The π-calculus as an abstraction for biomolecular systems

Aviv Regev[1] and Ehud Shapiro[2]

[1] Bauer Center for Genomics Research, Harvard University, Cambridge, Ma. aregev@cgr.harvard.edu

[2] Dept. of Computer Science and Applied Math, Weizmann institute, Rehovot, Israel Ehud.Shapiro@weizmann.ac.il

1 Introduction

Biochemical processes, carried out by networks of proteins, underly the major functions of living cells ([8],[61]). Although such systems are the focus of intensive experimental research, the mountains of knowledge about the function, activity, and interaction of molecular systems in cells remain fragmented. While computational methods are key to addressing this challenge ([8],[61]), they require the adoption of a meaningful mathematical abstraction [51]. The research of biomolecular systems has yet to identify and adopt such a unifying abstraction.

An abstraction - a mapping from a real-world domain to a mathematical domain - highlights some essential properties while ignoring other, complicating, ones. A good scientific abstraction has four properties: it is *relevant*, capturing an essential property of the phenomenon; *computable*, bringing to bear computational knowledge about the mathematical representation; *understandable*, offering a conceptual framework for thinking about the scientific domain; and *extensible*, allowing the capture of additional real properties in the same mathematical framework.

For a good abstraction for biomolecular processes to be *relevant*, it should capture two essential properties of these systems in one unifying view: their molecular organization and their dynamic behavior. A *computable* abstraction will then allow both the simulation of this dynamic behavior, and the qualitative and quantitative reasoning on these systems' properties. An *understandable* abstraction will correspond well to the informal concepts and ideas of molecular biology, while opening up new computational possibilities for understanding molecular systems, by *e.g.* suggesting formal ways to ascribe biomolecular function to a biological system, or by suggesting objective measures for behavioral similarities between systems. Finally, the desired abstraction should be *extensible*, scaling to higher levels of organization, in which molecular systems play a key, albeit not exclusive, role.

In this work we aim to adopt the much-needed abstraction for biomolecular systems from computer science. We investigate the "molecule-as-computation" abstraction, in which a system of interacting molecular entities is described and modeled by a system of interacting computational entities. Based on this abstraction, we adapt process algebras for the representation and simulation of biomolecular systems, including regulatory, metabolic and signaling pathways, as well as multicellular processes.

1.1 Previous work

Biochemical systems are traditionally studied using "dynamical systems theory". In recent years, additional approaches from computer science have been adapted for the representation of pathways. The various approaches can be roughly divided into four groups.

- **Chemical kinetic models.** Dynamical systems theory views biomolecular systems based on the "cell-as-collection-of-molecular-species" abstraction. Such models describe different molecular systems from a pure biochemical perspective (*e.g.* [3], [5], [7], [34], [38], [37], [56], [71], [57]). This well-developed approach has several advantages: it is clear well understood, and extremely powerful, with an extensive theoretical background, and a variety of methodologies and tools. However, it has been convincingly argued that this approach, adopted from the physical science, is lacking in *relevance* and *understandability*, when handling biolgical systems [16]. Its main limitation lies in its indirect treatment of biomolecular *objects*. Biological systems are composed of molecular objects, which maintain their overall identity, while changing in specific attributes, such as their chemical modification, activation state, or location. Chemical models, however, handle the cell as a a monolithic entity and are mostly insusceptible to more structured descriptions that are typical to biological thinking [16]. Thus, each modified state of a molecule, a molecular complex, machine or compartment, must be handled as a distinct variable rather than as one entity with multiple states.
- **Generalized models of regulation.** Boolean network models use a "molecule-as-logical-expression" abstraction to describe and simulate gene regulatory circuits. These models were introduced by Kaufmann and subsequently applied to various biological systems (e.g.[1], [2], [13], [39], [63], [54], [55], [64], [65]). These approaches have proven highly useful when studying general properties of large networks, and in handling system for which only qualitative knowledge is available. However, as a general model of biomolecular system (rather than a specific model of regulation) they are limited in *relevance* (as they apply only to the regulation aspect of molecular systems), *computability* (as they have only limited predictive power) and *extensibility* (as, apparently, a Boolean abstraction of activation and inhibition suffices to handle only specific types of molecular systems).

- **Functional object-oriented databases.** Pathway databases are based on the "molecule-as-object" abstraction. They store information on molecular interactions (*e.g.* the DIP [70], BIND [6], and INTERACT [14] databases) and complete pathways (*e.g.* the MetaCyc [30], WIT [58], KEGG [43], CSNDB [27], aMAZE [66], and TransPath [69] databases). Each such database has a sophisticated object-oriented schema that provides a biologically-appealing hierarchical view of molecular entities. Most databases are equipped with querying tools of variable levels of sophistication, from simple queries to pathway reconstruction. Functional databases provide an excellent solution for organizing, manipulating and (sometimes) visualizing pathway data. However, in most cases, they provide little if any dynamic capabilities (*e.g.* simulation) and their querying tools are thus seriously limited. Note, that recently developed **exchange languages** (*e.g.* [15], [31], and [42]) attempt to address these limitations by providing means to integrate models and tools from various sources. Most are XML-based markup languages (*e.g.* CellML and SBML). However, the utility of the language depends on its underlying pathway model, which is still primarily a kinetic one.
- **Abstract process languages applied to biomolecular systems.** Approaches based on abstract process languages use the "molecule-as-computation" abstraction, and have gained increasing importance during the past few years. They were first proposed by Fontana and Buss [16], who employed the λ-calculus and linear logic as alternatives to dynamics systems theory for studying (bio)chemistry. Notable examples use existing formalisms for concurrent computation (often with a strong graphical component) to model real biomolecular systems. The most comprehensive works used Petri nets (e.g. [20], [21], [25], [33], [35], and [36]) for representation, simulation and analysis of metabolic pathways. Petri nets considerably improve over tranditional kinetic model, due to their clarity and graphical convenience, and augment it further with specific analytic tools. However, Petri nets essential suffer from the same *relevance* problem as "dynamics systems theory": each molecule (and state or modification) is represented by an individual place, and there is no immediate use of more structured representation of complex molecular entities. Indeed, Petri nets are mostly (and succesfully) used for the study of metabolic pathways, handled from a largely chemical perspective. Recent advances [36] try to overcome these limitations by combining Petri nets with object oriented representation.

  Recent studies employed Statecharts ([26], [28], [29]) to build qualitative graphical models primarily for cellular systems with a molecular component. Unlike typical graph-based visualization of pathways, Statecharts provide a rich and expressive process language with clear semantics. Statecharts are highly expressive, and are proving succesful in qualitatively representing complex biological system, from a molecular to an organismic scale. Unfortunately, statecharts are currently a primarily qualitative framework, and do not handle quantitative aspects in a direct and accessible manner. Statecharts are, however, extremely useful in describing complex multi-cellular scenarios.

These recent attempts highlight the promise in using abstract process languages for pathway informatics. Our approach belongs to this last category and is based on the "molecule as computation" abstraction, using the computational framework of the π-calculus and its extensions ([52], [47]). The potential use of the π-calculus in molecular biology was also discussed by Fontana [16], and the π-calculus was also employed by Ciobanu ([10], [11]) for modeling molecular systems, primarily those related to DNA methylation. Here, we develop ways to represent and simulate biomolecular processes, describing the set of steps taken to adapt, extend and implement this core language to conform to the unique requirements of biochemical systems, first on a semi-quantitative and then on a fully quantitative, stochastic scale. We present the language intuitively[3] and provide a multitude of simple and complex examples to which it is applied, highlighting its capabilities to represent and study a variety of molecular systems, including compositional and modular ones.

2 Abstracting biomolecular systems as concurrent computation: An overview

An abstraction is a mapping from a real-world domain to a mathematical one, which preserves some essential properties of the real-world domain while ignoring other properties, considered superfluous. Building the abstraction has three components. First, we informally organize our knowledge on the real world domain, identifying essential entities in this domain, their properties and behavior. Second, we select a mathematical domain which we believe can be useful for the abstraction of the real world one, motivated by some informal correspondence between those properties or behaviors we deem essential in the real-world domain and those of the mathematical one. Third, we design and perform the abstraction, starting by laying down the principles for the mapping, and then using these *pragmatic* guidelines to build specific representations of actual real world entities .

The real-world domain: Essential properties of biomolecular systems

We identify several essential aspects of biomolecular systems (also illustrated by a toy example in Figure 1).

[3] Due to space limitation we do not formally present the π-calculus, but rather use an informal presentation of the language, focused on biological examples. We assume that biological readers will benefit the most from this decision. Readers from a formal backgroud may skip the informal introduction (Section 2) if familiar with the calculus, or are referred to [41] for details.

- **Molecular species and populations.** Biomolecular systems are composed of a population of molecules. The molecular population may include multiple molecular species(*e.g.* Figure 1A), with multiple copies of each molecule type .
- **Molecular composition.** Each protein molecule may be composed of several domains – independent structural and functional parts (Figure 1A). Each domain may have one or more structural-chemical motifs (Figure 1A).
- **Complementarity and specific interaction.** Different domains specifically interact through complementary motifs. Different motifs in domains allow for different specific interactions (Figure 1A,B).
- **Interaction outcomes: Reconstitution, modification and change in molecular state.** One or more things can happen to molecules following interaction. First, the molecule (*e.g.* an enzyme) may be reconstituted without any change. In other cases, the molecule may be changed following interaction (Figure 1C,D). The change may be attributed to a specific chemical modification of one of the motifs in the molecule by its interaction partner, or may be regarded as a more general change in the state of the molecule, such as a change from an inactive to an active state. In each state the molecule may potentially participate in different interactions.
- **Alternative interactions.** In many cases a molecule may be able to participate in more than one interaction. The alternative interactions may be independent. For example, protein A may interact with protein B on one domain and with protein C on another, such that one interaction does not preclude the other. In other cases, the same domain may have several potential interaction partners, which compete with each other.

The mathematical domain: Concurrent computation

In the "molecule-as-computation" abstraction, a system of interacting molecular entities is described and modeled by a system of interacting computational entities. Several abstract computer languages such as Petri nets [53], Statecharts [22], concurrent logic programming [59], and the π-calculus [41], were developed for the specification and study of such systems.

We focus on the π-calculus language. π-calculus programs specify networks of *communicating* processes. Communication occurs on complementary *channels*, that are identified by specific names (Figure 2). The programs describe the potential behavior of processes: what communications the processes can participate in on such channels.

There are two different kinds of communications in the π-calculus. In the first type (also present in simpler languages, such as CCS [41]), the processes only send *alerts* (or empty *"nil"* messages) to one another. In this case (depicted in Figure 3A), a sender process may communicate with a receiver process if they share the same communication channel. Following communication,

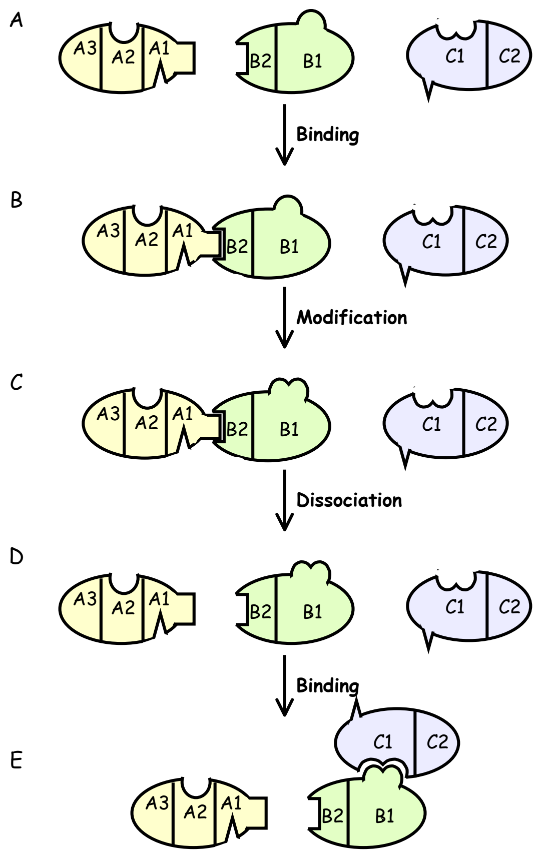

Fig. 1. A toy biomolecular system. A. Three molecular species: A, B, and C, each with several sub-domains (dividing lines) and motifs (protrusions and depressions). B. Domains A1 and B2 interact via complementary motifs. C. A motif on domain B1 is modified. D. The proteins dissociate E. The modified motif on B1 is used for interaction with domain C1.

the action is removed, and each process may either iterate or change its state, becoming another process with different channels and behavior.

The second type of communication is more complex and unique to the π-calculus. In this case (Figure 3B), the sender process sends a *message* to the receiving process. The content of this message is one or more channel names. These channels can be used by the receiving process to communicate with other processes. As before, following communication the processes may either iterate or change state. However, passing channels as messages allows

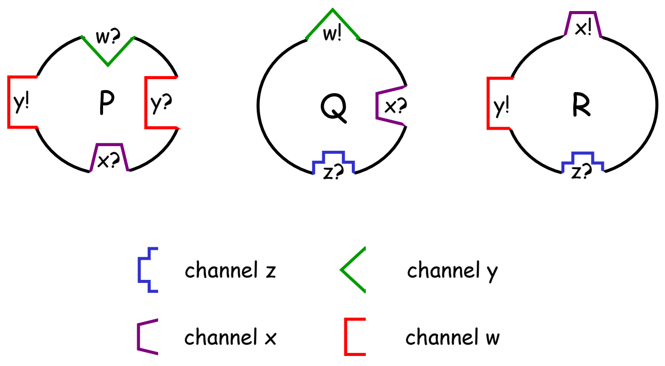

Fig. 2. π-calculus processes and channels: An intuitive view. Three processes, P, Q, R (ovals) with four communication channels (complementary shapes of protrusions and depressions).

the process (and its continuations) to acquire communication capabilities dynamically, that were not specified *a priori* in its explicit program. Such a change in future communication capabilities as a result of passing messages is termed *mobility*.

As shown in Figure 3 the two communication capabilities may serve to describe similar systems. However, in the former mechanism the communication capabilities of a process are all strictly pre-defined (and thus may change only as a result of a pre-specified state change), while in the latter messages may change the communication capabilities of a process dynamically. Thus, messages allow us to leave certain components under-specified, to be determined only following interaction with different processes which may send different messages along the same channel.

The molecule-as-computation abstraction

We find an intuitive correspondence between the world of computational processes and of biomolecular systems, based on the representation of molecules as the processes in which they may participate. Note, that this view is essentially a biological one. Biologists typically characterize molecules by *what they can do*. For example, enzymes are named by the reaction that they can catalyze, and binding domains in proteins – by the entities which they bind.

Table 1 gives a general intuition of the guidelines underlying this abstraction. In the next one we build the abstraction step-by-step with a real biological system.

Note, that we may represent modification either as a special case of molecular state change or by employing the mobility mechanism of the π-calculus,

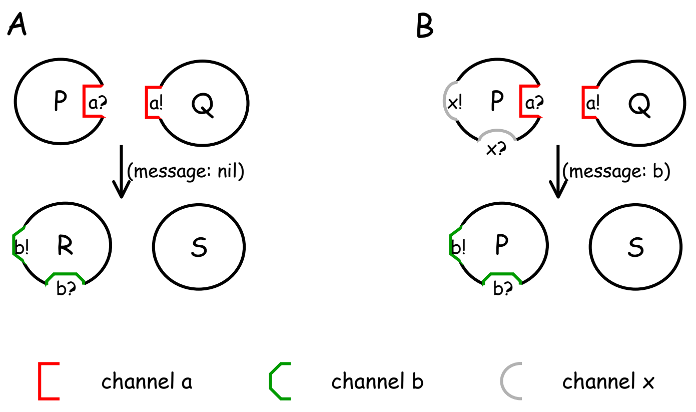

Fig. 3. π-calculus communication: A schematic view. A. Communication with state change. **B.** Communication with mobility. P,Q,S,R (ovals) - processes; a,b - communication channels.

Table 1. Guidelines for the abstraction of biomolecular systems to the π-calculus.

Biomolecular entity	**π-calculus entity**
Molecular species	Process species
Molecular population	System of concurrent processes
Complementary motif types	Complementary input and output occurrences on the same channel name
Motif occurrence in molecule or domain	Communication offers on channels in process
Biomolecular event	**π-calculus event**
Specific interaction on complementary motif	Specific communication on complementary channels
Outcome following interaction	Communication *prefix* preceding process creation or other communication
Reconstitution following interaction	Communication *prefix* preceding process *recreation*
Changed molecular state following interaction	Communication *prefix* preceding creation of new process
Modification of motifs during interaction	Non-mobile approach: re-creation of a different type of process with a different channel set
	Mobile approach: Message (a tuple of channel names) sent in communication to replace channels in the receiving process

abstracting state change and modification as a *message* sent from one of the processes to the other one. Each of these two abstractions for molecular interactions has benefits and drawbacks. In the non-mobile approach, the abstracted system is fully specified, and thus often simpler to write and follow. The mobile approach is more complicated, but also closer to biological reasoning: the same process continues to abstract molecules throughout modification, and message passing corresponds directly and specifically to the modification event.

3 Biomolecular systems in the π-calculus

To fully describe the abstraction of biomolecular process in the π- calculus, we now turn to a series of biological examples, which we will abstract step by step using the various constructs and rules of the π-calculus. A formal presentation of the calculus is given in the next section. Our examples are derived from a model of a canonical signal transduction pathway starting from a receptor tyrosine kinase (RTK) on a cell's membrane, and ending with transcriptional activation of immediate early genes.

3.1 Molecules and domains as concurrent processes

We abstract each biomolecular system as a process, denoted by a capitalized name, *e.g.* P. For example, the RTK-MAPK signal transduction pathway is a system, and will be denoted as

$$RTK\_MAPK\_pathway$$

Each constituent molecule in the pathway is a process, too, as are its domains. For example, the growth factor unbound ligand, receptor tyrosine kinase, and Ras molecules will be denoted as $Free\_Ligand$, RTK, and Ras, respectively. Similarly, the extracellular, intra-cellular and transmembranal domains of the receptor molecule will be denoted by $Extra$, $Transmem$, and $Intra$ processes.

We next define a system process as a collection of molecule processes, occuring in parallel. This concurrency is denoted by the PAR operator, $|$, inserted between the process names. For example,

$$\begin{gathered} RTK\_MAPK\_pathway ::= \\ Free\_ligand \mid RTK \mid RTK \mid RTK \mid Ras \mid Ras \mid Ras \mid Ras\ . \end{gathered}$$

denotes that the RTK-MAPK pathway is composed of several ligand, RTK, and Ras molecules.

Analogously, we define a protein molecule as several domain processes composed in parallel. For example, we represent a homo-dimer ligand composed of two identical binding domains, and a receptor composed of intra-cellular, transmembranal and extracellular domains by

$$\begin{aligned} Free\_ligand &::= Free\_binding\_domain \mid Free\_binding\_domain \ . \\ RTK &::= Extra \mid Transmem \mid Intra \ . \end{aligned}$$

3.2 Molecular complementarity as communication channels

Two molecules (or domains) interact with each other based on their structural and chemical complementarity [46]. We abstract molecular complementary as complementary offers on communication channels. For example, if the ligand's free binding domain and the receptor's extracellular domain have complementary binding motifs, we denote this motif pair as a *ligand_binding* channel, which appears as an output *ligand_binding* ! [ ] offer in the ligand's *Free_binding_domain* process and as an input *ligand_binding* ? [ ] offer in the receptor's *Extra* process:

$$\begin{aligned} Free\_binding\_domain &::= ligand\_binding \ ![\ ] \cdots \\ Free\_extracellular\_domain &::= ligand\_binding \ ?[\ ] \cdots \end{aligned}$$

Molecules (processes) with complementary motifs (communication offers) are graphically depicted in Figure 4.

The motifs on molecules may often vary based on different biochemical modifications. We often abstract such motifs as *channel parameters* of the process. For example, the intra-cellular domain of the receptor may have three potential binding motifs, which we will represent as parameters when defining the process:

$$Free\_intracellular\_domain(motif1, motif2, motif3) ::= \cdots$$

Then, when we *create* the *Free_intracellular_domain* process as part of a particular *RTK*, we replace these parameters with actual channels, representing the specific state for those motifs, *e.g.*

$$RTK ::= \ Free\_intracellular\_domain(p\_tyr1, tyr2, sh2) \mid \cdots$$

Molecule processes with motif channel parameters are graphically depicted in Figure 4C.

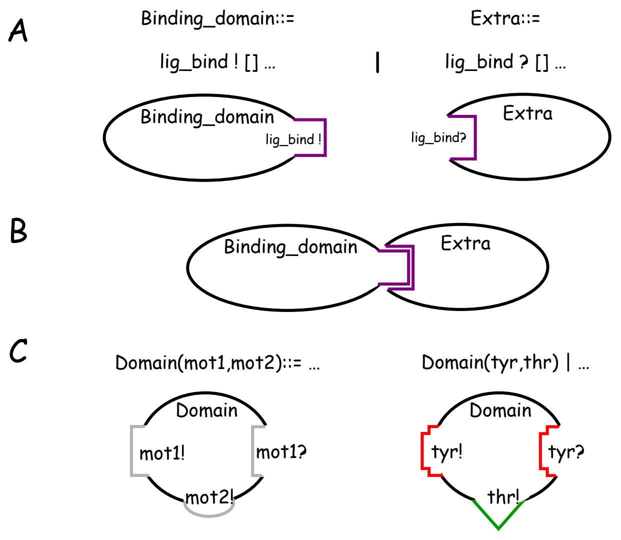

Fig. 4. Molecular complementarity as input and output channels. A. The *Free_binding_domain* and the *Extra* processes (ovals) have complementary output (protrusion) and input (depression) offers on the *ligand_binding* channel. B. Process (molecular) interaction is enabled by channel (motif) complementarity. C. We may define a process with channel parameters (left, grey), to be instantiated with actual channels (right, color) only when the process is *created*.

3.3 Sequential and mutually exclusive events

Biochemical events may occur in sequence, in parallel with other independent events, or in a mutually exclusive, competitive fashion. We abstract a sequence of interactions as a sequence of input and output offers, separated by a prefix operator: "," (the coma sign). For example, if the extracellular domain may first bind a ligand, and then bind to another domain of another molecule, we define

$$Extra ::= \ ligand\_binding \ ? \ [\,],\ rtk\_binding \ ? \ [\,], \ \cdots$$

In other cases, there may be several alternative interactions in which a molecule may participate: once one occurs, the other options are no longer possible. Such mutually exclusive offers are *summed* together, using the "+"

or ";" *choice* operator[4]. For example, if the receptor may be able to bind either the (agonist) ligand or an antagonist but not both, we write

$$\begin{aligned} Extra ::= {} & ligand\_binding\ ?[\,],\ rtk\_binding\ ?[\,],\ \cdots + \\ & antagonist\_binding\ ?[\,],\ \cdots \end{aligned}$$

Finally, several interactions may occur in parallel, without directly affecting each other. Such cases will be handled by composing different offers in parallel, as shown above.

3.4 Compartmentalization as private channels

A pathway is not merely a bag of molecules and their domains. Rather, it is composed of defined compartments, from individual molecules, through molecular complexes form, to cellular compartments, often bounded by membranes. In all cases molecules which share a common compartment may interact with each other, while molecules excluded from the compartment may not [24]. We abstract only the limits that compartment impose on communication by introducing "private" channels with restricted communication scopes([50] will discuss an extension that handles compartments directly).

A private channel x is created using the "new x" operator. For example, if we wish to represent the fact that the three domains of a receptor are linked by a common backbone and belong to a single molecule, we introduce a new *backbone* channel, that is shared by the three processes, and is distinct from all other (external) channels:

$$RTK ::= \ (\mathsf{new}\ backbone)(Extra|Transmem|Intra).$$

The private *backbone* channel can be used for communication only between the three sub-processes of one RTK, and thus represents the limits that a shared backbone poses on interaction.

3.5 Interaction and biochemical modification as communication

Each biochemical interaction in a system may affect subsequent interactions in several ways. First, interaction is often accompanied by modification of one molecule by the other, such that the modified molecule may now be complementary to (and interact with) other motifs, which it was not "suitable for" before (and vice versa). Second, a particular interaction may lead to the exclusion of other possibilities. Finally, an interaction may lead to a general

[4] We will use both notations interchangeably. For their syntactic distinction, see the implementation section.

"state change" in both interacting molecules, enabling events that were disabled before.

We will represent these three results of interaction (and modification) by the communication rules of the π-calculus, using either either a *non-mobile* or a *mobile* (message passing) abstraction.

Non-mobile communication

In the abstraction of interaction as *non-mobile* communication we represent modification and state change *together* by the introduction of a newly created (or re-created) process into the system. The different channel set owned by the created process will represent the modified residues in the corresponding molecule. For example, consider an active protein kinase (*Active_kinase* process) and a target binding domain (*Mod_Bind_domain* process). The binding domain has a motif (*phosph_site* ?) that may be identified and bound by a complementary motif (*phosph_site* !) in the kinase. The kinase may then phosphorylate a tyrosine residue in another motif in the binding site. This modification is represented by the creation of a new instance of a different process type (*Phospho_Bind_domain*), that uses the *phosphotyrosine* channel rather than the *tyrosine* one.

$$\begin{aligned} Active\_kinase &::= phosph\_site \; ! \; [\;] \; , \; Kinase \\ Mod\_Bind\_domain &::= phosph\_site \; ? \; [\;] \; , Phospho\_Bind\_domain \\ Phospho\_Bind\_domain &::= phosphotyrosine \; ! \; [\;] \; , \; \cdots \end{aligned}$$

If an interaction takes place between *Active_kinase* and *Mod_Bind_domain*, two things will happen simultaneously. First, both the input and output events on the *phosph_site* channel will be removed. Second, the remainder of the two processes will be allowed to continue: creating a *Kinase* process and a *Phospho_Bind_domain* process. The latter one uses a the *phosphotyrosine* channel, representing the modification. Overall, the non-mobile communication process can be summarized in the following reduction:

$$\begin{gathered} phosph\_site \; ! \; [\;] \; , \; Kinase \quad | \quad phosph\_site \; ? \; [\;] \; , \; Phospho\_Bind\_domain \\ \longrightarrow \\ Kinase \quad | \quad Phospho\_Bind\_domain \end{gathered}$$

The newly created *Phospho_Bind_domain* can use its *phosphotyrosine* channel for further communications with other processes which harbor this motif, such as the $SH2$ process:

$$SH2 ::= \mathit{phosphotyrosine} \; ? \, [\,] \; , \; \cdots$$

The full sequence of events is summarized in Figure 5.

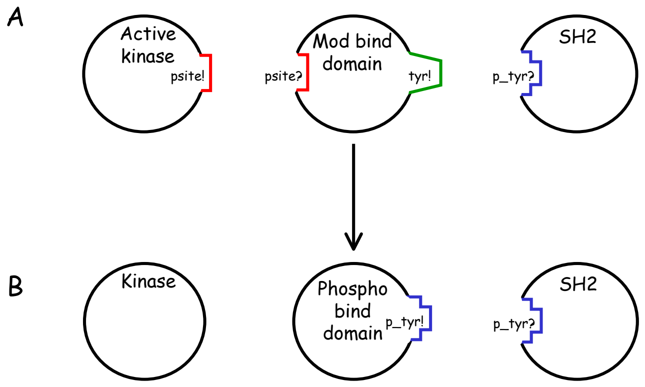

Fig. 5. Interaction and modification as communication and state change. A. Pre-communication. Three processes (*Active_kinase*, *Mod_Bind_domain*, $SH2$) and their respective channels. *Active_kinase* and *Mod_Bind_domain* may interact on *phosph_site*. B. Post-communication. A *Phsopho_Bind_domain* process is introduced into the system instead of the *Mod_Bind_domain* process. It may communicate with $SH2$ on *p_tyr*.

Mobile communication

In the mobile abstraction we represent *modification* as sending and receiving *messages*. Each message is composed of one or more channel names, representing the modified motif(s). These names are received into "placeholders" in the receiving process which they substitute, representing the modification. Then, they may be used by the receiving process in subsequent communication, representing the effect of modification on subsequent interactions. For example, consider again the active protein kinase (*Active_kinase* process) and a target binding domain (*Mod_Bind_domain* process), where the kinase may phosphorylate a tyrosine residue in another motif in the binding site. In the mobile approach, this modification is represented by the sending of a *p_tyr* message, to be received by the *tyr* "placeholder":

$$\mathit{Active\_kinase} ::= \mathit{phosph\_site} \; !\{\mathit{p\_tyr}\} \; , \; \mathit{Kinase}$$
$$\mathit{Mod\_Bind\_domain} ::= \mathit{phosph\_site} \; ?\{\mathit{tyr}\} \; , \; \mathit{tyr}!\{\cdots\} \; , \; \cdots$$

If an interaction takes place between *Active_kinase* and *Mod_Bind_domain*, the received *p_tyr* channel will replace the *tyr* channel in the receiving *Mod_Bind_domain*. Overall, the communication-with-message process can be summarized in the following reduction:

$$\begin{array}{c} phosph\_site\ !\{p\_tyr\}\ ,\ \cdots \quad | \quad phosph\_site\ ?\{tyr\}\ ,\ tyr!\{\cdots\}\ ,\ \cdots \\ \rightarrow \\ Kinase\ \ |\ \ p\_tyr!\{\cdots\}\ ,\ \cdots \end{array}$$

representing the interaction, modification and release events involved in the chemical reaction in the system. Most importantly the *p_tyr* received by the *Mod_Bind_domain* process may now be used for further communications with other processes which harbor this motif, such as the *SH*2 process:

$$SH2 ::= p\_tyr?\{\cdots\}\ ,\ \cdots$$

The full sequence of events is summarized in Figure 6.

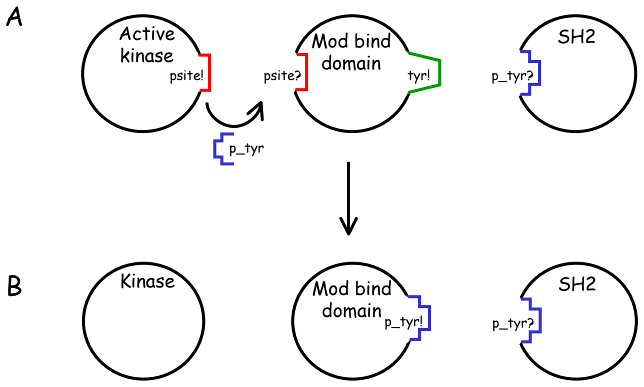

Fig. 6. Interaction and modification as communication and mobility. A. Pre-communication. Three processes (*Active_kinase*, *Mod_Bind_domain*, *SH*2) and their respective channels. *Active_kinase* and *Mod_Bind_domain* may interact on *phosph_site*. B. Post-communication. The *tyr* channel in *Mod_Bind_domain* is replaced with *p_tyr*, which may be used for communication with *SH*2.

3.6 Compartment change as extrusion of a private channel's scope

We also use the *mobility* mechanism to abstract compartment changes, such as complex formation. For example, consider a three-molecule complex that forms between a dimeric ligand and the extracellular domain of two RTK receptors. As a result of complex formation, the two RTK domains may interact "privately", without the participation of any other RTK molecule. To represent this, the private *backbone* channel is sent from the *Ligand*'s *Binding_domain* sub-processes to the *RTK*s' *Extra* processes:

$$\begin{gathered} Ligand ::= \ (\mathsf{new}\ backbone)(Binding\_domain \mid Binding\_domain). \\ Binding\_domain ::= \ ligand\_binding\ !\ \{backbone\},\ Bound\_domain. \\ Extra ::= \ ligand\_binding?\{cross\_backbone\}, Bound\_Extra(cross\_backbone). \end{gathered}$$

As a result of two communication events (of each of the *Binding_domain*s with a different *RTK*'s *Extra* sub-process, we end up with two *Extra* processes which both "own" the same private *backbone* channel, representing their indirect link via a commonly bound ligand molecule.

$$\begin{gathered} (\mathsf{new}\ backbone) \\ (\ ligand\_binding\ !\ \{backbone\},\ Bound\_domain \mid \\ ligand\_binding\ !\ \{backbone\},\ Bound\_domain\ ) \mid \\ ligand\_binding\ ?\ \{cross\_backbone\}\ ,\ Bound\_Extra(cross\_backbone) \mid \\ ligand\_binding\ ?\ \{cross\_backbone\}\ ,\ Bound\_Extra(cross\_backbone) \\ \longrightarrow \\ (\mathsf{new}\ backbone) \\ (\ Bound\_domain \mid Bound\_domain \mid \\ Bound\_Extra(backbone) \mid Bound\_Extra(backbone)\ \ ) \end{gathered}$$

The *Bound_Extra* processes will subsequently use this private *backbone* channel to interact with each other, without external interruptions.

3.7 Molecular objects as parametric processes

The use of parametric process definitions allows us to abstract the constant identity of a molecule (process) as its structure (public channels) and compartment (private channels) change with interaction. For example, we may define the *Mod_Bind_domain* described above (which can either interact with a modifying kinase, or bind to a SH2 domain) as:

$$\begin{gathered} Mod\_bind\_domain(kinase\_site, sh2\_site) ::= \\ kinase\_site?\{mod\_sh2\_site\}, Mod\_bind\_domain(kinase\_site, mod\_sh2\_site) + \\ sh2\_site\ ?\ [\,],\ \cdots \end{gathered}$$

3.8 Competition as choice

A molecule may often participate in one of several mutually exclusive interactions. We abstract this by the *choice* operator (denoted ";" or "+"). For example, consider a *System* in which the *Extra* process may interact either with an agonist *Ligand*'s *Binding_domain* (on the *ligand_binding* channel), or with an *Antagonist* (on the *antagonist_binding* channel):

$$\begin{aligned} System ::= &\quad Extra \mid Ligand \mid Antagonist\ . \\ Extra ::= &\quad ligand\_binding\ !\ [\,]\ ,\ Extra\_Bound\_Agonist\ + \\ &\quad antagonist\_binding\ !\ [\,]\ ,\ Extra\_Bound\_Atagonist. \\ Ligand ::= &\quad ligand\_binding\ ?\ [\,]\ ,\ Bound\_ligand. \\ Antagonist ::= &\quad antagonist\_binding\ ?\ [\,]\ ,\ Bound\_antagonist. \end{aligned}$$

If *ligand_binding* is chosen, then the *antagonist_binding* option is discarded and the resulting system is

$$Extra\_Bound\_Agonist \mid Ligand\_bound \mid Antagonist$$

Vice versa, if *antagonist_binding* is chosen, then the *ligand_binding* option is discarded and the resulting system is

$$Extra\_Bound\_Antagonist \mid Ligand \mid Bound\_antagonist$$

Choice is resolved non-deterministically: All enabled communications (where both input and output are available) are equi-potent. A more biologically realistic model would assign different probabilities to different interactions, based on reaction rates. We extend the language to handle such a model in the next section.

4 Simple examples

The general principles outlined in Sections 2 and 3 allow us to formally represent detailed information on complex pathways, molecules and biochemical events. In this section we illustrate these capabilities with several small programs representing different aspects of molecular systems.

4.1 Molecular complexes

Our first two examples handle the formation and breakage of a bi-molecular complex. To represent the formation and breakage of a complex between two molecules, Molecule1 and Molecule2, we use both public and private channels (Figure 7). Each of the molecules is represented by a process ($Molecule1$ and $Molecule2$, Figure 8A). The two processes share a public *bind* channel, on which one process ($Molecule1$) is offering to send a message, and the other ($Molecule2$) is offering to receive (Figure 8A) . These complementary communication offers represent the molecular complementarity of the two molecules, and the communication event represents binding. The private *backbone* channel sent from $Molecule1$ to $Molecule2$ represents the formed complex, and the two processes change to a "bound" state ($Bound\_Molecule1$ and $Bound\_Molecule2$, Figure 8B). A communication between the two "bound" processes on the shared private *backbone* channel represents complex breakage. As a result, the two processes return to the initial "free" state ($Molecule1$ and $Molecule2$), completing a full cycle. Note, that in a system with many copies of these processes, any two particular copies of $Molecule1$ and $Molecule2$ may communicate (*i.e.* "bind") on the *bind* channel. However, the two resulting "bound" processes share a private channel, which is distinct from all other channels, and may allow only this particular pair to communicate ("unbind") with each other. Three such complexes and their specific private channels are shown in Figure 8B.

```
System::= Molecule1 | Molecule2 .
Molecule1::= (new backbone) . bind ! {backbone} ,
                           Bound_Molecule1(backbone) .
Bound_Molecule1(bb)::= bb ! [ ] , Molecule1 .
Molecule2::= bind ? {cross_backbone} ,
                           Bound_Molecule2(cross_backbone) .
Bound_Molecule2(cbb)::= cbb ? [ ] , Molecule2 .
```

Fig. 7. **π-calculus code for a heterodimer complex**

Special case: Homodimerization

The π-calculus program presented above for a heterodimer must be modified to represent homodimerization (Figure 9). The reason is that a homodimer forms between two identical molecules that are represented by two copies of the same process. Since it is impossible *a priori* to break down the symmetry between the two components of the homodimer, both input and output must be offered simultaneously by the same process, analogously to the way molecules should have a "zipper" like structure to allow homodimerization. The asymmetry is broken by using *non-deterministic choice*: while

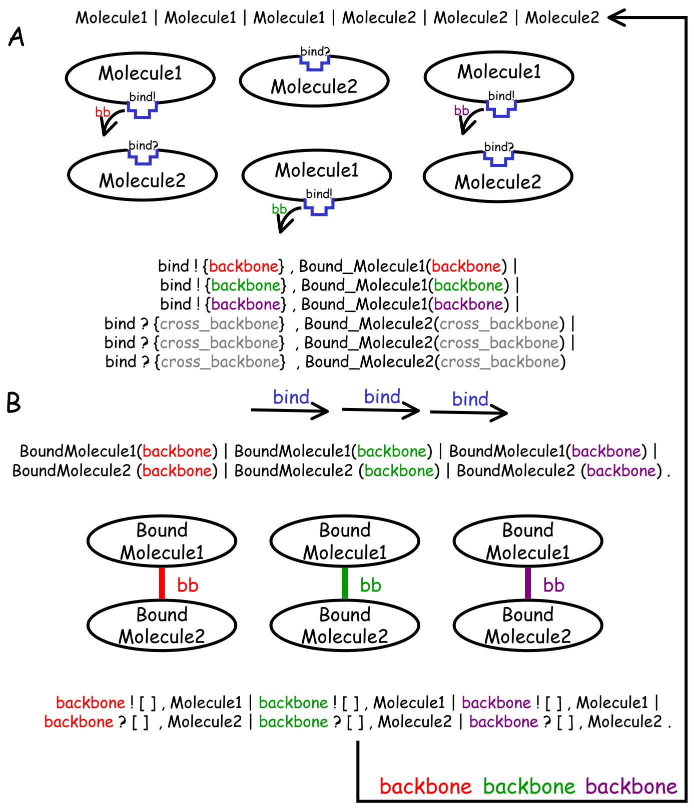
A
Molecule1 | Molecule1 | Molecule1 | Molecule2 | Molecule2 | Molecule2
Molecule1
Molecule2
Molecule1
bind!
bind?
bb
Molecule2
Molecule1
Molecule2
bind ! {backbone} , Bound_Molecule1(backbone) |
bind ! {backbone} , Bound_Molecule1(backbone) |
bind ! {backbone} , Bound_Molecule1(backbone) |
bind ? {cross_backbone} , Bound_Molecule2(cross_backbone) |
bind ? {cross_backbone} , Bound_Molecule2(cross_backbone) |
bind ? {cross_backbone} , Bound_Molecule2(cross_backbone)
B
bind
BoundMolecule1(backbone) | BoundMolecule1(backbone) | BoundMolecule1(backbone) |
BoundMolecule2 (backbone) | BoundMolecule2 (backbone) | BoundMolecule2 (backbone) .
Bound Molecule1
bb
Bound Molecule2
backbone ! [ ] , Molecule1 | backbone ! [ ] , Molecule1 | backbone ! [ ] , Molecule1 |
backbone ? [ ] , Molecule2 | backbone ? [ ] , Molecule2 | backbone ? [ ] , Molecule2 .
backbone backbone backbone

Fig. 8. Formation and breakage of a heterodimer complex. A. Separate molecules. B. Three complexes, represented by distinct private *backbone* channels.

both processes offer both a send and a receive communication, one will (non-deterministically) send and the other receive. Since once an option is chosen, the other is automatically withdrawn, a process cannot communicate with itself (in the same sense that a molecule cannot bind onto itself). In the example (Figure 9), each *Molecule* process offers a choice between sending and receiving a message on the *bind* channel. Once one is selected, the other is withdrawn, and the *Molecule* changes to a "bound" state (*Bound_Mol* process). The message, as in heterodimerization, is a private *backbone* channel, to be used for an "unbinding" communication. This, too, is done by a symmetric choice construct.

```
System::= Molecule | Molecule | Molecule | Molecule | Molecule .
Molecule::= (new backbone) . bind ! {backbone} , Bound_Mol(backbone);
                             bind ? {cbb}      , Bound_Mol(cbb) .
Bound_Mol(bb)::= bb ! [ ] , Molecule ;
                 bb ? [ ] , Molecule .
```

Fig. 9. π-calculus code for a homodimer complex

4.2 Enzymes

Our next use-case tackles "classical", single-substrate, enzymatic reactions typical of metabolic pathways.

The Michaelis-Menten mechanism provides a detailed account of an enzymatic reaction [62], breaking it to its elementary steps. A single-substrate reversible enzymatic reaction includes four elementary reactions: binding of an enzyme to the substrate or to the product and formation of an EX complex, and release of either a product or a substrate from this complex. In this case (coded in Figure 10, and schematically depicted in Figure 11), we have five process types, representing the free enzyme (*Enzyme*), bound enzyme (EX), substrate (*Substrate*), product (*Product*), and intermediate (X). *Enzyme* includes a *choice* between an an interaction with *Substrate* (on *bind_s*) and an interaction with *Product* (on *bind_p*). In both cases, two private channels, *rel_p* and *rel_s*, are sent from *Enzyme* to its counterpart (Figure 11A). Following communication, *Enzyme* changes to EX ("bound enzyme"), and its counterpart (be it *Substrate* or *Product*) changes to X (reaction intermediate) (Figure 11B). The two private channels shared between EX and X represent the formed complex, and are used to finish the reaction, resulting in *Enzyme* reconstitution and either *Product* or *Substrate* release (Figure 11C). Note, that as a result of the non-determinism of the *choice* construct, *Substrate* binding to *Enzyme* may end up either as a *Product* (by reaction of X and EX on *rel_p*) or be released as an intact *Substrate* (reaction of X and EX on *rel_s*). The same is true for *Product*.

4.3 Enzymes in signal transduction

Enzymatic reactions play a critical role in signal transduction (ST) pathways. However, unlike metabolic pathways where enzymes (proteins) and substrates (metabolites) are distinct kinds of biochemical entities, in ST pathways most substrates are proteins, serving as binding partners, transcription factors, or enzymes. We represent such "modification of modifiers" by extensively using the *mobility* mechanism of the π-calculus.[5] Previously, we distinguished the

[5] An alternative non-mobile representation of such events was discussed above, and will not be presented in further detail.

```
System::= Enzyme | Substrate .
Enzyme::= (new rel_s, rel_p) .
               bind_s ! {rel_s,rel_p} , EX(rel_s,rel_p) ;
               bind_p ! {rel_s,rel_p} , EX(rel_s,rel_p) .
EX(release_s,release_p)::= release_s ! [ ] , Enzyme ;
                           release_p ! [ ] , Enzyme .
Substrate::= bind_s ? {erel_s , erel_p} , X(erel_s , erel_p) .
Product::= bind_p ? {erel_s , erel_p} , X(erel_s , erel_p) .
X(rel_es,rel_ep)::= rel_es ? [ ] , Substrate ;
                    rel_ep ? [ ] , Product .
```

Fig. 10. π-calculus code for single-substrate enzymatic reactions. An elementary (Michaelis-menten) reaction model for a single-substrate reversible reaction with one product.

substrate from the product by using distinct process names. In representing the modification of ST molecules, we maintain process identity throughout modification.

Phosphorylation and de-phosphorylation by protein kinases and phosphatases

We consider a toy example involving a binding protein (*Binding_Protein* process), a protein tyrosine kinase (*Kinase* process) and a protein tyrosine phosphatase (*Phosphatase* process). In this system, the protein kinase phosphorylates a tyrosine residue, and the phosphatase de-phosphorylates it. For simplicity, we handle the enzymatic reaction as a "single-step" reaction, rather than the elaborate model discussed above (Figure 12 and Figure 13).

We represent the modifiable residue as two channels: *tyr* for the non-phosphorylated residue, and *p_tyr* for the phosphorylated one. *Kinase* sends *p_tyr* as a message on the *tyr* channel, representing the fact that the tyrosine kinase identified non-phosphorylated tyrosines and modifies them to the phosphorylated form. Conversely, *Phosphatase* sends a *tyr* message on the *p_tyr* channel.

The *Binding_Protein* has a channel parameter, representing the phosphorylation state of its tyrosine residue. When the system is initialized we assume that it is non phosphorylated, so we create a *Binding_Protein*(*tyr*). This process offers to receive on *tyr*, and may thus interact with *Kinase*, but not with *Phosphatase*. Upon communication, the *Binding_Protein* receives *p_tyr*, and is re-created, but now with the newly received channel (*i.e. Binding_Protein*(*p_tyr*)). This process offers to receive on *p_tyr*, and may thus interact with *Phosphatase*, but not with *Kinase*. As before, upon communication, *Binding_Protein* is re-created with the received *tyr* channel, returning to the initial state.

In this way, the iteration of *Binding_Protein* between *tyr* and *p_tyr* parameters represents the cycling of the protein molecule between a phospho-

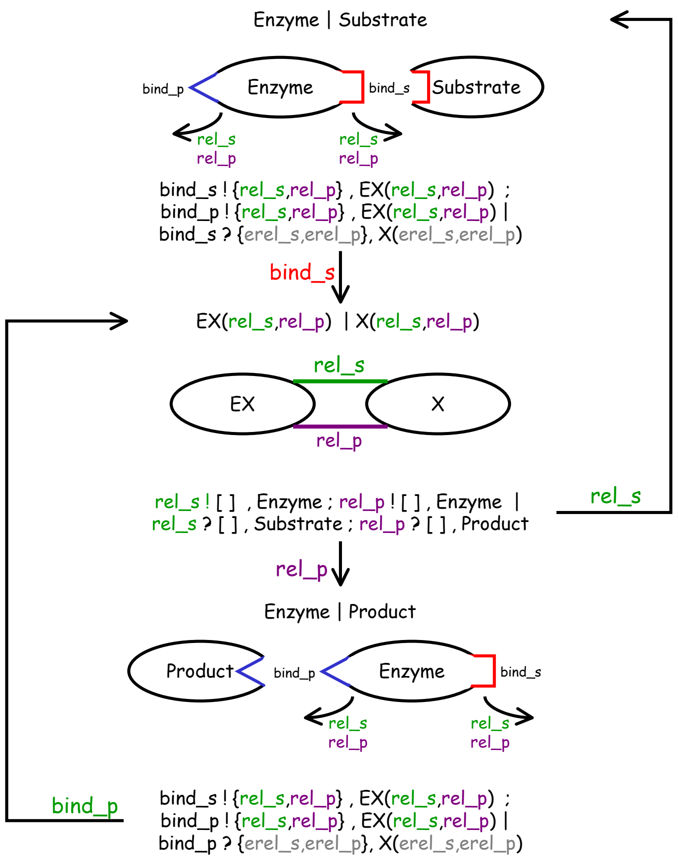
Enzyme | Substrate
bind_p
Enzyme
bind_s
Substrate
rel_s
rel_p
rel_s
rel_p
bind_s ! {rel_s,rel_p} , EX(rel_s,rel_p) ;
bind_p ! {rel_s,rel_p} , EX(rel_s,rel_p) |
bind_s ? {erel_s,erel_p}, X(erel_s,erel_p)
bind_s
EX(rel_s,rel_p) | X(rel_s,rel_p)
rel_s
EX
X
rel_p
rel_s ! [ ] , Enzyme ; rel_p ! [ ] , Enzyme |
rel_s ? [ ] , Substrate ; rel_p ? [ ] , Product
rel_s
rel_p
Enzyme | Product
Product
bind_p
Enzyme
bind_s
rel_s
rel_p
rel_s
rel_p
bind_p
bind_s ! {rel_s,rel_p} , EX(rel_s,rel_p) ;
bind_p ! {rel_s,rel_p} , EX(rel_s,rel_p) |
bind_p ? {erel_s,erel_p}, X(erel_s,erel_p)

Fig. 11. Reversible, single-substrate enzymatic reaction with a single product: Elementary reaction model. A. *Enzyme* may interact with either *Substrate* (on *bind_s*) or *Product* (on *bind_p*). B. Following communication, *Enzyme* becomes *EX*, while *Substrate* (or *Product*) becomes "intermediate" *X*. C. *X* (intermediate) and *EX* (bound enzyme) may interact either on *rel_p* (releasing *Product*) or *rel_s* (releasing *Substrate*).

```
System::= Binding_Protein(tyr) | Kinase | Phosphatase .
Binding_Protein(res)::= res ? {mod_res} , Binding_Protein(mod_res) .
Kinase::= tyr ! {p_tyr} , Kinase .
Phosphatase::= p_tyr ! {tyr} , Phosphatase .
```

Fig. 12. π-calculus code for phosphorylation and de-phosphorylation of a binding domain. Model is based on a simplified (single step) enzymatic reaction.

rylated and a non-phosphorylated state. These channel parameters affect the communication capabilities of *Binding_Protein*, reflecting the effect of the residue's modification state on the protein's ability to be identified by kinases and phosphatases. This principle can be extended and incorporated into more detailed models. For example, we can devise a model that combines the elementary reaction (Michaelis-Menten) scheme with the *tyr*/*p_tyr* channel parameter scheme.

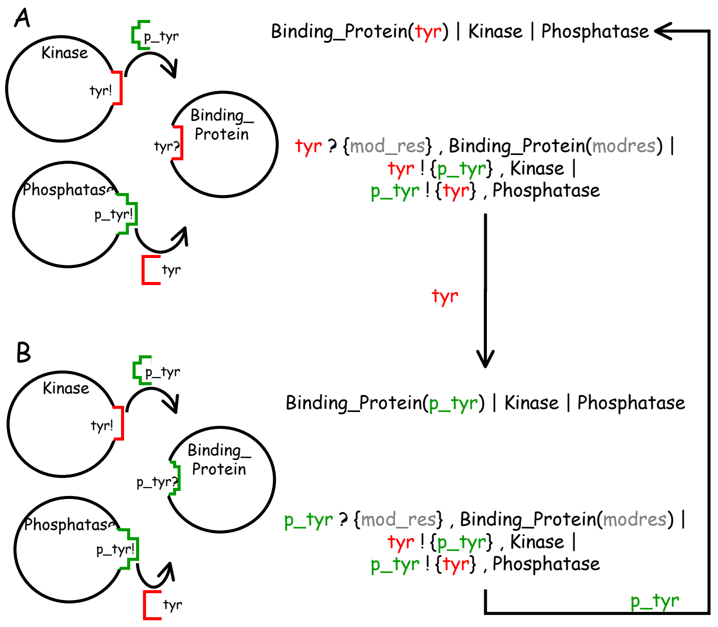
A
Kinase
tyr!
p_tyr
Binding_ Protein
tyr?
Phosphatase
p_tyr!
tyr
Binding_Protein(tyr) | Kinase | Phosphatase
tyr ? {mod_res} , Binding_Protein(modres) |
tyr ! {p_tyr} , Kinase |
p_tyr ! {tyr} , Phosphatase
tyr
B
Kinase
tyr!
p_tyr
Binding_ Protein
p_tyr?
Phosphatase
p_tyr!
tyr
Binding_Protein(p_tyr) | Kinase | Phosphatase
p_tyr ? {mod_res} , Binding_Protein(modres) |
tyr ! {p_tyr} , Kinase |
p_tyr ! {tyr} , Phosphatase
p_tyr

Fig. 13. Modification of a protein residue by kinases and phosphatases. A. A system composed of a *Kinase*, *Phosphatase* and a *Binding_Protein* with a *tyr* channel. B. Following communication between *Kinase* and *Binding_Protein* on *tyr*, *Kinase* is recreated, and *Binding_Protein* now has a *p_tyr* channel, rather than a *tyr* one, allowing interaction with *Phosphatase*. C. Following communication between *Phosphatase* and *Binding_Protein* on *p_tyr*, *Phosphatase* is recreated, and *Binding_Protein* now has a *tyr* channel, rather than a *p_tyr* one, returning to the original state (A).

Modifiable residues in activity and binding

The *channel parameter* approach can be extended to handle the effect of modified residues on the binding and activity of the proteins that harbor them. For example, assume that the phosphorylated tyrosine (but not the non-phosphorylated one) of the binding Protein discussed above can be bound by an SH2 domain in an adaptor protein. To model this situation we now extend the previous program (Figure 14).

```
System::= Binding_Domain(tyr_mod,tyr_bind) |
          Kinase | Phosphatase | Adaptor .
Binding_Domain(res_mod,res_bind)::=
   res_mod ? {res_mod1,res_bind1} ,
           Binding_Domain(res_mod1,res_bind1) ;
   res_bind ? {unbind} ,
           Bound_Domain(res_mod,res_bind,unbind) .
Bound_Domain(res_mod,res_bind,ub)::=
   ub ? [ ] , Binding_Domain(res_mod,res_bind) .
Kinase::= tyr_mod ! {p_tyr_mod, p_tyr_bind} , Kinase .
Phosphatase::= p_tyr_mod ! {tyr_mod,tyr_bind} , Phosphatase .
SH2_Adaptor::= (new unbind) . p_tyr_bind ! {unbind} ,
                                 Bound_Adaptor(unbind) .
Bound_Adaptor(ub)::= ub ! [ ] , SH2_Adaptor .
```

Fig. 14. π-calculus code for phosphorylation and de-phosphorylation of a binding domain

Rather than using a single channel parameter in *Binding_Protein* (either *tyr* or *p_tyr*), we now introduce two channel parameters. The first parameter, *res_mod* (either *tyr_mod* or *p_tyr_mod*), is used for communication with *Kinase* and *Phosphatase*. The second parameter, *res_bind* (either *tyr_bind* or *p_tyr_bind*), represents the ability (or lack thereof) to bind to the adaptor protein. *Kinase* and *Phsophatase* send a tuple of two channels to *Binding_Protein*, representing the fact that residue modification affects both the ability to interact with the modifying enzyme and to bind to the adaptor protein.

5 The biochemical stochastic π-calculus: A quantitative extension

The original framework of the π-calculus is semi-quantitative by definition: all *individual* communications are equally likely to occur. Thus, the π-calculus abstraction of the molecular realm reflects the number of molecules, but not the rates of the reactions in which these molecules participate. This results in two inter-dependent inaccuracies. First, reactions do not occur at the correct rate. Second, all time steps are equal and do not represent the time evolution of the real system.

While previous studies with qualitative (*e.g.* [39]) and semi-quantitative (*e.g* [23] and our work [52]) modeling of biomolecular systems show that even such coarse abstractions have some merit, quantitative aspects are key to the function of many biomolecular systems, a fact that is recently gaining growing recognition (*e.g.* [32]), and must be appropriately addressed when devising a relevant abstraction of biomolecular systems.

In the π-calculus abstraction, we have associated our representation so far with a particular non-deterministic and discrete semantics. While the discrete aspect of the calculus is fundamental, the non-deterministic semantics is not, and will now be replaced with a stochastic one, in which different communications have different rates, and communications are selected based on probabilistic conditions. The programming language mechanism is realized using a previously proven Monte Carlo algorithm for the stochastic simulation of systems of coupled chemical reactions, known as the Gillespie algorihtm [18].

To provide the π-calculus with a stochastic extension we introduce several changes into the π-calculus domain and the π-calculus based abstraction:

- **Channels with base rates as elementary reactions with mesoscopic rate constants.** Each channel is now associated with a *base rate*. The channel's base rate is identical to the mesoscopic rate constant of the corresponding elementary reaction.
- **Channels' actual rates as reactions' actual rates.** At each state in the π-calculus system we determine the actual rate of a channel based on that channel's base rate, and the number of input and output offers on the channel at that state (which represent the number of reactant molecules in the corresponding reaction). The actual channel rate is identical in calculation and value to the actual rate of the corresponding reaction.
- **π-calculus time steps as the time evolution of chemical system.** The discrete even time steps implicit to the unfolding of a semi-quantitative simulation[6] are replaced by an explicit clock. The clock is advanced in uneven steps, according to the Gillespie algorithm, based on the actual channel rates. The resulting time evolution of the abstract π-calculus system corresponds to the time evolution of a single (representative) trajectory of the chemical system.
- **Stochastic selection of communication according to the probability of a reaction.** The non-deterministic way in which *enabled* communication actions are chosen at each step in the unfolding of a π-calculus system is replaced by a stochastic selection process, based on the actual

[6] The semantics of the π-calculus and similar languages is originally purely concurrent, where events can happen in parallel. In practice, however, concurrency is implemented by an *interleaving semantics*, where systems unfold in a step-by-step manner, with a single event chosen at each step. Here we refer to this standard, well-accepted, approach.

channel rates at each state of the system. The selection process is identical to the one specified in the Gillespie algorithm. The resulting state evolution of the π-calculus system corresponds to the state evolution of a (statistically representative) trajectory of the chemical system.

The actual rates of chemical reactions are dependent on their rate costants (our "base rate") and reactant quantities. However, we distinguish several types of elementary reactions, each of which uses a different kind of rate calculation rule and different abstraction guidelines (summarized in Table 2). Briefly, each reaction type will be represented by a different channel type, and channel type will determine which rate calculation rule to apply. Asymmetric bimolecular reactions involving two reactants from two different species are represented by two different processes using a regular type channel. Symmetric bimolecular reactions involving two reactants from the same species are represented by two identical processes using a symmetric type channel (each offering a choice between a send and a receive action). Unimolecular reactions involve only a single reactant. As the π-calculus allows only pair-wise interaction, we represent these either as an asymmetric communication (on a regular channel) with an additional, single copy, *Timer* process, or by a symmetric communication (on a *private* symmetric channel, one per molecule) between two sub-processes of the *Reactant* process. Finally, we also allow instantaneous reactions, occuring immediately when enabled, on an instantenous type channel. These are highly useful for various encoding needs, as well as when we would like to focus on slower events.

The different ways in which the actual rate of channels is calculated for different types of reactions, and the different handling of communications on instantaneous channels, mean that a given channel name may only be used in one way throughout the unfolding of a stochastic π-calculus system. Thus, once a channel name is used as one type, it may not be used as another.

6 BioSpi: Simulating π-calculus programs

Once a detailed π-calculus model of a particular pathway is built, one would like to utilize it in different ways. The most natural use is to treat the model as an executable computer program, and simulate the behavior of the pathway by running the program.

To this end, we have developed a computer application, called BioSpi. BioSpi is based on the Logix system [60], which implements Flat Concurrent Prolog (FCP [59]). Two unique features of FCP made it suitable for our purposes. First, the ability to pass logical variables in messages was used to implement the mobility ("sending channels as messages") mechanism of the π-calculus. Second, its support of guarded atomic unification allowed the implementation of synchronized interaction with both input and output guards. Note, that previous implementations of the π-calculus or of related formalisms

Table 2. Bimolecular, unimolecular and instantaneous reactions in the stochastic extension of the π-calculus

Reaction type	**Channel type**	**Explanation**
Asymmetric bimolecular reaction	Regular	One reactant is represented by a process offering to send on the reaction channel and the other by a process offering to receive on the reaction channel. **Rate**: Base rate × #senders × #receivers
Symmetric bimolecular reaction	Symmetric	The reactants are represented by two instances of the same process, each offering a *choice* between send and a receive on the reaction channel. **Rate**: Base rate × $\frac{\#\text{senders}\times(\#\text{receivers}-1)}{2}$ (The number of senders and receivers is equal)
Unimolecular reaction	Regular (public)	The single reactant is represented by a process offering to receive on the public reaction channel. A single *Timer* process offers the complementary communication. **Rate**: Base rate × 1 × #receivers
	Regular (private)	The internally interacting parts of a single reactant are represented as two parallel processes communicating on a shared private reaction channel. Each potential "copy" of the reaction is represented by a separate private channel. **Rate**: Base rate × 1 × 1 (An actual rate is calculated for each individual private reaction channel)
Instantaneous reaction	Instant-aneous	Same as for bimolecular reaction **Rate**: Infinite: Executed immediately when enabled, prior to any reaction on channels with finite rates, and without advancing the clock

(*e.g. [41]* and references therein) do not provide such full guarded synchronous communication.

BioSpi receives as input π-calculus code and executes it. An appropriate surface syntax was devised for the π-calculus syntax [49], allowing also for a simple but useful arithmetic extension.[7] The ASCII-based[8] syntax was devised in such a way that the original π-calculus primitives and operations (including *match/mismatch* constructs) are clearly separated from the arithmetic ones. Furthermore, the syntax is insulated from general Logix procedures, and a

[7] The arithmetic extension allows us to employ variables and arithmetically manipulate their values as well as set conditions on these values.

[8] In order to maintain pure ASCII representation and comply with some limitations of the Logix system, some of the original π-calculus notation was replaced in the BioSpi syntax. Only BioSpi notation is used throughout this work with the exception of the (new x) construct

pure π-calculus representation is maintained in spite of the use of an FCP-based platform.

In the simulation, each instance of a π-calculus process is realized as a running computational one. Processes run concurrently and interact using channel objects, following the reaction rules of the calculus. The simulation follows, step by step, the evolution of the system. At each step, a single communication (reaction) step is realized in one atomic operation: a pair of complementary communication actions (input and output on the same channel in two concurrent processes) is chosen, a message is transmitted between the processes, the communication actions and alternative *choices* are eliminated, the received channel(s) are substituted (*instantiated*) for the appropriate place holder, and the processes are allowed to continue.

BioSpi has several versions, consistent with our biologically-motivated extensions of the π-calculus. In BioSpi 1.0, communication selection is non-deterministic, and all enabled communication actions are equally likely to be selected. As a result, the simulation process is *semi-quantitative*: communication (*i.e.* reaction) rates are effectively identical, therefore process (*i.e.* molecule) numbers determine which reactions are more probable than others, and higher-copy-number processes are more likely to participate in communication than lower-copy ones.

Several tools are available for tracing the execution of BioSpi 1.0 programs. Thus, not only the net outcome of a computation can be studied, but also the specific scenario that has led to this outcome. First, the simulation may be executed in a step-by-step fashion, with the ability to set specific break points. This approach is useful to follow small detailed systems. Second, an ordered trace of all the communications in the system and the processes that participated in them is maintained. The trace can be viewed in a form of a tree, ordered chronologically, or according to process evolution. Third, the simulation can be suspended at any desired moment, and the contents (so-called *resolvent*) of the system's current state of computation are examined. The level of detail in which a system is traced (process, channels, messages, senders, etc.) can be determined dynamically throughout a session. We applied the BioSpi 1.0 system to study a complex full model of the RTK-MAPK pathway, gleaning meaningful biological insights even in this limited, semi-quantitative framework (see [52] for details).

BioSpi 2.0 implements the Gillespie algorithm and serves as a stochastic simulation platform. In BioSpi 2.0, each channel object is also associated with a base rate and type (instantaneous, symmetric or asymmetric). The channel base rate is supplied when the channel is defined. An "infinite" base rate serves to define an instantaneous channel type. Asymmetric and symmetric channel types are identified upon their first use.

The central monitor in BioSpi 2.0 maintains the simulation clock and is responsible for the correct execution of the Gillespie algorithm: the selection of the next communication and time step. The monitor operates in the following way: Requests to instantaneous channels are satisfied as soon as possible.

Requests to channels which have a finite rate (> 0) are queued. Each time that a new event is required the central monitor and all channel objects with a finite, non-zero rate, jointly determine a communication event. In each iteration, each channel object determines its actual rate, according to its type, its basal rate, and the numbers of send and receive offers. The monitor then uses these actual rates to select the next reaction channel and time step, exactly according to the selection procedure of the Gillespie algorithm [18]. The monitor then advances a "clock" counter according to the selected time step and allows the chosen channel to complete one transmission (send/receive pair), relaying the sent message to the receiver.[9] As in BioSpi 1.0, the completion of the send and receive requests is synchronized by the channel. In addition, other messages offered on this and other channels by the same two processes whose requests were completed, are withdrawn. The withdrawals are not synchronized, but they do precede continuation of their respective processes, and the next step of the clock.

In addition to the tracing and debugging tools available for BioSpi 1.0 simulations, BioSpi 2.0 allows us to maintain a full *record* of the time evolution of the system. The record specifies each communication in the system, the time at which it occured, the communicating processes, the channel on which the communication occured and the resulting processes. The record file can be post-processed to generate the time evolution of the systems state (*i.e.* number of each process species at each time point) at any desired level of resolution. In the next section, we will see the utility of the record and trace tools in studying the stochastic behavior of specific systems.

Importantly, the changes in the stochastic extension of the π-calculus abstraction are mostly confined to the *semantic* interpretation of π-calculus programs, and to the *pragmatic* use of the abstraction (*e.g.* the representation of unimolecular reactions). As a result, the new guidelines for stochastic abstraction can be seamlessly composed on top of the existing semi-quantitative ones. Practically, this means that with the addition of appropriate channel rates, any semi-quantitative π-calculus abstraction (BioSpi 1.0-compatibale) can become stochastic (and BioSpi 2.0-compatible).

[9] To be fully accurate, the selection of the send/receive pair should be done in a fully random way. In the current implementation, the pair is selected in a "top-of-the-queue" fashion, which is non-random. In most cases, there are only two process species communicating on a given channel, and this does not raise a problem, as all senders (or receivers) are equivalent. In some cases, however, this may be inaccurate. We are currently developing a fully random selection of a communication pair to be available in the next release.

7 Studying biomolecular systems with BioSpi 2.0: A compositional abstraction of glycogen biosynthesis

The incorporation of a stochastic semantics allows us to build more accurate abstractions of biomolecular systems. In this section we study glycogen biosynthesis as one example of these benefits. In this example, we use the distinction between channels with finite rates and instantaneous channels to build a detailed model of glycogen biosynthesis. We also extensively use an arithmetic extension of the calculus and its ability to abstract the *composition* of entities of growing complexity from simple building blocks.

7.1 Glycogen biosynthesis

Polymerization is a key event in the creation of various biomolecules including DNA, RNA, proteins, various prtoein filaments (*e.g.* actin, microtubuli) and polysaccharides. In contrast to proteins and nucleic acids, polysaccharides can form both branched and linear polymers.

Glycogen is a branched polysaccharide composed of glucose monomers [68]. Glycogen is biosynthesized in a combination of two alternative enzymatic reactions. In the first *elongation* reaction, catalyzed by the glycogen synthase enzyme, an $\alpha(1 \rightarrow 4)$ glycosidic bond is formed between the C_4 of the polymerized glucose at the end of a growing polymer and the C_1 of an "incoming" UDP-glucose monomer.[10] In the second *branching* reaction, catalyzed by the glycogen branching enzyme, an $\alpha(1 \rightarrow 6)$ glycosidic bond is formed between the C_6 of a polymerized glucose within the glycogen polymer and the C_1 of an "incoming" glycogen oligomer. The oligomer is branched "off" an existing polymer chain. In addition, the glycogen polymer is *initiated* by a third enzyme, called glycogenin. This enzyme forms an initial 7-residue primer, which we term a *seed*.

The biosynthesis of glycogen follows specific "rules". First, the specificity of glycogen synthase ensures that elongation is allowed only from growing 4' ends. After branching, there may be more than one such end, and elongation may proceed independently at each of these ends. The branching "rules" are more complex. A branch is specifically created by transferring a 7-residue segment from the end of one chain to the $C_6 - OH$ group of a polymerized glucose residue on the same or another glycogen chain. Each transferred segment must come from a chain of at least 11 residues, and the new branch point must be at least 4 residues away from any other branch point.

The balance between elongation and branching determines the architecture of the glycogen molecule: chain length and extent of branching. This architecture determines the compactness of glucose storage and its availability and is thus critical for the functional role of glycogen as an energy reserve.

[10] The glucose $\rightarrow$ UDP-glucose reaction is catalyzed by a separate enzyme which we will not discuss here. We will use the terms glucose and UDP-glucose interchangeably from here on.

7.2 Building the abstraction in the π-calculus

The first step in building an abstraction is identifying and defining the basic entities in the real world domain. To this end, we now break down the above description into several components.

First, each glycogen strand is directional (Figure 15). We term one end, with a C_1 carbon that cannot grow, as the *root side*. We term the other end, with a C_4 carbon that can grow, as the *leaf side*.

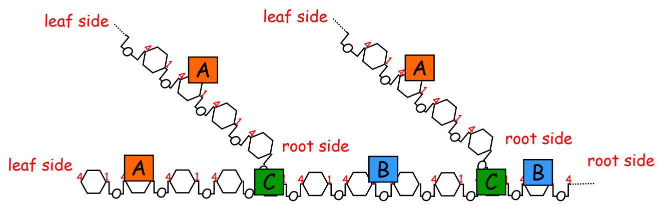

Fig. 15. The architecture of a glycogen molecule: definitions. The C_1 and C_4 end of a glycogen strand defined as "Root" and "Leaf" respectively. A polymerized residue may assume one of three potential positions: on a straight segment (A), on a branched segment (B) or at a branch point (C).

Second, we distinguish three potential positions for a polymerized residue (Figure 15). First, it may reside on a *straight segment*, where no branch points exist to the leaf side. Second, it may be part of a *branched segment*, that has a branch point to the leaf side. Third, the residue may be the *branch point* itself.

The exact position of the residue determines the types of reactions in which it may participate. A *polymerization enabled* residue may participate in an elongation reaction (as the C_4 donor). A *cleavage enabled* residue may be cleaved in a branching reaction and linked into another glycogen branch. A *branch enabled* residue may serve as a branch point onto which a new glycogen oligomer is added. The various types of residues and their reaction capabilities are listed in Table 3.

We are now ready to abstract the system in the stochastic π-calculus. First, each glucose residue is abstracted as a process. Different process states represent the different positions of a glucose residue (Table 3, rightmost column). Processes also represent the two types of enzyme molecules (*Glycogen_Synthase* and *Branching_Enzyme*) and the seed (*Seed_Glucose* for the "free" seed and *Root_Glucose* for the polymerized seed). The formation of bonds is abstracted as private channels, with one private channel shared between each pair of immediate neighbors. Thus, each process representing a fully polymerized residue has two private channels: one linking it with its

Table 3. Different types of residues defined by their reaction capabilities and abstracted as π-calculus processes.

General position	Residue capabilities	Detailed position	Process name
Non-polymerized	Polymerization enabled	Free UDP-glucose (1' end)	*UDP_Glucose*
Straight segment	Polymerization enabled	Leaf (4' end)	*Leaf_Glucose*
	Cleavage and branch enabled	At distance 7 exactly from leaf and at distance 4 at least from closest branch point or root	*BCE_Glucose*
	Branch but not cleavage enabled	At distance other than 7 from leaf and at distance 4 at least from closest branch point or root	*BNCE_Glucose*
	Disabled	Not leaf and distance less than 4 from closest branch point or root	*Disabled_Glucose*
Branched segment	Branch but not cleavage enabled	At distance at least 4 from both flanking branch points/root	*BNCE_Glucose*
	Disabled	At distance less than 4 from at least one of its flanking branch points/root	*Disabled_Branched_Glucose*
Branch point	Disabled	At the branch point (C_6 donor)	*Branch_Point*

root-side neighbor and the other with its leaf-side neighbor. The processes representing the root and leaf processes share only a one private channel with a neighbor (Figure 16).

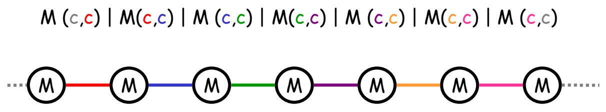
M (c,c) | M(c,c) | M (c,c) | M(c,c) | M (c,c) | M(c,c) | M (c,c)
M
M
M
M
M
M
M

Fig. 16. A polymer abstracted as a chain of processes linked by private channels. Circles represent processes. Connecting colored lines represent individual private channels.

The exact position of a residue is critical to determine its behavior. Similarly, the exact "position" of the corresponding process is critical to determine its state. We abstract a residue's position in the glycogen polymer by the *values* of three *position variables* that are associated with each individual residue process (Figure 17): A

- **Leaf counter (LC)** measures the distance of the residue from the leaf. It is initialized to zero, incremented (+1) upon extension, and decreased (-7) upon cleavage in the remaining segment residues.
- **Root counter (RC)** measures the distance of the residue from root or from the flanking branch point on the root side. It is initialized to the value of the RC variable of the neighbor residue on the root side + 1, updated upon cleavage in the cleaved segment residues to the RC of the new root side neighbor + 1, and updated upon insertion in the segment on the leaf side of the new branch point to RC - (RC of new branch point).
- **Leaf-branch counter (LBC)** measures the distance of the residue from the flanking branch point on the leaf side. It is initialized to zero and updated upon insertion in the segment on the root-side of the new branch point to the LBC of the leaf side neighbor+1.

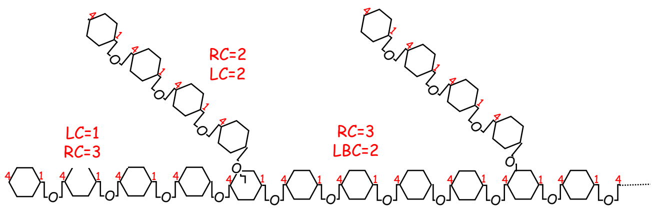

Fig. 17. The leaf (LC), root (RC), and leaf-branch (LBC) counters: An example. The values of the LC, RC, and LBC counters for several residues are shown.

As counters abstract the position of each residue in the polymer, and as the residue's position determines its state, we use the counters to define the corresponding process state. We use a *choice* construct, in which the counter values are checked according to the different rules:

$$\begin{array}{ll}
\{LBC\ =\ 0\}\ , & \text{\%Straight segment} \\
& (\ \{LC\ =\ 0\}\ ,\ Leaf\_Glucose\ ; \\
& \{LC = 7\ ,\ RC >= 4\}\ ,\ BCE\_Glucose\ ; \\
& \{LC > 0\ ,\ LC = / = 7\ ,\ RC >= 4\}\ ,\ BNCE\_Glucose\ ; \\
& \{LC > 0\ ,\ RC < 4\}\ ,\ Disabled\_Glucose\ )\ ; \\
\{LBC > 0\}\ , & \text{\%Branched segment} \\
& (\ \{RC >= 4\ ,\ LBC >= 4\}, BNCE_{G}lucose; \\
& \{RC < 4\}\ ,\ Disabled\_Branched\_Glucose; \\
& \{LBC < 4\}\ ,\ Disabled\_Branched\_Glucose\ )\ .
\end{array}$$

The choice construct is examined each time that the counter value changes.

The position of a residue may change *directly*, when the residue participates in an interaction (polymerization or cleave/branch) or *indirectly*, when the residue is part of a segment which participates in those reactions (*e.g.* upon branching). In order for the abstract counters to correctly represent the position of the corresponding residues in the "real world" polymer, they must be constantly updated. Since only two processes participate in the immediate interaction, we incorporate an *update propagate mechanism* into our abstract model.

Counter updating will be performed by communication on the private channels linking the process residues. Importantly, all such private channels will be instantaneous, while all the communications representing *actual* reactions (elongation, cleavage and branching) will be carried out on channels with a real finite rate. As a result, counter updating will be done in zero simulation time, and will not interfere with the kinetics of biochemical reactions, while allowing us to maintain a detailed view of the glucose monomer's position within the glycogen polymer.

The full abstraction of the glycogen biosynthetic system in the π-calculus is given in Figure 18. The enzymatic elongation reaction is represented by the *glycogen* and *udp_glucose* channels, and the cleave and branch reaction by the *branch* and *cleave* channels. Each *UDP_Glucose* process is defined with two private channels (*to_root* and *to_leaf*) that will be subsequently used as specific links to residues in the resulting polymer. For simplicity, all the "reaction" channels are given an equal base rate of 1 (we will revisit this decision below), while all private channels are instantaneous with an infinite base rate.

Abstraction of initiation

In the initiation step (after a seed oligomer is already produced by glycogenin), glycogen synthase catalyzes the formation of a glycosidic bond between a UDP-glucose and the 4' leaf end of the seed oligomer. In the abstract π-calculus model we represent the initiation as two consecutive communication steps. In the first step, the *to_root* private channel of *UDP_Glucose* is sent to *Glycogen_Synthase* on the *udp_glucose* channel. In the second step, this private channel name is further relayed in a communication from *Glycogen_Synthase* to *Seed_Glucose* on *glycogen* (Figure 19A and B).

Following the two communications, *UDP_Glucose* and *Seed_Glucose* now share a private (instantaneous) channel. In the next two steps, the position counters of each of the processes are updated to reflect their relative position using this shared channel(Figure 19C and D). First, the *RC* counter is sent on the private channel from *Seed_Glucose* to *UDP_Glucose*, and is used to update *UDP_Glucose*'s *RC* value. This represents the change in the position of this now polymerized residue relative to its newly acquired root. Then, the *LC* and *LBC* counters are sent on the private channel from *UDP_Glucose* to the now *Root_Glucose*, and are used to update the respective counters in *Root_Glucose*. This represents the concomitant change in the position of the root oligomer relative to its newly acquired leaf. Finally ((Figure 19E), the new counter values are evaluated in *UDP_Glucose*, changing it to its new *Leaf_Glucose* state.

Abstraction of elongation

Elongation is similar to initiation and involved the same processes and communications, with the exception of *Seed_Glucose* which is "replaced" by *Leaf_Glucose*. Two consecutive communications (first on *udp_glucose* and then on *glycogen*) serve to relay a private channel from *UDP_Glucose* to *Leaf_Glucose* via *Glycogen_Synthase*. This private channel is then used to initiate an update of the positional counters of *all* the processes along the growing chain (each time using the relevant private channel). Finally, the updated counters determine the new state of each of the linked processes.

Abstraction of branching

Consider a relatively simple branching scenario in which a 12-residue chain is cleaved at the fifth residue. The 4-residue "rooted" part is then elongated by

```
global(glycogen(1), udp_glucose(1), dummy(1), branch(1), cleave(1)).
baserate(infinite).

Seed_Glucose(RC,LC,LBC)::= glycogen ? {to_leaf},to_leaf ! {RC,LBC},
                                   Root_Glucose(to_leaf,RC,LC,LBC).
Root_Glucose(to_leaf,RC,LC,LBC)::=
   to_leaf ? {LC,LBC} , {LC++} , Root_Glucose(to_leaf,RC,LC,LBC) .
UDP_Glucose(LC,LBC)+(to_root,to_leaf)::=
   udp_glucose ! {to_root} , to_root ? {RC,LBC} , {RC++} ,
         to_root ! {LC,LBC} , Glucose(to_root,to_leaf,RC,LC,LBC) .
Glucose(to_root, to_leaf, RC, LC, LBC)::=
{LC>=0},
<< {LBC = 0} , << {LC = 0} ,  Leaf_Glucose ;
            {LC = 7 , RC >= 4} , BCE_Glucose ;
  {LC > 0 , LC =\= 7 , RC >= 4} , BNCE_Glucose ;
  {LC > 0 , RC < 4} , Disabled_Glucose >> ;
   {LBC > 0} , << {RC >= 4 , LBC >=4} , BNCE_Glucose ;
          {RC < 4} , Disabled_Branched_Glucose ;
  {LBC < 4} , Disabled_Branched_Glucose >> .
```

Fig. 18. π-calculus code for the glycogen system. The entire system is shown with the exception of initialization of position counters. The code is continued on the next page.

```
Leaf_Glucose::=
   glycogen ? {to_leaf} , to_leaf ! {RC,LBC} , to_leaf ? {LC,LBC} , {LC++} ,
                            to_root ! {LC,LBC} , Glucose(to_root,to_leaf,RC,LC,LBC);
   to_root ? {RC,_} , <<  {RC >=0} , {RC++} , Glucose ;
  {RC < 0} , Disabled_Leaf_Glucose >> .
Disabled_Leaf_Glucose::=
   to_root ? {RC,_} , {RC++} , Glucose .
BNCE_Glucose::=
   to_leaf ? {LC,LBC} , {LC++} , << {LBC = 0} , to_root ! {LC,LBC} , Glucose ;
               {LBC > 0} , {LBC++} ,  to_root ! {LC,LBC} , Glucose >> ;
   to_root ? {RC,_} , << {RC >=0} , {RC++} , to_leaf ! {RC,LBC} , Glucose ;
 {RC < 0} , to_leaf ! {RC, LBC} , Disabled_Glucose >> ;
   branch ? {to_branch} , Branch_Synch1(to_branch,RC,LC,LBC) .
Branch_Synch1(to_branch,RC,LC,LBC)+(RC1,LBC1)::=
  {RC1=0} | {LBC1=1} |
    << to_branch ! {RC1,LBC} ,
        << to_leaf ! {RC1,LBC} , to_root ! {LC,LBC1} ,
             Branch_Point(to_root,to_branch,to_leaf) >> >>  .
Disabled_Glucose::=
   to_leaf ? {LC,LBC} , {LC++} ,
         << {LBC = 0} , to_root ! {LC,LBC} , Glucose ;
    {LBC > 0} , {LBC++} ,  to_root ! {LC,LBC} , Glucose >> ;
   to_root ? {RC,_} , {RC++} , to_leaf ! {RC,LBC} , Glucose .
BCE_Glucose+(new_to_root,RC1,LC1,LBC1)::=
   << to_leaf ? {LC,LBC} , {LC++} , << {LBC = 0} , to_root ! {LC,LBC} , Glucose ;
                 {LBC > 0} , {LBC++} ,  to_root ! {LC,LBC} , Glucose >> ;
      to_root ? {RC,_} , << {RC >=0} , {RC++} , to_leaf ! {RC,LBC} , Glucose ;
    {RC < 0} , to_leaf ! {RC,LBC} , Disabled_Glucose >> ;
      branch ? {to_branch} , Branch_Synch(to_branch,RC,LC,LBC) ;
      cleave ! {new_to_root} , {LC1 = -1} | {RC1 = -1} | Cleave_Synch(to_leaf) .
Cleave_Synch(to_leaf)::=
   to_root ! {LC1,LBC} , to_leaf ! {RC1, LBC} , new_to_root ? {RC,_} ,
{RC++} , to_leaf ! {RC,LBC} , Glucose(new_to_root,to_leaf, RC, LC, LBC) >> .
Branch_Synch(to_branch,RC,LC,LBC)+(RC1,LBC1)::=
   {RC1=0} | {LBC1=1} |
                 << to_branch ! {RC1,LBC} ,
              << to_leaf ! {RC1,LBC} ,
 to_root ! {LC,LBC1} , Branch_Point(to_root,to_branch,to_leaf) >> >> .
Disabled_Branched_Glucose::=
   to_leaf ? {LC,LBC} , {LC++} ,
        << {LBC = 0} , to_root ! {LC,LBC} , Glucose ;
           {LBC > 0} , {LBC++} ,  to_root ! {LC,LBC} , Glucose >> ;
   to_root ? {RC,_} , {RC++} , to_leaf ! {RC,LBC} , Glucose >> .
Branch_Point(to_root,to_branch,to_leaf)::=
   to_root ? {_,_} , self ;
   to_branch ? {_,_} , self ;
   to_leaf ? {_,_} , self .
Glycogen_Synthase::=
   udp_glucose ? {to_root} , glycogen ! {to_root} , Glycogen_Synthase  .
Branching_Enzyme::=
   cleave ? {to_branch} , branch ! {to_branch} , Branching_Enzyme .
```

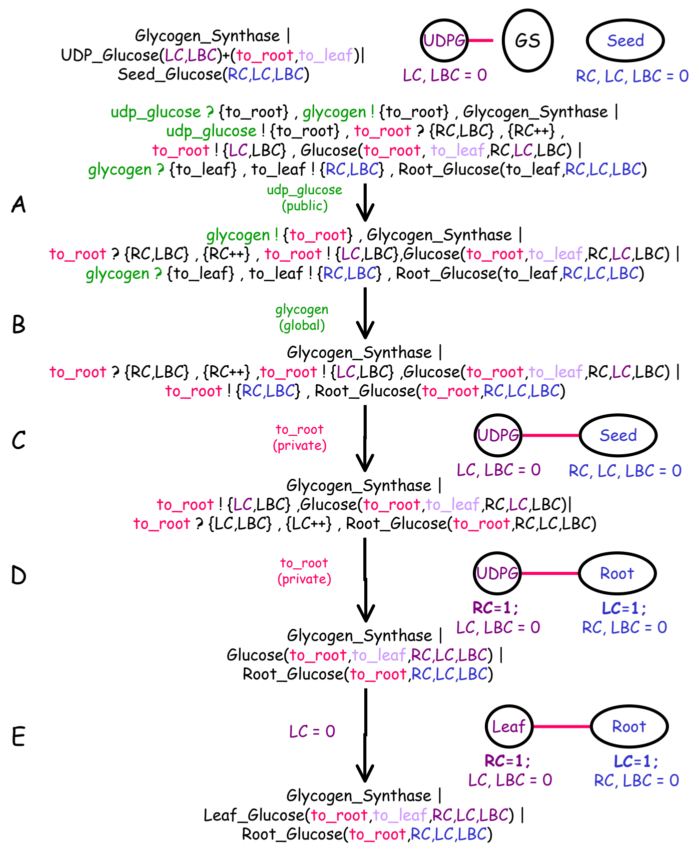
Glycogen_Synthase |
UDP_Glucose(LC,LBC)+(to_root,to_leaf)|
Seed_Glucose(RC,LC,LBC)
UDPG
GS
Seed
LC, LBC = 0
RC, LC, LBC = 0
udp_glucose ? {to_root} , glycogen ! {to_root} , Glycogen_Synthase |
udp_glucose ! {to_root} , to_root ? {RC,LBC} , {RC++} ,
to_root ! {LC,LBC} , Glucose(to_root, to_leaf,RC,LC,LBC) |
glycogen ? {to_leaf} , to_leaf ! {RC,LBC} , Root_Glucose(to_leaf,RC,LC,LBC)
A
udp_glucose (public)
glycogen ! {to_root} , Glycogen_Synthase |
to_root ? {RC,LBC} , {RC++} , to_root ! {LC,LBC},Glucose(to_root,to_leaf,RC,LC,LBC) |
glycogen ? {to_leaf} , to_leaf ! {RC,LBC} , Root_Glucose(to_leaf,RC,LC,LBC)
B
glycogen (global)
Glycogen_Synthase |
to_root ? {RC,LBC} , {RC++} ,to_root ! {LC,LBC} ,Glucose(to_root,to_leaf,RC,LC,LBC) |
to_root ! {RC,LBC} , Root_Glucose(to_root,RC,LC,LBC)
C
to_root (private)
UDPG
Seed
LC, LBC = 0
RC, LC, LBC = 0
Glycogen_Synthase |
to_root ! {LC,LBC} ,Glucose(to_root,to_leaf,RC,LC,LBC)|
to_root ? {LC,LBC} , {LC++} , Root_Glucose(to_root,RC,LC,LBC)
D
to_root (private)
UDPG
Root
RC=1;
LC, LBC = 0
LC=1;
RC, LBC = 0
Glycogen_Synthase |
Glucose(to_root,to_leaf,RC,LC,LBC) |
Root_Glucose(to_root,RC,LC,LBC)
E
LC = 0
Leaf
Root
RC=1;
LC, LBC = 0
LC=1;
RC, LBC = 0
Glycogen_Synthase |
Leaf_Glucose(to_root,to_leaf,RC,LC,LBC) |
Root_Glucose(to_root,RC,LC,LBC)

Fig. 19. Initiation as a multi-step communication between *UDP_Glucose*, *Glycogen_Synthase* **and** *Seed_Glucose***.** A. Communication between *UDP_Glucose* and *Glycogen_Synthase*. B. Communication between *Glycogen_Synthase* and *Seed_Glucose*. C,D. Counter update using a shared private channel. E. *UDP_Glucose* changed to *Leaf_Glucose* based on counter values.

two more residues, followed by linking of the 8-residue "clipped" part on its fifth residue (Figure 20).

A process chain of size 12 is shown in Figure 20, where process state is determined according to its relative position. Each pair of processes is linked by a unique private channel, established during the elongation step, as described above. Cleavage is abstracted as a communication on *cleave* between a *Branching_Enzyme* and a *BCE_Glucose*. Due to the short chain length in this example, there is only a single *BCE_Glucose*. In the communication, a private channel (*new_to_root*) is sent from *BCE_Glucose* to

Branching_Enzyme. Following the communication, the *LC* and *RC* counters of *BCE_Glucose* are set to special values (-1), followed by a series of communications on the chains of private channels to update the relevant positional information. First, the "cleave event" is propagated to the root side of the cleave point. As a result, the immediate root-side neighbor of the cleaved residue process will become a *Leaf_Glucose*. The other root-side processes will be updated accordingly with the communication propagated up to the *Root_Glucose*.

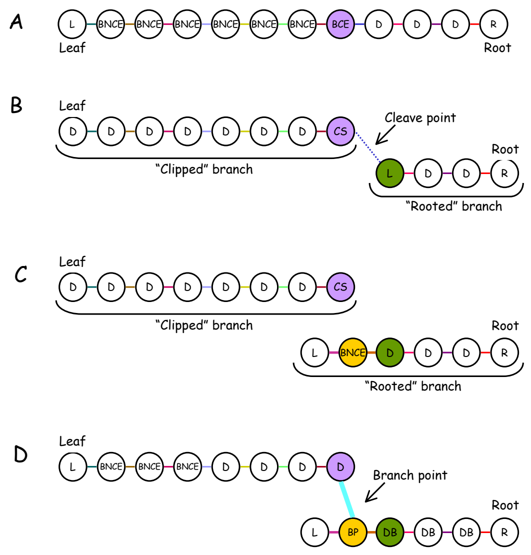

Fig. 20. A simple branch and cleave scenario. A. A chain of length 12. B. Following cleavage: a rooted chain (length 4) and a clipped chain (length 8). C. The rooted chain is elongated by two more residues. D. Linking the clipped chain to residue 5 of the rooted chain. Corresponding process names are shown (R - *Root_Glucose*, L - *Leaf_Glucose*, D - *Disabled_Glucose*, BP - *Branch_Point*, DB - *Disabled_Branched_Glucose*, CS - *Cleave_Synch*)

In parallel, the cleavage event is propagated to the leaf side of the cleavage point. Since cleavage and branching may be separated by additional steps (*e.g.* in the example scenario we have additional elongation of the "rooted branch"), the "clipped" branch is "frozen" until it is be linked to a new position. The "freeze" is based on propagating the special *RC* counter value (-1) throughout the clipped chain. The special value serves to set each process to a *Disabled_Glucose*, and will change only upon updating the positional counters (following branch linking, see below). On the other hand, the "rooted" chain may participate like any other chain in elongation (*e.g.* the two elongation steps in our example scenario).

We now examine how to abstract the linking of a cleaved ("frozen") branch to form a branch point. Assuming two additional elongation steps, we start with a system with two process chains, one representing a "rooted" branch of size 6 and the other representing a cleaved branch of size 8. Branching is represented by a communication between *Branching_Enzyme* and *BNCE_Glucose* on *branch*, in which the private *new_to_root* channel, originating in the "cleaved" *BCE_Glucose* is relayed to *BNCE_Glucose*. This results in a private link between the two *Glucose* processes. A series of communications on private channels propagate the change as an update in positional counters from the cleave point (to the leaf direction, resulting in "unfreezing" of the cleaved branch) and from the branch point (to both the root and leaf direction).

7.3 Studying glycogen metabolism with BioSpi 2.0

The detailed glycogen biosynthesis model, based on a "monomer-as-process" abstraction, allows us to monitor the exact architecture of glycogen molecules through the architecture of the process "polymer". The tracing tools of BioSpi 2.0 provide us with the information (process names, counter values and private channel identity) necessary to reconstruct this architecture.

A simple example, based on a system initialized with a single *Seed_Glucose*, 15 *UDP_Glucose* processes, a single *Glycogen_Synthase* process and a single *Branching_Enzyme* is shown in Figure 21. The processes available in the system after it is run until suspension (no additional enabled communications) are shown in the so-called resolvent in Figure 21A, together with the channels they use and the values for the *RC*, *LC* and *LBC* counters. This information is sufficient to specify the molecular architecture of the corresponding glycogen molecule (Figure 21B).

BioSpi 2.0 also allows us to follow the time evolution of various quantitative properties in the abstracted population of glycogen molecules, such as the number of branch points, leaves and seeds, giving us an overall measure of molecular architecture. For example, Figure 22 shows the number of various processes representing different properties of the polymers (leaves, polymerized residues etc.) under different relative polymerization and branching rates.

8 Perspectives: Molecules as computation

The behavior of biomolecular systems is traditionally studied with *Dynamical Systems Theory* (described via differential equations) [16], which abstracts the cell and its molecular constituents to their quantifiable properties (e.g. concentration, position) and their couplings. While this approach is powerful, A

```
...
.Root_Glucose.comm(.UDP_Glucose.to_root!)
Disabled_Branched_Glucose.comm(.UDP_Glucose.to_root!, .UDP_Glucose.to_root!, 1, 4,
4, global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
Disabled_Branched_Glucose.comm(.UDP_Glucose.to_root!,.UDP_Glucose.to_root!, 2, 3,
3, global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
Disabled_Branched_Glucose.comm(.UDP_Glucose.to_root!,.UDP_Glucose.to_root!, 3, 2,
2, global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
.Branch_Point.comm(.UDP_Glucose.to_root!, BCE_Glucose.new_to_root!,
.UDP_Glucose.to_root!)
Disabled_Glucose.comm(BCE_Glucose.new_to_root!,.UDP_Glucose.to_root!, 1, 8, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
Disabled_Glucose.comm(.UDP_Glucose.to_root!,.UDP_Glucose.to_root!, 2, 7, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
Disabled_Glucose.comm(.UDP_Glucose.to_root!,.UDP_Glucose.to_root!, 3, 6, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
BNCE_Glucose.comm(.UDP_Glucose.to_root!, .UDP_Glucose.to_root!, 4, 5, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
BNCE_Glucose.comm(.UDP_Glucose.to_root!, .UDP_Glucose.to_root!, 5, 4, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
BNCE_Glucose.comm(.UDP_Glucose.to_root!, .UDP_Glucose.to_root!, 6, 3, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
BNCE_Glucose.comm(.UDP_Glucose.to_root!, .UDP_Glucose.to_root!, 7, 2, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
BNCE_Glucose.comm(.UDP_Glucose.to_root!, .UDP_Glucose.to_root!, 8, 1, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
Disabled_Glucose.comm(.UDP_Glucose.to_root!,.UDP_Glucose.to_root!, 1, 1,
0, global.branch(1)!, global.cleave(1)!,global.glycogen(1)!)
Leaf_Glucose.comm(.UDP_Glucose.to_root!, .UDP_Glucose.to_leaf, 2, 0, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
Leaf_Glucose.comm(.UDP_Glucose.to_root!, .UDP_Glucose.to_leaf, 9, 0, 0,
global.branch(1)!, global.cleave(1)!, global.glycogen(1)!)
.Glycogen_Synthase.comm(global.glycogen(1)!, global.udp_glucose(1)!)
.Branching_Enzyme.comm(global.branch(1)!, global.cleave(1)!)
```

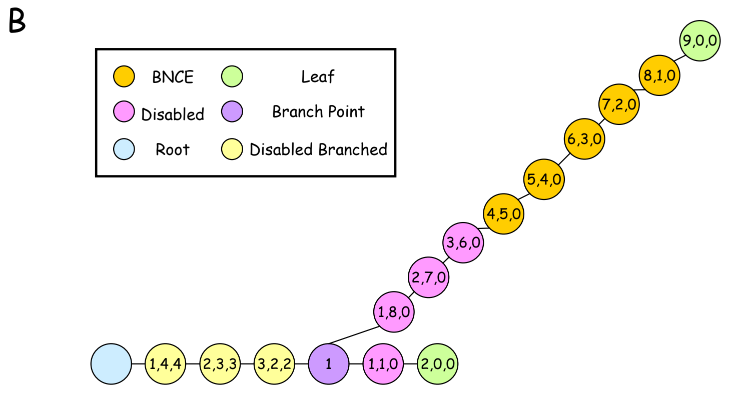

Fig. 21. A BioSpi 2.0 resolvent allows reconstruction of the molecular architecture of a glycogen molecule. A. A BioSpi resolvent following a complete run of a system initialized with a single *Seed_Glucose*, 15 *UDP_Glucose*, a single *Glycogen_Synthase* and a single *Branching_Enzyme*. B. The molecular architecture reconstructed from the resolvent. Counter values for *RC*, *LC* and *LBC* shown inside circles.

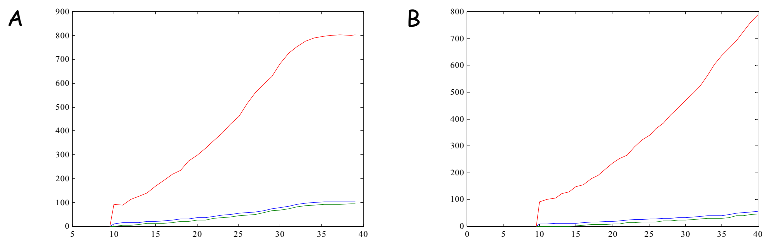

Fig. 22. BioSpi 2.0 simulation of glycogen synthesis under different conditions. The number of *Branch_point* (green), *Leaf_Glucoose* (blue) and "polymerized" (sum of *BCE_Glucose*, *BNCE_Glucose*, *Diabled_Glucose* and *Disabled_Branched_Glucose*) processes (red) is shown as a function of time either when the base rates of the polymerization and branching communications are equal (A) or when the polymerization rate is 10 times faster (B).

clear, and well understood, the abstraction it makes is problematic when describing the behavior of complex biomolecular systems [16]. It treats the cell as a monolithic unstructured entity, and does not identify molecular *objects* as they are modified. Biological systems are composed of dynamic molecular *objects*, but Dynamical Systems Theory is not compositional. Thus, it only indirectly captures the identity and fate of a single molecule, molecular complex or machine with several states (and/or sub-components).

8.1 The molecule-as-computation abstraction

Computer science offers an alternative starting point in the search for good abstractions of biological systems composed of dynamically changing objects. The view of computational systems as composed of interacting processes, renders them a tempting source of novel abstractions for the behavior of systems of interacting biological entities molecules, such as molecules and cells. In this work, we began the search for this abstraction with the π-calculus, a process algebra for the representation and study of *concurrent mobile systems.*

As a first step we identified the basic entities of biomolecular systems and the events in which they participate. We then developed general guidelines for the abstraction of these entities to the mathematical domain of the π-calculus, which we then employed for the modeling and study of real complex biomolecular systems, such as the receptor tyrosine kinase (RTK) - MAP kinase (MAPK) signaling pathway [52]. While the function of certain biomolecular systems may be abstracted in a qualitative or semi-quantitative way, the functionality of others, critically depends on quantitative aspects. We therefore further extended the original abstraction with a stochastic component [47], embedding the well-accepted Gillespie algorithm [18] within the

π-calculus framework. Although the original stochastic abstraction is limited to elementary reactions, BioSpi 2.0 allows, in principle, to define alternative rate calculation rule per (asymmetric) channel and extend the channel type scheme accordingly. In this case, however, the correctness of the simulation of the resulting system is no longer ensured.

8.2 Benefits and limitations

How *relevant*, *computable*, *understandable* and *extensible* is the "molecule-as-computation" abstraction we devised?

Relevance

A *relevant* abstraction of biomolecular systems should capture two essential properties of these systems: their molecular organization and their dynamic behavior. We believe that the π-calculus abstraction is indeed highly relevant. On the one hand, essential biomolecular entities are directly abstracted to computational ones and the abstraction handles a variety of biomolecular events. On the other hand, the dynamic behavior of the derived computational system closely mimics that of the molecular one.

The two alternative (non-mobile and mobile) approaches we present for abstracting molecular interaction offer a tradeoff between relevance and utility. The non-mobile approach is often simpler, but suffers from a *relevance* problem, similar to that of Dynamical Systems Theory, as each state and modification are represented as an individual entity. The mobile approach combines mobility with channel parameters to represent chemical modification as a change in the process' channel set. Thus, processes are under-defined in such a way that allows them to assume different states based on the channels they received. While we believe this approach is more *relevant*, we also appreciate the greater difficulty to specify and understand mobile systems. In practice, when building individual abstractions we use a mixture of the two approaches, guided by pragmatic considerations.

A key benefit of our proposed abstraction is in its ability to handle complex entities, composed on lower-level entities in a dynamic fashion. This is demonstrated by our glycogen biosynthesis example, representing the challenge posed by *open ended* biological entities such as polymers whose boundaries and structure change dynamically and are difficult to pre-define, but are composed of (relatively) simple monomers, whose capabilities are well-known. The *compositionality* of the "molecule as computation" abstraction – the ability to compose complex entities from constituent parts according to pre-defined rules – offers a unique way to handle such entities, as shown with the glycogen biosynthesis model. Note, that the compositional approach has drawbacks as well. First,it requires us to maintain each monomer as an independent entity (process) in the abstract model (and simulation). Second, in a compositional representation all properties are "distributed" among the individual components. In the glycogen example this required constant status updates of a series of counters throughout the affected area. This can pose both a computational burden and affect the readability of the abstraction. One solution to this problem is to build a hybrid abstraction in which sections of the polymer are abstracted as a single process.

Computability

A *computable* abstraction should allow both the simulation of dynamic behavior and qualitative and quantitative reasoning on systems' properties. We have seen that with the aid of BioSpi we can simulate the behavior of biomolecular systems by executing their computational π-calculus abstraction. However, simulations of molecular systems as computational ones can also suffer from performance and scaling problems. The main difficulty lies in the abstraction of each molecule as a separate *instance* of a computational process. One solution is to handle only *molecular species* explicitly, while the specific molecules are represented by counter variables. While this approach significantly improves performance and scalability, it loses some of the immediate transparency of the "molecule-as-process" abstraction.

A π-calculus abstraction also supports qualitative and quantitative reasoning on the modeled system's properties. An extensive behavioral theory has been developed for the mathematical domain of the π-calculus (and similar languages), prior to and regardless of its proposed use for biological systems. This provides methods and tools to *formally verify* certain properties of systems described in the π-calculus (*e.g.* [44], [67]). In principle, we can verify certain qualitative assertions on biomolecular systems (*e.g.* "will a signal reach a particular molecule?") by verifying a corresponding assertion on the π-calculus abstraction of the system (*e.g.* "will a particular communication be realized in a particular process?"). Furthermore, methods exist to *compare* two π-calculus program to determine the degree of mutual similarity of their behavior, termed *bi-simulation* ([41], [44], [67]). Different levels of similarity, of weakening strength, have been defined [41].

While such methods have been generally developed they have not been applied in the past to computational systems at the scale and complexity of the ones we are building as abstractions of biomolecular systems. Furthermore, the types of queries and analysis methods developed for computational purposes may not be the same necessary to elucidate biological questions. In particular, stochastic and quantitative issues (such as the ones explored in the next chapter) were hardly handled in the purely computational setting.

Thus, while we deem these *computable* aspects of the "molecule as computation" abstraction as critical components of its success, they are beyond the scope of this work, and will require further extensive research. Nevertheless, the potential computational possibilities opened by the use of the π-calculus serve as additional support for its importance. Indeed, in recent work several researchers have started to adapt analysis methods from process algebra to the study of biomolecular systems (*e.g.* [4], [9], [17], [45], [12]).

Understandability

An *understandable* abstraction offers a conceptual framework for thinking about the scientific domain: it corresponds well to the informal concepts and ideas of science, while opening up new computational possibilities for understanding it. The properties which lend relevance and computability to the "molecule as computation" abstraction also render it *understandable.* On the one hand, the abstraction attempts to translate our informal knowledge of the biological realm to formal concepts in a transparent and visible way.

On the other hand, the *computational* theory for the study and analysis of concurrent computational systems described above open up new possibilities for understanding molecular systems. For example, once biological behavior is abstracted as computational behavior, we can distinguish between implementaions (related to a real biological system, for example the MAP kinase cascade) and its corresponding specification (related to its biological function, for example, an amplifier or a switch). Then, the theory and methodology of semantic equivalence can be applied in this new context to formally ascribe a biological function to a biomolecular system. Similarly, two molecular-level abstractions of similar systems in different cells or organisms can be compared for their behavioral similarity, to determine their level of *homology* (if evolutionary related) or *analogy* (if evolutionary distinct).

The abstraction we presented in this work however is still *obscure* in certain respects. The use of private channels as an abstraction of biomolecular compartments raises several problems including cumbersome, multi-step encoding of the formation of multi-molecular complexes, and a difficulty to express simultaneous conditions on proximity and biochemical complementarity. Another limitation lies in the need for each communication to involve exactly two processes. While intra-molecular events can still be modeled by using sub-processes, interactions between more than two molecules cannot be abstracted by a single event in the computational realm. Although this limitation is biochemically correct (elementary termolecular interactions are extremely rare), it may raise problems in abstracting systems for which knowledge is limited and cannot be broken down to elementary processes. Another current limitation to the transparency and utility of the abstraction is its purely textual nature and the lack of a graphical component for the specification and visualization of the abstracted system and its unfolding.

Extensibility

Such limitations in the *understandability* of the abstraction can often be overcome by appropriate extensions to the abstraction or to the mathematical domain. Since we do not expect a single concise computational language to be an immediate all-inclusive solution, the desired abstraction (or the mathematical domain) should be easily *extensible* to capture additional real-world properties without introducing major changes to the core abstraction. As we have shown in the stochastic case, the well-defined and concise domain of the π-calculus is relatively amenable to extensions. Indeed, we have recently extended the calculus and the abstraction to directly handle compartments, as will be reported elsewhere [50].

A graphical extension or translation of the abstraction is beyond the scope of this work. Note, however, that graphical variants of the π-calculus have been previously proposed (*e.g.* [40]) and can serve as a starting point for the development of a graphical extension. Alternatively, the abstraction may be translatable into other well-developed graphical formalisms, such as the unified modeling language (UML, [48]).

Comparison to other representations

The two prominent existing computational approaches to biomolecular systems are "dynamical systems theory", based on the "cell as a collection of molecular species" abstraction, and the functional pathway databases, that employ the "molecule as object" abstraction. In order to render the former abstractions *understandable* another layer of representation (graphics, object scheme etc.) is typically required; while in order to render the latter *computable* an additional level of semantics and implementation is required used that adds dynamics and behavioral properties to an essentially static scheme.

Abstract computer languages, such as the π-calculus, offer a synthesis of the benefits of both representations. On the one hand, the molecular world is clearly abstracted into the computational one, in a way which is as *relevant* and *understandable* as possible. On the other hand, the generated abstract model is *computable* allowing both simulation and analysis of the modeled system's behavior.

How does the π-calculus abstraction compare to that offered by other, related languages, such as Petri nets [20], Statecharts [29] and linear logic [16], that are now being used to represent various biomolecular systems? While the abstraction to linear logic has only been rudimentary developed [16], both Petri nets and Statecharts are actively employed, and must be considered as alternatives when evaluating our π-calculus based approach.

Petri nets (*e.g.* [20]) are one of the first uses of the "molecule as computation" abstraction, and have been primarily successful as an abstraction of metabolic systems. The *computability* of this abstraction is well-developed, both in terms of simulation tools (*e.g.* [20]) and verification ones [19]. Furthermore, it is accompanied by a pleasing graphical representation which greatly enhances its utility (and which the π-calculus lacks). However, the Petri net based abstraction is close to a "cell as collection of molecular species", and suffers from the same *relevance* problems: it does not allow for the structured representation of molecular objects (which the π-calculus does).

Statecharts [29] are a highly *relevant* and *understandable* model of biomolecular and multi-cellular systems, providing a structured and intuitive framework for the abstraction of biomolecular systems. The graphical nature of the language is much more developed than the Petri net one and very useful, and a variety of computational tools exist for qualitative simulation and verification. Despite these major advantages, the abstraction lacks in *understandability* as, unlike the π-calculus, it is a purely qualitative framework, and does not currently accommodate process quantities or reaction rates.

Clearly, each of these approaches has a different balance of benefits and limitations, and none represents a final solution to the challenge posed by biological systems. A full understanding of these differences requires further dedicated research. We believe such research is important as a first step towards distilling and synthesizing an optimal abstraction of molecules as computations.

References

1. T. Akutsu, S. Miyano, and S. Kuhara. Algorithms for identifying boolean networks and related biological networks based on matrix multiplication and fingerprint function. *Journal of Computational Biology*, 7(3):331–343, 2000.
2. T. Akutsu, S. Miyano, and S. Kuhara. Algorithms for inferring qualitative models of biological networks. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 5, pages 293–304, Singapore, 2000. World Scientific Press.
3. R. Alves and M. A. Savageau. Extending the method of mathematically controlled comparison to include numerical comparisons. *Bioinformatics*, 16(9):786–798, 2000.
4. M. Antoniatti, B. Mishra, C. Piazza, A. Policriti, and M. Simeoni. Modeling cellular behavior with hybrid automata: bisimulation and collapsing. In *Proceedings of the first international workshop on Computational Methods in Systems Biology*, Lecture Notes in Computer Science. Springer-Verlag, 2003. Accepted for publication.
5. A. Arkin, J. Ross, and H.H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected escherichia coli cells. *Genetics*, 149:1633–1648, 1998.
6. G. D. Bader, I. Donaldson, C. Wolting, B. F. Ouellette, T. Pawson, and C. W. Hogue. Bind-the biomolecular interaction network database. *Nucleic Acids Research*, 29(1):242–245, 2001.
7. U. S. Bhalla and R. Iyengar. Emergent properties of networks of biological signaling pathways. *Science*, 283(5400):381–387, 1999.
8. D. Bray. Reductionism for biochemists: how to survive the protein jungle. *Trends in Biochemical Sciences*, 22(9):325–326, 1997.
9. N. Chabrier and F. Fages. Symbolic model checking of biochemical networks. In *Proceedings of the first international workshop on Computational Methods in Systems Biology*, Lecture Notes in Computer Science. Springer-Verlag, 2003. Accepted for publication.

10. G. Ciobanu. Formal description of the molecular processes. In C.Martin-Vide and Gh. Paun, editors, *Recent Topics in Mathematical and Computational Linguistics*, pages 82–96. Romanian Academy, Bucharest, 2000.
11. G. Ciobanu and M. Rotaru. Molecular interaction. *Journal of Theoretical Computer Science*, 289(1):801–827, 2002.
12. M. Curti, P. Degano, and C.T. Baldari. Casual calculus for biochemical modeling. In *Proceedings of the first international workshop on Computational Methods in Systems Biology*, Lecture Notes in Computer Science. Springer-Verlag, 2003. Accepted for publication.
13. P. Dhaeseleer, S. Liang, and R. Somogyi. Genetic network inference: from co-expression clustering to reverse engineering. *Bioinformatics*, 16(8):707–726, 2000.
14. K. Eilbeck, A. Brass, N. Paton, and C. Hodgman. Interact: an object oriented protein-protein interaction database. In *Intelligent Systems for Molecular Biology*, volume 7, pages 87–94, Palo Alto, 1999. AAAI Press.
15. A. Finney, H. Sauro, M. Hucka, and H. Bolouri. An xml-based model description language for systems biology simulations. Technical report, California Institute of Technology, September 2000. Technical report.
16. W. Fontana and L. W. Buss. The barrier of objects: From dynamical systems to bounded organizations. In J. Casti and A. Karlqvist, editors, *Boundaries and Barriers*, pages 56–116. Addison-Wesley, 1996.
17. G.Ciobanu, V.Ciubotariu, and B.Tanasa. A pi-calculus model of the na pump. In *Genome Informatics*, pages 469–472, Tokyo, 2002. Universal Academy Press.
18. D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
19. C. Girault and R. Valk. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag, 2002.
20. P. J. E. Goss and J. Peccoud. Quantitative modeling of stochastic systems in molecular biology by using stochastic petri nets. *Proceedings of the National Academy of Sciences USA*, 95(12):6750–6755, 1998.
21. P. J. E. Goss and J. Peccoud. Analysis of the stabilizing effect of rom on the genetic network controlling cole1 plasmid replication. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 4, pages 65–76, Singapore, 1999. World Scientific Press.
22. D. Harel and E. Gery. Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42, 1997.
23. K. R. Heidtke and S. Schulze-Kremer. Deriving simulation models from a molecular biology knowledge base. In *Proceeding of the 4th Workshop on Engineering Problems for Qualitative Reasoning of the 16th International Joint Conference on Artificial Intelligence*, 1999.
24. C-H. Heldin and M. Purton, editors. *Signal Transduction*. Chapman and Hall, London, 1996. Modular Texts in Molecular and Cell Biology 1.
25. R. Hofestadt and S. Thelen. Quantitative modeling of biochemical networks. *In Silico Biology*, 1(1):39–53, 1998.
26. M. Holcombe and A. Bell. Computational models of immunological pathways. In M. Holcombe and R. Paton, editors, *Information Processing in Cells and Tissues: Proceeding of IPCAT '97*, pages 213–226, New York, 1998. Plenum Press.

27. T. Igarashi and T. Kaminuma. Development of a cell signalling networks database. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Proccedings of the Pacific Symposium of Biocomputing '97*, pages 187–197, Singapore, 1997. World Scientific Press.
28. N. Kam, I.R. Cohen, and D. Harel. The immune system as a reactive system: Modeling t cell activation with statecharts. *Bulletin of Mathematical Biology*, 2002. to appear. An extended abstract of this paper appeared in the Proceeding of the Symposia on Human-Centric Computing Languages and Environments, pages 15-22, Stresa, Italy, September 2001.
29. N. Kam, D. Harel, and I.R. Cohen. Modeling biological reactivity: Statecharts vs. boolean logic. In *Proceeding of the Second International Conference on Systems Biology*, Pasadena, CA, 2001.
30. P. D. Karp, M. Krummenacker, S. Paley, and J. Wagg. Integrated pathway/genome databases and their role in drug discovery. *Trends in Biotechnology*, 17(7):275–281, 1999.
31. T. Kazic. Semiotes: a semantics for sharing. *Bioinformatics*, 16(12):1129–1144, 2000.
32. H. Kitano. Computational systems biology. *Nature*, 420:206–210, 2002.
33. R. Kuffner, R. Zimmer, and T. Lengauer. Pathway analysis in metabolic databases via differential metabolic display. *Bioinformatics*, 16(9):825–836, 2000.
34. K.M. Kyoda, M. Muraki, and H. Kitano. Construction of a generalized simulator for multi-cellular organisms and its application to smad signal transduction. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 5, pages 317–328, Singapore, 2000. World Scientific Press.
35. H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid petri net representation of gene regulatory network. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 5, pages 341–352, Singapore, 2000. World Scientific Press.
36. H. Matsuno, R. Murakani, R. Yamane, N. Yamasaki, S. Fujita, H. Yoshimori, and S. Miyano. Boundary formation by notch signaling in drosophila multicellular systems: Experimental observations and gene network modeling by genomic object net. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing. In press.*, Singapore, 2003. World Scientific Press.
37. H. H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences USA*, 94(3):814–819, 1997.
38. H. H. McAdams and L. Shapiro. Circuit simulation of genetic networks. *Science*, 269(5224):650–656, 1995.
39. L. Mendoza, D. Thieffry, and E. R. Alvarez-Buylla. Genetic control of flower morphogenesis in arabidopsis thaliana: a logical analysis. *Bioinformatics*, 15(7–8):593–606, 1999.
40. R. Milner. π-nets: a graphical form of the π-calculus. In D. Sannella, editor, *Proceedings of the Fifth European Symposion on Programming (ESOP '94)*, volume 788 of *Lecture Notes in Computer Science*, pages 26–42, Berlin, 1994. Springer-Verlag.
41. R. Milner. *Communicating and Mobile Systems: The π-Calculus.* Cambridge University Press, Cambridge, 1999.

42. P. Nielsen, D. Bullivant, C. Lloyd, D. Nickerson, S. Lett, K. Jim, and P. Noble. The CellML modeling language. http://www.cellml.org/public/specification/, 2000.
43. H. Ogata, S. Goto, W. Fujibuchi, and M. Kanehisa. Computation with the kegg pathway database. *Biosystems*, 47(1–2):119–128, 1998.
44. F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4(6):497–543, 1992.
45. A. Pagnoni and A. Visconti. Detection and analysis of unexpected state components in biological systems. In *Proceedings of the first international workshop on Computational Methods in Systems Biology*, Lecture Notes in Computer Science. Springer-Verlag, 2003. Accepted for publication.
46. T. Pawson and P. Nash. Protein-protein interactions define specificity in signal transduction. *Genes and Development*, 14:1027–1047, 2000.
47. C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80:25–31, 2001.
48. T. Quatrani. *Visual Modeling with Rational Rose 2000 and UML*. Addison-Wesley, 2000.
49. A. Regev. The full code for this model and the BioSpi surface ASCII syntax are available in the attached disk.
50. A. Regev, K. Panina, W. Silverman, L. Cardelli, and E. Shapiro. Bioambients - a calculus for biological compartments. manuscript in preparation.
51. A. Regev and E. Shapiro. Cellular abstractions. *Nature*, 419:343, 2002.
52. A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 6, pages 459–470, Singapore, 2001. World Scientific Press.
53. W. Reisig and G. Rozenberg. Informal introduction to petri nets. *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, 1491, 1998.
54. M. G. Samsonova and V. N. Serov. Network: an interactive interface to the tools for analysis of genetic network structure and dynamics. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 4, pages 102–111, Singapore, 1999. World Scientific Press.
55. L. Sanchez, J. van Helden, and D. Thieffry. Establishement of the dorso-ventral pattern during embryonic development of drosophila melanogasater: a logical analysis. *Journal of Theoretical Biology*, 189(4):377–389, 1997.
56. H. M. Sauro. Scamp: a general-purpose simulator and metabolic control analysis program. *Computer Applications in Bioscience*, 9(4):441–450, 1993.
57. S. Schuster, D. A. Fell, and T. Dandekar. A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks. *Nature Biotechnology*, 18(3):326–332, 2000.
58. E. Selkov, Y. Grechkin, N. Mikhailova, and E. Selkov. Mpw: the metabolic pathways database. *Nucleic Acids Research*, 26(1):43–45, 1998.
59. E. Shapiro. Concurrent prolog: a progress report. In E. Shapiro, editor, *Concurrent Prolog (vol. I)*, pages 157–187. MIT Press, Cambridge, Massachusetts, 1987.
60. W. Silverman, M. Hirsch, A. Houri, and E. Shapiro. The Logix system user manual, version 1.21. In E. Shapiro, editor, *Concurrent Prolog (vol. II)*, pages 46–78. MIT Press, Cambridge, Massachusetts, 1987.

61. T. F. Smith. Functional genomics - bioinformatics is ready for the challenge. *Trends in Genetics*, 14(7):291–293, 1998.
62. L. Stryer. *Biochemistry.* W.H. Freeman, 1995.
63. Z. Szallasi. Genetic network analysis in light of massively parallel biological data. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 4, pages 5–16, Singapore, 1999. World Scientific Press.
64. D. Thieffry and D. Romero. The modularity of biological regulatory networks. *Biosystems*, 50(1):49–59, 1999.
65. D. Thieffry and R. Thomas. Qualitative analysis of gene networks. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 3, pages 77–88, Singapore, 1998. World Scientific Press.
66. J. van Helden, A. Naim, R. Mancuso, M. Eldridge, L. Wernisch, D. Gilbert D, and S. J. Wodak. Representing and analysing molecular and cellular function using the computer. *Biological Chemistry*, 381(9–10):921–935, 2000.
67. B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In David Dill, editor, *CAV'94: Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.
68. D. Voet and J. Voet. *Biochemistry. 2nd edition.* John Wiley and sons, 1995.
69. E. Wingender, X. Chen, E. Fricke, R. Geffers, R. Hehl, I. Liebich, M. Krull M, V. Matys, H. Michael, R. Ohnhauser, M. Pruss, F. Schacherer, S. Thiele, and S. Urbach. The transfac system on gene expression regulation. *Nucleic Acids Research*, 29(1):281–283, 2001.
70. I. Xenarios, E. Fernandez E, L. Salwinski, X. J. Duan, M. J. Thompson, E. M. Marcotte, and D. Eisenberg. Dip: the database of interacting proteins: 2001 update. *Nucleic Acids Research*, 29(1):239–241, 2001.
71. T-M. Yi, Y. Huang, M. I. Simon, and J. Doyle. Robust perfect adaptation in bacterial chemotaxis through integral feedback control. *Proceedings of the National Academy of Sciences USA*, 97(9):4649–4653, 2000.