times. Figure 11 shows the probability of lysogeny for MOIs from 0 to 10 for both the original REB model and the abstracted one. The results are nearly the same, yet with a substantial acceleration in runtime.

3 N-ary Transformation

To further improve the analysis time, the reduced REB model can be converted into a SAC model using the *n-ary transformation*. This section first describes the SAC model formally. Next, it describes each of the steps of n-ary transformation in turn. Then, it describes how the resulting SAC model can be analyzed using an iterative Markov chain method. Finally, it presents results using n-ary transformation on the phage λ model.

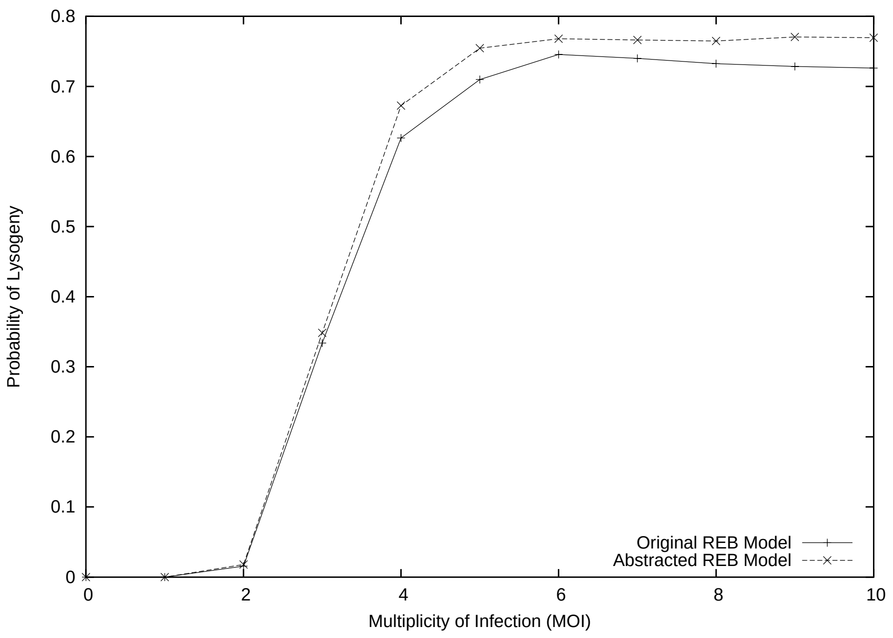


Fig. 11. Comparison of simulation results before and after abstraction where each data point has a margin of error of less than 0.01 with a 95 percent confidence

3.1 Stochastic Asynchronous Circuit Model

The SAC model is a continuous-time discrete-event system with which one can efficiently analyze the stochastic behavior of each species. So, instead of taking a computationally expensive approach of monitoring every single molecular state, the SAC model keeps track of aggregate levels of molecular counts. This is inspired by switch-like behaviors often seen in genetic regulatory networks.

Definition 2. *A SAC Model is specified using a 3-tuple* $\langle \mathbf{B}, \mathbf{b_0}, \mathbf{C} \rangle$ *where* $\mathbf{B} = \langle B_1, \ldots, B_n \rangle$ *is a vector of Boolean random variables, and thus* $\mathbf{B}(t)$ *represents the system state at time* t*. The initial state* $\mathbf{b_0}$ *contains the values of all the Boolean variables at time* 0*.* $\mathbf{C}$ *is the set of guarded commands that change the values of the Boolean variables. Each guarded command,* c_j*, has a form:*

$$G_j(\mathbf{b}) \xrightarrow{q_j} B_i := v_j$$

where the function $G_j(\mathbf{b}) : \{0,1\}^n \mapsto \{0,1\}$ *is the guard for* c_j *when the system state is* $\mathbf{b}$*,* q_j *is the transition rate for* c_j*, and* $v_j \in \{0,1\}$ *is the value assigned to* B_i *as a result of* c_j*.*

A guard is a conjunction of literals of the form $(B_i = v_j)$. Each guarded command, c_j, is required to change the state of some Boolean variable in $\mathbf{B}$. Therefore, if the state of B_i is changed to v_j by c_j, then the guard must include the term $(B_i = (1 - v_j))$. If the system state is $\mathbf{b}$ at time t (i.e., $\mathbf{B}(t) = \mathbf{b}$), c_j can be executed if its guard is satisfied (i.e., $G_j(\mathbf{b}) = 1$). The result of executing the guarded command in time step τ is that a new state is reached in which $B_i(t + \tau) = v_j$ and for all $k \neq i$, $B_k(t + \tau) = B_k(t)$. The probability that c_j is executed within the next infinitesimal The probability that, given the state is $\mathbf{b}$, c_j is executed within the next infinitesimal time step dt is:

$$P(c_j, dt \mid \mathbf{b}) = G_j(\mathbf{b}) \cdot q_j \cdot dt.$$

Consequently, the probability that no transition is taken within the next time step dt is:

$$1 - (\textstyle\sum_{l=1}^{|C|} G_l(\mathbf{b}) \cdot q_l \cdot dt).$$

A stochastic simulation of a process described using a SAC model begins in the state $\mathbf{b_0}$ at time 0 and selects either a guarded command to execute or no guarded commands to execute in a small time step Δt using the probability functions just defined. If a guarded command is executed at time t_1, then the system moves to a new state $\mathbf{B}(t_1) = \mathbf{b_1}$. It then recalculates all the transition probabilities, and continues until terminated.

This simulation process is inexact and inefficient since Δt is not a true infinitesimal yet for a sufficiently small Δt most simulation steps do not result in a state change. Therefore, the exact SSA which skips over the time steps where no state change occurs can be used instead [31,32] by using the expression $G_j(\mathbf{b}) \cdot q_j$ as the propensity function for the guarded command c_j when the system state is $\mathbf{b}$. In addition to stochastic simulation, a SAC model can be analyzed by constructing a homogeneous continuous-time Markov chain and applying an associated efficient analysis method [33]. This is the approach that is taken in this paper to analyze the phage λ decision circuit.

3.2 Reaction Splitization

The n-ary transformation requires the REB model to satisfy the property that all reactions should have either one reactant *or* one product, but not both. This is often the case after applying the abstractions described earlier as it is for the phage λ model. If this property does not hold, however, it can be made to hold using *reaction splitization*. One form of reaction splitization is called *single reactant single product reaction splitization*, which splits an irreversible reaction with a single reactant and a single product into an irreversible reaction with no reactant and a single product and an irreversible reaction with a single reactant and no product. In order to illustrate this transformation, consider the reaction shown in Figure 12(a) that converts species s_1 into species s_2 with a rate law

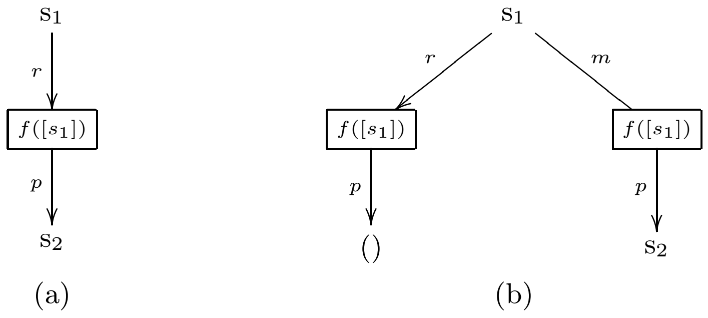


Fig. 12. Reaction splitization: (a) original reaction and (b) split-up reactions

$f([s_1])$. After splitization, this is transformed into the two reactions shown in Figure 12(b). This includes a degradation reaction for s_1 and a production reaction for s_2, with the same rate law. In addition, there is also *multiple reactants reaction splitization* to split a reaction with multiple reactants into multiple reactions with a single reactant, and *multiple products reaction splitization* that splits a reaction with multiple products into multiple reactions with a single product.

3.3 Boolean Variable Generation

Let X be a random variable representing the state of species s. Our method partitions the states of X into an ordered set $\mathbf{A} : (A_0, A_1, \ldots, A_n)$ such that, $\forall i.\ A_i = [\theta_i, \theta_{i+1})$ where $\theta_0 = 0$, and $\theta_{n+1} = \infty$. We call $A_0, \ldots, A_n$ *critical intervals*, and $\theta_0, \ldots, \theta_n$ *critical levels*. Depending on the nature of the application, these critical levels can be either specified by the user and taken to be model inputs—such as might be the case when our system is utilized by an expert already familiar with the *in situ* behavior of the underlying regulatory network—or estimated automatically from the kinetic rate laws as described next.

In order to identify the critical levels of species s, our method first automatically finds all reactions with kinetic rate laws that include a denominator term of the form $K[s]^n$. For each such reaction, one critical level of s is generated with the form $\sqrt[n]{a/(K - aK)}$ where a is an amplifier in the range $[0.5, 1.0)$ selected by the user. Figure 13(a) shows two reactions which have kinetic rate laws containing *CII* terms. Assuming that a equals 0.5, these two reactions imply the following four critical levels:

$$0, \frac{k_9}{k_8}, \frac{k_{-2}}{k_2 \cdot RNAP_0}, \text{ and } \frac{k_{-3}}{k_3}.$$

These levels come from the fact that θ_0 is by definition 0, the denominator of the left reaction rate law in Figure 13(a) has the term $k_8/k_9[CII]$, and the denominator of the right reaction rate law has two terms of this form, $k_2/k_{-2}[CII]RNAP_0$ and $k_3/k_{-3}[CII]$.

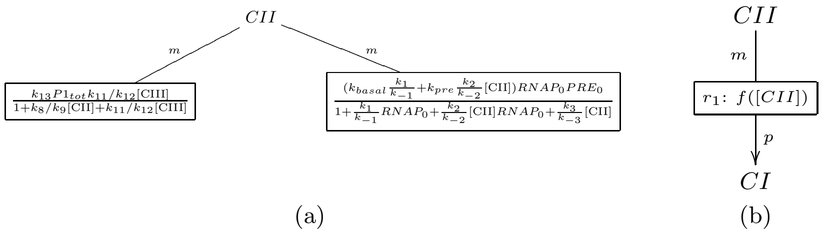

CII
m
m
$\frac{k_{13}P1_{tot}k_{11}/k_{12}[\mathrm{CIII}]}{1+k_8/k_9[\mathrm{CII}]+k_{11}/k_{12}[\mathrm{CIII}]}$
$\frac{(k_{basal}\frac{k_1}{k_{-1}}+k_{pre}\frac{k_2}{k_{-2}}[\mathrm{CII}])RNAP_0PRE_0}{1+\frac{k_1}{k_{-1}}RNAP_0+\frac{k_2}{k_{-2}}[\mathrm{CII}]RNAP_0+\frac{k_3}{k_{-3}}[\mathrm{CII}]}$
(a)
CII
m
r_1: $f([CII])$
p
CI
(b)

Fig. 13. (a) Critical level identification. (b) Production of *CI* with activator *CII*.

If there are $n+1$ critical levels, $\theta_0, \ldots, \theta_n$, for s, our method creates n Boolean variables $B_1 \ldots B_n$ with initial values $b_1 \ldots b_n$ in which $b_i = 1$ if $[s]_0 \geq \theta_i$ and 0 otherwise. Y is a discrete random variable that denotes the state of critical levels that species s is in. So, the relationship between Y and $B_1, \ldots, B_n$ is: $Y(t) = i$ iff $(\forall j \in [1, i].\ B_j(t) = 1) \wedge (\forall j \in [i+1, n].\ B_j(t) = 0)$.

3.4 Guarded Command Generation

The guard for a reaction is derived from the Boolean variables for the species used in that reaction. Suppose species *CII* is an activator in the reaction r_1 for the production of *CI* shown in Figure 13(b). Also, suppose that three critical levels are used for *CII* which are $(0, \theta_1^{CII}, \theta_2^{CII})$, and the critical levels for *CI* are $(0, \theta_1^{CI}, \theta_2^{CI})$, respectively. Since r_1 is a production reaction for species CI, the legal moves that Y_{CI} (the random variable for the levels of CI) can take are only two: $0 \rightarrow 1$ and $1 \rightarrow 2$. The guarded commands for r_1 are below:

$$\begin{aligned}
B_2^{CI} = 0 \wedge B_1^{CI} = 0 \wedge B_2^{CII} = 0 \wedge B_1^{CII} = 0 &\xrightarrow{q_1} B_1^{CI} := 1\\
B_2^{CI} = 0 \wedge B_1^{CI} = 0 \wedge B_2^{CII} = 0 \wedge B_1^{CII} = 1 &\xrightarrow{q_2} B_1^{CI} := 1\\
B_2^{CI} = 0 \wedge B_1^{CI} = 0 \wedge B_2^{CII} = 1 \wedge B_1^{CII} = 1 &\xrightarrow{q_3} B_1^{CI} := 1\\
B_2^{CI} = 0 \wedge B_1^{CI} = 1 \wedge B_2^{CII} = 0 \wedge B_1^{CII} = 0 &\xrightarrow{q_4} B_2^{CI} := 1\\
B_2^{CI} = 0 \wedge B_1^{CI} = 1 \wedge B_2^{CII} = 0 \wedge B_1^{CII} = 1 &\xrightarrow{q_5} B_2^{CI} := 1\\
B_2^{CI} = 0 \wedge B_1^{CI} = 1 \wedge B_2^{CII} = 1 \wedge B_1^{CII} = 1 &\xrightarrow{q_6} B_2^{CI} := 1
\end{aligned}$$

3.5 Transition Rate Generation

The final step to generate a SAC model is to assign a transition rate, q_i, to each guarded command. The random variable, X, which represents the state of s can be approximately expressed in terms of the Boolean variables by defining a discrete random variable, X', as follows:

$$X'(t) = (\theta_n - \theta_{n-1})B_n(t) + \cdots + (\theta_2 - \theta_1)B_2(t) + (\theta_1 - \theta_0)B_1(t),$$

and then approximating evolution of X by the average of X'. Taking the derivative with respect to the mean of $B_i(t)$ results in:

$$\frac{\partial X(t)}{\partial \langle B_i(t)\rangle} \approx (\theta_i - \theta_{i-1}).$$

Using this approximation, the time derivative of the mean of B_i is:

$$\frac{d\langle B_i(t)\rangle}{dt} = \frac{\partial \langle B_i(t)\rangle}{\partial X(t)}\frac{dX(t)}{dt} \approx \frac{1}{\theta_i - \theta_{i-1}}\frac{dX(t)}{dt}.$$

Notice $\langle B_i(t)\rangle$ is a continuous variable in the range $[0,1]$. By letting $\langle B_i(t)\rangle$ be the probability that $B_i = 1$ at t, our method finds the transition rate functions for B_i to move from 0 to 1 and from 1 to 0 from the rate laws of reactions that change the value of $[s]$. The transition rate function of a guarded command changing the value of B_i, which is generated from reaction r is:

$$f = \frac{E \cdot \mathbf{K}(r)}{\theta_i - \theta_{i-1}} \qquad \text{where } E = \begin{cases} \mathbf{E}(s,r) & \text{if } s \text{ is a reactant of } r \\ \mathbf{E}(r,s) & \text{if } s \text{ is a product of } r \end{cases}$$

Finally, our method must evaluate the transition rate functions with appropriate values. Consider reaction r with $\mathbf{K}(r)$ containing X. Given that the corresponding Y is i, our method uses θ_i as the value of X. For example, the transition rates of the guarded commands in Figure 13(b) are derived from $\mathbf{K}(r_1)$. Since the derived transition rate function is $f([CII])/(\theta_i^{CI} - \theta_{i-1}^{CI})$, the transition rates for the guarded commands for reaction r_1 are:

$$\begin{aligned} q_1 &= f(0)/\theta_1^{CI} \\ q_2 &= f(\theta_1^{CII})/\theta_1^{CI} \\ q_3 &= f(\theta_2^{CII})/\theta_1^{CI} \\ q_4 &= f(0)/(\theta_2^{CI} - \theta_1^{CI}) \\ q_5 &= f(\theta_1^{CII})/(\theta_2^{CI} - \theta_1^{CI}) \\ q_6 &= f(\theta_2^{CII})/(\theta_2^{CI} - \theta_1^{CI}) \end{aligned}$$

3.6 Markov Analysis

A SAC model can be efficiently analyzed using Markov chain analysis within the `ATACS` tool [20]. Beginning in the initial state, $\mathbf{b_0}$, a transition can occur to a new state, $\mathbf{b_1}$, by executing a guarded command which has its guard satisfied in $\mathbf{b_0}$. A depth-first-search can be used to generate a state graph containing all states reachable from $\mathbf{b_0}$. If there exists a state transition from state $\mathbf{b_i}$ to $\mathbf{b_j}$ due to the execution of the guarded command c_k, then this state transition can be annotated with its transition rate, q_k. The result of this annotation is that the state graph is now a continuous-time Markov chain. This can be analyzed by first converting it into its embedded Markov chain by normalizing each transition rate by the sum of all the rates leaving the state resulting in a transition probability. Finally, this embedded Markov chain can be analyzed to determine the stationary probability distribution using an efficient iterative method [33].

3.7 N-ary Transformation Results

The n-ary transformation is able to automatically convert our reduced REB model for the phage λ decision circuit into a SAC model. However, since the species *CI* and *Cro* influence many reactions, our automated analysis finds that 10 critical levels are needed for species *CI* and 10 are needed for species *Cro*. This is too many critical levels for the Markov chain analyzer within ATACS. Fortunately, many of these critical levels are very close together and can be combined with little loss in accuracy. Therefore, while we decided to use eight Boolean variables for species *CI* and three for *CII*, we only used one Boolean variable for each of the species *Cro*, *N*, and *CIII*.

We analyzed the SAC model using Markov chain analysis. The probability of lysogeny is calculated by summing the probability of states that reach the highest level of *CI*. We compare our results with both experimental data and previous simulations performed by Arkin et al. on a complete master equation model. The experimental results are from Kourilsky [30]. Since it was not practical to measure the number of phages that infect any given cell, Kourilsky measured the fraction of cells that commit to lysogeny versus *average phage input* (API) (i.e., the proportion of phages to *E. coli* within the population). Kourilsky performed experiments for both "starved" *E. coli* and those in a "well-fed" environment. He found that the fraction that commits to lysogeny increases with increasing API, and that this fraction increases by more than an order of magnitude in a starved environment over a well-fed environment.

To map simulated MOI data onto API data, Arkin et al. used a Poisson distribution of the phage infections over the populations:

$$P(M,A) = \frac{A^M}{M!}e^{-A}$$

$$F_{\text{lysogens}}(A) = \sum_M P(M,A) \cdot F(M)$$

where M is the MOI, A is the API, and $F(M)$ is the probability of lysogeny determined by Markov analysis. We also used this method to map our MOI data. The results are shown in Figure 14. The individual points represent experimental measurements while the lines represent simulation results. Both Arkin et al.'s simulation and our SAC model results track the starved data points reasonably well. Our SAC model results, however, are found in less than 7 minutes of computation time on a 3GHz Pentium4 with 1GB of memory. While modern computer technology and algorithmic improvements would greatly improve the simulation time of Arkin et al.'s model, these results would still take several hours to generate on a similar computer to ours. Another notable benefit of our SAC method is that it can also produce simulation results for the well-fed case in about 7 minutes. These results could likely not be generated even today using Arkin's master equation simulation method, since the number of simulation runs necessary is inversely proportional to the probability of lysogeny (i.e., about two orders of magnitude greater in the well-fed case than in the starved one).

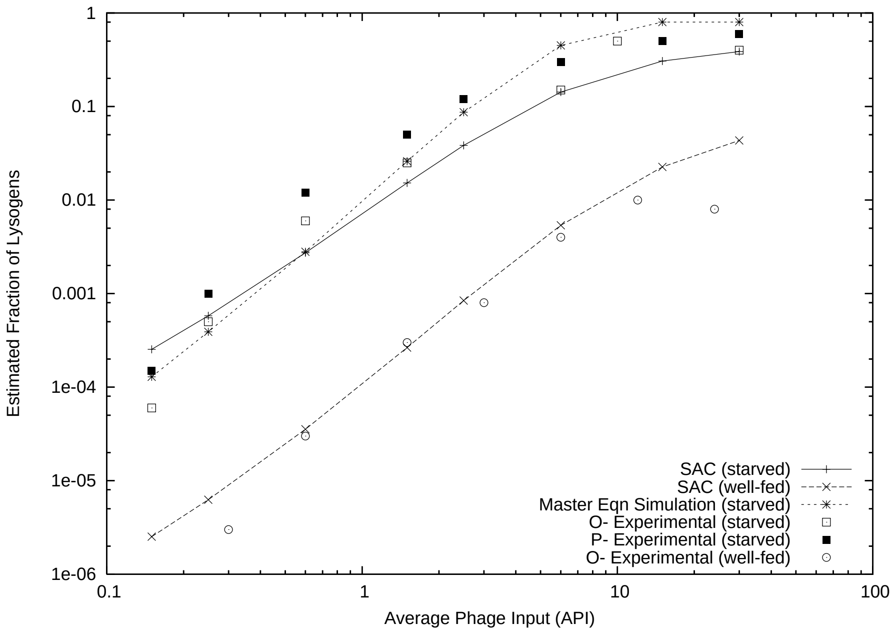


Fig. 14. Comparison of SAC results to experimental data

4 Conclusions

This paper presents a general methodology for systematically and automatically abstracting the complexities of large-scale biochemical reaction-based networks (REBs) to a reduced stochastic asynchronous circuit (SAC) representation. It significantly facilitates efficient non-deterministic analysis of such systems by substantially reducing the problem dimensionality in both reaction and molecular state spaces, thus potentially allowing for both simulation time acceleration and computability gains while facilitating a high-level view of the network. Furthermore, since our approach allows for multiple levels of abstraction, it is broadly applicable to a wide range of biological systems and their representations—from classical differential equation models to fully discrete and stochastic bio-molecular pathways—including the genetic regulatory networks upon which we have chosen to focus in this work.

As a case study, we have applied our method to the phage λ developmental decision pathway. The preliminary results are promising. Among other things, we are able to: (1) ascertain the internal self-consistency of our approach by successfully cross-validating each abstraction level output against the results of the full underlying discrete-stochastic model simulations; and (2) accurately estimate the biologically relevant (observable) pathway selection probabilities, which typically require substantial numbers of hours of computation time via the original REB representation, yet could be computed in only minutes using our SAC approach.

Future work includes the development of more abstraction methods, refinement of the critical level assignment algorithm, and integration of a more scalable Markov chain analyzer. We are also working on a tighter integration with other tools for modeling and analysis of (bio)chemical networks such as BioSPICE [23]. Finally, we are applying our abstraction methodology to efficient analysis of other systems—such as the *E. coli* Fim mechanism and *B. Subtilis* stress response network—that may benefit from our automated abstraction methodology due to, among others, their inherently stochastic behavior *in situ*.

Acknowledgments. The authors would like to thank David Dill (Stanford University), Satoru Miyano (University of Tokyo), Hiroaki Kitano (Sony Corporation), and Arkin's research group (University of California, Berkeley) for numerous helpful discussions.

References

1. Jong, H.D.: Modeling and simulation of genetic regulatory systems: A literature review. J. Comp. Biol. **9**(1) (2002) 67–103
2. Baldi, P., Hatfield, G.W.: DNA Microarrays and Gene Expression. Cambridge University Press (2002)
3. Arkin, A., Ross, J., McAdams, H.: Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected escherichia coli cells. Genetics **149** (1998) 1633–1648
4. Elowitz, M.B., Levine, A.J., Siggia, E.D., Swain, P.S.: Stochastic gene expression in a single cell. Science **297** (2002) 1183–1186
5. Rao, C.V., Wolf, D.M., Arkin, A.P.: Control, exploitation and tolerance of intracellular noise. Nature **420** (2002) 231–238
6. Samoilov, M., Plyasunov, S., Arkin, A.P.: Stochastic amplification and signaling in enzymatic futile cycles through noise-induced bistability with oscillations. Proceedings of the National Academy of Sciences US **102**(7) (2005) 2310–5
7. Raser, J.M., O'Shea, E.K.: Control of stochasticity in eukaryotic gene expression. Science **304** (2004) 1811–1814
8. Kierzek, A.M., Zaim, J., Zielenkiewicz, P.: The effect of transcription and translation initiation frequencies on the stochastic fluctuations in prokaryotic gene expression. J. Biol. Chem **276** (2001) 8165
9. Gillespie, D.T.: A rigorous derivation of the chemical master equation. Physica A **188** (1992) 404–425
10. Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics **22** (1976) 403–434
11. Turner, T.E., Schnell, S., Burrage, K.: Stochastic approaches for modelling in vivo reactions. Computational Biology **28** (2004)
12. Gibson, M., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. J. Phys. Chem. **A 104** (2000) 1876–1889

13. Gibson, M., Bruck, J.: An efficient algorithm for generating trajectories of stochastic gene regulation reactions. Technical report, California Institute of Technology (1998)
14. Gillespie, D.T.: Approximate accelerated stochastic simulation of chemically reacting systems. Journal of Chemical Physics **115**(4) (2001) 1716–1733
15. Rathinam, M., Cao, Y., Petzold, L., Gillespie, D.: Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. Journal of Chemical Physics **119** (2003) p12784–94
16. Gillespie, D., Petzold, L.: Improved leap-size selection for accelerated stochastic simulation. Journal of Chemical Physics **119** (2003)
17. Cao, Y., Gillespie, D., Petzold, L.: Avoiding negative populations in explicit tau leaping. Journal of Chemical Physics **123** (2005)
18. Rao, C.V., Arkin, A.P.: Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the gillespie algorithm. J. Phys. Chem. **118**(11) (2003)
19. Schnell, S., Maini, P.K.: A century of enzyme kinetics: Reliability of the k_m and v_{max} estimates. Comments on Theoretical Biology **8** (2003) 169–187
20. Myers, C.J., Belluomini, W., Killpack, K., Mercer, E., Peskin, E., Zheng, H.: Timed circuits: A new paradigm for high-speed design. (2001) 335–340
21. Berry, R.S., Rice, S.A., Ross, J.: Physical Chemistry (2nd Edition). Oxford University Press, New York (2000)
22. Systems Biology Workbench Development Group. (`http://www.sbw-sbml.org/`)
23. BioSPICE. (`http://www.biospice.org/`)
24. Schnell, S., Mendoza, C.: Enzyme kinetics of multiple alternative substrates. Journal of Mathematical Chemistry **27** (2000) 155–170
25. Ackers, G.K., Johnson, A.D., Shea, M.A.: Quantitative model for gene regulation by λ phage repressor. Proc. Natl. Acad. Sci. USA **79** (1982) 1129–1133
26. Santillán, M., Mackey, M.C.: Why the lysogenic state of phase λ is stable: A mathematical modeling approch. Biophysical Jounal **86** (2004)
27. Dacol, D., Rabitz, H.: Sensitivity analysis of stochastic kinetic models. J. Math. Phys. **25** (1984)
28. Gunawan, R., Cao, Y., Petzold, L., Doyle, F.J.: Sensitivity analysis of discrete stochastic systems. Biophysical Journal **88** (2005) 2530–2540
29. REB2SAC. (`http://www.async.ece.utah.edu/tools/`)
30. Kourilsky, P.: Lysogenization by bacteriophage lambda: I. multiple infection and the lysogenic response. Mol. Gen. Genet. **122** (1973) 183–195
31. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25) (1977) 2340–2361
32. Gillespie, D.T.: Markov Processes An Introduction for Physical Scientists. Academic Press, Inc. (1992)
33. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press (1994)

P Systems, a New Computational Modelling Tool for Systems Biology

Mario Jesús Pérez-Jiménez and Francisco José Romero-Campero*

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
{marper, fran}@us.es

Abstract. In this paper we present P systems as a reliable computational modelling tool for Systems Biology that takes into account the discrete character of the quantity of components of biological systems, the inherently randomness in biological phenomena and the key role played by membranes in the functioning of living cells. We will introduce two different strategies for the evolution of P systems, namely, *Multi-compartmental Gillespie's Algorithm* based on the well known Gillespie's Algorithm but running on more than one compartment; and *Deterministic Waiting Times Algorithm*, an *exact* deterministic method. In order to illustrate these two strategies we have modelled two biological systems: the EGFR Signalling Cascade and the Quorum Sensing System in the bacterium Vibrio Fischeri. Our simulations results show that for the former system a deterministic approach is valid whereas for the latter a stochastic approach like *Multi-compartmental Gillespie's Algorithm* is necessary.

1 Introduction

Membrane Computing is an emergent branch of Natural Computing introduced by G. Păun in [18]. Since then it has received important attention from the scientific community. In fact, Membrane Computing has been selected by the Institute for Scientific Information, USA, as a fast *Emerging Research Front* in Computer Science, and [17] was mentioned in [26] as a highly cited paper in October 2003.

This new model of computation starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called P systems. Roughly speaking, a P system consists of a cell-like membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules.

Most variants of membrane systems have been proved to be computationally complete, that is equivalent in power to Turing machines, and computationally

* To whom correspondence should be addressed.

efficient, that is able to solve computationally hard problems in polynomial time, see [28]. Although most research in P systems concentrates on computational powers, lately they have been used to model biological phenomena [2,3,5].

As P systems are inspired from the functioning of the living cell, it is natural to consider them as modelling tools for biological systems, within the framework of systems biology, being an alternative to more classical approaches like ODEs. Differential equations have been used successfully to model kinetics of conventional macroscopic coupled chemical reactions. Nevertheless there is an implicit assumption of continuously varying chemical concentration and deterministic dynamics. Two critical characteristics of this approach are that the number of molecules of each type in the reaction mix is large and that for each type of reaction in the system, the number of reactions is large within each observation interval, that is reaction rates are fast.

When concentrations of the reacting species are low and reaction rates are slow, which is frequently the case in genetic circuits, both of the previous presumptions are invalid and the deterministic continuous approach to chemical kinetics breaks down. Instead one has to recognise that the individual chemical reaction steps occur discretely and are separated by time intervals of random length.

In contrast to differential equations, P systems are an unconventional model of computation which takes into consideration the discrete character of the quantity of components and the inherently randomness in biological phenomena. Besides, the key feature of P systems is the so called membrane structure which represents the heterogeneity of the structural organisation of the cells, and where one can take into account the role played by membranes in the functioning of the system, for example the fluctuation effects caused by diffusion.

In this paper we will present P systems as a reliable tool for Systems Biology. We will discuss two different strategies for their evolution. The first strategy is based on the well known Gillespie's Algorithm but running on more than one compartment and so it will be called *Multi-compartmental Gillespie Algorithm*. The second one will be a deterministic strategy called *Deterministic Waiting Times* which consists in an *exact* deterministic method in the sense that we do not approximate infinitesimal intervals of time by Δt as it is the case in ODEs, but we will associate a waiting time, computed in a deterministic way, to each reaction and will use them to determine the order in which the reactions take place.

In order to illustrate these strategies two different biological systems will be modelled; the Epidermal Growth Factor Receptor Signalling Cascade and the Quorum Sensing System in the marine bacterium Vibrio Fischeri. The former system is known to play a key role in tumour cell proliferation, angiogenesis and metastasis, being a key biological target for the development of novel anticancer therapies. Whereas the latter phenomenon is one of the best known quorum sensing systems; it consists in a gene regulation system that allows an entire population of bacterial cells to communicate in order to regulate the expression of specific genes involved in the production of light in a coordinated way depending on the size of the population.

The paper is organised as follows. In the next section we introduce P systems. In section 3 two strategies for the evolution of P systems are discussed. A study of EGFR Signalling Cascade and of the Quorum Sensing System in Vibrio Fischeri are given in sections 4 and 5. Finally, conclusions are presented in the last section.

2 P Systems as a Computational Modelling Tool for Systems Biology

A P system is usually defined as a hierarchical arrangement of a number of membranes identifying a corresponding number of regions inside the system, and with these regions having associated a finite multiset of objects and a finite set of rules. In what follows we give a precise definition of the main ingredients of a P system.

Definition 1 (membrane structure). *A membrane structure is a hierarchical arrangement of membranes where all the membranes but one must be included in a unique main membrane, which defines the boundary of the system with respect to the external environment. This particular membrane is called* skin membrane*. The membrane structure can be represented formally, as a rooted tree, where the nodes are called membranes, the root is called skin, and the relationship of membrane being inside another one is represented by the relationship of the node being the descendent of another one.*

Informally we can represent a membrane structure using Venn diagrams.

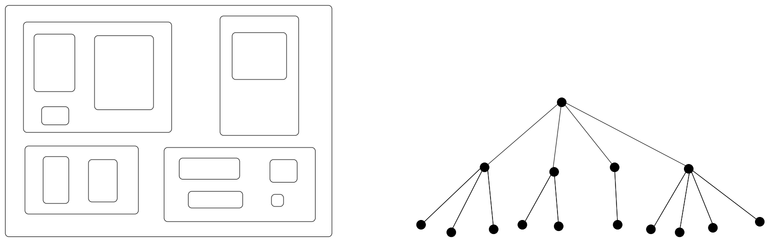


Fig. 1. A membrane structure represented using a Venn Diagram and a rooted tree

Rules of many different forms have been considered for P systems in order to encode the operation of modifying the objects inside the membrane, moving objects from one region to the other, dissolving, creating, dividing membranes etc. Here, in order to capture the features of most of these rules, we consider rules of the form:

$$u\,[\,v\,]_l \rightarrow u'\,[\,v'\,]_l \tag{1}$$

with u, v, u', v' some finite multisets of objects and l the label of a membrane. These rules are multiset rewriting rules that operate on both sides of the membranes, that is, a multiset u placed outside a membrane labelled by l and a multiset

v placed inside the same membrane can be simultaneously replaced by a multiset u' and a multiset v' respectively. In this way, we are able to capture in a concise way the features of both the communication rules and the transformation rules considered in [4]. Moreover, rules like (1) allow us to express any sort of interactions occurring at the membrane level, and, in particular, they are useful to model the binding of a signal molecule to its corresponding receptor that occurs at the cell-surface level.

We generalise this concept by associating to each rule a boolean predicate π expressing a generic property over the objects contained inside a membrane and the objects contained in the surrounding region or in one of the regions that exists inside the current one. Such a predicate π is meant to specify a condition that need to be satisfied to make the rule applicable inside a given membrane. Finally, we associate to each rule a finite set of attributes which are meant to capture the quantitative aspects that are often necessary to characterise the *reality* of the phenomenon to be modelled. The necessity of taking into account these quantitative aspects has been made clear in a few recent application of P systems to the modelling of biological systems [3,5,6,7].

Therefore, we introduce the following notion of program as the basic feature describing a generic process occurring inside a membrane.

Definition 2 (program). *Let O be an alphabet for the objects and let L be an alphabet of labels. A program is a construct*

$$\langle \pi >> u\,[\,v\,]_l \rightarrow u'\,[\,v'\,]_l, A\rangle$$

with $u, v, u', v' \in O^*$ some finite multisets of objects, π a generic boolean predicate, l is a label from L and A a finite set of attributes associated with the rule.

The predicate π is used to express a condition that needs to be satisfied in order to make the rule applicable inside a membrane. The set of attributes in A can instead be used to associate to each rule a kinetic constant [3,6], a probability [2,5], or a more general function returning the number of occurrences of the multisets u, v to be consumed and the number of occurrences of the multisets u', v' to be produced. As well as this, the attributes might be used to associate to a rule some *side-effect* in order to alter other properties of the membrane where the rule is applied.

Our notion of program bears some similarities with, and is somehow inspired by, the notion of attribute grammars used for syntax-directed language translation and automatic code generation [1], as well as the notion of parametric L systems augmented with C code used for modelling plant growth and development [13]. In a sense, our programs also resemble the concept of guarded commands for non-deterministic programming introduced by Dijkstra in 1975 which has then led to CCS, CSP and the modern theory of concurrent systems based on π-calculus [15].

Now, we can define a P system by simply associating a finite multiset of objects to each membrane in a given membrane structure and by considering a finite set of programs to make these objects evolve from one configuration to the other.

Definition 3 (P system). *A P system is a construct*

$$\Pi = (O, L, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$$

where:

- *O is a finite alphabet of symbols representing objects;*
- *L is a finite alphabet of symbols representing labels for the compartments;*
- *μ is a membrane structure containing $n \geq 1$ membranes labelled with elements from L;*
- *$M_i = (w_i, l_i)$, for each $1 \leq i \leq n$, is the initial configuration of membrane i with $l_i \in L$ and $w_i \in O^*$ a finite multiset of objects;*
- *R_i, for each $1 \leq i \leq n$, is a finite set of programs in membrane i of the form specified in Definition 2 with objects in O and labels in L.*

Thus, the initial configuration of each membrane i, with $1 \leq i \leq n$, is given by a finite multiset of objects from O and by a label from L. A program consists of a rule of the form $u\,[\,v\,]_{l_i} \rightarrow u'\,[\,v'\,]_{l_i}$, with l_i the label of membrane i; this program may be either associated with the membrane labelled with l_i or in the membrane surrounding it. Moreover, we can now precisely say that the evaluation of the predicate π in such a program must be done by considering both the content of membrane i and the content of the membrane $upper(\mu, i)$ (i.e., the membrane that directly contains membrane i). The evaluation of the predicate π, which has to be done before the application of the rule inside membrane i, is then denoted by $\pi(i, upper(\mu, i))$.

P systems are usually considered as being distributed maximal parallel multiset rewriting systems [19]. That is, in each step, for each membrane, a maximal set of programs to be applied is non-deterministically selected by making sure that no further programs can be applied to the objects left inside the membranes. On the other hand, a few recent works, [3,5,6,7], have addressed the issue of introducing new strategies for the application of the programs where the set of programs to be applied in any step is not maximal, but it is somehow bounded. The reasons for the introduction of new derivation/evolution strategies may be different, but, in the context of modelling biological systems, one can say that restrictions to maximal parallelism are often required in order to close the gap between the abstractness of the model and the *reality* of the phenomenon to be modelled. In this work we present two novel strategies for the evolution of P systems substantially different from the ones previously introduced.

3 Two Strategies for the Evolution of P systems

At the microscopic level of functioning of cellular processes the interactions between molecules follow the laws of physics. A fundamental result of theoretical statistical physics is the famous $\sqrt{n}$ law, which says that randomness or fluctuation level in a system are inversely proportional to the square root of the number of particles. As a result systems with low number of molecules show high fluctuations; whereas the evolution of systems with large number of reactants molecules can be considered deterministic. Having this in mind we introduce two strategies for the evolution of P systems, a stochastic strategy based on Gillespie's Algorithm and a deterministic strategy which will be valid for systems with high number of particles.

Gillespie's algorithm [9] (see also [11,12] for some recent improvements) provides an exact method for the stochastic simulation of systems of bio-chemical reactions; the validity of the method is rigorously proved and it has been already successfully used to simulate various biochemical processes [14]. As well as this, the Gillespie's algorithm is used in the implementation of stochastic π-calculus [21,29], and in its application to the modelling of biological systems [22]. Here we present an extension of the classical Gillespie's algorithm called *Multi-compartmental Gillespie Algorithm*. This method is developed by taking into account the fact that, with respect to the original algorithm where only one volume is studied, in P systems we have a membrane structure delimiting different regions or compartments, each one can be seen as a volume with its own set of rules, besides the application of a rule inside a compartment can also affect the content of another one; for example the application of a communication rule.

Specifically, let $\Pi = (O, Lab, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$ be a P system as specified in Definition 3 with the membranes $M_i = (w_i, L_i)$ and the programs R_i, $1 \leq i \leq n$. The set R_i of programs that are active inside membrane contains elements of the form $(j, \pi_j, r_j, p_j, k_j)$ where:

- j is the index of a program from R_i;
- π_j is the predicate; in this section this will be always true and will be omitted;
- r_j is the boundary rule contained in the program j;
- p_j is the probability of the rule contained in the program j to be applied in the next step of evolution; this probability is computed by multiplying a stochastic constant k_j, specifically associated with program j, by the number of possible combinations of the objects present on the left-side of the rules with respect to the multiset w_i (or the multiset $w_{i'}$, with $i' = upper(\mu, i)$) - the current content of membrane i (i').

First, each membrane i will be considered to be a compartment enclosing a volume, therefore the index of the next program to be used inside membrane i and its waiting time will be computed using the classical Gillespie's algorithm which we recall below:

1. calculate $a_0 = \sum p_j$, for all $(j, r_j, p_j, k_j) \in R_i$;
2. generate two random numbers r_1 and r_2 uniformly distributed over the unit interval $(0, 1)$;
3. calculate the waiting time for the next reaction as $\tau_i = \dfrac{1}{a_0} ln(\dfrac{1}{r_1})$;
4. take the index j, of the program such that $\displaystyle\sum_{k=1}^{j-1} p_k < r_2 a_0 \leq \sum_{k=1}^{j} p_k$;
5. return the triple (τ_i, j, i).

Notice that the larger the stochastic constant of a rule and the number of occurrences of the objects placed on the left-side of the rule inside a membrane are, the greater the chance that a given rule will be applied in the next step of the simulation. There is no constant time-step in the simulation. The time-step is determined in every iteration and it takes different values depending on the configuration of the system.

Next, the *Multi-compartmental Gillespie's Algorithm* is described in detail:

- **Initialisation**
  - set time of the simulation $t = 0$;
  - for each membrane i in μ compute a triple (τ_i, j, i) by using the procedure described above; construct a list containing all such triples;
  - sort the list of triple (τ_i, j, i) according to τ_i;
- **Iteration**
  - extract the first triple, (τ_m, j, m) from the list;
  - set time of the simulation $t = t + \tau_m$;
  - update the waiting time for the rest of the triples in the list by subtracting τ_m;
  - apply the rule contained in the program j only once changing the number of objects in the membranes affected by the application of the rule;
  - for each membrane m' affected by the application of the rule remove the corresponding triple $(\tau'_{m'}, j', m')$ from the list;
  - for each membrane m' affected by the application of the rule j re-run the Gillespie algorithm for the new context in m' to obtain $(\tau''_{m'}, j'', m')$, the next program j'', to be used inside membrane m' and its waiting time $\tau''_{m'}$;
  - add the new triples $(\tau''_{m'}, j'', m')$ in the list and sort this list according to each waiting time and iterate the process.
- **Termination**
  - Terminate simulation when time of the simulation t reaches or exceeds a preset maximal time of simulation.

Therefore, in this approach, the waiting time computed by the Gillespie's algorithm is used to select the membranes which are allowed to evolve in the next step of computation. Specifically, in each step, the membranes associated to programs with the same minimal waiting time are selected to evolve by means of the corresponding rules. Moreover, since the application of a rule can affect more than one membrane at the same time (e.g., some objects may be moved from one place to another), we need to reconsider a new program for each one of these membranes by taking into account the new distribution of objects inside them. Note that in this point our approach differs from [24] where only one program is applied at each step without taking into account the rest of programs that are waiting to be applied in the other membranes, neither it is considered the disruption that the application of one program can produce in various membranes.

As mentioned before in systems with large number of molecules deterministic approaches are valid; in what follows we present a deterministic *exact* strategy for the execution of the programs, that we will refer to as *Deterministic Waiting Times Algorithm.*

Given a P system, in this strategy, using *mass action law*, we associate a velocity, v_i, to every program i in each membrane j by multiplying the kinetic constant

k_i by the multiplicities of the reactants. Then we compute the waiting time for the first execution of the program as $\tau_i = \frac{1}{v_i}$ and return the triple (τ_i, i, j).

Below we give a detailed description of the *Deterministic Waiting Times Algorithm*:

- **Initialisation**
  - set time of the simulation $t = 0$;
  - for every program i associated with a membrane j in μ compute the triple (τ_i, i, j) by using the procedure described before; construct a list containing all such triples;
  - sort the list of triple (τ_i, i, j) according to τ_i;
- **Iteration**
  - extract the first triple, (τ_j, j, m) from the list;
  - set time of the simulation $t = t + \tau_j$;
  - update the waiting time for the rest of the triples in the list by subtracting τ_j;
  - apply the rule contained in the program j only once changing the number of objects in the membranes affected by the application of the rule;
  - for each membrane m' affected by the application of the rule remove the corresponding all the triples $(\tau'_{j'}, j', m')$ from the list;
  - for each membrane m' affected by the application of the rule j re-run the Gillespie algorithm for the new context in m' to obtain a triple $(\tau''_{j'}, j', m')$ for all the program s j' associated with the membrane m';
  - add the new triples $(\tau''_{j'}, j', m')$ in the list and sort this list according to each waiting time and iterate the process.
- **Termination**
  - Terminate simulation when time of the simulation t reaches or exceeds a preset maximal time of simulation.

Note that in this algorithm instead of associating a waiting time to a single program in each membrane (as it is the case in the *Multi-compartmental Gillespie's Algorithm*) every program in each membrane has a waiting time computed in a deterministic way that is used to determine the order in which the programs are executed. Also highlight that this is an *exact* method in the sense that we do not approximate infinitesimal intervals of time by Δt as it is the case in ODEs, but the time step varies across the evolution of the system and it is computed in each step depending on the current state of the system.

These two strategies have been implemented using Scilab, a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific applications [30]. The program and configuration files of the case studies presented in this paper are available from `www.gcn.us.es`.

In the following two sections we provide two biological systems to illustrate these strategies. In section 4 we study the EGFR Signalling Cascade where the *Deterministic Waiting Times Algorithm* is suitable for describing its evolution; and in section 5 the Quorum Sensing System in the bacterium Vibrio Fischeri is used as an example where stochastic approaches like the *Multi-compartmental Gillespie's Algorithm* are necessary.

4 EGFR Signalling Cascade

The epidermal growth factor receptor (EGFR) belongs to the tyrosine kinase family of receptors. Binding of the epidermal growth factor (EGF) to the extracellular domain of EGFR induces receptor dimerisation and autophosphorylation of intracellular domains. Then a multitude of proteins are recruited starting a complex signalling cascade and the receptor follows a process of internalisation and degradation in endosomals. Two principal pathways lead to activation of Ras-GTP by hydrolisation of Ras-GDP. One of these pathways depends on the Src homology and collagen domain protein (Shc) and the other one is Shc-independent. Ras-GTP acts like a *switch* that stimulates the Mitogen Activated Protein (MAP) kinase cascade by phosphorylating the proteins Raf, MEK and ERK. Subsequently phosphorylated MEK and ERK regulates several cellular proteins and nuclear transcription factors. Disregulated EGFR expression, ligand production and signalling have been proved to have a strong association with tumourgenesis. As a result of this, EGFR has been identified as a key biological target for the development of novel anticancer therapies. In figure 2 it is shown a detailed graphical representation of the signalling cascade.

We have developed a model of the signalling cascade described on the previous page using the following P system:

$$\mathbf{\Pi}_{EGF} = (O, \{e, s, c\}, \mu, (w_1, e), (w_2, s), (w_3, c), \mathcal{R}_e, \mathcal{R}_s, \mathcal{R}_c)$$

Our model consists of more that 60 proteins and complexes of proteins and 160 chemical reactions. Due to the limitation of space we will not give all the details of the model. A complete description of $\mathbf{\Pi}_{EGF}$ with some supplementary information is available from the web page `www.gcn.us.es/egfr.pdf`. In what follows we give an outline of our model.

• **Alphabet:** In the alphabet O we represent all the proteins and complexes of proteins that take part in the signalling cascade. Some of the objects from the alphabet and the chemical compounds that they represent are listed below.

Object	**Protein or Complex**
EGF	Epidermal Growth Factor
EGFR	EGF Receptor
EGFR-EGF$_2$	Dimerazated Receptor
EGFR-EGF$_2^*$-Shc	EGFR-EGF$_2^*$ and Shc complex
$\vdots$	$\vdots$
MEK	Mitogenic external regulated kinase
ERK	External regulated Kinase

• **Membrane Structure:** In the EGFR Signalling Cascade there are three relevant regions, namely the *environment*, the *cell surface* and the *cytoplasm*. We represent them in the membrane structure as the membranes labelled with: e for the environment, s for the cell surface and c for the cytoplasm. Figure 3 is a Venn diagram representation of the membrane structure.

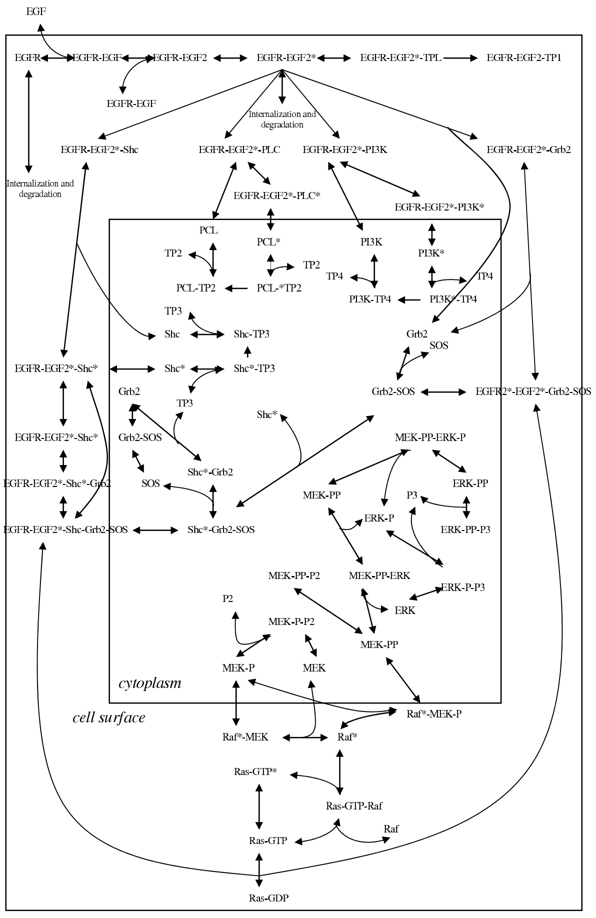


Fig. 2. EGFR Signalling Cascade

• **Initial Multisets:** In the initial multisets we represent the initial number of molecules (nM) of the chemical substances in the environment, the cell surface and the cytoplasm. These estimations has been obtained from [16,23].

$$w_1 = \{EGF^{200}\}$$
$$w_2 = \{EGFR^{250}, Ras\text{-}GDP^{200}\}$$
$$w_3 = \{Shc^{250}, PLC_{\gamma}^{150}, PI3K^{50}, SOS^{40}, Grb2^{80}, TP_1^{100}, TP_2^{450}, TP_3^{450}, TP_4^{125}, Raf^{80}, MEK^{400}, ERK^{400}, P_1^{80}, P_2^{80}, P_3^{300}\}$$

• **Programs:** In the programs we model the 160 chemical reactions which form the signalling cascade. Due to the limitation of space only a few programs representing some of the reactions will be presented in this paper. For a detailed enumeration of all the programs, their attributes and the references from where

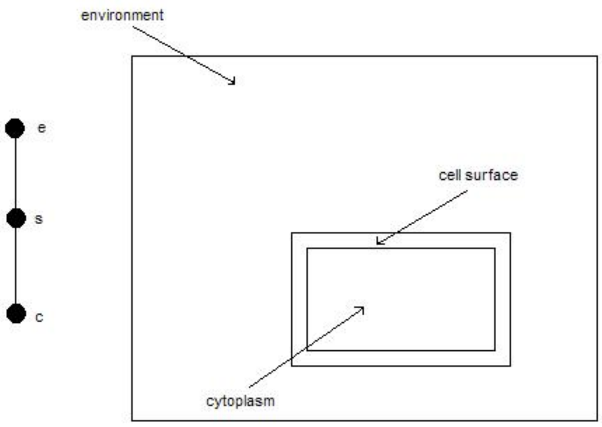


Fig. 3. Membrane Structure

these parameters were taken see the supplementary information available from `www.gcn.us.es/egfr.pdf`.

As it can be seen in the initial multisets specified before, in the system of the EGFR Signalling Cascade the number of molecules is quite large, as a consequence of the $\sqrt{n}$ *law* important fluctuations and stochastic behaviour are not expected in the evolution of the system. Because of this we have chosen the *Deterministic Waiting Times Algorithm* as the strategy for the evolution of the P system $\mathbf{\Pi}_{EGF}$.

In what follows we show two examples of programs from the set of programs associated with each membrane.

The set of programs associated with the environment, $\mathcal{R}_e$, consists only of one program which models the binding of the signal, EGF, to the receptor $EGFR$.

$$EGF\,[\,EGFR\,]_s \;\rightarrow\; [\,EGF\text{-}EGFR\,]_s, k = 0.003$$

We assume that the previous program is applicable when there are signals and receptors available and so, the predicate π is omitted. The meaning of the previous rule is the following: the object EGF in the membrane containing the membrane with label s (the environment), and the object $EGFR$ inside the membrane with label s (the cell surface) are replaced with the object $EGFR\text{-}EGF$ in the membrane with label s; this object represents the complex receptor-signal on the cell surface. As attributes we associate the kinetic constant k, which measures the affinity between the signal and the receptor.

The *Deterministic Waiting Times Algorithm* is used in the evolution of the system and the waiting time associated to this program will be computed using the next formula:

$$\tau = \frac{1}{0.003|EGF||EGFR|}$$

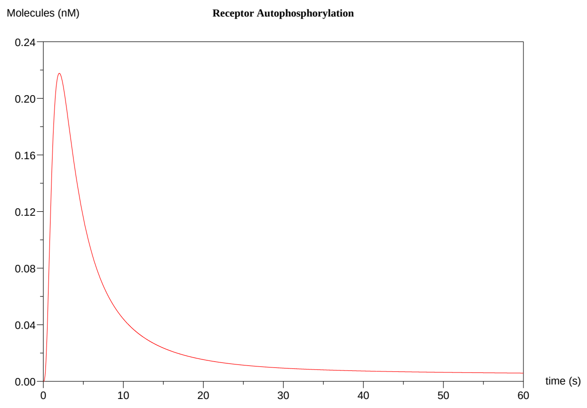


Fig. 4. Autophosphorylated EGFR evolution

One example from the set of programs $\mathcal{R}_s$ associated to the cell surface is the dimerisation of the receptor, that is the formation of a complex consisting of two receptors:

$$[\,EGFR,\, EGFR\,]_s \;\rightarrow\; [\,EGFR_2\,]_s, k = 0.011$$

When this program is executed two objects $EGFR$ representing receptors are replaced with one object $EGFR_2$, representing a complex formed with two receptors, in the membrane with label s, the cell surface. The kinetic constant k is used to computed the waiting time:

$$\tau = \frac{1}{0.011|EGFR|^2}$$

Using the software mentioned in the previous section and developed in Scilab we run some experiments; in what follows we present some of the results obtained.

In figure 4 it is depicted the evolution of the number of autophosphorylated receptors and, in figure 5 the number of doubly phosphorylated MEK (Mitogen External Kinase), one of the target proteins of the signalling cascade that regulates some nuclear transcription factors involved in the cell division.

Note that the activation of the receptor is very fast reaching its maximum within the first 5 seconds and then it decays fast to very low levels; on the other hand the number of doubly phosphorylated MEK is more sustained around 3 nM. These results agree well with empirical observations, see [16,23].

In tumours it has been reported an overexpression of signals EGF in the environment and of receptors, EGFR, on the cell surface of cancerous cells. Here we investigate the effect of different EGF concentrations and number of receptors on the signalling cascade.

First, we study the effect on the evolution of the number of autophosphorylated receptors and doubly phosphorylated MEK of a range of signals, EGF, from 100 nM to 2000 nM.

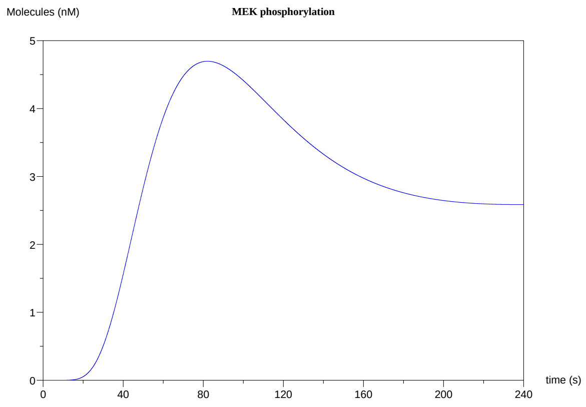


Fig. 5. Doubly Phosphorylated MEK evolution

In figure 6, it can be seen that the receptor autophosphorylation is clearly concentration dependent showing different peaks for different number of signals in the environment. According to the variance in the receptor activation it is intuitive to expect different cell responses to different EGF concentrations. Here we will see that this is not the case.

Observe, in figure 7, that the number of doubly phosphorylated MEK does not depend on the number of signals in the environment. This shows the surprising robustness of the signalling cascade with regard to the number of signals from outside due to EGF concentration. The signal is either attenuated or amplified to get the same concentration of one of the most relevant kinases in the signalling cascade, MEK. Note that after 100 seconds, when the response gets sustained, the lines representing the response to different external EGF concentrations are identical.

Now we analyse the effect on the dynamics of the signalling cascade of different numbers of receptors on the cell surface.

In figure 8, it is shown the evolution of the number of doubly phosphorylated MEK when there is 100 nM and 1000 nM of receptors on the cell surface. Note that now the response is considerably different; the number of activated MEK is greater when there is an overexpression of receptors on the cell surface. As a consequence of this high number of activated MEK the cells will undergo an uncontrolled process of proliferation.

The key role played by the overexpression of EGFR on the uncontrolled growth of tumours has been reported before, as a consequence of this, EGFR is one of the main biological targets for the development of novel anticancer therapies.

Finally, stress that for this system we have used a deterministic approach obtaining results that map well experimental data. This is not always the case, in the next section we analyse a system where a stochastic approach is necessary to describe properly its behaviour.

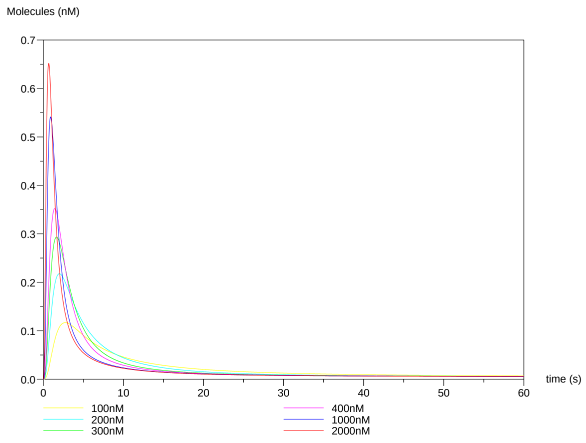


Fig. 6. Receptor Autophosphorylation for different environmental EGF concentrations

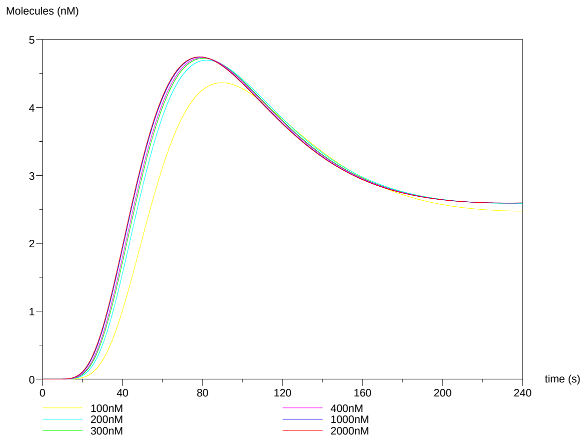


Fig. 7. MEK phosphorylation for different environmental EGF concentrations

5 Quorum Sensing System in the Bacterium Vibrio Fischeri

Quorum sensing is a cell density dependent gene regulation system that allows an entire population of bacterial cells to communicate in order to regulate the expression of certain or specific genes in a coordinated way depending on the size of the population. In this section we present a model of the Quorum Sensing System in the marine bacterium *Vibrio fischeri* using P systems. In this framework each bacterium and the environment are represented by membranes. This allows us to examine the individual behaviour of each bacterium as an agent as well as the behaviour of the colony as a whole and the processes of swarming and recruitment.

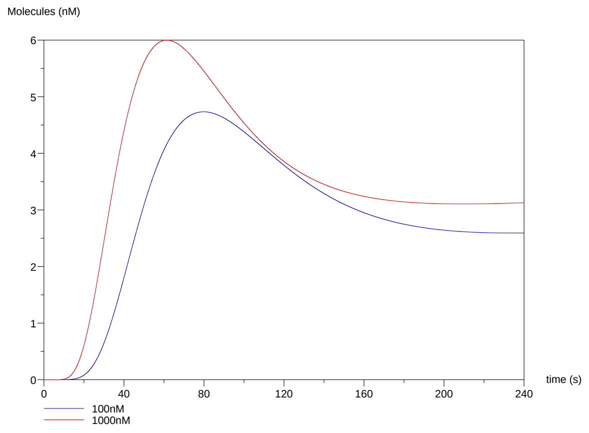


Fig. 8. MEK phosphorylation for different number of receptors

The marine bacterium *Vibrio fischeri* exists naturally either in a free-living planktonic state or as a symbiont of certain luminescent squid. The bacteria colonise specialised light organs in the squid, which cause it to luminesce. Luminescence in the squid is thought to be involved in the attraction of prey, camouflage and communication between different individuals. The source of the luminescence is the bacteria themselves. The bacteria only luminesce when colonising the light organs and do not emit light when in the free-living state.

The Quorum Sensing System in Vibrio Fischeri relies on the synthesis, accumulation and subsequent sensing of a signal molecule, 3-oxo-C6-HSL, an N-acyl homoserine lactone or AHL (we will call it OHHL). When only a small number of bacteria are present the signal is produced by the bacteria at a low level. OHHL diffuses out of the bacterial cells and into the surrounding environment. At high cell density the signal accumulates in the area surrounding the bacteria and can also diffuse to the inside of the bacterial cells. The signal is able to interact with the LuxR protein to form the complex LuxR-OHHL. This complex binds to a region of DNA called the Lux Box causing the transcription of the luminescence genes, a small cluster of 5 genes, luxCDABE. Adjacent to this cluster are two regulatory genes for the transcription of LuxR and OHHL. In this sense OHHL and LuxR are said to be autoinducer because they activate their own synthesis.

The bacteria are effectively communicating, as a single bacterium is able to detect and respond to signals produced by the surrounding bacteria. Bacteria sense their cell density by measuring the amount of signal present; quorum sensing can therefore explain why the bacteria are dark when in the free living planktonic state at low cell density and light when colonising the light organ of squid at high cell density. A large number of Gram negative bacteria have been found to have AHL-based quorum sensing systems similar to Vibrio fischeri.

In what follows we present a model of the Quorum Sensing System in Vibrio fischeri using a parametric P system, $\mathbf{\Pi}_{Vf}(N)$; where N represents the number of bacteria in the colony. We will study the behaviour of such colony placed inside a

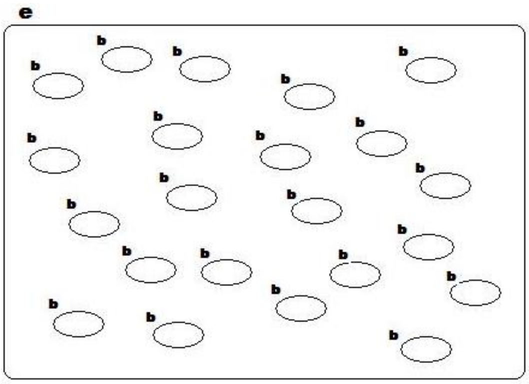


Fig. 9. Membrane Structure

single environment by examining the evolution of the following P system:

$$\mathbf{\Pi}_{Vf}(N) = (O, \{e, b\}, \mu, (w_1, e), (w_2, b) \ldots, (w_{N+1}, b), \mathcal{R}_b, \mathcal{R}_e) \quad \text{where:}$$

(1) Alphabet: In the alphabet we represent the signal OHHL, the protein LuxR, the complex protein-signal, the regulatory region LuxBox and the regulatory region occupied by the complex.

$$O = \{\text{OHHL}, \text{LuxR}, \text{LuxR-OHHL}, \text{LuxBox}, \text{LuxBox-LuxR-OHHL}\}$$

(2) Membrane Structure and Labels: In the Quorum Sensing System of Vibrio fischeri there are two relevant regions, namely the environment and the bacteria. The environment will be represented by the membrane labelled with e and each bacterium is represented by a membrane with label b. We can represent the membrane structure μ as a Venn diagram in figure 9.

(3) Initial Multisets: In the initial multisets we represent the initial conditions of the system. We are interested in examining how bacteria communicate to coordinate their behaviours and how the system moves from a downregulated state, where the protein and the signal are produced at basal rates, to an upregulated state, where the bacteria produce light. Therefore, in the initial multisets we will suppose that there is nothing in the environment and in the bacteria we will only have the genome (LuxBox) to start the production of the signal and protein at basal rates.

$$w_1 = \emptyset, \ w_i = \{\text{LuxBox}\} \quad 2 \leq i \leq N+1$$

(4) Programs: In the programs we model the chemical reactions forming the Quorum Sensing System. Next we specify the set of programs associated with the bacteria, $\mathcal{R}_b$, and with the environment, $\mathcal{R}_e$; we also describe briefly the chemical reactions they represent.

- Set of programs associated with the bacteria, $\mathcal{R}_b$:
  In an unstressed bacterium the transcription of the signal OHHL and the protein LuxR takes place at basal rates.

$r_1 : [\,\text{LuxBox}\,]_b \xrightarrow{k_1} [\,\text{LuxBox}\,,\ \text{OHHL}\,]_b$

$r_2 : [\,\text{LuxBox}\,]_b \xrightarrow{k_2} [\,\text{LuxBox}\,,\ \text{LuxR}\,]_b$

The protein LuxR acts as a receptor and OHHL as its ligand. Both together form the complex LuxR-OHHL which in turn can dissociate into OHHL and LuxR again.

$r_3 : [\,\text{LuxR}\,,\ \text{OHHL}\,]_b \xrightarrow{k_3} [\,\text{LuxR-OHHL}\,]_b$

$r_4 : [\,\text{LuxR-OHHL}\,]_b \xrightarrow{k_4} [\,\text{LuxR}\,,\ \text{OHHL}\,]_b$

The complex LuxR-OHHL acts as a transcription factor binding to the regulatory region of the bacterium DNA called LuxBox. The complex LuxR-OHHL can also dissociate from the LuxBox.

$r_5 : [\,\text{LuxBox}\,,\ \text{LuxR-OHHL}\,]_b \xrightarrow{k_5} [\,\text{LuxBox-LuxR-OHHL}\,]_b$

$r_6 : [\,\text{LuxBox-LuxR-OHHL}\,]_b \xrightarrow{k_6} [\,\text{LuxBox}\,,\ \text{LuxR-OHHL}\,]_b$

The binding of the complex LuxR-OHHL to the LuxBox produce a massive increase in the transcription of the signal OHHL and of the protein LuxR.

$r_7 : [\,\text{LuxBox-LuxR-OHHL}\,]_b \xrightarrow{k_7} [\,\text{LuxBox-LuxR-OHHL}\,,\ \text{OHHL}\,]_b$

$r_8 : [\,\text{LuxBox-LuxR-OHHL}\,]_b \xrightarrow{k_8} [\,\text{LuxBox-LuxR-OHHL}\,,\ \text{LuxR}\,]_b$

OHHL can diffuse outside the bacterium and accumulate in the environment.

$r_9 : [\,\text{OHHL}\,]_b \xrightarrow{k_9} \text{OHHL}\ [\ ]_b$

OHHL, LuxR and the complex LuxR-OHHL undergo a process of degradation in the bacterium.

$r_{10} : [\,\text{OHHL}\,]_b \xrightarrow{k_{10}} [\ ]_b$

$r_{11} : [\,\text{LuxR}\,]_b \xrightarrow{k_{11}} [\ ]_b$

$r_{12} : [\,\text{LuxR-OHHL}\,]_b \xrightarrow{k_{12}} [\ ]_b$

- Set of programs associated with the environmet, $\mathcal{R}_e$:
  The signal OHHL in the environment can diffuse inside the bacteria and undergo a process of degradation.

$r_{13} : \text{OHHL}\ [\ ]_b \xrightarrow{k_{13}} [\,\text{OHHL}\,]_b$

$r_{14} : [\,\text{OHHL}\,]_e \xrightarrow{k_{10}} [\ ]_e$

In order to implement our model we have chosen the following set of parameters, $k_1 = 2, k_2 = 2, k_3 = 9, k_4 = 1, k_5 = 10, k_6 = 2, k_7 = 250, k_8 = 200, k_9 = 1, k_{10} = 50, k_{11} = 30, k_{12} = 15, k_{13} = 20, k_{14} = 20$, these parameters has been taken or suggested by the literature listed in [25].

As it can be seen in the initial multisets, the Quorum Sensing System has a very small number of molecules. At the beginning of the evolution there is only a single molecule in each bacterium representing its genome. Therefore as a consequence of the $\sqrt{n}$ *law* stochastic behaviour is expected. To check if this is true we have used the two strategies introduced in section 3 for the evolution of the P system $\mathbf{\Pi}_{Vf}(N)$.

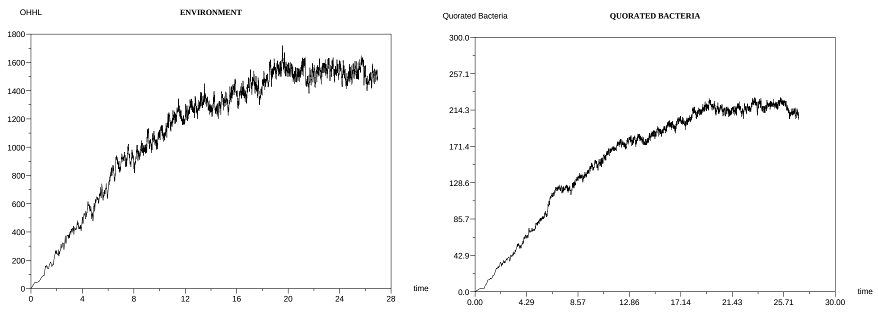


Fig. 10. Number of signals in the environment and quorated bacteria

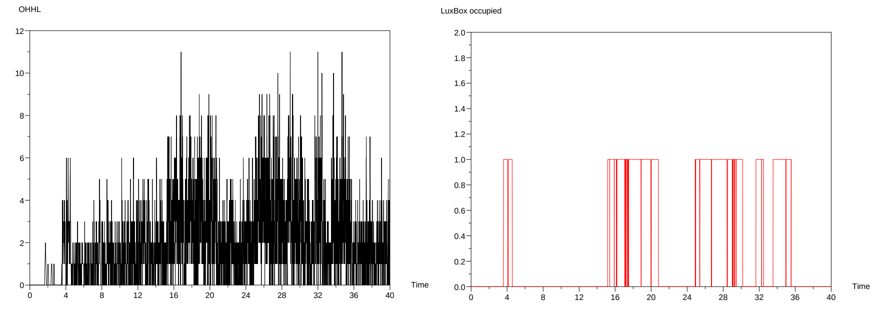


Fig. 11. Number of signals and occupation of the LuxBox in a bacterium

First, we use the *Multi-compartmental Gillespie's Algorithm*. We have studied the behaviour of the system for two populations of different size to examine how bacteria can sense the number of bacteria in the population and produce light only when the number of individuals is big enough.

To start with we have considered a population of 300 bacteria. We examine the evolution over time of the number of quorated bacteria [1] (figure 10 right) and the number of signals (OHHL) in the environment (figure 10 left).

Observe that the signal, OHHL, accumulates in the environment until saturation and then, when this threshold is reached, bacteria are able to detect that the size of the population is big enough. At the beginning, a few bacteria get quorated and then they accelerate a process of recruitment that makes the whole population behave in a coordinated way.

There exists a correlation between the number of signal in the environment and the number of quorated bacteria such that, when the number of signal in the environment drops, so does the number of quorated bacteria and when the signal goes up it produces a recruitment of more bacteria.

[1] We will say that a bacterium is quorated if the LuxBox in this bacterium is occupied by the complex producing the transcription of the enzymes involved in the production of light.

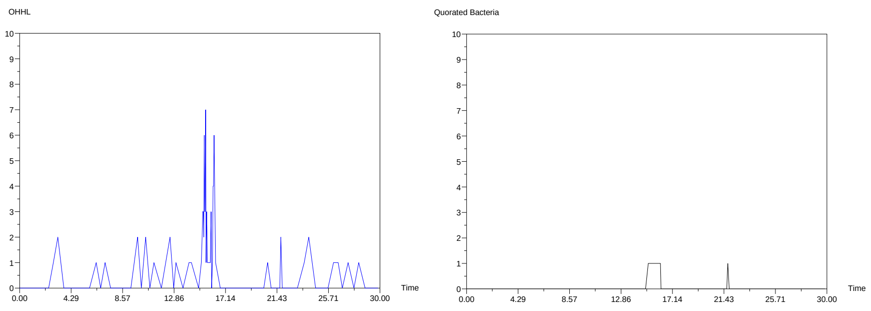


Fig. 12. Signals in the environment and quorated bacteria

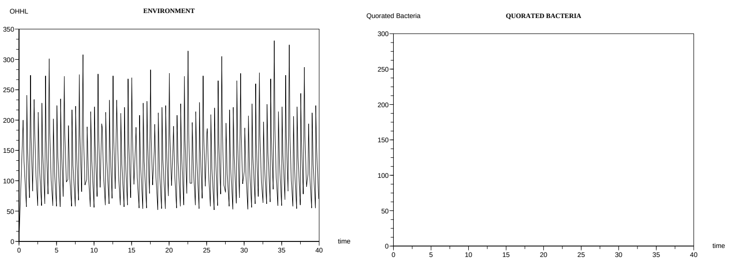


Fig. 13. Number of signals and quorated bacteria

In our approach the behaviour of each individual in the colony can be tracked. In figure 11, for one bacterium it is shown the correlation between the number of signal inside the bacterium (left) and the occupation of the LuxBox by the complex (right), which represents that the bacterium has been quorated.

It can be seen, in figure 11, that the number of signal molecules inside the bacterium has to exceed a threshold of approximately 6 molecules in order to recruit the bacterium. Observe that when the number of molecules is greater than 6 the LuxBox is occupied, that is, the bacterium is quorated or upregulated but when there is less than six signals the bacterium switches off the system and stops producing light.

Now, in order to study how bacteria can sense the number of individuals in the colony and get quorated only when the size of the colony is big enough we have examined the behaviour of a population of only 10 bacteria.

Observe in figure 12 that the signal does not accumulate in the environment and no recruitment process takes place. Only one of the bacteria guessed wrong the size of the population and got upregulated. But then, after sensing that the signal does not accumulate in the environment, it switched off its systems.

Finally, in order to show the inherently random character of this system, we will show the results obtained using the *Deterministic Waiting Times Algorithm* with the same set of parameters.

In figure 13, it is depicted the number of signals (left) and the number of quorated bacteria (right) for a population of 300 bacteria. Note that using the

Deterministic Waiting Times Algorithm the signal does not accumulate and not even a single bacterium got quorated. These results do not agree with empirical results which show that at high cell densities the signal saturates the environment and the colony of bacteria produces light. The way a bacterium can sense the size of the colony is by *guessing* that the number of bacteria is big enough to start to produce light, then the bacterium start to send signals to the environment. If the signal accumulates in the environment it means that the population is big enough otherwise it is too small and the bacterium switches off the system. This *guessing* can not be modelled using a deterministic approach. Therefore here we have an example of a system with intrinsic random behaviour where stochastic approaches like the Multicompartmental Gillespie's Algorithm are necessary.

6 Conclusions

In this paper we have presented P systems as a new computational modelling tool for systems biology; stressing their main characteristics, consideration of the key role played by membranes in the structure and functioning of the cells and the representation of the discrete character of the quantity of components in biosystems. P system models are also a general specification of the biological phenomena that can be evolved using different strategies/algorithms. In this work two novel strategies have been introduced; *Multi-compartmental Gillespie's Algorithm* based on the well known Gillespie's Algorithm but running on more than one compartment; and *Deterministic Waiting Times Algorithm*, an *exact* deterministic method. These two strategies required much computational resources and so the authors will study the possibility to adapt improved version of Gillespie's algorithm, like [11,12], in order to develop more efficient strategies for the evolution of P systems.

Two different biological phenomena have been used two illustrate these strategies. First, in the EGFR Signalling Cascade the *Deterministic Waiting Times Algorithm* was used and second in the Quorum Sensing System in the bacterium Vibrio Fischeri the *Multi-compartmental Gillespie's Algorithm* was necessary in order to obtain results in accordance with empirical observations.

The fact that our results agree well with previously formulated hypothesis shows the reliability of P systems as computational modelling tools to produce postdiction and perhaps as the field evolves they will be able to produce plausible predictions.

Acknowledgement

This work is supported by Ministerio de Ciencia y Tecnología of Spain, by *Plan Nacional de I+D+I* (TIN2005-09345-C04-01), cofinanced by FEDER funds, and by a FPU fellowship from the Ministerio de Ciencia y Tecnología of Spain.

References

1. Aho, A.V., Sethi, R., Ulmann, J.D. (1986). *Compilers: Principles, Techniques, and Tools*. Addison-Wesley.
2. Ardelean, I., Cavaliere, M. (2003). Playing with a Probabilistic P Simulator: Mathematical and Biological Problems. In: *Brainstorming Week on Membrane Computing, Tarragona, Feb 5-11 2003* (Cavaliere, M., Martin-Vide, C., Păun, Gh., eds.). Tech. Rep. **26/03**, Universitat Rovira i Virgili, Tarragona, Spain, 37–45.
3. Bernardini, F., Gheorghe, M., Muniyandi, R.C., Krasnogor, N., Pérez-Jiménez, M.J., Romero-Campero, F.J. (2006). On P Systems as a Modelling Tool for Biological Systems. *Lecture Notes in Computer Science*, **3850**, 114–133.
4. Bernardini, F., Manca, V. (2003). P Systems with Boundary Rules. In: [20], 107–118.
5. Besozzi, D. (2004). *Computational and Modelling Power of P systems*, Ph.D. Thesis, Università degli Studi di Milano, Milan, Italy.
6. Bianco, L., Fontana, F., Franco, G., Manca, V. (2005). P Systems for Biological Dynamics. In: *Applications of Membrane Computing* (Ciobanu, G., Păun, Gh., Pérez-Jiménez, M.J., eds.), Springer-Verlag, Berlin, Heidelberg, New York, 81–126.
7. Bianco, L., Fontana, F., Manca, V. (2005). P Systems and the Modelling of Biochemical Oscillation. In: *Pre-Proceedings of WMC6 - Vienna 2005*, 214–225.
8. Gibson, M.A., Bruck, J., (2000). Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *Journal of Physical Chemistry*, **104**, 25, 1876–1889.
9. Gillespie, D.T. (1976). A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *J Comput Physics*, **22**, 403–434.
10. Gillespie, D.T. (1977). Exact Stochastic Simulation of Coupled Chemical Reactions. *The Journal of Physical Chemistry*, **81**, 25, 2340–2361.
11. Gillespie, D.T. (2001). Approximate Accelerated Stochastic Simulation of Chemically Reacting Systems. *Journal of Chemical Physics*, **115**, 4, 1716–1733.
12. Gillespie, D.T. (2003). Improved Leap-size Selection for Accelerated Stochastic Simulation. *Journal of Chemical Physics*, **119**, 16, 8229–8234.
13. Karwowski, R., Prusinkiewicz, P. (2003). Design and Implementation of the L+C Modelling Language. *Electronics Notes in Theoretical Computer Science*, **82**, 2, 1–19.
14. Meng, T.C., Somani S., Dhar, P. (2004). Modelling and Simulation of Biological Systems with Stochasticity. *In Silico Biology*, **4**, 0024.
15. Milner, R. (1999). *Communicating and Mobile System: The π-Calculus*. Cambridge University Press.
16. Moehren G. et al (2002). Temperature Dependence of the Epidermal Growth Factor Receptor Signaling Network Can Be Accounted for by a Kinetic Model, *Biochemistry* **41** 306–320.
17. Păun, A.; Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport,*New Generation Computing*, **20**, 3 (2002), 295–305.
18. Păun, Gh.: Computing with Membranes, *Journal of Computer and System Sciences*, **61**(1) (2000) 108 – 143.
19. Păun, Gh. (2002). *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, Heidelberg, New York.
20. Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C., eds. (2003). Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers. *Lecture Notes in Computer Science*, **2597**, Springer-Verlag, Berlin, Heidelberg, New York.

21. Philips, A., Cardelli. L. (2004). A Correct Abstract Machine for the Stochastic Pi-calculus. *Electronical Notes in Theoretical Computer Science*, to appear.
22. Priami, C., Regev, A., Shapiro, E., Silverman, W. (2001). Application of a Stochastic Name-Passing Calculus to Representation and Simulation of Molecular Processes. *Information Processing Letters*, **80**, 25–31.
23. Schoeberl, B. et al. (2002). Computational Modeling of the Dynamics of the MAP Kinase Cascade Activated by Surface and Internalized EGF Receptors, *Nature Biotech.*, **20**, 370–375.
24. Stundzia, A.B., Lumsden, C.J. (1996). Stochastic Simulation of Coupled Reaction-Diffusion Processes. *Journal of Computational Physics*, **127**, 196–207.
25. Nottingham Quorum Sensing Web Site `http://www.nottingham.ac.uk/quorum/`
26. ISI Web Site `http://esi-topics.com/erf/october2003.html`
27. Scilab Web Pages: `http://scilabsoft.inria.fr`.
28. The P Systems Web Page: `http://psystems.disco.unimib.it`.
29. The Stochastic Pi-Machine: `http://www.doc.ic.ac.uk/~anp/spim/`.
30. SciLab Web Site `http://scilabsoft.inria.fr/`

Equivalence of Metabolite Fragments and Flow Analysis of Isotopomer Distributions for Flux Estimation[⋆]

Ari Rantanen[1], Hannu Maaheimo[2], Esa Pitkänen[1], Juho Rousu[1], and Esko Ukkonen[1]

[1] Dept. of Computer Science and HIIT Basic Research Unit
University of Helsinki, Finland
Firstname.Lastname@cs.Helsinki.Fi
[2] VTT Technical Research Centre of Finland
Hannu.Maaheimo@vtt.Fi

Abstract. The most accurate estimates of the activity of metabolic pathways are obtained by conducting isotopomer tracer experiments. The success of this method, however, is intimately dependent on the quality and amount of data on isotopomer distributions of intermediate metabolites. In this paper we present a novel method for discovering sets of metabolite fragments that always have identical isotopomer distributions, regardless of the velocities of the reactions in the metabolic network. We outline several applications of this equivalence concept, including improved propagation of measurements, experiment planning and consistency checking of metabolic network. Our computational experiments in measurement propagation indicate that the improvement via the use of this technique may be substantial.

1 Introduction

The goal of metabolic flux analysis is to discover the steady state velocities of chemical reactions that constitute a metabolic network of an organism. Information about reaction velocities, or fluxes, constitutes an important aspect of the physiological state of the cell [15] that can be harnessed in many different applications ranging from pathway optimization in metabolic engineering [22] and from characterization of the physiology of an organism [10] to more efficient drug design [3].

The basic method to analyze the flux distribution is to rely only on measurements of extracellular fluxes and stoichiometric description of the metabolic network of an organism. This kind of analysis can give interesting information about the feasible fluxes but can not in general discover the complete flux distribution.

[⋆] Preliminary version "Flow analysis of metabolite fragments for flux estimation" appeared in the proceedings of CMSB 2005 (Third International Workshop on Computational Methods in Systems Biology), Edinburgh.

Separate fluxes for forward and backward reactions and the fluxes of cycles and alternative pathways between metabolites remain unknown [28].

In isotopomer tracer experiments the cell is fed with a mixture of natural and labelled nutrients. The most common labelling technique is to use nutrients with ^{13}C carbons in their backbones. If alternative pathways manipulate the carbon chains of metabolites differently, the *isotopomer distributions* [1] of metabolic products and intermediates depend on the fluxes of a network. The isotopomer distributions can be observed by measuring the metabolites with NMR [24] or mass spectrometry [4,31]. Based on these observations more constraints to the fluxes can be obtained. The flux estimation using isotopomer data has been successfully applied in numerous cases to solve (key) fluxes in specific metabolic networks and experimental conditions of interest [13,18,22,23].

A popular framework for estimating the flux distribution of an arbitrary metabolic network is based on non-linear optimization procedure where candidate values for the fluxes are generated successively and the isotopomer distributions of metabolites corresponding candidate fluxes are computed from the network model until the isotopomer distributions fit the measured data well enough [7,18,20,29].

Recently Rousu *et al.* [17] proposed a direct flux estimation method that first propagates the measurement information in the metabolic network and then augments the stoichiometric constraints to the fluxes with generalized isotopomer balances. This linear flux estimation method can be seen as a member of another flux estimation framework originating from METAFoR (metabolic flux ratio) analysis [6,21,23,24].

Both flux estimation frameworks have their strengths and weaknesses. The quality of flux estimations given by both frameworks depend heavily on the correctness of the model of a metabolic network used. Metabolic reactions constituting the central carbon metabolism of some model organisms such as *Saccharomyces cerevisiae* are regarded as well understood, but for most organisms metabolic pathways must be carefully reconstructed before further analysis [5].

The iterative flux estimation methods require often fewer measured metabolites than METAFoR methods to produce an estimate of a flux distribution. On the other hand, due to the nonlinearity of the optimization task, it is hard to guarantee that the iterative method converges to the global, unique optimum. If measured data does not fully determine the flux distribution, iterative methods output only points from the solution space, not the set of all feasible solutions. By performing an a priori identifiability analysis [9,26] and analyzing the sensitivity of a point solution [14] it is possible to examine the uniqueness of the solution.

In the method of [17] the equation system bounding the flux distribution is linear. Thus the equation system is easily solvable in a fully determined case.

[1] By different isotopomers of a metabolite we mean molecules that have specific combination of ^{12}C and ^{13}C atoms in different positions of the carbon chain of the metabolite. For example, pyruvate CH3COCOOH has three carbons and $2^3 = 8$ different isotopomers. Isotopomer distribution of the metabolite then gives the relative concentrations of different isotopomers.

If the equation system is underdetermined the linear subspace containing all solutions can be explicitly stated and well known techniques can be applied to solve as many fluxes as possible [11] as well as to obtain upper and lower bounds to the rest of the fluxes. The general sensitivity of a solution to measurement errors can be easily estimated by inspecting the condition number of a linear equation system [19].

Linear flux estimation techniques thus have attractive properties compared to nonlinear, iterative techniques. However, the need for more measurement data is a problem that should not be overlooked. Developing measurement techniques for intermediate metabolites and conducting the measurements using the technique are nontrivial, laborious and costly processes.

In this article we present a method for discovering the sets of fragments of metabolites that always have identical isotopomer distributions, regardless of the flux distribution. The method is based on the flow analysis of the fragments in the metabolic network. Information about the sets of fragments with identical isotopomer distributions can be used to improve the utilization of the measurement data in the linear flux estimation methods and thus reduce the need for measurements. Our computational experiments show that this improvement can be significant in real metabolic networks. The sets of fragments with identical isotopomer distributions also give powerful tools for the design of isotopomer tracer experiments, for calculability analysis and for the estimation of measurement errors.

Section 2 of this paper defines the formal model of a metabolic network and introduces elementary flux balances that are valid in a steady state. In Section 3 elementary flux balances are augmented with isotopomer balances. In Section 4 flow analysis techniques are applied to discover the sets of fragments that have equal isotopomer distributions in every steady state. In Section 5 the relation of these sets to the propagation of isotopomer data in a metabolic network and thus to the construction of isotopomer balances is described. Also, a method for utilizing the sets of fragments with identical isotopomer distributions in the design of carbon labelling experiments is introduced in Section 5, as well as extensions to the metabolic network model given in Section 2. Section 6 summarizes the results of computational experiments done with a model of central metabolism of *Saccharomyces cerevisiae.*

2 Metabolic Networks, Carbon Maps, and Flux Balance Equations

Basic Assumptions

A metabolic network is a formal model of the flow of atoms between metabolites (chemical compounds) in a metabolic system. Such a system consists of a collection of metabolites (chemical compounds) and metabolic reactions between them. Each reaction takes some molecules of certain metabolites and transforms them into molecules of some other metabolites. The available molecules of a metabolite constitute a *pool.* We assume that the pools are uniformly spatially mixed with the pools of the other metabolites, and that all reactions are continuously occurring simultaneously in parallel, consuming and producing molecules uniformly in the associated pools. During a reaction, carbons transported from a reactant molecule to a product molecule stay together all the time. We model reversible reactions as two separate unidirectional reactions. Metabolites that are produced by two or more reactions are called the *junction metabolites* of the network.

While the reaction rates and the pool sizes of the system may dynamically change in general, we will concentrate in this paper on the important case of steady state analysis. In a steady state of the system, the reaction rates and pool sizes stay invariant over time. Some metabolites are considered external to the system in the sense that there is an incoming or outgoing flow of molecules between the outside world and the corresponding pool. The other metabolites are internal.

When modelling the flow of atoms, we restrict the consideration to the *carbon atoms* of the metabolites; the measured data will in practice concern the carbons. The flow of carbon atoms from the reactant metabolites to the product molecules is given in detail by the carbon map of each reaction. Each metabolite is treated simply as a set of its uniquely named carbon locations. For example, pyruvate CH3COCOOH has three such locations.

Metabolic Networks

We let $M_1, \ldots, M_m$ be the different *metabolites* of the system. Each M_i is written as $M_i = (c_{i1}, \ldots, c_{i|M_i|})$ where the c_{ij}'s are the carbon locations of M_i. This gives a template for the molecules of M_i. The available molecules of M_i in the metabolic system constitute the molecule pool of M_i.

A *metabolic reaction* is represented as $\rho = (\alpha, \lambda)$ where $\alpha = (\alpha_1, \ldots, \alpha_m) \in \mathbf{Z}^m$ gives the *stoichiometric coefficients* $\alpha_i \in \mathbf{Z}$ of the reaction, and λ is the *carbon mapping* describing the transitions of carbon atoms in a reaction event. Coefficients α_i indicate the usage of different metabolites in an event of ρ: For each $\alpha_i < 0$, an event consumes $|\alpha_i|$ molecules of M_i; for each $\alpha_i > 0$ an event produces α_i molecules of M_i; and for each $\alpha_i = 0$, metabolite M_i does not participate in ρ. Metabolites M_i with $\alpha_i < 0$ and $\alpha_i > 0$ are called the *reactants* and *products* of ρ, respectively. Mapping λ is a relation between the carbon locations of the reactant and product molecules of ρ such that $(c, c') \in \lambda$ indicates that an reaction event of ρ takes the carbon in the reactant molecule location c to the product molecule location c'.

To simplify many considerations we assume that all reactions have a *simple stoichiometry*, that is, all stoichiometric coefficients are in $\{-1, 0, 1\}$, and all carbon mappings are one-to-one relations. Then we may also use the function notation $\lambda(F)$ to denote the carbon mapping image of any set F of reactant carbon locations. Generalization of our techniques to unrestricted stoichiometries as well as to symmetric molecules introducing non one–to–one carbon mappings will be briefly considered at the end of Section 5.

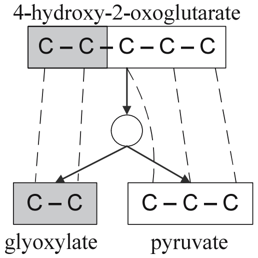


Fig. 1. An example of a metabolic reaction. In 4-hydroxy-2-oxoglutarate glyoxylate-lyase reaction a 4-hydroxy-2-oxoglutarate ($C_5H_6O_6$) molecule is split into pyruvate ($C_3H_4O_3$) and glyoxylate ($C_2H_2O_3$) molecules. Carbon maps are shown with dashed lines. The gray fragment of 4-hydroxy-2-oxoglutarate is equivalent to glyoxylate and the white fragment is equivalent to pyruvate.

Summarized, we define a *metabolic network* as a triplet $G = (\mathcal{M}, \mathcal{M}_E, \mathcal{R})$ where $\mathcal{M} = \{M_1, \ldots, M_m\}$ is the set of the metabolites, $\mathcal{M}_E \subseteq \mathcal{M}$ is the set of the *external metabolites*, and $\mathcal{R} = \{\rho_1, \ldots, \rho_n\}$ is the set of the reactions of G. Each metabolite consists of carbon locations $M_i = (c_{i1}, \ldots, c_{i|M_i|})$. Each reaction ρ_j is of the form $\rho_j = (\alpha_j, \lambda_j)$ where $\alpha_j = (\alpha_{j1}, \ldots, \alpha_{jm})$ gives the stoichiometric coefficients (in $\{-1, 0, 1\}$) and λ_j the one-to-one carbon mapping of ρ_j. Metabolites $\mathcal{M} \setminus \mathcal{M}_E$ are called *internal metabolites.*

Flux Balance Equations

The *state* of the network G is described by fixing the *reaction velocities* $v_j \geq 0$ for each reaction $\rho_j \in \mathcal{R}$ as well as the velocities β_j of the external reactions importing ($\beta_j \geq 0$) or exporting ($\beta_j < 0$) molecules to or from the metabolite pool of M_j. An external velocity β_j can be non-zero only for external metabolites $M_j \in \mathcal{M}_E$. Hence the state is given by vectors $V = (v_1, \ldots, v_n)$ and $B = (\beta_1, \ldots, \beta_m)$.

A velocity v_j gives the number of reaction events of ρ_j per time unit. As the stoichiometry is assumed simple, v_j gives in fact the number of molecules of each metabolite that are consumed/produced in a time unit by the reaction. The velocities are often called the *fluxes* of the network. The external velocities β_j similarly give the number of molecules transported to/from the metabolite pool.

Fig. 2 depicts an example metabolic network. The metabolites are A, B, C, D, and E, and the reactions (the reactants on the left-hand side and the products on the right-hand side of the arrow)

$$\rho_1 : \mathrm{A} \rightarrow \mathrm{B} + \mathrm{C} \qquad \rho_2 : \mathrm{B} \rightarrow \mathrm{D} \qquad \rho_3 : \mathrm{C} \rightarrow \mathrm{E} \qquad \rho_4 : \mathrm{B} + \mathrm{D} \rightarrow \mathrm{E}$$

Metabolites A and E are external. The arrow entering A illustrates the external inflow of A and the arrow leaving E the external outflow of E. Fig. 1 gives an example of a metabolic reaction with carbon maps.

A state (V, B) of a metabolic network G is a *steady state*, if the sizes of the metabolite pools stay invariant when the network is continuously running with

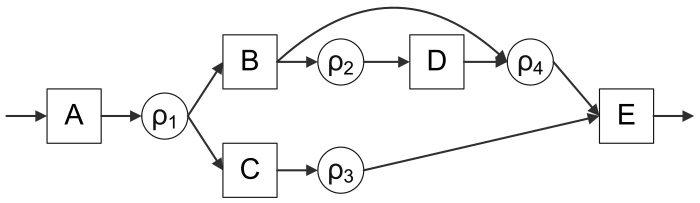


Fig. 2. Example metabolic network

the given velocities (V, B). Hence in a steady state the incoming and outgoing fluxes for each metabolite pool should be in balance, the balance equations for metabolites M_i yielding a linear system

$$\sum_{j=1}^{n} \alpha_{ji} v_j = \beta_i \qquad (i = 1, \ldots, m). \tag{1}$$

The balance equations (1) of our example network in Fig. 2 are

$$v_1 = v_2 + v_4 \qquad v_1 = v_3 \qquad v_2 = v_4$$

for the internal metabolites B, C, and D, respectively.

From the system (1) of balance equations one can completely solve the fluxes V if the external fluxes B are known from measurements and the metabolic network has a simple topology, which has no cycles or alternative pathways with independent fluxes between metabolites. (A metabolic network in Fig. 2 has a simple topology despite alternative pathways producing metabolite E: in every steady state the fluxes through these alternative pathways are equally strong). Unfortunately, real metabolic networks do not have simple topologies. Thus, the system (1) is left under-determined which means that for some fluxes we only can give linear constraints rather than a point solution. This problem can be tackled with isotopic tracing experiments, as described in the next section.

3 Flux Analysis Using Isotopomer Data

Isotopic tracing measurements can be used for relieving the problem of under-determination of the system (1). The metabolic network is fed with metabolites that are mixtures of different isotopic variants of the original metabolite. The variants, called *isotopomers*, have in certain locations the carbon isotope ^{13}C instead of the naturally more abundant ^{12}C. Measuring the isotopomer distributions of some intermediate metabolites makes it possible to add new equations to the system (1).

To define isotopomers formally, let $(c_1, \ldots, c_k)$ be the carbon locations of a metabolite M, and let $b = (b_1, \ldots, b_k) \in \{0, 1\}^k$ be a binary sequence of length k. Then a molecule of M belongs to the *b–isotopomer* of M if the molecule has carbon ^{13}C in its carbon location c_i when $b_i = 1$ and carbon ^{12}C when $b_i = 0$, i.e., the 1's of sequence b give the labelling pattern by the ^{13}C. We denote the b–isotopomer by $M(b)$.

We also need isotopomers restricted to some fragments of M, defined by subsets $F \subseteq \{c_1, \dots, c_k\}$ of the carbon locations of M. Let $F = (f_1, \dots, f_h)$ be such a fragment, and let $b = (b_1, \dots, b_h) \in \{0,1\}^h$. Then a molecule of M belongs to the *$F(b)$–isotopomer* of M if the ^{13}C labelling of locations in F follow the pattern b, i.e., if $c_i = f_j$ for some j, then the molecule has ^{13}C in location c_i if $b_j = 1$ and ^{12}C if $b_j = 0$. A fragment F of M is denoted as $M|F$ and the $F(b)$–isotopomer as $M|F(b)$.

The *isotopomer distribution*

$$D(M) = (P(M(0,\dots,0)), P(M(0,\dots,0,1)), \dots, P(M(1,\dots,1))) \in [0,1]^{2^{|M|}}$$

of metabolite M gives the relative abundances $P(M(b))$ of each isotopomer $M(b)$ in the pool of M. Hence $\sum_{b=(0,\dots,0)}^{(1,\dots,1)} P(M(b)) = 1$. Isotopomer distribution $D(M|F)$ of fragment $M|F$ is defined analogously. Hence $P(M|F(b_1,\dots,b_h))$ gives the relative abundance of molecules having carbon pattern $(b_1,\dots,b_h)$ in locations belonging to F, i.e.,

$$P(M|F(b_1,\dots,b_h)) = \sum_{(a_1,\dots,a_k)|F=(b_1,\dots,b_h)} P(M(a_1,\dots,a_k))$$

where $(a_1,\dots,a_k)|F$ denotes the restriction of binary sequence $(a_1,\dots,a_k)$ to locations that belong to F.

In order to use the isotopomers to improve flux estimation, we need to make some further assumptions about the metabolic system. First, we assume that the isotopomer pools are completely mixed with each other, and that the reactions draw their reactant molecules randomly and independently in accordance with their isotopomer distributions (i.e., the carbon labels do not introduce any bias to the random behaviour of the reactions). Second, we assume a stronger form of a steady state, called the *isotopomeric steady state*. In such a state, in addition to the metabolite pools as a whole, the isotopomer distributions of pools stay constant over time. Intuitively, cell (population) reaches isotopomeric steady state when it is kept in a metabolic steady state while feeding it with nutrients whose isotopomer distribution is kept constant. Given enough time, every practically relevant metabolic system reaches isotopomeric steady state [30].

Given the above assumptions we can write a version of (1) separately for each isotopomer of the metabolite, or, in general, for any linear combination of isotopomers [17]. If the pathways leading to the junction manipulate the carbons of the metabolite differently, often at least some of the equations are linearly independent. Consequently, the fluxes will be better determined.

Denote by M_{ij} the *subpool* of the pool M_i that contains the molecules produced by the inflow from reaction ρ_j. In particular, let M_{i0} denote the subpool produced by the external inflow ($\beta_i \geq 0$) or by the external outflow ($\beta_i < 0$) of M_i. As metabolite molecules produced by different reactions are mixed in a metabolite pool it is not possible to directly measure isotopomer distributions of inflow subpools. However, the isotopomer distributions of inflow subpools can be (in some cases) derived from the isotopomer distributions of the reactants of reactions ρ_j (see Section 4). All flux estimation methods that are based on tracing isotopomers, in some form or another, rely on this fact. All outflows from M_i use the resulting mixture pool which we denote by M_i. For convenience we will use the subpool notation $M_{ij'}$ also for the outflow to a reaction $\rho_{j'}$ and for the external outflow (then $j' = 0$), although there are no separately identifiable outflow subpools. In the outflow case $M_{ij'}$ just refers to the entire pool M_i.

Using the relative abundances of the b–isotopomer in the flows adjacent to M_i, the equation (1) gets the form

$$\sum_{j=1}^{n} \alpha_{ji} v_j P(M_{ij}(b)) = \beta_i P(M_{i0}(b)). \tag{2}$$

This can be written not only for a single b–isotopomer but for any fixed combination of them. For an $M_i|F(b)$–isotopomer, which is the union of the disjoint full–length isotopomers whose restriction to F equals b, we thus obtain the balance

$$\sum_{j=1}^{n} \alpha_{ji} v_j P(M_{ij}|F(b)) = \beta_i P(M_{i0}|F(b)). \tag{3}$$

To write equations (2) and (3) we need to know the isotopomer distributions of every subpool M_{ij} of a metabolite M_i. In practice, measurements are not available for each metabolite, and the isotopomer distributions of measured metabolites and fragments are only partially determined. To counteract this problem, in [17] a method was developed where NMR or MS measurements, given as linear combinations $D_k = \sum_b s_b P(M_k(b))$ of isotopomers could be propagated up and downstream in the metabolic network towards junctions. This propagation resulted in generalized isotopomer balance equations

$$\sum_{j=1}^{m} \alpha_{ji} v_j d_j = \beta_i d_0 \tag{4}$$

for the junctions where coefficients d_j depend on measured values D_k.

A shortcoming of the method of [17] is that the propagation of information stops at the nearest junction metabolite. Hence, for example, if in between two junctions there are no measurements, (4) cannot be formed and fluxes around that junction remain undetermined.

4 Flux Analysis Using Fragment Equivalences

In this section we develop a data-flow analysis method for finding metabolite fragments (of different metabolites) that are equivalent in the sense that they have the same isotopomer distribution, irrespective of the fluxes. Any isotopomer information known for one fragment can be propagated to the equivalent fragment, even if the fragments are on opposite sides of a junction. This leads to more efficient and accurate propagation of isotopomer data than with the method given in [17]. Equivalence of fragments can also be utilized in the design of labelling experiments as described in Section 5.

Equivalent Sets of Fragments

Metabolite fragments $M|F$ and $M'|F'$ are *equivalent* in network G, denoted $M|F \equiv M'|F'$, if there is a permutation π of the carbon locations of F' such that in all isotopomeric steady states of G

$$D(M|F) = D(M'|\pi(F')).$$

The equivalence of fragments is obviously a transitive relation.

We now develop techniques for finding sets of fragments that are equivalent in this sense. By analyzing the flow of carbon atoms locally, within one reaction, it is easy to find equivalent fragments, namely the fragments whose atoms stay intact through the reaction. To derive the basic rule, consider some reaction $\rho_j = (\alpha_j, \lambda_j)$. Let $M|F$ be a fragment of a reactant metabolite M of ρ_j. As mentioned in Section 2, we assume that ρ_j transports the carbons of F from M to a product metabolite M_i together, without splitting them to separately treated groups (this would create mixtures of subfragments taken from different molecules destroying our simple argument). Reaction ρ_j samples its input pools randomly, and hence $M|F$ is sampled according to the distribution $D(M|F)$. As the atoms of F go through ρ_j together, its image $\lambda_j(F)$ is an integral fragment of M_i and still has the original isotopomer distribution of $M|F$ in the inflow subpool M_{ij} of M_i, independently of the rest of the molecule it belongs to. We only need to permute the carbon locations of $\lambda_j(F)$ according to the carbon map λ_j to get the components of the distribution in the same order as they are given for F.

This justifies our basic observation concerning fragments in inflow subpools.

Fact 1. *Let $M|F$ be a fragment of a reactant metabolite M of a reaction $\rho_j = (\alpha_j, \lambda_j)$. If $\lambda_j(F)$ is a fragment of a single product metabolite M_i of ρ_j, then $M|F \equiv M_{ij}|\lambda_j(F)$.*

The basic step of Fact 1 will be our tool for finding more equivalences.

Construction of Equivalence Sets

We will use the following sufficient condition to find equivalent fragments: If a fragment F gets all its carbons from a fragment E such that

1. on each reaction path from E to F all carbons are transported together without cleaving them; and
2. the induced carbon map from E to F does not depend on the path followed,

then F is equivalent to E. Note that if a cleaving to separate subfragments takes place, the fragment will not necessarily carry its original isotopomer distribution because the separate molecules containing the subfragments are sampled independently by the reactions.

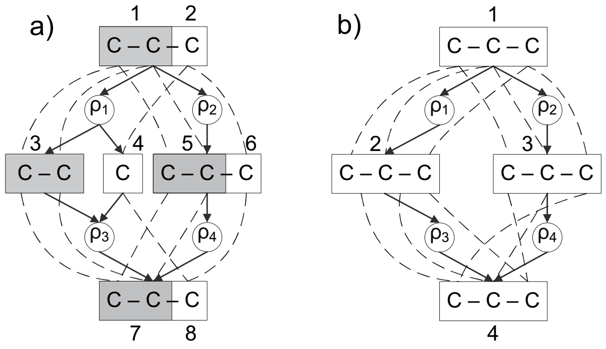


Fig. 3. a) An example of equivalence that contains junction fragment and its reactants. Carbon maps are showed with dashed lines. Junction fragment 7 is produced by two unbranched pathways (ρ_1, ρ_3) and (ρ_2, ρ_4) from precursor fragment 1. Composite carbon mappings from 1 to 7 are the same in both pathways, so $1 \equiv 3 \equiv 5 \equiv 7$. Also, $2 \equiv 4 \equiv 6 \equiv 8$. b) An example of a weak domination. Metabolite 4 is produced by two pathways from 1 without cleaving any bonds. Thus 1 weakly dominates 4. However, composite carbon mappings from 1 to 4 in pathway (ρ_1, ρ_3) are different from the composite carbon mappings of pathway (ρ_2, ρ_4). Thus 1 does not dominate 4.

First we observe that if in Fact 1 the metabolite M_i has only one inflow, then $M_{ij} = M_i$ and hence $M|F \equiv M_i|\lambda_j(F)$.

Next consider junction metabolites that have more than one inflow. To extend fragment equivalence to the junctions, we check whether or not a fragment originates from a common source along all routes to the junction. The carbons should stay together on each route, and to preserve the distribution, the carbon maps along different routes should be identical. Fig. 3 a) gives an example of a junction metabolite whose fragments originate intact from a common source. We use data-flow analysis techniques to find sets of equivalent fragments in the entire metabolic network. The *fragment flow graph* $\mathcal{F}(G) = (V, W)$ of the metabolic network $G = (\mathcal{M}, \mathcal{M}_E, \mathcal{R})$ is a directed graph defined as follows. The set V of the nodes consists of

N1. all fragments (i.e., subsets of carbon locations) of all metabolites of G; and
N2. the root node Δ which formally is a one-carbon fragment.

As the nodes of $\mathcal{F}(G)$ are metabolite fragments, the nodes will sometimes also be called fragments. For two nodes E and F we may write $E \subset F$ if E is a subfragment of F.

The set W of the arcs model the flow of carbon atoms, when grouped in fragments, in the network G. There is in W

A1. a directed arc (F, F') for each reaction $\rho_j = (\alpha_j, \lambda_j) \in \mathcal{R}$ that uses the metabolites of F and F' such that the carbon mapping λ_j satisfies $\lambda_j(F) = F'$ (hence the graph is actually a multigraph possibly with more than one arc between the same pair of nodes);
A2. an arc (Δ, F) for each fragment F of an external input metabolite; and

A3. an arc (Δ, F) for all fragments F, $F \neq \Delta$, if there exists a reaction $\rho_j \in \mathcal{R}$ that maps at least two carbons from different reactant metabolites to F.

It should be obvious that the arcs by rule A1 model the local equivalence preserving step of Fact 1 while the arcs (Δ, F) created by rules A2 and A3 need more explanation. These arcs model the unknown inflow to fragment F. In the case of rule A2, F is a fragment of an external input metabolite, and hence the arc models the inflow from outside. In the case of rule A3, fragment F gets, in addition to the inflow given by the arcs due to rule A1, some inflow that combines smaller fragments of two or more separate reactants to build the entire F. This combination event violates the sufficient condition given at the beginning of this section and we can safely ignore modelling the detailed structure of this inflow and model it just by the generic arc (Δ, F).

If we assume that all other metabolites of a network must be producible from the external input metabolites the fragment flow graph is connected.

Lemma 1. *Let every internal metabolite $M \in (\mathcal{M} \setminus \mathcal{M}_E)$ of a metabolic network $G = (\mathcal{M}, \mathcal{M}_E, \mathcal{R})$ be producible from external input metabolites $\mathcal{M}_E$ by reactions $\mathcal{R}$. Now in the fragment flow graph $\mathcal{F}(G)$ there exists a path from the root Δ to every other node.*

Proof. As every internal metabolite $(\mathcal{M} \setminus \mathcal{M}_E)$ is producible from external metabolites $\mathcal{M}_E$ there exists the smallest set $\mathcal{R}^M \subseteq \mathcal{R}$ of reactions producing fragment $M|F$ from external metabolites $\mathcal{M}_E$ for every fragment $M|F$ of every internal metabolite M. Let us first assume that F is transported intact from some $E \subseteq M_E \in \mathcal{M}_E$ in $\mathcal{R}^M$. From the construction of $\mathcal{F}(G)$ we see that now there exists a path in $\mathcal{F}(G)$ from E to F (rule A1). As Δ is connected with arc to E (rule A2) there exists a path from Δ to F in $\mathcal{F}(G)$. Let us then assume that F is not transported intact from any $E \subseteq M_E \in \mathcal{M}_E$ in $\mathcal{R}^M$. Now $\mathcal{R}^M$ defines a path $p = (p_1, p_2)$ to $\mathcal{F}(G)$. Subpath p_2 ends to F and starts from fragment $M'|H$ that is transferred intact to F by $\mathcal{R}^M$ but is not produced from any single reactant metabolite of reactions of $\mathcal{R}^M$ (here we allow $H = F$). In other words, there does not exist a reaction $\rho_j = (\alpha_j, \lambda_j) \in \mathcal{R}^M$ such that $\lambda_j(J) = H$ for any fragment J. By rule A3 there now exists an arc p_1 from Δ to H. Now the path $p = (p_1, p_2)$ connects Δ to F. □

In the rest of this paper we assume that all internal metabolites of a network must be producible from the external metabolites.

Dominator analysis on the flow graph $\mathcal{F}(G)$ will be used as a tool in equivalence search. A node E of $\mathcal{F}(G)$ *weakly dominates* another node F if all directed paths from the root Δ to F go through E. Node E is an *immediate weak dominator* of F, iff

(i) $F \neq E$;
(ii) E weakly dominates F;
(iii) E does not weakly dominate any other weak dominator of F.

The *weak dominator tree* of $\mathcal{F}(G)$ is a tree with the same node set V as $\mathcal{F}(G)$ and with an arc from each node to its immediate weak dominator. The weak dominator tree can be constructed in linear time [8]; in our implementation we have used the O(e log n) algorithm of [12] where e is the number of edges and n is the number of nodes. The flow graph $\mathcal{F}(G)$ has $n = O(m2^N)$ nodes where m is the number of metabolites and N denotes the maximum number of carbons in a metabolite.

We show next Theorem 1 stating that if E weakly dominates F, $E \neq \Delta$, then F gets all its carbons via E in uncleaved packages. Let $(e_0 = \Delta, e_1, \ldots, e_k)$ be a directed path in $\mathcal{F}(G)$ such that each e_i is a single–carbon node and $e_k \subset F$. We call such a path a *carbon path* from root to F. The carbon path is *consistent* with a path from E to F in $\mathcal{F}(G)$, if $\mathcal{F}(G)$ has a path $(E_0 = E, E_1, \ldots, E_t = F)$ such that $e_{k-t} \subset E, e_{k-t+1} \subset E_1, \ldots, e_{k-1} \subset E_{t-1}$, that is, towards its end the carbon path follows carbons that belong to a path from E to F.

From Fact 1, one-to-one carbon mappings of reactions and the construction of $\mathcal{F}(G)$ it immediately follows:

Fact 2. *Let a k–carbon fragment E weakly dominate another k–carbon fragment F, and let $C_1, \ldots, C_k \subset E$ and $D_1, \ldots, D_k \subset F$ be the corresponding single–carbon nodes. Let T be a path from from E to F. Then there is a carbon path from each C_i to some D_j that is consistent with path T. These k paths are separate and together they transport the carbons of E to the carbons of F as an uncleaved package.*

Furthermore, all the carbons of fragment F must originate from its weak dominator.

Lemma 2. *Let E weakly dominate F in $\mathcal{F}(G)$. Then every carbon path from the root to F is consistent with some path from E to F.*

Proof. If a carbon path $p = (e_0 = \Delta, e_1, \ldots, e_k)$ is not consistent with any path from E to F, then there must be a node $E' \neq E$ such that path p is consistent with a path from E' to F, and in p there is an arc from carbon $e'' \in E''$ that is not consistent with any path from E to F to some carbon of E'. From that it follows that E'' cannot reside in any path from E to F. From Lemma 1 we know that E'' is connected to root Δ in $\mathcal{F}(G)$. This means that not every path from Δ to F must go via E, that is, E does not weakly dominate F. □

By Lemma 2, all carbons to F must come via its weak dominator E. As the paths from E to F contain a separate 'track' for each carbon of E, the carbons are transported together by the reactions (Fact 2). We have:

Theorem 1. *If a k–carbon fragment E, $E \neq \Delta$, weakly dominates F then F gets all its carbons from E in uncleaved k–carbon packages.*

The weak dominance (which is the standard dominance relation of data flow analysis [1]) is not yet enough as it does not check the consistency of the carbon maps along alternative paths. A stronger form of dominance captures the proper concept as follows. We augment the arcs of our flow graph $\mathcal{F}(G)$ with the corresponding carbon maps: with each arc (F, F') we associate the corresponding λ_j

restricted to the carbons of F and F', and with each arc (Δ, F) we associate the trivial one–to–many carbon mapping that relates the single carbon of Δ with each carbon of F.

We say that a node E of $\mathcal{F}(G)$ *dominates* another node F if

(i) E weakly dominates F, and
(ii) the composite carbon map from E to F, which is induced by the arc–wise carbon maps of some directed path in $\mathcal{F}(G)$ from E to F, is the same for all such paths, i.e., every path from E to F gives the same mapping of the carbons of E to the carbons of F.

Immediate domination and *dominator tree* are defined as the corresponding weak versions. Note that for nodes representing one–carbon fragments the dominance and the weak dominance coincide. Fig. 3 b) gives an example of a metabolic network where a metabolite weakly dominates another metabolite but does not dominate that same metabolite.

The following theorem gives a method for converting weak dominator trees into dominator trees.

Theorem 2. *Let E be the immediate weak dominator of F in $\mathcal{F}(G)$. If the immediate weak dominators of the nodes that correspond to the carbons of F are nodes that correspond to the carbons of E, then E is the immediate dominator of F. Otherwise Δ is the immediate dominator of F.*

Proof. Assume first that the immediate weak dominator D_i of a carbon C_i of F is a carbon of E, and this is true for every carbon C_i of F. Then $D_i \neq D_j$ if $C_i \neq C_j$; otherwise some carbon of E would not map onto F contradicting the assumption that E is the immediate weak dominator of F. But this means that the (one–to–one) mapping (D_i, C_i) between the carbons of E and F is the only one induced by the paths from E to F, and hence E is also the immediate dominator of F.

Assume then that some D_i is not a carbon of E. Then D_i must be a carbon of some weak dominator of $E \neq \Delta$ and, as there is no weak dominator of C_i in the carbons of E, at least two different carbons of E must map to C_i. So the carbon map between E and F is not one–to–one but depends on the path. Then only Δ can be the immediate dominator of F as then the maps trivially become path independent. Any other node above E can not be a dominator as the paths to F must go via E (Lemma 2) and hence the composite carbon maps depend on the path taken between E and F. □

Theorem 2 leads to the following algorithm for constructing a dominator tree.

1. Construct the flow graph $\mathcal{F}(G)$ and the weak dominator tree T for it;
2. Transform the weak dominator tree T into dominator tree by performing the test of Theorem 2 for each carbon of each fragment.

There are $O(mN2^N)$ carbons to be tested in Step 2. Each test can be done in constant time using the dominator tree for single-carbon nodes that is contained in the large weak dominator tree produced in Step 1. The running time of the tests becomes also $O(mN2^N)$. This is the running time of the entire procedure as the constructions of Step 1 can be done by the method of [8] in time linear in the number of the nodes in the graph, which is inferior to the time of Step 2.

Lemma 3. *Let a node F be a fragment of metabolite M and node F' a fragment of metabolite M'. If F is an immediate dominator of F' in $\mathcal{F}(G)$, then $M|F \equiv M'|F'$.*

Proof. By Theorem 1 F' gets all its carbons from F in uncleaved packages. Thus molecule fragments F' retain their labelling patterns in all paths from F to F' in G. From the definition of dominance relation we know that the induced carbon map from F to F' does not depend on the path followed. □

Theorem 3. *If nodes $F = M|F$ and $F' = M'|F'$ belong to the same subtree of the root Δ in the dominator tree of $\mathcal{F}(G)$, then $M|F \equiv M'|F'$ under the permutation of carbon locations of F' that is induced by the composite carbon maps from the root of the subtree to F and to F'.*

Proof. Theorem follows directly from Lemma 3 and from the transitivity of equivalence. □

Fig. 4 gives an example network with carbon maps. Five subtrees of the dominator tree are also shown, representing equivalence classes of fragments. The two trees for the one–carbon subfragments of the other trees are omitted.

Finally we describe an alternative way to construct dominator subtrees and the corresponding equivalent fragment sets.

The full fragment flow graph $\mathcal{F}(G)$ constructed in the above method can be quite large: for each metabolite M there are $O(2^{|M|})$ nodes in $\mathcal{F}(G)$. It is sufficient to build the fragment flow graph and dominator tree only for fragments corresponding one and two carbon fragments. Now there will be only $O(|M|^2)$ nodes in $\mathcal{F}(G)$ for each metabolite M. Let T be the dominator tree of the new fragment flow graph. To find dominance subtree T_F for a fragment F larger than 2, take some subfragment $E = (a, b) \subset F$ of size 2. Let T_E be the subtree of the root of T that contains E. Let $E' = (a, b') \subset F$ be another subfragment which shares carbon location a with E. Then the dominator subtree $T_{E \cup E'}$ for $E \cup E'$ can be build from the trees T_E and $T_{E'}$ with a simple intersection operation. The operation compares the two trees using the natural correspondence of nodes (the corresponding nodes have non–empty intersection) and arcs, and retains the nodes and arcs that according to this correspondence are present in both. Next we take another pair $E'' = (a, b'') \subset F$, and intersect $T_{E \cup E'}$ by $T_{E''}$ and so on, until we have T_F.

Tree T_F can be constructed from T in this way in time $O(|F|m)$ where m is the number of metabolites: The intersection operation has to be done $O(|F|)$ times, and each operation obviously takes time proportional to the sizes of the trees intersected. As the dominator subtrees contain at most one fragment from each metabolite, the trees are of size $O(m)$, and the time bound follows.

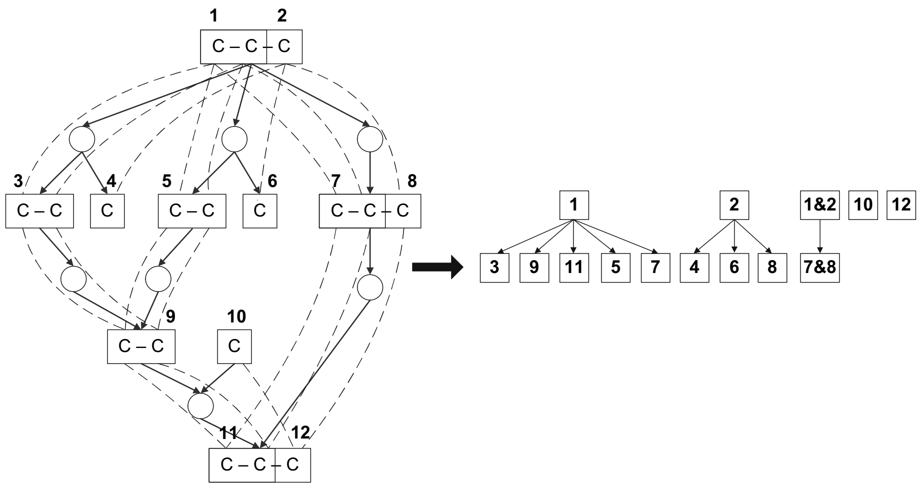


Fig. 4. Example network with carbon maps; the subtrees of the corresponding dominator tree on the right. "1&2" and "7&8" denote metabolites consisting of corresponding fragments.

Finally we note that the running time of the above algorithm can be slightly further improved by constructing the fragment flow graph only for single carbons and such two-carbon fragments that consist of neighbouring carbons in graphs representing the chemical bond structures of the carbon backbones of molecules.

5 Applications and Extensions

In this section we show how the fragment equivalence sets can be utilized in different subtasks of metabolic flux estimation. We also sketch few extensions to the basic framework introduced above.

Data Propagation for Flux Analysis

Equivalences and domination relation can be used to analyze conditions under which the isotopomer balances (3) add new constraints to the basic stoichiometric balances (1) and hence help solving the fluxes at metabolite M_i.

If M_i is not a junction metabolite (i.e., it has only one inflow), equations (3) and (1) do not differ as there is only the isotopomer distribution of the single inflow available. In this case isotopomer balances (3) do not add any new constraints.

Junctions are more interesting. By the equivalence analysis we sometimes know for sure that (3) must be linearly dependent on equations (1).

Theorem 4. *Let F be a fragment of a junction metabolite M_i. If F has a dominator $\neq \Delta$ in the fragment graph $\mathcal{F}(G)$, then equation (3) for F is equivalent to the equation (1) for M_i.*

Proof. For each reaction ρ_j that produces M_i, $F_j = \lambda_j^{-1}(F)$ (the inverse image of F in the reactants of ρ_j) must be a fragment of some reactant metabolite. As F has a dominator, fragment F as well as all fragments F_j must belong to the same subtree of Δ in the dominator tree. Hence they are equivalent by Theorem 3. This means that the abundance of an isotopomer $F(b)$ is the same in each inflow subpool M_{ij} of M_i as in the entire pool M_i. Thus $P(M_{ij}|F(b))$ has the same value for all j in Equation (3), that is, (3) equals (1). □

Theorem 4 tells us that dominated fragments of a junction metabolite are not useful in estimating fluxes at that junction because the isotopomers have the same origin and hence do not help in separating the different fluxes. In the non–dominated case the equation (3) can be independent of system (1), but its actual usefulness depends on the available measurements.

The equivalence sets facilitate efficient propagation of the measured data to the junctions. Assume that we know (by measurements) the isotopomer distributions $D(M_k|H_k)$ of some fragments H_k. We want to know what linear constraints (3) these measurements give.

We say that the measurement of H_k *covers* all fragments of the metabolic network that are equivalent to H_k. Let then M_i be a junction metabolite and $F = M_i|F$ a fragment of M_i. Assume that the isotopomer measurements of fragments H_k cover F, and for each reaction ρ_j producing M_i, the measurements also cover the reactant fragment E_j such that $\lambda_j(E_j) = F$. It then follows from the equivalence of fragments that the measurement data, when permuted suitably according to the carbon maps, gives the isotopomer distribution of F and of each E_j. Using Fact 1 the distributions can be further propagated from each E_j to the inflow subpools $M_{ij}|F$ of the fragment F of M_i. Hence we know the values of the relative abundances $P(M_{ij}|F(b))$ in the linear equations (3) for M_i, i.e., we have enough data to include these equations to the linear system that constrain the fluxes.

Design of Labelling Experiments

Above, a sufficient condition for writing isotopomer balance equations (3) for fragment $F = M_i|F$ was described. A measurement was needed from the equivalence sets of F and from the equivalence sets of reactant fragments E_j such that $\lambda_j(E_j) = F$ for each reaction ρ_j producing M_i. This observation allows us to choose the subset of metabolites to measure that gives us the same information on metabolite labelling that would have been available by measuring every metabolite in the network. Such subset contains at least one fragment from the equivalence sets of junction fragments and from the equivalence sets of all reactant fragments of these junction fragments. In the best case the size of the subset is significantly smaller than the number of metabolites in total, reducing the cost and burden of experiments. In [16] it is shown that the problem can be formalized as an NP-hard set cover problem. Solutions to this metabolite selection problem can be found by a greedy set cover algorithm. As most metabolic network models used in flux estimation are quite small, also global optimization techniques like mixed integer linear programming can be applied without increasing computational time prohibitively. By combining the selection of metabolites to measure with equivalence sets dependent on input labelling, it could be possible to simultaneously design good input labellings and an optimal set of metabolites to measure.

Checking the Consistency of a Network Model and Measurement Data

Fragment equivalence sets can also be utilized to make simple consistency checks to measurement data. If the isotopomer distributions of two equivalent fragments differ, we know that either our model of a metabolic network is incomplete or measurements are corrupted. Thus fragment equivalence sets can be used to find errors from the topology of the model network and to study the repeatability and the effect of metabolite concentration to the accuracy of the measurements. For example, if measured isotopomer distributions of fragments are clearly unconsistent with equivalence sets, we must augment our network with reactions that partition violated equivalence sets to subsets that are consistent with measurements.

The fragment equivalence sets can also be used to enumerate useful targets for METAFoR analysis [23].

Symmetric Metabolites and General Stoichiometries

A metabolite is called *symmetric* if it has several indistinguishable numberings of its carbons. We assume that such numberings are explicitly given. For example, a 6–carbon cyclic molecule could have numberings (123456, 345612, 561234), and a 3–carbon centrally symmetric molecule numberings (123, 321).

We can model symmetries in our formalism by adding new reactions to the system. Let M be a symmetric metabolite with t carbons. Then introduce two versions of M: the basic version M^0 and 'symmetrized' version M^s. For each item $I = (i_1, \ldots, i_t)$ in the numberings list of M, add reaction which takes one M^0 and produces one M^s and maps the carbons of M^0 to the carbons of M^s according to I (the jth carbon goes to location i_j). The other reactions that refer to M should now use M^0 if M is a product and M^s if M is a reactant of the reaction.

General stoichiometries that have also other coefficients than -1, 0, and 1 can be treated in similar fashion. If a reaction consumes or produces more than one molecule of the same metabolite, introduce formal reactions that divide a metabolite pool into several 'copies' or combine several copies into one. The carbon maps should be expanded accordingly.

Fixed Input Labelling

In the fragment equivalence sets introduced above the isotopomer distributions of two fragments belonging to the same sets are equivalent regardless of fluxes and labelling of external metabolites. If the labelling of external metabolites is fixed, some equivalence sets can be merged. Merging can be done if it can be shown that the isotopomer distributions of fragments in two equivalence sets are equivalent when a given input labelling is used. Such larger equivalence sets for a given input labelling $\mathcal{L}$ can be found with a slight modification to the construction of the fragment flow graph. For each isotopomer distribution found from the subfragments of external metabolites a "labelling node" is added to the fragment flow graph. Labelling nodes are connected to Δ and to all subfragments of external metabolites having a corresponding isotopomer distribution. Then dominator subtrees are constructed. Now fragments dominated by the labelling nodes are equivalent when input labelling $\mathcal{L}$ is used.

6 Experiments

We tested our method for finding fragment equivalence sets with a model of the central carbon metabolism of *Saccharomyces cerevisiae* containing glycolysis, penthose phosphate pathway and citric acid cycle. Carbon mappings were provided by the ARM project [2]. The network consisted of 31 metabolites and 33 reactions of which 17 were bidirectional. Bidirectional reactions were modelled as two unidirectional reactions. The rank of the linear equation system (1) for the 50 fluxes of the model network was 30. Cofactor metabolites not accepting or donating carbon atoms were excluded from the analysis. The only carbon source of the network was glucose. There were 22 metabolites that were produced by more than one reaction and thus formed junctions.

Equivalence Sets

Our analysis discovered seven nontrivial equivalence sets that had more than one member. In our existing method for flux estimation [17], that does not utilize flow analysis, the propagation of isotopomer information always stops to junction metabolites meaning that the equivalence of fragments that reside in different sides of a junction remains undetected. Thus without flow analysis there in general exists greater number of smaller equivalence sets. The size distributions of the equivalence sets obtained with and without flow analysis are given in Table 1. The flow analysis uncovered 18 dominated maximal fragments, that is, fragments that were not subsets of other dominated fragments. Nontrivial equivalence sets covered 76 carbon atoms out of 133 carbons in the system. Dominated fragments of junction metabolites contained 27 carbons, undominated junction fragments 67 carbons.

Efficiency of the Propagation of Isotopomer Data

We compared the efficiency of the propagation of isotopomer information by flow analysis against our earlier method that does not utilize flow analysis [17]. We selected randomly and uniformly a set of metabolites considered as measured and tested how many carbon locations could in the best case get isotopomer information from the measured metabolites with both methods. Technically the number

Table 1. Number of fragments in nontrivial equivalence sets in the model of central carbon metabolism of *S. cerevisiae* with and without utilizing the flow analysis

# of fragments	2	3	4	5	9	Σ
# of eq. classes (flow analysis)	1	1	2	2	1	7
# of eq. classes (without flow analysis)	6	2				8

of locations equals the total amount of carbons in all equivalence sets that have at least one fragment from the measured metabolite. The sampling was repeated 10000 times for each number of selected metabolites. The mean and standard deviations of the number of carbon locations reached are given in Fig. 5. The computational experiment shows that when the amount of measurable metabolites is limited, as usually is the case in real experiments, our new method can propagate isotopomer information more efficiently than the previous one.

We also tested the effect of improved propagation to the amount of information obtained from the flux distribution with another simulation experiment. We again selected measured metabolites randomly and uniformly. We assumed that if a metabolite was selected as measured, the positional enrichments, that is, labelling degrees of each individual carbon of the metabolite were available. This assumption about the quality of measurement data leads to situation where generalized isotopomer balance equations (4) will be formed for each undominated junction carbon separately. We then counted the difference between the total number of balance equations writable for undominated junction carbons (see Theorem 4) with and without the flow analysis in the propagation of measurement data. The number of writable balance equations gives us an upper limit to the independent constraints available for flux distribution. Thus the higher the number, the more information about the flux distribution is potentially available. The maximum number of writable balance equations was 67 as there were that many undominated junction carbons in the network. The test was repeated 10000 times for each number of selected metabolites.

For most randomly selected sets of measured metabolites flow analysis was not able to greatly increase the number balance equations constructed. However, with some sets increase in the number of available equations was substantial. For example, measuring the positional enrichments of five metabolites (d-6-phospho-gluono 1,5-lactone, 1,3-bisphospho-d-glyceric acid, d-erythrose 4-phosphate, d-fructose 6-phosphate, d-fructose 1,6-biphosphate) from the model network gave in total six balance equations for two junctions (d-fructose 6-phosphate, d-fructose 1,6-biphosphate) when flow analysis was utilized while no equations were obtained without the flow analysis. Differences in the number of balance equations obtained with and without the flow analysis when 18 metabolites were randomly selected as measured is given in Table 2. With the other numbers of selected metabolites differences between the amounts of equations were approximately equal or smaller. The given differences cannot be negative as, compared to our earlier method, the use of flow analysis will never decrease the number of obtained balance equations.

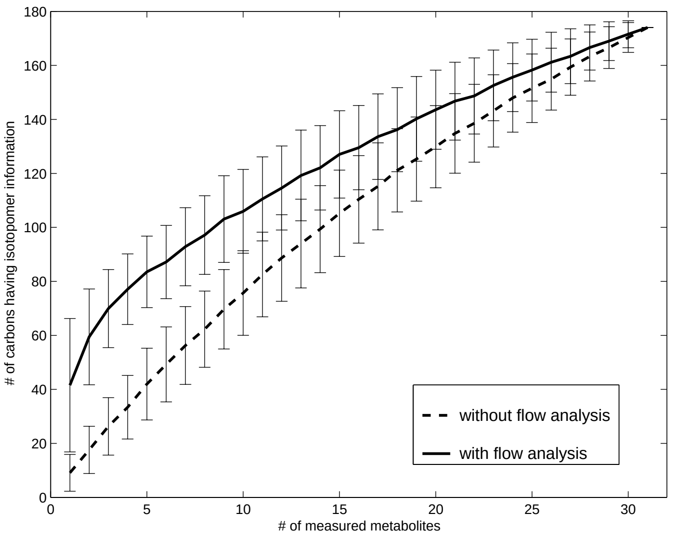


Fig. 5. Efficiency of propagation of isotopomer information with the flow analysis method of this article and with the method of [17]. Y-axis gives the mean number of carbon locations that can get isotopomer information from the metabolites randomly selected as measured. Total lengths of the symmetric error bars equal two standard deviations.

Table 2. Differences in the number of isotopomer balance equations obtained with and without the flow analysis when positional enrichment information of 18 randomly selected metabolites is available (10000 repeats). The maximum number of balance equations is 67.

difference	0–1	2–3	4–5	6–7	8–9	10–11	12–13	14–15	16–17	18–19	20–21	22–23
instances	4494	1835	1306	1108	550	229	230	154	64	24	4	2

Selection of Metabolites to Measure

The results of simulations where measured metabolites were selected randomly show that the number of constraints obtained to a flux distribution depends heavily on the selection of metabolites. The development of measurement protocols is laborious so it is worthwhile to carefully design experiments such a way that the flux distribution is uncovered as completely as possible by measuring as few metabolites as possible. As sketched at the end of previous section, fragment equivalence sets can be utilized in such experimental planning. By using the experiment planning method developed recently [16] we compared the sizes of minimal sets of measured metabolites that will give us maximal information about the fluxes when positional enrichments of each metabolite were assumed to be measurable. If the flow analysis was not applied, it was necessary to measure 23 metabolites to write as many balance equations, 67 in total, as the measurement of every metabolite would have made possible. When the flow analysis was applied, it was sufficient to measure 20 metabolites to write the maximum number of useful balance equations for undominated carbons.

The requirement that every undominated carbon of every junction metabolite receives a balance equation (4) is a conservative one. Each junction M can have at most as many linearly independent balance constraints as there are reactions producing M. As the stoichiometric balance equation (1) already gives us one constraint we can require only (*the number of producing reactions - 1*) of equations (4) for each junction. If these balance equations are linearly independent they give maximal information about the flux distribution. If only (*the number of producing reactions - 1*) of equations (4) for each junction were required it was enough to construct only 22 instead 67 balance equations (4) in our example network. If the flow analysis was not applied we still had to measure 23 metabolites to write these equations. With flow analysis it was sufficient to measure 18 metabolites.

Visualization of the metabolic network used in the experiments is available at http://www.cs.helsinki.fi/group/sysfys/images/tcsb_metabolic_net.pdf.

7 Discussion

In this article we have presented a novel method for computing sets of fragments of metabolites that have identical carbon isotopomer distributions, regardless of the fluxes of the metabolic network. The analysis is based on the flow analysis of the metabolic fragments and draws from the domination analysis techniques [12]. These equivalence sets have several uses. Firstly, they enable better propagation of isotopomer data in the metabolic network, thus making direct flux estimation more feasible. Secondly, isotopomer measurements can be planned more accurately with this method, as redundancies between measurements become explicit. The same flux information can be obtained with sometimes substantially fewer measurements. Thirdly, the equivalence sets enable one to detect errors in network topology and measurements.

There are several interesting routes for further research. Utilizing the notion of weak dominance for improved propagation and design of optimal isotopomer labellings with the aid of the equivalence sets are our immediate goals.

Acknowledgments. We thank Markus Heinonen and Arto Åkerlund for their contribution to the experimental work reported in this paper. This work has been supported by the SYSBIO programme of Academy of Finland, under the SYSFYS project (project number 203668), NEOBIO programme of TEKES and by Marie Curie Individual Fellowship grant HPMF-CT-2002-02110.

References

1. A. Appel. *Modern Compiler Implementation in Java.* Cambridge University Press, 1998.
2. M. Arita. In silico atomic tracing of substrate-product relationships in escherichia coli intermediary metabolism. *Genome Research*, 13:2455–2466, 2003.

3. L. Boros, N. Serkova, M. Cascante, and W-N. Lee. Use of metabolic pathway flux information in targeted cancer drug design. *Drug Discovery Today: Therapeutic Strategies*, 1(4):435–443, 2004.
4. B. Christensen and J. Nielsen. Isotopomer analysis using GC-MS. *Metabolic Engineering*, 1:E8–E16, 1999.
5. W. Eisenreich, J. Slaghuis, Laupitz R., J. Bussemer, J. Stritzker, C. Schwarz, R. Schwarz, T. Dankekar, W. Goebel, and A. Bacher. ^{13}c isotopologue perturbation studies of listeria monocytogenes carbon metabolism and its modulation by the virulence regulator prfa. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 103:2040–2045, 2006.
6. E. Fisher, Zamboni N., and U. Sauer. High-throughput metabolic flux analysis based on gas chromatography-mass spectrometry derived ^{13}C constraints. *Analytical Biochemistry*, 325:308–316, 2004.
7. S. Ghosh, T. Zhu, I.E. Grossmann, M.M. Ataai, and M.M. Domach. Closing the loop between feasible flux scenario identification for construct evaluation and resolution of realized fluxes via nmr. *Journal of Bacteriology*, 183:1441–1451, 2001.
8. D. Harel. A linear algorithm for finding dominators in flow graphs and related problems. In *Proceedings of the 17th annual ACM symposium on Theory of computing*, pages 185–194, 1985.
9. N. Isermann and W. Wiechert. Metabolic isotopomer labeling systems. part ii: structural identifibiality analysis. *Mathematical Biosciences*, 183:175–214, 2003.
10. J. Kelleher. Flux estimation using isotopic tracers: Common ground for metabolic physiology and metabolic engineering. *Metabolic Engineering*, 3:100–110, 2001.
11. S. Klamt and S. Schuster. Calculating as many fluxes as possible in underdetermined metabolic networks. *Molecular Biology Reports*, 29:243–, 2002.
12. T. Lengauer and R. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1:121–141, 1979.
13. A. Marx, A. de Graaf, W. Wiechert, L. Eggeling, and H. Sahm. Determination of the fluxes in the central metabolism of corynebacterium glutamicum by nuclear magnetic resonance spectroscopy combined with metabolite balancing. *Biotechnology and Bioengineering*, 49:111–129, 1996.
14. M. Möllney, W. Wiechert, D. Kownatzki, and A. de Graaf. Bidirectional reaction steps in metabolic networks IV: Optimal design of isotopomer labeling systems. *Biotechnology and Bioengineering*, 66:86–103, 1999.
15. J. Nielsen. It is all about metabolic fluxes. *Journal of Bacteriology*, 185, 2003.
16. A. Rantanen, T. Mielikäinen, J. Rousu, H. Maaheimo, and E. Ukkonen. Planning optimal measurements of isotopomer distributions for estimation of metabolic fluxes. *Bioinformatics*, Advance Access, February 27 2006. In print.
17. J. Rousu, A. Rantanen, H. Maaheimo, E. Pitkänen, K. Saarela, and E. Ukkonen. A method for estimating metabolic fluxes from incomplete isotopomer information. In *Computational Methods in Systems Biology, Proceedings of the First International Workshop, CMSB 2003*, volume 2602 of *Lecture Notes in Computer Science*, pages 88–103, 2003.
18. K. Schmidt, M. Carlsen, J. Nielsen, and J. Viladsen. Modeling isotopomer distributions in biochemical networks using isotopomer mapping matrices. *Biotechnology and Bioengineering*, 55:831–840, 1997.
19. H. Schwarz. *Numerical Analysis: A Comprehensive Introduction.* John Wiley & Sons, 1989.
20. V. Selivanov, J. Puigjaner, A. Sillero, J. Centelles, A. Ramos-Montoya, P. Lee, and M. Cascante. An optimized algorithm for flux estimation from isotopomer distribution in glucose metabolites. *Bioinformatics*, 20:3387–3397, 2004.

21. A. Sola, H. Maaheimo, K. Ylönen, P. Ferrer, and T. Szyperski. Amino acid biosynthesis and metabolic flux profiling of pichia pastoris. *FEBS Journal*, 271:2462–2470, 2004.
22. G. Stephanopoulos, A. Aristidou, and J. Nielsen. *Metabolic engineering: Principles and Methodologies.* Academic Press, 1998.
23. T. Szyperski. Biosynthetically directed fractional ^{13}C-labelling of proteinogenic amino acids. *European Journal of Biochemistry*, 232:433–448, 1995.
24. T. Szyperski, R. Glaser, M. Hochuli, J. Fiaux, U. Sauer, J. Bailey, and K. Wütrich. Bioreaction network topology and metabolic flux ratio analysis by biosynthetic fractional ^{13}C labeling and two-dimensional NMR spectrometry. *Metabolic Engineering*, 1:189–197, 1999.
25. J.J. Vallino and G. Stephanopoulos. Metabolic flux distribution in corynebacterium glutamicum during growth and lysine overproduction. *Biotechnology and Bioengineering*, 41:633–646, 1993.
26. W. van Winden, J. Heijnen, P. Verheijen, and J. Grievink. A priori analysis of metabolic flux identifiability from (13)c-labeling data. *Biotechnology and Bioengineering*, 74:505–516, 2001.
27. A. Varma and B.O. Palsson. Metabolic flux balancing: basic concepts, scientific and practical use. *Nature Biotechnology*, 12:994–998, 1994.
28. W. Wiechert. Modeling and simulation: tools for metabolic engineering. *Journal of Biotechnology*, 94:37–63, 2002.
29. W. Wiechert, M. Möllney, S. Petersen, and A. de Graaf. A universal framework for ^{13}C metabolic flux analysis. *Metabolic Engineering*, 3:265–283, 2001.
30. W. Wiechert and M. Wurzel. Metabolic isotopomer labeling systems part i: global dynamic behavior. *Mathematical Biosciences*, 169:173–205, 2001.
31. C. Wittmann and E. Heinzle. Mass spectrometry for metabolic flux analysis. *Biotechnology and Bioenginering*, 62:739–750, 1999.

Multiple Representations of Biological Processes

Carolyn Talcott[1] and David L. Dill[2]

[1] SRI International
clt@csl.sri.com
[2] Stanford University
dill@cs.stanford.edu

Abstract. This paper describes representations of biological processes based on Rewriting Logic and Petri net formalisms and mappings between these representations used in the Pathway Logic Assistant. The mappings are shown to preserve properties of interest. In addition a relevant subnet transformation is defined, that specializes a Petri net model to a specific query to reduce the number of transitions that must be considered when answering the query. The transformation is shown to preserve the query in the sense that no answers are lost.

Keywords: Signal transduction, biological process, Pathway Logic, Rewriting Logic, Petri Net.

1 Introduction

Pathway Logic [1,2,3] is an approach to modeling cellular processes based on formal methods. In particular, formal executable models of processes such as signal transduction, metabolic pathways, and immune system cell-cell signaling are developed using the rewriting logic language Maude [4,5] and a variety of formal tools are used to query these models. An important objective of Pathway logic is to reflect the ways that biologists think about problems using informal models, and to provide bench biologists with tools for computing with and analyzing these models that are natural.

Using the reflective capabilities of Maude, several alternative representations are derived to support use of different tools for visualization and analysis. The Pathway Logic Assistant (PLA) manages these different representations and, using the IOP+IMaude framework [6], provides a user interface that supports visualization and interaction with the models, and access to tools such as the Pathalyzer for carrying out in silico experiments. In particular, PLA uses Petri nets which provide visual representations and algorithms for answering reachability queries interactively, displaying the results in a way that is natural for biologists.

Being able to use alternative representations with different expressive capabilities and tools for visualization and analysis is important both for managing complexity and being able to focus on different properties. In the presence of multiple representations it is crucial to be able move between representations in semantically meaningful ways, preserving relevant properties. In this paper we describe a class of rewriting logic models of biological processes and a mapping of these models to Petri net models. We show that this mapping preserves computations and satisfaction of temporal formulae. Finally, we describe transformations that specialize Petri net models to specific queries by reducing the set of transitions that need to be considered, and show that transformations are safe in the sense that query results are not changed. Specializing simplifies a Petri net model and allows the user to focus attention on the part of the model relevant to the question under consideration.

Plan. The remainder of this paper is organized as follows. Related work is discussed in §2. To provide context and motivate the technical results, a brief overview of Pathway Logic and the ways that biologists can compute with and query Pathway Logic models is given in section 3. In section 4 the notion of occurrence-based rewrite theory is defined and mappings between such theories and Petri net models are defined and shown correct. The notion of subnet relevant for a particular query is introduced in section 5, and transformations for producing a safe approximation to the relevant subnet are defined. Section 6 concludes with a summary and discussion of future directions.

2 Related Work

Computational models of biological processes such as signal transduction fall into two main categories: differential equations to model kinetic aspects; and symbolic/logical formalisms to model structure, information flow, and properties of processes such as what events (interactions/reactions) are checkpoints for or consequences of other events.

Models of system kinetics based on differential equations use experimentally derived or inferred information about concentrations and rates to simulate changes in response to stimuli as a function of time [7,8,9,10]. Such models are crucial for rigorous understanding of, for example, the biochemistry of signal transduction. However, the creation of such models is impeded by the great difficulty of obtaining accurate intra-cellular rate and concentration information, and by the possibly stochastic nature of cellular scale populations of signaling molecules [11,12]. Analysis of such models by numerical and probabilistic simulation techniques becomes intractable as the number of reactions to be considered grows [13]. Furthermore, for the present purpose the questions we want to ask of a model involve qualitative concepts such as causality and interference rather than detailed quantitative questions.

Symbolic/logical models allow one to represent partial information and to model and analyze systems at multiple levels of detail, depending on information available and questions to be studied. Such models are based on formalisms that provide language for representing system states and mechanisms of change such as reactions, and tools for analysis based on computational or logical inference. Symbolic models can be used for simulation of system behavior. In addition properties of processes can be stated in associated logical languages and checked using tools for formal analysis. A variety of formalisms have been used to develop symbolic models of biological systems, including Petri nets [14], the pi-calculus [15], statecharts [16], and rule-based systems such as rewriting logic [17]. Each of these formalisms was initially developed to model and analyze computer systems with multiple processes executing concurrently. Several tools for finding pathways in reaction and interaction network graphs have been developed. However as pointed out in [18], paths found in these graphs do have not much to do with biochemical pathways.

There are many variants of the Petri net formalism and a variety of languages and tools for specification and analysis of systems using Petri nets. Petri nets have a graphical representation that corresponds naturally to conventional representations of biochemical networks. They have been used to model metabolic pathways and simple genetic networks (e.g., see [19,20,21,22,23,24,25,26]). These studies have been largely concerned with dynamic or kinetic models of biochemistry. In [27] a more abstract and qualitative view is taken, mapping biochemical concepts such as stoichiometry, flux modes, and conservation relations to well-known Petri net theory concepts.

A pi-calculus model for the receptor tyrosine kinase/mitogen-activated protein kinase (RTK/-MAPK) signal transduction pathway is presented in [28]. BioSPI, a tool implementing a stochastic variant of the pi-calculus, has been used to simulate both the time and probability of biochemical reactions [29]. So far, symbolic/logical analysis tools have not be used to analyze BioSPI models.

In [30] a continuous stochastic logic and the probabilistic symbolic model checker, PRISM, is used to express and check a variety of temporal queries for both transient behaviors and steady state behaviors. Proteins modeled as synchronous concurrent processes, and concentrations are modeled by discrete, abstract quantities.

BioAmbients [31], an adaptation of the Ambients formalism for mobile computations has been developed to model dynamics of biological compartments. BioAmbient type models can be simulated using an extension of the BioSPI tool. A technique for analysis of control and information flow in programs has been applied to analysis of BioAmbient models [32]. This can be used, for example, to show that according to the model a given protein could never appear in a given compartment, or a given complex could never form.

Statecharts naturally express compartmentalization and hierarchical processes as well as flow of control among subprocesses. They have been used to model T-cell activation [33,34]. Although Statecharts is a mature technology with a number of associated analysis and verification tools, it does not appear that these have been applied to the T-cell model. Live Sequence Charts [35] are an extension of the Message Sequence Charts modeling notation for system design. This approach has been used to model the process of cell fate acquisition during C.elegans vulval development [36].

P-systems is a multiset rewriting formalism that provides a built in notion of location. A continuous variant of P-systems is used in [37] to model intra-cellular signaling. Locations are used to represent compartmental structure of a cell. Abstract objects represent proteins and small molecules, with different objects used to represent different modifications / states of the same protein. The underlying relation between a protein and its modifications is not made explicit. A system state specifies the quantity of each object in each location. A rate function associates to each rule a function from system states to real numbers, representing the rate of the reaction in that state. This determines how a system state evolves over time. Such models can be used to predict concentration of objects, for example phosphorylated ERK, over time by a discrete step approximation method.

A simple formalism for representing interaction networks using an algebraic rule-based approach very similar to the Pathway Logic approach is presented in [38,39].

The language has three interpretations: a qualitative binary interpretation much like the Pathway Logic models; a quantitative interpretation in which concentrations and reaction rates are used; and a stochastic interpretation. Queries are expressed in a formal logic called Computation Tree Logic (CTL) and its extensions to model time and quantities. CTL queries can express reachability (find pathways having desired properties), stability, and periodicity. Techniques for learning new rules to achieve a desired system specification are described in [40].

BioSigNet (BSN) [41] is a system for representing and reasoning about signaling networks. A BSN knowledge base encodes knowledge about a signal network, including logical statements based on symbols termed fluents and actions. Fluents represent the various properties of the cell and its components while actions denote biological processes (e.g. biochemical reactions, protein interactions) or external interventions. The logical statements describe the impact of these actions on the fluents, how actions can be triggered or inhibited inside the cell. A BSN knowledge base is queried using a temporal logic language over propositions expressing presence or absence of particular fluents. Three classes of queries are identified: prediction (can a state be reached); explanation (find initial conditions that lead to a specified condition); and planning (determining when an action should occur in order to achieve a desired result). In [42] BSN is used to model the ERK signaling network.

Models that rely on quantitative information (BioSPI, PRISM, P-systems) are limited by the difficulty in obtaining the necessary rate data. Missing or inconsistent data (from experiments carried out under different conditions, and on different cell types) are likely to yield less reliable predictions. Models that abstract from quantitative details avoid this problem, but the abstractions may lead to prediction of unlikely behavior, or miss subtle interactions.

The Pathway Logic Assistant extends the basic representation and execution capability with the ability to support multiple representations, to use different formal tools to simplify and analyze the models, and to visualize models and query results. Other efforts to integrate tools for manipulating models include the Systems Biology Workbench [43] the Biospice Dashboard [44], IBM Discoverylink [45], and geneticXchange, Inc [46].

3 Pathway Logic

As mentioned above, Pathway Logic models of biological processes are developed using the Maude system [4,5] a formal language and tool set based on rewriting logic. Rewriting logic [17] is a logical formalism that is based on two simple ideas: states of a system are represented as elements of an algebraic data type; and the behavior of a system is given by local transitions between states described by *rewrite rules*. The process of application of rewrite rules generates computations (also thought of as deductions). In the case of biological processes these correspond to paths. Using reflection, modules and computations are represented as terms of the Maude meta language. This makes it easy to compute with models and paths.

3.1 Pathway Logic Basics

Pathway Logic models are structured in four layers: (1) sorts and operations, (2) components, (3) rules, and (4) queries. The *sorts and operations* layer defines the main sorts, subsort relations, and operations for representing cell states. The sorts of entities include `Chemical`, `Protein`, `DNA`, `Complex`, and `Enclosure` (cells and other compartments). These are all subsorts of the sort, `Soup`, that represents 'liquid' mixtures, as multisets. The sort `Dish` is introduced to encapsulate a soup as a state to be observed. Post-translational protein modification is represented by terms of the form `[P - mods]` where `P` is a protein and `mods` is a set of modifications. Modifications can be abstract, just specifying being activated, bound, or phosphorylated, or more specific, such as, phosphorylation at a particular site. For example, the term `[Cas - act]` represents the activation of the protein `Cas`. A cell state is represented by a term of the form `{CM | cm { cyto }}` where `cm` stands for a soup of entities in or at the cell membrane and `cyto` stands for a soup of entities in the cytoplasm.

The *components* layer specifies particular entities (proteins, chemicals, DNA) and introduces additional sorts for grouping proteins in families. For example `ErbB1L` is declared to be a subsort of `Protein`. This is the sort of `ErbB1` ligands whose elements include the epidermal growth factor `EGF`. The *rules* layer contains rewrite rules specifying individual signal transduction steps representing processes such as activation, phosphorylation, complex formation, or translocation. The *queries* layer specifies initial states and properties of interest.

Below we give a brief overview of the representation in Maude of signal transduction processes, illustrated using a model of Rac1 activation. This model and several others are available as part of the Pathway Logic Demo available from the Pathway Logic web site `http://pl.csl.sri.com/` along with papers, tutorial material and download of the Pathway Logic Assistant tool.

3.2 Modeling Activation of Rac1 in Pathway Logic

Rac1 is a small signaling protein of the Ras superfamily. It functions as a protein switch that is "on" when it binds the nucleotide triphosphate GTP, and "off" when it binds the hydrolysis product GDP. The Pathway Logic model of Rac1 activation was curated using [47] and many other references (cited as metadata associated with individual rules). In the following we show an initial state for study of Rac1 activation and two example rules, and briefly sketch some of the ways one can compute with the model. The initial state (called `rac1demo`) is a dish `PD( ... )` with a single cell and two stimuli in the supernatant, `EGF` and `FN`, represented by the following term.

```
rac1demo = PD(FN EGF
{CM | EGFR Ia5Ib1 Src PIP2[Actin - poly][HRas - GDP][Rac1 - GDP]
        {Crk2 Erk2 Mek1  PI3K Shp2 bRaf C3g Dock Sos1
         Cas E3b1 Elmo Eps8 Fak Gab1 Grb2 Vav2 }} )
```

The cell membrane (shown on the line beginning `CM`) has an EGF receptor (`EGFR`) and an integrin (`Ia5Ib1`) that binds to `FN`. The term `[Rac1 - GDP]` represents the Rac1 protein in its 'off' state. The cell cytoplasm (shown on the last two lines) contains additional proteins that participate in the signaling process.

One way to activate `Rac1` begins with the activation of the `EGFR` receptor due to the presence of the `EGF` ligand. The following rule represents this signaling step.

```
rl[1.EGFR.is.act]:
   ?ErbB1L:ErbB1L {CM | cm  EGFR        {cyto }} =>
   ?ErbB1L:ErbB1L {CM | cm [EGFR - act] {cyto }} .
   *** ErbB1Ls are AR EGF TGFa Btc Epr HB-EGF
```

The term `?ErbB1L:ErbB1L` is a variable ranging over the sort `ErbB1L`. The terms `cm` and `cyto` are variables standing for the remaining components in the membrane and cytoplasm, respectively. The rule matches a part of the `rac1demo` dish contents by binding the variable `?ErbB1L:ErbB1L` to `EGF`, the variable `cm` to `Ia5Ib1 ...[Rac1 - GDP]` (every thing in the cell membrane except `EGFR`), and the variable `cyto` to the contents of the cytoplasm `{Crk2...Vav2}`. Applying the rule replaces `EGFR` by `[EGFR - act]` resulting in the dish

```
  PD(FN EGF
    {CM | [EGFR - act] Ia5Ib1 Src PIP2 [Actin - poly]
          [HRas - GDP][Rac1 - GDP]
        {Crk2 Erk2 Mek1  PI3K Shp2 bRaf C3g Dock Sos1
         Cas E3b1 Elmo Eps8 Fak Gab1 Grb2 Vav2}} )
```

The following is one of three rules characterizing conditions for the Rac1 switch to be turned on.

```
rl[256.Rac1.is.act-3]:
 {CM | cm [Cas - act][Crk2 - act][Dock - act] Elmo [Rac1 - GDP]
      {cyto }} =>
 {CM | cm [Cas - act][Crk2 - act][Dock - act] Elmo [Rac1 - GTP]
      {cyto }} .
```

This rule describes activation resulting from assembly of `Elmo` with activated `Cas`, `Crk2`, and `Dock` at the cell membrane. Executing the rule replaces `[Rac1 - GDP]` by `[Rac1 - GTP]`, turning Rac1 on, and leaves the remaining components unchanged.

Maude provides several ways to compute with a model. One can rewrite an initial state such as `rac1demo` above, to see a possible final state, or search for all states satisfying some predicate. To find a path satisfying some (temporal logic) property the Maude model-checker can be used. The properties of interest for Pathway Logic are expressed in Maude as patterns matching states with specific proteins, possibly with modifications, occurring in particular compartments (called goals), or requiring that particular proteins do not appear (avoids).

Example 1: racAct3 Property. As an example, to find the path stimulated by `FN` alone, we define a property (called `racAct3`) that is satisfied when Rac1 is activated and the EGF stimulus is not used (`EGFR` is not activated), thus forcing the FN stimulus to be selected. The property `racAct3` is axiomatized by assertions stating which dishes satisfy the property (the relation `|=`) using patterns such as the following.

```
ceq PD (out:Soup
        {CM | cm [Rac1 - GTP] {cyto}}) |= racAct3 = true .
    if not(cm has [EGFR - act])
```

The model-checker is asked to check the assertion that there is no computation satisfying this property and a path can be extracted from a counterexample if one is found.

3.3 The Pathway Logic Assistant

The textual representation of cell states and pathways quickly becomes difficult to use as the size of a model grows, and an intuitive graphical representation becomes increasingly important. In addition, it becomes important to take advantage of the simple structure of PL models when searching for paths and carrying out other analyses. A Pathway Logic model, such as the Rac1 model, meeting certain simple conditions, can be transformed into a Petri net model by specializing the rules to the model's initial state. Petri nets have a natural graphical representation and can be analyzed using special purpose analysis tools.

Our Petri net models are a special case of Place-Transition Nets given by a set of occurrences (places in Petri net terminology) and a set of transitions [48]. Occurrences can be thought of as atomic propositions asserting that a protein (in a given state) or other component occurs in a given compartment. A system state is a set of occurrences (called a marking in Petri net terminology), giving the propositions that are true. A transition is a pair of sets of occurrences. A transition can fire if the state contains the first set of occurrences. In which case the first set of occurrences is replaced by the second set. PL goal properties translate to Petri net properties expressed as occurrences that must be present (places to be marked) and avoids properties translate to occurrences that must not appear (places not to be marked) in a computation. Paths leading from an initial state to a state satisfying a set of goals can be represented compactly as a Petri net consisting of the transitions fired in the path, thus giving query results a natural graphical representation. Execution of the path net starting with the initial state, leads to a state satisfying the goals, and the net representation makes explicit the dependency relations between transitions: some can fire concurrently (order doesn't matter), and some require the output of other transitions to be enabled.

The Pathway Logic Assistant (PLA) manages the different model and computation representations and provides functions for moving from one representation to another, for answering user queries, displaying and browsing the results. The principle data structures are: PLMaude models, Petri net models, Petri subnets, PNMaude modules, computations (paths), and Petri graphs. Here we give an overview of the PLMaude and Petri net models and mappings between them, illustrated by the model of Rac1 activation. Details are given in the next section, where we define the notion of *occurrence-based rewrite theory* that abstracts the relevant features of PLMaude models, and specify the main properties required for mappings between and transformations of representations in order to preserve meaning. *Petri graphs* are used to represent Petri nets as data structures that have a natural visual representation. A Petri graph has two kinds of node: occurrence and rule. Edges connect nodes representing occurrences of a rule premise (lhs) to the rule node and the rule node to the nodes representing occurrences of the rule conclusion (rhs).

PLMaude models are Maude modules, such as the modules specifying the model of Rac1 activation discussed in §3.2, having the four layer structure described in §3.1. A

PLMaude model must also obey the *conservation law* that components can be modified, composed and decomposed, but are not created out of nothing. As discussed in §3.1-3.2, PLMaude states are represented as a mixture of cells and ligands where location of proteins and other chemicals is represented by the algebraic structure of a term. To make the Petri net structure explicit an alternative representation is defined using multisets of occurrences. An occurrence is a pair consisting of a protein, complex, or other chemical component and a location. The location uniquely identifies the position of the component within the algebraic term (modulo multiset equality). For example, the dish

```
PD(EGF {CM | EGFR {Erk2}})
```

is represented by the occurrences

```
< EGF,out > < EGFR,cm > < Erk2,cyto >
```

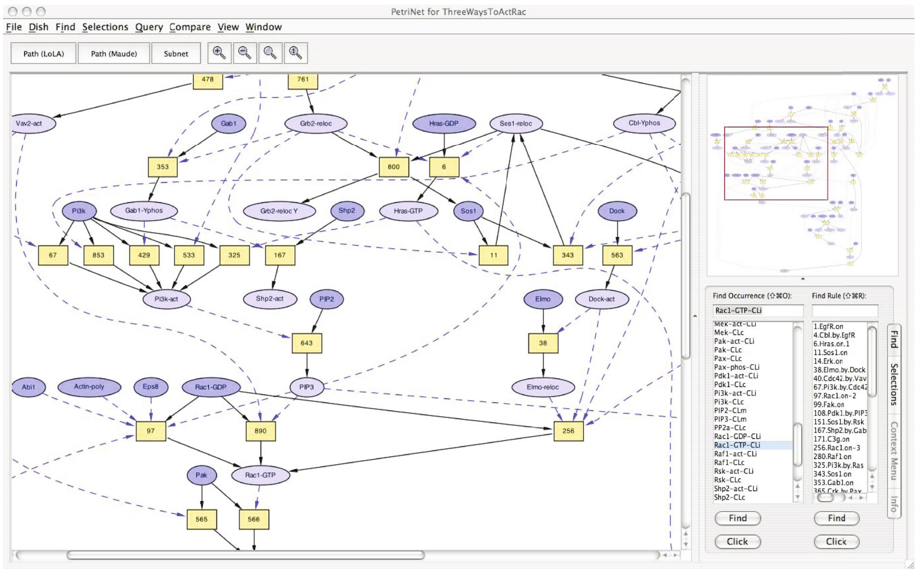
Although soups and occurrences are formalized as multisets, initial states contain only sets (no duplication) and we have required that PLMaude rules preserve this property.

A *Petri net model* is a pair (T, I) consisting of a set of transitions T, and an initial state I (a set of occurrences). Each transition consists of a rule identifier, a pair of occurrence sets (the pre-occurrences and the post-occurrences). The mapping of a PLMaude model, with a specified initial dish D, to a Petri net model first determines an upper approximation to the set of components that might occur in each dish location by a collecting operation. This is done by starting with D, and repeating the collection cycle until nothing new is collected. In the collection cycle, for each rule that can be applied to the current dish, the result of applying the rule is merged into the current dish (by adding any new components to each compartment). For example, applying the rule `[1.EGFR.is.act]` to the dish `rac1demo` in collecting mode would add `[EGFR -act]` to the membrane rather using it to replace `EGFR`. The set of transitions T is then the set of rule instances that apply to the collected dish, converted to occurrence pairs. For example the rule `[1.EGFR.is.act]` instantiated with `EGF` for `?ErbB1L:ErbB1L` is represented by the triple

```
(1.EGFR.is.act, < EGF, out > < EGFR, cm >,
                < EGF, out > < [EGFR - act], cm >)
```

The initial state I is the conversion of the dish D to the occurrence representation. Figure 1 shows the Petri net representation of the model of Rac1 activation produced by this mapping.

A *Petri subnet* is a tuple (T, I, G, A) consisting of a set of transitions, T, an initial marking, I a goal marking G, and an avoids set A. A Petri subnet specifies an analysis problem, namely finding a computation starting from the initial marking, and reaching a state with the goals marked, using the transitions in the given set, without ever marking an avoid. Petri subnets are generated by a ‘relevant subnet’ computation that simplifies the specified analysis problem. Although a Petri subnet is a Petri net, it is only equivalent to the original net for a goal set that is a subset of G or an avoid set that is a superset of A. For example, for a goal that is not in G there may be a path in the original net, but



Rac1-GTP-CLi

Mek-act-CLi	1.EgfR.on
Mek-CLc	4.Cbl.by.EgfR
Pak-act-CLi	6.Hras.on.1
Pak-CLc	11.Sos1.on
Pax-CLc	14.Erk.on
Pax-phos-CLi	38.Elmo.by.Dock
Pdk1-act-CLi	40.Cdc42.by.Vav
Pdk1-CLc	67.Pi3k.by.Cdc42
Pi3k-act-CLi	97.Rac1.on-2
Pi3k-CLc	99.Fak.on
PIP2-CLm	108.Pdk1.by.PIP3
PIP3-CLm	151.Sos1.by.Rsk
PP2a-CLc	167.Shp2.by.Gab
Rac1-GDP-CLi	171.C3g.on
Rac1-GTP-CLi	256.Rac1.on-3
Raf1-act-CLi	280.Raf1.on
Raf1-CLc	325.Pi3k.by.Ras
Rsk-act-CLi	343.Sos1.on
Rsk-CLc	353.Gab1.on
Shp2-act-CLi	365.Crk.by.Pax
Shp2-CLc	

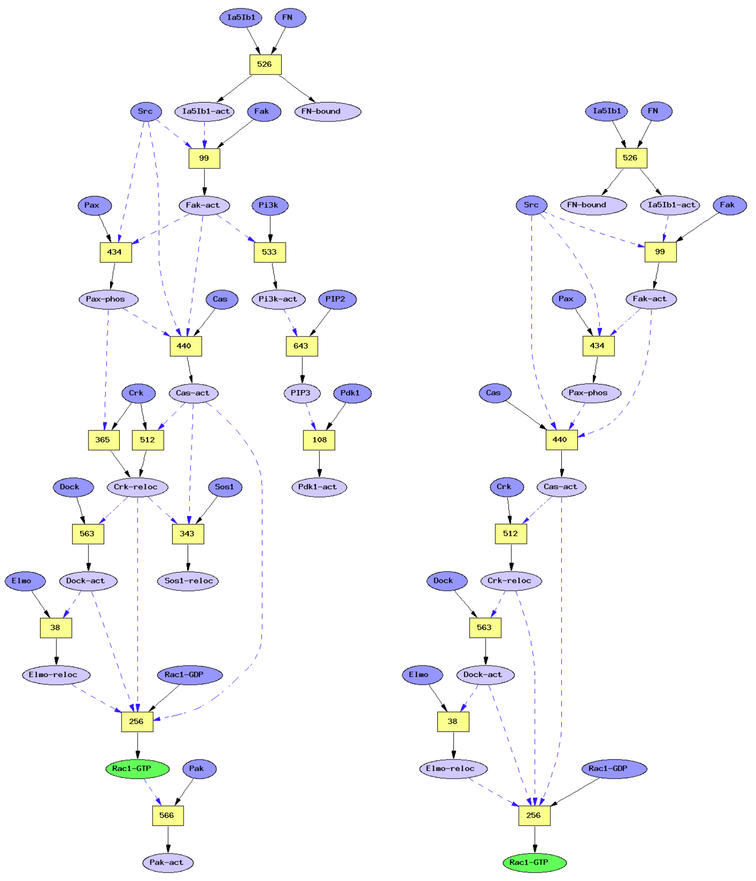
Fig. 1. Rac1 activation model as a Petri net. Ovals are occurrences, with initial occurrences darker. Rectangles are transitions. Two way dashed arrows indicate an occurrence that is both input and output. The full net is shown in the upper right thumbnail. A magnified view of the portion in the red rectangle is shown in the main view.

not in the subnet, since transitions needed for this goal may have been discarded as not being relevant. Figure 2(a) shows the relevant subnet of the Rac1 activation model when the goal is activation of Rac1, avoiding activation of EGFR (Maude property `racAct3` in Example 1 of §3.2).

Computations are data structures used to represent system executions. We model a computation as a sequence of steps, each step being a triple consisting of a source state, a rule instance or transition that applies to that state, and a target state, the state resulting from the rule application. The target state of the ith step of a computation must be equal to the source state of the $i + 1$st step. A compact representation of a computation is the Petri net consisting of the initial state together with the set of rule instances occurring as computation steps. We call this net a *path*. Figure 2(b) shows a path in the subnet of Figure 2(a). It can be executed as follows: If all of the ovals connected to a box by an incoming arrow (solid or dashed) are colored dark then color the outputs dark and make the inputs connected by solid arrows light color. Repeating this procedure, a state can be reached with Rac1 activated (Rac1-GTP colored dark).

3.4 Efficient Analysis of Petri Nets

The path shown in Figure 2(b) was found by the LoLA (Low Level Analyzer) Petri net analysis tool [49,50]. LoLA uses "stubborn set reduction", which is a technique that exploits the ease of determining the independence of certain transitions in the Petri nets. Specifically, a marking (state) m may have many possible successor markings,



(a) Subnet (b) Path

Fig. 2. FN stimulation of Rac1 activation

each resulting from a transition that is enabled in m. In many cases, transitions t_1 and t_2 are enabled in m because they are concurrent—either one can fire first, but they will both fire eventually. Nets with a lot of concurrency will also have a lot of states resulting from all the permutations of firing orders of concurrent transitions.

Stubborn set reduction prunes the state graph by recognizing concurrently enabled transitions and allowing them to be fired in a fixed order, instead of considering all the permutations. Subtle conditions must be met to ensure that a condition is reachable in the reduced graph if and only if it is reachable in the original graph (the reader is referred to [51]). For highly concurrent graphs, stubborn set reduction can accelerate the solution of the reachability problem by many orders of magnitude.

For reachability queries on Pathway Logic nets, answering a reachability query that would have taken hours using a general purpose model-checking tool takes on the order of a second in LoLA—fast enough to permit interactive use. As an example, LoLA was run on 5 examples, with and without the stubborn set reduction option turned on. The examples were 5 queries on a single Petri net, each causing exploration of a different part of the network. The experiments were conducted on a an IBM ThinkPad X22 with an 800 MHz Pentium III CPU and 256 MB of RAM. All the examples completed within a fraction of a second with stubborn set reduction turned on. With stubborn set reduction turned off 4 of the examples completed within a fraction of a second. The 5th example was stopped after 28.5 minutes. At this point LoLA had generated 2,495,854 states, traversed 35,400,000 edges, and was using 500 MB of virtual memory, and all 256MB of physical memory.

In our experience, these results are typical. Without stubborn set reduction, LoLA either finishes quickly or runs for a very long time. With stubborn set reduction, it very reliably finishes in a fraction of a second on the many examples we have tried.

It is beyond the scope of this paper to compare different model-checking tools for Pathway Logic models. The interested reader can find such a comparison in [52]. We note that for the goals and avoids type queries, LoLA's performance is by far the best.

4 Relating PLMaude and Petri Nets

It is well known that Petri nets can be represented in rewriting logic [48]. The various forms that PLMaude models have taken as the modeling ideas have matured have led us to identify a special class of rewrite theories, called *occurrence-based* rewrite theories, that, restricted to terms reachable from a given initial term, have a natural representation as Petri nets. The idea is to build on the equivalence of the dish and occurrences representations of states and to identify the features of PLMaude models needed to ensure that the translation to the Petri net formalism preserves computations and goals-avoids properties. Furthermore, the resulting Petri net models can be transformed back into rewriting logic, again preserving computations and goals-avoids properties. In this section we define the mappings between PLMaude and Petri net models and the subnet reduction, and sketch proofs of correctness. These mappings are implemented in Maude and used in PLA.

4.1 Some Rewriting Logic Notation

We first introduce some notation for talking about rewrite theories. A rewrite theory, $\mathcal{R}$, is a triple $((\Sigma, E), R)$ where (Σ, E) is an equational theory (for example, in order sorted logic) with sorts and operations given by Σ and equations E, and R is a set of rules of the form $(t_0 \Rightarrow t_1 \text{ if } c)$ where t_0, t_1 are terms, the rule premise and conclusion respectively, and c is a boolean term, the rule condition. Viewing PLMaude as a rewrite theory, (Σ, E) is given by the first two layers (sorts and operations, components) and R is given by the rules layer.

A *context*, C, is a term with a single hole, denoted by $[\,]$, used to indicate the location of a rewrite application. $C[t]$ is the result of placing t in the hole of C.

A *substitution* σ is a finite mapping from variables to terms, preserving sort, and $\sigma(t)$ is the result of applying σ to the term t.

A *rule instance* is a triple $\rho = (r, C, \sigma)$ where r is a rule, C is context, and σ is a substitution. For a rule instance ρ as above we write $t \xrightarrow{\rho} t'$ if $t = C[\sigma(t_0)]$, $t' =$

$C[\sigma(t_1)]$, and $\sigma(c)$ holds (rewrites to true). In this case we say that ρ is an application of r to t. We write $t \xrightarrow{r} t'$ if there is some $\rho = (r, C, \sigma)$ such that $t \xrightarrow{\rho} t'$. A computation over $\mathcal{R}$ is a sequence of rewrites of the form

$$\mathcal{R} \vdash s_0 \xrightarrow{\rho_1} s_1 \ldots \xrightarrow{\rho_k} s_k$$

with steps $s_{i-1} \xrightarrow{\rho_i} s_i$ for $1 \le i \le k$.

Note that rewriting is modulo E, that is the meaning of the symbol '$=$' in the matching equations is defined by the equational theory E. The context makes explicit the location within a term where the rule applies. This is needed because when rewriting modulo equations the usual notion of path to a subterm of a syntax tree is not meaningful.

Example 2: Rewriting concepts. Consider the following:

– S_0 = `EGF {CM | EGFR {Mek1 [Mekk3 - act]}}`
– S_1 = `EGF {CM | EGFR {[Mek1 - act] [Mekk3 - act]}}`
– r_{mek} = `[Mekk3 - act]  Mek1 => [Mekk3 - act] [Mek1 - act]`
– C = `EGF {CM | EGFR {[]}}`

Then $\rho = (r_{mek}, C, \emptyset)$ is rule instance (with empty substitution, $\emptyset$) such that $S_0 \xrightarrow{\rho} S_1$. Note that S_0 can also be written `EGF {CM | {Mek1 [Mekk3 - act]} EGFR}`. Syntactically the subterm that matches the rule right hand side is at a different position in this case, but modulo associativity and commutativity the two ways of writing the term have the same meaning. The corresponding context `EGF {CM | {[]} EGFR}` is also equivalent to C, thus giving a representation of position that is independent to the representation of equivalence class.

4.2 Occurrence-Based Rewrite Theories

There are five conditions to be met for a rewrite theory to be an occurrence-based rewrite theory, two conditions on the representation of state (**SC1** and **SC2**) and two conditions on rules (**RC1**, **RC2**) and one condition on the interaction of states and rules **SRC**).

In the following assume we are given a rewrite theory, $\mathcal{R}$, with distinguished sort **S** of elements representing system states to be analyzed.

SC1. The first condition is that **S** is generated from a base sort, by constructors such as the PLMaude enclosure constructors, in such a way that one only needs to know the 'location' of the base subterms to determine an element of **S**. More precisely, we require that there be a base sort **B**, a sort **L**, of locations, and a sort **O** of occurrences, where elements of **O** have the form $<b, l>$ for base element b and location l, and two functions

$$s2o(\_) : \mathbf{S} \to \mathbf{P}_\omega[\mathbf{O}] \;\text{ and }\; o2s(\_) : \mathbf{P}_\omega[\mathbf{O}] \xrightarrow{\mathrm{p}} \mathbf{S}$$

such that $o2s(s2o(s)) = s$ where $\xrightarrow{\mathrm{p}}$ denotes partial functions and $\mathbf{P}_\omega[\mathbf{O}]$ denotes finite sets from **O**.

SC2. We extend $s2o(\_)$ to contexts and terms with variables, by treating holes and variables as basic terms, and we require a function $cloc(C)$ that gives the location of the hole in a context. We also relativize the map from states to occurrences so that $s2o(t, l)$ gives the occurrences for t in a context with hole location l. Thus

$$s2o(C[t]) = O \cup (s2o(t, l)) \text{ where } l = cloc(C), s2o(C) = O \cup <[\,], l>.$$

The **SC2** requirement is that if $s2o(s_0) = O \cup s2o(\sigma(t_0), l)$ then we can find C such that $cloc(C) = l$ and $s_0 = C[\sigma(t_0)]$.

Example 3: Checking SC1, SC2 for PLMaude. In PLMaude, the base sort is called `Thing`, which has subsorts `Protein` and `Chemical` amongst others. Each membrane enclosed compartment has two associated locations, the membrane and the interior. For example, a cell has locations `cm` and `cyto`, and things not inside a cell have location `out`. Continuing the notation of Example 2 from §4.1, `EGF` has location `out` and `EGFR` has location `cm` and we have

– $s2o(S_0)$ =
  `< EGF,out > < EGFR,cm > < Mek1,cyto > < [Mekk3 - act],cyto >`
– $cloc(C)$ = `cyto`
– $s2o(S_2, cloc(C))$ = `< Mek1,cyto > < [Mekk3 - act],cyto >`

where S_2 is the left-hand side of r_{mek}. From the discussion in the previous sections, it is easy to see that PLMaude modules satisfy conditions SC1 and SC2.

SRC. We require that there is an associative and commutative operation on states $merge : \mathbf{S} \times \mathbf{S} \to \mathbf{S}$ such that rewriting is preserved by merging. Specifically, if $s_0 \xrightarrow{(r,C,\sigma)} s'_0$, $s_1 = merge(s_0, s')$ and $s'_1 = merge(s'_0, s')$ then $s_1 \xrightarrow{(r,C',\sigma')} s'_1$ for some C', σ', such that $\sigma/B = \sigma'/B$ and $cloc(C) = cloc(C')$ where σ/B is the restriction of σ to basic variables. Using associativity and commutativity we extend the $merge$ operation to sets: $merge(s, S)$ is the result of merging elements of S into s in some order.

Example 4: Merging. Continuing with the notation of example 2 we have

– $merge(S_0, S_1)$ = `EGF {CM | EGFR{Mek1 [Mek1 - act] [Mekk3 - act]}}`

RC1. We require that the variables appearing in rule terms either have basic sorts, or 'mixture' sorts (for example finite sets). This allows us to convert a rule application instance (r, C, σ) into a pair of occurrence sets that represent the actual change described by the rule. The mixture variables stand for the remaining basic terms and substructure at each location of interest that are not changed by the rule. Furthermore, we assume that the variables occurring in the rule condition have basic sorts.

Definition: Collection. Now we define a (partial) function that iteratively merges the reachable states into one state $\hat{s}$ in which each location contains all basic elements that could appear at that location in a reachable state. Given $s \in \mathbf{S}$ define $\hat{s}$ by

$$\hat{s} = s_k \text{ if } s_k = s_{k+1} \quad \text{where } s_0 = s \text{ and } s_{i+1} = merge(s_i, \{s' \mid (\exists\rho)(s_i \xrightarrow{\rho} s')\})$$

RC2. The final condition for $\mathcal{R}$ to be occurrence-based (relative to a choice of initial states) is that for any initial state s collection terminates, i.e. there is some k such that $s_k = s_{k+1}$.

4.3 Mapping Occurrence-Based Rewrite Theories to Petri Nets

To define the mapping we need some Petri net notation. A transition τ over an occurrence set **O** is a pair $(O_i, O_o) \in \mathbf{P}_\omega[\mathbf{O}] \times \mathbf{P}_\omega[\mathbf{O}]$ (for simplicity we omit the transition labels). We define the pre- and post-occurrences of a transition as follows:

$$pre(O_i, O_o) = O_i \qquad post(O_i, O_o) = O_o.$$

The input and output occurrences are the pre- and post-occurrences with the shared occurrences removed.

$$in(O_i, O_o) = O_i - O_o \qquad out(O_i, O_o) = O_o - O_i.$$

Note that

$$\begin{aligned}
&in(O_i, O_o) \cap out(O_i, O_o) = \emptyset \\
&(pre(O_i, O_o) - in(O_i, O_o)) \\
&\qquad = (post(O_i, O_o) - out(O_i, O_o)) = (pre(O_i, O_o) \cap post(O_i, O_o))
\end{aligned}$$

A Petri net model over occurrences **O** is a pair (T, I) where T is a set of transitions and $I \in \mathbf{P}_\omega[\mathbf{O}]$ is the initial state/marking. A computation over T is a sequence

$$T \vdash O_0 \xrightarrow{\tau_1} O_1 \ldots \xrightarrow{\tau_k} O_k$$

such that $pre(\tau_{i+1}) \subseteq O_i$ and $O_{i+1} = (O_i - in(\tau_{i+1})) \cup out(\tau_{i+1})$.

Definition: Rule2Transition. We extend the occurrence mapping to map rule instances to transitions.

$$s2o((t_0, t_1, c), C, \sigma) = (s2o(t_0, C, \sigma), s2o(t_1, C, \sigma))$$

where

$$s2o(t_0, C, \sigma) = s2o((\sigma/B)(t_0), cloc(C))^\dagger.$$

Where the † means to drop variable occurrences $<V, l>$ for mixture variables V. Note that if (r, C, σ) and (r, C', σ') are as in **RC2** then $s2o(r, C, \sigma) = s2o(r, C', \sigma')$. (See below for examples of $s2o(\_)$ applied to rules.)

Definition: OccB2Petri. Assume $\mathcal{R}$ is occurrence-based, with state sort **S**, and s is an initial state. The Petri net model, $\mathcal{P}(\mathcal{R}, s)$, associated with $\mathcal{R}$ and s, has occurrences $s2o(\hat{s})$ (the result of collection), initial state $s2o(s)$, and a transition for each rule instance that applies to $\hat{s}$.

$$\mathcal{P}(\mathcal{R}, s) = (T_s, s2o(s)) \quad \text{where } T_s = \{s2o((r, C, \sigma)) \bigm| (\exists s')(\hat{s} \xrightarrow{(r, C, \sigma)} s')\}$$

Example 5: A Tiny model and its Petri net representation. We now introduce a tiny hypothetical model to illustrate the transformation to a Petri net in some detail. The resulting Petri net, TinyPN, will also be used in §5 to illustrate the transformations defined there. The tiny model defines two subsorts of `Protein`, `AP` and `BP`. There are six basic proteins: `A0,A1` of sort `AP`, `B0` of sort `BP`, and `E0,E1,C` of sort `Protein`.

```
sorts AP BP  . subsort AP BP < Protein .
ops A0 A1 : AP .  op B : BP .  ops E0 E1 C : Protein .
```

There are four rules. Rule `T0` expresses activation of `A0` by `E0` and recruitment of `A0` to the cell membrane. As in §3.2 `cm` and `cyto` are variables needed in order for the rule to apply to any cell with the specified components. Rule `T1` expresses activation and recruitment of any protein of sort `AP` by `E1`. Thus a variable `?A:AP` of sort `AP` is used in the rule. Rule `T2` says that any activated protein of sort `AP` can activate any protein of sort `BP`. Rule `T3` says that activated `A0` can activate protein `C`.

```
rl[T0]: {CM | cm E0 {cyto A0}}  =>
       {CM | cm E0 [A0 - act] {cyto}}

rl[T1]: {CM | cm E1 {cyto ?A:AP}}  =>
       {CM | cm E1 [?A:AP - act] {cyto}}

rl[T2]: {CM | cm [?A:AP - act] ?B:BP {cyto}} =>
       {CM | cm [?A:AP - act] [?B:BP - act] {cyto}}

rl[T3]: {CM | cm [A0 - act]  C {cyto}} =>
       {CM | cm [A0 - act] [C - act] {cyto}}
```

The initial state of the tiny model is given by the term `tinyDish`. It contains `B,C,E0,E1` at the cell membrane and `A0,A1` in the cytoplasm.

```
tinyDish = PD({CM | B C E0 E1 {A0 A1}}) .
```

The collection starting with the dish `tinyDish` using the rules `T0,T1,T2` yields the dish `tinyDish*`. Rule `T0` adds `[A0 - act]` to the membrane compartment, rule `T1` instantiated with `?A:AP` as `A1` adds `[A1 - act]` to the membrane compartment. The instantiation of rule `T1` with `?A:AP` as `A0` adds nothing new. Rule `T2` instantiated with `?B:BP` as `B0` adds `[B0 - act]` to the membrane compartment. For the dish `tinyDish` there is only one instantiation of rule `T2`. Rule `T3` adds `[C - act]` to the membrane compartment.

```
tinyDish* = PD({CM | [A0 - act] [A1 - act] [B - act] B
                     C [C -act] E0 E1
                     {A0 A1}}) .
```

The Petri net TinyPN has transitions `tinyT`, obtained by applying $s2o(\,)$ to rule instances for `tinyDish*`, and occurrences `tinyI`, the result of $s2o($`tinyDish`$)$. Here we added labels to the transitions for convenient reference. If there is more than one rule instance, the transition labels are indexed by instance numbers, just to keep labels unique.

```
TinyPN = (tinyT,tinyI)
tinyI = < B,cm >< C,cm >< E0,cm >< E1,cm >< A0,cyto >< A1,cyto >
tinyT =
   (T0,   < E0,cm> < A0,cyto > => < E0,cm > < [A0 - act],cm >)
   (T1.0, < E1,cm> < A0,cyto > => < E1,cm > < [A0 - act],cm >)
   (T1.1, < E1,cm> < A1,cyto > => < E1,cm > < [A1 - act],cm >)
   (T2.0, < [A0 - act],cm >< B,cm > =>
          < [A0 - act],cm >< [B - act],cm >
   (T2.1, < [A1 - act],cm >< B,cm >  =>
          < [A1 - act],cm >< [B - act],cm >
   (T3,   < [A0 - act],cm >< C,cm > =>
          < [A0 - act],cm >< [C - act],cm >
```

For example there are two instances of rule `T1`, $\rho_0 = ((t_0, t_1), C, \sigma_0)$ and $\rho_1 = ((t_0, t_1), C, \sigma_1)$ where

- $t_0 =$ {`CM | cm E1` {`cyto ?A:AP`}}
- $t_1 =$ {`CM | cm E1 [?A:AP - act]` {`cyto`}}
- $C =$ `PD([])`
- σ_0/B binds `?A:AP` to `A0`
- σ_1/B binds `?A:AP` to `A1`

Using the rule for transforming instances to transitions we obtain the transition labeled `T1.0` as follows.

$$s2o((t_0, t_1), C, \sigma_0)$$
$$= (s2o((\sigma_0/B)(t_0), cloc(C)), s2o((\sigma_0/B)(t_1), cloc(C)))$$
$$= (s2o(\{\texttt{CM | cm E1}\{\texttt{cyto A0}\}\}, \texttt{out}),$$
$$s2o(\{\texttt{CM | cm E1[A0 - act]}\{\texttt{cyto}\}\}, \texttt{out}))$$
$$= (\texttt{< E1, cm >< A0,cyto >}, \texttt{< E1,cm >< [A0 - act],cm >})$$

Theorem: OccB2Petri. For $\mathcal{R}$ an occurrence-based rewrite theory and s an initial state, the mapping to Petri nets preserves computations. Specifically, if $(T_s, s2o(s)) = \mathcal{P}(\mathcal{R}, s)$, then

$$\mathcal{R} \vdash s = s_0 \xrightarrow{\rho_1} s_1 \ldots \xrightarrow{\rho_k} s_k \Leftrightarrow T_s \vdash s2o(s_0) \xrightarrow{s2o(\rho_1)} s2o(s_1) \ldots \xrightarrow{s2o(\rho_k)} s2o(s_k).$$

Proof Sketch. By induction on the computation length k. If $s_i \xrightarrow{\rho_{i+1}} s_{i+1}$ then $s2o(\rho_{i+1}) \in T_s$ by SRC. Let $\rho_{i+1} = (r, C, \sigma)$ with $r = (t_0, t_1, c)$ and $l = cloc(C)$. Then $s_i = C[\sigma(t_0)]$, $s_{i+1} = C[\sigma(t_1)]$, and for some occurrence set O $s2o(s_i) = s2o(\sigma(t_0), l) \cup O$ and $s2o(s_{i+1}) = s2o(\sigma(t_1), l) \cup O$. Thus $s2o(s_i) \xrightarrow{s2o(\rho_{i+1})} s2o(s_{i+1})$. Conversely, let $O_i \xrightarrow{\tau_{i+1}} O_{i+1}$, and by induction $O_i = s2o(s_i)$ for some reachable s_i. Also $\tau = (O_0, O_1) = s2o(r, C, \sigma)$ where (r, C, σ) applies to $\hat{s}$. We can find O' such that $O_i = O' \cup O_0$ and $O_{i+1} = O' \cup O_1$. By SC3 we can find C', σ' such that $s_i = C'[\sigma'(t_0)]$, and $s_i \xrightarrow{(r, C', \sigma')} s_{i+1}$ where $s2o(s_{i+1}) = O_{i+1}$.

Counterexample. To see that requirement (**SRC**) that merging preserves rewrites is needed, consider the following rule variants in the Pathway Logic language:

```
[r1]:{CM | Ras {cyto Rac}} => {CM | Ras [Rac - act]{cyto}}
[r2]:{CM | cm Ras {cyto Rac}} => {CM | cm Ras [Rac - act]{cyto}}
```

where `cyto` and `cm` are variables standing for any other components located in the cytoplasm or cell membrane respectively. Consider the state `{CM | Ras Grb2 {Src}} { Rac}}` which can be obtained from `{CM | Ras  {Src Rac}}` by a merge. The rule `r2` applies but `r1` does not, although `r1` applies to the 'before merge' state. Both rules transform to the same Petri net transition:

```
< Ras,CM > < Rac,Cyto > => < Ras,CM > < [Rac - act],CM >
```

which indeed applies to the corresponding occurrence state

```
    < Ras,cm > < Grb2,cm >< Rac,cyto > < Src,cyto >
```

Definition: Petri2RWL. The conversion of an occurrence Petri net to a rewrite theory is simple. If $(T_s, s2o(s)) = \mathcal{P}(\mathcal{R}, s)$, then $PS(\mathcal{R}, s)$ is the rewrite theory with the equational part of $\mathcal{R}$ extended with the definition of occurrences, and rules

$$\{O_i \Rightarrow O_o \bigm| (O_i, O_o) \in T_s\}$$

Theorem: Petri2RWL. The mapping PS preserves computations.

$$PS(\mathcal{R}, s) \vdash s2o(s) \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_k} O_k \ \Leftrightarrow\ T_s \vdash s2o(s) \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_k} O_k$$

5 Relating and Transforming Queries

5.1 Preservation of Properties

The temporal logic used by the Maude model checker, LTL, is based on atomic propositions that can be defined by boolean functions in Maude. In the case of an occurrence-based rewrite theory, we restrict attention to propositions that are positive (goals) and negative (avoids) occurrence tests – basic component b occurs (or does not occur) at location l. These propositions translate to simple membership tests $<b, l> \in s2o(s)$ in the corresponding Petri net model. For example, the property `racAct3` presented in Example 1 of §3.2 contains one positive occurrence test (for the presence of `< [Rac1 - GTP], cm >`) and one negative occurrence test (for the absence of `< [EGFR - act], cm >`).

Let LTLO be the Maude LTL language with propositional part restricted to occurrence propositions. Let ψ be an LTLO formula expressed in the PLMaude language and let $s2o(\psi)$ be the same property expressed in terms of occurrence membership, lifting $s2o(\_)$ homomorphically (on syntax) to LTLO formulas.

Theorem: LTLO. Given an occurrence-based rewrite theory $\mathcal{R}$ and initial state s, let π be a computation of $\mathcal{R}, s$, π' be the corresponding computation of $\mathcal{P}(\mathcal{R}, s)$, and π'' be the corresponding computation of $PS(\mathcal{R}, s)$. Then for any LTLO formula ψ

$$\pi \models \psi \Leftrightarrow \pi' \models s2o(\psi) \Leftrightarrow \pi'' \models s2o(\psi)$$

and thus

$$(\mathcal{R}, s) \models \psi \Leftrightarrow \mathcal{P}(\mathcal{R}, s) \models s2o(\psi) \Leftrightarrow (PS(\mathcal{R}, s), s) \models s2o(\psi)$$

This is a consequence of the isomorphism of computations and the preservation of satisfaction of occurrence properties by the occurrence translations.

Note that the **LTLO** theorem implies that counterexamples are also preserved. This is important, since queries asking to find a computation having certain properties are answered by asking a model-checker to find a counterexample to the assertion that no such computation exists.

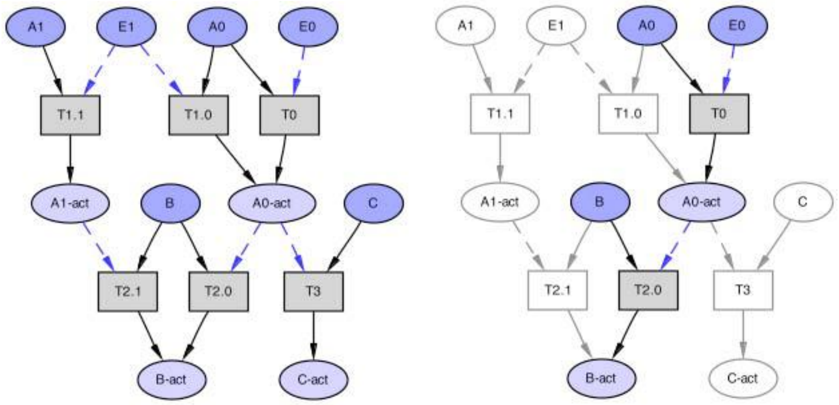
5.2 Relevant Subnets for Goals-Avoids Queries

As indicated in § 3, we are especially interested in answering queries of the form "given initial state I, find a path that satisfies (G, A)" where (G, A) is a basic goals-avoids property with goals G and avoids A. We interpret this as meaning find a path (that is, a computation) starting with the initial state I, that reaches a state satisfying goals G, and such that no state in the path contains any occurrence of A. Without loss, we further require the path to be minimal, in the sense of not using irrelevant transitions. That is, if any transition is removed from the set generating the path, the remaining transitions do not generate a path satisfying the goals. Ideally we would like to focus attention on the subnet of transitions of a Petri net model that might appear in any minimal path satisfying that property. We call these the *truly relevant* transitions. This is of interest both to help the biologist focus on a smaller set of transitions and to reduce the search space to be considered by an analysis tool.

Finding just the truly relevant transitions means finding exactly the minimal paths satisfying a given property, the problem we are trying to simplify. Thus we will look for a safe approximation, called the *relevant* transitions, that is a superset of truly relevant transitions set. Clearly, transitions that mention an occurrence to be avoided can be eliminated, as can transitions that do not contribute to reaching some goal, or transitions whose pre-set will not be a subset of a reachable state. In the following we define three transformations that formalize these intuitions. The first transformation removes transitions that mention an occurrence to be avoided. The second transformation is a backwards collection of transitions that contribute to reaching a goal, either because the post-occurrences contain a goal, or recursively contain a pre-occurrence of some contributing transition. The third transformation is a forward collection of transitions applicable to reachable states. Then given a Petri net (T, I), and a goals-avoids property (G, A), T is transformed/reduced to the corresponding set of relevant transitions $relTrans(T, I, G, A)$ by the following process:

$$\begin{tikzcd}
A \arrow[d] & & G \arrow[d] & & I \arrow[d] & & \\
{[T]} \arrow[rr, "\text{avoids}", "\text{elimination}"'] & & {[T/A]} \arrow[rr, "\text{backwards}", "\text{collection}"'] & & {[(T/A)^b_G]} \arrow[rr, "\text{forwards}", "\text{collection}"'] & & {[((T/A)^b_G)^f_I]}
\end{tikzcd}$$

We will show that any minimal path from initial state I meeting a goals-avoids property (G, A) using transitions in T, in fact uses only transitions in $\mathit{relTrans}(T, I, G, A)$, thus it is a safe approximation.



(a) TinyPN (b) Relevant Subnet

Fig. 3. Tiny Petri net (a) and a relevant subnet (b). Dark ovals represent initial state. In (b) the subnet is colored and the original net context is white.

Figure 3 shows the Petri net `tinyPN = (tinyT, tinyI)` from Example 5 of §4.3. In the graphical form we use simple names to label (and refer to) occurrences, leaving the location part implicit. This will be used to illustrate the concepts and transformations discussed below.

Definition: Minimal Paths. Let I (initial state), G (goals), A (avoids) be occurrence sets such that $(I \cup G) \cap A = \emptyset$. The set $P(T, I, G, A)$ is the set of Petri net computations, π, that start from the initial state I, and reach a state containing all occurrences in G, using transitions in T without ever marking A.

$$\pi = O_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_k} O_k \in P(T, I, G, A) \Leftrightarrow O_0 = I \wedge G \subseteq O_k \wedge \bigwedge_{0 \le i \le k} O_i \cap A = \emptyset$$

π is minimal if there is no computation π' in $P(T, I, G, A)$ that uses a proper subset of the transitions used in π, and we let $mP(T, I, G, A)$ be the set of computations of $P(T, I, G, A)$ that are minimal.

Example 6: Minimal and non Minimal Paths. In `tinyPN` (Figure 3) the transitions `T1.1`, `T1.0` form a path to the goal `A0-act`, but it is not minimal as `T1.1` can be removed since `T1.0` alone is a path.

Lemma: Path Monotonicity. The set of minimal paths monotonically increases with increasing initial state and decreasing goals and avoids. Specifically, if $T \subseteq T'$, $I \subseteq I'$, $G' \subseteq G$, $A' \subseteq A$, then

$$P(T, I, G, A) \subseteq P(T', I', G', A')$$

and

$$mP(T, I, G, A) \subseteq mP(T', I', G', A')$$

Definition: Removing Avoids. Assume given T and A as above. The result of removing rules that mention an element of A is defined by

$$T/A = \{\tau \in T \mid (pre(\tau) \cup post(\tau)) \cap A = \emptyset\}$$

Example 7. Removing Avoids. Taking A to be `A1-act` removing the avoids from the transitions of `tinyPN` means removing `T1.1` and `T2.1`, leaving `T0`, `T1.0`, `T2.0`, and `T3`, that is

$$\texttt{tinyT}/A = \{\texttt{T0}, \texttt{T1.0}, \texttt{T2.0}, \texttt{T3}\}.$$

Lemma: Removing Avoids is Safe. If $\pi \in mP(T, I, G, A)$, then $\pi \in mP(T/A, I, G, A)$.

Proof. Since by definition no transition in $T - T/A$ could be used in π.

Definition: Backward collection. Assume given T, G as above. The backward collection T^b_G of T relative to G is defined by

$$T^b_G = \bigcup_{j \in \mathbf{Nat}} T^b_j \quad \text{where}$$

$$G_0 = G \qquad G_{j+1} = G_j \cup \bigcup_{\tau \in T^b_j} pre(\tau)$$

$$T^b_j = \{\tau \in T \mid out(\tau) \cap G_j \neq \emptyset\}$$

Note that for some n, $G_j = G_{j+1}$ for $j > n$ since T is finite and thus only finitely many increments can be made.

Example 8. Backwards Collection. Backwards collection of `tinyT` for goal `B-act`, $\texttt{tinyT}^b_{\texttt{B-act}}$, is `tinyT` minus `T3`. The can be seen from the following steps in the collection:

$$G_0 = \texttt{B-act}$$

$$\texttt{tinyT}^b_0 = \{\texttt{T2.0}, \texttt{T2.1}\}$$

$$G_1 = \{\texttt{B-act}, \texttt{B}, \texttt{A0-act}, \texttt{A1-act}\}$$

$$\texttt{tinyT}^b_1 = \texttt{tinyT}^b_0 \cup \{\texttt{T0}, \texttt{T1.0}, \texttt{T1.1}\}$$

As another example, for goals `A0-act` and `A1-act` we have

$$\texttt{tinyT}^b_{\{\texttt{A0-act},\texttt{A1-act}\}} = \{\texttt{T0}, \texttt{T1.0}, \texttt{T1.1}\}$$

Lemma: Backward Monotonicity. Backwards collection is monotonic in transitions and goals. That is, if $T \subseteq T'$ and $G \subseteq G'$, then $T^b_G \subseteq (T')^b_{G'}$.

The lemma **Backwards 1** captures the essence of the reason that a transition that appears in some minimal path for a set of goals is one produced by backwards collection.

Lemma: Backwards 1. If $O \xrightarrow{\tau_1} O_1 \xrightarrow{\tau_2} O_2$ and $pre(\tau_2) \cap out(\tau_1) = \emptyset$ then we can find O_2' such that $O \xrightarrow{\tau_2} O_2'$. Furthermore, for any occurrence set G^*, if $out(\tau_1) \cap G^* = \emptyset$, then $G^* \cap O_2 \subseteq G^* \cap O_2'$.

Proof. With the assumptions of the lemma, $pre(\tau_2) \subseteq O$, letting $O_2' = (O - in(\tau_2)) \cup out(\tau_2)$ we have, by definition of transition, the desired transition. Also by definition of transition, $O_2 = ((O - in(\tau_1) \cup out(\tau_1)) - in(\tau_2)) \cup out(\tau_2)$. Assuming $out(\tau_1) \cap G^* = \emptyset$ we have $G^* \cap O_2 = G^* \cap ((O - in(\tau_1) - in(\tau_2)) \cup out(\tau_2)) \subseteq G^* \cap O_2'$.

The lemma **Backwards 2** identifies conditions under which a sequence of transitions can be restarted at a new state. For backwards collection, the state of interest is one resulting from deleting an irrelevant transition, such as τ_1 in **Backwards 1**.

Lemma: Backwards 2. If $O \cap G \subseteq O' \cap G$, $pre(\tau) \subseteq O'$ and $O \xrightarrow{\tau} O_1$, then we can find O_1' such that $O_1 \cap G \subseteq O_1' \cap G$ and $O' \xrightarrow{\tau} O_1'$.

Proof. By the assumptions, $pre(\tau) \subseteq O'$, so letting $O_1' = (O' - in(\tau)) \cup out(\tau)$ we have the desired transition. Since $O_1 = (O - in(\tau)) \cup out(\tau)$, if $g \in O_1$ either $g \in out(\tau)$ or $g \in O - in(\tau) \subseteq O' - in(\tau)$. Thus $g \in O_1'$.

Theorem: Backward safety. If $\pi \in mP(T, I, G, A)$, then $\pi \in mP(T^b_G, I, G, A)$.

Proof Sketch. Let $\pi = I \xrightarrow{\tau_1} O_1 \ldots \xrightarrow{\tau_k} O_k \in mP(T, I, G, A)$. We show that $\tau_j \in T^b_G$ for $1 \leq j \leq k$. Suppose not. Let G^* be the union of the G_js in the definition of T^b_G, and let j be the largest number such that $\tau_j \notin T^b_G$. Thus $out(\tau_j) \cap G^* = \emptyset$. We construct $\pi' \in P(T, I, G, A)$ using fewer transitions, contradicting minimality of π. If $j = k$ then $G \subseteq O_{k-1}$ and π' is the first $k - 1$ transitions of π. If $j < k$ then let $O_j' = O_{j-1}$, and $O_{i+1}' = (O_i' - in(\tau_{i+1})) \cup out(\tau_{i+1})$ for $j \leq i < k$. By maximality of j, $\tau_{i+1} \in T^b_G$ and thus $pre(\tau_{i+1}) \subseteq G^*$ for $j \leq i < k$. We claim that $O_{i+1} \cap G^* \subseteq O_{i+1}' \cap G^*$ and $O_i' \xrightarrow{\tau_{i+1}} O_{i+1}'$ for $j \leq i < k$. For $i = j$ this follows by backwards lemma 1 and for $i > j$ it follows by backwards lemma 2. Thus taking $\pi' = I \xrightarrow{\tau_1} O_1 \ldots \xrightarrow{\tau_{j-1}} O_j' \xrightarrow{\tau_{j+1}} \ldots \xrightarrow{\tau_k} O_k'$ we are done.

Definition: Forward collection. The forward collection T^f_I of T relative to I is defined by

$$T^f_I = \bigcup_{j \in \mathbf{Nat}} T^f_j \qquad I^f = \bigcup_{j \in \mathbf{Nat}} I_j \quad \text{where}$$

$$I_0 = I \qquad I_{j+1} = \bigcup_{\tau \in T^f_j} post(\tau)$$

$$T^f_j = \{\tau \in T \mid pre(\tau) \subseteq I_j\}$$

Again, we have that for some n, $I_j = I_n$ for $j \geq n$.

Example 9. Forward Collection. Suppose we remove `E1` from the inital state, call this I_1, then forward collection of `tinyT` from I_1 omits `T1.0`, `T1.1` and `T2.1`. Thus $\texttt{tinyT}^f_{I_1} = \{\texttt{T0}, \texttt{T2.0}, \texttt{T3}\}$.

Lemma: Forward Monotonicity. If $T \subseteq T'$ and $I \subseteq I'$, then $T^f_I \subseteq (T')^f_{I'}$

Theorem: Forward safety. If $\pi \in mP(T, I, G, A)$, then $\pi \in mP(T^f_I, I, G, A)$.

Proof. This is because for each transition τ_{j+1} in π, using the notation of the definition, $pre(\tau_{j+1}) \subseteq I_j$, and thus $\tau_{j+1} \in T^f_j$ for $0 \leq j < k$.

Definition: Relevant Subnet. The subnet of transitions from T relevant to initial state I, goals G, and avoids A is defined by

$$relTrans(T, I, G, A) = ((T/A)^b_G)^f_I.$$

Corollary: Relevant Subnet. If $\pi \in mP(T, I, G, A)$ is non-empty, then

$$\pi \in mP(relTrans(T, I, G, A), I, G, A)$$

Thus search for such paths can be carried out in the relevant subnet $relTrans(T, I, G, A)$. Note that if $G \not\subseteq I^f$ then $P(T, I, G, A)$ is empty.

Example 10. Relevant subnets. The relevant subnet of `tinyPN` for goals `B-act`, avoids `A1-act` and initial state $I_1 = \texttt{tinyI} - \texttt{E1}$

$$relTrans(\texttt{tinyT}, I_1, \texttt{B-act}, \texttt{A1-act}) = \{\texttt{T0}, \texttt{T2.0}\}$$

is shown in figure 3(b).

6 Summary and Future Work

The main contributions of the paper are: a definition of mappings between rewriting logic and petri net representations of biological processes (and similar concurrent processes) that satisfy certain conditions; proof that these mappings preserve properties of interest; and definition of a relevant subnet transformation that reduces the number of transitions that must be considered in search for a path satisfying a goals-avoids property.

As context we presented an overview of Pathway Logic illustrated with a model of Rac1 activation as Maude rules and the representation as a Petri net. We also discuss the advantages of analyses based on Petri nets.

As models grow in size, we expect to need to explore alternative path finding algorithms. Possibilities include employing more highly tuned model checkers, discovering new simplification and abstraction transformations, and developing constraint solving approaches to take advantage of the rapid advances being made in this area. Another big challenge is refining PLMaude models to incorporate semi-quantitative information about expression levels and relative preference for competing reactions, and to be able compute with and visualize the refined models in ways that are meaningful to working biologists.

Acknowledgments. The authors would like to thank the Pathway Logic team and the anonymous reviewers for helpful criticisms. The work was partially funded by NIH/ NIGMS grant GM68146.

References

1. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., Sonmez, K.: Pathway Logic: Symbolic analysis of biological signaling. In: Proceedings of the Pacific Symposium on Biocomputing. (2002) 400–412
2. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Talcott, C.: Pathway Logic: Executable models of biological networks. In: Fourth International Workshop on Rewriting Logic and Its Applications (WRLA'2002), Pisa, Italy, September 19 — 21, 2002. Volume 71 of Electronic Notes in Theoretical Computer Science., Elsevier (2002) `http://www.elsevier.nl/locate/entcs/volume71.html`.
3. Talcott, C., Eker, S., Knapp, M., Lincoln, P., Laderoute, K.: Pathway logic modeling of protein functional domains in signal transduction. In: Proceedings of the Pacific Symposium on Biocomputing. (2004)
4. Clavel, M., Durán, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J., Talcott, C.: Maude 2.0 Manual (2003) `http://maude.cs.uiuc.edu`.
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: The Maude 2.0 system. In Nieuwenhuis, R., ed.: Rewriting Techniques and Applications (RTA 2003). Volume 2706 of Lecture Notes in Computer Science., Springer-Verlag (2003) 76–87
6. Mason, I.A., Talcott, C.L.: IOP: The InterOperability Platform & IMaude: An interactive extension of Maude. In: Fifth International Workshop on Rewriting Logic and Its Applications (WRLA'2004). Electronic Notes in Theoretical Computer Science, Elsevier (2004)
7. Kohn, K.W.: Functional capabilities of molecular network components controlling the mammalian g1/s cell cycle phase transition. Oncogene **16** (1998) 1065–1075
8. Weng, G., Bhalla, U.S., Iyengar, R.: Complexity in biological signaling systems. Science **284** (1999) 92–96
9. Shvartsman, S.Y., Hagan, M.P., Yacoub, A., Dent, P., Wiley, H., Lauffenburger, D.: Autocrine loops with positive feedback enable context-dependent cell signaling. Am J Physiol Cell Physiol **282** (2002) C545–559
10. Smolen, P., Baxter, D.A., Byrne, J.: Mathematical modeling of gene networks. Neuron **26** (2000) 567–580

11. Lok, L.: Software for signaling networks, electronic and cellular. Science STKE **PE11** (2002)
12. Rao, C.V., Wolf, D.M., Arkin, A.P.: Control, exploitation and tolerance of intracellular noise. Nature **420** (2002) 231–237
13. Endy, D., Brent, R.: Modelling cellular behaviour. Nature **409** (2001) 391–395
14. Peterson, J.L.: Petri Nets: Properties, analysis, and applications. Prentice-Hall (1981)
15. Milner, R.: A Calculus of Communicating Systems. Springer Verlag (1980)
16. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming **8** (1987) 231–274
17. Meseguer, J.: Conditional Rewriting Logic as a unified model of concurrency. Theoretical Computer Science **96**(1) (1992) 73–155
18. Croes, D., Couche, F., van Helden, J., Wodak, S.: Path finding in metabolic networks: measuring functional distances between enzymes. In: Open Days in Biology, Computer Science and Mathematics JOBIM 2004. (2004)
19. Hofestädt, R.: A Petri net application to model metabolic processes. Syst. Anal. Mod. Simul. **16** (1994) 113–122
20. Reddy, V.N., Liebmann, M.N., Mavrovouniotis, M.L.: Qualitative analysis of biochemical reaction systems. Comput. Biol. Med. **26** (1996) 9–24
21. Goss, P.J., Peccoud, J.: Quantitative modeling of stochastic systems in molecular biology using stochastic Petri nets. Proc. Natl. Acad. Sci. U. S. A. **95** (1998) 6750–6755
22. Küffner, R., Zimmer, R., Lengauer, T.: Pathway analysis in metabolic databases via differential metabolic display (DMD). Bioinformatics **16** (2000) 825–836
23. Matsuno, H., Doi, A., Nagasaki, M., Miyano, S.: Hybrid Petri net representation of gene regulatory network. In: Pacific Symposium on Biocomputing. Volume 5. (2000) 341–352
24. Oliveira, J.S., Bailey, C.G., Jones-Oliveira, J.B., Dixon, D.A., Gull, D.W., Chandler, M.L.: An algebraic-combinatorial model for the identification and mapping of biochemical pathways. Bull. Math. Biol. **63** (2001) 1163–1196
25. Genrich, H., Küffner, R., Voss, K.: Executable Petri net models for the analysis of metabolic pathways. Int. J. STTT **3** (2001)
26. Oliveira, J.S., Bailey, C.G., Jones-Oliveira, J.B., Dixon, D.A., Gull, D.W., Chandler, M.L.: A computational model for the identification of biochemical pathways in the Krebs cycle. J. Computational Biology **10** (2003) 57–82
27. Zevedei-Oancea, I., Schuster, S.: Topological analysis of metabolic networks based on Petri net theory. In Silico Biology **3**(0029) (2003)
28. Regev, A., Silverman, W., Shapiro, E.: Representation and simulation of biochemical processes using the pi-calculus process algebra. In Altman, R.B., Dunker, A.K., Hunter, L., Klein, T.E., eds.: Pacific Symposium on Biocomputing. Volume 6., World Scientific Press (2001) 459–470
29. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Information Processing Letters (2001) in press.
30. Calder, M., Vyshemirsky, V., Gilbert, D., Orton, R.: Analysis of signalling pathways using the PRISM model checker. In Plotkin, G., ed.: Proceedings of the Third International Conference on Computational Methods in System Biology (CMSB 2005). (2005)
31. Regev, A., Panina, E., Silverman, W., Cardelli, L., Shaprio, E.: Bioambients: An abstraction for biological compartments (2003) to appear TCS.
32. Nielson, F., Nielson, H.R., Priami, C., Rosa, D.: Control flow analysis for bioambients. In: BioConcur. (2003)
33. Kam, N., Cohen, I., Harel, D.: The immune system as a reactive system: Modeling t cell activation with statecharts. In: Proc. Visual Languages and Formal Methods (VLFM'01). (2001) 15–22

34. Efroni, S., Harel, D., Cohen, I.: Towards rigorous comprehension of biological complexity: Modeling, execution and visualization of thymic t-cell maturation. Genome Research (2003) Special issue on Systems Biology, in press.
35. Damm, W., Harel, D.: Breathing life into message sequence charts. Formal Methods in System Design **19**(1) (2001)
36. Kam, N., Harel, D., Kugler, H., Marelly, R., Pnueli, A., Hubbard, J., Stern, M.: Formal modeling of C.elegans development: A scenario-based approach. In: First International Workshop on Computational Methods in Systems Biology. Volume 2602 of Lecture Notes in Computer Science., Springer (2003) 4–20
37. Prez-Jimnez, M., Romero-Campero, F.: Modelling EGFR signalling cascade using continuous membrane systems. In Plotkin, G., ed.: Proceedings of the Third International Conference on Computational Methods in System Biology (CMSB 2005). (2005)
38. Fages, F., Soliman, S., Chabrier-Rivier, N.: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. Journal of Biological Physics and Chemistry **4**(2) (2004) 64–73
39. Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., Schächter, V.: Modeling and querying biomolecular interaction networks. Theoretical Computer Science **351**(1) (2004) 24–44
40. Calzone, L., Chabrier-Rivier, N., Fages, F., Gentils, L., Soliman, S.: Machine learning biomolecular interactions from temporal logic properties. In Plotkin, G., ed.: Proceedings of the Third International Conference on Computational Methods in System Biology (CMSB 2005). (2005)
41. Baral, C., Chancellor, K., Tran, N., Tran, N., Joy, A., Berens, M.: A knowledge based approach for representing and reasoning about signaling networks. Bioinformatics **20** (2004) i15–i22
42. Shankland, C., Tran, N., Baral, C., Kolch, W.: Reasoning about the ERK signal transduction pathway using BioSigNet-RR. In Plotkin, G., ed.: Proceedings of the Third International Conference on Computational Methods in System Biology (CMSB 2005). (2005)
43. Hucka, M., Finney, A., Sauro, H., Bolouri, H., Doyle, J., Kitano, H.: The ERATO systems biology workbench: Enabling interaction and exchange between software tools for computational biology. In: Proceedings of the Pacific Symposium on Biocomputing. (2002)
44. BioSpice (2004) `https://community.biospice.org/`.
45. IBM Discoverylink (2004) `http://publib-b.boulder.ibm.com/Redbooks.nsf/0/3c7a635147cf20d785256a540064e287?OpenDocument&Highlight=0,DiscoveryLink`.
46. geneticXchange (2004) `http://midas-10.cs.ndsu.nodak.edu/bio/ppts/chung.pdf`.
47. Schmidt, A., Hall, A.: Guanine nucleotide exchange factors for rho gtpases: turning on the switch. Genes Dev. **16** (2002) 1587–15609
48. Stehr, M.O.: A rewriting semantics for algebraic nets. In Girault, C., Valk, R., eds.: Petri Nets for System Engineering – A Guide to Modelling, Verification, and Applications. Springer-Verlag (2000)
49. Schmidt, K.: LoLA: A Low Level Analyser. In Nielsen, M., Simpson, D., eds.: Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000). Volume 1825 of Lecture Notes in Computer Science., Springer (2000) 465–474
50. LoLA: Low Level Petri net Analyzer (2004) `http://www.informatik.hu-berlin.de/~kschmidt/lola.html`.
51. Schmidt, K.: Stubborn sets for standard properties. In: International Conference on Application and Theory of Petri nets. Volume 1639 of Lecture Notes in Computer Science., Springer (1999) 46–65
52. Porter, D.: An Interpreter for `JLambda` (2004) `http://mcs.une.edu.au/~iop/Data/Papers/`.

Author Index